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

System Development User Guide for the AD9371 and AD9375 Integrated Dual-Channel RF Transceivers

INTRODUCTION

This user guide is the main source of information for systems engineers and software developers using the AD9371 family of software defined radio transceivers. This family includes the AD9371 and the AD9375. For this the user guide, these devices are interchangeable unless otherwise stated. The sections in this user guide are organized to simplify navigation for users to find the information pertinent to their area of interest.

PLEASE SEE THE LAST PAGE FOR AN IMPORTANT WARNING AND LEGAL TERMS AND CONDITIONS.

TABLE OF CONTENTS

REVISION HISTORY

5/2017—Rev. A to Rev. B

Added AD9375 ..... Throughout
Updated Layout ..... Universal
Changes to Title and Introduction 1
Added User Guide Section Description Section 7
Changes to Overview Section 8
Added Figure 2; Renumbered Sequentially 10
Changes to Table 5 22
Changes to MYKONOS_setupDeserializers(…) Section 46
Changes to External Tx LO Leakage Initial Calibration Section 90
Changes to System Considerations for Tracking Calibrations Section 91
Changes to External Channel Section 96
Changes to ORx Path Control Section 104
Added Power Amplifier (PA) Protection Section and Table 92; Renumbered Sequentially 109
Add Figure 48 110
Changes to MYKONOS_setupPaProtection(…) Section 111
Changes to MYKONOS_getPaProtectErrorFlagStatus(…) Section 112
Changes to Table 95 Caption 117
Changes to RF PLL Frequency Change Procedure 122
Changes to RF PLL Resolution Limitations Section 126
Changes to Gain Control Section 127
Changed Receiver Datapath—Variable Gain Elements Section to Variable Gain Elements in the Receiver Datapath Section 127
Changes to Figure 59 and Figure 60 132
Changes to PMD Measurement Duration Section and PMD Measurement Configuration Section 153
Changes to Real IF Section 164
Changes to Rx Signal Path Example Section, Figure 77, and Figure 78 165
Changes to THB2 Section and THB1 Section 166
Changes to Figure 85 169
Changes to Figure 87 170
Changed Programming Filter Settings to the AD9371 via API Section to Programming Filter Settings via API Section 171
Changes to Datapath Trigger Modes Section and Table 138 .. 183
Changes to Table 139 184
Changes to Table 155 207
Changes to MYKONOS_getGpioMonitorOut Section 209
Deleted Figure 112; Renumbered Sequentially 224
Changes to MYKONOS_getGpio3v3PinLevel Section 224
Changes to Table 163 226
Added Auxiliary DAC Control Software Control Procedure Section 227
Added Auxiliary (AUX) ADC Readback Software Control Procedure Section 230
Changes to Example 1: SEDZ to PEDZ Conversion Section 236
Changes to Example 2: PEDZ to SEDZ Conversion Section 237
Changes to Impedance Matching Network Section 249
Changes to Figure 151 252
Changes to Table 170 to Table 172 253
Changes to Table 173 to Table 176 254
Added Mykonos Tx1 and Tx2 Port Impedance Section 255
Changes to Figure 159 Caption 262
Changes to Figure 161 Caption 263
Changes to Line Design Examples Section 264
Changed Transmitter Bias and Port Interface Section to Transmitter Bias Design Considerations Section 268
Changes to Isolation Techniques Used on the ADRV9371-N/PCBZ Evaluation Board Section 273
Changed AD9371 Demonstration System Overview Section to Demonstration System Overview Section 281
Changes to Hardware Setup Section 281
Changed AD9371 Transceiver Evaluation Software (TES) Section to Transceiver Evaluation Software (TES) Section and AD9371 Starting the Transceiver Evaluation Software Section to Starting the Transceiver Evaluation Software Section 287
Changes to Installation Section and Figure 183 Caption 287
Changes to Figure 184 288
Changes to GPIO Configuration Tab Section 293
Changes to Figure 207 301
Changes to File Dropdown Menu Section, Figure 209 Caption, Figure 210 Caption, ... 302
Changes to Tools Dropdown Menu Section, Figure 212 Caption, Figure 213 Caption, and Table 188 303

Changes to Help Dropdown Menu, Figure 214 Caption, System, System Status Bar Section, Figure 215 Caption, and Figure 217 Caption 304
Changes to Rx Signal Chain Section..306
Added DPD, CLGC, and VSWR Measurement (AD9375 Only) Section, DPD Overview Section, and Figure 229 319
Added Figure 230 320
Added Figure 231 to Figure 233 321
Added Figure 234 and Equation 57; Renumbered Sequentially...322
Added CLGC Overview Section, Voltage Standing Wave Ratio Measurement Overview Section, Equation 58, and Figure 235 ...323
Added DPD GUI Section amd Figure 236...............................324
Added Figure 237, Waveform Setup Section, and Figure 238.......325
Added DPD Setup Section, Figure 239, and Figure 240 327
Added Figure 241 328
Added PA Calibration Configuration Section and Figure 242.....329
Added Error Messages and Debug Information Section and Figure 243 331
Added DPD API Section, ARM Setup Commands Section, and Table 190 332
Added Table 191 333
Added DPD API Data Structures Section, Figure 244, and Table 192 334
Added Figure 245 335
Added Table 193 336
Added Figure 246 and Table 194 338
Added DPD Functionality API Functions Section..................339
Added CLGC Tracking Calibration Section, CLGC Configurations Section, Figure 247, and Figure 248 341
Added CLGC API Section, CLGC API Data Structures Section, Figure 249, and Table 195...342
Added Figure 250 and Table 196 344
CLGC Functionality API Functions 345
Added VSWR Tracking Calibration Section, VSWR Monitoring Section, and Figure 251 346
Added VSWR API Section, VSWR API Data Structures Section, Figure 252, and Table 197...347
Added Figure 243 and Table 198 349
Added VSWR Functionality API Functions Section................350
Added Systems Design Considerations Section, DPD Tuning Procedure Section, and DPD Model Save/Restore Functionality Section 351
Added Guidelines for Developing and Troubleshooting a DPD System Section 352
Added Designing a System for a TDD Application Section and Table 199 353
Added External Loopback Flatness Requirements and Figure 254 354
Added Figure 255 355
Added CLGC Convergence Time Section, Figure 256, DPD Limitations Section, and Other Considerations Section 356
Added Typical Test Setup and DPD Performance Section and Figure 257 357
Added Figure 258 358
Added TDD Setup Instructions Section and Figure 259 359
Added Figure 260 260

1/2017—Rev. 0 to Rev. A

Changes to Introduction Section 1
Deleted /src/example Section 7
Changes to Figure 3 8
Changes to Developing the Application Section 10
Changes to API Sequence Section 11
Changes to Figure 4 and Figure 5 15
Change to t_{CO} Parameter, Table 4 and Figure 6 16
Changes to Table 38 38
Added Hardware Considerations for Sync Signals Section.......53
Changes to Data Structure Member Initialization Section 56
Deleted Folder Location Section and Figure 28; Renumbered Sequentially 58
Changes to headless.c Section 58
Added Reference Clock and SYSREF Connections Section, Connections for External Clock (DEV_CLK_IN) Section, Figure 47, DEV_CLK Phase Noise Requirements Section, Table 92, and Table 93; Renumbered Sequentially 104
Added SYSREF Requirements Section and Figure 48.............105
Added Minimum Delay Requirements Between SYSREF Pulses Section, Timing of SYSREF Compared to DEV_CLK Section, Figure 50, and Figure 51 106
Added Connections for External LO Section, Figure 52, and Table 94...107
Added Performance Limitations Section and Figure 53..........109
Changes to Example 1 Section, Example 2 Section, and Example 3 Section 118
Deleted DEV_CLK, SYSREF, and External LO Section, Overview Section, Connections for the External Clock (DEV_CLK_IN) Section, Figure 51, and Figure 52;................120
Deleted DEV_CLK Phase Noise Requirements Section, Figure 53, Table 94, and Table 95; Renumbered Sequentially.......121
Deleted Figure 54 and Figure 55 122
Deleted Multichip Synchronization (JESD204B Mode) Section and Figure 56 123
Deleted Connections for External LO Section, Figure 57, and Table 96...124
Changes to Analog Peak Detector (APD) Basics Section........127
Changes to Half-Band 2 (HB2) Overload Detector Basics Section 129
Added TDD Configuration and Setup Section, TDD in the Mykonos Evaluation System Section, ARM Input and Output Signals Section, and ARM Pin Mode Configurations Section......168
Added Figure 88 169
Added ARM Acknowledge Signals Section, Table 133, and Figure 89 170
Added Table 134 and Table 135...171
Added Figure 90 and Table 136..172
Added FPGA Output Signals Section, Data Path Trigger Modes Section, and Table 137 173
Added Table 138..174
Added TDD Finite State Machine Class Section, Quick Help for Programming FPGA/Mykonos Section, and Table 139...........175

Added ORX_MODE[2:0] and ORX_TRIGGER, as Seen by the FPGA Section and Example TDD Script in IronPython Section 176
Added API/DLL Commands for TDD Configuration Section 180
Added Secondary Serial Peripheral Interface (SPI2) Section, Figure 97, and Figure 98 204
Added Table 160, SPI2 Register Map Section, API Description Section, and Table 161 205
Moved Impedance Matching Network Section 237
Moved Figure 144 to Figure 146 238
Moved Figure 147 to Figure 149 239
Moved Figure 150 240
Moved Selected Balun and Component Values Section, and Table 169 to Table 172 241
Moved Table 173 to Table 176 and Mykonos Tx1 and TX2 Port Impedance Section 242
Moved Figure 151 and Figure 152 243
Added Board Layout Design Recommendations Section 243
Added Printed Circuit Board Layout Guidelines Section and PCB Material and Stack Up Selection Section 244
Added Figure 153 and Table 177 245
Added Fanout and Trace Space Guidelines Section and Figure 154 246
Added Component Placement and Routing Priorities Section, Signals with Highest Routing Priority Section, and Figure 155 .. 247
Added Figure 156 248
Added Signals with Second Routing Priority Section and Figure 157 249
Added Figure 158 and Figure 159 250
Added Signals with Lowest Routing Priority and Figure 160 251
Added RF and JESD204B Transmission Line Layout Section 252
Changes to Table 178 to Table 181 253
Changes to Example 5 Section, Table 182, and RF Line Design Summary Section 254
Added Figure 161 and Figure 162 254
Added Transmitter Bias and Port Interface Section, Figure 163, Figure 164, and Tx Balun DC Supply Options Section 255
Added Figure 165, DC Balun Section, and Chokes Section ... 256
Added Figure 166 257
Added Figure 167, Figure 168, JESD204B Trace Routing Recommendations Section, and Routing Recommendations Section 258
Added Stripline vs. Microstrip Section, Isolation Techniques Used on the ADRV9371-N/PCBZ Evaluation Card Section, and Figure 169 259
Added Figure 170 260
Added Isolation Between JESD204B Lines Section and Figure 171 261
Added Unused Pins Section and Table 183 262
Added Power Management Considerations Section and Table 184 263
Added ADRV9371-N/PCBZ Evaluation Card Power Supply Block Diagram Section and Figure 173 265
Added RF and Clock Synthesizer Supplies Section 266
Changes to Figure 202 and AD9528 Description Section 286
Changes to Figure 213 291

7/2016—Revision 0: Initial Version

USER GUIDE SECTION DESCRIPTION

For simplified navigation of this user guide, an overview of each section follows:

- System Overview. This section explains the capability of the device and serves as an introduction to all the subsystems and functions, including the block diagrams and interfaces.
- System Architecture Description. This section explains the software design approach using the application programming interface (API) and all details required to develop code on the device.
- Software Integration. This section describes the process for developing code using the APIs. This section lists common API functions for user integration into the code base.
- Serial Peripheral Interface (SPI). The SPI is the main control interface between the baseband processor (BBP) and the integrated transceiver.
- JESD204B Interface. This section provides a description of the JESD204B digital interface, setup, and configuration options.
- System Initialization. This section provides the sequence of steps required at startup.
- Quadrature Error Correction, Calibration, and ARM Configuration. This section describes the calibration and error correction functions and setup guidelines for configuring the ARM processor to perform scheduled adjustments.
- System Control. This section describes the commands and sequences for setting up the different radio channels.
- Tx Power Control. This section explains the commands and procedures for adjusting Tx power control during normal operation.
- Power Amplifier (PA) Protection. This section describes how to setup the PA protection features to help prevent the power amplifier from being overdriven. The transmitter channels feature~~s~~ a protection mechanism that can help prevent damage to the PA connected to either of the transmitter outputs. When the full-scale output power of the device exceeds the maximum input to the PA, it can damage the PA. The PA protection feature implements feedback in the system to prevent an overload by measuring the signal level and by comparing it to the user-programmable threshold. This information can reduce the transmit output level on the flagged channel and eliminate the damage threat. This section describes how to set up the reference clocks needed for the internal clock and for signal generation as well as data synchronization.
- Synthesizer Configuration. This section describes how to configure the synthesizers for different modes of operation. This section includes details for receiver (Rx), transmitter (Tx), observation receiver (ORx), and clock phase-locked loop (PLL) setup, as well as the calibration PLL setup.
- Gain Control. This section describes the options for controlling receiver gain settings, including manual gain control (MGC) provided by the BBP and automatic gain control (AGC) provided by the integrated transceiver.
- Filter Configuration. This section describes all the digital filter blocks in the receivers and transmitters, and explains the programmable FIR filters and how to set their coefficients.
- Observation Receiver. This section describes the ORx inputs and the sniffer receiver (SnRx) inputs, including system implementation and setup APIs.
- TDD Configuration and Setup. This section describes software configuration for operating in a time division duplexed (TDD) system.
- General-Purpose Input/Output (GPIO) Configuration. This section describes the options for configuring standard digital GPIO pins.
- 3.3 V General-Purpose Input/Output Overview. This section describes the options for configuring the 3.3 V supplied GPIO pins.
- General-Purpose Interrupt Overview. This section describes setup and operation of the general-purpose interrupt pin.
- Auxiliary Converters—AUXDAC_x, AUXADC. This section describes the capability of the AUXADC_x inputs, the AUXDAC_x outputs, and how to properly configure the inputs and outputs for various applications.
- RF Port Interface. This section explains all the details necessary to properly match the RF impedances of each differential input and output port.
- Printed Circuit Board Layout Guidelines. This section describes the printed circuit board (PCB) construction, layout, routing, and isolation techniques necessary to optimize device performance.
- Power Management Considerations. This section describes the power supply design and all the considerations needed to optimize device performance.
- Demonstration System Overview. This section describes the demonstration system, including the evaluation board, motherboard, and hardware integration setup needed to properly evaluate device performance.
- Transceiver Evaluation Software. This section describes the transceiver evaluation software (TES) that provides a graphical user interface that controls the evaluation system.
- DPD, CLGC, and VSWR Measurement (AD9375 Only). This section describes operation of the closed-loop transmitter control functions available only in the AD9375 device.

SYSTEM OVERVIEW

Analog Devices, Inc., provides a variety of highly integrated RF agile transceivers, including the AD9371 and AD9375. This transceiver family provides dual-channel receivers, dual-channel transmitters, integrated synthesizers, digital signal processing functions, and a high speed serial interface. The AD9375 provides the added capability or integrated digital predistortion (DPD) for the transmitter channels to improve linearity and decrease power consumption. The devices operate over the wide frequency range of 300 MHz to 6 GHz and can support a transmit synthesis bandwidth up to 250 MHz, as well as a receiver bandwidth up to 100 MHz. The information in this document applies equally to the AD9371 and the AD9375, except for the section that describes DPD operation for the AD9375. To avoid confusion, the term device is used throughout the user guide to refer to both devices interchangeably. In sections that refer to only one device, the part number referenced to clearly delineate which device is being described. Note that references in the diagrams and the application programming interface (API) code examples keep the text that appears in the code, even when it applies to both devices.

The device provides three receiver inputs with limited bandwidth (20 MHz) to monitor signals on other channels of interest. These receivers, commonly referred to as sniffer receivers (SnRxs), can be matched to different frequency range antennae to monitor a wider spectrum during normal operation in a more narrow band. The independent synthesizer associated with these receivers allows the receivers to operate on different frequency channels during normal transmit/receive operation.

An additional pair of receiver channels can be used as dedicated observation receivers used to monitor the transmitter channels. These receivers provide the same bandwidth and gain capability as the main signal channel receivers, but are dedicated for use as monitors for transmitter performance. These receivers provide a feedback path to implement calibration and error correction algorithms on the transmit data.

All signal data transfers are accomplished using a JESD204B high speed serial interface with eight separate lanes. Four lanes are dedicated as inputs to the transmitter system and four lanes are configurable to serve as outputs for the receiver system. When one or two main signal chain receivers are active and an observation/sniffer receiver is active, the main signal chain receivers can be assigned one or two receiver lanes, and the observation/sniffer receiver can also be assigned only one or two lanes. Note that only one observation receiver or sniffer receiver can be operational at any given time, but all four lanes can be assigned to that channel or shared between this channel and the active signal chain receivers.

A serial peripheral interface (SPI) transmits and receives control information between the device and a baseband processor. All software control is communicated via this interface. There is also a control interface that utilizes GPIO lines to provide hardware control to and from the device. These pins can be configured to provide dedicated sets of functions for different application scenarios. Some GPIOs are intended for digital control, while others are supplied by a 3.3 V analog supply for use in controlling external analog components. There are also ten auxiliary digital to analog converters (DACs), known as auxiliary DACs, that can be muxed with 3.3 V GPIO pins to be used as control voltage sources for other devices requiring variable control voltages. Included in this block is a set of three low speed auxiliary analog to digital converters (ADCs) that monitor external voltages of interest to system operation.

Figure 1 and Figure 2 show block diagrams for the AD9371 and the AD9375, respectively. Software control of each block is described in the following sections of this user guide. Note that all software code is taken from the API that is supplied with the device. References to Mykonos in the API refer to the Analog Devices development name for the device family.

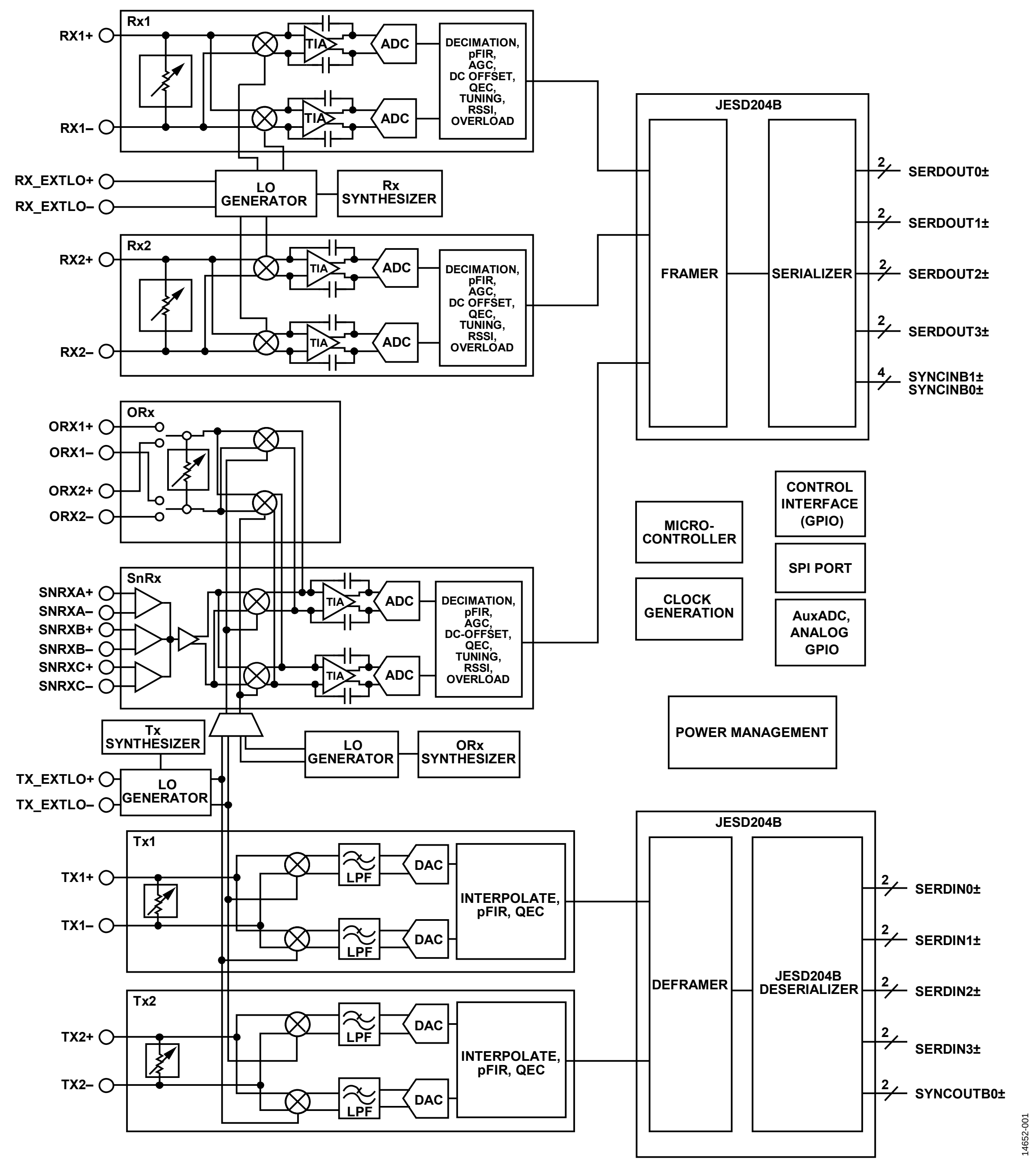


Figure 1. AD9371 Functional Block Diagram

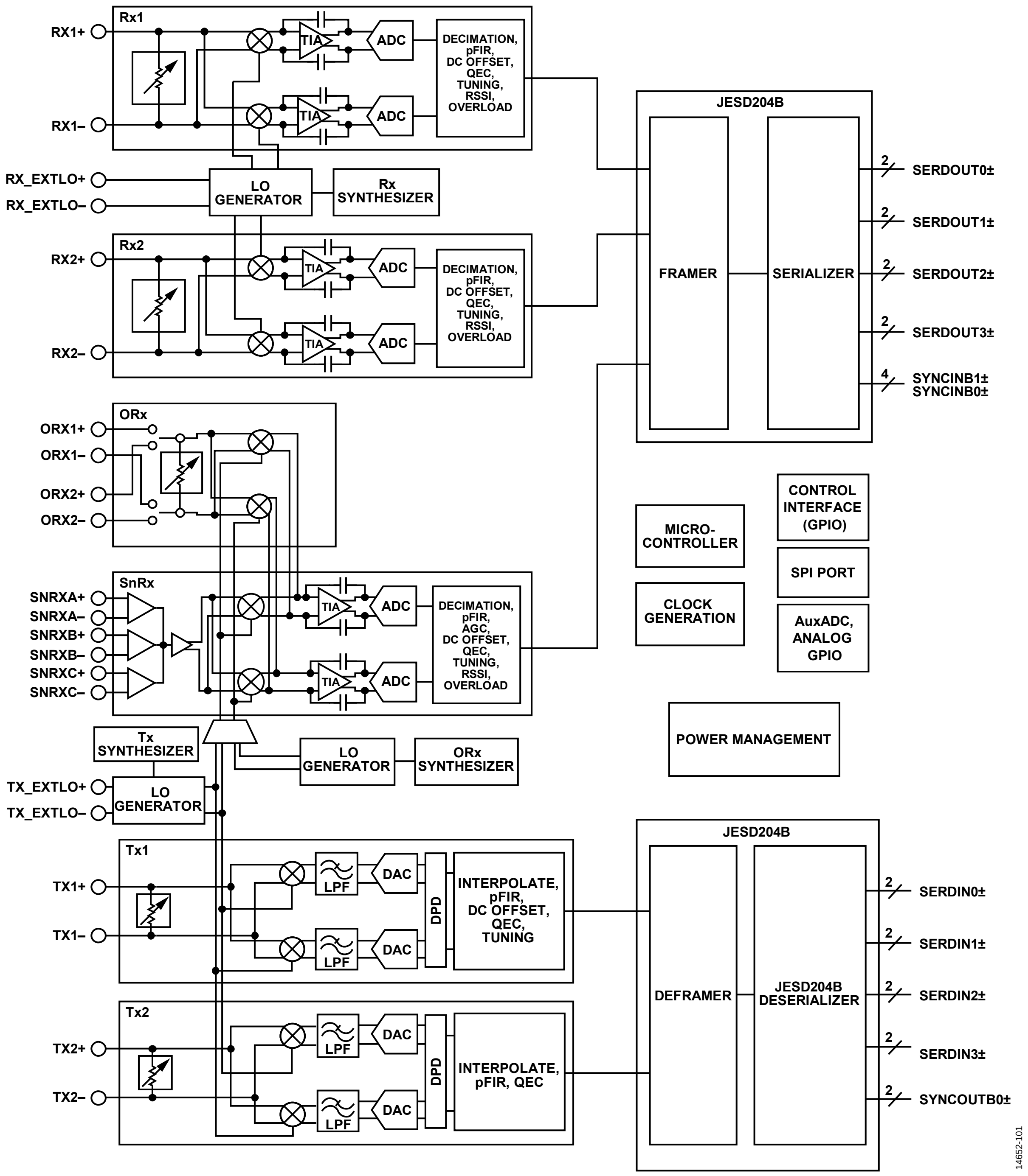


Figure 2. AD9375 Functional Block Diagram

SYSTEM ARCHITECTURE DESCRIPTION

This user guide provides information about the API software developed by Analog Devices for the AD9371 transceiver family. This user guide outlines the overall architecture, folder structure, and methods for using the application programming interface software on any platform. This user guide does not describe the API library functions. Detailed information regarding the application programming interface functions is located in the **/src/doc** file in the software package directory structure. This file can also be viewed in the **Help** tab on the transceiver evaluation software (TES) used for controlling the evaluation platform.

SOFTWARE ARCHITECTURE

Figure 3 shows the software architecture.

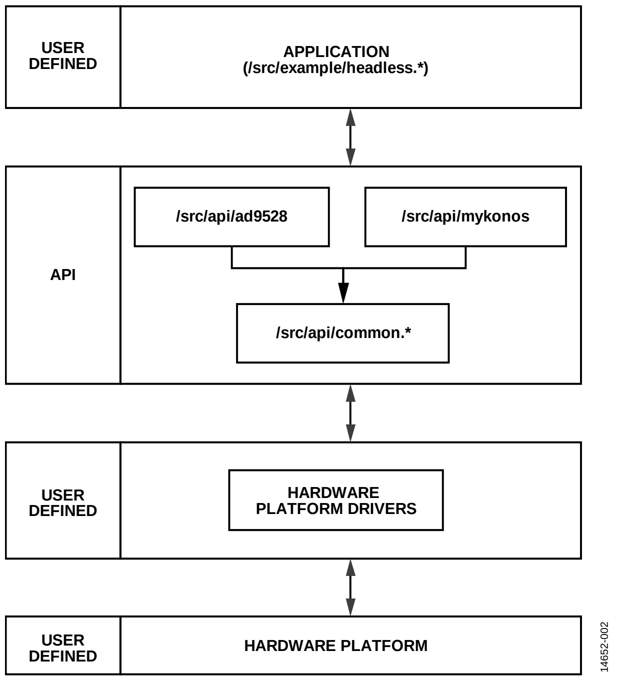


Figure 3. API Software Architecture

FOLDER STRUCTURE

The source files are located in the folder structure shown in Figure 4. Each branch is explained in the following sections.

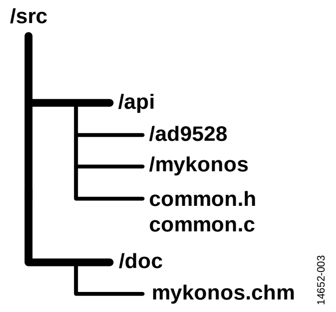


Figure 4. API Folder Structure

/src/api

This folder includes the main application programming interface code for the AD9371 transceiver family, as well as the Analog Devices AD9528 clock chip. The application programming interface (API) is composed of all files located in the **/mykonos** folder, as well as the **common.h** file and **common.c** file. The **/mykonos** folder contains the high level function prototypes, data types, macros, and source code used to build the final user software system. The user is strictly forbidden to modify the files contained in the **/mykonos** folder and the **/ad9528** folder. Analog Devices maintains this code as intellectual property and all changes are at their sole discretion. The **common.h** and **common.c** files provide the means for a user to insert their hardware driver code for system integration with the AD9371 API. A description regarding the use of **common.c** is contained in the Software Integration section.

/src/doc

This folder contains the API doxygen (**mykonos.chm**) file for user reference. It is in compressed HTML format.

SOFTWARE INTEGRATION

The current application programming interface (API) package was developed on a Xilinx® ZC706 reference platform using a dual-core ARM A9 processor running a Linux® variant. Users are required to integrate the API with their platform specific code base. This is readily accomplished because the API abides 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.

MODIFYING COMMON.C

Users develop code on their own hardware specific platforms. Therefore, users maintain different drivers for the peripherals, such as the SPI and GPIO, than what is included in the API. Users can use their own drivers for these peripherals, or they may use standard drivers if they use an operating system, such as Linux.

The API is designed with the intent that developers may use any driver of their choice for their platform requirements. Users are permitted to substitute their driver code within the function bodies located in **common.c file** in the **/mykonos_api** directory for their platform requirements. However, users may not modify the parameter declarations for these functions or any other code because doing so breaks the API. Analog Devices does not support any user application containing unauthorized API code. The functions in the **common.c** file for which a developer can substitute their own hardware specific information are described in Table 1.

Table 1. Common API Functions for User Integration

Function	Description
void CMB_hardReset(uint8_t spiChipSelectIndex)	This function performs the required platform dependent resets.
void CMB_setGPIO(uint32_t GPIO)	This function sets the GPIOs based on the platform requirement.
void CMB_setSPIOptions(spiSettings_t *spiSettings)	The device and the clock chip use the same SPI settings, but different chip selects. This function assigns SPI settings to each SPI enabled device on the board.
void CMB_setSPIChannel(uint16_t chipSelectIndex)	This function assigns the chip select to the device on the board. Users know the chip select assigned to each device on their platform.
CMB_SPIWriteByte(spiSettings_t *spiSettings, uint16_t addr, uint8_t data)	Use this function to write a byte to an SPI register.
CMB_SPIReadByte (spiSettings_t *spiSettings, uint16_t addr, uint8_t *readdata)	Use this function to read a 1-byte SPI register.
CMB_SPIWriteField(spiSettings_t *spiSettings, uint16_t addr, uint8_t field_val, uint8_t mask, uint8_t start_bit)	This function writes a bit field to an SPI register.
CMB_SPIReadField (spiSettings_t *spiSettings, uint16_t addr, uint8_t *field_val, uint8_t mask, uint8_t start_bit)	This function reads a bit field from an SPI register.
CMB_wait_ms(uint32_t time_ms)	This function instructs the API to wait for units of ms.
CMB_wait_us(uint32_t time_us)	This function instructs the API to wait for units of µs.
CMB_setTimeout_ms(uint32_t timeOut_ms)	This function sets the timeout in ms.
CMB_setTimeout_us(uint32_t timeOut_us)	This function sets the timeout in µs.
BOOL CMB_hasTimeoutExpired()	This function shows if timeout happened based on the timeout already set in the system.

DEVELOPING THE APPLICATION

The **headless.c** file provides a user example for top level configuration and control. Users can reference this to develop their application. However, the initialization sequence is unique and includes the assignment of data structure values. Data structures containing device configuration and operation variables are throughout the application programming interface. A **headless.c** file incorporating user selected settings can be obtained by creating a C script using transceiver evaluation software (TES) as described in the Other TES Features section.

Data Structures

The application programming interface functions use a specific set of data structures. The application code initializes these data structures. All application programming interface functions use a pointer to a device data structure to convey configuration and control settings. It is imperative that structure initialization is complete before attempting system operation. Table 2 contains a list of these structures.

The **headless.c** file illustrates the structure initialization sequence at the beginning of the file. Explanations for each data structure are contained in the mykonos.chm document.

Note that all unchanged JESD204B parameters must use the default JESD204B parameters in the application programming interface. Failure to do so results in erroneous link operation.

Table 2. List of Data Structures Used in Application Programming Interface

Data Structure	Location	Description
spiSettings_t	/src/api/common.h	This data structure contains the SPI settings for all system device types.
mykonosFir_t	/src/api/mykonos/t_mykonos.h	This data structure contains the FIR filter gain, the number of coefficients, and a pointer to a filter coefficient array.
mykonosJesd204bFramerConfig_t	/src/api/mykonos/t_mykonos.h	This data structure contains the JESD204B framer configuration parameters.
mykonosJesd204bDeframerConfig_t	/src/api/mykonos/t_mykonos.h	This data structure contains the JESD204B deframer configuration parameters.
mykonosRxProfile_t	/src/api/mykonos/t_mykonos.h	This data structure contains the Rx profile information.
mykonosTxProfile_t	/src/api/mykonos/t_mykonos.h	This data structure contains the Tx profile information.
mykonosSnifferGainControl_t	/src/api/mykonos/t_mykonos.h	This data structure contains the sniffer Rx manual gain control information.
mykonosORxGainControl_t	/src/api/mykonos/t_mykonos.h	This data structure contains the observation Rx manual gain control information.
mykonosRxGainControl_t	/src/api/mykonos/t_mykonos.h	This data structure contains the Rx manual gain control information.
mykonosAgcCfg_t	/src/api/mykonos/t_mykonos.h	This data structure contains the automatic gain control (AGC) information.
mykonosTxSettings_t	/src/api/mykonos/t_mykonos.h	This data structure contains the Tx setting information.
mykonosRxSettings_t	/src/api/mykonos/t_mykonos.h	This data structure contains the Rx setting information.
mykonosObsRxSettings_t	/src/api/mykonos/t_mykonos.h	This data structure contains the observation Rx setting information.
mykonosGpio3v3_t	/src/api/mykonos/t_mykonos.h	This data structure contains the 3.3 V dc GPIO setting information.
mykonosGpio1v8_t	/src/api/mykonos/t_mykonos.h	This data structure contains the 1.8 V dc GPIO setting information.
mykonosAuxIo_t	/src/api/mykonos/t_mykonos.h	This data structure contains the auxiliary ADC, DAC, and pointers to the GPIO setting information.
mykonosDigClocks_t	/src/api/mykonos/t_mykonos.h	This data structure contains the digital clock parameters.
mykonosDevice_t	/src/api/mykonos/t_mykonos.h	This data structure is inclusive of all previous data types, which are instantiated as pointers. The PROFILESVALID bit field identifies which profile is valid. This data type is used to instantiate one device for configuration and control after member structure initialization.
mykonosArmGpioConfig_t	/src/api/mykonos/t_mykonos.h	This data structure holds the ARM GPIO pin assignments for each ARM input/output pin.
mykonosPeakDetAgcCfg_t	/src/api/mykonos/t_mykonos.h	This data structure holds the peak detector settings for the AGC.
mykonosPowerMeasAgcCfg_t	/src/api/mykonos/t_mykonos.h	This data structure holds the power measurement settings for the AGC.

Data Structure	Location	Description
mykonosInitCalStatus_t	/src/api/mykonos/t_mykonos.h	This data structure reads back the initialization calibration status.
mykonosTxLolStatus_t	/src/api/mykonos/t_mykonos.h	This data structure holds the Tx local oscillator leakage (LOL) status.
mykonosTxQecStatus_t	/src/api/mykonos/t_mykonos.h	This data structure holds the Tx quadrature error correction (QEC) status.
mykonosRxQecStatus_t	/src/api/mykonos/t_mykonos.h	This data structure holds the Rx QEC status.
mykonosOrxQecStatus_t	/src/api/mykonos/t_mykonos.h	This data structure holds the Orx QEC status.
mykonosGainComp_t	/src/api/mykonos/t_mykonos_gpio.h	This data structure holds the gain compensation settings for the main receive channels.
mykonosObsRxGainComp_t	/src/api/mykonos/t_mykonos_gpio.h	This data structure holds the gain compensation settings for the observation channel.
mykonosFloatPntFrmt_t	/src/api/mykonos/t_mykonos_gpio.h	This data structure holds the floating point formatter settings for the floating point number generation
mykonosTempSensorConfig_t	/src/api/mykonos/t_mykonos_gpio.h	This data structure configures the on die temperature sensor.
mykonosTempSensorStatus_t	/src/api/mykonos/t_mykonos_gpio.h	This data structure stores the temperature sensor related values.
mykonosLaneErr_t	/src/api/mykonos/mykonos_debug/t_mykonos_dbgjesd.h	This data structure holds the error counters per a given lane.
mykonosDeframerStatus_t	/src/api/mykonos/mykonos_debug/t_mykonos_dbgjesd.h	This data structure holds the deframer status.

Using API Functions

Direct SPI read/write operation is not permitted when configuring the device or the Analog Devices clock chip device. Only use the high level API functions defined in the **/src/api/mykonos/mykonos.h**, **/src/api/mykonos/mykonos_gpio.h**, **/src/api/mykonos/mykonos_debug/mykonos_dbgjesd.h**, or **/src/api/ad9528/ad9528.h** files. Users must not directly use any SPI read/write function located in **common.c** in their application code for device configuration or control. Analog Devices does not support any user code containing SPI writes reverse engineered from the original API.

Adding Gain Tables and Device Profiles

The **/src/api/mykonos/mykonos_user.h** and **/src/api/mykonos/mykonos_user.c** files provide setup information for the gain tables. The default gain table settings for the Rx, the ORx, and the sniffer Rx are located in **mykonos_profiles.c**. Each gain table is organized according to a descending gain index normalized to a maximum gain for each respective API calling function. The tables consist of a two-dimensional array construct where the subarray order for each gain table type is a code comment at the beginning of the declaration. Users can modify these gain tables to include their own custom configurations. Users must submit their custom gain settings to Analog Devices for approval. Analog Devices does not support any custom gain tables not submitted for approval prior to user use. Details of these files are available in the device **/src/doc** file in the software package directory structure

API Sequence

The outline of the correct API initialization sequence illustrated in **headless.c** is as follows:

1. Instantiate all data structures and load their members required by the user application (myk_init.c contents).
2. Initialize and set up all clocks (the platform clock source and the JESD204B SYSREF signals are set up).
3. Initialize the hardware platform (hardware dependent devices such as FPGA/ASIC/BBP interfaces are initialized).
4. Reset the device (call MYKONOS_resetDevice for the reset of the transceiver device in preparation for initialization).
5. Initialize the device (call MYKONOS_initialize function for configuration of the device).
6. Check CLKPLL status for lock (call MYKONOS_checkPllLockStatus and perform check with user defined code).
7. Perform multichip synchronization (all JESD204B lanes are synchronized together for deterministic latency requirements).
8. Initialize the ARM processor (call MYKONOS_initArm).
9. Load the ARM binary file (call MYKONOS_loadArmFromBinary with user defined binary array pointer).
10. Set the RF PLL frequencies (call MYKONOS_setRfPllFrequency for each channel used by the application).
11. Perform the RF PLL lock check (call MYKONOS_checkPllLockStatus and perform the check with user-defined code).
12. Set the GPIO functions with the desired configuration (check **headless.c** for API calls to be made).

13. Run the initialization calibrations (call MYKONOS_runInitCals and MYKONOS_waitInitCals with user defined code).
14. Enable the SYSREF for the Rx and ORx deframer (call MYKONOS_enableSysrefTo … functions).
15. Send the SYSREF signal to bring up the JESD204B interface.
16. Check deframer and framer status (call MYKONOS_readDeframerStatus and MYKONOS_readRxFramerStatus).
17. Verify the sync and link status for the hardware platform.
18. Enable tracking calibrations (call MYKONOS_enableTrackingCals).
19. Turn the radio on for all transmitters and receivers that were previously set up (call MYKONOS_radioOn).

Note that hardware designs with multiple AD9371 or AD9375 devices require each device to have its own unique configuration data initialized for all data structures.

Restrictions

Do not modify any code located in the **/src/api/*** folder, other than changing the common.c code bodies for hardware driver insertion and gain table and profile changes in mykonos_profiles.c, as previously discussed. Analog Devices maintains the code in the **/src/api/mykonos** and **/src/api/ad9528** folders as intellectual property and all changes are at their sole discretion. Analog Devices provides new releases to fix any code bugs in these folders. Verification of all code bugs is independent of any user code.

SERIAL PERIPHERAL INTERFACE (SPI)

The SPI bus provides the mechanism for all digital control of the device by a baseband processor (BBP). Each SPI register is 8 bits wide, and each register contains control bits, status monitors, or other settings that control all functions of the device. This section is mainly an informational section meant to give the user an understanding of the interface used by the BBP for digital control. All control functions are implemented using the API detailed within this user guide. The following sections explain the specifics of this interface.

SPI CONFIGURATION USING API FUNCTION

The SPI bus is configured by the MYKONOS_setSpiSettings (mykonosDevice_t *device) function. Users can configure SPI settings for the device for use in different configurations by calling the MYKONOS_setSpiSettings function with the following parameters.

- To use the SPI as a 4-wire interface,

  ```
  MYKONOS_setSpiSettings  (device->
  spiSettings->fourWireMode = 1);
  ```

- To use the SPI as a 3-wire interface (default configuration),

  ```
  MYKONOS_setSpiSettings  (device->
  spiSettings->fourWireMode = 0);
  ```

- To use the SPI in LSB first mode,

  ```
  MYKONOS_setSpiSettings  (device->
  spiSettings->MSBFirst = 0);
  ```

- To use the SPI in MSB first mode (default configuration),

  ```
  MYKONOS_setSpiSettings  (device->
  spiSettings->MSBFirst = 1);
  ```

- To enable single instruction SPI data transfer mode,

  ```
  MYKONOS_setSpiSettings  (device->
  spiSettings->enSpiStreaming = 0);
  ```

- To enable SPI streaming mode,

  ```
  MYKONOS_setSpiSettings  (device->
  spiSettings->enSpiStreaming = 1);
  ```

SPI streaming allows the device to automatically change the register address after each operation. Users can select an autoincrement or autodecrement SPI address with the following parameters:

- For an autoincrement SPI address, where next addr = addr + 1, program the following:

  ```
  MYKONOS_setSpiSettings  (device->
  spiSettings->autoIncAddrUp = 1);
  ```

- For an utodecrement SPI address, where next addr = addr – 1, program the following:

  ```
  MYKONOS_setSpiSettings  (device->
  spiSettings->autoIncAddrUp = 0);
  ```

SPI BUS SIGNALS

The SPI bus consists of the signals described in the following sections.

CSB

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

Forcing CSB 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 transaction is aborted and the state machine returns to the ready state. Any complete data byte transfers prior to CSB 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 BBP to the device. It is only active while CSB is low. The maximum SCLK frequency is 50 MHz.

SDIO and SDO

When configured as a 4-wire bus, the SPI uses two data signals: SDIO and SDO. SDIO is the data input line driven from the baseband processor to the device, and SDO is the data output from the AD9371 to the baseband processor in this configuration. When configured as a 3-wire bus, SDIO is used as a bidirectional data signal that both receives and transmits serial data. In this mode, the SDO port is disabled.

The data signals are launched on the falling edge of SCLK and sampled on the rising edge of SCLK by both the baseband processor and the device. SDIO carries the control field from the baseband processor to the AD9371 during all transactions, and it carries the write data fields during a write transaction. In a 3-wire SPI configuration, SDIO carries the returning read data fields from the device to the BBP during a read transaction. In a 4-wire SPI configuration, SDO carries the returning data fields to the baseband processor.

The SDO and SDIO pins transition to a high impedance state when the CSB input is high. The device does not provide any weak pull-up or pull-down on these pins. When SDO is inactive, it is floated in a high impedance state. If a valid logic state on SDO is required at all times, add an external, weak pull-up/pull-down on the printed circuit board (PCB).

SPI DATA TRANSFER PROTOCOL

The device SPI is a flexible, synchronous serial communications bus allowing seamless interfacing to many industry-standard microcontrollers and microprocessors. The serial input/output (I/O) is compatible with most synchronous transfer formats, including both the Motorola, Inc., SPI and Intel® SSR protocols. The control field width is limited to 16 bits, and multibyte I/O operation is allowed. The device cannot be used to control other devices on the bus; it only operates as a slave.

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

Phase 1 Instruction Format

The 16-bit control field contains information shown in Table 3.

Table 3. Phase 1 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 0 indicates a write operation.

[D14:D0]

Bits A[14:0] specify the starting byte address for the data transfer during Phase 2 of the input/output 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 input/output operation continues as if the address space is valid. For write operations, the written bits are discarded, and the read operations result in Logic 0s at the output

Single-Byte Data Transfer

When enSpiStreaming = 0, a single-byte data transfer is chosen. In this mode, CSB goes active low, the SCLK signal activates, and the address is transferred from the baseband processor to the device.

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

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

Multibyte Data Transfer

When enSpiStreaming = 1, a multibyte data transfer is allowed. In this mode, data transfers across the bus as long as the CSB pin is low. The autoIncAddrUp controls how the address changes for subsequent writes or reads. When autoIncAddrUp = 1, the address increments from the starting address for each subsequent data transfer until CSB is driven high. If the last register address is reached, the next address accessed is 0x000. When this bit is clear, the address decrements from the starting address for each subsequent data transfer. If this bit is clear and Address 0x000 is reached, the next address to be accessed is the last register location defined in the register map. It is strongly recommended that any data transfer be controlled so that Address 0x000 is only written once at startup.

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

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

Example: LSB First Multibyte Transfer, Autoincrementing Address

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

- Ensure that fourWireMode = 1 (the device is configured to work with the 4-wire interface).
- Ensure that MSBFirst = 0 (the SPI works in LSB first mode).
- Ensure that autoIncAddrUp = 1 (the address pointer automatically increments).
- Ensure that enSpiStreaming = 1 (a multibyte data transfer is allowed).
- Force the CSB line low and keep it low until the last byte is transferred.
- Send the instruction word 0101 0100 0000 000_0 (the last 0 indicates a write operation) to select Register 0x02A as the starting address.
- Use the next 32 clock cycles to send the data to be written to the registers, LSB to MSB for each 8-bit word.
- Ensure the CSB line is driven high after the last bit is sent to Register 0x02D to end the data transfer

Example: MSB First Multibyte Transfer, Autodecrementing Address

To complete a 4-byte write starting at Register 0x02A and ending with Register 0x027 in LSB first format, follow these instructions when programming the master:

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

TIMING DIAGRAMS

The diagrams in Figure 5 and Figure 6 show the SPI bus waveforms for a single register write operation and a single register read operation, respectively. In Figure 5, the value 0x55 is written to Register 0x00A. In Figure 6, Register 0x00A is read and the value returned by the device is 0x55. If the same operations are performed with a 3-wire bus, the SDO line in Figure 5 is eliminated, and the SDIO and SDO lines in Figure 6 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.

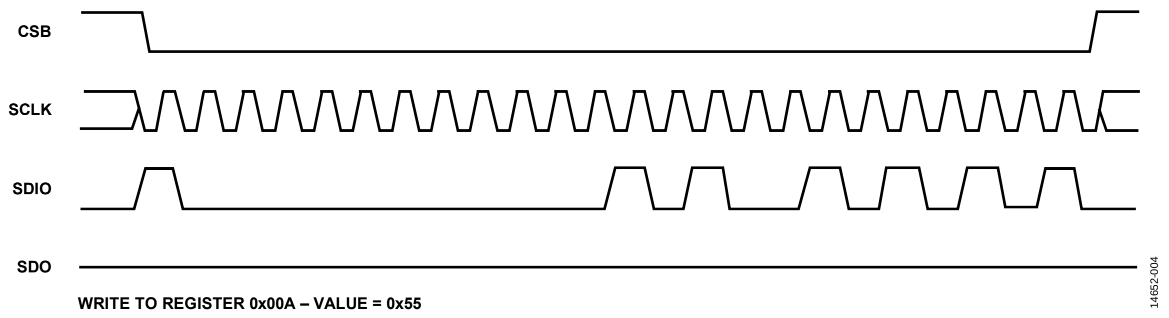


Figure 5. Nominal Timing Diagram, SPI Write Operation

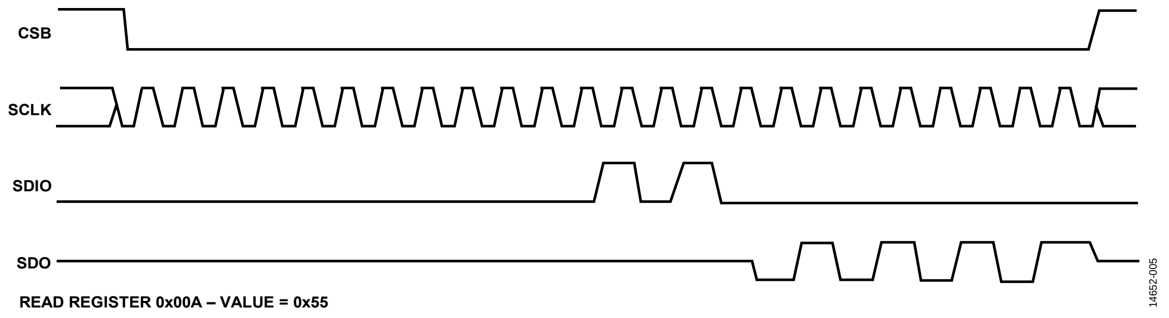


Figure 6. Nominal Timing Diagram, SPI Read Operation

Table 4 lists the timing specifications for the SPI bus. The relationship between these parameters is shown in Figure 7. This diagram shows a 3-wire SPI bus timing diagram with the device returning a value of 0xD4 from Register 0x00A; the timing parameters are marked. Note that this is a single-read operation; therefore, the bus ready parameter after the data is driven from the device (t_{HZS}) is not shown in the diagram.

Table 4. SPI Bus Timing Constraint Values

Parameter	Min	Typ	Max	Description
t_{CP}	20 ns			SCLK cycle time (clock period)
t_{MP}	10 ns			SCLK pulse width
t_{SC}	3 ns			CSB setup time to first SCLK rising edge
t_{HC}	0 ns			Last SCLK falling edge to CSB hold
t_S	3 ns			SDIO data input setup time to SCLK
t_H	0 ns			SDIO data input hold time to SCLK
t_{CO}	3 ns		8 ns	SCLK falling edge to output data delay (3-wire or 4-wire mode)
t_{HZM}	t_H		t_{CO}	Bus turnaround time after baseband processor drives the last address bit
t_{HZS}	3 ns		t_{CO}	Bus turnaround time after the device drives the last data bit (not shown in Figure 7)

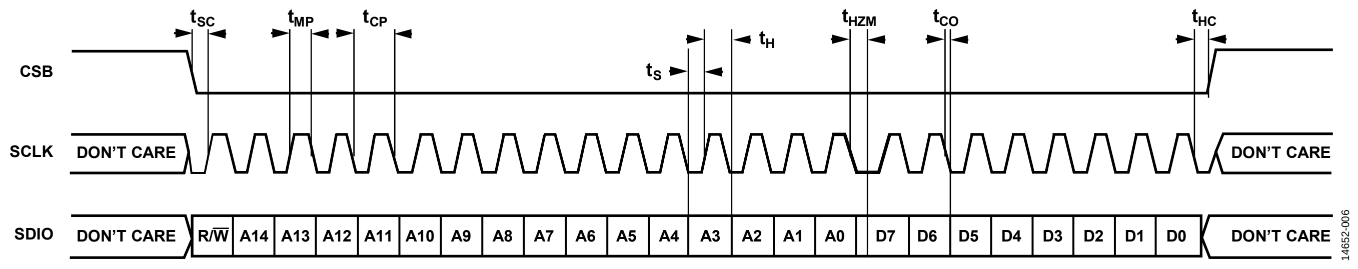


Figure 7. 3-Wire SPI Timing with Parameter Labels, SPI Read Operation

JESD204B INTERFACE

The device employs the JESD204B Subclass 1 standard to transfer ADC and DAC samples between the device and a baseband processor. JESD204B Subclass 1 devices use a system reference (SYSREF) signal to synchronize the establishment of the links to provide deterministic latency through the link. For details on deterministic latency, refer to Section 6 of the JEDEC Standard No. 204B.

The device supports high speed serial lane rates from 614.4 Mbps to 6144 Mbps. An external clock distribution solution provides a device clock and a SYSREF to both the device and the baseband processor. The SYSREF signal ensures deterministic latency between the transceiver and the baseband processor. This signal is also used to provide digital synchronization when more than one device is used; it is also required to maintain data timing synchronization among the devices. The Multichip Synchronization section describes the setup required to achieve the desired results.

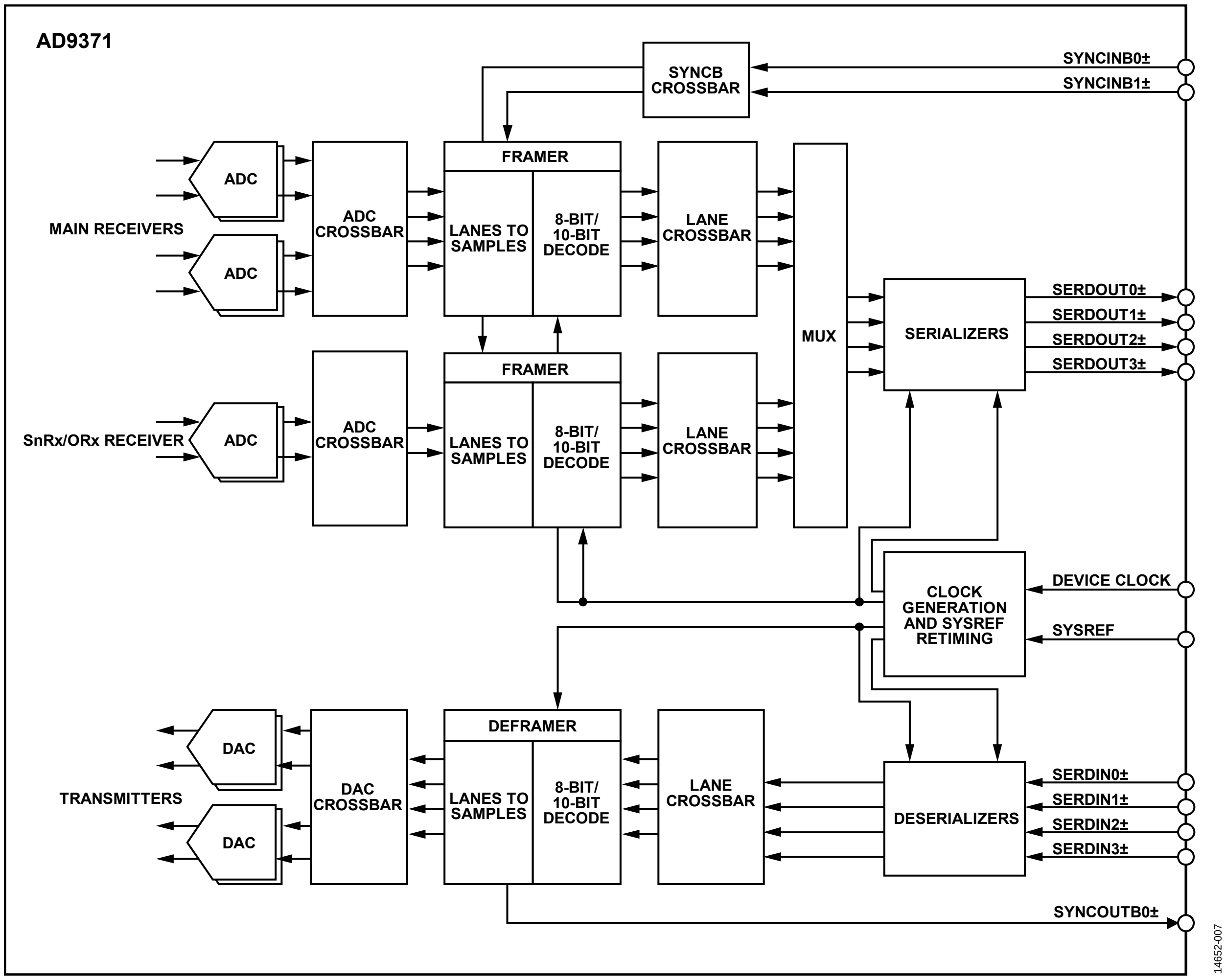


Figure 8. High Level JESD204B Interface Block Diagram

RECEIVERS (ADC) DATAPATH

The transport and link layers for JESD204B are performed in the framers. The device has two JESD204B framers that multiplex into four serial lanes. Samples from the main receivers are sent to the first framer. Samples from the sniffer/observation receiver are sent to the second framer. Each framer has its own SYNCB signal. This allows the sniffer/observation JESD204B lanes to be brought down for reconfiguration without interrupting the main receiver lanes.

The two framers are capable of operating at different sample rates. The higher sample rate must be a power of two multiples of the lower sample rate (for example, 2×, 4×, or 8×). For example, the two main receivers can be configured for a 122.88 MHz sample rate into the framer, and a sniffer with a sample rate of 30.72 MHz can be connected to the second framer. There are two options to make this work: oversample at the framer input or bit repeat at the serializer output. Oversample mode repeats sample values at the framer input of the slower rate, allowing all the serializers to run at the same bit rate. Bit repeat mode repeats each bit in the lane or lanes that carry the slower data as it exits the serializer. Because this is after the 8-bit/10-bit encoding, it appears as if the lane is running at a slower data rate than the other lanes.

Both framers must share the four serializers. Each framer must be configured for 0, 1, 2, or 4 lanes such that the two framers combine for no more than 4 lanes. If one framer uses all 4 lanes, then the other framer cannot be used.

Each framer has an ADC crossbar that can connect any ADC to any framer input. The ADC crossbar for the main receiver framer has an optional automatic channel selection feature to automatically shift the Rx2 ADCs to Framer 0 and Framer 1. If the framer is configured for only two inputs (M = 2), Rx1 is disabled, and Rx2 is enabled, the ADCs for Rx2 must be connected to the Framer 0 and Framer 1 inputs. The automatic channel selection feature allows this shift without reconfiguring the ADC crossbar.

Each framer is capable of generating a pseudorandom bit sequence (PRBS) on the enabled lanes. After the PRBS is enabled, errors can be injected. Enabling the PRBS generator disables the normal JESD204B framing, and may cause the SYNCB signal to deassert.

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

The device supports a subset of possible JESD204B link configurations. The number of ADCs and the number of JESD204B lanes implemented in the silicon limit these configurations.

Table 5. Static JESD204B Parameters

JESD204B Parameter	AD9371/AD9375 Value	Description
S	1	Samples transmitted/single converter/frame cycle
N	14	Converter resolution
N′	16	Total number of bits per sample
CF	0	Number of control words/frame clock cycle/converter device
CS	2	Number of control bits/conversion sample
HD	0 … 1	High density mode (only M2L4 uses HD = 1)
K	Variable, suggested: 32	Number of frames in 1 multiframe, ($20 \leq F \times K \leq 256$), $F \times K$ must be a multiple of 4, $K \leq 32$

Table 6. JESD204B Parameters Dependent on Number of Lanes and Number of ADCs

Number of ADCs (M)	Number of Lanes (L)	Number of Bytes in 1 Frame (F) ($F = 2 \times M/L$)
2	1	4
2	2	2
2	4	1
4	1	8
4	2	4
4	4	2

For a particular converter sample rate, not all combinations listed in Table 6 are valid. For the JESD204B configuration mode to be valid, the lane rate for that mode must be within the 614.4 Mbps to 6144 Mbps range. The lane rate is the serial bit rate for one lane of the JESD204B link. Calculate the lane rate using Equation 1.

$$Lane\ Rate = IQ\ Sample\ Rate \times M \times 16\text{ bits} \times (10 \div 8) \div L \qquad (1)$$

Serializer Configuration

A 5-bit number that is not linearly weighted represents the amplitude of the serializer. Not all settings are unique, and not all settings meet the JESD204B transmitter mask. The JESD204B transmitter mask requires a differential amplitude greater than 360 mV and less than 770 mV. To meet the JESD204B transmitter mask, it is recommended to set the serializer amplitude to a decimal value between 18 to 26. The default amplitude is 22 mV p-p.

Table 7. Serializer Amplitude Settings That Meet the JESD204B Transmitter Mask

Serializer Amplitude (Decimal)	Differential Amplitude (mV p-p)
18	400
19	440
20	480
21	520
22 (default)	560
23	600
24	640
25	680
26	720

The values shown in Table 7 are calculated values based on the design. Measured values are slightly lower than the calculated values. It is always recommended to verify the eye diagram in the system after building a PCB to verify any layout related performance differences.

The serializer preemphasis allows boosting the amplitude any time the serial bit changes state. If bit transition does not occur, the amplitude is deemphasized. Preemphasis helps open the eye diagram 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 measuring the eye diagram with a high speed scope at the receiver is recommended. A 3-bit number represents the serializer preemphasis. The range in differential amplitude can be seen in Table 8, and its effects are shown in Figure 9.

Table 8. Preemphasis Amplitude Settings

Emphasis (Decimal)	Differential Amplitude (mV p-p)
0	0
1	40
2	80
3	120
4	160
5	200
6	240
7	280

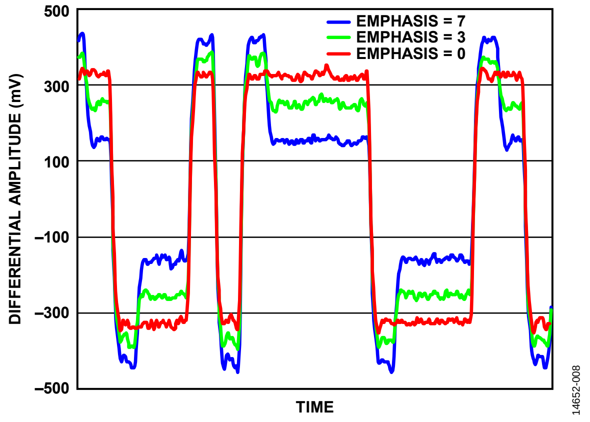


Figure 9. Serializer Preemphasis Measured on 3 Gbps Serial Data, Serializer

Another metric for the effect of the serializer preemphasis is how much insertion loss each preemphasis setting can overcome. Different PCBs have different insertion loss due to factors such as different materials, stackups, and trace geometry. However, the insertion loss can be measured with a network analyzer or simulated to estimate how much loss a particular PCB has. Note that the preemphasis gain has some dependency on the main serializer amplitude setting.

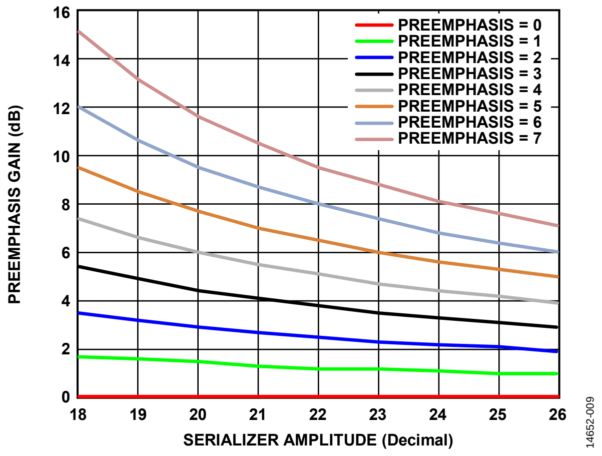


Figure 10. Gain (in dB) of Each Preemphasis and Amplitude Setting

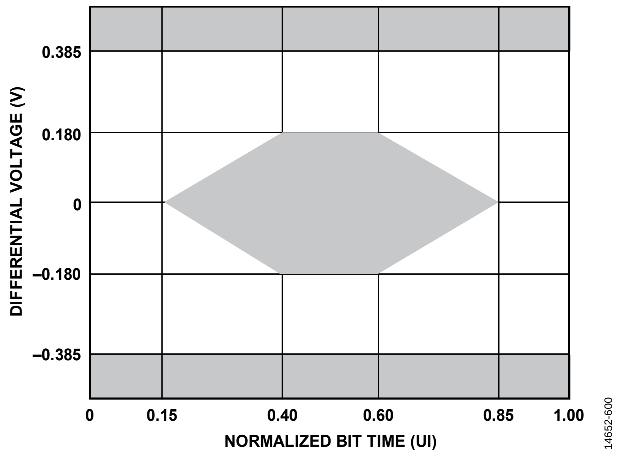


Figure 11. Serializer Eye Diagram Requirements Mask

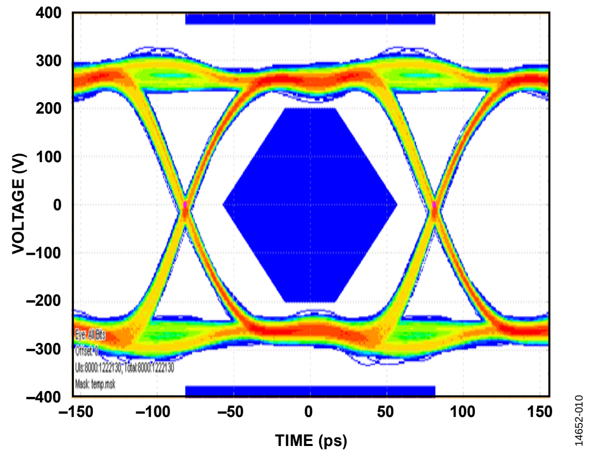


Figure 12. Example 6.144 Gbps Eye Diagram at the Serializer Output with Specifications Mask Superimposed

Framer

The framer receives 16-bit ADC samples and maps them to high speed serial lanes. The mapping changes depending on the JESD204B configuration chosen, specifically the number of lanes and the number of converters. Table 6 summarizes the valid framer configurations for the device.

The responsibilities of the framer include the following:

- JESD204B link initialization—state machine to progress link from code group synchronization (CGS) to initial lane assignment sequence (ILAS), then to user data.
- Character replacement. This allows frame and multiframe synchronization during user data.
- Map ADC samples to JESD204B lanes.
- Perform 8-bit/10-bit encoding.

The ADC sample inputs into the framer pass through a sample crossbar, allowing the framer to map any ADC input to any framed sample location during the framing process. For example, this can be used to swap I and Q samples. The framer lane data outputs also pass through a lane crossbar, allowing mapping any framer output lane (internal to the silicon) to any physical JESD204B lane at the package pin. The framer packs the ADC samples into lane data following the JESD204B specification.

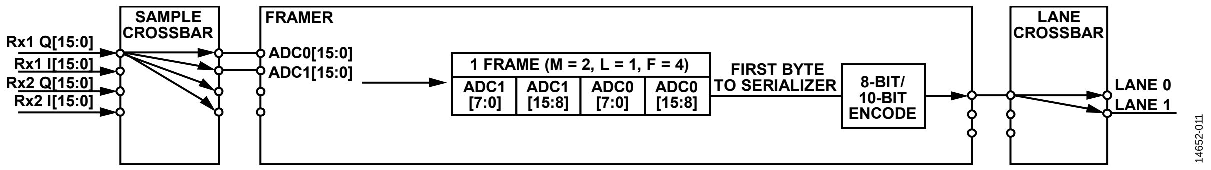


Figure 13. Framer Data Packing for M = 2, L = 1

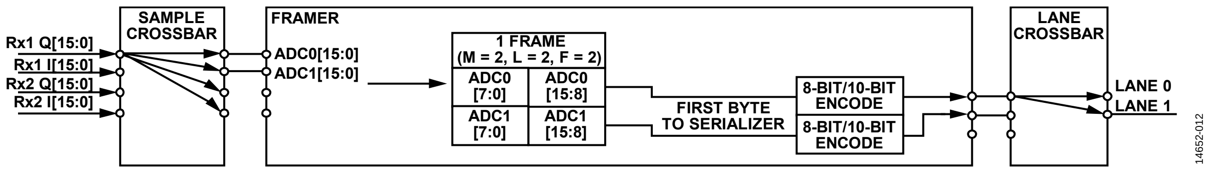


Figure 14. Framer Data Packing for M = 2, L = 2

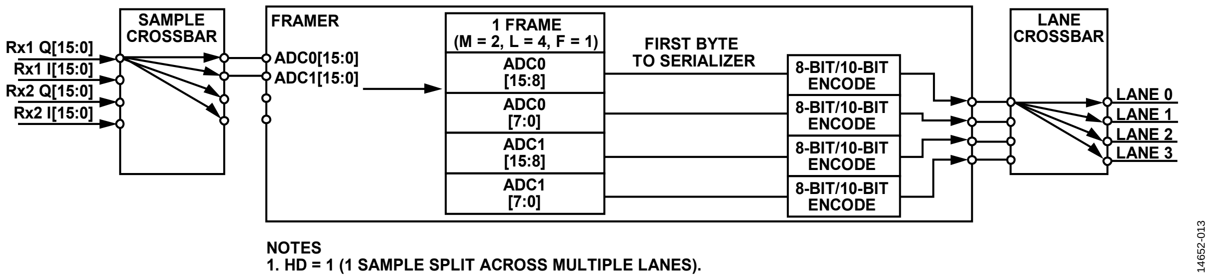


Figure 15. Framer Data Packing for M = 2, L = 4

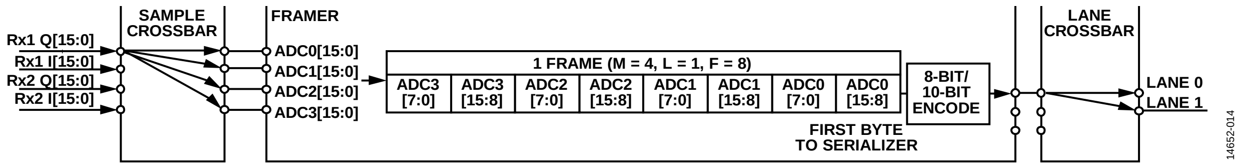


Figure 16. Framer Data Packing for M = 4, L = 1

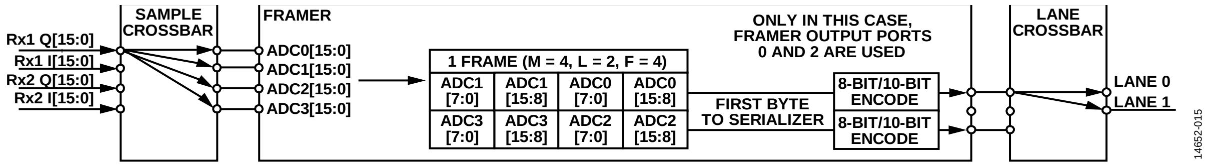


Figure 17. Framer Data Packing for M = 4, L = 2

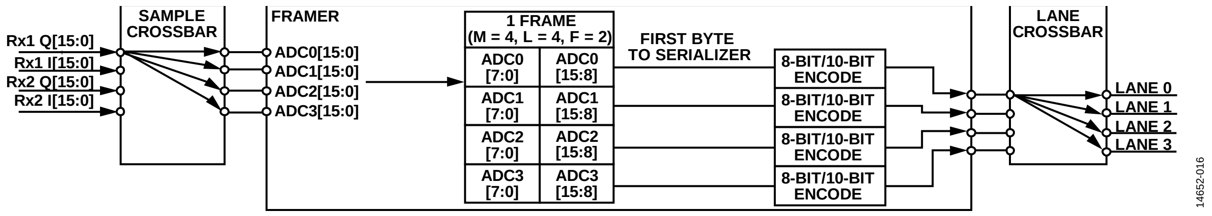


Figure 18. Framer Data Packing for M = 4, L = 4

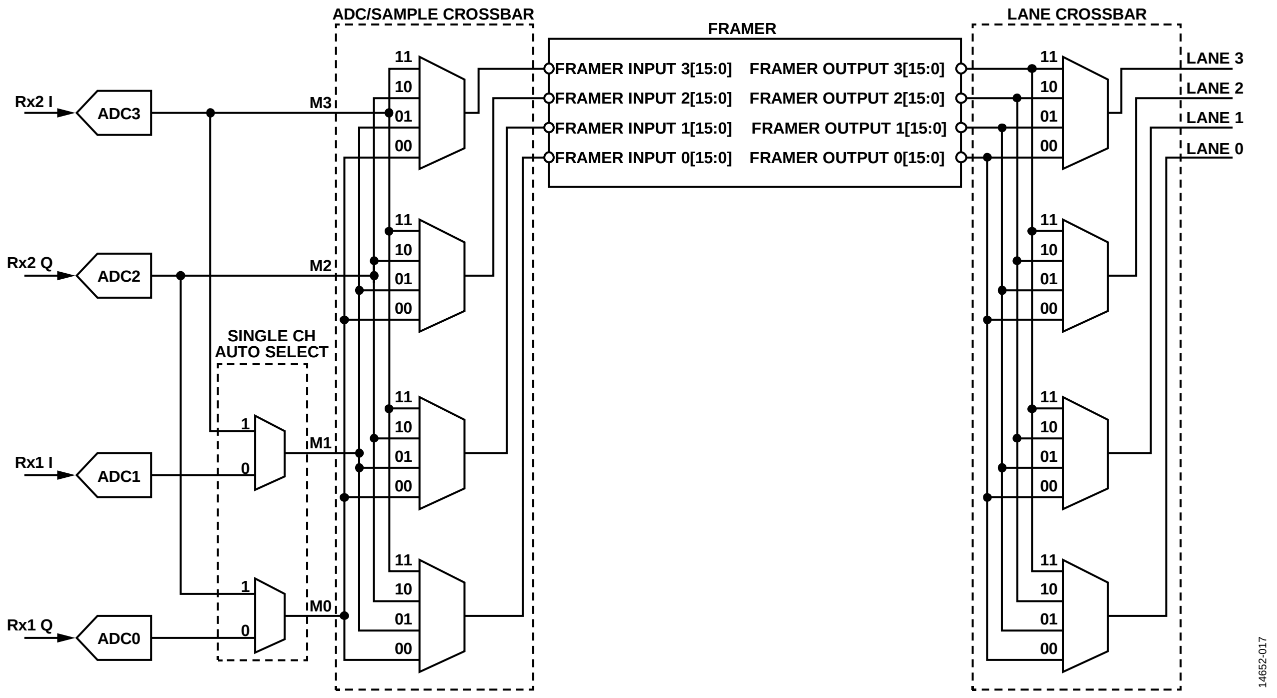


Figure 19. Framer Crossbar Detail

Other Useful Framer IP Features

Serializer PRBS

The serializer has a built in pseudorandom bit sequence (PRBS) test pattern it can output to aid in debugging the JESD204B serial link. If errors caused by signal integrity exist, it may be difficult to get the JESD204B framer/deframer to work properly. The PRBS generator built into the serializer allows the device to output serial data, even when the link may be causing bit errors. The PRBS generator can be configured to transmit PRBS7, PRBS15, or PRBS31 sequences. With this mode enabled, the serializer amplitude and emphasis can be adjusted to find the best setting to minimize bit errors on the serial link. For this mode to be fully utilized, the baseband processor must have a PRBS checker to check the PRBS sequence for errors.

The typical usage sequence is as follows:

1. Initialize the device as outlined in the Link Establishment section.
2. Run the MYKONOS_enableRxFramerPrbs(…) with the required PRBS order and set it to enable = 1.
3. Enable the PRBS checker on the baseband processor and reset its error count.
4. Wait a specific amount of time to allow a good number of samples to be transmitted, and then check the PRBS error count of the baseband processor.

API Software Integration

The MYKONOS_initialize(…) API function handles the configuration of the serializer, Rx1/Rx2 framer, and ORx framer. Set any JESD204B link options in the mykonosDevice_t data structure before calling MYKONOS_initialize(…). After initialization, there are some other API functions to aid in debugging and monitoring the status of the JESD204B link.

JESD204B Framer API Data Structures

mykonosJesd204bFramerConfig_t

The mykonosJesd204bFramerConfig_t data structure contains the information required to properly configure each framer. Details of each member can be found in Table 9. The transceiver evaluation software (TES) has the option to output example data structures with values chosen from the configuration tab of the software.

```
typedef struct
{
    uint8_t bankId;
    uint8_t deviceId;
    uint8_t lane0Id;
    uint8_t M;
    uint8_t K;
    uint8_t scramble;
    uint8_t externalSysref;
    uint8_t serializerLanesEnabled;
    uint8_t serializerLaneCrossbar;
    uint8_t serializerAmplitude;
    uint8_t preEmphasis;
    uint8_t invertLanePolarity;
    uint8_t lmfcOffset;
    uint8_t newSysrefOnRelink;
    uint8_t enableAutoChanXbar;
    uint8_t ObsRxSyncbSelect;

    uint8_t overSample;
} mykonosJesd204bFramerConfig_t;
```

Table 9. JESD204B Framer Configuration Structure Member Description

Structure Member	Valid Values	Description
bankId	0 … 15	JESD204B configuration bank ID—extension to device ID.
deviceId	0 … 255	JESD204B configuration device ID—link identification number.
lane0Id	0 … 31	JESD204B configuration lane ID—if more than one lane is used, each subsequent lane increments from this number.
M	0, 2, 4	Number of ADC converters—two converters per receive chain.
K	1 … 32	Number of frames in a multiframe; the default value is 32. F × K must be a multiple of 4.
scramble	0.. …	Scrambling enabled. If scramble = 0, then scrambling is disabled. If scramble > 0, then scrambling is enabled.
externalSysref	0 … 255	External SYSREF enabled. If externalSysref = 0, then use internal SYSREF. If externalSysref > 0, then use external SYSREF.
serializerLanesEnabled	0x0 … 0xF	Serializer lane enabled—one bit per lane. If Bit 0 = 0, then Lane 0 is disabled; if Bit 0 = 1, then Lane 0 is enabled. If Bit 1 = 0, then Lane 1 is disabled; if Bit 1 = 1, then Lane 1 is enabled. If Bit 2 = 0, then Lane 2 is disabled; if Bit 2 = 1, then Lane 2 is enabled. If Bit 3 = 0, then Lane 3 is disabled; if Bit 3 = 1, then Lane 3 is enabled.

Structure Member	Valid Values	Description
serializerLaneCrossbar	0x0 … 0xFF	Serializer lane crossbar—two bits per lane. Bits[1:0] identify the framer lane that connects to Serializer Lane 0. Bits[3:2] identify the framer lane that connects to Serializer Lane 1. Bits[5:4] identify the framer lane that connects to Serializer Lane 2. Bits[7:6] identify the framer lane that connects to Serializer Lane 3.
serializerAmplitude	0 … 31	Serializer amplitude—default is 22.
preEmphasis	0 … 7	Serializer preemphasis—default is 4.
invertLanePolarity	0x0 … 0xF	Serializer lane polarity inversion. If Bit 0 = 0, then Lane 0 is unaffected; if Bit 0 = 1, then Lane 0 is inverted. If Bit 1 = 0, then Lane 1 is unaffected; if Bit 1 = 1, then Lane 1 is inverted. If Bit 2 = 0, then Lane 2 is unaffected; if Bit 2 = 1, then Lane 2 is inverted. If Bit 3 = 0, then Lane 3 is unaffected; if Bit 3 = 1, then Lane 3 is inverted.
lmfcOffset	0 … 31	Local multiframe counter (LMFC) offset—local multi frame counter offset value for deterministic latency setting, set this such that 0 ≤ lmfcOffset ≤ (K − 1).
newSysrefOnRelink	0 … 255	New SYSREF on relink—flag to indicate that a SYSREF is required to reestablish the link. If newSysrefOnRelink = 0, then no SYSREF is required. If newSysrefOnRelink > 0, then SYSREF is required.
enableAutoChanXbar	0 … 255	Enable automatic channel select for the ADC crossbar. If enableAutoChanXbar = 0, then the auto channel selection is disabled. If enableAutoChanXbar > 0, then the auto channel selection is enabled.
obsRxSyncbSelect	0 … 1	SYNCB selection—selects which SYNCINB input is connected to the framer. If obsRxSyncbSelect = 0, then SYNCINB0 is connected to the framer. If obsRxSyncbSelect = 1, then SYNCINB1 is connected to the framer.
overSample	0 … 1	Oversample mode—selects which method is chosen when oversample or bit repeat is required. If overSample = 0, then bit repeat mode is selected. If overSample = 1, then oversample is selected.

API Functions

MYKONOS_setupJesd204bFramer(…)

```
mykonosErr_t
   MYKONOS_setupJesd204bFramer(mykonosDevice
   _t *device);
```

This function is called directly from MYKONOS_Intialize(). It is not necessary to call this function if MYKONOS_Intialize() is used. This function sets up the JESD204B Rx framer.

DEPENDENCIES

The dependencies for the MYKONOS_ setupJesd204bFramer(…) function are as follows:

- device → spiSettings
- device → spiSettings → chipSelectIndex
- device → rx → framer → M
- device → rx → realIfData
- device → rx → framer → bankId
- device → rx → framer → lane0Id
- device → rx → framer → serializerLanesEnabled
- device → rx → framer → obsRxSyncbSelect
- device → rx → framer → K
- device → rx → framer → externalSysref
- device → rx → rxChannels
- device → rx → framer → newSysrefOnRelink
- device → rx → framer → enableAutoChanXbar
- device → rx → framer → lmfcOffset
- device → rx → framer → scramble

Table 10. MYKONOS_setupJesd204bFramer(…) Parameters

Parameter	Description
device	Pointer to the device settings structure

Table 11. MYKONOS_setupJesd204bFramer(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_FRAMER_INV_REAL_IF_DATA_PARM	Invalid framer M, M can only = 1 in real IF mode
MYKONOS_ERR_FRAMER_INV_M_PARM	Invalid framer M (valid 1, 2, 4)
MYKONOS_ERR_FRAMER_INV_BANKID_PARM	Invalid BankId (valid 0 to 15)
MYKONOS_ERR_FRAMER_INV_LANEID_PARM	Invalid Lane0Id (valid 0 to 31)
MYKONOS_ERR_RXFRAMER_INV_FK_PARAM	Invalid F × K value (F × K must be >20 and divisible by 4)
MYKONOS_ERR_FRAMER_INV_K_OFFSET_PARAM	Invalid K offset, must be less than K

MYKONOS_setupJesd204bObsRxFramer(…)

```
mykonosErr_t
   MYKONOS_setupJesd204bObsRxFramer(mykonosD
   evice_t *device);
```

This function is called directly from `MYKONOS_Intialize()`. It is not necessary to call this function if `MYKONOS_Intialize()` is used. This function sets up the JESD204B OBSRX framer.

DEPENDENCIES

The dependencies for the MYKONOS_ setupJesd204bObsRxFramer(…) function are as follows:

- device → spiSettings
- device → spiSettings → chipSelectIndex
- device → rx → framer → M
- device → rx → realIfData
- device → rx → framer → bankId
- device → rx → framer → lane0Id
- device → rx → framer → serializerLanesEnabled
- device → rx → framer → obsRxSyncbSelect
- device → rx → framer → K
- device → rx → framer → externalSysref
- device → rx → rxChannels
- device → rx → framer → newSysrefOnRelink
- device → rx → framer → lmfcOffset
- device → rx → framer → scramble

Table 12. MYKONOS_setupJesd204bObsRxFramer(…) Parameters

Parameter	Description
device	Pointer to the device settings structure

Table 13. MYKONOS_setupJesd204bObsRxFramer(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_OBSRX_FRAMER_INV_REAL_IF_DATA_PARM	Invalid framer M, M can only = 1 in real IF mode
MYKONOS_ERR_OBSRX_FRAMER_INV_M_PARM	Invalid framer M (valid 1, 2, 4)
MYKONOS_ERR_OBSRX_FRAMER_INV_BANKID_PARM	Invalid BankId (valid 0 to 15)
MYKONOS_ERR_OBSRX_FRAMER_INV_LANEID_PARM	Invalid Lane0Id (valid 0 to 31)
MYKONOS_ERR_OBSRX_RXFRAMER_INV_FK_PARAM	Invalid F × K value (F × K must be >20 and divisible by 4)
MYKONOS_ERR_OBSRXFRAMER_INV_K_OFFSET_PARAM	Invalid K offset, must be less than K

MYKONOS_setupSerializers(…)

```
mykonosErr_t
   MYKONOS_setupSerializers(mykonosDevice_t
   *device);
```

This function is called directly from MYKONOS_Intialize(). It is not necessary to call this function if MYKONOS_Intialize() is used. This function sets up the JESD204B serializers. This function uses the Rx framer and ObsRx framer structures to setup the four serializer lanes that are shared between the two framers. If the Rx profile is valid, the serializer amplitude and preemphasis are used from the Rx framer. If only the ObsRx profile is valid, the obsRx framer settings are used.

DEPENDENCIES

The dependencies for the MYKONOS_setupSerializers(…) function are as follows:

- device → spiSettings
- device → spiSettings → chipSelectIndex
- device → rx → framer → M
- device → rx → framer → serializerAmplitude
- device → rx → framer → preEmphasis
- device → rx → framer → serializerLanesEnabled
- device → rx → framer → invertLanePolarity
- device → obsrx → framer → M
- device → obsrx → framer → serializerAmplitude
- device → obsrx → framer → preEmphasis
- device → obsrx → framer → serializerLanesEnabled
- device → obsrx → framer → invertLanePolarity

Table 14. MYKONOS_setupSerializers(…) Parameters

Parameter	Description
device	Pointer to the device settings structure

Table 15. MYKONOS_setupSerializers(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_INITSER_INV_VCODIV_PARM	CLKPLL has invalid VCO divider, verify CLKPLL configuration
MYKONOS_ERR_SER_LANE_CONFLICT_PARM	When both Rx and ObsRx framers are enabled, framers must not share the same physical lane
MYKONOS_ERR_SER_INV_REAL_IF_DATA_PARM	Rx Framer M can only = 1 when real IF mode is enabled
MYKONOS_ERR_SER_INV_M_PARM	Invalid Rx Framer M (valid 1, 2, 4)
MYKONOS_ERR_SER_INV_LANEEN_PARM	Invalid Rx framer serializerLanesEnabled (valid 0 to 15)
MYKONOS_ERR_SER_INV_AMP_PARM	Invalid Rx serializer amplitude (valid 0 to 31)
MYKONOS_ERR_SER_INV_PREEMP_PARM	Invalid Rx serializer preemphasis (valid 0 to 7)
MYKONOS_ERR_SER_INV_LANEPN_PARM	Invalid Rx serializer pseudonoise (PN) invert setting (valid 0 to 15)
MYKONOS_ERR_SER_INV_L_PARM	Invalid Rx serializer lanes enabled (must use 1, 2, or 4 lanes)
MYKONOS_ERR_SER_INV_LANERATE_PARM	Invalid Rx serializer lane rate (valid 614.4 Mbps to 6144 Mbps)
MYKONOS_ERR_SER_LANE_RATE_CONFLICT_PARM	Necessary lane rates for Rx and ObsRx framer cannot be obtained with possible divider settings
MYKONOS_ERR_SER_INV_HSCLK_PARM	Invalid HSCLK frequency (check CLKPLL config, HSCLK must be ≤ 6144 GHz)
MYKONOS_ERR_HS_AND_LANE_RATE_NOT_INTEGER_MULT	HSCLK is not an integer multiple of the lane clock rate
MYKONOS_ERR_SER_INV_TXSER_DIV_PARM	No valid Tx serializer divider to obtain desired lane rates
MYKONOS_ERR_INITSER_INV_PROFILE	Rx/ObsRx and sniffer profiles are not valid, cannot configuration serializers
MYKONOS_ERR_INV_RXFRAMER_PCLKDIV_PARM	Invalid Rx framer PCLK divider
MYKONOS_ERR_INV_OBSRXFRAMER_PCLKDIV_PARM	Invalid ORx/sniffer framer PCLK divider

MYKONOS_enableSysrefToRxFramer(…)

```
mykonosErr_t
   MYKONOS_enableSysrefToRxFramer(mykonosDev
   ice_t *device, uint8_t enable);
```

This function can gate or allow the SYSREF at the outside of the device to reach the SYSREF input in the Rx framer IP in the device. Typically, the framers ignore SYSREF (enable = 0) until after multichip sync is completed. When the baseband processor (BBP) is ready to bring up the JESD204B link, call this function with enable = 1 to allow the device Rx framer to retime the local multiframe counter (LMFC) to the SYSREF.

DEPENDENCIES

The dependencies for the MYKONOS_ enableSysrefToRxFramer(…) function are as follows:

- device → spiSettings

Table 16. MYKONOS_enableSysrefToRxFramer(…) Parameters

Parameter	Description
device	Pointer to the device settings structure
enable	1 enables SYSREF to the Rx framer, 0 disables SYSREF to the Rx framer

Table 17. MYKONOS_enableSysrefToRxFramer(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully

MYKONOS_enableSysrefToObsRxFramer(…)

```
mykonosErr_t
   MYKONOS_enableSysrefToObsRxFramer(mykonos
   Device_t *device, uint8_t enable);
```

This function gates or allows the SYSREF at the outside of the device to reach the SYSREF input in the ORx framer IP in the device. Typically, the framers ignore SYSREF (enable = 0) until after multichip sync. When the baseband processor is ready to bring up the JESD204B link, call this function with enable = 1 to allow the ORx framer to retime the local multiframe counter to the SYSREF.

DEPENDENCIES

The dependencies for the MYKONOS_enableSysrefToObsRx-Framer(…) function are as follows:

- device → spiSettings

Table 18. MYKONOS_enableSysrefToObsRxFramer(…) Parameters

Parameter	Description
device	Pointer to the device settings structure
enable	1 enables the SYSREF to the ORx framer, 0 disables the SYSREF to the ORx framer

Table 19. MYKONOS_enableSysrefToObsRxFramer(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully

MYKONOS_enableRxFramerLink(…)

```
mykonosErr_t
   MYKONOS_enableRxFramerLink(mykonosDevice_
   t *device, uint8_t enable);
```

This function is normally not necessary. In the event that the link must be reset, this function allows the Rx framer to be disabled and reenabled.

DEPENDENCIES

The dependencies for the MYKONOS_ enableRxFramerLink(…) function are as follows:

- device → spiSettings
- device → rx → framer → serializerLanesEnabled

Table 20. MYKONOS_enableRxFramerLink(…) Parameters

Parameter	Description
device	Pointer to the device settings structure
enable	1 enables the Rx framer, 0 disables the Rx framer

Table 21. MYKONOS_enableRxFramerLink(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_ENFRAMERLINK_INV_LANESEN_PARAM	Invalid serializerLanesEnabled parameter in the device data structure (valid 0 to 15)

MYKONOS_enableObsRxFramerLink(…)

```
mykonosErr_t
   MYKONOS_enableObsRxFramerLink(mykonosDevi
   ce_t *device, uint8_t enable);
```

This function is normally not necessary. In the event that the link must be reset, this function allows the ORx framer to be disabled and reenabled.

DEPENDENCIES

The dependencies for the MYKONOS_ enableObsRxFramerLink(…) function are as follows:

- device → spiSettings
- device → obsRx → framer → serializerLanesEnabled

Table 22. MYKONOS_enableObsRxFramerLink(…) Parameters

Parameter	Description
device	Pointer to the device settings structure
enable	1 enables the ORx framer, 0 disables the ORx framer

Table 23. MYKONOS_enableObsRxFramerLink(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_ENFRAMERLINK_INV_LANESEN_PARAM	Invalid serializerLanesEnabled parameter in the device data structure (valid 0 to 15)

MYKONOS_readRxFramerStatus (…)

```
mykonosErr_t
   MYKONOS_readRxFramerStatus(mykonosDevice_
   t *device, uint8_t *framerStatus);
```

This function reads back the Rx framer status to determine the state of the JESD204B Rx framer link. The framerStatus return value returns an 8-bit status word.

DEPENDENCIES

The dependencies for the MYKONOS_readRxFramerStatus (…) function are as follows:

- device → spiSettings

Table 24. MYKONOS_readRxFramerStatus (…) Parameters

Parameter	Description
device	Pointer to the device settings structure
framerStatus	Rx framer status byte as described in Table 25

Table 25. Rx Framer Status Byte

framerStatus	Description
7	SYSREF phase error—a new SYSREF has different timing than the first that set the local multiframe counter (LMFC) timing.
6	Framer lane first in, first out (FIFO) read/write pointer delta has changed. Can help debug issues with deterministic latency.
5	Framer has received the SYSREF and has retimed its LMFC.
[4:2]	Framer initial lane assignment sequence (ILAS) state. 0 = code group synchronization (CGS). 1 = 1st multframe. 2 = 2nd multiframe. 3 = 3rd multiframe. 4 = 4th multiframe. 5 = last multiframe. 6 = invalid. 7 = ILAS complete.
[1:0]	Framer Tx state. 0 = CGS. 1 = ILAS. 2 = ADC data.

Table 26. MYKONOS_readRxFramerStatus (…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_READ_RXFRAMERSTATUS_NULL_PARAM	Function parameter framerStatus has null pointer

MYKONOS_readObsRxFramerStatus (…)

```
mykonosErr_t
   MYKONOS_readObsRxFramerStatus(mykonosDevi
   ce_t *device, uint8_t *framerStatus);
```

This function read back the observation Rx framer status to determine the state of the JESD204B ORx framer link. The framerStatus return value returns an 8-bit status word.

DEPENDENCIES

The dependencies for the MYKONOS_readObsRxFramerStatus (…) function are as follows:

- device → spiSettings

Table 27. MYKONOS_readObsRxFramerStatus (…) Parameters

Parameter	Description
device	Pointer to the device settings structure
obsFramerStatus	ORx framer status byte as described in Table 28

Table 28. ORx Framer Status Byte

obsFramerStatus	Description
7	SYSREF phase error—a new SYSREF had different timing than the first that set the local multiframe counter timing.
6	Framer lane FIFO read/write pointer delta has changed. Can help debug issues with deterministic latency.
5	Framer has received the SYSREF and has retimed its local multiframe counter.
[4:2]	Framer initial lane assignment sequence (ILAS) state. 0 = code group synchronization (CGS). 1 = 1st multframe. 2 = 2nd multiframe. 3 = 3rd multiframe. 4 = 4th multiframe. 5 = last multiframe. 6 = invalid. 7 = ILAS complete.
[1:0]	Framer Tx state. 0 = CGS. 1 = ILAS. 2 = ADC data.

Table 29. MYKONOS_readObsRxFramerStatus (…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_READ_ORXFRAMERSTATUS_NULL_PARAM	Function parameter obsFramerStatus has null pointer

MYKONOS_enableRxFramerPrbs (…)

```
mykonosErr_t
   MYKONOS_enableRxFramerPrbs(mykonosDevice_
   t *device, mykonosPrbsOrder_t polyOrder,
   uint8_t enable);
```

After the serializer and framer are configured with MYKONOS_initialize(…), this function can be called to enable the Rx framer to output a pseudorandom bit sequence (PRBS) pattern on the serializer lanes. Only the serializer lanes connected to the Rx framer are output. Review the lane crossbar settings configured in the Rx framer data structure to verify which lanes are affected.

The PRBS order can be set to MYK_PRBS7, MYK_PRBS15, or MYK_PRBS31 by the enumerated list value passed into the polyOrder parameter. The PRBS generator can be enabled (1) or disabled (0) by the enable parameter.

Note that if both the Rx framer and ORx framer crossbar settings overlap and use the same lanes, the data from the two framers is logically OR'ed together. This results in unexpected data being output from the serializer lanes.

DEPENDENCIES

The dependencies for the MYKONOS_enableRxFramerPrbs (…) function are as follows:

- device → spiSettings

Table 30. MYKONOS_enableRxFramerPrbs (…) Parameters

Parameter	Description
device	Pointer to the device settings structure
polyOrder	Selects the pseudorandom bit sequence (PRBS) type based on enumeration value: MYK_PRBS7, MYK_PRBS15, MYK_PRBS31
enable	1 enables PRBS generator in the Rx framer, 0 disables the PRBS generator

Table 31. MYKONOS_enableRxFramerPrbs (…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_RX_FRAMER_INV_PRBS_POLYORDER_PARAM	Invalid polyOrder parameter, use proper enumerator

MYKONOS_enableObsRxFramerPrbs (…)

```
mykonosErr_t
   MYKONOS_enableObsRxFramerPrbs(mykonosDevi
   ce_t *device, mykonosPrbsOrder_t
   polyOrder, uint8_t enable);
```

After the serializer and framer are configured with MYKONOS_initialize(…), this function can be called to enable the ORx framer to output a pseudorandom bit sequence (PRBS) pattern on the serializer lanes. Only the serializer lanes connected to the ORx framer are output. Review the lane crossbar settings configured in the ORx framer data structure to verify which lanes are affected.

The PRBS order can be set to MYK_PRBS7, MYK_PRBS15, or MYK_PRBS31 by the enumeration value passed into the polyOrder parameter. The PRBS generator can be enabled (1) or disabled (0) by the enable parameter.

Note that if both the Rx framer and the ORx framer crossbar settings overlap and use the same lanes, the data from the two framers is logically OR'ed together. This results in unexpected data being output from the serializer lanes.

DEPENDENCIES

The dependencies for the MYKONOS_enableObsRxFramerPrbs (…) function are as follows:

- device → spiSettings

Table 32. MYKONOS_enableObsRxFramerPrbs(…) Parameters

Parameter	Description
device	Pointer to the device settings structure
polyOrder	Selects the pseudorandom bit sequence (PRBS) type based on enumeration value: MYK_PRBS7, MYK_PRBS15, MYK_PRBS31
enable	1 enables the PRBS generator in the ORx framer, 0 disables the PRBS generator

Table 33. MYKONOS_enableObsRxFramerPrbs(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_OBSRX_FRAMER_INV_PRBS_POLYORDER_PARAM	Invalid polyOrder parameter, use proper enumerator

MYKONOS_rxInjectPrbsError

```
mykonosErr_t
   MYKONOS_rxInjectPrbsError(mykonosDevice_t
   *device);
```

To verify pseudorandom bit sequence (PRBS) is working correctly on a good link, errors can be injected to force the PRBS checker to increment. Calling this function injects one to three errors into the framer PRBS generation.

DEPENDENCIES

The dependencies for the MYKONOS_rxInjectPrbsError function are as follows:

- device → spiSettings

Table 34. MYKONOS_rxInjectPrbsError Parameters

Parameter	Description
device	Pointer to the device settings structure

Table 35. MYKONOS_rxInjectPrbsError Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully

MYKONOS_obsRxInjectPrbsError

```
mykonosErr_t
   MYKONOS_obsRxInjectPrbsError(mykonosDevic
   e_t *device);
```

To verify PRBS is working correctly on a good link, errors can be injected to force the PRBS checker to increment. Calling this function injects one to three errors into the ORx framer PRBS generation.

DEPENDENCIES

The dependencies for the MYKONOS_obsRxInjectPrbsError function are as follows:

- device → spiSettings

Table 36. MYKONOS_obsRxInjectPrbsError Parameters

Parameter	Description
device	Pointer to the device settings structure

Table 37. MYKONOS_obsRxInjectPrbsError Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully

TRANSMITTERS (DAC) DATAPATH

The device has one JESD204B deframer configurable for up to four lanes and four DAC converters. All converters must run at the same sample rate. Likewise, all lanes must run at the same data rate. The deframer is capable of receiving a pseudorandom bit sequence (PRBS) sequence and accumulating error counts. The deserializers have adjustable equalization circuits (fixed setting, not adaptive) to counteract the insertion loss due to various PCB trace lengths and materials.

Supported Deframer Link Parameters

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

For a particular converter sample rate, not all combinations listed in Table 39 are valid. For the JESD204B configuration mode to be valid, the lane rate for that mode must be within the 614.4 Mbps to 6144 Mbps range. The lane rate is the serial bit rate for one lane of the JESD204B link. Calculate the lane rate using Equation 1.

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 clock PLL, but there are separate dividers to generate the deserializer clock data recovery (CDR) clock and the serializer clock.

Deserializer Configuration

The deserializer includes an equalizer that can be set to a fixed setting. Table 40 summarizes the amount of insertion loss each equalizer setting can overcome. Note that the measured length is the value at which the eye diagram is nearly failing the receive mask for each equalizer setting.

Deframer

The deframer receives 8-bit/10-bit encoded data from the deserializer and decodes the data into 16-bit DAC samples. Because the DAC samples are only 14-bit, the device uses the upper 14-bits of the 16-bit word DAC samples by default. The deserializer to DAC sample mapping changes depending on the JESD204B link configuration setting. The responsibilities of the deframer are as follows:

- Monitor the JESD204B link for running disparity errors (controls the SYNCOUT_B signal to reset the link or report errors).
- Control the JESD204B interrupt signal (can output on general-purpose interrupt pin) to signal baseband processor when certain JESD204B error conditions arise.
- Remove character replacement.
- Perform 8-bit/10-bit decoding.
- Map JESD204B lane data to DAC samples.

A lane crossbar provides the ability to reorder the lanes into the deframer input. A sample crossbar provides the ability to reorder the DAC samples at the output of the deframer. The lane and sample crossbars enable flexiblity on which physical lanes are used and what data is on each lane. Figure 20 to Figure 25 demonstrate the valid deframer configurations.

Table 38. Static JESD204B Parameters

JESD204B Parameter	AD9371/AD9375 Value	Description
S	1	Samples transmitted/single converter/frame cycle
N	16	Converter resolution
N′	16	Total number of bits per sample
CF	0	Number of control words/frame clock cycle/converter device
CS	0	Number of control bits/conversion sample
HD	0 or 1	High density mode (only M = 2, L = 4 uses HD = 1)
K	Variable, suggested: 32	Number of frames in 1 multiframe, (20 ≤ F × K ≤ 256), F × K must be a multiple of 4, K ≤ 32

Table 39. JESD204B Parameters Dependent on Number of Lanes and Number of DACs

Number of DACs (M)	Number of Lanes (L)	Number of Bytes in 1 Frame (F) (F = 2 × M/L)
2	1	4
2	2	2
2	4	1
4	1	8
4	2	4
4	4	2

Table 40. Measured Deserializer Equalizer Correction (Nominal 1.3 V, 25°C)

Equalizer (EQ) Setting	3 GHz Loss (dB)	6 GHz Loss (dB)	FR408HR Length (Inches)	FR4 Length (Inches)
0	6.5	14	20	12
1	11.5	21	30	20
2	18	31	46	32
3	21.5	38	56	40
4	22	39	60	43

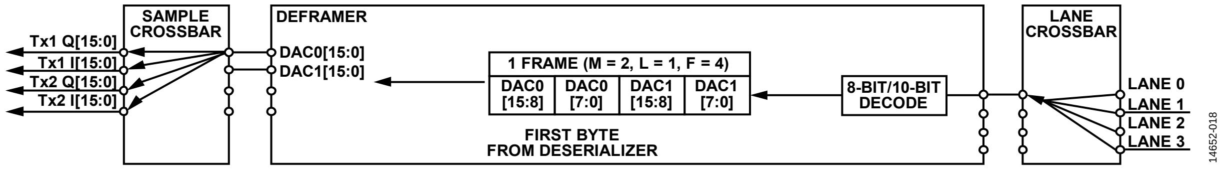


Figure 20. M2L1 Lane to DAC Byte Order

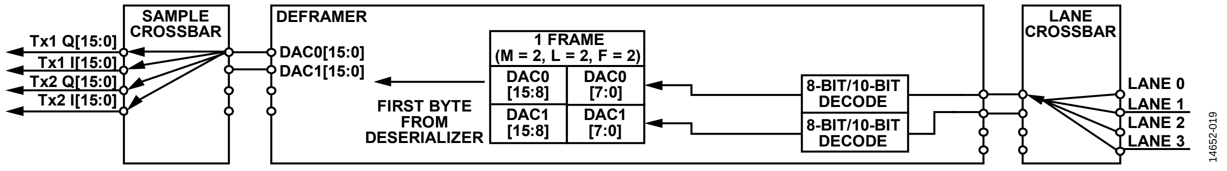


Figure 21. M2L2 Lane to DAC Byte Order

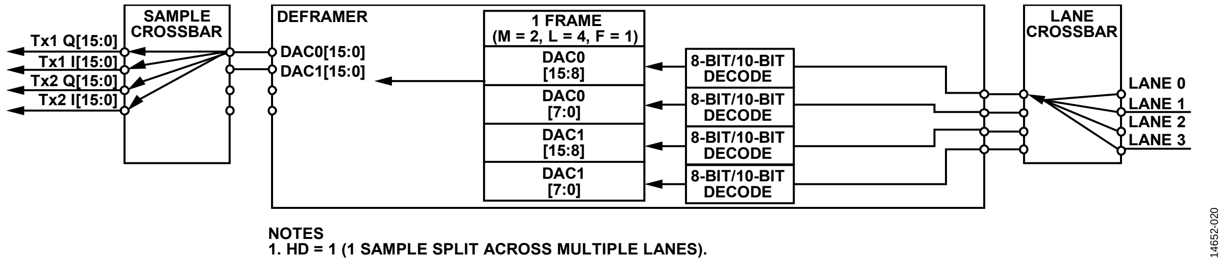


Figure 22. M2L4 Lane to DAC Byte Order

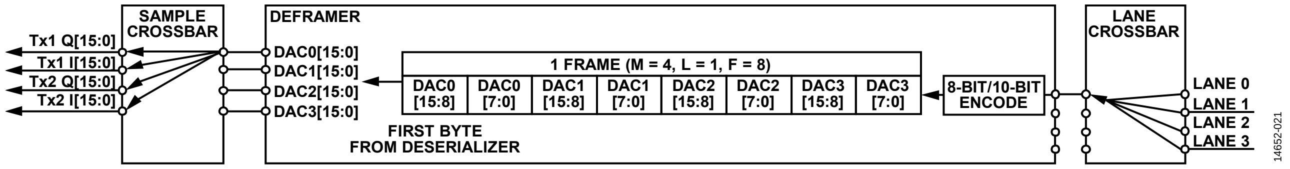


Figure 23. M4L1 Lane to DAC Byte Order

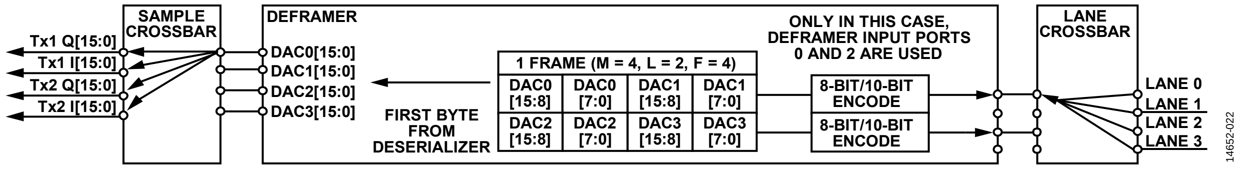


Figure 24. M4L2 Lane to DAC Byte Order

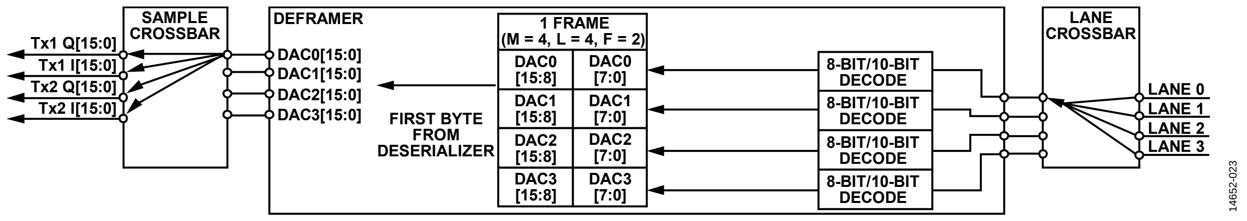


Figure 25. M4L4 Lane to DAC Byte Order

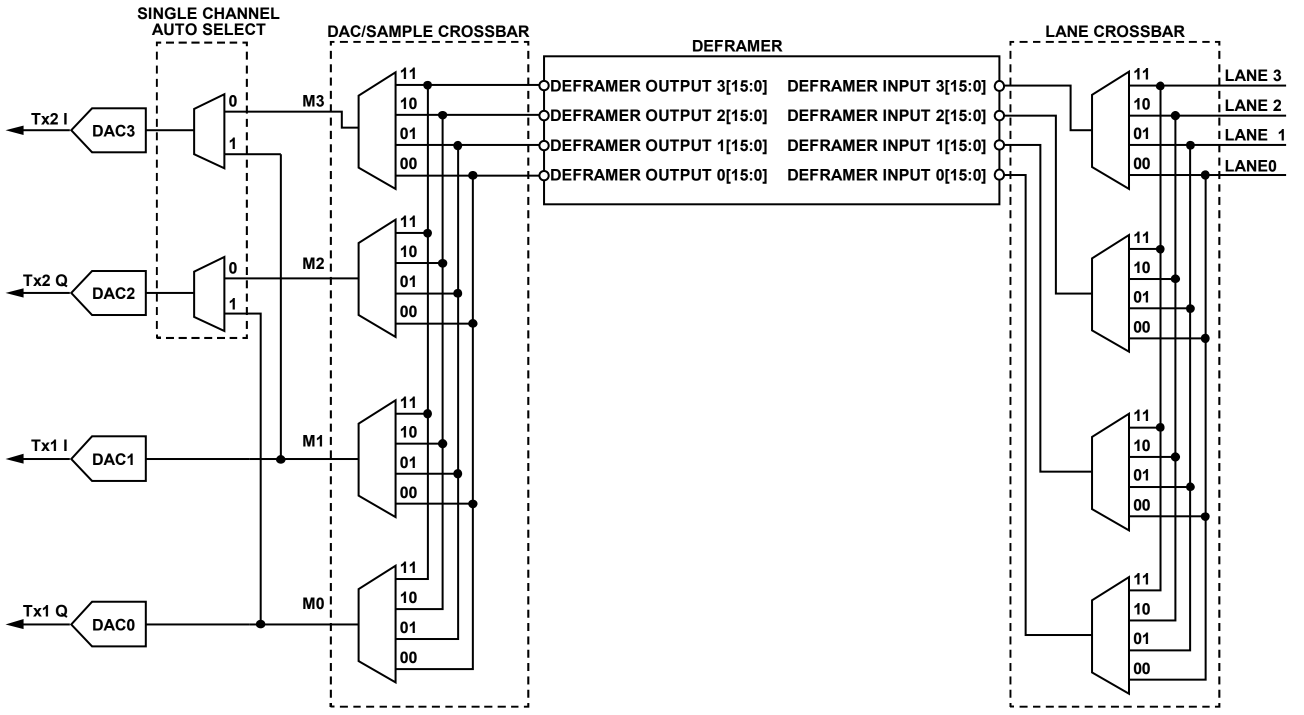


Figure 26. Deframer Lane Mux and Sample Mux Details

Other Useful Deframer IP Features

Deserializer PRBS

The deserializer has a built in pseudorandom bit sequence (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 deframer is unable to work properly. This mode can be used to check the robustness of the physical link during initial testing and/or factory test. For this mode to be fully utilized, the BPP must have a PRBS generator capable of creating PRBS7, PRBS15, or PRBS31 data.

A typical usage sequence is as follows:

1. Initialize the device as outlined in the Link Establishment section.
2. Enable the PRBS generator on the baseband processor with the same PRBS sequence required.
3. Call the application programming interface (API) MYKONOS_enableDeframerPrbsChecker(…) passing the actual device being evaluated, the PRBS sequence to check and enable bit set to 1.
4. After some amount of time, call the API function to check the PRBS errors. This can be done by calling the API function MYKONOS_readDeframerPrbsCounters(…) passing the actual device being evaluated, the counter selection lane to be read and the error count are 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 (MYKONOS_ clearDeframerPrbsCounters(…)).

API Software Integration

the MYKONOS_initialize(…) API function handles the configuration of the deserializer and Tx1/Tx2 deframer. Set any JESD204B link options in the mykonosDevice_t data structure before calling MYKONOS_initialize(…). After initialization, there are some other API functions to aid in debugging and monitoring the status of the JESD204B link.

JESD204B Deframer API Data Structures

mykonosJesd204bDeframerConfig_t

```
typedef struct
{
    uint8_t bankId;
    uint8_t deviceId;
    uint8_t lane0Id;
    uint8_t M;
    uint8_t K;
    uint8_t scramble;
    uint8_t externalSysref;
    uint8_t deserializerLanesEnabled;
    uint8_t deserializerLaneCrossbar;
    uint8_t EQSetting;
    uint8_t invertLanePolarity;
    uint8_t lmfcOffset;
    uint8_t newSysrefOnRelink;
    uint8_t enableAutoChanXbar;
} mykonosJesd204bDeframerConfig_t;
```

Table 41. Deframer Structure Member Description

Structure Member	Valid Values	Description
bankID	0 … 15	JESD204B configuration bank ID—extension to device ID.
deviceID	0 … 255	JESD204B configuration device ID—link identification number.
lane0ID	0 … 31	JESD204B configuration lane ID—if more than one lane is used, each subsequent lane increments from this number.
M	0, 2, 4	Number of DAC converters—2 converters per receive chain.
K	1 … 32	Number of frames in a multiframe—default is 32. F × K must be a multiple of 4.
scramble	0 … 255	Scrambling enabled. If scramble = 0, then scrambling is disabled. If scramble > 0, then scrambling is enabled.
externalSysref	0 … 255	External SYSREF enabled. If externalSysref = 0, then use internal SYSREF. If externalSysref > 0, then use external SYSREF.
deserializerLanesEnabled	0x0 … 0xF	Deserializer lane enabled—one bit per lane. If Bit 0 = 0, then Lane 0 is disabled; if Bit 0 = 1, then Lane 0 is enabled. If Bit 1 = 0, then Lane 1 is disabled; if Bit 1 = 1, then Lane 1 is enabled. If Bit 2 = 0, then Lane 2 is disabled; if Bit 2 = 1, then Lane 2 is enabled. If Bit 3 = 0, then Lane 3 is disabled; if Bit 3 = 1, then Lane 3 is enabled.
deserializerLaneCrossbar	0x0 … 0xFF	Deserializer lane crossbar—two bits per lane. Bits[1:0] identify the deserializer lane that connects to Deframer Lane 0. Bits[3:2] identify the deserializer lane that connects to Deframer Lane 1. Bits[5:4] identify the deserializer lane that connects to Deframer Lane 2. Bits[7:6] identify the deserializer lane that connects to Deframer Lane 3.
EQSetting	0 … 4	Equalizer setting, see Table 40 for details.
invertLanePolarity	0x0 … 0xF	Deserializer lane polarity inversion. If Bit 0 = 0, then Lane 0 is unaffected; if Bit 0 = 1, then Lane 0 is inverted. If Bit 1 = 0, then Lane 1 is unaffected; if Bit 1 = 1, then Lane 1 is inverted. If Bit 2 = 0, then Lane 2 is unaffected; if Bit 2 = 1, then Lane 2 is inverted. If Bit 3 = 0, then Lane 3 is unaffected; if Bit 3 = 1, then Lane 3 is inverted.
lmfcOffset	0 … 31	Local multiframe counter (LMFC) offset—local multiframe counter offset value for deterministic latency setting, set this such that 0 ≤ lmfcOffset ≤ (K − 1).
newSysrefOnRelink	0 … 255	New SYSREF on Relink—flag to indicate that a SYSREF is required to reestablish the link. If newSysrefOnRelink = 0, then no SYSREF is required. If newSysrefOnRelink > 0, then SYSREF is required.
enableAutoChanXbar	0 … 255	Enable automatic channel select for the ADC crossbar. If enableAutoChanXbar = 0, then the auto channel selection is disabled. If enableAutoChanXbar > 0, then the auto channel selection is enabled.

API Functions

MYKONOS_setupJesd204bDeframer(…)

```
mykonosErr_t
   MYKONOS_setupJesd204bDeframer(mykonosDevi
   ce_t *device);
```

This function is called directly from MYKONOS_Intialize(). It is not necessary to call this function if MYKONOS_Intialize() is used. This function sets up the JESD204B deframer.

DEPENDENCIES

The dependencies for the MYKONOS_ setupJesd204bDeframer(…) function are as follows:

- device → spiSettings
- device → spiSettings → chipSelectIndex
- device → tx → deframer → M
- device → tx → deframer → bankId
- device → tx → deframer → lane0Id
- device → tx → deframer → deserializerLanesEnabled
- device → tx → deframer → K
- device → tx → deframer → externalSysref
- device → tx → deframer → newSysrefOnRelink
- device → tx → deframer → enableAutoChanXbar
- device → tx → deframer → lmfcOffset
- device → tx → deframer → scramble

Table 42. MYKONOS_setupJesd204bDeframer(…) Parameters

Parameter	Description
device	Pointer to the device settings structure

Table 43. MYKONOS_setupJesd204bDeframer(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_DEFRAMER_INV_M_PARM	Invalid framer M (valid 1, 2, 4)
MYKONOS_ERR_DEFRAMER_INV_BANKID_PARM	Invalid bankId (valid 0 to 15)
MYKONOS_ERR_DEFRAMER_INV_LANEID_PARM	Invalid lane0Id (valid 0 to 31)
MYKONOS_ERR_DEFRAMER_INV_K_PARAM	Invalid K parameter in deframer structure (valid 1 to 32 with other constraints)
MYKONOS_ERR_DEFRAMER_INV_FK_PARAM	Invalid F × K value (F × K must be >20 and divisible by 4)
MYKONOS_ERR_DEFRAMER_INV_K_OFFSET_PARAM	Invalid K offset, must be less than K

MYKONOS_setupDeserializers(…)

```
mykonosErr_t
   MYKONOS_setupDeserializers(mykonosDevice_
   t *device);
```

This function is called directly from MYKONOS_Intialize(). It is not necessary to call this function if MYKONOS_Intialize() is used. This function sets up the JESD204B deserializers. This function enables the necessary deserializer lanes, sets the deserializer clocks polarity inversion settings, and determines equalizer settings based on the information found in the device data structure.

DEPENDENCIES

The dependencies for the MYKONOS_setupDeserializers(…) function are as follows:

- device → spiSettings
- device → spiSettings → chipSelectIndex
- device → tx → txProfile → clkPllVcoDiv
- device → tx → txProfile → vcoFreq_kHz
- device → tx → txProfile → txIqRate_kHz
- device → tx → deframer → M
- device → tx → deframer → deserializerLanesEnabled
- device → tx → deframer → invertLanePolarity
- device → tx → deframer → EQSetting

Table 44. MYKONOS_setupDeserializers(…) Parameters

Parameter	Description
device	Pointer to the device settings structure

Table 45. MYKONOS_setupDeserializers(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_INITDES_INV_TXPROFILE	Tx profile is not valid in data structure; cannot setup deserializer
MYKONOS_ERR_INITDES_INV_VCODIV_PARM	CLKPLL VCO divider is invalid
MYKONOS_ERR_DESER_INV_M_PARM	Invalid M (valid 2 or 4)
MYKONOS_ERR_DESER_INV_L_PARM	Invalid L (valid 1, 2, 4)
MYKONOS_ERR_DESER_INV_HSCLK_PARM	Invalid HSCLK, must be 6.144 G or less after CLKPLL VCO divider; verify CLKPLL configuration
MYKONOS_ERR_DESER_INV_LANERATE_PARM	Invalid lane rate, must be between 614.4 Mbps to 6144 Mbps
MYKONOS_ERR_DESER_INV_LANEEN_PARM	Invalid deserializerLanesEnabled (valid 0 to 15 in 1, 2, and 4 lane combinations)
MYKONOS_ERR_DESER_INV_EQ_PARM	Invalid equalizer parameter (valid 0 to 4)
MYKONOS_ERR_DESER_INV_LANEPN_PARM	Invalid pseudonoise (PN) invert setting, (valid 0 to 15, invert bit per lane)
MYKONOS_ERR_DES_HS_AND_LANE_RATE_NOT_INTEGER_MULT	Invalid clock settings, HSCLK is not an integer multiple of lane rate

MYKONOS_resetDeframer(…)

```
mykonosErr_t
   MYKONOS_resetDeframer(mykonosDevice_t
   *device);
```

It is important to reset the deframer after the baseband processor (BBP) begins outputting code group synchronization (CGS) characters on the JESD204B link. The lane FIFOs in the deframer path use the clock data recovery (CDR), recovered, clock. If the BBP ever resets the PLLs or clocking that drive the BBP serializer data, the recovered CDR clock in the device is lost. This can lead the lane FIFO to have underflow/overflow errors. This function resets the lane FIFOs and deframer IP. This also resets the SYSREF received internal signal. Therefore, send a SYSREF after this reset to properly retime the deframers local multiframe counter (LMFC).

DEPENDENCIES

The dependencies for the MYKONOS_resetDeframer(…) function are as follows:

- device → spiSettings

Table 46. MYKONOS_resetDeframer(…) Parameters

Parameter	Description
device	Pointer to the device settings structure

Table 47. MYKONOS_resetDeframer(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully

MYKONOS_enableSysrefToDeframer(…)

```
mykonosErr_t
   MYKONOS_enableSysrefToDeframer(mykonosDev
   ice_t *device, uint8_t enable);
```

This function can gate or allow the SYSREF at the outside of the device to reach the SYSREF input in the deframer IP in the device. Typically, the deframer ignores SYSREF (enable = 0) until after multichip sync. When the baseband processor (BBP) is ready to bring up the JESD204B link, call this function with enable = 1 to allow the deframer to retime its local multiframe counter (LMFC) to the SYSREF.

DEPENDENCIES

The dependencies for the MYKONOS_ enableSysrefToDeframer(…) function are as follows:

- device → spiSettings

Table 48. MYKONOS_enableSysrefToDeframer(…) Parameters

Parameter	Description
device	Pointer to the device settings structure
enable	1 enables SYSREF to deframer, 0 disables SYSREF to deframer

Table 49. MYKONOS_enableSysrefToDeframer(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully

MYKONOS_getDeframerFifoDepth(…)

```
mykonosErr_t
   MYKONOS_enableSysrefToDeframer(mykonosDev
   ice_t *device, uint8_t *fifoDepth,
   uint8_t *readEnLmfcCount);
```

This function reads the JESD204B deframer deterministic FIFO depth. To verify that the deterministic latency FIFO is not close to a underflow or overflow condition, it is recommended to check the FIFO depth. If the FIFO is close to an overflow or underflow condition, it is possible that, from power-up to power-up, deterministic latency may not be met. If an underflow or overflow occurs, the data may still be correct, but it might slip by one multiframe (losing deterministic latency). To correct an overflow/underflow, the baseband processor must add delay from SYSREF until the first symbol in a multiframe.

DEPENDENCIES

The dependencies for the MYKONOS_ getDeframerFifoDepth(…) function are as follows:

- device → spiSettings

Table 50. MYKONOS_getDeframerFifoDepth(…) Parameters

Parameter	Description
device	Pointer to the device settings structure
fifoDepth	Returns the depth of the deframer deterministic latency FIFO
readEnLmfcCount	Returns the local multiframe counter (LMFC) count value when the deterministic latency FIFO read enable was asserted; counts are at the internal deframer PCLK frequency

Table 51. MYKONOS_getDeframerFifoDepth(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_READ_DEFFIFODEPTH_NULL_PARAM	Function parameter fifoDepth is a null pointer
MYKONOS_ERR_READ_DEFFIFODEPTH_LMFCCOUNT_NULL_PARAM	Function parameter readEnLmfcCount is a null pointer

MYKONOS_readDeframerStatus(…)

```
mykonosErr_t
   MYKONOS_readDeframerStatus(mykonosDevice_
   t *device, uint8_t *deframerStatus);
```

After bringing up the deframer JESD204B link, the baseband processor (BBP) checks the status of the deframer.

DEPENDENCIES

The dependencies for the MYKONOS_readDeframerStatus(…) function are as follows:

- device → spiSettings

Table 52. MYKONOS_readDeframerStatus(…) Parameters

Parameter	Description
device	Pointer to the device settings structure
deframerStatus	Deframer status byte as described in Table 53

Table 53. deframerStatus Byte

deframerStatus	Bit Name	Description
7	Unused	Unused.
6	Deframer IRQ	This bit indicates that the IRQ interrupt is asserted.
5	Deframer SYSREF received	When this bit is set, it indicates that the SYSREF pulse is received by the deframer IP.
4	Deframer receiver error	This bit is set when pseudorandom bit sequence (PRBS) receives an error.
u	Valid checksum	This bit is set when the received initial lane assignment sequence (ILAS) checksum is valid.
2	EOF event	This bit captures the internal status of the framer end of frame (EOF) event. Value = 1 if framing error during ILAS.
1	EOMF event	This bit captures the internal status of the framer end of multiframe (EOMF) event. Value = 1 if framing error during ILAS.
0	FS lost	This bit captures the internal status of the framer frame symbol (FS) event. Value = 1 if framing error during ILAS or user data (invalid replacement characters).

Table 54. MYKONOS_readDeframerStatus(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_READ_DEFRAMERSTATUS_NULL_PARAMETER	Function parameter deframerStatus has a null pointer

MYKONOS_jesd204bIlasCheck(…)

```
mykonosErr_t
   MYKONOS_jesd204bIlasCheck(mykonosDevice_t
   *device, uint16_t *mismatch);
```

This function allows the baseband processor (BBP) to verify that the initial lane assignment sequence (ILAS) configuration sent by the BBP to the deframer matches the configuration programmed during MYKONOS_initialize(). The mismatch parameter is a bit field that alerts the BBP of exactly which field has a mismatch.

DEPENDENCIES

The dependencies for the MYKONOS_jesd204bIlasCheck(…) function are as follows:

- device → spiSettings

Table 55. MYKONOS_jesd204bIlasCheck(…) Parameters

Parameter	Description
device	Pointer to the device settings structure
mismatch	Bit encoded word to indicate mismatch as described in Table 56

Table 56. mismatch Parameter Descriptions

mismatch Bit	Description
15	Mismatch detected bit: Bits[0:14] are OR'ed together to set this bit.
14	JESD204B FCHK0: configuration checksum OK bit, where 0 = fail and 1 = pass.
13	JESD204B HD: high density bit, where 0 = samples are contained with single lane and 1 = samples are divided over more than one lane.
12	JESD204B CF: 0 = control bits appended to each sample, 1 = control bits appended to end of frame.
11	JESD204B S: number of samples per converter per frame.
10	JESD204B NP: JESD204B word size based on the highest data converter resolution.
9	JESD204B CS: number of control bits transferred per sample per frame.
8	JESD204B N: data converter sample resolution.
7	JESD204B M: number of data converters.
6	JESD204B K: frames per multiframe.
5	JESD204B F: octets per frame.
4	JESD204B SCR: scramble setting.
3	JESD204B L: lanes per data converter.
2	JESD204B LID0: lane ID.
1	JESD204B BID: bank ID.
0	JESD204B DID: device ID.

Table 57. MYKONOS_jesd204bIlasCheck(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_JESD204B_ILAS_MISMATCH_NULLPARAM	Function parameter mismatch has a null pointer

MYKONOS_enableDeframerPrbsChecker(…)

```
mykonosErr_t
   MYKONOS_enableDeframerPrbsChecker(mykonos
   Device_t *device, uint8_t lanes,
   mykonosPrbsOrder_t polyOrder, uint8_t
   enable);
```

This function enables the pseudorandom bit sequence (PRBS) checker in the deframer IP. When enabled, the JESD204B deframer is disabled and PRBS data is expected on the link instead of framed JESD204B data. The checker is a self synchronizing PRBS checker, as specified in the JESD204B specification. It is capable of checking PRBS7, PRBS15, and PRBS31 sequences.

The lanes parameter is a bit mask (bit per lane), allowing the checker to be enabled for a particular JESD204B deserializer lane. The polyOrder enumeration can be set to MYK_PRBS7, MYK_PRBS15, or MYK_PRBS31. The prbsOrder applies to all enable lanes. The enable parameter allows turning on and off the PRBS checker function.

DEPENDENCIES

The dependencies for the MYKONOS_ enableDeframerPrbsChecker(…) function are as follows:

- device → spiSettings

Table 58. MYKONOS_enableDeframerPrbsChecker(…) Parameters

Parameter	Description
device	Pointer to the device settings structure
lanes	Selects the lane for pseudorandom bit sequence (PRBS) checking based on a 4-bit mask, where each bit selects a different lane: 1 = Lane 0, 2 = Lane 1, 4 = Lane 2, 8 = Lane 3
polyOrder	Selects the PRBS type based on enumerator values: MYK_PRBS7, MYK_PRBS15, MYK_PRBS31
enable	1 enables checking, 0 disables checking

Table 59. MYKONOS_enableDeframerPrbsChecker(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_DEFRAMER_INV_PRBS_POLYORDER_PARAM	Invalid polyOrder parameter, use proper enumeration
MYKONOS_ERR_DEFRAMER_INV_PRBS_ENABLE_PARAM	Invalid enable (valid 0 to 1) or lanes (valid 0 to 15) parameter

MYKONOS_clearDeframerPrbsCounters(…)

```
mykonosErr_t
   MYKONOS_clearDeframerPrbsCounters(mykonos
   Device_t *device);
```

This function allows the baseband processor to clear the deframer pseudorandom bit sequence (PRBS) counters. It resets the PRBS error counters for all lanes. It is recommended to clear the error counters after enabling the deframer PRBS checker.

DEPENDENCIES

The dependencies for the MYKONOS_ clearDeframerPrbsCounters(…) function are as follows:

- device → spiSettings

Table 60. MYKONOS_clearDeframerPrbsCounters(…) Parameters

Parameter	Description
device	Pointer to the device settings structure

Table 61. MYKONOS_clearDeframerPrbsCounters(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully

MYKONOS_readDeframerPrbsCounters(…)

```
mykonosErr_t
   MYKONOS_readDeframerPrbsCounters(mykonosD
   evice_t *device, uint8_t counterSelect,
   uint32_t *prbsErrorCount);
```

After enabling the deframer pseudorandom bit sequence (PRBS) checker and clearing the PRBS error counters, use this function to read back the PRBS error counters. The counterSelect parameter allows the baseband processor (BBP) 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 BBP calls this function four times. Note that the counterSelect parameter selects the deframer input, not the physical lane at the outside of the chip. The counterSelect value required to read a particular lane depends on the lane crossbar of the deframer.

The 24-bit PRBS error count is returned in the prbsErrorCount parameter.

DEPENDENCIES

The dependencies for the MYKONOS_ readDeframerPrbsCounters(…) function are as follows:

- device → spiSettings

Table 62. MYKONOS_readDeframerPrbsCounters(…) Parameters

Parameter	Description
device	Pointer to the device settings structure
counterSelect	Selects the pseudorandom bit sequence (PRBS) error counter to be read based on values 0 to 3
prbsErrorCount	Number of errors detected in the PRBS pattern

Table 63. MYKONOS_readDeframerPrbsCounters(…) Return Values

Return Value	Description
MYKONOS_ERR_OK	Function completed successfully
MYKONOS_ERR_READ_DEFRAMERPRBS_NULL_PARAM	Function parameter prbsErrorCount has a null pointer
MYKONOS_ERR_DEFRAMER_INV_PRBS_CNTR_SEL_PARAM	Invalid counterSelect parameter (valid from 0 to 3)

LINK ESTABLISHMENT

After applying power to the device, the serializers, framers, deserializer, deframer, and the rest of the JESD204B circuits are powered down. Several steps are required to successfully power up the JESD204B link.

Suggested JESD204B API Initialization Sequence

The steps required to initialize the device JESD204B links (both ADCs and DACs datapaths) are as follows:

1. Initialize the mykonosDevice_t application programming interface (API) data structure and substructures with the desired settings. The transceiver evaluation software (TES) can output a **.c/.h** file with the data structures initialized to the values from the configuration tab.
2. Call the MYKONOS_initialize(…) API command to configure the device. This sets up the device to use the chosen Rx/Tx/ORx/sniffer profiles, program the clock PLL and digital clocks, set up the serializer, framer, deserializer, deframer, and so on, for the Rx/Tx/ORx/sniffer profiles that are valid.
3. Perform multichip synchronization. Send at least two initial SYSREF rising edges for multichip sync between multiple devices. For proper synchronization, the same SYSREF pulses must be seen at each device during the same device clock cycle. If only a single device is used, this step is still required to ensure JESD204B deterministic latency. For proper operation, it is recommended to disable the SYSREF, enable multichip synchronization, then reenable SYSREF. SYSREF may either be a single pulse or free running. If a periodic pulse is used, the phase must not change between rising edges. Single-pulse mode can be implemented by gating a free running SYSREF after the falling edge occurs, allowing a single pulse to output.

```
  //Disable SYSREF from clock device
if free running

  uint8_t mcsStatus = 0;
  mykonosErr_t mykError =
MYKONOS_ERR_OK; //default to no error

  //Enable MCS in the AD9371.
  mykError =
MYKONOS_enableMultichipSync(pthe
AD9371Device, 1, &mcsStatus);
  if (mykError != MYKONOS_ERR_OK)
  {
       //function threw error code
  }
  //Request at least 2 SYSREF pulses
from clock device
  Ad9528.requestSysref(true);
  Ad9528.requestSysref(true);

  //Disable MCS in the AD9371 and
readback MCS status
  mykError =
MYKONOS_enableMultichipSync(pthe
AD9371Device, 0, &mcsStatus);
  if (mykError != MYKONOS_ERR_OK)
  {
       //function threw error code
  }
```

5. Complete the normal sequence to load the ARM processor and to run initialization calibrations. The JESD204B link initialization is usually performed at the very end of initialization. The ARM loading and calibrations have no impact on the JESD204B link.
6. If the baseband processor (BBP) requires the DAC transmit datapath, instruct the BBP to run the required initialization for the baseband processor. Enable the JESD204B serializer in the BBP to the state in which it outputs code group synchronization (CGS) K characters.
7. Perform a reset to the deframer to clear any disparity bit errors previously detected. Also, if the SERDES PLL inside the FPGA resets, it can cause the lane FIFOs to overflow/underflow, requiring a deframer reset.

```
  MYKONOS_resetDeframer(pthe
  AD9371Device);
```

8. Enable the JESD204B IP blocks to accept a SYSREF signal for internal local multiframe counter (LMFC) timing reset. Only the calls to the desired framers/deframers are necessary. Send a third SYSREF pulse to the device and BBP to reset the JESD204B LMFC timing locally in each device to guarantee deterministic latency. The device does not reset its LMFC timing on any future SYSREF pulses unless the newSysrefOnRelink option is enabled in the framer/deframer data structures.

```
  MYKONOS_enableSysrefToRxFramer(pthe
AD9371Device, 1);
  MYKONOS_enableSysrefToObsRxFramer(pt
he AD9371Device, 1);
  MYKONOS_enableSysrefToDeframer(pthe
AD9371Device, 1);

  //Request a SYSREF pulse or several
from clock device
  Ad9528.requestSysref(true);
```

Framers

When the baseband processor (BBP) asserts the SYNCIN_B signal (low), the framer transmits the code group synchronization (CGS) K characters. The framer continues to transmit the CGS until the SYSREF is received. After the framer receives the SYSREF pulse, the framer resets the local multiframe counter (LMFC) counter and waits for the deassertion of SYNCIN_B. After the BBP deasserts the SYNCIN_B signal (high), the framer transmits the initial lane assignment sequence (ILAS) at the beginning of the subsequent LMFC boundary. After the ILAS sequence is complete, the framer begins transmitting the ADC data. Details on the initial lane synchronization can be found in Section 5.3.3.5 of the

JEDEC Standard No. 204B. Refer to Figure 35 in the JEDEC Standard No 204B for details on the ILAS.

Deframer

Out of reset, the deframer asserts the SYNCOUT_B signal (low). With the SYNCOUT_B signal asserted, the baseband processor (BBP) transmits the code group synchronization (CGS) K characters. When the deframer receives the SYSREF pulse, the deframer resets the local multiframe counter (LMFC) counter. After the deframer synchronizes to the received data stream and properly decodes the K characters, it deasserts the SYNCOUT_B signal (high) on the subsequent LMFC boundary. The deassertion of the SYNCOUT_B signal prompts the BBP to begin the ILAS. If the link is established over multiple lanes, the ILAS may arrive at the device skewed in time across the lanes. The lane FIFOs buffer the data in each lane until the next LMFC boundary, at which time it releases all lanes aligned in time. After the lanes are aligned and the configuration data in the ILAS is verified, the user data following the ILAS is sent to the DACs. For more details, refer to Section 6.3 of the JEDEC Standard No. 204B. Refer to Figure 36 of the JEDEC Standard No.204B for details on establishing a link with deterministic latency.

Ensure that the scrambling setting matches in the BBP and the device. It is possible for the JESD204B link to successfully link and the data to appear corrupt because only the data is scrambled, not the ILAS. This can result in a transmit spectrum that looks like noise.

HARDWARE CONSIDERATIONS FOR SYNC SIGNALS

The device features pins digital input/outputs pins designated SYNCOUTB0+, SYNCOUTB0−, SYNCINB0+, SYNCINB0−, SYNCINB1+, and SYNCINB1−. In general, each SYNC pin allows the JESD204B receiver to communicate with the JESD204B transmitter. The SYNC pins can be configured as LVDS differential pairs or as a single ended CMOS pin. In LVDS mode for the SYNCINB0± and SYNCINB1±, a 100 Ω on-chip termination is enabled.

When configured in CMOS mode, only the positive polarity pin is used. If CMOS mode is used for the SYNCINB0 signal or the SYNCINB1 signal, the negative polarity pin should be connected to ground through a pull down resistor. If CMOS mode is used for the SYNCOUTB0 signal, the negative polarity pin should not be connected.

API Configuration

To configure the device SYNC pins for either CMOS or LVDS mode, there are three device data structure members to consider:

- To set LVDS or CMOS mode for ObsRx signal(s) SYNCINB1±, refer to the device data structure member located at: **device** > **obsRx** > **framer** > **rxSyncbMode**.
- To set LVDS or CMOS mode for Rx signals SYNCINB0±, refer to the device data structure member located at: **device** > **rx** > **framer** > **rxSyncbMode**.
- To set LVDS or CMOS mode for the Tx signals SYNCOUTB0±, refer to the device data structure member located at: **device** > **tx** > **deframer** > **txSyncbMode**.

For the parameters previously mentioned, if the txSyncbMode or rxSyncbMode parameter is set to 0, this corresponds to LVDS operation. If the parameter is set greater than 0, CMOS operation is enabled.

COMPATIBILITY WITH XILINX JESD204B FPGA IP

Analog Devices uses the Xilinx JESD204B IP bundled with the XC7Z045 FFG900 for demonstration with the provided Analog Devices evaluation platform.

Some versions of the Xilinx JESD204B IP include a watchdog timer that resets the high speed serial PLLs if SYNCOUT_B is held low for more than 10 ms. This feature causes the lane FIFOs in the deserializers to overflow/underflow because the lane FIFOs derive the write clock from the recovered clock data recovery (CDR) clock. When the field programmable gate array (FPGA) resets its SERDES PLLs, the CDR clock in the device unlocks and causes the lane FIFO to underflow/overflow. Typically, this is not a problem because SYNCOUT_B is not held low for longer than 10 ms in normal use. In debug mode, however, a user may choose to hold SYNCOUT_B low to test the link. It is recommended to disable the 10 ms watchdog reset in the Xilinx IP wrapper to prevent unnecessary issues caused by randomly resetting PLLs in the system.

MULTICHIP SYNCHRONIZATION

For multiple input, multiple output (MIMO) systems requiring more than two input or two output channels, multiple devices and a common reference oscillator are required. The device provides the capability to accept an external reference clock and synchronize operation with other devices. Each device includes its own baseband PLL that generates sampling and data clocks from the reference clock input, so an additional control mechanism is required to synchronize multiple devices. A set of logical pulses on the SYSREF_IN input is required to align the data clock on each device with a common reference. This user guide describes the hardware connections necessary to synchronize two devices. This user guide also provides detailed information about timing requirement for SYSREF pulse in reference to DEV_CLK signal clock. Figure 27 shows the hardware connections required to perform this synchronization.

When working with multiple transceivers, or even a single transceiver that requires deterministic latency between the Tx and observation and or main Rx JESD204B datapath, multichip sync is necessary. The series of three (or more) SYSREF pulses must be synchronous to the device clock. Typically, this is an output from the same clock generation IC that generates the device clock, allowing the setup and hold times to be guaranteed. The device evaluation system hardware uses the AD9528 for clock and SYSREF generation and distribution.

The frequency of the SYSREF pulse train must be a submultiple of the JESD204B local multiframe counter (LMFC) rate. The first two pulses synchronize the digital circuits and the third and following pulses are passed along to the JESD204B interface. It is possible for more than three pulses to be required if their frequency occurs faster than the JESD204B interface PLL can lock and synchronize. For detailed recommendations regarding establishing deterministic latency on the JESD204B interface, refer to the Link Establishment section.

Figure 28 shows where each SYSREF pulse is applied inside the device. The pulses reset the chip in the following order:

1. Reset the device clock input scaler.
2. Reset the digital core clock generation dividers. These dividers generate the ADC/DAC and digital clocks.
3. Reset provides JESD204B lane alignment and deterministic lane synchronization.

Note that the multichip synchronization (MCS) function does not include RF synchronization. The ability to synchronize RF local oscillators (LOs) is not available in devices. The only alignment among multiple chips that is possible using this feature is digital timing alignment.

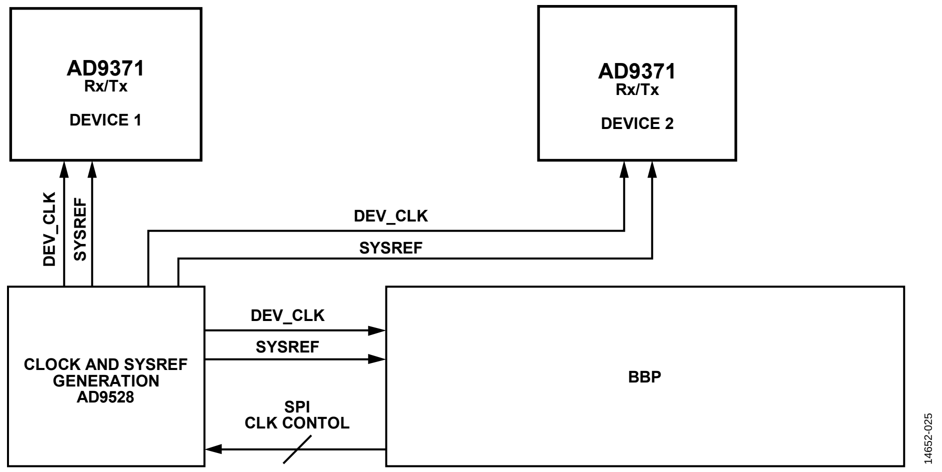


Figure 27. Multichip Connectivity—DEV_CLK Clock and SYSREF Signal Connections from the Clock Generation

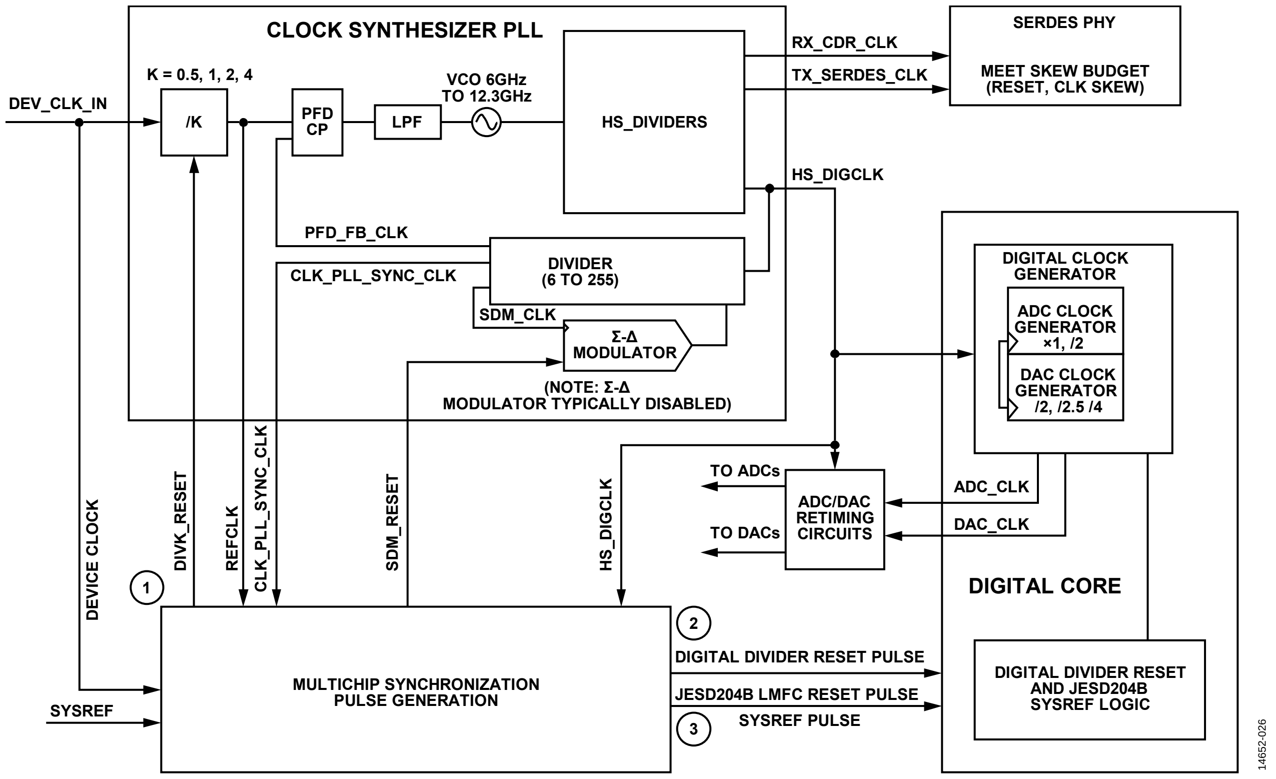


Figure 28. Clocking Architecture Indicating SYSREF Reset Sequence

MULTICHIP API FUNCTION DESCRIPTION

The application programming interface (API) package provided with the device contains the MYKONOS_enableMultichipSync function. This function sets up the device to route SYSREF pulses through the chip to reset the clock synthesizer, all digital clocks, and the JESD204B interface. Run this function after a single transceiver is initialized (or all transceivers, if more than one is used).

```
mykonosErr_t
   MYKONOS_enableMultichipSync(mykonosDevice
   _t *device, uint8_t enableMcs, uint8_t
   *mcsStatus)
```

The function parameters are as follows:

`enableMcs`

When set to = 1, enable the multichip synchronization (MCS) state machine. When set to = 0, allow reading back MCS status.

`mcsStatus`

If pointer is not null, then this parameter returns the MCS status word. Each bit of this status word represents the following functions:

- Bit 0, MCS JESD204B SYSREF status (1 = sync occurred). This bit indicates that the clock synthesizer input divider scaler has been reset.
- Bit 1, MCS digital clocks sync status (1 = sync occurred). This bit indicates that the clock synthesizer sigma delta modulator has been synchronized. This feature is disabled. The PLL operates in integer mode only for JESD204B support.
- Bit 2, MCS clock PLL Σ-Δ modulator sync status (1 = sync occurred). This bit indicates that the digital dividers after the clock synth are synchronized.
- Bit 3, MCS device clock divider sync status (1 = sync occurred). This bit indicates that MCS for the JESD204B framer/deframer required for deterministic latency has been performed.

After the SYSREF pulses are sent, call the MYKONOS_ enableMultichipSync() function again with the enableMcs parameter set to 0. When enableMcs = 0, the MCS status is returned in the multichip sync status parameters.

The typical sequence for multichip synchronization is as follows:

1. Initialize all devices in system using MYKONOS_initialize().
2. Run MYKONOS_enableMultichipSync with enableMcs = 1
3. Send at least three SYSREF pulses.
4. Run MYKONOS_enableMultichipSync with enableMcs = 0.
5. Load ARM, run ARM cals and continue to active transmit/ receive/.

An example sequence of multichip sync sequence copied from an IronPython script generated by the transceiver evaluation software (TES) is as follows:

```
 :
      Link.the AD9371.resetDevice()
      Link.the AD9371.initialize()
      pllStatus = Link.the
      AD9371.checkPllsLockStatus()
      mcsStatus = 0
      Link.the AD9371.enableMultichipSync(1,
      mcsStatus)
      Link.Ad9528.requestSysref(True)
      Link.Ad9528.requestSysref(True)
      Link.Ad9528.requestSysref(True)
      Link.the AD9371.enableMultichipSync(0,
      mcsStatus)
      :
```

The key to successful synchronization is to disable external SYSREF pulses before configuring the clock synth in all devices in the system (while the SYSREF is disabled, it must be held in a low state). With SYSREF still disabled, use MYKONOS_ enableMultichipSync as described previously. After three pulses are applied, all the devices sync at the same time with the SYSREF pulses. Exact details regarding SYSREF pulse timing relative to DEV_CLK are described in this user guide.

The device only synchronizes on the first three applied SYSREF pulses. More than three pulses may be supplied, but they do not have any further effect on synchronization (additional pulses are passed to the JESD204B interface). To resync, it is necessary to reset the device with a hard reset (applied to the $\overline{\text{RESET}}$ pin).

SYSTEM INITIALIZATION

This section provides information about the initialization process for the transceiver device using the application programming interface (API) developed by Analog Devices. The following sections describe the developer preparation requirements, initialization sequence, and example code for using the API software on any platform. This section does not explain the API library functions. Detailed information regarding the API functions can be found in the device API doxygen document, located in the **/src/doc** file in the software package directory structure

MODIFICATION OF COMMON.C FOR USER CODE INTEGRATION

The user is required to integrate their platform level drivers into common.c before using any application programming interface (API) function calls. Details regarding this are contained in the Software Integration section. The API used in the initialization process does not execute properly on the user hardware platform if this step is ignored.

DATA STRUCTURE MEMBER INITIALIZATION

The application programming interface (API) functions use sets of data structures to convey configuration and control data. These structures must be instantiated and their members loaded (initialized) with valid settings in the user code before the initialization sequence can take place. The transceiver evaluation software (TES) can generate valid data structure member values based on user settings. The TES generates the **myk_init.c** and **myk_init.h** files for direct porting to the user code or can be cut and pasted as required. The **myk_init.*** files contain the preloaded data structures and accompanying header file, respectively. The data structures that must be instantiated and loaded are contained in Table 64.

Note that hardware designs with multiple devices require each device to have its own unique configuration data initialized for all data structures; **headless.c** illustrates the structure initialization sequence at the beginning of the file. Explanations for each data structure are contained in the **mykonos.chm** document.

Table 64. Data Structures Requiring Initialization

Data Structure	Location	Description
spiSettings_t	/src/api/common.h	This contains the SPI settings for all system device types.
mykonosFir_t	/src/api/mykonos/t_mykonos.h	This contains the finite impulse response (FIR) filter gain, number of coefficients, and a pointer to a filter coefficient array.
mykonosJesd204bFramerConfig_t	/src/api/mykonos/t_mykonos.h	This contains the JESD204B framer configuration parameters.
mykonosJesd204bDeframerConfig_t	/src/api/mykonos/t_mykonos.h	This contains the JESD204B deframer configuration parameters.
mykonosRxProfile_t	/src/api/mykonos/t_mykonos.h	This contains the Rx profile information.
mykonosTxProfile_t	/src/api/mykonos/t_mykonos.h	This contains the Tx profile information
mykonosSnifferGainControl_t	/src/api/mykonos/t_mykonos.h	This contains the sniffer Rx manual gain control information.
mykonosORxGainControl_t	/src/api/mykonos/t_mykonos.h	This contains the observation Rx manual gain control information.
mykonosRxGainControl_t	/src/api/mykonos/t_mykonos.h	This contains the Rx manual gain control information.
mykonosAgcCfg_t	/src/api/mykonos/t_mykonos.h	This contains the automatic gain control (AGC) information.
mykonosTxSettings_t	/src/api/mykonos/t_mykonos.h	This contains the Tx setting information.
mykonosRxSettings_t	/src/api/mykonos/t_mykonos.h	This contains the Rx setting information.
mykonosObsRxSettings_t	/src/api/mykonos/t_mykonos.h	This contains the observation Rx setting information.
mykonosGpio3v3_t	/src/api/mykonos/t_mykonos.h	This contains the 3.3 V dc GPIO setting information.
mykonosGpio1v8_t	/src/api/mykonos/t_mykonos.h	This contains the 1.8 V dc GPIO setting information.
mykonosAuxIo_t	/src/api/mykonos/t_mykonos.h	This contains the auxiliary ADC, DAC, and pointers to the GPIO setting information.
mykonosDigClocks_t	/src/api/mykonos/t_mykonos.h	This contains the digital clock parameters.
mykonosDevice_t	/src/api/mykonos/t_mykonos.h	This data structure is inclusive of all previous data types, which are instantiated as pointers. The profilesValid bit field identifies which profile is valid. This data type is used to instantiate one device for configuration and control after member structure initialization.
mykonosArmGpioConfig_t	/src/api/mykonos/t_mykonos.h	Data structure to hold ARM GPIO pin assignments for each ARM input/output pin.
mykonosPeakDetAgcCfg_t	/src/api/mykonos/t_mykonos.h	Data structure to hold peak detector settings for the AGC.
mykonosPowerMeasAgcCfg_t	/src/api/mykonos/t_mykonos.h	Data structure to hold power measurement settings for the AGC.

Data Structure	Location	Description
mykonosInitCalStatus_t	/src/api/mykonos/t_mykonos.h	Data structure used to read back the initialization calibration status.
mykonosTxLolStatus_t	/src/api/mykonos/t_mykonos.h	Data structure to hold Tx local oscillator leakage (LOL) status.
mykonosTxQecStatus_t	/src/api/mykonos/t_mykonos.h	Data structure to hold Tx quadrature error correction (QEC) status.
mykonosRxQecStatus_t	/src/api/mykonos/t_mykonos.h	Data structure to hold Rx QEC status.
mykonosOrxQecStatus_t	/src/api/mykonos/t_mykonos.h	Data structure to hold Orx QEC status.
mykonosGainComp_t	/src/api/mykonos/t_mykonos_gpio.h	Data structure to hold gain compensation settings for the main receive channels.
mykonosObsRxGainComp_t	/src/api/mykonos/t_mykonos_gpio.h	Data structure to hold gain compensation settings for the observation channel.
mykonosFloatPntFrmt_t	/src/api/mykonos/t_mykonos_gpio.h	Data structure to hold floating point formatter settings for the floating point number generation.
mykonosTempSensorConfig_t	/src/api/mykonos/t_mykonos_gpio.h	Data structure used to configure the on die temperature sensor.
mykonosTempSensorStatus_t	/src/api/mykonos/t_mykonos_gpio.h	Data structure used to store temperature sensor related values.

INITIALIZATION SEQUENCE

The initialization sequence is comprised of application programming interface (API) calls mixed 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 their code into the initialization sequence specific to the hardware platform requirements. These platform requirements include, but are not limited to, user clock device, user field programmable gate array (FPGA)\application specific IC (ASIC)\baseband processor (BBP) JESD204B interface, datapath control, and various system checks governed by the application. The source code contained in **headless.c** provides a basic initialization sequence with code comments to help guide the user with the insertion of their application specific code.

Sequence Order

The initialization sequence order follows these steps:

1. Instantiate all data structures and load their members required by the user application (**myk_init.c** contents).
2. Initialize and setup of all clocks (platform clock source and JESD204B SYSREF signals are set up).
3. Initialize hardware platform (hardware dependent devices such as FPGA/ASIC/BBP interfaces are initialized).
4. Reset the device (call MYKONOS_resetDevice for reset of transceiver device in preparation for initialization).
5. Initialize the device (call MYKONOS_initialize function for configuration of the device).
6. Check CLKPLL status for lock (call MYKONOS_checkPllLockStatus and perform check with user defined code).
7. Multichip synchronization (all the JESD204B lanes are synchronized together for deterministic latency requirements).
8. Initialize the ARM processor (call MYKONOS_initArm).
9. Load the ARM binary file (call MYKONOS_loadArmFromBinary with user defined binary array pointer).
10. Set the RF PLL frequencies (call MYKONOS_setRfPllFrequency for each channel used by the application).
11. Perform RF PLL lock check (call MYKONOS_checkPllLockStatus and perform check with user defined code).
12. Set GPIO functions with the desired configuration (check **headless.c** for API calls to be made).
13. Run the initialization calibrations (call MYKONOS_runInitCals and MYKONOS_waitInitCals with user defined code).
14. Enable the SYSREF for Rx and ORx deframer (call MYKONOS_enableSysrefTo… functions).
15. Send the SYSREF signal to bring up the JESD204B interface.
16. Check deframer and framer status (call MYKONOS_readDeframerStatus and MYKONOS_readRxFramerStatus).
17. User verifies sync and link status for hardware platform.
18. Enable tracking calibrations (call MYKONOS_enableTrackingCals).
19. Turn the radio on for all transmitters and receivers that were set up previously (call MYKONOS_radioOn).

EXAMPLE CODE

headless.c

The **headless.c** file contains the initialization sequence example code. The sequence of the code written in this file matches the aforementioned initialization order. The **headless.c** file works with **myk_init.c** and **myk_init.h**. These source code files are generated by the TES based on user settings, as previously mentioned. The comment header at the top of **headless.c** describes the default TES user settings used to generate **myk_init.c** and **myk_init.h**. The data structures and their generated values in these files are subject to change based on TES revision. The **headless.c** file was written with the intent to be used as a template by the user when developing their application. Specially formatted code comments are placed throughout file, such as **Action** (which are user needed actions) and **Info** to help the user properly identify where they can insert their application specific code.

headless.h

The accompanying header file for **headless.c** is **headless.h**. It contains no code and is provided as a convenience to the user.

Disclaimer

Users may not modify any code located in the **/src/api** folder other than changing the **common.c** code bodies for hardware driver insertion and gain table changes in **mykonos_user.c**. Analog Devices maintains the code in **/src/api/mykonos** and **/src/api/ad9528** as intellectual property and all changes are at their sole discretion. Analog Devices provides new releases to fix any code bugs in these folders. Verification of all code bugs is independent of any user code.

SOURCE CODE EXAMPLES

myk_init.h

```
/**
 * \file myk_init.h
 *
 * \brief Contains structure definitions for myk_init.c
 *
 * The top level structure mykonosDevice_t mykDevice uses keyword
 * extern to allow the application layer main() to have visibility
 * to these settings.
 */

#ifndef MYK_INIT_H_
#define MYK_INIT_H_

#ifdef __cplusplus
extern "C" {
#endif

extern mykonosDevice_t mykDevice;

#ifdef __cplusplus
}
#endif

#endif
```

myk_init.c

```
/**
 * \brief Contains init setting structure declarations for the _instance API
 *
 * The top level structure mykonosDevice_t mykDevice uses keyword
 * extern to allow the application layer main() to have visibility
 * to these settings.
 *
 * All data structures required for operation have been initialized with values which
reflect these settings:
 *
 * Device Clock:
 * 122.88MHz
 *
 * Profiles:
 * Rx 20MHz, IQrate 30.72MSPS, Dec5
 * Tx 20/100MHz, IQrate 122.88MSPS, Dec5
 * ORX 100MHz, IQrate 122.88MSPS, Dec5
 * SRx 20MHz, IQrate 30.72MSPS, Dec5
 *
 */

#include <stddef.h>
#include "t_mykonos.h"
#include "t_mykonos_gpio.h"
#include "myk_init.h "

static int16_t txFirCoefs[] = {-94,-26,282,177,-438,-368,756,732,-1170,-1337,1758,2479,-
2648,-5088,4064,16760,16759,4110,-4881,-2247,2888,1917,-1440,-1296,745,828,-358,-
474,164,298,-16,-94};

static mykonosFir_t txFir =
{
    6,              /* Filter gain in dB*/
    32,             /* Number of coefficients in the FIR filter*/
    &txFirCoefs[0]  /* A pointer to an array of filter coefficients*/
};

static int16_t rxFirCoefs[] = {-13,-53,-50,-20,88,197,231,80,-239,-576,-654,-
268,538,1359,1585,749,-1060,-3028,-3847,-2340,1835,7799,13660,17289,17289,13660,7799,1835,-
2340,-3847,-3028,-1060,749,1585,1359,538,-268,-654,-576,-239,80,231,197,88,-20,-50,-53,-
13};

static mykonosFir_t rxFir =
{
    -6,             /* Filter gain in dB*/
    48,             /* Number of coefficients in the FIR filter*/
    &rxFirCoefs[0]  /* A pointer to an array of filter coefficients*/
};

static int16_t obsrxFirCoefs[] = {-14,-19,44,41,-89,-95,175,178,-303,-317,499,527,-779,-
843,1184,1317,-1781,-2059,2760,3350,-4962,-7433,9822,32154,32154,9822,-7433,-
4962,3350,2760,-2059,-1781,1317,1184,-843,-779,527,499,-317,-303,178,175,-95,-89,41,44,-
19,-14};
static mykonosFir_t obsrxFir =
{
    -6,             /* Filter gain in dB*/
    48,             /* Number of coefficients in the FIR filter*/
    &obsrxFirCoefs[0]/* A pointer to an array of filter coefficients*/
};

static int16_t snifferFirCoefs[] = {-1,-5,-14,-23,-16,24,92,137,80,-120,-378,-471,-
174,507,1174,1183,98,-1771,-3216,-2641,942,7027,13533,17738,17738,13533,7027,942,-2641,-
3216,-1771,98,1183,1174,507,-174,-471,-378,-120,80,137,92,24,-16,-23,-14,-5,-1};
static mykonosFir_t snifferRxFir=
{
    -6,             /* Filter gain in dB*/
    48,             /* Number of coefficients in the FIR filter*/
    &snifferFirCoefs[0]/* A pointer to an array of filter coefficients*/
};

static mykonosJesd204bFramerConfig_t rxFramer =
{
    0,          /* JESD204B Configuration Bank ID -extension to Device ID (Valid 0..15)*/
    0,          /* JESD204B Configuration Device ID - link identification number. (Valid
0..255)*/
    0,          /* JESD204B Configuration starting Lane ID.  If more than one lane used,
each lane will increment from the Lane0 ID. (Valid 0..31)*/
    4,          /* number of ADCs (0, 2, or 4) - 2 ADCs per receive chain*/
    32,         /* number of frames in a multiframe (default=32), F*K must be a multiple of
4. (F=2*M/numberOfLanes)*/
    1,          /* scrambling off if framerScramble= 0, if framerScramble>0 scramble is
enabled.*/
    1,          /* 0=use internal SYSREF, 1= use external SYSREF*/
    0x00,       /* serializerLanesEnabled - bit per lane, [0] = Lane0 enabled, [1] = Lane1
enabled*/
    0xE4,       /* serializerLaneCrossbar*/
    22,         /* serializerAmplitude - default 22 (valid (0-31)*/
    4,          /* preEmphasis - < default 4 (valid 0 - 7)*/
    0,          /* invertLanePolarity - default 0 ([0] will invert lane [0], bit1 will
invert lane1)*/
    0,          /* lmfcOffset - LMFC_Offset offset value for deterministic latency
setting*/
    0,          /* Flag for determining if SYSREF on relink should be set. Where, if > 0 =
set, 0 = not set*/
```

```
    0,          /* Flag for determining if auto channel select for the xbar should be set.
Where, if > 0 = set, '0' = not set*/
    0,          /* Selects SYNCb input source. Where, 0 = use RXSYNCB for this framer, 1 =
use OBSRX_SYNCB for this framer*/
    0,          /* Flag for determining if CMOS mode for RX Sync signal is used. Where, if
> 0 = CMOS, '0' = LVDS*/
    0           /* Selects framer bit repeat or oversampling mode for lane rate matching.
Where, 0 = bitRepeat mode (changes effective lanerate), 1 = overSample (maintains same lane
rate between ObsRx framer and Rx framer and oversamples the ADC samples)*/
};

static mykonosJesd204bFramerConfig_t obsRxFramer =
{
    0,      /* JESD204B Configuration Bank ID -extension to Device ID (Valid 0..15)*/
    0,      /* JESD204B Configuration Device ID - link identification number. (Valid
0..255)*/
    0,      /* JESD204B Configuration starting Lane ID.  If more than one lane used, each
lane will increment from the Lane0 ID. (Valid 0..31)*/
    2,      /* number of ADCs (0, 2, or 4) - 2 ADCs per receive chain*/
    32,     /* number of frames in a multiframe (default=32), F*K must be a multiple of 4.
(F=2*M/numberOfLanes)*/
    1,      /* scrambling off if framerScramble= 0, if framerScramble>0 scramble is
enabled.*/
    1,      /* 0=use internal SYSREF, 1= use external SYSREF*/
    0x00,   /* serializerLanesEnabled - bit per lane, [0] = Lane0 enabled, [1] = Lane1
enabled*/
    0xE4,   /* Lane crossbar to map framer lane outputs to physical lanes*/
    22,     /* serializerAmplitude - default 22 (valid (0-31)*/
    4,      /* preEmphasis - < default 4 (valid 0 - 7)*/
    0,      /* invertLanePolarity - default 0 ([0] will invert lane [0], bit1 will invert
lane1)*/
    0,      /* lmfcOffset - LMFC_Offset offset value for deterministic latency setting*/
    0,      /* Flag for determining if SYSREF on relink should be set. Where, if > 0 = set,
0 = not set*/
    0,      /* Flag for determining if auto channel select for the xbar should be set.
Where, if > 0 = set, '0' = not set*/
    1,      /* Selects SYNCb input source. Where, 0 = use RXSYNCB for this framer, 1 = use
OBSRX_SYNCB for this framer*/
    0,      /* Flag for determining if CMOS mode for RX Sync signal is used. Where, if > 0
= CMOS, '0' = LVDS*/
    1       /* Selects framer bit repeat or oversampling mode for lane rate matching.
Where, 0 = bitRepeat mode (changes effective lanerate), 1 = overSample (maintains same lane
rate between ObsRx framer and Rx framer and oversamples the ADC samples)*/
};

static mykonosJesd204bDeframerConfig_t deframer =
{
    0,     /* bankId extension to Device ID (Valid 0..15)*/
    0,     /* deviceId  link identification number. (Valid 0..255)*/
    0,     /* lane0Id Lane0 ID. (Valid 0..31)*/
    4,     /* M  number of DACss (0, 2, or 4) - 2 DACs per transmit chain */
    32,    /* K  #frames in a multiframe (default=32), F*K=multiple of 4.
(F=2*M/numberOfLanes)*/
    1,     /* scramble  scrambling off if scramble= 0.*/
    1,     /* External SYSREF select. 0 = use internal SYSREF, 1 = external SYSREF*/
    0x00,  /* Deserializer lane select bit field. Where, [0] = Lane0 enabled, [1] = Lane1
enabled, etc */
    0xE4,  /* Lane crossbar to map physical lanes to deframer lane inputs [1:0] = Deframer
Input 0 Lane section, [3:2] = Deframer Input 1 lane select, etc */
    1,     /* Equalizer setting. Applied to all deserializer lanes. Range is 0..4*/
    0,     /* PN inversion per each lane.  bit[0] = 1 Invert PN of Lane 0, bit[1] = Invert
PN of Lane 1, etc).*/
```

```
    0,      /* LMFC_Offset offset value to adjust deterministic latency. Range is 0..31*/
    0,      /* Flag for determining if SYSREF on relink should be set. Where, if > 0 = set,
'0' = not set*/
    0,      /* Flag for determining if auto channel select for the xbar should be set.
Where, if > 0 = set, '0' = not set*/
    0       /* Flag for determining if CMOS mode for TX Sync signal is used. Where, if > 0 =
CMOS, '0' = LVDS*/
};

static mykonosRxGainControl_t rxGainControl =
{
    MGC,              /* Current Rx gain control mode setting*/
    255,              /* Rx1 Gain Index, can be used in different ways for manual and AGC
gain control*/
    255,              /* Rx2 Gain Index, can be used in different ways for manual and AGC
gain control*/
    255,              /* Max gain index for the currently loaded Rx1 Gain table*/
    195,              /* Min gain index for the currently loaded Rx1 Gain table*/
    255,              /* Max gain index for the currently loaded Rx2 Gain table*/
    195,              /* Min gain index for the currently loaded Rx2 Gain table*/
    0,                /* Stores Rx1 RSSI value read back from the AD9371*/
    0                 /* Stores Rx2 RSSI value read back from the AD9371*/
};

static mykonosORxGainControl_t orxGainControl =
{
    MGC,              /* Current ORx gain control mode setting*/
    255,              /* ORx1 Gain Index, can be used in different ways for manual and AGC
gain control*/
    255,              /* ORx2 Gain Index, can be used in different ways for manual and AGC
gain control*/
    255,              /* Max gain index for the currently loaded ORx Gain table*/
    237               /* Min gain index for the currently loaded ORx Gain table*/
};

static mykonosSnifferGainControl_t snifferGainControl =
{
    MGC,              /* Current Sniffer gain control mode setting*/
    255,              /* Current Sniffer gain index. Can be used differently for Manual Gain
control/AGC*/
    255,              /* Max gain index for the currently loaded Sniffer Gain table*/
    203               /* Min gain index for the currently loaded Sniffer Gain table*/
};

static mykonosPeakDetAgcCfg_t rxPeakAgc =
{
    0x1F,    /* apdHighThresh: */
    0x16,    /* apdLowThresh */
    0xB5,    /* hb2HighThresh */
    0x80,    /* hb2LowThresh */
    0x40,    /* hb2VeryLowThresh */
    0x06,    /* apdHighThreshExceededCnt */
    0x04,    /* apdLowThreshExceededCnt */
    0x06,    /* hb2HighThreshExceededCnt */
    0x04,    /* hb2LowThreshExceededCnt */
    0x04,    /* hb2VeryLowThreshExceededCnt */
    0x4,     /* apdHighGainStepAttack */
    0x2,     /* apdLowGainStepRecovery */
    0x4,     /* hb2HighGainStepAttack */
    0x2,     /* hb2LowGainStepRecovery */
    0x4,     /* hb2VeryLowGainStepRecovery */
    0x1,     /* apdFastAttack */
    0x1,     /* hb2FastAttack */
    0x1,     /* hb2OverloadDetectEnable */
    0x1,     /* hb2OverloadDurationCnt */
    0x1             /* hb2OverloadThreshCnt */
};

static mykonosPowerMeasAgcCfg_t rxPwrAgc =
{
    0x01,    /* pmdUpperHighThresh */
    0x03,    /* pmdUpperLowThresh */
    0x0C,    /* pmdLowerHighThresh */
    0x04,    /* pmdLowerLowThresh */
    0x4,     /* pmdUpperHighGainStepAttack */
    0x2,     /* pmdUpperLowGainStepAttack */
    0x2,     /* pmdLowerHighGainStepRecovery */
    0x4,     /* pmdLowerLowGainStepRecovery */
    0x08,    /* pmdMeasDuration */
    0x02     /* pmdMeasConfig */
};

static mykonosAgcCfg_t rxAgcConfig =
{
    255,     /* AGC peak wait time */
    195,     /* agcRx1MinGainIndex */
    255,     /* agcRx2MaxGainIndex */
    195,     /* agcRx2MinGainIndex: */
    255,     /* agcObsRxMaxGainIndex */
    203,     /* agcObsRxMinGainIndex */
    1,       /* agcObsRxSelect */
    1,       /* agcPeakThresholdMode */
    1,       /* agcLowThsPreventGainIncrease */
    30720,   /* agcGainUpdateCounter */
    3,/* agcSlowLoopSettlingDelay */
    2,/* agcPeakWaitTime */
    0,/* agcResetOnRxEnable */
    0,/* agcEnableSyncPulseForGainCounter */
    &rxPeakAgc,
    &rxPwrAgc
};

static mykonosPeakDetAgcCfg_t obsRxPeakAgc =
{
    0x1F,    /* apdHighThresh: */
    0x16,    /* apdLowThresh */
    0xB5,    /* hb2HighThresh */
    0x80,    /* hb2LowThresh */
    0x40,    /* hb2VeryLowThresh */
    0x06,    /* apdHighThreshExceededCnt */
    0x04,    /* apdLowThreshExceededCnt */
    0x06,    /* hb2HighThreshExceededCnt */
    0x04,    /* hb2LowThreshExceededCnt */
    0x04,    /* hb2VeryLowThreshExceededCnt */
    0x4,     /* apdHighGainStepAttack */
    0x2,     /* apdLowGainStepRecovery */
    0x4,     /* hb2HighGainStepAttack */
    0x2,     /* hb2LowGainStepRecovery */
    0x4,     /* hb2VeryLowGainStepRecovery */
    0x1,     /* apdFastAttack */
    0x1,     /* hb2FastAttack */
    0x1,     /* hb2OverloadDetectEnable */
    0x1,     /* hb2OverloadDurationCnt */
    0x1            /* hb2OverloadThreshCnt */
```

```
};

static mykonosPowerMeasAgcCfg_t obsRxPwrAgc =
{
    0x01,    /* pmdUpperHighThresh */
    0x03,    /* pmdUpperLowThresh */
    0x0C,    /* pmdLowerHighThresh */
    0x04,    /* pmdLowerLowThresh */
    0x4,     /* pmdUpperHighGainStepAttack */
    0x2,     /* pmdUpperLowGainStepAttack */
    0x2,     /* pmdLowerHighGainStepRecovery */
    0x4,     /* pmdLowerLowGainStepRecovery */
    0x08,    /* pmdMeasDuration */
    0x02     /* pmdMeasConfig */
};

static mykonosAgcCfg_t obsRxAgcConfig =
{
    255,     /* agcRx1MaxGainIndex */
    195,     /* agcRx1MinGainIndex */
    255,     /* agcRx2MaxGainIndex */
    195,     /* agcRx2MinGainIndex: */
    255,     /* agcObsRxMaxGainIndex */
    203,     /* agcObsRxMinGainIndex */
    1,       /* agcObsRxSelect */
    1,       /* agcPeakThresholdMode */
    1,       /* agcLowThsPreventGainIncrease */
    30720,   /* agcGainUpdateCounter */
    3,       /* agcSlowLoopSettlingDelay */
    2,       /* agcPeakWaitTime */
    0,       /* agcResetOnRxEnable */
    0,       /* agcEnableSyncPulseForGainCounter */
    &obsRxPeakAgc,
    &obsRxPwrAgc
};


static mykonosRxProfile_t rxProfile =
{/* Rx 20MHz, IQrate 30.72MSPS, Dec5 */
    1,              /* The divider used to generate the ADC clock*/
    &rxFir,         /* Pointer to Rx FIR filter structure*/
    4,              /* Rx FIR decimation (1,2,4)*/
    5,              /* Decimation of Dec5 or Dec4 filter (5,4)*/
    1,              /* If set, and DEC5 filter used, will use a higher rejection DEC5 FIR
filter (1=Enabled, 0=Disabled)*/
    2,              /* RX Half band 1 decimation (1 or 2)*/
    30720,          /* Rx IQ data rate in kHz*/
    20000000,       /* The Rx RF passband bandwidth for the profile*/
    20000,          /* Rx BBF 3dB corner in kHz*/
    NULL            /* pointer to custom ADC profile*/
};

static mykonosRxProfile_t orxProfile =
{/* ORX 100MHz, IQrate 122.88MSPS, Dec5 */
    1,              /* The divider used to generate the ADC clock*/
    &obsrxFir,      /* Pointer to Rx FIR filter structure or NULL*/
    2,              /* Rx FIR decimation (1,2,4)*/
    5,              /* Decimation of Dec5 or Dec4 filter (5,4)*/
    0,              /* If set, and DEC5 filter used, will use a higher rejection DEC5 FIR
filter (1=Enabled, 0=Disabled)*/
    1,              /* RX Half band 1 decimation (1 or 2)*/
    122880,         /* Rx IQ data rate in kHz*/
    100000000,      /* The Rx RF passband bandwidth for the profile*/
    100000,         /* Rx BBF 3dB corner in kHz*/
    NULL            /* Pointer to custom ADC profile*/
};

static mykonosRxProfile_t snifferProfile =
{ /* SRx 20MHz, IQrate 30.72MSPS, Dec5 */
    1,              /* The divider used to generate the ADC clock*/
    &snifferRxFir,  /* Pointer to Rx FIR filter structure or NULL*/
    4,              /* Rx FIR decimation (1,2,4)*/
    5,              /* Decimation of Dec5 or Dec4 filter (5,4)*/
    0,              /* If set, and DEC5 filter used, will use a higher rejection DEC5 FIR
filter (1=Enabled, 0=Disabled)*/
    2,              /* RX Half band 1 decimation (1 or 2)*/
    30720,          /* Rx IQ data rate in kHz*/
    20000000,       /* The Rx RF passband bandwidth for the profile*/
    100000,         /* Rx BBF 3dB corner in kHz*/
    NULL            /* pointer to custom ADC profile*/
};



static mykonosTxProfile_t txProfile =
{ /* Tx 20/100MHz, IQrate 122.88MSPS, Dec5 */
    DACDIV_2p5,     /* The divider used to generate the DAC clock*/
    &txFir,         /* Pointer to Tx FIR filter structure*/
    2,              /* The Tx digital FIR filter interpolation (1,2,4)*/
    2,              /* Tx Halfband1 filter interpolation (1,2)*/
    1,              /* Tx Halfband2 filter interpolation (1,2)*/
    1,              /* TxInputHbInterpolation (1,2)*/
    122880,         /* Tx IQ data rate in kHz*/
    20000000,       /* Primary Signal BW*/
    100000000,      /* The Tx RF passband bandwidth for the profile*/
    710539,         /* The DAC filter 3dB corner in kHz*/
    50000,          /* Tx BBF 3dB corner in kHz*/
    0               /* Enable DPD, only valid for AD9373*/
};

static mykonosDigClocks_t mykonosClocks =
{
    122880,         /* CLKPLL and device reference clock frequency in kHz*/
    9830400,        /* CLKPLL VCO frequency in kHz*/
    VCODIV_2,       /* CLKPLL VCO divider*/
    4               /* CLKPLL high speed clock divider*/
};

static mykonosRxSettings_t  rxSettings =
{
    &rxProfile,     /* Rx datapath profile, 3dB corner frequencies, and digital filter
enables*/
    &rxFramer,      /* Rx JESD204b framer configuration structure*/
    &rxGainControl, /* Rx Gain control settings structure*/
    &rxAgcConfig,   /* Rx AGC control settings structure*/
    3,              /* The desired Rx Channels to enable during initialization*/
    0,              /* Internal LO = 0, external LO*2 = 1*/
    2490000000U,    /* Rx PLL LO Frequency (internal or external LO)*/
    0               /* Flag to choose if complex baseband or real IF data are selected for
Rx and ObsRx paths. Where, if > 0 = real IF data, '0' = zero IF (IQ) data*/
};

static mykonosDpdConfig_t dpdConfig =
{
    5,              /* 1/2^(damping + 8) fraction of power `forgotten' per sample (default:
`1/8192' = 5, valid 0 to 15), 0 = infinite damping*/
    1,              /* number of weights to use for int8_cpx weights weights member of this
structure (default = 1)*/
    2,              /* DPD model version: one of four different generalized polynomial
models: 0 = same as R0 silicon, 1-3 are new and the best one depends on the PA (default:
2)*/
    1,              /* 1 = Update saved model whenever peak Tx digital RMS is within 1dB of
historical peak Tx RMS*/
    20,             /* Determines how much weight the loaded prior model has on DPD
modeling (Valid 0 - 32, default 20)*/
    0,              /* Default off = 0, 1=enables automatic outlier removal during DPD
modeling */
    512,            /* Number of samples to capture (default: 512, valid 64-32768)*/
    4096,           /* threshold for sample in AM-AM plot outside of 1:1 line to be thrown
out. (default: 50% = 8192/2, valid 8192 to 1)*/
    0,              /* 16th of an ORx sample (16=1sample), (default 0, valid -64 to 64)*/
    255,            /* Default 255 (-30dBFs=(20Log10(value/8192)), (valid range  1 to
8191)*/
    {{64,0},{0,0},{0,0}}/* DPD model error weighting (real/imag valid from -128 to 127)*/
};

static mykonosClgcConfig_t clgcConfig =
{
    -2000,          /* (value = 100 * dB (valid range -32768 to 32767) - total gain and
attenuation from the AD9371 Tx1 output to ORx1 input in (dB * 100)*/
    -2000,          /* (value = 100 * dB (valid range -32768 to 32767) - total gain and
attenuation from the AD9371 Tx2 output to ORx2 input in (dB * 100)*/
    0,              /* (valid range 0 - 40dB), no default, depends on PA, Protects PA by
making sure Tx1Atten is not reduced below the limit*/
    0,              /* (valid range 0 - 40dB), no default, depends on PA, Protects PA by
making sure Tx2Atten is not reduced below the limit*/
    75,             /* valid range 1-100, default 75*/
    75,             /* valid range 1-100, default 45*/
    0,              /* 0= allow CLGC to run, but Tx1Atten will not be updated. User can
still read back power measurements.  1=CLGC runs, and Tx1Atten automatically updated*/
    0,              /* 0= allow CLGC to run, but Tx2Atten will not be updated. User can
still read back power measurements.  1=CLGC runs, and Tx2Atten automatically updated*/
    0,              /* 16th of an ORx sample (16=1sample), (default 0, valid -64 to 64)*/
    255             /* Default 255 (-30dBFs=(20Log10(value/8192)), (valid range  1 to
8191)*/
};

static mykonosVswrConfig_t vswrConfig =
{
    0,              /* 16th of an ORx sample (16=1sample), (default 0, valid -64 to 64)*/
    255,            /* Default 255 (-30dBFs=(20Log10(value/8192)), (valid range  1 to
8191)*/
    0,              /* 3p3V GPIO pin to use to control VSWR switch for Tx1 (valid 0-11)
(output from the AD9371)*/
    1,              /* 3p3V GPIO pin to use to control VSWR switch for Tx2 (valid 0-11)
(output from the AD9371)*/
    0,              /* 3p3v GPIO pin polarity for forward path of Tx1, opposite used for
reflection path (1 = high level, 0 = low level)*/
    0,              /* 3p3v GPIO pin polarity for forward path of Tx2, opposite used for
reflection path (1 = high level, 0 = low level)*/
    1,              /* Delay for Tx1 after flipping the VSWR switch until measurement is
made. In ms resolution*/
    1               /* Delay for Tx2 after flipping the VSWR switch until measurement is
made. In ms resolution*/
};
```

```
static mykonosTxSettings_t txSettings =
{
    &txProfile,     /* Tx datapath profile, 3dB corner frequencies, and digital filter
enables*/
    &deframer,      /* the AD9371 JESD204b deframer config for the Tx data path*/
    TX1_TX2,        /* The desired Tx channels to enable during initialization*/
    0,              /* Internal LO=0, external LO*2 if =1*/
    2503000000U,    /* Tx PLL LO frequency (internal or external LO)*/
    TXATTEN_0P05_DB,/* Initial and current Tx1 Attenuation*/
    10000,          /* Initial and current Tx1 Attenuation mdB*/
    10000,          /* Initial and current Tx2 Attenuation mdB*/
    NULL,           /* DPD,CLGC,VSWR settings. Only valid for AD9373 device, set pointer to
NULL otherwise*/
    NULL,           /* CLGC Config Structure. Only valid for AD9373 device, set pointer to
NULL otherwise*/
    NULL            /* VSWR Config Structure. Only valid for AD9373 device, set pointer to
NULL otherwise*/
};

static mykonosObsRxSettings_t obsRxSettings =
{
    &orxProfile,    /* ORx datapath profile, 3dB corner frequencies, and digital filter
enables*/
    &orxGainControl,/* ObsRx gain control settings structure*/
    &obsRxAgcConfig,/* ORx AGC control settings structure*/
    &snifferProfile,/* Sniffer datapath profile, 3dB corner frequencies, and digital filter
enables*/
    &snifferGainControl,/* SnRx gain control settings structure*/
    &obsRxFramer,   /* ObsRx JESD204b framer configuration structure */
    (MYK_ORX1_ORX2 | MYK_SNRXA_B_C),/* obsRxChannel */
    OBSLO_TX_PLL,   /* (obsRxLoSource) The Obs Rx mixer can use the Tx Synth(TX_PLL) or
Sniffer Synth (SNIFFER_PLL) */
    2600000000U,    /* SnRx PLL LO frequency in Hz */
    0,              /* Flag to choose if complex baseband or real IF data are selected for
Rx and ObsRx paths. Where if > 0 = real IF data, '0' = complex data*/
    NULL,           /* Custom Loopback ADC profile to set the bandwidth of the ADC response
*/
    OBS_RXOFF       /* Default ObsRx channel to enter when radioOn called */
};

static mykonosArmGpioConfig_t armGpio =
{
    0, // useRx2EnablePin; /*!< 0= RX1_ENABLE controls RX1 and RX2, 1 = separate
RX1_ENABLE/RX2_ENABLE pins */
    0, // useTx2EnablePin; /*!< 0= TX1_ENABLE controls TX1 and TX2, 1 = separate
TX1_ENABLE/TX2_ENABLE pins */
    0, // txRxPinMode;     /*!< 0= ARM command mode, 1 = Pin mode to power up Tx/Rx chains
*/
    0, // orxPinMode;      /*!< 0= ARM command mode, 1 = Pin mode to power up ObsRx
receiver*/

    /*the AD9371 ARM input GPIO pins -- Only valid if orxPinMode = 1 */
    0, // orxTriggerPin; /*!< Select desired GPIO pin (valid 4-15) */
    0, // orxMode2Pin;   /*!< Select desired GPIO pin (valid 0-18) */
    0, // orxMode1Pin;   /*!< Select desired GPIO pin (valid 0-18) */
    0, // orxMode0Pin;   /*!< Select desired GPIO pin (valid 0-18) */

    /* the AD9371 ARM output GPIO pins  --  always available, even when pin mode not
enabled*/
    0, // rx1EnableAck;  /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    0, // rx2EnableAck;  /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    0, // tx1EnableAck;  /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    0,// tx2EnableAck;  /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    0,// orx1EnableAck;  /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    0,// orx2EnableAck;  /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    0,// srxEnableAck;  /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    0 // txObsSelect;   /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    /* When 2Tx are used with only 1 ORx input, this GPIO tells the BBP which Tx channel is
*/
    /* active for calibrations, so BBP can route correct RF Tx path into the single ORx
input*/
};

static mykonosGpio3v3_t gpio3v3 =
{
    0,                    /*!< Oe per pin, 1=output, 0 = input */
    GPIO3V3_BITBANG_MODE,/*!< Mode for GPIO3V3[3:0] */
    GPIO3V3_BITBANG_MODE,/*!< Mode for GPIO3V3[7:4] */
    GPIO3V3_BITBANG_MODE,/*!< Mode for GPIO3V3[11:8] */
};

static mykonosGpioLowVoltage_t gpio =
{
    0,/* Oe per pin, 1=output, 0 = input */
    GPIO_MONITOR_MODE,/* Mode for GPIO[3:0] */
    GPIO_MONITOR_MODE,/* Mode for GPIO[7:4] */
    GPIO_MONITOR_MODE,/* Mode for GPIO[11:8] */
    GPIO_MONITOR_MODE,/* Mode for GPIO[15:12] */
    GPIO_MONITOR_MODE,/* Mode for GPIO[18:16] */
};

static mykonosAuxIo_t mykonosAuxIo =
{
    0,//auxDacEnableMask uint16_t
    {0,0,0,0,0,0,0,0,0,0},       //AuxDacValue uint16[10]
    {0,0,0,0,0,0,0,0,0,0},       //AuxDacSlope uint8[10]
    {0,0,0,0,0,0,0,0,0,0},       //AuxDacVref uint8[10]
    &gpio3v3,      //pointer to gpio3v3 struct
    &gpio,   //pointer to gpio1v8 struct
    &armGpio
};

static spiSettings_t mykSpiSettings =
{
    1, /* chip select index - valid 1~8 */
    0, /* the level of the write bit of a SPI write instruction word, value is inverted for
SPI read operation */
    1, /* 1 = 16-bit instruction word, 0 = 8-bit instruction word */
    1, /* 1 = MSBFirst, 0 = LSBFirst */
    0, /* clock phase, sets which clock edge the data updates (valid 0 or 1) */
    0, /* clock polarity 0 = clock starts low, 1 = clock starts high */
    0, /* Not implemented in ADIs platform layer. SW feature to improve SPI throughput */
    1, /* Not implemented in ADIs platform layer. For SPI Streaming, set address increment
direction. 1= next addr = addr+1, 0:addr=addr-1 */
    1  /* 1: Use 4-wire SPI, 0: 3-wire SPI (SDIO pin is bidirectional). NOTE: ADI's FPGA
platform always uses 4-wire mode */
};

static mykonosTempSensorConfig_t tempSensor =
{
    7,              /* 3-bit value that controls the AuxADC decimation factor when used for
temp sensor calculations; AuxADC_decimation = 256 * 2^tempDecimation*/
    67,             /* 8-bit offset that gets added to temp sensor code internally*/
    1,              /* this bit overrides the factory-calibrated fuse offset and uses the 
value stored in the offset member*/
    15,             /* 4-bit code with a resolution of 1°C/LSB, each time a temperature 
measurement is performed, the device compares the current temperature against the previous 
value.*/
};

static mykonosTempSensorStatus_t tempStatus =
{
    0,              /* 16-bit signed temperature value (in deg C) that is read back*/
    0,              /* If the absolute value of the difference is greater than the value in 
temperature configuration tempWindow, the windowExceeded flag is set.*/
    0,              /* when windowExceeded member gets set, this bit is set to 1 if current 
value is greater than previous value, else reset*/
    0,              /* when the reading is complete and a valid temperature value stored in 
tempCode.*/
};

mykonosDevice_t mykDevice =
{
    &mykSpiSettings,    /* SPI settings data structure pointer */
    &rxSettings,        /* Rx settings data structure pointer */
    &txSettings,        /* Tx settings data structure pointer */
    &obsRxSettings,     /* ObsRx settings data structure pointer */
    &mykonosAuxIo,          /* Auxiliary IO settings data structure pointer */
    &mykonosClocks,     /* Holds settings for CLKPLL and reference clock */
    0                   /* the AD9371 initialize function uses this as an output to 
remember which profile data structure pointers are valid */
};
```

headless.h

```
/**
 * \file headless.h
 *
 * \brief Contains definitions for headless.c
 */

#ifndef HEADLESS_H_
#define HEADLESS_H_

#ifdef __cplusplus
extern "C" {
#endif


#ifdef __cplusplus
}
#endif

#endif /* HEADLESS_H_ */
```

headless.c

```
/**
 * \file headless.c
 *
 * \brief Contains example code for user integration with their application
 *
 * All data structures required for operation have been initialized with values which 
reflect
 * these settings:
 *
 * Device Clock:
 * 122.88MHz
 *
 * Profiles:
 * Rx 20MHz, IQrate 30.72MSPS, Dec5
 * Tx 20/100MHz, IQrate 122.88MSPS, Dec5
 * ORX 100MHz, IQrate 122.88MSPS, Dec5
 * SRx 20MHz, IQrate 30.72MSPS, Dec5
 *
 * \def Action User needed action
 * \def Info information section
 */

#include <stdlib.h>
#include "headless.h"
#include "mykonos.h"
#include "mykonos_gpio.h"
#include "myk_init.h"
/****< Action: Insert rest of required Includes Here >***/

int main()
{
    const char* errorString;
    uint8_t mcsStatus = 0;
    uint8_t pllLockStatus = 0;
    uint8_t binary[98304] = {0}; /*** < Action: binary should contain ARM binary file as
array  > ***/
    uint32_t count = sizeof(binary);
    uint8_t errorFlag = 0;
    uint8_t errorCode = 0;
    uint32_t initCalsCompleted = 0;
    uint16_t errorWord = 0;
    uint16_t statusWord = 0;
    uint8_t status = 0;
    mykonosInitCalStatus_t initCalStatus = {0};

    uint8_t deframerStatus = 0;
    uint8_t obsFramerStatus = 0;
    uint8_t framerStatus = 0;
    uint32_t initCalMask = TX_BB_FILTER | ADC_TUNER | TIA_3DB_CORNER | DC_OFFSET |
TX_ATTENUATION_DELAY | RX_GAIN_DELAY
            | FLASH_CAL | PATH_DELAY | TX_LO_LEAKAGE_INTERNAL | TX_QEC_INIT |
LOOPBACK_RX_LO_DELAY
            | LOOPBACK_RX_RX_QEC_INIT | RX_LO_DELAY | RX_QEC_INIT;

    uint32_t trackingCalMask = TRACK_RX1_QEC | TRACK_RX2_QEC | TRACK_TX1_QEC |
TRACK_TX2_QEC | TRACK_ORX1_QEC
            | TRACK_ORX2_QEC;

    mykonosErr_t mykError = MYKONOS_ERR_OK;
    mykonosGpioErr_t mykGpioErr = MYKONOS_ERR_GPIO_OK;

    /* Allocating memory for the errorString */
    errorString = (const char*)malloc(sizeof(char) * 200);

    /*** < Action: Insert System Clock(s) Initialization Code Here          > ***/

    /*** < Action: Insert BBP Initialization Code Here                     > ***/

    /*********************************************************************/
    /*****                the AD9371 Initialization Sequence              *****/
    /*********************************************************************/
    /*** < Action: Toggle RESET pin on the AD9371 device                  > ***/
    if ((mykError = MYKONOS_resetDevice(&mykDevice)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    if ((mykError = MYKONOS_initialize(&mykDevice)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    /*************************************************************************/
    /*****                the AD9371 CLKPLL Status Check                *****/
    /*************************************************************************/
    if ((mykError = MYKONOS_checkPllsLockStatus(&mykDevice, &pllLockStatus)) !=
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    if (pllLockStatus & 0x01)
    {
        /*** < User: code here for actions once CLKPLL locked  > ***/
    }
    else
    {
        /*** < User: code here here for actions since CLKPLL not locked
         * ensure lock before proceeding - > ***/
    }

    /*************************************************************************/
    /*****                the AD9371 Perform MultiChip Sync             *****/
    /*************************************************************************/
    if ((mykError = MYKONOS_enableMultichipSync(&mykDevice, 1, &mcsStatus)) !=
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    /*** < Action: minimum 3 SYSREF pulses from Clock Device has to be produced
     * for MulticChip Sync > ***/

    /************************************************************************/
    /*****                the AD9371 Verify MultiChip Sync             *****/
    /************************************************************************/
    if ((mykError = MYKONOS_enableMultichipSync(&mykDevice, 0, &mcsStatus)) !=
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }
    if ((mcsStatus & 0x0B) == 0x0B)
    {
        /*** < Info: MCS successful  > ***/
        /*** < Action: extra User code   > ***/
    }
    else
    {
        /*** < Info: MCS failed  > ***/
        /*** < Action: ensure MCS before proceeding  > ***/
    }

    /*************************************************************************/
    /*****                the AD9371 Load ARM file                       *****/
    /*************************************************************************/
    if (pllLockStatus & 0x01)
    {
        if ((mykError = MYKONOS_initArm(&mykDevice)) != MYKONOS_ERR_OK)
        {
            /*** < Info: errorString will contain log error string in order to debug
failure > ***/
            errorString = getthe AD9371ErrorMessage(mykError);
        }

        /*** < Action: User must load ARM binary byte array into variable binary[98304]
before calling next command > ***/
        if ((mykError = MYKONOS_loadArmFromBinary(&mykDevice, &binary[0], count)) !=
MYKONOS_ERR_OK)
        {
            /*** < Info: errorString will contain log error string in order to debug why
             *  ARM did not load properly - check binary and device settings  > ***/
            /*** < Action: User code > ***/
            errorString = getthe AD9371ErrorMessage(mykError);
        }

    }
    else
    {
        /*** < Action: check settings for proper CLKPLL lock > ***/
    }

    /*************************************************************************/
    /*****                the AD9371 Set RF PLL Frequencies              *****/
    /*************************************************************************/
    if ((mykError = MYKONOS_setRfPllFrequency(&mykDevice, RX_PLL, mykDevice.rx-
>rxPllLoFrequency_Hz)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    if ((mykError = MYKONOS_setRfPllFrequency(&mykDevice, TX_PLL, mykDevice.tx-
>txPllLoFrequency_Hz)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    if ((mykError = MYKONOS_setRfPllFrequency(&mykDevice, SNIFFER_PLL, mykDevice.obsRx-
>snifferPllLoFrequency_Hz))
            != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    /*** < Action: wait 200ms for PLLs to lock > ***/

    if ((mykError = MYKONOS_checkPllsLockStatus(&mykDevice, &pllLockStatus)) !=
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    if ((pllLockStatus & 0x0F) == 0x0F)
    {
        /*** < Info: All PLLs locked > ***/
    }
    else
    {
        /*** < Info: PLLs not locked  > ***/
        /*** < Action: Ensure lock before proceeding - User code here > ***/
    }

    /*************************************************************************/
    /*****                the AD9371 Set GPIOs                           *****/
    /*************************************************************************/
    if ((mykGpioErr = MYKONOS_setupGpio(&mykDevice)) != MYKONOS_ERR_GPIO_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getGpiothe AD9371ErrorMessage(mykGpioErr);
    }

    /*************************************************************************/
    /*****                the AD9371 Set manual gains values             *****/
    /*************************************************************************/
    if ((mykError = MYKONOS_setRx1ManualGain(&mykDevice, 255)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    if ((mykError = MYKONOS_setRx2ManualGain(&mykDevice, 255)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    if ((mykError = MYKONOS_setObsRxManualGain(&mykDevice, OBS_RX1_TXLO, 255)) !=
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }
    if ((mykError = MYKONOS_setObsRxManualGain(&mykDevice, OBS_RX2_TXLO, 255)) !=
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    if ((mykError = MYKONOS_setObsRxManualGain(&mykDevice, OBS_SNIFFER_A, 255)) !=
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    if ((mykError = MYKONOS_setObsRxManualGain(&mykDevice, OBS_SNIFFER_B, 255)) !=
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }
    if ((mykError = MYKONOS_setObsRxManualGain(&mykDevice, OBS_SNIFFER_C, 255)) !=
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    /*************************************************************************/
    /*****                 the AD9371 Initialize attenuations               *****/
    /*************************************************************************/
    if ((mykError = MYKONOS_setTx1Attenuation(&mykDevice, 0)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    if ((mykError = MYKONOS_setTx2Attenuation(&mykDevice, 0)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    /*************************************************************************/
    /*****            the AD9371 ARM Initialization Calibrations            *****/
    /*************************************************************************/

    if ((mykError = MYKONOS_runInitCals(&mykDevice, (initCalMask &
~TX_LO_LEAKAGE_EXTERNAL))) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    if ((mykError = MYKONOS_waitInitCals(&mykDevice, 60000, &errorFlag, &errorCode)) !=
MYKONOS_ERR_OK)
```

```
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    if ((errorFlag != 0) || (errorCode != 0))
    {
        if ((mykError = MYKONOS_getInitCalStatus(&mykDevice, &initCalStatus)) !=
MYKONOS_ERR_OK)
        {
            /*** < Info: errorString will contain log error string in order to debug
failure > ***/
            errorString = getthe AD9371ErrorMessage(mykError);
        }

        /*** < Info: abort init cals > ***/
        if ((mykError = MYKONOS_abortInitCals(&mykDevice, &initCalsCompleted)) !=
MYKONOS_ERR_OK)
        {
            /*** < Info: errorString will contain log error string in order to debug
failure > ***/
            errorString = getthe AD9371ErrorMessage(mykError);
        }
        if (initCalsCompleted)
        {
            /*** < Info: which calls had completed, per the mask > ***/
        }

        if ((mykError = MYKONOS_readArmCmdStatus(&mykDevice, &errorWord, &statusWord)) !=
MYKONOS_ERR_OK)
        {
            /*** < Info: errorString will contain log error string in order to debug
failure > ***/
            errorString = getthe AD9371ErrorMessage(mykError);
        }

        if ((mykError = MYKONOS_readArmCmdStatusByte(&mykDevice, 2, &status)) !=
MYKONOS_ERR_OK)
        {
            /*** < Info: errorString will contain log error string in order to debug why
failed > ***/
            errorString = getthe AD9371ErrorMessage(mykError);
        }
        if (status != 0)
        {
            /*** < Info: Arm Mailbox Status Error errorWord > ***/
            /*** < Info: Pending Flag per opcode statusWord, this follows the mask > ***/
        }
    }
    else
    {
        /*** < Info: Calibrations completed successfully  > ***/
    }

    /*************************************************************************/
    /*****  the AD9371 ARM Initialization External LOL Calibrations with PA *****/
    /*************************************************************************/
    /*** < Action: Please ensure PA is enabled operational at this time > ***/
    if (initCalMask & TX_LO_LEAKAGE_EXTERNAL)
    {
        if ((mykError = MYKONOS_runInitCals(&mykDevice, TX_LO_LEAKAGE_EXTERNAL)) !=
MYKONOS_ERR_OK)
        {
            /*** < Info: errorString will contain log error string in order to debug
failure > ***/
            errorString = getthe AD9371ErrorMessage(mykError);
        }
        if ((mykError = MYKONOS_waitInitCals(&mykDevice, 60000, &errorFlag, &errorCode)) !=
MYKONOS_ERR_OK)
        {
            /*** < Info: errorString will contain log error string in order to debug
failure > ***/
            errorString = getthe AD9371ErrorMessage(mykError);
        }
        if ((errorFlag != 0) || (errorCode != 0))
        {
            if ((mykError = MYKONOS_getInitCalStatus(&mykDevice, &initCalStatus)) !=
MYKONOS_ERR_OK)
            {
                /*** < Info: errorString will contain log error string in order to debug
failure > ***/
                errorString = getthe AD9371ErrorMessage(mykError);
            }

            /*** < Info: abort init cals > ***/
            if ((mykError = MYKONOS_abortInitCals(&mykDevice, &initCalsCompleted)) !=
MYKONOS_ERR_OK)
            {
                /*** < Info: errorString will contain log error string in order to debug
failure > ***/
                errorString = getthe AD9371ErrorMessage(mykError);
            }
            if (initCalsCompleted)
            {
                /*** < Info: which calls had completed, per the mask > ***/
            }

            if ((mykError = MYKONOS_readArmCmdStatus(&mykDevice, &errorWord, &statusWord))
!= MYKONOS_ERR_OK)
            {
                /*** < Info: errorString will contain log error string in order to debug
failure > ***/
                errorString = getthe AD9371ErrorMessage(mykError);
            }

            if ((mykError = MYKONOS_readArmCmdStatusByte(&mykDevice, 2, &status)) !=
MYKONOS_ERR_OK)
            {
                /*** < Info: errorString will contain log error string in order to debug
failure > ***/
                errorString = getthe AD9371ErrorMessage(mykError);
            }
            if (status != 0)
            {
                /*** < Info: Arm Mailbox Status Error errorWord > ***/
                /*** < Info: Pending Flag per opcode statusWord, this follows the mask >
***/
            }
        }
        else
        {
            /*** < Info: Calibrations completed successfully  > ***/
        }
    }
    /*************************************************************************/
    /*****              SYSTEM JESD bring up procedure                  *****/
    /*************************************************************************/
    /*** < Action: Make sure SYSREF is stopped/disabled > ***/
    /*** < Action: Make sure BBP JESD is reset and ready to recieve CGS chars> ***/

    if ((mykError = MYKONOS_enableSysrefToRxFramer(&mykDevice, 1)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }
    /*** < Info: the AD9371 is waiting for sysref in order to start
     * transmitting CGS from the RxFramer> ***/

    if ((mykError = MYKONOS_enableSysrefToObsRxFramer(&mykDevice, 1)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }
    /*** < Info: the AD9371 is waiting for sysref in order to start
     * transmitting CGS from the ObsRxFramer> ***/

    /*** < User: Make sure SYSREF is stopped/disabled > ***/
    if ((mykError = MYKONOS_enableSysrefToDeframer(&mykDevice, 0)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    if ((mykError = MYKONOS_resetDeframer(&mykDevice)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    /*** < User: make sure BBP JESD framer is actively transmitting CGS> ***/
    if ((mykError = MYKONOS_enableSysrefToDeframer(&mykDevice, 1)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    /*************************************************************************/
    /*****             Enable SYSREF to the AD9371 and BBP                  *****/
    /*************************************************************************/
    /*** < Action: Sends SYSREF Here > ***/

    /*** < Info: the AD9371 is actively transmitting CGS from the RxFramer> ***/

    /*** < Info: the AD9371 is actively transmitting CGS from the ObsRxFramer> ***/

    /*** < Action: Insert User: BBP JESD Sync Verification Code Here > ***/

    /*************************************************************************/
    /*****               Check the AD9371 Framer Status                     *****/
    /**********************************************************************/
    if ((mykError = MYKONOS_readRxFramerStatus(&mykDevice, &framerStatus)) != 
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure > 
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }
    if ((mykError = MYKONOS_readOrxFramerStatus(&mykDevice, &obsFramerStatus)) != 
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure > 
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    /**********************************************************************/
    /*****               Check the AD9371 Deframer Status                 *****/
    /**********************************************************************/
    if ((mykError = MYKONOS_readDeframerStatus(&mykDevice, &deframerStatus)) != 
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure > 
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    /*** < Action: When links have been verified, proceed > ***/

    /**********************************************************************/
    /*****           the AD9371 enable tracking calibrations            *****/
    /**********************************************************************/
    if ((mykError = MYKONOS_enableTrackingCals(&mykDevice, trackingCalMask)) != 
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug why 
enableTrackingCals failed > ***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    /*** < Info: Allow Rx1/2 QEC tracking and Tx1/2 QEC tracking to run when in the radioOn 
state
     *  Tx calibrations will only run if radioOn and the obsRx path is set to 
OBS_INTERNAL_CALS > ***/

    /*** < Info: Function to turn radio on, Enables transmitters and receivers
     * that were setup during MYKONOS_initialize() > ***/
    if ((mykError = MYKONOS_radioOn(&mykDevice)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure > 
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    /*** < Info: Allow TxQEC to run when User: is not actively using ORx receive path > 
***/
    if ((mykError = MYKONOS_setObsRxPathSource(&mykDevice, OBS_RXOFF)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure > 
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }
    if ((mykError = MYKONOS_setObsRxPathSource(&mykDevice, OBS_INTERNALCALS)) !=
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will contain log error string in order to debug failure >
***/
        errorString = getthe AD9371ErrorMessage(mykError);
    }

    return 0;
}
```

QUADRATURE ERROR CORRECTION, CALIBRATION, AND ARM CONFIGURATION

The device comes with a built in ARM processor. This ARM processor is tasked with performing some initial calibrations of the signal paths of the device, as well as maintaining quadrature error correction (QEC) and local oscillator (LO) leakage performance during device operation through tracking algorithms. This chapter outlines the application programming interface (API) functions used to load the ARM, perform the initial calibrations, and run tracking calibrations.

ARM STATE MACHINE OVERVIEW

See Figure 29 for the ARM state machine flowchart.

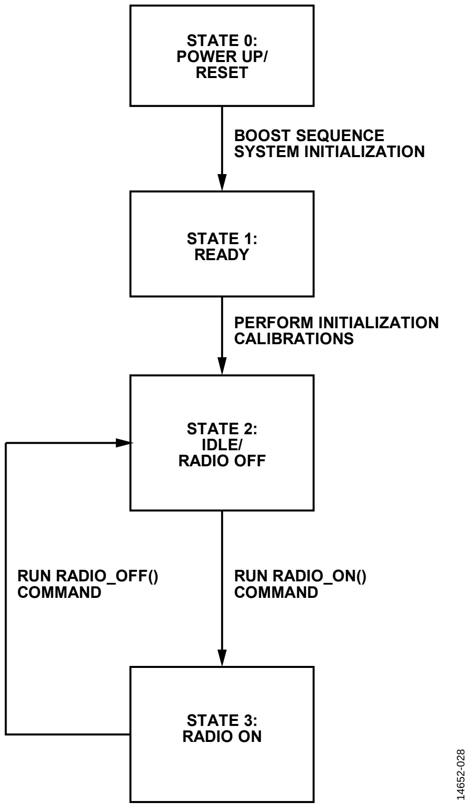


Figure 29. ARM State Machine Flowchart

State 0

When the ARM core is powered up, the ARM moves into its power-up/reset state. An image is to be loaded at this point. The process for loading that ARM is explained in the next section. After the ARM image is loaded, the ARM can be enabled and begins its boot sequence.

State 1

After the ARM has been successfully booted, it enters its ready state. In this state, it can receive configuration settings or commands (instructions), such as to perform the initial calibrations of the device. This is explained in the Initial ARM Calibrations section.

State 2

After the initial calibrations are performed, the ARM enters its idle state. In this state, it can receive configuration settings, such as which tracking calibrations are to be enabled. This is explained in the Tracking Calibrations section.

State 3

After the required tracking calibrations are enabled, a radio on command is provided to the ARM, which moves it into State 3. In this state, the ARM scheduler is active, and the ARM runs tracking calibrations when the necessary signal chains are available. The RF paths are also made available for use.

LOADING THE ARM

When the device is powered up or reset, it is necessary for the ARM image to be loaded to the device (this is towards the end of the initialization process, see an initialization script for further details). Prior to loading the ARM image, the ARM core is reset and prepared to receive its image with the following API function:

```
MYKONOS_initARM(mykonosDevice_t *device)
```

where *device is the structure pointer to the data structure.

After this function is run, the ARM image is then loaded with the following function:

```
MYKONOS_loadArmFromBinary(mykonosDevice_t
   *device, uint8_t *binary, uint32_t count)
```

where *binary is a pointer to the byte array containing ARM program memory bytes, and count is the number of bytes in this byte array.

The ARM image is provided though the **AD9371_M3.bin** file, provided in the Resources folder of the TES install.

After the ARM image is loaded, the MYKONOS_ loadArmFromBinary function enables the ARM, and the ARM automatically begins its boot sequence. As part of the boot sequence, the ARM calculates a checksum for the loaded image. The following application programming interface (API) function verifies the ARM load has been completed successfully:

```
MYKONOS_verifyArmChecksum(mykonosDevice_t
   *device)
```

This function ensures that the boot sequence completes, before reading back the calculated checksum from the relevant ARM memory location. It compares this to the precalculated checksum embedded in the ARM image. A successful load is verified when the checksums are equal.

INITIAL ARM CALIBRATIONS

The ARM processor in the device is tasked with scheduling/performing initial calibrations to optimize the performance of the signal paths prior to device operation. These calibrations are called by the following application programming interface (API) function:

```
MYKONOS_runInitCals(mykonosDevice_t  *device,
   uint32_t calMask)
```

where calMask is a 32-bit mask that informs the ARM processor which calibrations to run.

Table 65 shows the bit assignments of the calibration mask. The ARM processor runs the selected initial calibration for each enabled channel.

The calMask can thus be created using a bit map from Table 65, or by using the appropriate enums. For example, the following enable the ADC tuner and ADC flash calibration in a calMask to be passed to the MYKONOS_runInitCals( ) function:

```
unit32_t initCalMask = ADC_TUNER |
   FLASH_CAL;
```

The initial calibrations follow a specific order and must occur in a sequential manner. The ARM proceeds through these calibrations in the appropriate sequential order. It is important, however, that the user wait for these routines to complete prior to performing any further configuration of the device. The following API command is used to verify that these initial calibrations have been completed by the ARM:

```
MYKONOS_waitInitCals(mykonosDevice_t
   *device, uint32_t timeoutMs, uint8_t
   *errorFlag, uint8_t *errorCode)
```

where timeoutMs is the time in ms the function must wait for the calibrations to complete before returning an error, and *errorFlag and *errorCode indicate if an error has occurred, and if so, which calibration returned the error (see the Initial Calibration Errors section).

Table 65. Calibration Mask Bit Assignments[1]

Bits	Corresponding Enumerator	Function	Description
D0	TX_BB_FILTER	Tx baseband filter calibration	This tunes the corner frequency of the Tx baseband filter.
D1	ADC_TUNER	ADC tuner calibration	This configures the ADC for the required profile bandwidth.
D2	TIA_3DB_CORNER	Rx TIA filter calibration	This tunes the corner frequency of the Rx transimpedance amplifier (TIA) filter.
D3	DC_OFFSET	Rx dc offset calibration	This corrects for dc offset within the Rx chain.
D4	TX_ATTENUATION_DELAY	Tx attenuation delay	This is used to offset the onset of Tx analog and digital attenuations to compensate for the path delay between these blocks.
D5	RX_GAIN_DELAY	Rx gain delay	This offsets the onset of Rx analog attenuator and digital gain/attenuation block to compensate for the path delay between these blocks.
D6	FLASH_CAL	ADC flash calibration	This optimally configures the ADC flash.
D7	PATH_DELAY	Path delay calibration	This computes the Tx to loopback Rx path delay, which is required for the Tx quadrature error correction (QEC) and Tx local oscillator leakage (LOL) algorithms.
D8	TX_LO_LEAKAGE_INTERNAL	Tx LO leakage internal initial calibration	This performs an initial internal LO leakage calibration for the Tx path. It utilizes the Tx path, the internal loopback and ORx path (see Figure 34).
D9	TX_LO_LEAKAGE_EXTERNAL	Tx LO leakage external initial calibration	This performs an initial external LO leakage calibration for the Tx path. It utilizes the Tx path, a required external loopback path and the ORx path (see Figure 35).
D10	TX_QEC_INIT	Tx QEC initial calibration	This performs an initial QEC calibration for the Tx path. It utilizes the Tx path, the internal loopback path and the ORx path (see Figure 34).
D11	LOOPBACK_RX_LO_DELAY	Loopback ORx LO delay	This performs an LO delay calibration for the loopback receiver path.
D12	LOOPBACK_RX_RX_QEC_INIT	Loopback RxQEC initial calibration	This performs an initial QEC calibration for the Rx path.
D13	RX_LO_DELAY	Rx LO delay	This performs an LO delay calibration for the receiver path.
D14	RX_QEC_INIT	Rx QEC initial calibration	This performs an initial QEC calibration for the Rx path.
[D15:D31]	Not applicable	Not used	Not applicable

[1] There are requirements on a system level for these initialization calibrations to perform successfully. These requirements are described in the System Considerations for ARM Calibrations section.

TRACKING CALIBRATIONS

The ARM processor is tasked with ensuring that quadrature error correction (QEC) and local oscillator leakage (LOL) corrections are optimal throughout device operation, for example, over time, attenuation, and temperature. It achieves this by performing calibrations at regular intervals. These calibrations are termed tracking calibrations and utilize normal traffic data to update the path correction coefficients.

The following application programming interface (API) function enables the tracking calibrations in the ARM:

```
MYKONOS_enableTrackingCals(mykonosDevice_t
    *device, uint32_t enableMask)
```

where enableMask is a 32-bit mask that informs the ARM processor which calibrations to run.

Table 66 shows the bit assignments of the enable mask. There is also an equivalent function to read which tracking calibrations are enabled, which uses the same mask:

```
MYKONOS_getEnabledTrackingCals(mykonosDevice
    _t *device, uint32_t *enableMask)
```

This API function must be run in the radio off state, or before the device is operational. It cannot be run when the device is operational (ARM is in radio on state) because the ARM does not accept changes to the enabled tracking calibrations when the device is actively sending and receiving traffic data. The device can be returned to radio off, where traffic data is not being sent or received, if changes to the active tracking calibrations must be made. However, it is recommended that the respective tracking calibrations for the enabled channels be active at all times when the device is in the radio on state.

The ARM is tasked with the scheduling of the tracking calibrations. No user input is required to initiate a tracking calibration. The ARM schedules its calibrations based on the periodicity required for each calibration. Transmit tracking calibrations are run only at times when the user advises that the ORx path is available to the ARM for calibrations. The requirements of this are detailed in the Tracking Calibration Scheduler section, whereas the control of the ORx path to allow calibrations is covered in the System Control section.

The Rx/ORx channels also have dc correction tracking, which is active at all times. This calibration is not an ARM-based calibration.

Table 66. Tracking Calibrations Enable Mask Bit Assignments

enableMask Bit(s)	Function
D0	Rx1 QEC tracking
D1	Rx2 QEC tracking
D2	ORx1 QEC tracking
D3	ORx2 QEC tracking
D4	Tx1 LOL tracking
D5	Tx2 LOL tracking
D6	Tx1 QEC tracking
D7	Tx2 QEC tracking
[D8:D31]	Unused

TRACKING CALIBRATION SCHEDULER

The ARM is tasked with the scheduling of the tracking calibrations, scheduling its calibrations based on the periodicity required for each calibration. No user input is required to initiate a tracking calibration. Receive calibrations are only run when the receive chains are enabled; likewise, transmit tracking calibrations are only run when the transmit chains are enabled. Transmit tracking calibrations also require the user to assign the ORx path to the ARM for calibrations for a specified proportion of time, to allow the Tx data to be observed.

After the device is initialized, the ARM enters the idle/radio off state. When the ARM is in this state, the device is not transmitting/receiving data. For the device to transmit/receive data, the ARM must be in the radio on state with the tracking calibrations enabled. See Figure 29 for an overview of the ARM state machine.

Radio Off

The scheduler is not active in this state. The signal chains are powered down and the device is not receiving or transmitting data.

Radio On

The scheduler is active in this state, and tracking calibrations are run. The signal chains are available for use (see the System Control section).

The ARM is advised to move to the radio off/radio on states with the following application programming interface (API) functions:

- `MYKONOS_radioOn(mykonosDevice_t *device)`
- `MYKONOS_radioOff(mykonosDevice_t *device)`

When the ARM moves the state machine into its radio on state, it initiates its tracking calibration scheduler. It is therefore necessary that the required tracking calibrations be specified prior to calling the MYKONOS_radioOn() function.

It is possible to determine which state the ARM is in by using the following API function:

```
MYKONOS_getRadioState(mykonosDevice_t
    *device, uint32_t *radioStatus)
```

where *radioStatus indicates the current ARM state, as indicated in Table 67.

Table 67. Radio Status/ARM State

radioStatus	Function
0	Power-up/reset
1	Ready
2	Radio off
3	Radio on
>3	ARM error—check profile configuration

After the state machine is in the radio on state, the ARM scheduler performs the tracking calibrations on a periodic basis, ensuring that the correction values are optimal. For each tracking calibration enabled in the tracking calibration mask, a corresponding calibration task is initiated when the ARM is moved into the radio on state, as shown in Figure 30.

Each calibration task follows the same sequence of processes as shown in Figure 31. As is shown, each calibration task is responsible for indicating to the scheduler when it must run, through its own pending bit. This bit is set by the calibration task periodically when the calibration timer expires.

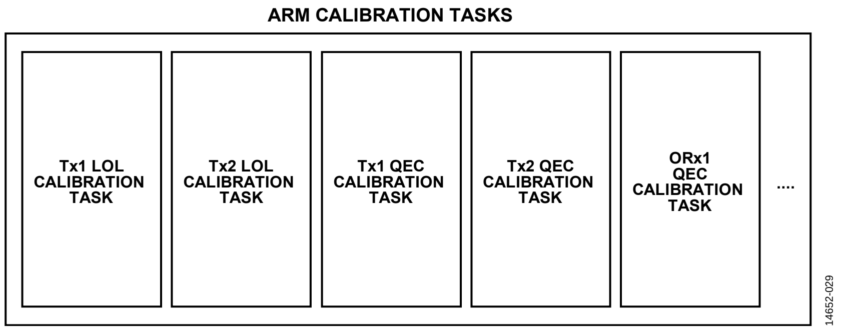


Figure 30. Calibration Tasks Run in the ARM Processor Based on the Tracking Calibration Mask Indicated by the User

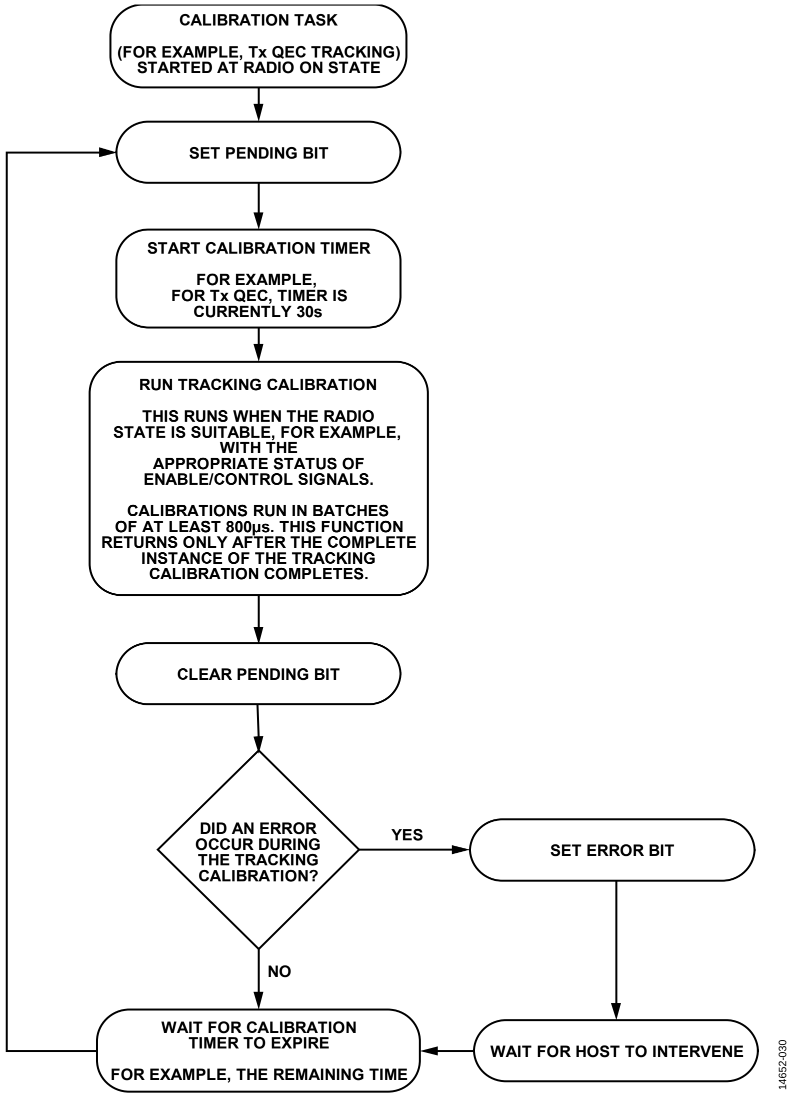

CALIBRATION TASK
(FOR EXAMPLE, Tx QEC TRACKING) STARTED AT RADIO ON STATE
SET PENDING BIT
START CALIBRATION TIMER
FOR EXAMPLE, FOR Tx QEC, TIMER IS CURRENTLY 30s
RUN TRACKING CALIBRATION
THIS RUNS WHEN THE RADIO STATE IS SUITABLE, FOR EXAMPLE, WITH THE APPROPRIATE STATUS OF ENABLE/CONTROL SIGNALS.
CALIBRATIONS RUN IN BATCHES OF AT LEAST 800µs. THIS FUNCTION RETURNS ONLY AFTER THE COMPLETE INSTANCE OF THE TRACKING CALIBRATION COMPLETES.
CLEAR PENDING BIT
DID AN ERROR OCCUR DURING THE TRACKING CALIBRATION?
YES
SET ERROR BIT
NO
WAIT FOR CALIBRATION TIMER TO EXPIRE
FOR EXAMPLE, THE REMAINING TIME
WAIT FOR HOST TO INTERVENE
14652-030

Figure 31. Flowchart of the Scheduling of a Single Process

Table 68. Possible Examples of Pending Bits for the Individual Tracking Calibrations

Pending Bits							
Tx1 LOL	**Tx2 LOL**	**Tx1 QEC**	**Tx2 QEC**	**ORx1 QEC**	**ORx2 QEC**	**Rx1 QEC**	**Rx2 QEC**
1	0	0	1	1	0	1	1

To read back the pending bits, use the following application programming interface (API) function:

```
MYKONOS_getPendingTrackingCals(mykonosDevice
    _t*device, uint32_t*pendingCalMask)
```

where pendingCalMask is the returned mask that advises if a calibration is pending or has returned an error, as indicated in Table 69.

Table 69. PendingCalMask Bits Descriptions

pendingCalMask Bit	Description
D0	Rx1 quadrature error correction (QEC) tracking pending bit
D1	Rx1 QEC tracking error bit
D2	Rx2 QEC tracking pending bit
D3	Rx2 QEC tracking error bit
D4	ORx1 QEC tracking pending bit
D5	ORx1 QEC tracking error bit
D6	ORx2 QEC tracking pending bit
D7	ORx2 QEC tracking error bit
D8	Tx1 local oscillator leakage (LOL) tracking pending bit
D9	Tx1 LOL tracking error bit
D10	Tx2 LOL tracking pending bit
D11	Tx2 LOL tracking error bit
D12	Tx1 QEC tracking pending bit
D13	Tx1 QEC tracking error bit
D14	Tx2 QEC tracking pending bit
D15	Tx2 QEC tracking error bit

The scheduler is then tasked with running each calibration when its corresponding pending bit is set. At any one time, however, more than one calibration task can be pending, as indicated by a possible example shown in Table 68, and it is the responsibility of the scheduler to determine which calibration must be run at any time.

The scheduler determines which calibration task to run at any time based on three conditions:

1. Pending bits. The scheduler reads the pending bits and determines which calibrations are requesting to run.
2. Priority. Each calibration task is given its own priority level. The calibration of the highest priority is given preference (highest priority being 1). The order of priority is shown in Table 70.

Table 70. Priority Levels of the Calibration Tasks

Priority	Calibration Task
1	Tx local oscillator leakage (LOL)
2	Tx quadrature error correction (QEC)
3	ORx QEC
4	Rx QEC

   Note there is no set priority between the individual channels calibrations (such as Tx1 LOL and Tx2 LOL). For calibration tasks of the same priority; the scheduler prioritizes the calibration task that completed first.
3. Availability of the required paths. The scheduler also determines if the calibration task can be performed. For example, as illustrated in Figure 32, the Tx QEC task requires the Tx to be enabled and the ORx to be assigned to ARM calibrations. If both conditions are not true, then the calibration cannot be run. The scheduler determines this, and, if the calibration cannot run, continues through its priority list to find a calibration which is pending and can be run (for example, Rx1 QEC may be run at this time). See the System Considerations for the Tracking Calibrations section for more details on the required paths for each tracking calibration.

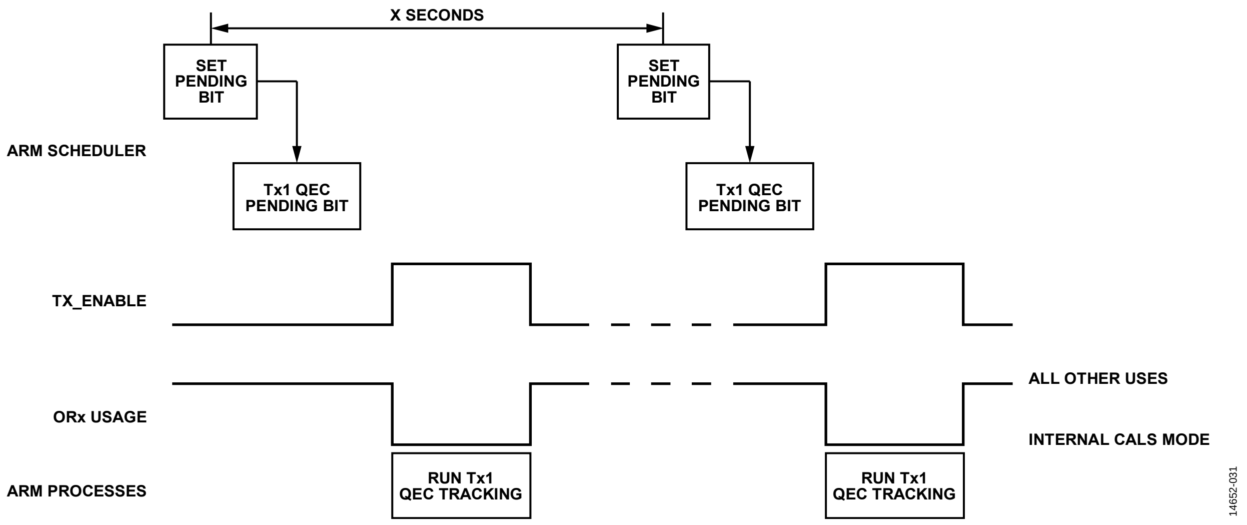


Figure 32. ARM Scheduler Operation

The scheduler runs the tracking calibrations in batches, as noted in the run tracking calibrations event (see Figure 31). Tx tracking calibrations typically require Tx data observation in the tens of milliseconds before the calibrations make an update to the correction parameters. It is recognized that the user may not provide sufficient time in a single run for the tracking calibration to complete; therefore, the scheduler performs calibrations in batches, where the Tx data can be observed in chunks of 800 µs. When sufficient batches of a tracking calibration run, the algorithm then computes its correction based on the observed data across all the batches. It is only after the correction parameters update that the pending bit is cleared, as shown in Figure 31.

This batch operation means that when a calibration is pending, and is selected by the scheduler to be run (based on the three conditions described previously), it initiates a batch to observe the Tx data for 800 µs. When this batch is complete, the scheduler again determines which calibrations can be run. If the same calibration cannot continue to run, for example, in time division duplex (TDD) mode, when the path to be calibrated may be no longer active, or if a higher priority calibration is pending, it awaits its next opportunity before it takes another batch of data.

Note that, if a tracking calibration batch is started but the observation is disrupted (for example, if the Tx/Rx path is disabled, or if the ORx path is reacquired by user for digital predistortion (DPD) captures) before 800 µs has completed, then the observation that has been made up to this point is discarded. This does not affect the algorithm; it waits for another batch, as normal. Therefore, when assigning the use of the ORx path for tracking calibrations (internal cals mode), it is recommended that the user do so in slots of at least 800 µs, or multiples thereof.

The final requirement is that the ORx path must be assigned to internal calibrations for 400 ms of transmit time every 2 sec. There are no further requirements on this; no exact period of tracking calibration batches that must be maintained. It is up to the user to determine the structure of the ORx path assignment to fit around their own ORx path requirements (such as DPD/ voltage standing wave ratio (VSWR), and so on.). It can be supplied in one full section, or in batches of 800 µs spread across the 2 sec in a nonperiodic fashion. The ARM never takes control of the assignment of the ORx path, and is reliant on the user to assign the ORx for calibrations. If the user fails to provide such instruction, then the calibrations never run.

In summary, for Tx calibrations to run successfully:

- The ORx must be assigned to internal calibrations by the user for 400 ms of transmit time every 2 sec.
- Assign this time in batches of at least 800 µs, or multiples thereof.

Note that the assignment of the ORx path for internal calibrations mode, as required to run Tx tracking calibrations, is discussed in the System Control section.

SYSTEM CONSIDERATIONS FOR ARM CALIBRATIONS

This section indicates what is necessary from a system perspective for the ARM to run its calibrations, for example, input/output path conditions for during initial calibrations and enable signal status for tracking calibrations. This section is split between initial and tracking calibration considerations.

System Considerations for the Initial Calibrations

The figures in this section show how the device is configured for notable calibrations with external system requirements, for example, the quadrature error correction (QEC) and local oscillator leakage (LOL) calibrations. In all diagrams, greyed out lines and blocks are not active in the calibration. Lines showing the path of the local oscilators (LOs) are shown to distinguish them from the signal paths. A brief explanation of the calibration is provided. As the ARM performs each of the calibrations, it is tasked with configuring the device as per the figures in this section, for example, enabling/disabling paths, and so on. No user input is required in this regard.

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

Rx QEC Initial Calibration

The Rx quadrature error correction (QEC) initialization calibration algorithm improves the Rx path QEC performance. The Rx QEC calibration functions by sweeping a number of internally generated test tones across the band, measuring quadrature performance, and calculating correction coefficients (see Figure 33).

The following is a system requirement:

- For optimum performance and lower calibration duration, run the Rx QEC initialization at attenuations between 0 dB and 5 dB. For optimal Rx path calibration performance, ensure a maximum signal power of –92 dBm/MHz is present at the Rx input. In addition, it is recommended that the Rx input be properly terminated while the calibration is running, as test tones are output from the receive port.

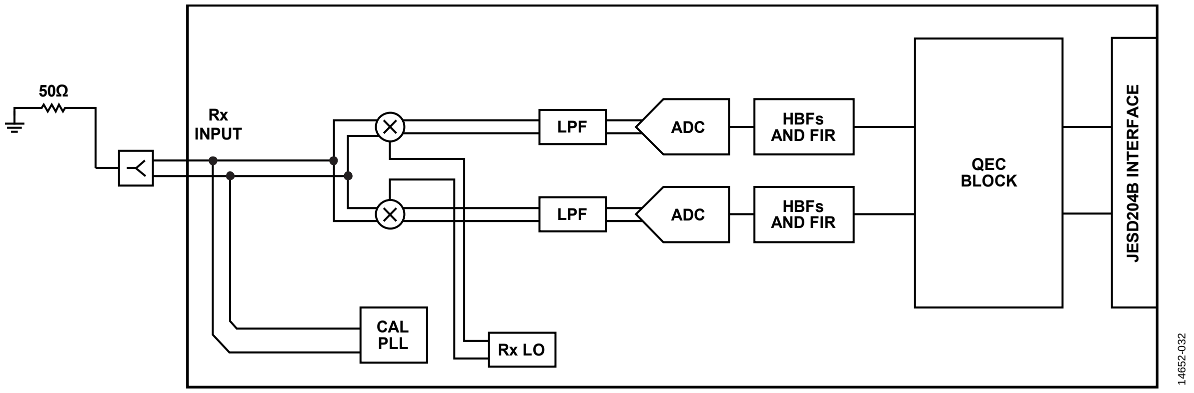


Figure 33. Rx QEC Initial Calibration System Configuration

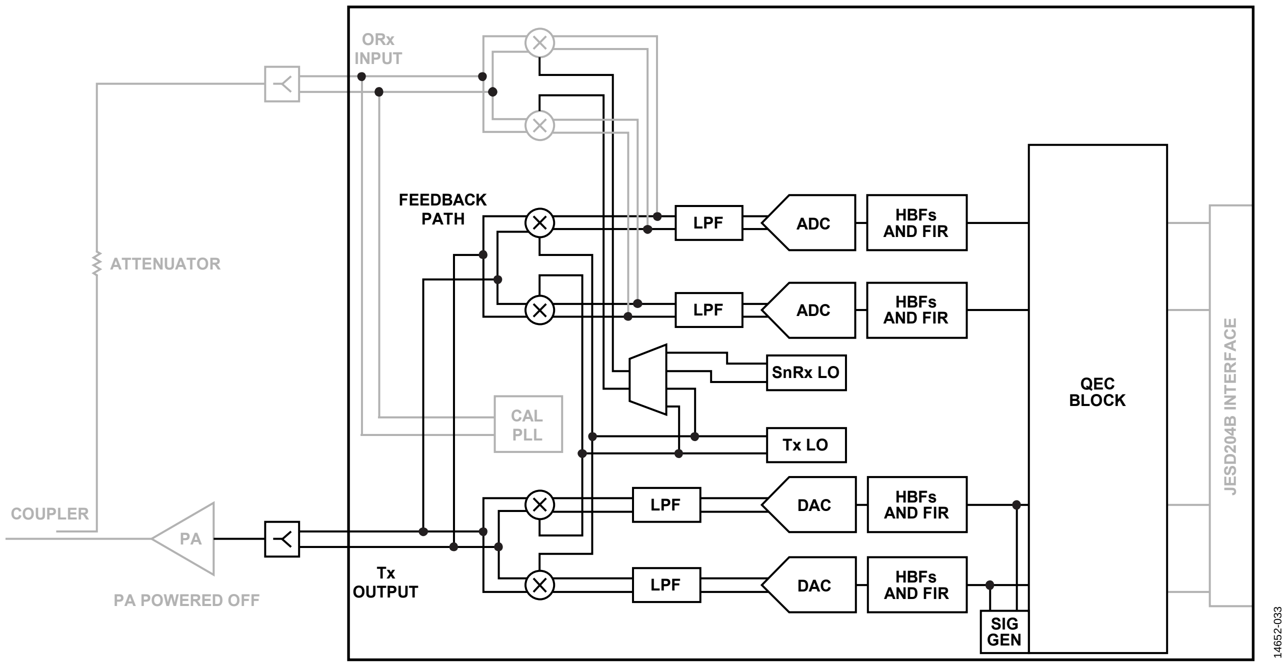


Figure 34. Device Path Configuration for the Tx LOL and QEC Initial Calibrations

Internal Tx LO Leakage and Tx QEC Initial Calibrations

The Tx internal local oscillator (LO) leakage and Tx quadrature error correction (QEC) initial calibrations use the internal loopback (feedback) path and the ORx baseband path to calculate initial correction factors. During these calibrations, test signals (tones and wideband signals) are output. These appear at the Tx output; therefore, it is important that the power amplifier at the output of the device be switched off. Both calibrations sweep through a series of attenuation values, creating a table of initial calibration values. Then, upon application of a Tx attenuation setting, the corresponding QEC and local oscillator leakage (LOL) correction values are applied to the Tx channel by the ARM. The device configuration for this calibration is shown in Figure 34.

The following is a system requirement:

- Power off the power amplifier in the Tx path during these calibrations.

External Tx LO Leakage Initial Calibration

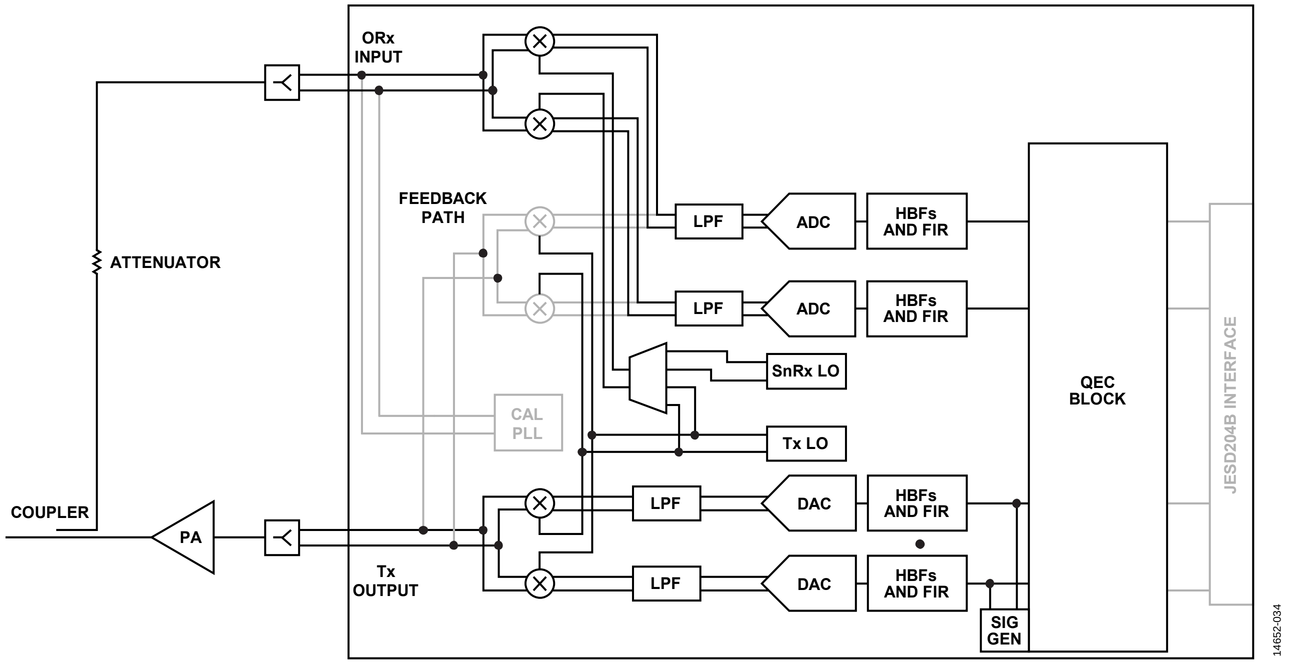


Figure 35. External LOL System Configuration (Greyed Out Circuitry Not Used)

The external local oscillator leakage (LOL) initialization calibration requires that the power amplifier be enabled such that a full external loop is made between the Tx outputs and the ORx inputs. The purpose of this calibration is to obtain a reasonable estimate of the external loop channel conditions (gain/phase) prior to operation. The device configuration is shown in Figure 35. The calibration uses a pseudorandom noise signal to estimate the channel conditions, which is a broadband signal with a nominal signal level of −78 dBFS out of the DAC.

It is important that a suitable attenuator be chosen between the power amplifier output and the ORx input. This is to prevent Tx data from saturating the ORx input. This is also be necessary from the perspective of digital predistortion (DPD) operation. The full-scale input of the ORx path is −13 dBm (with a 0 dB attenuation setting) for a single-tone input.

The system requires that the output of the Tx channel to be calibrated be routed to the utilized ORx path for the calibration signal to be observed. The device must be configured prior to the calibration to indicate which Tx is routed back to which ORx.

Note that the external Tx LOL initialization calibration makes certain assumptions in terms of which Tx is fed back to which ORx. The ARM bases this on the following parameters within the device data structure:

- For Tx channels, device → tx → txChannels.
- For ORx channels, device → obsRx → obsRxChannelsEnable.

When both Tx channels are used (txChannels = TX1_TX2) and both ORx channels are used (obsRxChannelsEnable = MYK_ORX1_ORX2), the ARM configures Tx1 to calibrate using ORx1, and Tx2 to calibrate using ORx2 (the user does not need to configure this). The ARM cycles through both Tx1 external LOL calibration, and then Tx2 external LOL calibration, so it is imperative that both feedback paths are enabled before the calibration is called.

Alternatively, if both Tx channels are used (txChannels = TX1_TX2); however, only the ORx1 channel is used (obsRxChannelsEnable = MYK_ORX1), then the ARM configures Tx1 to calibrate using ORx1, and Tx2 to also calibrate using ORx1, which also applies vice versa if ORx2 is selected. This approach is illustrated in Figure 36.

In this case, the calibration must advise the user which path it wishes to calibrate. It does this through the GPIO pin configured for the txObsSelect output. The user must configure the txObsSelect output before the external LOL initialization calibration is called (see the ARM GPIOs section). By default, the txObsSelect output indicates that the Tx1 output is to be fed back to the required ORx with a low output on this pin, while a high output indicates that Tx2 is to be fed back. Again, the initialization calibration cycles through both calibrations consecutively; therefore, it is important that both paths are active and that the request to toggle the external switch is responded to (the ARM expects to see the feedback path settled within 35 μs of the state change indicated by the txObsSelect output).

Note that this calibration does not provide good performance if an external LO is provided as the Tx LO. In such cases, LOL performance is reliant solely on the initialization calibration, and subsequently degrades.

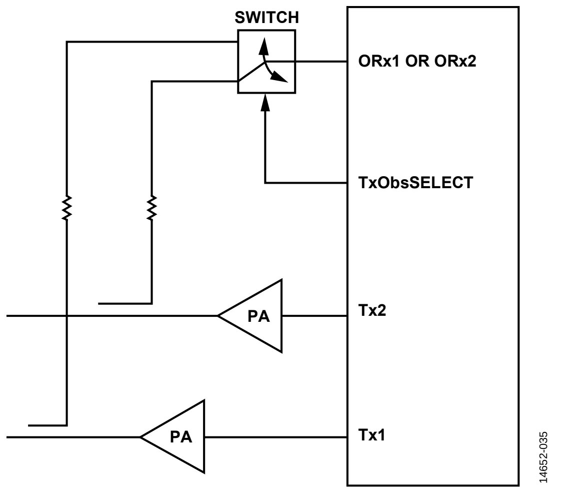


Figure 36. The xObsSelect GPIO Used to Toggle an External Switch (Alternatively, the Output Can Be Fed Back for the BBP to Toggle the Switch)

Initial Calibrations in Two Passes

Due to system considerations, whereby the power amplifier must be off for all calibrations except for the external local oscillator leakage (LOL) initial calibration, it is necessary to run two instances of the following functions:

- `AD9371_runInitCals()`
- `AD9371_waitInitCals()`

In the first instance, all calibrations are set in the calibration mask except for the external LOL initial calibration (D9). The PA is turned off per Figure 34, and the Rx input is terminated as shown in Figure 33. The ARM cycles through each of the calibrations in turn. Upon a successful return from the AD9371_ waitInitCals() function, the baseband processor (BBP) turns on the power amplifiers used in the Tx paths.

In the second instance, only the external LOL initial calibration is run (only D9 is set in the cal mask). The signal chains in the device are then fully calibrated after successful completion of this calibration.

System Considerations for the Tracking Calibrations

This section describes the operation of the tracking calibrations. Figure 37 through Figure 44 shows how the device is configured for each calibration, and a brief explanation of the calibration is provided. In Figure 37, Figure 39, Figure 41, and Figure 43, the grayed out lines and blocks are not active in the calibration. Lines showing the path of the local oscillators (LOs) are shown in black to distinguish them from the signal paths. As the ARM performs each of the calibrations, it is tasked with configuring whether the feedback path or the ORx input is selected. No user input is required in this regard. When utilizing external LOL tracking, however, the ensure that the feedback path is available to use.

Rx QEC Tracking Calibration

The Rx quadrature error correction (QEC) tracking algorithm improves the Rx path QEC performance during operation. It uses normal traffic data to calculate updated corrected coefficients. It runs continuously in the background while the receivers are active. The Rx QEC tracking uses the overload detectors (utilized by the gain control algorithms of the device) to indicate an overload condition within the device. If the device is reporting a high overload condition, it refrains from updating the coefficients on the basis that the data is not representative of QEC performance.

The system requires that the Rx channels be enabled. In time division duplexed (TDD) mode, Rx QEC tracking only runs during Rx periods. If only one channel is enabled, the Rx QEC only runs on this channel.

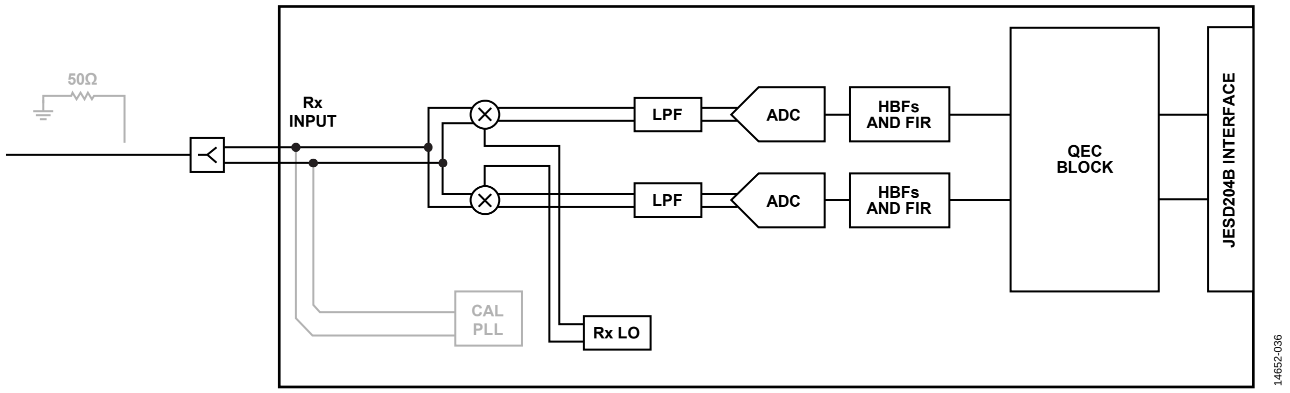


Figure 37. Rx QEC Tracking

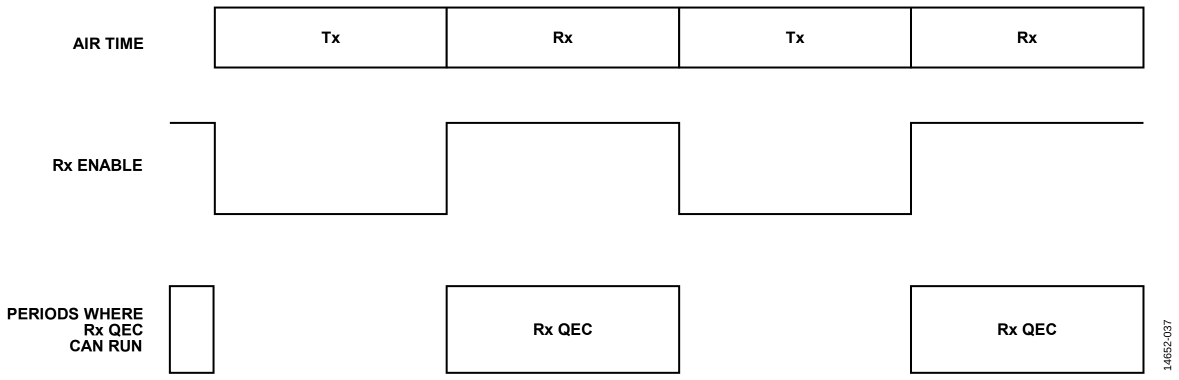


Figure 38. Timing Diagram Showing When Rx QEC Can Run in TDD Mode (In FDD Modes, Rx Enable is High at All Times; Rx Enable Refers to the Enablement of Rx1 and/or Rx2)

ORx QEC Tracking Calibration

The ORx quadrature error correction (QEC) tracking algorithm improves the ORx path QEC performance during operation. It uses normal traffic data (for example, digital predistortion (DPD) capture data) to calculate updated corrected coefficients. It runs continuously in the background while the observation receiver is active.

The system requires that the ORx channel be enabled, as in time division duplexed (TDD) mode; ORx QEC tracking only runs during ORx periods.

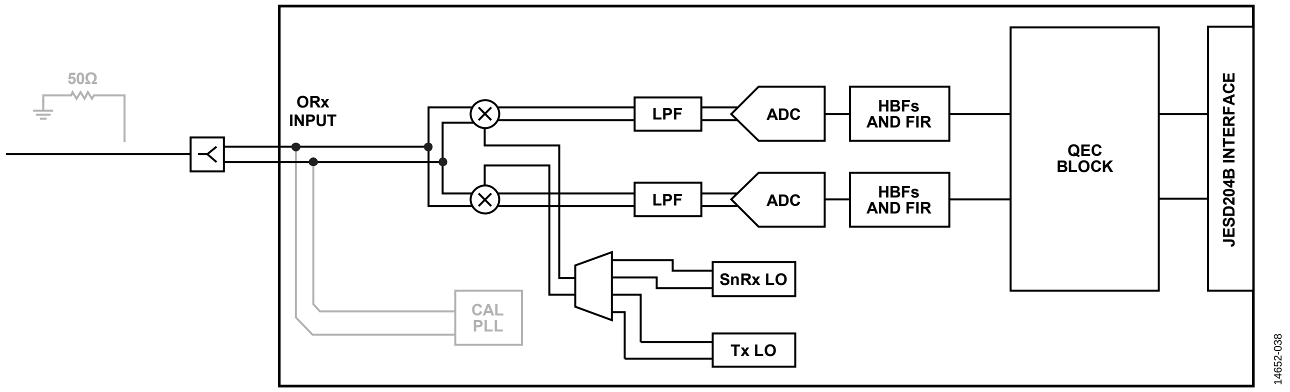


Figure 39. ORx QEC Tracking

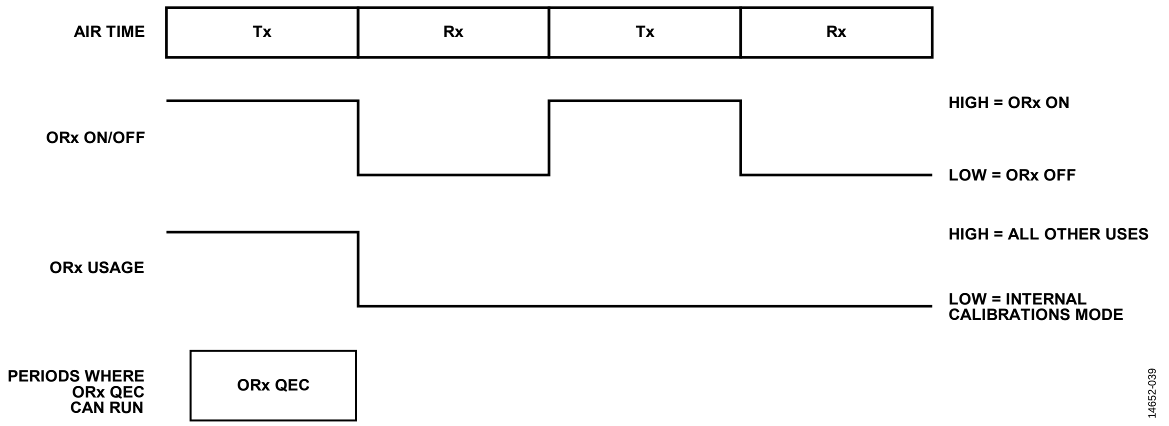


Figure 40. Timing Diagram Showing When ORx QEC Can Run in TDD Mode
(ORx On/Off and ORx Usage Are Not Real Signals Used in the Control of the Device, But are Generalizations of the Control of the ORx Path)

Tx QEC Tracking Calibration

The Tx quadrature error correction (QEC) tracking is an online calibration that runs during transmission to improve the QEC performance. It utilizes the internal loopback path for operation. Therefore, the Tx QEC tracking must be interleaved with normal digital predistortion (DPD) captures (or channel sniffing functions) that utilize the ORx path. This tracking determines optimal coefficients for the current gain setting, updating the table stored during the Tx QEC initialization to make sure this table has the best values for the current operating conditions. Figure 41 shows the device configuration for Tx QEC tracking calibration.

The system requires that the Tx channel(s) be enabled and that the ORx path be available for the ARM to use (for internal calibrations mode).

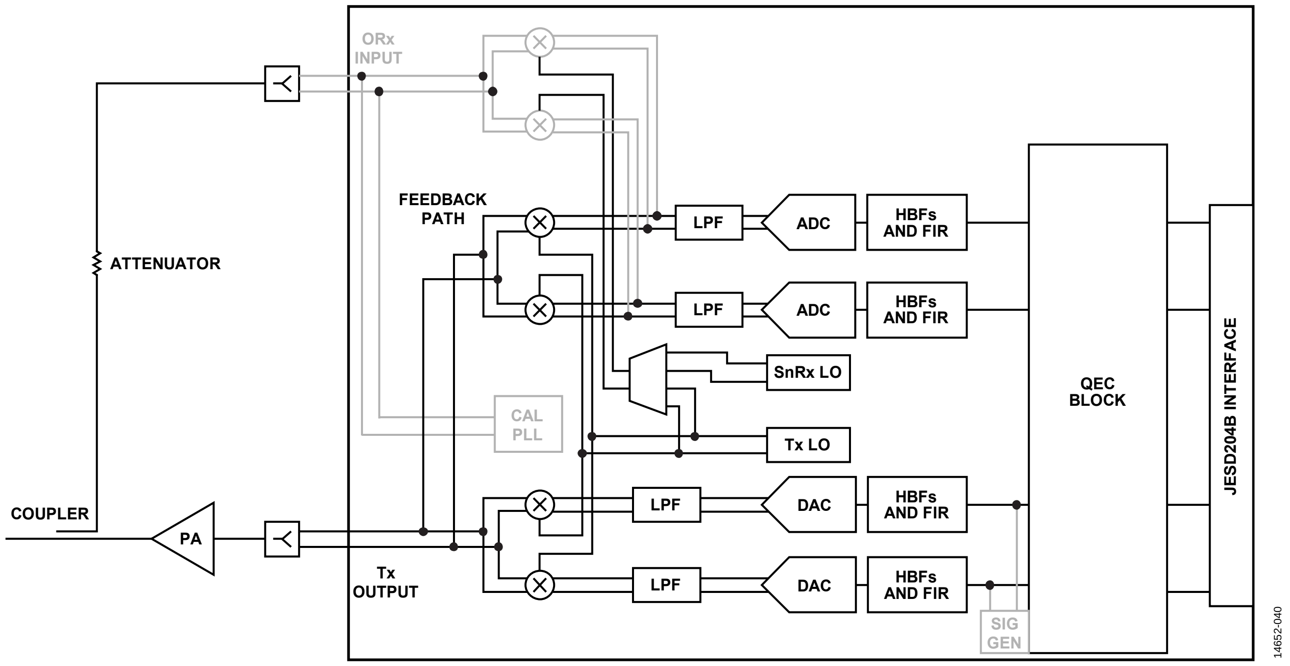


Figure 41. Tx QEC Tracking Calibration Configuration

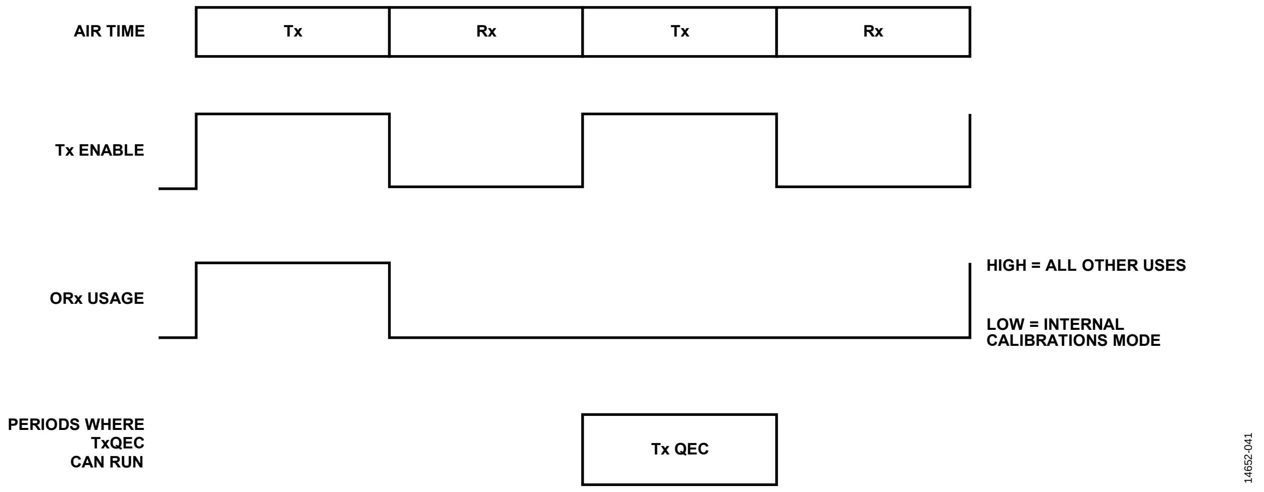


Figure 42. Timing Diagram Showing When Tx QEC Can Run in TDD Mode (In FDD Modes, Tx Enable is High at All Times; Tx Enable Refers to the Enable of Tx1 and/or Tx2; ORx Usage Refers to Either ORx1 if Considering Tx1, and ORx2 is Considering Tx2, as Tx1 is Calibrated with the Internal Feedback Path of ORx1, and So On; Note that ORx Usage is Not a Real Signal Used in the Control of the Device, But is a Generalization of How the ORx is Controlled)

Tx LOL Tracking Calibration

The Tx local oscillator leakage (LOL) tracking calibration uses the external digital predistortion (DPD) path to measure LOL and calculate correction factors. This calibration is run while user data is being transmitted (with the power amplifier operational). Because it utilizes the loopback path, it must be interleaved with normal DPD captures. Figure 43 shows the device configuration for the Tx LOL tracking calibration with the Tx output looped back to the ORx input.

Note that this calibration does not provide good performance as an external LO is provided as the Tx LO. Thus, in such cases, LOL performance is reliant solely on the initialization calibration, and is therefore reduced.

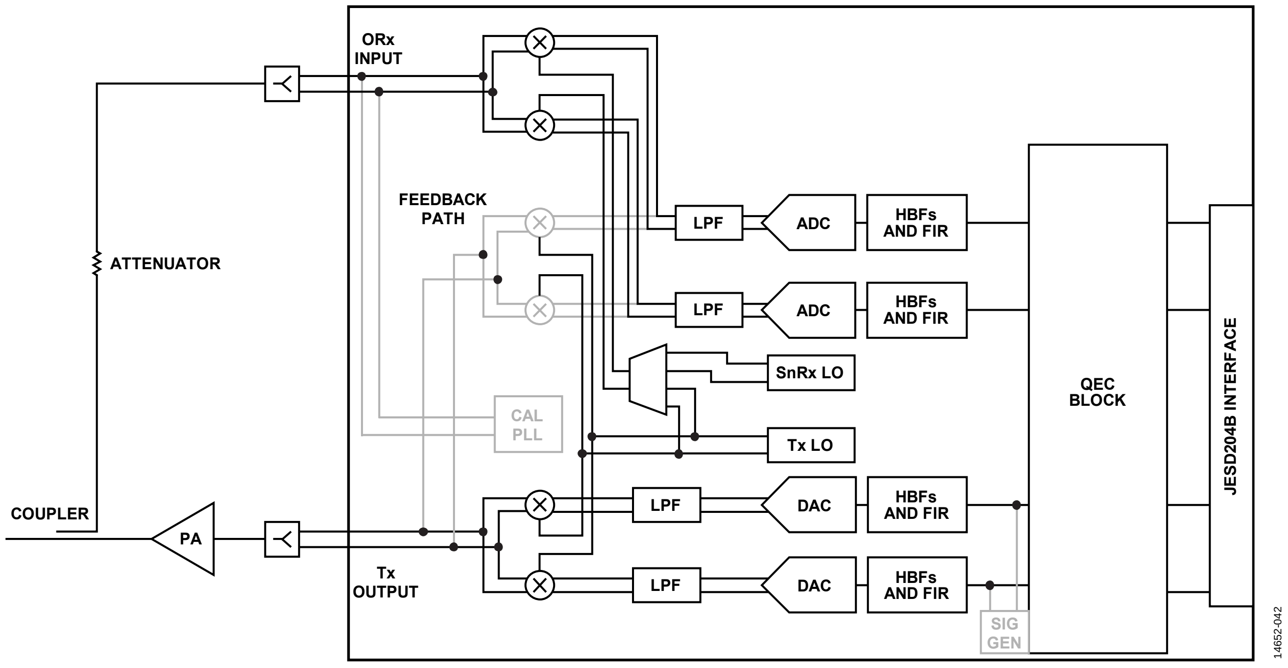


Figure 43. Tx LOL Tracking Configuration

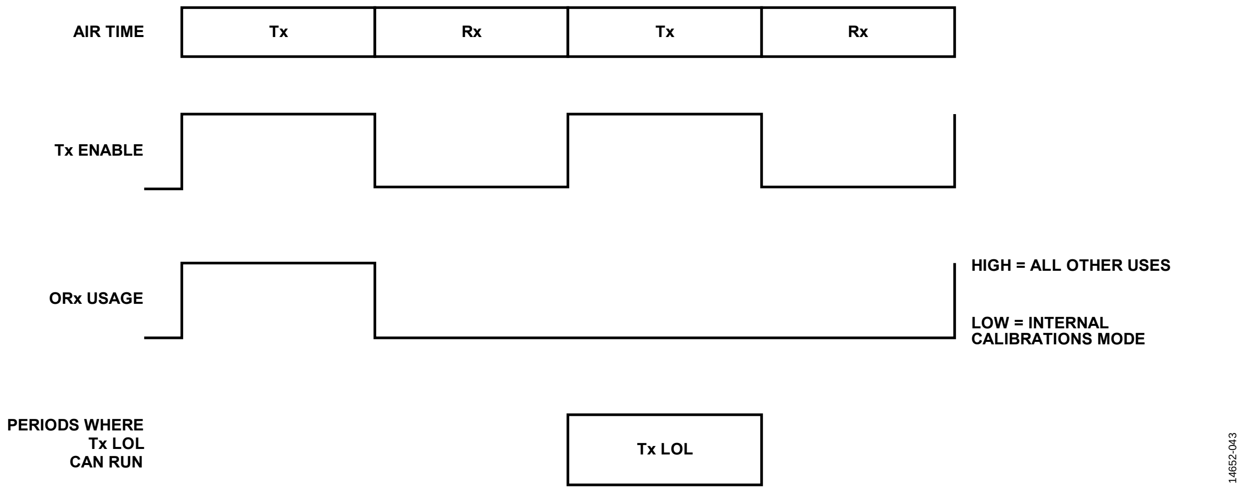


Figure 44. Timing Diagram Showing When Tx LOL Can Run in TDD Mode (In FDD Modes, Tx Enable is High at All Times; Tx Enable Refers to the Enable of Tx1 and/or Tx2; ORx Usage Refers to the Corresponding ORx Path of the Tx Identified for Tx External LOL Calibration; Note that ORx Usage is Not a Real Signal Used to Control the Device But is Instead a Generalization of How the ORx is Controlled)

External Channel

The user must ensure that the appropriate external feedback path is available when the ARM is given access to the ORx path to perform the Tx tracking calibrations. In the case where both Tx channels are being fed back to the same ORx input, the user must ensure that the status of the txObsSelect output is monitored, and that the requested path is available for calibration (see the Initial ARM Calibrations section).

Note that the external local oscillator leakage (LOL) tracking calibration uses an estimate of the external channel (gain/phase rotation) to calculate the correction coefficients. This estimate is updated over time while tracking on Tx data; therefore, any phase, gain drift over time, and temperature can be tracked out. However, sudden changes in the phase and gain of the external path may result in reduced performance until such time as the algorithm tracks the channel changes out.

By default, the algorithm acquires 67% of the new channel estimate in 200 sec. This slow update rate is chosen because the external channel typically changes slowly over time. To obtain an optimal estimate of the external channel in a shorter time, reset the external channel estimate by using the following application programming interface (API) function:

```
MYKONOS_resetExtTxLolChannel(mykonosDevice_t
*device, mykonosTxChannels_t channelSel)
```

where channelSel is the channel for which the external LOL channel estimate must be reset, as per Table 71.

Table 71. Description of channelSel for MYKONOS_resetExtTxLolChannel()

Channel	Enumeration
Tx1	TX1
Tx2	TX2
Tx1 and Tx2	TX1_TX2

After the external LOL channel estimate is reset, the next six tracking instances are used to estimate the external channel. The correction is not updated during this time; it is frozen with the values applied before the API call was issued. After the first six tracking instances, an optimal channel estimate is obtained, and further instances of Tx LOL tracking update the LOL correction coefficients.

If the sudden changes are large enough, the external channel estimate must be reset using the previous command, whereas at other times, it is at the discretion of the user. For cases where the external channel must be reset, include the following:

- If the LO frequency of the device has changed
- If the gain and phase have suddenly changed

Table 72. Gain Error vs. Maximum Phase Error

Gain Error (dB)	Maximum Phase Error (Degrees)
−3	69.26949155
−2.5	67.97895638
−2	66.59898696
−1.5	65.12136412
−1	63.53663696
−0.5	61.83382241
0	60
+0.5	58.0197531
+1	55.87437871
+1.5	53.54073591
+2	50.98950693
+2.5	48.18245508
+3	45.0678624

ARM GPIOs

The ARM has the following interfaces over pins:

- Signal chain enables. The ARM is in control of activating the signal chains of the device under instruction of the user. This can be done through SPI control (application programming interface (API) functions), or through the TXx_ENABLE pin or the RXx_ENABLE pin. (which is explained in the System Control section).
- ORx chain control. The ARM controls the assignment of the ORx path based on instruction from the user. This can be either controlled over the SPI (API functions), or through the four GPIO pins (which is explained in the System Control section).
- ARM acknowledge signals. The ARM can also use the GPIOs to advise that it has activated the chains as per the previously described conditions over SPI input/output pins, which is explained in the System Control section.
- Tx observation select bit. This bit is used by the ARM to indicate which Tx is to be calibrated for local oscillator leakage (see the External Tx LO Leakage Initial Calibration section).
- The GP_INTERRUPT pin. The general-purpose interrupt pin alerts the user when errors occur within the device. A single pin advises numerous potential errors, such as ARM errors, phase-locked loop (PLL) unlocking events, and JESD204B errors. The functionality and configuration of the general-purpose interrupt is discussed in the General-Purpose Input/Output (GPIO) Configuration section. This section also discusses how to mask only certain events trigger the GP_INTERRUPT pin and describes how to determine which event has occurred. If it is determined that the source of the error is an ARM error, reset and reinitialize the device.

INITIAL CALIBRATION ERRORS

This section describes how to determine what error has occurred in the event if an error occurs during the running of the initial calibration. If an error does occur during an initial calibration, isolate the cause of the issue using the following error codes. Then, reinitialize the device with whatever procedure changes are necessary.

An example of such an error is if the external local oscillator leakage (LOL) initial calibration runs without the external feedback path complete. In this event, the calibration reports that it was unable to observe the Tx channel and that the calibration was unsuccessful. This result may be due to an external switch in an incorrect position.

Use the following application programming interface (API) command to verify that these initial calibrations have been completed by the ARM, and it returns error information from the initialization calibrations:

```
MYKONOS_waitInitCals(mykonosDevice_t
    *device, uint32_t timeoutMs, uint8_t
    *errorFlag, uint8_t *errorCode)
```

where timeoutMs is the time in ms the function must wait for the calibrations to complete before returning an error, and *errorFlag and *errorCode are used to indicate if an error has occurred, and if so, which calibration returned the error.

MYKONOS_waitInitCals returns two error values: errorFlag and errorCode. errorFlag advises the error status of the initialization calibrations routine. The returned values are defined in Table 73.

Table 73. errorFlag Parameter Definitions Returned from waitInitCals( )

errorFlag	Description
0x00	Command completed successfully.
0x01	Reserved.
0x02	Command not allowed in radio on state. The calibrations are not run. The device must not be in a transmit or receive state when initial calibrations are called.
0x03	Reserved
0x04	Reserved.
0x05	Radio frequency phase-locked loop (PLL) frequencies are not set prior to running initial calibrations. Calibrations were not run.
0x06	Initialization sequence interrupted by an abort command.
0x07	Calibration error.

Table 74. errorCode Designators as Included in the errorCode Parameter Returned from waitInitCals( )

errorCode	Calibration
0x00	Tx baseband filter calibration
0x01	ADC tuner calibration
0x02	Rx transimpedance amplifier (TIA) filter calibration
0x03	Rx dc offset calibration
0x04	Tx attenuation delay
0x05	Rx gain delay
0x06	ADC flash calibration
0x07	Path delay calibration
0x08	Tx local oscillator leakage (LOL) initial calibration
0x09	Tx LOL external initial calibration
0x0A	Tx quadrature error correction (QEC) initial calibration
0x0B	Loopback ORx LO delay
0x0C	Loopback Rx QEC initial calibration
0x0D	Rx LO delay
0x0E	Rx QEC initial calibration

If the errorFlag returns as 0x07, this result indicates that a calibration error occurred. The errorCode parameter can then be analyzed to advise which calibration error occurred during the initialization calibration routine, as detailed in Table 74.

It is then possible to determine which error occurred using the following application programming interface (API) function:

```
MYKONOS_getInitCalStatus(mykonosDevice_t
   *device, mykonosInitCalStatus_t
   *initCalStatus)
```

where initCalStatus is the mykonosInitCalStatus structure returned with the following elements:

- calsDoneLifetime. This is a bit mask that indicates all the initialization calibrations that have been run since the ARM was booted. For the definition of the bit mask, see Table 65.
- calsDoneLastRun. This is a bit mask that indicates the specific calibrations that were run on the last call to MYKONOS_runInitCals( ). For the definition of the bit mask, see Table 65.
- calsMinimum. This is a bit mask that indicates the set calibrations that must be performed before the ARM allows the user to move it into its radio on state. For the definition of the bit mask, see Table 65.
- initErrCal. This is the code that indicates which calibration returned, if any, an error during MYKONOS_runInitCals( ). It is equivalent to errorCode returned by MYKONOS_waitInitCals( ). For the definition of the bit mask, see Table 74.
- initErrCode. This is the exact error code returned by the calibration if any occurs during MYKONOS_runInitCals( ). See Table 75 to Table 84 for details of the possible errors returned.

Table 75. initErrCodes for the ADC Tuner Calibration

initErrCode	Description
0	No error
1	Calibration timed out

Table 76. initErrCodes for the Rx DC Offset Calibration

initErrCode	Description
0	No error
1	Calibration timed out—Rx
2	Calibration timed out—ORx
3	Calibration timed out—loop back receiver (LBRx)
4	Calibration timed out—SRx

Table 77. initErrCodes for the ADC Flash Calibration

initErrCode	Description
0	No error
1	Calibration aborted
2	Calibration timed out
3	No channel is selected
4	Rx is disabled

Table 78. initErrCodes for the LO Delay Calibration

initErrCode	Description
0	No error
1	Rx is disabled
2	Tx is disabled
3	PLL calibration error
4	Reserved
5	Reserved
6	Reserved
7	Batch time too small

Table 79. initErrCodes for the Path Delay Calibration

initErrCode	Description
0	No error
1	Rx is disabled
2	Tx is disabled
3	Data captured timed out due to hardware setup
4	Data capture aborted

Table 80. initErrCodes for the Rx Quadrature Error Correction (QEC) Initial Calibration

initErrCode	Description
0	No error
1	Rx is disabled
2	Tx is disabled
3	PLL calibration error
4	Settling time error
5	Reserved
6	Reserved
7	Reserved
8	Batch time too small

Table 81. initErrCodes for the Rx Transimpedance Amplifier (TIA) Calibration

initErrCode	Description
0	No error
1	Error configuring PLL—ORx
2	Error during TIA calibration—ORx
3	Error configuring PLL—Rx
4	Error during TIA calibration—Rx

Table 82. initErrCodes for the Tx Baseband Filter Calibration

initErrCode	Description
0	No error
1	Reserved
2	Calibration timed out

Table 83. initErrCodes for the Tx Local Oscillator Leakage (LOL) Calibration

initErrCode	Description
0	No error
1	Reserved
2	Tx is disabled
3	Path delay not present (invalid)
4	Not applicable
5	Not applicable
6	Data capture timed out due to hardware setup
7	GPIO not configured in single ORx mode
8	Tx channel is not observable

Table 84. initErrCodes for the Tx Quadrature Error Correction (QEC) Calibration

initErrCode	Description
0	No error
1	Reserved
2	Tx is disabled
3	No path delay present

TRACKING CALIBRATION ERRORS

This section describes some methods of catching errors in the tracking calibrations. In the event of an ARM exception, the GP_INTERRUPT pin triggers (as indicated previously), advising the user to reset and reinitialize the device.

Alternatively, an error may occur in a calibration that can be read back by polling using the following application programming interface (API) command:

```
MYKONOS_getPendingTrackingCals(mykonosDevice
    _t *device, uint32_t *pendingCalMask)
```

where pendingCalMask is the returned mask that advises if a calibration is pending or has returned an error, as indicated in Table 69.

In the event of an error occurring during one of the calibrations, the error can be cleared and the calibration rescheduled using the following API command:

```
MYKONOS_rescheduleTrackingCal(mykonosDevice_
    t *device, mykonosTrackingCalibrations_t
    trackingCal)
```

where trackingCal is an enumeration that indicates that the calibration must be rescheduled, as indicated by Table 85.

Table 85. Definition of trackingCal Mask for MYKONOS_rescheduleTrackingCal( )

trackingCal	Corresponding Enum	Calibration
0x01	TRACK_RX1_QEC	Rx1 quadrature error correction (QEC) tracking calibration
0x02	TRACK_RX2_QEC	Rx2 QEC tracking calibration
0x04	TRACK_ORX1_QEC	ORx1 QEC tracking calibration
0x08	TRACK_ORX2_QEC	ORx2 QEC tracking calibration
0x10	TRACK_TX1_LOL	Tx1 LOL tracking calibration
0x20	TRACK_TX2_LOL	Tx2 LOL tracking calibration
0x40	TRACK_TX1_QEC	Tx1 QEC tracking calibration
0x80	TRACK_TX2_QEC	Tx2 QEC tracking calibration

READING THE ARM VERSION

After the ARM is booted, it is possible to read back its version using the following application programming interface (API) function:

```
MYKONOS_getArmVersion(mykonosDevice_t
    *device, uint8_t *majorVer, uint8
    minorVer, uint8_t *rcVer)
```

where:

majorVer is the major version of the ARM build.
minorVer is the minor version of the ARM build.
rcVer is the release candidate version (build number).

Each ARM build has a unique combination of these versions, and thus can be determined from these.

PERFORMING AN ARM MEMORY DUMP

As noted in the ARM GPIOs section, the ARM uses the GP_INTERRUPT pin to report if it has detected an error. At this stage, perform an ARM memory dump, and then provide this dump to Analog Devices for diagnostics. There is no application programming interface (API) written to perform a full ARM memory dump because the API is written to file a system diagnostic.

Example code is supplied in the following section for performing such an ARM memory dump operation. This code reads the ARM memory and writes the binary byte data directly to a binary file. Note that an exception is forced if an exception has not already occurred. When an exception occurs, important diagnostic information is stored in the ARM memory. Thus, in the event of the ARM being dumped for debug in situations where an exception has not occurred, this code calls an exception such that this diagnostic information is stored before the ARM memory is dumped.

Example Code for Performing an ARM Memory Dump Operation

```
/// <summary>

        /// Reads the ARM Memory and writes the binary byte array directly to a binary file.  Fi
rst 98304 bytes are program
        /// memory followed by 65536 bytes of data memory.
The binaryFilename is opened before reading the ARM memory to
        /// verify that the filepath is has valid write access before reading ARM memory.
A file IO exception will be
        /// thrown if write access is not valid for the binaryFilename path.
        /// </summary>
        /// <param name="binaryFilename">File path to save the binary data.  Make sure you have
write access to the location.</param>
        /// <exception cref="InvalidOperationException">Thrown if TCPIP is not connected</except
ion>"
        public void dumpArmMemory(string binaryFilename)
        {
            if (this.hw.Connected)
            {
                //Write in BINARY FILE format
                String filename = binaryFilename;
                System.IO.FileStream fileStream = new System.IO.FileStream(filename, System.IO.F
ileMode.Create, System.IO.FileAccess.Write);

                byte[] programMem = new byte[98304];
                byte[] dataMem = new byte[65536];

                //Check if exception has occurred
                byte[] exceptionArray = new byte[4];
                this.readArmMem(0x01017FF0, 4, 1, ref exceptionArray);
                UInt32 exceptionValue = (UInt32)(exceptionArray[0] | (exceptionArray[1] << 8) |
(exceptionArray[2] << 16) | (exceptionArray[3] << 24));

                if (exceptionValue == 0)
                {
                    byte armNotBusy = 0;
                    this.readEventStatus(WAIT_EVENT.ARMBUSY, ref armNotBusy);

                    if (armNotBusy > 0)
                    {
                        //Force an exception during ARM MEM dump for more useful information
                        this.sendArmCommand(0x0A, new byte[] { 0x69 }, 1);

                        System.Diagnostics.Stopwatch stopWatch = new System.Diagnostics.Stopwatc
h();
                        stopWatch.Start();
                        while (exceptionValue == 0)
                        {
                            this.readArmMem(0x01017FF0, 4, 1, ref exceptionArray);
                            exceptionValue = (UInt32)(exceptionArray[0] | (exceptionArray[1] <<
8) | (exceptionArray[2] << 16) | (exceptionArray[2] << 24));

                            //timeout to break while loop
                            if (stopWatch.ElapsedMilliseconds > 5000)
                            {
                                break;
                            }
                        }
                        exceptionValue = 0;
                        stopWatch.Stop();
                    }
                }

                this.readArmMem(0x01000000, programMem.Length, 1, ref programMem);
                this.readArmMem(0x20000000, dataMem.Length, 1, ref dataMem);
                if (exceptionValue == 0)
                {//if we forced an exception, clear the exception so the ARM will continue to ru
n.
                    this.writeArmMem(0x01017FF0, 4, new byte[] { 0, 0, 0, 0 });
                    this.readArmMem(0x01017FF0, 4, 1, ref exceptionArray);
                }

                fileStream.Write(programMem, 0, programMem.Length);
                fileStream.Write(dataMem, 0, dataMem.Length);

                fileStream.Close();
            }
            else
            {
                throw new InvalidOperationException("No Hardware Connection");
            }
        }
```

SYSTEM CONTROL

CONTROL OF SIGNAL CHAINS (Tx/Rx)

The ARM enables and disables the signal chains of the device, which can be performed either through pin control or over the SPI interface. In frequency division duplex (FDD) mode, it is possible to use the application programming interface (API) to enable or disable paths; however, in TDD mode, it is recommended to use pin control of the signal chains to adhere to the strict timing requirements of TDD operation.

ARM Control Mode

If the device is not in pin control mode, it defaults to command mode. In this mode, the ARM enables all signals paths (Tx/Rx) defined in the data structure provided during the initialization of the device upon entering the radio on (operational) state. Likewise, the ARM powers down the signal paths upon leaving the radio on state. The parameters in the device structure that determine the Rx and Tx chains enabled are as follows:

- For Tx channels, device → tx → txChannels.
- For Rx channels, device → rx → rxChannels.

Pin Control Mode

To enable pin control mode, run the following application programming interface (API) function:

```
MYKONOS_setRadioControlPinMode(mykonosDevice
    _t *device)
```

This function relies on the settings stored in the mykonos-ArmGpioConfig_t data structure, which must be included in the device structure at device → auxIO → armGpio. The specific members of the structure used by this function are shown in Table 86.

Table 86. ARM GPIO Configuration Structure Member Descriptions for setRadioControlPinMode

Structure Member	Valid Values	Description
txRxPinMode	0, 1	0 = ARM command mode for powering up or powering down the Rx/Tx chains 1 = pin control mode for powering up or powering down the Rx/Tx chains
orxPinMode	0, 1	0 = ARM command mode for controlling the ORx receiver 1 = pin control mode for controlling the ORx receiver
useRx2EnablePins	0, 1	0 = use the RX1_ENABLE pin to power up or power down both Rx1 and Rx2 1 = use the RX1_ENABLE pin to power up or power down Rx1, and use the RX2_ENABLE pin to power up or power down Rx2
useTx2EnablePins	0, 1	0 = use the TX1_ENABLE pin to power up or power down both Tx1 and Tx2 1 = use the TX1_ENABLE pin to power up or power down Tx1, and use the TX2_ENABLE pin to power up or power down Tx2

Pin control mode of the signal chains is performed with the TXx_ENABLE and RXx_ENABLE pins, as shown in Figure 45. When TX_ENABLE is high, the ARM activates the Tx chain. When TX_ENABLE is low, the Tx chain is disabled by the ARM, and vice versa for the RX_ENABLE signals.

Two enable pins exist for each receiver and transmitter. TX1_ENABLE and RX1_ENABLE can enable the Tx1 and Rx1 channels, respectively, while TX2_ENABLE and RX2_ENABLE can enable the Tx2 and Rx2 channels, respectively.

Alternatively, TX1_ENABLE can enable both Tx1 and Tx2 simultaneously, and RX1_ENABLE can enable Rx1 and Rx2 simultaneously.

Table 87 describes the minimum time allowed for the Tx/Rx chains to enable in any one instance. Note that the minimum time required for calibrations to complete is 800 µs. As noted in the Tracking Calibration Scheduler section, the tracking calibrations require at least 800 µs of data for a meaningful observation. Thus, a slot period must also be of this duration if a tracking calibration is run during this slot period.

Note that this function is automatically called at the end of MYKONOS_ loadArmFromBinary( ). Thus, the function is only called again if the power-up or power-down control method of the baseband processor (BBP) changes. The control method can only be changed when the device is in a radio off state.

Note also that if in pin control mode (txRxPinMode = 1), and Tx1 is used to control both chains (Tx1 and Tx2), TX2_ENABLE must still be controlled and low at all times. If the pin is not used in the reference design, ground it.

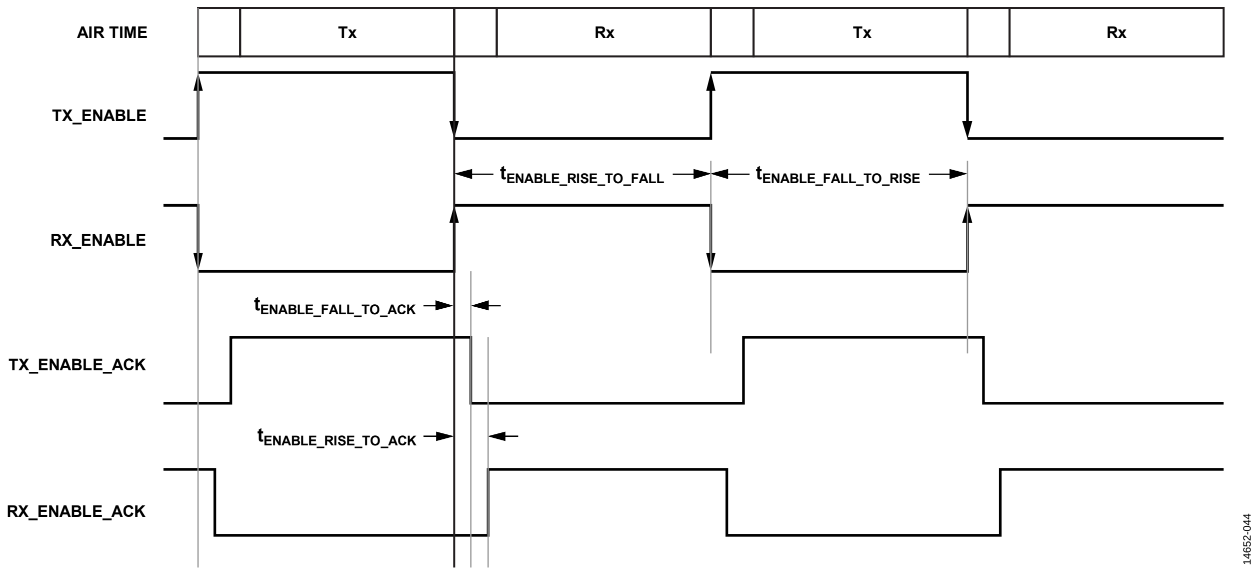

AIR TIME
Tx
Rx
Tx
Rx
TX_ENABLE
$t_{ENABLE\_RISE\_TO\_FALL}$
$t_{ENABLE\_FALL\_TO\_RISE}$
RX_ENABLE
$t_{ENABLE\_FALL\_TO\_ACK}$
TX_ENABLE_ACK
$t_{ENABLE\_RISE\_TO\_ACK}$
RX_ENABLE_ACK
14652-044

Figure 45. Control of Signal Chains Using RX_ENABLE and TX_ENABLE

Table 87. Minimum Time of Active Periods[1]

Symbol	Description	Min	Min for Calibrations	Max
$t_{ENABLE\_RISE\_TO\_FALL}$	Tx/Rx enable rising edge to enable falling edge—enable signal width high	10 µs	800 µs	N/A
$t_{ENABLE\_FALL\_TO\_RISE}$	Tx/Rx enable falling edge to enable rising edge—enable signal width low	10 µs	800 µs	N/A
$t_{ENABLE\_FALL\_TO\_ACK}$	Tx/Rx enable falling edge to acknowledge signal to baseband processor (BBP) going low	N/A	N/A	2 µs
$t_{ENABLE\_RISE\_TO\_ACK}$	Tx/Rx enable rising edge to acknowledge signal to BBP going high	N/A	N/A	2 µs

[1] N/A means not applicable.

ORx PATH CONTROL

Table 88 defines the different operational modes of the ORx path, each with a unique front-end configuration. Two methods can control the ORx path assignment:

1. Application programming interface (API) function, where the following API function can send instructions to the ARM over the SPI interface to change the ORx path assignment:

   ```
   MYKONOS_setObsRxPathSource(mykonosDevice_
       t *device, mykonosObsRxChannels_t
       obsRxCh)
   ```

   where obsRxCh is an enumeration defined in t_mykonos.h, detailed in Table 88.

2. Pin control, where the ARM can also monitor four GPIO pins to initiate a change in the ORx configuration. Three pins determine the ORx mode (ORX_MODE[D2:D0], see Table 88). The state of the other pin is assigned to the ORX_TRIGGER signal. The rising edge of the ORX_TRIGGER signal indicates to the ARM that the ORx mode pins must be immediately sampled to change the ORx mode. Figure 46 shows an example timing diagram where ORX_MODE_2 is Bit D2, and so on. The following API function is used to configure which GPIO pins are used for the ORx mode control if pin control is selected:

   ```
   MYKONOS_setArmGpioPins(mykonosDevice_t
       *device)
   ```

This function is explained in the ARM GPIOs section.

To select the required mode for ORx path control, the following application programming interface (API) function is used:

```
MYKONOS_
    setRadioControlPinMode(mykonosDevice_t
    *device)
```

This function is described in the Control of Signal Chains (Tx/Rx) section.

Table 88. ORX_MODE[D2:D0] Word Definitions

ORx Path Front End	ORX_MODE[D2:D0]	obsRxCh Enumeration
ORx Off	000	OBS_RXOFF
ORx1 with Tx Local Oscillator (LO)	001	OBS_RX1_TXLO
ORx2 with Tx LO	010	OBS_RX2_TXLO
ARM Calibrations	011	OBS_INTERNALCALS
Sniffer Channel	100	OBS_SNIFFER
ORx1 with Sniffer LO	101	OBS_RX1_SNIFFERLO
ORx2 with Sniffer LO	110	OBS_RX2_SNIFFERLO

As noted previously, the Tx tracking algorithms need access to the ORx path so that the Tx data can be monitored and the ARM can optimize the current correction coefficients. For this reason, it is necessary to share the ORx path between digital predisposition (DPD) data acquisition and tracking calibrations for the Tx path (the SRx also requires a share of the available ORx periods, if used). The ARM is tasked with scheduling its tracking algorithms and ensuring that its correction coefficients are optimal at all times.

The ARM is not in control of when it has access to the ORx path for calibration, which ensures that the ARM cannot interrupt a DPD data acquisition. It is tasked to the baseband processor (BBP) to ensure that the ARM has sufficient access to the ORx path to complete its calibrations. To ensure that the ARM can keep corrections optimal, it must have access to the ORx path for a time no less than 400 ms of the transmit time for every 2 sec.

An additional requirement of the ORx path assignment relates to the minimum duration of an ARM tracking calibration in any one instance. This duration is 800 µs (see Table 89). This requirement is because tracking calibrations require 800 µs of data to make a valid observation of the transmitter output signal. If durations of less than 800 µs are used, the algorithms cannot make proper adjustments in the path settings, and thus discard the result obtained during this instance. When only periods of less than 800 µs are used, the algorithms never update their coefficients. When using a duration of 1.6 ms, the calibration algorithm samples two valid observations sequentially. However, if the duration is 1 ms, the algorithm makes only one observation, taking 800 µs, and the result from the remaining 200 µs of data is discarded.

A tracking algorithm may need a number of observations to update the correction coefficients. However, the ARM scheduler pauses the tracking algorithms when the ORx path is used for other purposes and continues when the ORx is reassigned for its calibrations. Table 89 shows important timing parameters for ORx pin control mode. However, the $t_{ORX\_TRIGGER\_RISE\_TO\_RISE}$ parameter is applicable in both modes of operation, with 800 µs being the minimum duration for an ORx mode.

Figure 46 also shows ORX1_ENABLE_ACK, ORX2_ENABLE_ACK, and SRX_ENABLE_ACK. These acknowledge signals can output on the GPIOs and indicate that the ARM enabled the relevant channel for operation. A full list of the available acknowledge signals, and how they are configured, is provided in the ARM GPIO Operation section.

Table 89. Observation Receiver Signal Timings

Timing Parameter	Description	Min[1]	Max[1]
$t_{ORx\_TRIGGER\_RISE\_TO\_RISE}$	ORx trigger frequency—minimum duration in an ORx mode	800 µs	N/A
$t_{ORx\_TRIGGER\_HOLD}$	ORx trigger hold time	1 µs	N/A
t_{MODE_SETUP}	ORx mode setup time before ORx trigger rising edge	1 µs	N/A
t_{MODE_HOLD}	ORx mode hold time	2 µs	N/A
t_{MODE_ACK}	ORx mode acknowledge signal to baseband processor (BBP) from ORx trigger rising edge	N/A	2 µs

[1] N/A means not applicable.

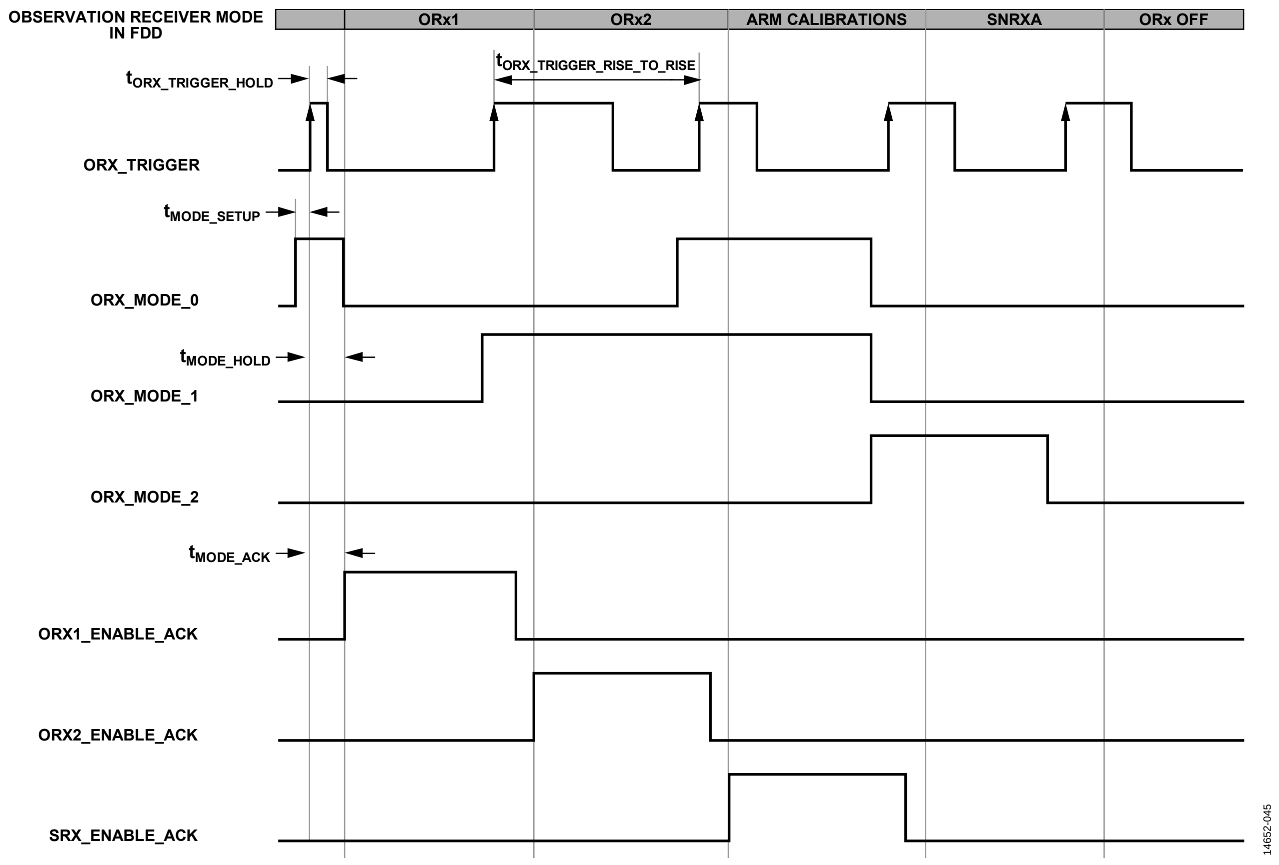


Figure 46. Observation Receiver Pin Control Timing Diagram

Table 90. ARM GPIO Configuration Structure Member Descriptions for setArmGpioPins( )

Structure Member	Input or Output	Available on GPIO Pins
orxTriggerPin	Input	4 … 15
orxMode2Pin	Input	0 … 15, 18
orxMode1Pin	Input	0 … 15, 17
orxMode0Pin	Input	0 … 15, 16
rx1EnableAck	Output	0 … 15
rx2EnableAck	Output	0 … 15
tx1EnableAck	Output	0 … 15
tx2EnableAck	Output	0 … 15
orx1EnableAck	Output	0 … 15
orx2EnableAck	Output	0 … 15
srxEnableAck	Output	0 … 15
txObsSelect	Output	0 … 15

ARM GPIO OPERATION

The ARM is assigned the task of enabling and disabling the chains of the device. In pin control mode, Tx and Rx enabling is performed with the defined Rx and Tx enable pins; however, the ORx enabling is performed through four GPIOs used to select from the different enabling options. The ARM can also use GPIOs to advise the baseband processor (BBP) of the current signal path control, with acknowledge signals that advise when the path is enabled. When high, an acknowledge signal advises that the path is enabled.

The following application programming interface (API) function is used to control which GPIO pins are used by the ARM for communication with the BBP, and to enable pin control mode of the Rx and Tx signal chains:

```
MYKONOS_setArmGpioPins(mykonosDevice_t
    *device)
```

This function is automatically called during the loadArm-FromBinary( ) function call. This function relies on the settings stored in the mykonosArmGpioConfig_t data structure, which is included in the device structure at device → auxIO → armGpio. The specific members of this structure used by this function are detailed in the following sections.

Tx POWER CONTROL

The device features transmitter power control (TPC) to provide precise control of the transmitter output power. The attenuation control allows 41.95 dB of attenuation within the transmitter datapath with a minimum resolution of 0.05 dB. Note that transmitter performance may degrade at attenuation settings greater than 20 dB.

Two transmitter signal path components have variable attenuation settings. These include the analog RF attenuator located after the mixer, and the digital attenuator located prior to the digital filters. Refer to Figure 47 for a simplified block diagram depicting the variable attenuation stages.

Two modes of interaction regarding the TPC are as follows:

- SPI mode. This mode uses the SPI to send a command to change the Tx1 or Tx2 attenuation. The resolution of the attenuation step size is a minimum of 0.05 dB. Separate commands exist for control of Tx1 or Tx2.
- GPIO mode. This mode allows changes of the Tx1 or Tx2 attenuation based on a low to high transition on selected low voltage GPIO pins. Separate pins can be assigned for Tx1 increment attenuation, Tx1 decrement attenuation, Tx2 increment attenuation, and Tx2 decrement attenuation. The resolution of the attenuation step size can be set to multiples of 0.05 dB up to 1.55 dB in the GPIO mode.

In SPI mode, resolution of attenuation control can be selected as 0.05 dB, 0.1 dB, 0.2 dB, or 0.4 dB. This control is set within the device data structure, in device → tx → txAttenStepSize. The control is of data type mykonosTxAttenStepSize_t, whose enumerated values are described in Table 91. Note that this value is programmed to device registers during the MYKONOS_initialize() command.

Table 91. mykonosTxAttenStepSize_t Enumeration Values and Interpretation

mykonosTxAttenStepSize_t Enumeration	Enumeration Value	Tx Attenuation Step Size (dB)
TXATTEN_0P05_DB	0	0.05
TXATTEN_0P1_DB	1	0.1
TXATTEN_0P2_DB	2	0.2
TXATTEN_0P4_DB	3	0.4

In SPI mode, the application programming interface (API) commands used to change the Tx attenuation setting are as follows:

```
MYKONOS_setTx1Attenuation(mykonosDevice_t
    *device, uint16_t tx1Attenuation_mdB)
MYKONOS_setTx2Attenuation(mykonosDevice_t
    *device, uint16_t tx2Attenuation_mdB)
```

These commands can be called in the radio on or radio off states. If the tx1Attenuation_mdB or tx2Attenuation_mdB to this function is not a multiple of the Tx attenuation step size, the value is rounded down to the nearest multiple of the txAttenStepSize value.

Additionally, API commands can retrieve the current Tx attenuation value. These commands can be used in either SPI or GPIO mode. If the Tx datapath is powered down when these commands are called, the last valid Tx attenuation setting when the Tx was powered up is read back. These commands are as follows:

```
MYKONOS_getTx1Attenuation(mykonosDevice_t
    *device, uint16_t *tx1Attenuation_mdB)
```

- `MYKONOS_getTx2Attenuation(mykonosDevice_t *device, uint16_t *tx2Attenuation_mdB)`

Refer to the General-Purpose Input/Output (GPIO) Configuration section for information regarding configuration and operation of GPIO TPC mode.

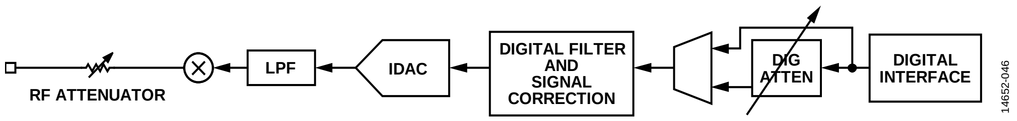


Figure 47. Variable Attenuation Elements for Transmitter Power Control (TPC)

POWER AMPLIFIER (PA) PROTECTION

The transmitter channels feature a protection mechanism that can be enabled to help prevent damage to the PA connected to either transmitter output. This feature is referred to as PA protection. When the full-scale output power of the device exceeds the maximum input to the PA, the overload can result in damage to the PA device. The PA protection feature implements feedback in the system to prevent such an overload by measuring the signal level and comparing it to the user-programmable threshold. This information can be used by the device to reduce the transmit output level on the flagged channel and to eliminate the threat of damage.

Even though the PA protection feature includes independent power measurement blocks for each transmitter channel, this feature cannot be enabled for one channel at a time. If an overload is detected on one channel and not another, the overloaded channel asserts a PA error flag specific to that channel.

The following sections describe the components of the PA protection system.

PA ERROR FLAG

An overload condition is noted by the PA error flag. This structure contains a bit for each transmitter that is asserted when an overload on that channel has occurred. Table 92 describes the PA error flag bits.

Table 92. PA Error Flag Conditions

PA Error Flag, [D1:D0]	Description
D0	1 = an overload on Tx1 has been detected. 0 = no overload condition detected on Tx1.
D1	1 = an overload on Tx2 has been detected. 0 = no overload condition detected on Tx2.

When one of the PA error flag bits asserts, the method by which PA protection operates depends on how the system has been configured. The options follow:

- The baseband processor (BBP) polls the device periodically to check the PA error flag status and uses the application programming interface (API) command, MYKONOS_getPaProtectErrorFlagStatus(…).
- The GP_INTERRUPT pin can be configured to allow PA error flag D1 or D0 to control the status of the pin (see the General-Purpose Interrupt Overview section for details on the MYKONOS_configGpInterrupt(…) command). Status can be monitored by the BBP through the GP_INTERRUPT pin, or by reading back the GP_INTERRUPT status with the API command, MYKONOS_readGpInterruptStatus(…).
- The assertion of D1 or D0 can trigger an automatic increase in Tx attenuation that persists until the signal level on the corresponding channel is reduced less than the programmed power threshold. The attenuation change is programmable in steps of 0.2 dB up to 25.4 dB. The attenuation setting occurs after the PA error flag is asserted and persists until the bit is cleared. Note that if the PA error flag is configured as a sticky error flag, the attenuation change stays in effect until the error flag is manually cleared. This mode is enabled during PA protection setup using the txAttenuationControlEnable parameter. The attenuation step size is set by the attenStepSize parameter.

PROTECTION ALGORITHM

The PA protection system monitors the 12 most significant bits (MSBs) sent to the Tx1 and Tx2 inputs during user defined measurement intervals. The signal level measurement is made prior to all on-chip digital filtering. PA protection is configured with the MYKONOS_setupPaProtection(…) command and enabled through the MYKONOS_enablePaProtection(…) command.

When the power measurement is completed, the PA protection block stores the results for the Tx1 and Tx2 datapaths and asserts a PA error flag if the level measured in either path is greater than the user defined threshold. The level measurement is an instantaneous power measurement performed by calculating $I^2 + Q^2$ of samples. The baseband processor (BBP) can read back the most recently computed measurement for the Tx1 and Tx2 channels by using the MYKONOS_getDacPower(…) command. This command can only be used when PA protection is enabled.

The PA error flag is asserted as soon as the power measurement determines that the measured channel power level is greater than the user programmed power threshold. This flag can be configured to remain high until the user issues the clear error command MYKONOS_clearPaErrorFlag(…) by setting the stickyFlagEnable field to 1. If an overload condition occurs with the stickyFlagEnable field set to 1 and txAttenControlEnable set to 1, the PA protection system will make a single attenuation reduction that persists until the user manually clears the PA error flag. If the stickyFlagEnable field is set to 0, the PA error flag remains high until the overload condition is not present. When the PA error flag is high, attenuation cannot be modified by the user.

The PA protection functionality is summarized in Figure 48, along with references to programming instructions.

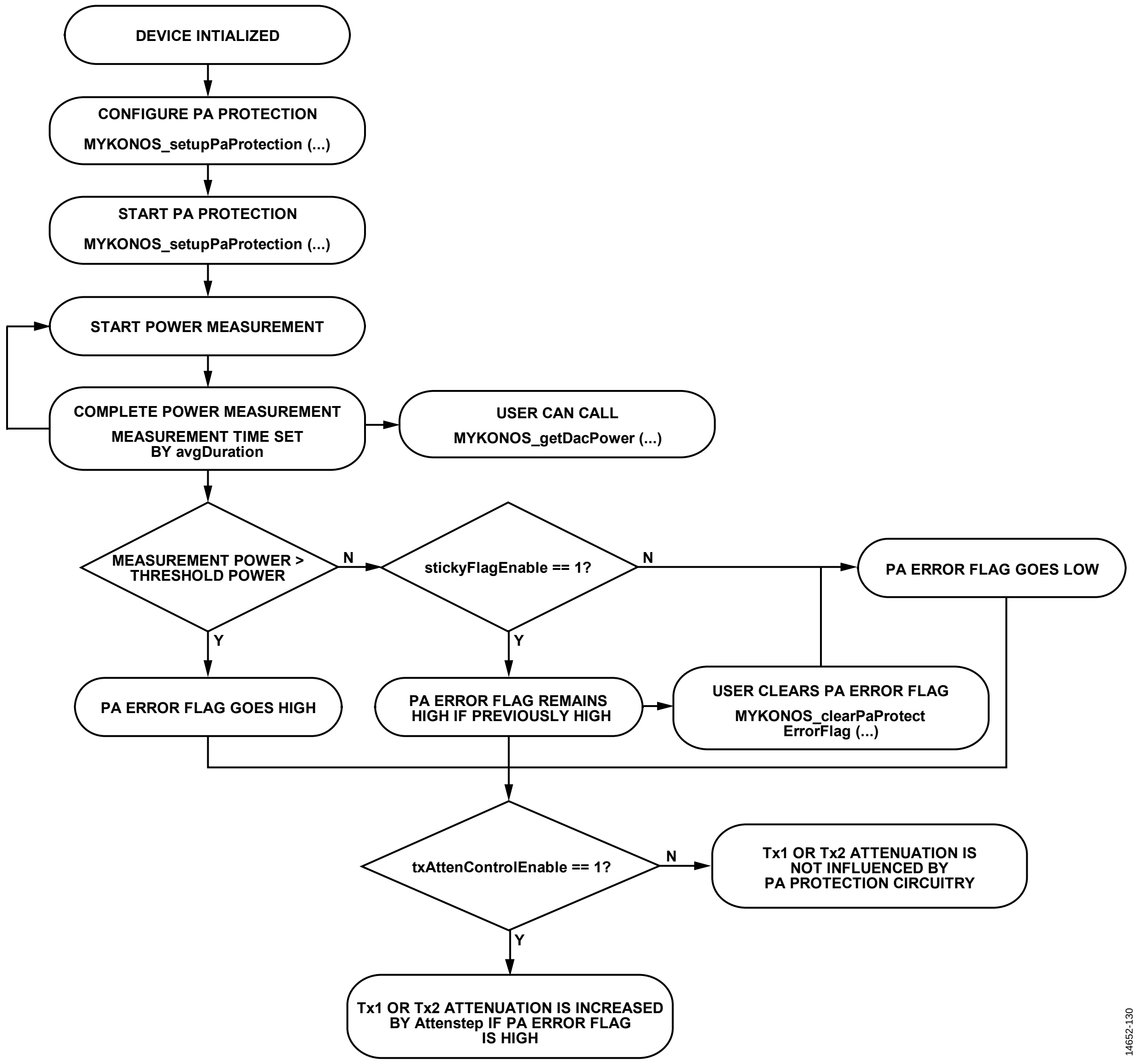

DEVICE INTIALIZED
CONFIGURE PA PROTECTION
MYKONOS_setupPaProtection (...)
START PA PROTECTION
MYKONOS_setupPaProtection (...)
START POWER MEASUREMENT
COMPLETE POWER MEASUREMENT
MEASUREMENT TIME SET BY avgDuration
USER CAN CALL
MYKONOS_getDacPower (...)
MEASUREMENT POWER > THRESHOLD POWER
N
stickyFlagEnable == 1?
N
PA ERROR FLAG GOES LOW
Y
Y
PA ERROR FLAG GOES HIGH
PA ERROR FLAG REMAINS HIGH IF PREVIOUSLY HIGH
USER CLEARS PA ERROR FLAG
MYKONOS_clearPaProtect ErrorFlag (...)
txAttenControlEnable == 1?
N
Tx1 OR Tx2 ATTENUATION IS NOT INFLUENCED BY PA PROTECTION CIRCUITRY
Y
Tx1 OR Tx2 ATTENUATION IS INCREASED BY Attenstep IF PA ERROR FLAG IS HIGH
14652-130

Figure 48. PA Protection Operational Flowchart

API COMMANDS FOR PA PROTECTION

The following sections provide detailed information regarding the application programming interface (API) commands used to setup, enable, and read back status information for the power amplifier protection system.

MYKONOS_setupPaProtection(…)

```
mykonosErr_t MYKONOS_setupPaProtection
(mykonosDevice_t *device, uint16_t
powerThreshold, uint8_t attenStepSize,
uint8_t avgDuration, uint8_t
stickyFlagEnable, uint8_t
txAttenControlEnable)
```

This command writes to device registers with settings for the PA protection block. It does not enable the PA protection functionality. Note that independent control of the PA protection feature for both Tx channels is not possible. The PA protection block allows for PA error flags to go high if the accumulated power in the datapath exceeds a programmable threshold level based on samples taken in a programmable duration.

Preconditions

This function can only be called after the MYKONOS_ initialize(…) command.

Parameters

- *device: This is the pointer to the device data structure.
- powerThreshold: This parameter sets the power level at which the power amplifier error flag is raised. This threshold applies to both the Tx1 and Tx2 data paths. The range is 0 to 4095. Use the following equation to calculate the desired power threshold in dBFS converted to the powerThreshold used in this function:

  $$powerThreshold = 4095 \times 10^{((txPowerThresh\_dBFS)/10)}$$

  where:
  powerThreshold is the value input to this API command.
  txPowerThresh_dBFS is the power threshold level in dBFS relative to the Tx DAC full scale.
- attenStepSize: this parameter sets the attenuation step size when the Tx attenuation control (txAttenControlEnable) is enabled. The range is 0 to 127 with a resolution of 0.2 dB per LSB.
- avgDuration: this parameter sets the number of clock cycles that the power amplifier protection power measurement block uses to compute an estimate for the power in the Tx1 or Tx2 datapath. The range is from 0 to 14. Each LSB corresponds to 2^5 samples; 0 corresponds to 32 samples, 1 corresponds to 64 samples. The samples are clocked at the Tx IQ data rate.
- stickyFlagEnable: 1 enables the power amplifier (PA) error flags to stay high after an overload occurs, even if the data path power later decreases less than the power threshold. When the PA error flag is sticky, the error condition persists until the user manually clears the PA error flag through the MYKONOS_clearPaProtectErrorFlag(…) command. When 0, it disables this functionality.
- txAttenControlEnable: When 1, it enables autonomous attenuation changes in response to the PA error flag state, and when 0, it disables this functionality.

MYKONOS_enablePaProtection(…)

```
mykonosErr_t MYKONOS_enablePaProtection
(mykonosDevice_t* device, uint8_t
paProtectEnable)
```

This command enables the power amplfier protection block according to the parameters passed in MYKONOS_setupPaProtection(…).

Preconditions

Before calling this function, setup the power amplifier protection block by calling MYKONOS_setupPaProtection(…).

Parameters

- * device: This is a pointer to the device data structure.
- paProtectEnable: When 1, it enables the power amplifier (PA) protection block, and when 0, it disables the PA protection block.

MYKONOS_clearPaProtectErrorFlags(…)

```
mykonosErr_t MYKONOS_clearPaErrorFlag
   (mykonosDevice_t* device)
```

This function manually clears the power amplifier (PA) error flags. Set up the PA protection block to enable sticky error flags. Sticky error flags require the user to clear the bit manually even if the accumulated power is less than the power threshold for the PA protection block.

Preconditions

Enable the power amplifier protection block with the sticky error flags field bit set to 1.

Parameters

- * device: This is a pointer to the device data structure.

MYKONOS_getDacPower(…)

```
mykonosErr_t MYKONOS_getDacPower
(mykonosDevice_t *device,
mykonosTxChannels_t channel, uint16_t
*channelPower)
```

This function obtains an estimate of the accumulated power of the Tx channel over the sample duration provided in MYKONOS_setupPaProtection(...). It uses the avgDuration parameter provided in MYKONOS_setupPaProtection to set the number of samples to accumulate to obtain an estimate for a Tx channel specified by the channel parameter. A 12-bit field estimating the channel power is returned in the *channelPower pointer. Use the following equation to calculate the dBFS value of the reading:

$$txChannelPower_{dBFS} = 10 \times \log_{10}(channelPower/4095)$$

where:
txChannelPower$_{dBFS}$ is the channel power when converted into units of dBFS relative to the Tx DAC full scale.
channelPower is the value of the pointer stored by this command.
For example, if channelPower is reading 409, the channel power in dBFS is −10 dBFS.

Preconditions

Enable the power amplifier protection block.

Parameters

- *device: This is a pointer to the device data structure.
- channel: selects the Tx channel power measurement to obtain. Only use Tx1 (1) or Tx2 (2) enumerations for mykonosTxChannels_t.
- *channelPower: This is a pointer that stores the power of the selected channel. Readback is provided as a 12-bit value.

MYKONOS_getPaProtectErrorFlagStatus(…)

```
mykonosErr_t
MYKONOS_getPaProtectErrorFlagStatus
(mykonosDevice_t *device, uint8_t
*errorFlagStatus)
```

This function provides a readback of the power amplifier (PA) protection error flag status through the *errorFlagStatus pointer.

Preconditions

Enable the power amplifier protection block.

Parameters

- *device: This is a pointer to the device data structure.
- *errorFlagStatus: This is a pointer that stores the error flag status indicating which Tx channel error flags are set.

REFERENCE CLOCK AND SYSREF CONNECTIONS

The external clock is used as the reference clock for the Rx PLL, Tx PLL, SnRx PLL, and the clocking PLL circuits in the device. To maintain the highest performance levels, a clean clock source is required. This external clock source must be input into the DEV_CLK_IN+ and DEV_CLK_IN− pins. Within this documentation, DEV_CLK refers to the reference clock signal supplied to the device, and DEV_CLK_IN refers to the differential pair input pins on the device.

CONNECTIONS FOR EXTERNAL CLOCK (DEV_CLK_IN)

The reference clock must be supplied as a differential signal connected to E7 and E8. This connection must be terminated with a 100 Ω resister and be ac-coupled as shown in Figure 49. The device input pins are biased to 618 mV. The inputs are high impedance, with less than 1 pF and 20 kΩ each. The frequency range of the DEV_CLK signal must be between 10 MHz and 320 MHz. The maximum voltage level for the DEV_CLK signal is 2.0 V p-p differential, and the minimum input level is 300 mV p-p differential.

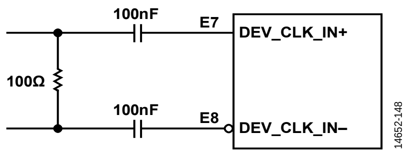


Figure 49. Reference Clock Input Connections

Printed circuit board (PCB) routing is made difficult by the location of the DEV_CLK_IN± balls: they are located in the middle of the ball grid array. To avoid potential coupling of the reference input clock to the RF signals, it is recommended to place the termination resistor and ac coupling capacitors on the opposite side of the PCB from the device, and to use vias to route the clock signals up to the device as close to the input balls as possible to complete the connections. More information regarding PCB routing can be found in the Printed Circuit Board Layout Guidelines section.

DEV_CLK PHASE NOISE REQUIREMENTS

To prevent performance degradation, the DEV_CLK reference must be a stable, low noise signal. Table 93 lists the required phase noise of the DEV_CLK signal to ensure device performance for a 153.6 MHz input and RF frequencies as high as 6000 MHz. Similar phase noise requirements for a 122.88 MHz input are displayed in Table 94.

Table 93. DEV_CLK Phase Noise Requirements, 153.6 MHz Reference

Frequency Offset from Carrier	Phase Noise Level (dBc/Hz)
100 Hz	−101
1000 Hz	−122
10 kHz	−132
100 kHz	−134
1 MHz	−145
10 MHz	−155

To scale the phase noise requirement for the DEV_CLK signal to different frequencies using the following equation:

$$Phase\,Noise\,Level = 20 \times \log \frac{New_{DEV\_CLK}}{153.6\text{ MHz}} \quad (2)$$

For example, for DEV_CLK = 122.88 MHz

$$20 \times \log(122.88\text{ MHz}/153.6\text{ MHz}) = -193\text{ dB}$$

Table 94. DEV_CLK Phase Noise Requirements, 122.88 MHz Reference

Frequency Offset from Carrier	Phase Noise Level (dBc/Hz)
100 Hz	−103
1000 Hz	−124
10 kHz	−134
100 kHz	−136
1 MHz	−147
10 MHz	−157

SYSREF REQUIREMENTS

The device provides two balls for the SYSREF interface: SYSREF_IN+ (K3) and SYSREF_IN− (K4). The SYSREF interface internally aligns the generated clocks. It also provides a mechanism for deterministic latency per the JESD204B standard. Figure 50 outlines where each SYSREF input pulse is directed.

The SYSREF input also provides multichip synchronization (MCS) for systems with more than one device. Synchronization is accomplished by driving multiple chips with the same SYSREF signal (see the Multichip Synchronization section).

This SYSREF input is by default a differential LVDS input. The device provides 100 Ω internal termination on the differential SYSREF input. This approach optimizes the termination by inserting it at the end of the route. It also reduces the routing complexity by allowing the traces to be routed directly to the device without any additional external components.

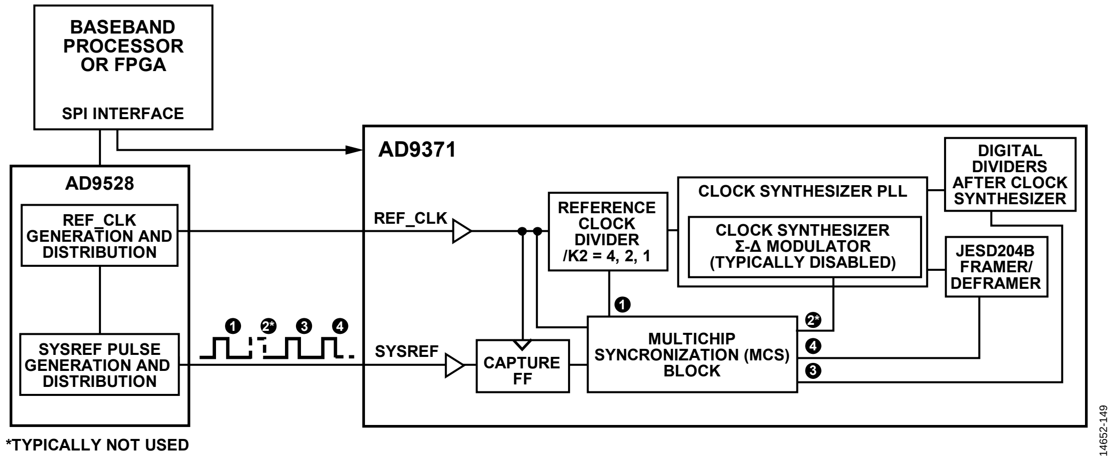

BASEBAND PROCESSOR OR FPGA
SPI INTERFACE
AD9528
REF_CLK GENERATION AND DISTRIBUTION
SYSREF PULSE GENERATION AND DISTRIBUTION
*TYPICALLY NOT USED
1 2* 3 4
AD9371
REF_CLK
SYSREF
CAPTURE FF
REFERENCE CLOCK DIVIDER /K2 = 4, 2, 1
CLOCK SYNTHESIZER PLL
CLOCK SYNTHESIZER Σ-Δ MODULATOR (TYPICALLY DISABLED)
DIGITAL DIVIDERS AFTER CLOCK SYNTHESIZER
JESD204B FRAMER/ DEFRAMER
MULTICHIP SYNCRONIZATION (MCS) BLOCK
14652-149

Figure 50. Use of Each SYSREF Pulse

Minimum Delay Requirements Between SYSREF Pulses

The first SYSREF pulse resets the device clock divider, which causes the clock phase-locked loop (PLL) to relock. There is a required wait period before any further SYSREF pulses are registered. The wait period is set to 1,024 phase frequency detector (PFD) reference clock periods. The PFD clock must always be less than 80 MHz. The following conditions exist for each option:

- If a device clock of 122.88 MHz or 245.76 MHz is used, it produces a PFD reference clock of 61.44 MHz, resulting in a minimum wait of 16.7 μs after the first SYSREF pulse before the next SYSREF pulse is recognized.
- If a device clock of 153.6 MHz or 307.2 MHz is used, it produces a PFD reference clock of 76.8 MHz, resulting in a minimum wait of 13.3 μs after the first SYSREF pulse, before the next SYSREF pulse is recognized.

Timing of SYSREF Compared to DEV_CLK

SYSREF can be either periodic signal, a one-shot (strobe type) pulse, or a gapped periodic signal. Ensure that no runt pulses or glitches result while turning the SYSREF signal on and off. SYSREF is an active high signal that is sampled by the device clock. SYSREF is latched internally by DEV_CLK; therefore, strictly adhered to the setup and hold times specified in the data sheet. Figure 51 and Figure 52 illustrate the idea of negative hold time due to the delay of the DEV_CLK signal vs. the SYSREF signal inside the device.

Figure 51 and Figure 52 illustrate the relationship between the SYSREF signal and the DEV_CLK signal. In the end application, ensure that the user generated SYSREF signal follows the recommendations.

In cases where periodic or gapped periodic SYSREF signals are used, the period must be an integer multiple of the local multiframe counter (LMFC) period. The LMFC and frame clock within a device must be phase aligned to the DEV_CLK sampling edge upon which the sampled SYSREF value transitions from 0 to 1.

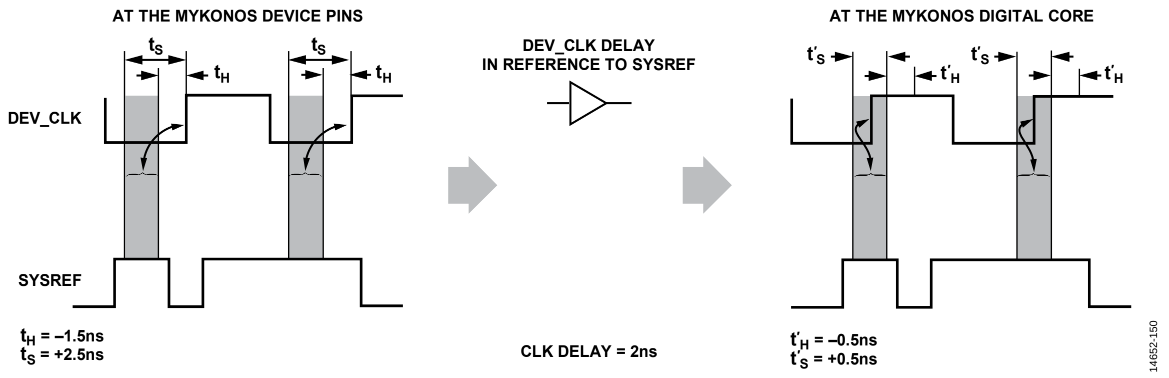


Figure 51. Timing Alignment of SYSFREF vs. DEV_CLK at the Device Pins

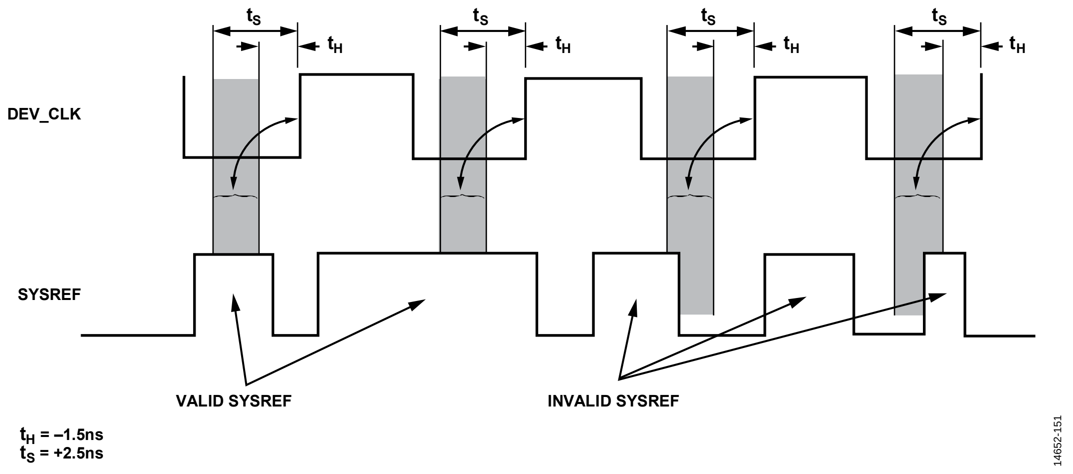


Figure 52. SYSREF Setup and Hold Timing with Examples of SYSREF Pulse

SYNTHESIZER CONFIGURATION

The device contains three radio frequency (RF) phased-locked loop (PLL) synthesizers for Tx, Rx, and ORx/sniffer channel tuning. Figure 53 shows these synthesizers and their interconnectivity with each of the RF signal paths. Each PLL synthesizer employs a fractional–N architecture with a completely integrated voltage controlled oscillator (VCO) and loop filter. No external devices are required to cover the entire frequency range of the device. This configuration allows the use of any convenient reference frequency for operation on any channel with any sample rate. The fundamental frequency of the PLL ranges from 6 GHz to 12 GHz. The local oscillator (LO) frequency is created by dividing down the PLL VCO frequency. The reference frequency for the PLL is scaled from the reference clock applied to the DEV_CLK_IN± pins.

The device also provides a clock synthesizer to generate all the clocking signals necessary to run the device. The reference frequency for the PLL is scaled from the reference clock applied to the chip DEV_CLK_IN± pins. Although it is a fractional–N architecture, note that the signal sampling relationships to the JESD204B interface rates typically require that the synthesizer operate in integer mode. Profiles that are included in the transceiver evaluation software (TES) configure the clock synthesizer appropriately. Reconfiguration of the clock synthesizer is typically not necessary after initialization. The most direct approach to the configuration is to follow the recommended programming sequence and use the provided application programming interface (API) functions to set the clock synthesizer to the desired mode of operation.

A calibration PLL (CALPLL) synthesizer is integrated into the device to generate the signals necessary to calibrate the device. The reference frequency for the CALPLL is scaled from the device clock applied to the DEV_CLK_IN± pins. The CALPLL output signal is injected into the input of the Rx signal path. This calibration is executed during the initialization sequence at startup. There must be no signal present at the Rx input during tone calibration time. Solely the internal ARM processor controls the CALPLL. This procedure is fully autonomous, and there is no user access to control the CALPLL state.

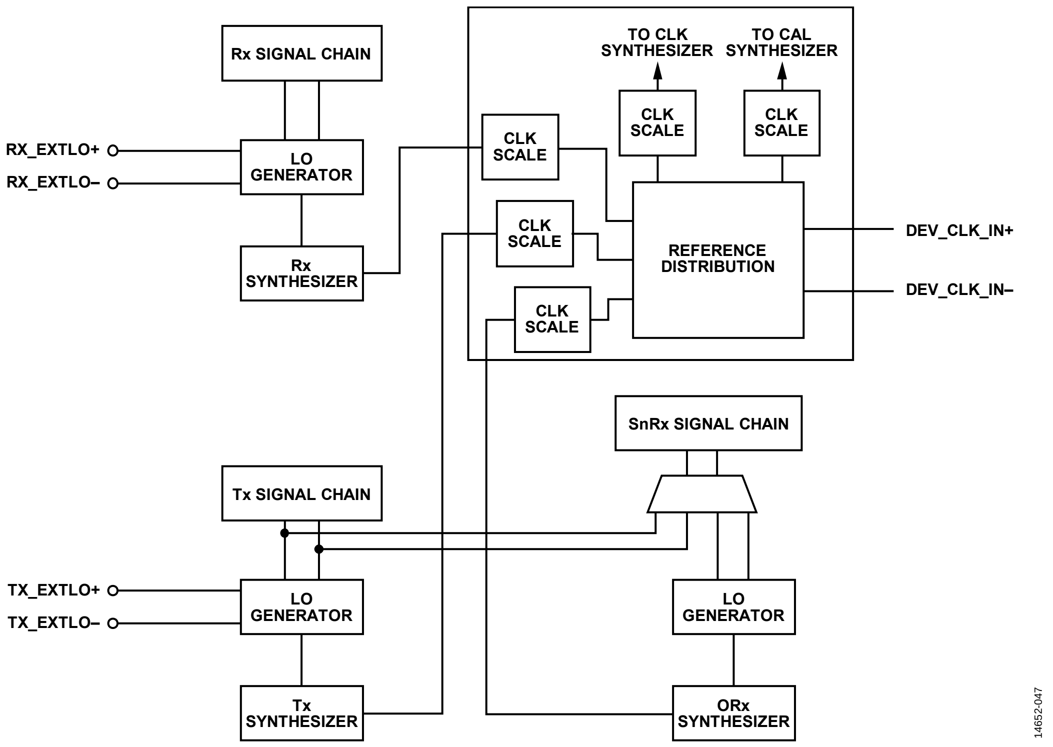


Figure 53. Synthesizer Interconnection Block Diagram

CONNECTIONS FOR EXTERNAL LO

The device provides the user with an option of using internal phase-locked loops (PLLs) to generate local oscillator (LO) frequencies or receiving the LO signals from two external LO sources. Unlike the internal synthesizers that always operate between 6 GHz to 12 GHz, regardless of the RF tune frequency, when an external LO is used, the frequency applied must be 2× the desired RF tune frequency. The signal is divided internally by 2 to generate the required LO quadrature relationship. The range of the external LO signal can be as low as 600 MHz and as high as 12 GHz, covering the RF tune frequency range from 300 MHz to 6 GHz.

The two separate differential external LO inputs follow:

- The external LO for the Rx signal chain uses the B7 (RX_EXTLO−) and B8 (RX_EXTLO+) balls.
- The external LO for the Tx signal chain use the E11 (TX_EXTLO−) and E12 (TX_EXTLO+) balls.

Both inputs present 100 Ω differential impedance. Differential signals applied to the external LO inputs must be ac-coupled. Place a 50 Ω termination resistor on each input line as close as possible to the external LO input balls. Figure 54 provides a high level overview of the recommended configuration.

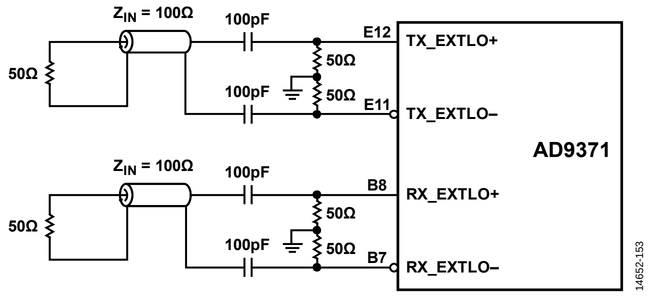


Figure 54. TX_EXTLO and RX_EXTLO Inputs

Higher external LO frequencies require higher input power. In general, higher input power produces better phase noise performance. To optimize system design, the minimum input power that results in phase noise meeting requirements (with some margin) must be used. If an on-board balun is used to connect a single-ended LO supply to the differential inputs, the loss of the balun must be taken into account when calculating input power. Table 95 describes the specifications for the RX_EXT_LO and TX_EXT_LO input pins. Note that operation is limited to LO frequencies lower than 4000 MHz. Higher frequencies require use of the internal LO generators.

Table 95. Specifications for RF External LO Inputs (RX_EXT_LO and TX_EXT_LO)

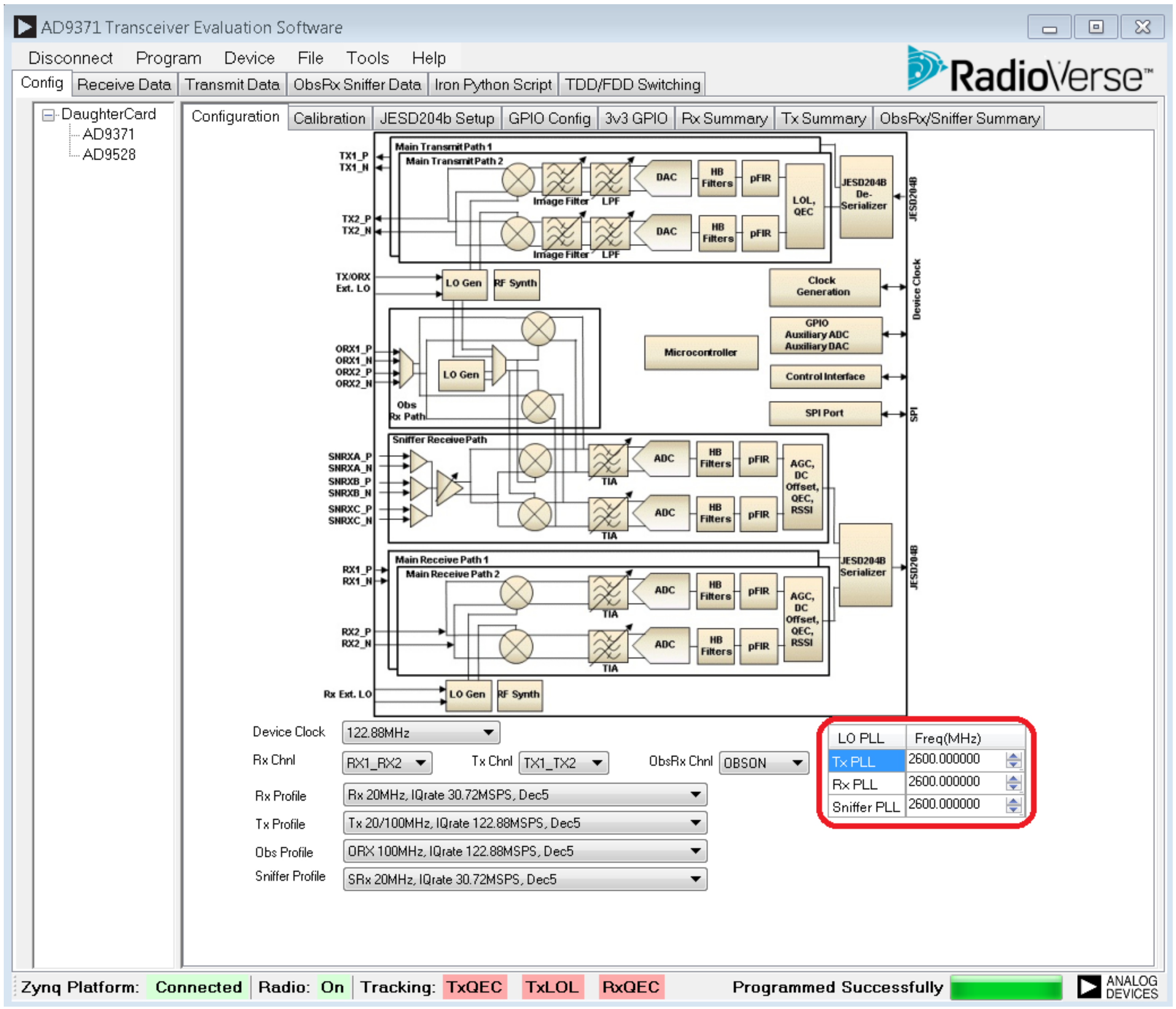
Parameter	Test Conditions/Comments	Min	Typ	Max	Unit
Input Frequency (f_{EXTLO})		600		8000	MHz
LO Frequency ($f_{CHANNEL}$)		300		4000	MHz
Input Power	50 Ω matching at the source. Signal amplitude depends on the f_{EXTLO} frequency. Typical = 3 dBm for $f_{EXTLO} \leq 2$ GHz.	0	3	6	dBm

Performance Limitations

Local oscillator (LO) quadrature phase error can be caused by phase imbalance in the external LO input differential signal; therefore, it is important to design the input circuit carefully to avoid mismatch between the two inputs that make up the differential pair for each external LO signal. Three main parameters can affect LO phase noise when the external LO input is used:

- Differential phase error (deviation from 180°)
- Differential amplitude error (deviation from equal amplitudes on each input pin)
- Duty cycle error (deviation from 50%)

The combination of these errors must not result in a LO quadrature phase noise error greater than 1.5 ps. At 1.8 GHz, this equates to approximately 1° of phase noise. Ensure that the external LO input signal has less than 10° of differential phase error, less than 1 dB of differential amplitude error, and a duty cycle between 49% and 51%.



Device Clock: 122.88MHz

Rx Chnl: RX1_RX2 Tx Chnl: TX1_TX2 ObsRx Chnl: OBSON

Rx Profile: Rx 20MHz, IQrate 30.72MSPS, Dec5

Tx Profile: Tx 20/100MHz, IQrate 122.88MSPS, Dec5

Obs Profile: ORX 100MHz, IQrate 122.88MSPS, Dec5

Sniffer Profile: SRx 20MHz, IQrate 30.72MSPS, Dec5

LO PLL	Freq(MHz)
Tx PLL	2600.000000
Rx PLL	2600.000000
Sniffer PLL	2600.000000

Zynq Platform: Connected | Radio: On | Tracking: TxQEC TxLOL RxQEC | Programmed Successfully

14652-048

Figure 55. Internal and External LO Configuration in the Transceiver Evaluation Software (TES)

SOFTWARE CONFIGURATION

Device configuration is dependent on the user application requirements. When using an external local oscillator (LO), use the transceiver evaluation software (TES) to generate initial values for application programming interface (API) structure members.

Figure 53 outlines a high level synthesizer block diagram, including the option to provide external LOs for Rx and Tx signal chains. To select an external LO for Rx or Tx RF signal paths, a series of API commands must be executed before initializing the device. Select the correct frequency settings by using the following commands:

```
static mykonosRxSettings_t  rxSettings =
      uint8_t rxPllUseExternalLo       /*
Internal LO = 0, external LO*2 = 1 */
      uint64_t rxPllLoFrequency_Hz     /*
Rx PLL LO Frequency (internal or external
LO/2) */
```

and

```
static mykonosTxSettings_t txSettings =
      uint8_t txPllUseExternalLo       /*
Internal LO = 0, external LO*2 if =1 */
      uint64_t txPllLoFrequency_Hz     /*
Tx PLL LO frequency (internal or external
LO/2) */
```

It is important to note that when an external LO is used, the value of the RF frequency must still be programmed for the Tx and Rx channels (see the red box in Figure 55) and the rxPllLoFrequency_Hz or rxPllLoFrequency_Hz structure members. For more details regarding the initialization procedure, refer to the System Initialization section.

Part of the initialization procedure includes setting up internal clock generation. All internal clocks are generated based on the selected profile; therefore, there is no need for reconfiguration of the clock synthesizer after the device finishes the initialization sequence. Initialization of the clock generation block (see Figure 53) is done by the API function described in the following section.

MYKONOS_initDigitalClocks

```
mykonosErr_t MYKONOS_initDigitalClocks (
   mykonosDevice_t *  device )
```

This function updates the clock synthesizer and loop filter settings based on a voltage controlled oscillator (VCO) frequency lookup table (LUT). The VCO frequency break points for the synthesizer LUT can be found in the vcoFreqArrayHz array. This function has no parameters, and there is no need for interaction with it from the user. This function is automatically called inside the main initialization application programming interface (API) function.

```
mykonosErr_t
   MYKONOS_initialize(mykonosDevice_t
   *device)
```

Another user configurable option shown in Figure 53 is the selection of the local oscillator (LO) for the observation Rx signal path. The user can select the desired LO source (SnRx, LO, or Tx LO) for ORx1 or ORx2 using the TES dropdown menu, as shown in Figure 56.

The same operation can be performed using the API function described in the following section.

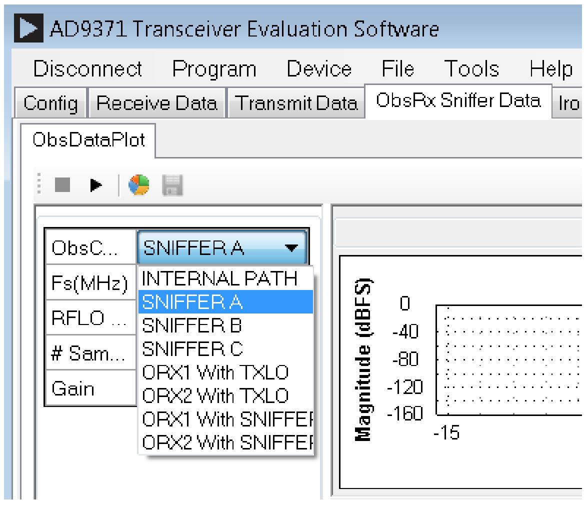


Figure 56. LO Selection for Sniffer and ORx Path in the Transceiver Evaluation Software (TES)

MYKONOS_setObsRxPathSource

```
mykonosErr_t MYKONOS_setObsRxPathSource (
   mykonosDevice_t * device,
   mykonosObsRxChannels_t obsRxCh )
```

When the ARM radio control is in ARM command mode, this function allows the user to selectively power up or power down the desired ObsRx datapath.

The value set in device → obsRx → obsRxChannel determines the mode of operation for the SnRx path. The user options are as follows:

- OBS_RXOFF: The SnRx path is disabled.
- OBS_RX1_TXLO: The SnRx operates in observation mode on ORx1 with the Tx LO synthesizer.
- OBS_RX2_TXLO: The SnRx operates in observation mode on ORx2 with the Tx local oscillator (LO) synthesizer.
- OBS_INTERNALCALS: This enables scheduled Tx calibrations while using SnRx path. The enableTrackingCals function must be called in the radio off state. It sets the calibration mask, which the scheduler uses later to schedule the desired calibrations. This command is issued in radio off. After the device moves to the radio on state, the internal scheduler uses the enabled calibration mask to schedule calibrations whenever possible, based on the state of the transceiver. The Tx calibrations are not be scheduled until OBS_INTERNALCALS is selected, and the Tx calibrations are enabled in the calibration mask.

- OBS_SNIFFER: The SnRx operates in sniffer mode with the latest selected sniffer input (for hardware pin control operation). In pin mode, the GPIO pins designated for ORX_MODE select sniffer mode. Then, the MYKONOS_ setSnifferChannel function chooses the channel.
- OBS_RX1_SNIFFERLO: The SnRx operates in observation mode on ORx1 with the sniffer LO synthesizer.
- OBS_RX2_SNIFFERLO: The SnRx operates in observation mode on ORx2 with the sniffer LO synthesizer.
- OBS_SNIFFER_A: The SnRx operates in sniffer mode on SnRxA with the sniffer LO synthesizer.
- OBS_SNIFFER_B: The SnRx operates in sniffer mode on SnRxB with the sniffer LO synthesizer.
- OBS_SNIFFER_C: The SnRx operates in sniffer mode on SnRxC with the sniffer LO synthesizer.

Note that if this function is called when the ARM is expecting the GPIO pin control of the ORx path source, an error returns.

To change the frequency of operation of each radio frequency local oscillator, use the application programming interface (API) function described in the following section.

MYKONOS_setRfPllFrequency

```
mykonosErr_t MYKONOS_setRfPllFrequency  (
   mykonosDevice_t * device,
   mykonosRfPllName_t  pllName,uint64_t
   rfPllLoFrequency_Hz )
```

This function sets the radio frequency (RF) phase-locked loop (PLL) local oscillator (LO) frequency (RF carrier frequency). This function has two parameters:

- pllName: This parameter designates the name of the PLL to configure, which includes the following:
  - RX_PLL changes the operating frequency of the Rx LO.
  - TX_PLL changes the operating frequency of the Tx LO.
  - SNIFFER_PLL changes the operating frequency of the sniffer LO.
- rfPllLoFrequency_Hz: This parameter designates the desired LO frequency in Hz.

To read back the value programmed for a particular PLL LO frequency, use the application programming interface (API) function described in the following section.

MYKONOS_getRfPllFrequency

```
mykonosErr_t MYKONOS_getRfPllFrequency  (
   mykonosDevice_t * device,
   mykonosRfPllName_t  pllName,uint64_t
   rfPllLoFrequency_Hz )
```

This function obtains the frequency of the radio frequency (RF) phase-locked loop (PLL). It can obtain the frequency for the Rx PLL, Tx PLL, sniffer PLL, and clock PLL.

This function has two parameters:

- pllName: This parameter designates the name of the PLL to configure, which includes the following.
  - RF_PLL: reads back the Rx LO operating frequency.
  - TX_PLL: reads back the Tx LO operating frequency.
  - SNIFFER_PLL: reads back the sniffer LO operating frequency.
  - CLKPLL: reads back the clock LO operating frequency.
- rfPllLoFrequency_Hz: This parameter is the LO frequency currently set for the specified PLL.

The application programming interface (API) function described in the following section checks if the lock detector bit for a particular PLL indicates that the corresponding synthesizer has achieved lock.

MYKONOS_checkPllsLockStatus

```
mykonosErr_t MYKONOS_checkPllsLockStatus  (
   mykonosDevice_t *  device,  uint8_t *
   pllLockStatus )
```

This function updates the pllLockStatus pointer with a lock status per phase-locked loop (PLL) according to the following assignments:

- pllLockStatus[0]: clock PLL locked
- pllLockStatus[1]: Rx PLL locked
- pllLockStatus[2]: Tx PLL locked
- pllLockStatus[3]: sniffer PLL locked
- pllLockStatus[4]: calibration PLL locked

The following is an example of how MYKONOS_ setRfPllFrequency and MYKONOS_checkPllsLockStatus can program the radio frequency (RF) PLL frequencies and check the PLL lock status bits. More information can be found in the System Initialization section.

```
/*******************************/
/**** Set RF PLL Frequencies ***/
/*******************************/
mykError = MYKONOS_setRfPllFrequency(&mykDevice, RX_PLL, mykDevice.rx->rxPllLoFrequency_Hz);
mykError = MYKONOS_setRfPllFrequency(&mykDevice, TX_PLL, mykDevice.tx->txPllLoFrequency_Hz);
mykError = MYKONOS_setRfPllFrequency(&mykDevice, SNIFFER_PLL, mykDevice.obsRx-
>snifferPllLoFrequency_Hz);

/*** < wait 200ms for PLLs to lock - user code here > ***/

mykError = MYKONOS_checkPllsLockStatus(&mykDevice, &pllLockStatus);
if ((pllLockStatus & 0x0F) == 0x0F)
{
       /*** < All PLLs locked - user code here > ***/
}
else
{
       /*** < PLLs not locked - ensure lock before proceeding - user code here > ***/
}
```

RF PLL FREQUENCY CHANGE PROCEDURE

Small Frequency Step Procedure

Use the following procedure when a radio frequency (RF) phased-lock loop (PLL) change is required with the following conditions:

- A desire to change the Rx, Tx, and ORx frequencies
- The frequency step change is less than 100 MHz.
- The frequency step does not cross the divide by 2 boundaries outlined in Table 96.

1. Move the device into the radio off state by executing the following command:

```
if ((mykError =
MYKONOS_radioOff(&mykDevice)) !=
MYKONOS_ERR_OK)
   {
/*** < Info: errorString will contain log
error string in order to debug failure >
***/
       errorString = getthe
AD9371ErrorMessage(mykError);
   }
```

2. Program the new local oscillator (LO) frequency. For example, set the Rx LO to 2550 MHz and the Tx LO to 2500 MHz by executing the following commands:

```
if ((mykError =
MYKONOS_setRfPllFrequency(&mykDevice,
RX_PLL, 2550000000)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
        errorString = getthe
AD9371ErrorMessage(mykError);
    }

if ((mykError =
MYKONOS_setRfPllFrequency(&mykDevice,
TX_PLL, 2500000000)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
          errorString = getthe
AD9371ErrorMessage(mykError);
    }

      /*** < Action: wait 200ms for PLLs
to lock > ***/

if ((mykError =
MYKONOS_checkPllsLockStatus(&mykDevice,
&pllLockStatus)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
          errorString = getthe
AD9371ErrorMessage(mykError);
    }
if ((pllLockStatus & 0x0F) == 0x07)
      {
      /*** < Info: Clock, Rx and Tx PLLs
locked > ***/
      }
      else
      {
      /*** < Info: Clock, Rx and Tx PLLs
not locked  > ***/
      /*** < Action: Ensure lock before
proceeding - User code here > ***/
      }
```

3. Reset the external channel by executing the following command:

```
If((mykError =
MYKONOS_resetExtTxLolChannel(&mykDevice,
TX1_TX2)) != MYKONOS_ERR_OK) ; //where
TX1_TX2 is part of the enum
mykonosTxChannels_t
{
/*** < Info: errorString will contain log
error string in order to debug failure >
***/
          errorString = getthe
AD9371ErrorMessage(mykError);
}
```

4. Move the device back into the radio on state by executing the following command:

```
if ((mykError =
MYKONOS_radioOn(&mykDevice)) !=
MYKONOS_ERR_OK)
   {
/*** < Info: errorString will contain log
error string in order to debug failure >
***/
        errorString = getthe
AD9371ErrorMessage(mykError);
   }
```

5. Some tracking calibrations are active only when the ORx path is set to the internal calibrations input. The user can reenable tracking calibrations by selecting the internal calibrations ORx path by executing the following command when using application programming interface (API) commands to control the ORx input:

```
if ((mykError =
MYKONOS_setObsRxPathSource(&mykDevice,
OBS_INTERNALCALS)) != MYKONOS_ERR_OK)
      {
          /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
        errorString = getthe
AD9371ErrorMessage(mykError);
      }
```

If the ORx input is controlled by the GPIO interface, the baseband processor (BBP) must configure its pins to select INTERNALCALS. Refer to the ARM GPIOs section and the Quadrature Error Correction, Calibration, and ARM Configuration section for more information.

Large Frequency Step Procedure

Use the following procedure if a radio frequency (RF) phased-lock loop (PLL) change is required with the following conditions:

- A desire to change the Rx, Tx, and ORx frequencies.
- The frequency step change is more than 100 MHz.
- The frequency step does cross the divide by 2 boundaries outlined in Table 96.

Use the following procedure:

1. Move the device into the radio off state by executing the following command:

```
   if ((mykError =
MYKONOS_radioOff(&mykDevice)) !=
MYKONOS_ERR_OK)
   {
       /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
        errorString = getthe
AD9371ErrorMessage(mykError);
   }
```

2. Program the new local oscillator (LO) frequency. For example, set the Rx LO to 2550 MHz and the Tx LO to 2500 MHz by executing the following commands:

```
if ((mykError =
MYKONOS_setRfPllFrequency(&mykDevice,
RX_PLL, 2550000000)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
        errorString = getthe
AD9371ErrorMessage(mykError);
    }

if ((mykError =
MYKONOS_setRfPllFrequency(&mykDevice,
TX_PLL, 2500000000)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
          errorString = getthe
AD9371ErrorMessage(mykError);
    }

      /*** < Action: wait 200ms for PLLs
to lock > ***/

if ((mykError =
MYKONOS_checkPllsLockStatus(&mykDevice,
&pllLockStatus)) != MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
          errorString = getthe
AD9371ErrorMessage(mykError);
    }

if ((pllLockStatus & 0x0F) == 0x07)
      {
      /*** < Info: Clock, Rx and Tx PLLs
locked > ***/
      }
      else
      {
      /*** < Info: Clock, Rx and Tx PLLs
not locked  > ***/
      /*** < Action: Ensure lock before
proceeding - User code here > ***/
      }
```

3. Rerun the initialization calibrations by executing the following set of commands:

```
uint32_t initCalMask = TX_QEC_INIT |
LOOPBACK_RX_LO_DELAY |
LOOPBACK_RX_RX_QEC_INIT |
       RX_LO_DELAY | RX_QEC_INIT;

                if ((mykError =
MYKONOS_runInitCals(&mykDevice,
initCalMask)) != MYKONOS_ERR_OK)
      {
          /*** < Info: errorString
will contain log error string in order
to debug failure > ***/
          errorString = getthe
AD9371ErrorMessage(mykError);
      }

      if ((mykError =
MYKONOS_waitInitCals(&mykDevice,
60000, &errorFlag, &errorCode)) !=
MYKONOS_ERR_OK)
      {
          /*** < Info: errorString
will contain log error string in order
to debug failure > ***/
          errorString = getthe
AD9371ErrorMessage(mykError);
      }

      if ((errorFlag != 0) ||
(errorCode != 0))
      {
            if((mykError =
MYKONOS_getInitCalStatus(&mykDevice,
&initCalStatus)) != MYKONOS_ERR_OK)
            {
                  /*** < Info:
errorString will contain log error
string in order to debug failure >
***/
                errorString = getthe
AD9371ErrorMessage(mykError);
            }

            /*** < Info: abort init
cals > ***/
            if((mykError =
MYKONOS_abortInitCals(&mykDevice,
&initCalsCompleted)) !=
MYKONOS_ERR_OK)
            {
                  /*** < Info:
errorString will contain log error
string in order to debug failure >
***/
                errorString = getthe
AD9371ErrorMessage(mykError);
            }
            if(initCalsCompleted)
            {
                /*** < Info: which
calls had completed, per the mask >
***/
            }

            if((mykError =
MYKONOS_readArmCmdStatus(&mykDevice,
&errorWord, &statusWord)) !=
MYKONOS_ERR_OK)
            {
                  /*** < Info:
errorString will contain log error
string in order to debug failure >
***/
                errorString = getthe
AD9371ErrorMessage(mykError);
            }

            if((mykError =
MYKONOS_readArmCmdStatusByte(&mykDevic
e, 2, &status)) != MYKONOS_ERR_OK)
        {
                /*** < Info:
errorString will contain log error
string in order to debug why  failed >
***/
                errorString = getthe
AD9371ErrorMessage(mykError);
        }
            if(status!=0)
            {
                /*** < Info: Arm
Mailbox Status Error errorWord > ***/
               /*** < Info: Pending
Flag per opcode statusWord, this
follows the mask > ***/
            }
      }
      else
      {
          /*** < Info: Calibrations
completed successfully  > ***/
      }
```

4. Move the device back into the radio on state by executing the following command:

```
   if ((mykError =
MYKONOS_radioOn(&mykDevice)) !=
MYKONOS_ERR_OK)
   {
       /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
        errorString = getthe
AD9371ErrorMessage(mykError);
   }
```

5. Some tracking calibrations are active only when the ORx path is set to the internal calibrations input. The user can reenable tracking calibrations by selecting internal calibrations ORx path by executing the following command when using application programming interface (API) commands to control the ORx input:

```
if ((mykError =
MYKONOS_setObsRxPathSource(&mykDevice,
OBS_INTERNALCALS)) != MYKONOS_ERR_OK)
      {
          /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
        errorString = getthe
AD9371ErrorMessage(mykError);
      }
```

If the ORx input is controlled by the GPIO interface, the baseband processor (BBP) must configure its pins to select INTERNALCALS. Refer to the ARM GPIOs section and the Quadrature Error Correction, Calibration, and ARM Configuration section for more information.

Sniffer Receiver PLL Procedure

The sniffer phased-lock loop (PLL) can also be configured when in the radio off state or when in the radio on state if the sniffer PLL is not in use. This is the case when the obsRxCh parameter of the MYKONOS_setObsRxPathSource() function is set to OBS_RXOFF, OBS_RX1_TXLO, or OBS_RX2_TXLO. To change the sniffer frequency, use the following procedure:

1. Move the device into the radio off state by executing the following command:

```
   if ((mykError =
MYKONOS_radioOff(&mykDevice)) !=
MYKONOS_ERR_OK)
   {
       /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
        errorString = getthe
AD9371ErrorMessage(mykError);
   }
```

2. Program the new sniffer local oscillator (LO)frequency. For example, set the sniffer LO to 2600 MHz by executing the following commands:

```
      if ((mykError =
MYKONOS_setRfPllFrequency(&mykDevice,
SNIFFER_PLL, 2600000000)) !=
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
          errorString = getthe
AD9371ErrorMessage(mykError);
    }


      if ((pllLockStatus & 0x0F) == 0x08)
      {
          /*** < Info: Sniffer PLL locked
> ***/
      }
      else
      {
          /*** < Info: Sniffer PLL not
locked  > ***/
          /*** < Action: Ensure lock
before proceeding - User code here > ***/
      }
```

3. Move the device back into the radio on state by executing the following command:

```
if ((mykError =
MYKONOS_radioOn(&mykDevice)) !=
MYKONOS_ERR_OK)
   {
       /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
        errorString = getthe
AD9371ErrorMessage(mykError);
   }
```

Alternatively, do the following:

4. While the device is operating in radio on mode, change the ORx path LO to Tx LO, or disable it by executing any of following commands listed:

```
if ((mykError =
MYKONOS_setObsRxPathSource(&mykDevice,
OBS_RXOFF)) != MYKONOS_ERR_OK)
      {
          /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
        errorString = getthe
AD9371ErrorMessage(mykError);
      }

or

if ((mykError =
MYKONOS_setObsRxPathSource(&mykDevice,
RX1_TXLO)) != MYKONOS_ERR_OK)
      {
          /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
        errorString = getthe
AD9371ErrorMessage(mykError);
      }

Or

if ((mykError =
MYKONOS_setObsRxPathSource(&mykDevice,
RX2_TXLO)) != MYKONOS_ERR_OK)
      {
          /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
        errorString = getthe
AD9371ErrorMessage(mykError);
      }
```

5. Program the new sniffer local oscillator (LO) frequency. For example, set the sniffer LO to 2600 MHz by executing the following commands:

```
if ((mykError =
MYKONOS_setRfPllFrequency(&mykDevice,
SNIFFER_PLL, mykDevice.obsRx-
>snifferPllLoFrequency_Hz)) !=
MYKONOS_ERR_OK)
    {
        /*** < Info: errorString will
contain log error string in order to
debug failure > ***/
          errorString = getthe
AD9371ErrorMessage(mykError);
    }
      if ((pllLockStatus & 0x0F) == 0x08)
      {
          /*** < Info: Sniffer PLL locked
> ***/
      }
      else
      {
          /*** < Info: Sniffer PLL not
locked  > ***/
          /*** < Action: Ensure lock
before proceeding - User code here > ***/
      }
```

Table 96. Divide by 2 Boundaries vs. Desired LO Frequency

	LO Frequency Limits (MHz)							
	Lower Limit	**Upper Limit**	**Lower Limit**	**Upper Limit**	**Lower Limit**	**Upper Limit**	**Lower Limit**	**Upper Limit**
	400	750	750	1500	1500	3000	3000	6000
Divide by	16		8		4		2	

RF PLL RESOLUTION LIMITATIONS

The MYKONOS_setRfPllFrequency command and the MYKONOS_getRfPllFrequency command both operate with 1 Hz resolution. The real frequency to which the Rx PLL, Tx PLL, and SnRx PLL are tuned can vary by a small amount depending on the region of operation. Table 97 outlines the Rx, Tx, and SnRx PLL frequency step variations vs. the operating band LO frequency. Note that the upper limit is noninclusive; if at the limit, use the next step size where the limit is lower.

See the following examples to use Table 97 to determine the correct local oscillator (LO) frequency setting.

Example 1

The DEV_CLK_IN± input is 153.6 MHz. The user wants to tune the Tx LO to a frequency equal to 3,600,000,002 Hz. The local oscillator (LO) step size for this range is 4.578754579 Hz. The count number for the code required to obtain this frequency is as follows:

Count = (3600000002/4.578754579) = 786240000.395 →

round to 786,240,000

Actual Tx LO Frequency = (786240000) × (4.578754579) = 3600000000.2 Hz

Example 2

The DEV_CLK_IN± input is 245.76 MHz. The user wants to tune the Tx LO to a frequency equal to 5,000,000,002 Hz. The LO step size for this range is 3.663003663 Hz. The count number for the code required to obtain this frequency is as follows:

Count = (5000000002/3.663003663) = 1365000000.5473 →

round to 1365000001

Actual Tx LO Frequency = (1365000001) × (3.663003663) = 5000000003.658 Hz

Example 3

The DEV_CLK_IN± input is 245.76 MHz. The user wants to tune the Tx LO to a frequency equal to 400,000,001 Hz. The LO step size for this range is 0.457875458 Hz. The count number for the code required to obtain this frequency is as follows:

Count = (400000001/0.457875458) = 873600002.184 →

round to 873 600 002

Actual Tx LO Frequency = (873600002) × (0.457875458) = 400000000.9 Hz.

Table 97. LO Steps Size vs. Desired LO Frequencies

		Desired LO Frequency Ranges (MHz)							
		Lower Limit	Upper Limit	Lower Limit	Upper Limit	Lower Limit	Upper Limit	Lower Limit	Upper Limit
	DEV_CLK_IN± (MHz)	400	750	750	1500	1500	3000	3000	6000
LO Step Size (Hz)	153.6 307.2	0.572344322		1.144688645		2.289377289		4.578754579	
	122.88 245.76	0.457875458		0.9157510916		1.831501832		3.663003663	

GAIN CONTROL

The device main receivers (Rx1 and Rx2) and sniffer receivers (SnRxA, SnRxB, and SnRxC) feature automatic and manual gain control modes that provide flexible gain control in a wide array of applications. The observation receivers (ORx1 and ORx2) feature manual gain control (MGC) only. Automatic gain control (AGC) allows the receivers to autonomously adjust the receiver gain depending on variations of the input signal, such as the onset of a strong interferer overloading the receiver datapath. All the receivers are also capable of operating in MGC mode where changes in gain are initiated by the baseband processor (BBP) over the SPI or the GPIO control mode. The gain control blocks are configured by the application programming interface (API) data structures, and several API commands exist to allow user interaction with the gain control mechanisms.

This section begins by explaining the variable gain elements in the receiver datapaths, the structure of the gain tables, and how to develop and program custom gain tables. This information is followed by a description of the AGC peak detectors, overload detectors, and power measurement detectors to provide insight into the configurable settings of the AGC engine. Following the receiver gain control programming descriptions, the remaining sections examine the gain compensation methods available in the device (slicer/floating point formatter). Details of the API commands and data structures are provided throughout this section.

VARIABLE GAIN ELEMENTS IN THE RECEIVER DATAPATHS

Gain Control Block Diagram Overview

The receivers have several variable gain elements within their datapaths. For the Rx and ORx datapaths, the variable gain stages include an internal RF attenuator, an (optional) external RF attenuator, and a digital gain/attenuation block. The external attenuator is an optional stage outside of the device that can be controlled by using the GPIO pins. An example of an external attenuator is a digital step attenuator (DSA). The datapath for the Rx channel is shown in Figure 57.

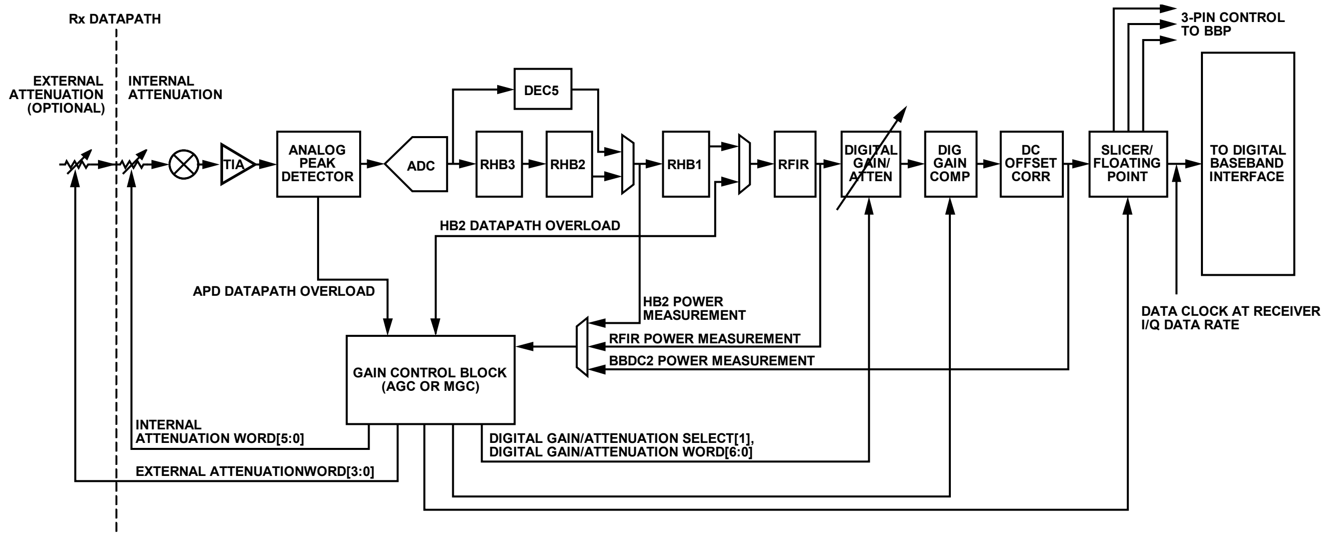


Figure 57. Rx Datapath, Highlighting the Gain Control Block and the Variable Gain Elements (Not the Complete Datapath)

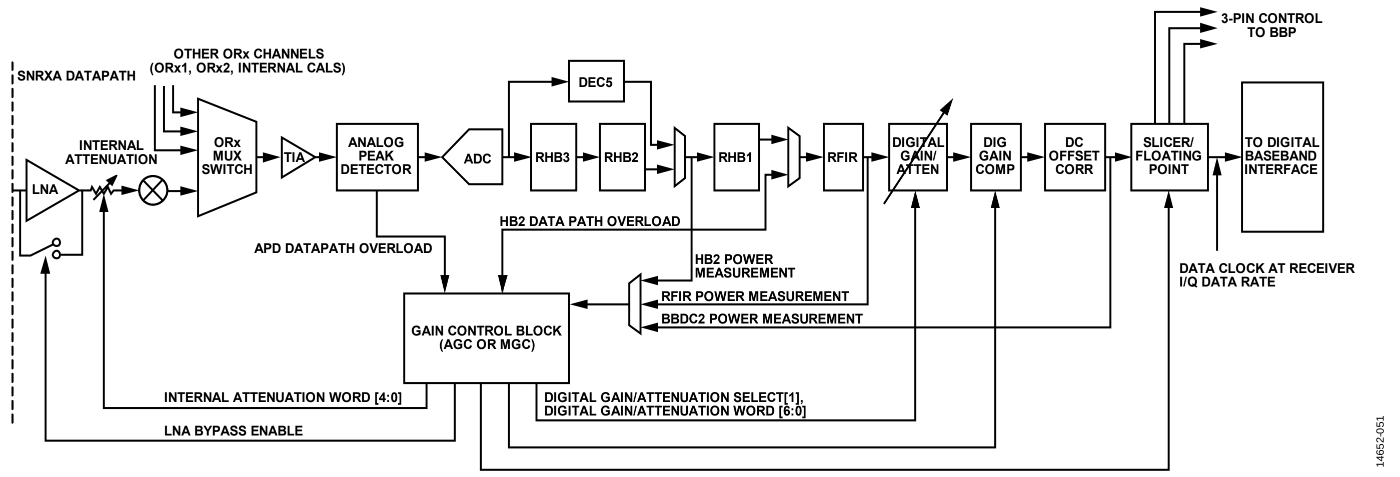


Figure 58. SNRXA Datapath, Highlighting the Gain Control Block and the Variable Gain Elements (Not the Complete Datapath)

The variable gain elements (the external attenuator control, internal attenuator, and digital gain/attenuator) in the Rx datapath are also present in the ORx datapath. However, AGC is not supported on the ORx channels.

The variable gain elements in the SnRx datapath are similar to those of the Rx/ORx paths, excluding the low noise amplifier (LNA) at the front end of each sniffer channel. The LNA can be bypassed to reduce the front-end gain, if desired. Other variable gain elements in the SnRx datapath include the internal attenuator and the digital gain and attenuation stage. There is no external attenuator control for the SnRx input.

The datapath for the SNRXA input is shown in Figure 58.

Note that the ORx1, ORx2, SNRXA, SNRXB, SNRXC, and internal loopback (OBS_INTERNALCALS) paths share a common baseband datapath. The mixer and internal attenuator for SNRXA, SNRXB, and SNRXC are shared but have separate front-end LNAs.

Internal Attenuator

The ORx and Rx internal attenuators each have a 6-bit control word in the first column of the gain table (see Table 98 for the gain table format). The internal attenuator has 64 attenuation settings in the ORx and Rx datapaths. The valid range of internal attenuator index values is 0 to 63. The amount of attenuation provided by the internal attenuator depends on the value set in the internal attenuator column of the gain table. The maximum gain condition for the internal attenuator is met when the internal attenuator word is set to 0.

Increasing values of the attenuator corresponds to an increase of attenuation. Equation 3 relates the 6-bit internal attenuator word to the internal attenuator attenuation in the ORx and Rx channels.

$$Attenuation = 20\log((64 - N)/64) \quad (3)$$

where N is a 6-bit value.

The SnRx internal attenuator is similar to the ORx and Rx internal attenuators. However, the valid range of internal attenuator index values extends from 0 to 19 (5-bit control) on the SnRx. The maximum gain condition for the internal attenuator in the SnRx is met when the internal attenuator word is set to 0.

External Attenuator

The device can control an external DSA with the GPIO pins. A 4-bit word set in the second column of the gain table controls an external attenuator. If an external attenuator is not used, zeros may be set in the external attenuator column of the gain table.

The 4-bit external attenuator control word is output on the 3.3 V GPIO pins. Rx1 uses GPIO_3P3_3 to GPIO_3P3_0 and Rx2 uses GPIO_3P3_7 to GPIO_3P3_4. The ORx channels use GPIO_3P3_11 to GPIO_3P3_7. The external attenuator requires the 3.3 V GPIO source control to be set to the GPIO3V3_EXTATTEN_LUT_MODE parameter and that the 3.3 V GPIO pins are set to output mode.

External attenuator control is available on the Rx and ORx ports only.

Digital Gain/Attenuation

The digital gain/attenuation block allows finer resolution gain or attenuation adjustments than the internal RF attenuator, resulting in a finer level of receiver datapath attenuation or gain control than the internal attenuator word, which attempts to compensate for unequal analog gain steps.

The digital gain/attenuation block acts as an attenuator if the digital attenuation enable holds a 1. The digital gain/attenuation word corresponds to 0.05 dB/LSB of digital attenuation in attenuation mode.

The digital gain/attenuation block provides digital gain if the digital attenuation enable holds 0. The digital gain/attenuation word corresponds to 0.25 dB/LSB of digital gain in gain mode.

LNA Bypass Enable

The SnRx channel features an integrated LNA at its input. The SnRx features an LNA bypass mode. The LNA bypass is enabled for a particular gain index if the second column of the a SnRx gain table row holds a 1. LNA bypass is disabled when the second column of the selected SnRx gain table row holds a 0. LNAs are not present on the ORx and Rx channels.

GAIN TABLE FORMAT

The device gain control block, in AGC or MGC, points to a gain index row in the receiver gain table and programs settings for the variable gain elements in the receiver datapath. Separate gain tables can be implemented for the Rx1, Rx2, ORx, and SnRx channels.

The default gain tables can be found in the **mykonos_user.c** file. The **mykonos_user.c** and **mykonos_user.h** files can be customized to modify, add, or delete the default gain table settings. Consult Analog Devices applications engineering prior to changing the gain tables.

The Rx1 and Rx2 channels can operate simultaneously. Rx1 and Rx2 may use separate gain tables and point to different gain index values within those tables. In the default gain tables, Rx1 and Rx2 use the same gain table. The Rx1 and Rx2 gain tables support up to 255 gain index rows.

The observation receiver system (ORx, including the ORx1, ORx2, SNRXA, SNRXB, and SNRXC receivers) uses a common baseband datapath; only one ORx front end can connect to the common baseband at any time. The gain index set or readback for the ORx, therefore, applies only to the selected input. The ORx gain table supports up to 47 gain index rows. The SnRx gain table supports up to 127 gain index rows.

Rx and ORx Gain Table

The format of the columns in the Rx and ORx gain table rows are as follows:

- Internal attenuator word
- External attenuator word
- Digital gain/attenuation word
- Digital attenuation enable

An example of the gain table structure is provided in Table 98 for the Rx gain table. Because the Rx and ORx datapaths have the same variable gain elements, they have the same gain table column format.

The first row of all gain tables corresponds to Gain Index 255 and the successive rows correspond to Gain Index 254, Gain Index 253, and so on. In the default gain tables, the first gain index in the table, Gain Index 255, is the maximum gain condition. The first column in Table 98 indicates the gain index. The two right most columns provide information regarding the attenuation level relative to maximum gain condition (defined in the default tables as Gain Index 255) and the attenuation level relative to the previous gain index.

Table 98. Sample Elements From the Default Rx Gain Table

Gain Table Index	Internal Attenuator[5:0][1]	External Attenuator[3:0][1]	Digital Gain/ Attenuation[6:0][1]	Digital Attenuation Enable[1]	Attenuation in dB (Relative to Maximum Gain)	Difference (dB) (Step Size = 1)
255	0	0	0	0	0.00	Not applicable
254	3	0	2	1	−0.52	0.52
253	6	0	3	1	−1.01	0.49
252	10	0	0	0	−1.47	0.47

[1] Only the elements shown in these columns are programmed to the device registers.

The following example demonstrates the calculations involved in Gain Table Index 253:

$$Atten(GainIndex) = A_{IntAtten}(IntAtten[5{:}0]) + A_{extAtten}(ExtAtten[3{:}0]) + A_{digAttenGain}(digAttenEn, digGainAtten[6{:}0])$$

$$Atten(253) = A_{IntAtten}(IntAtten[5{:}0]) + A_{digAttenGain}(1, 3)$$

$$Atten(253) = 20\log_{10}((64 - 6)/64) + A_{digAttenGain}(1, 3)$$

$$Atten(253) = -0.855 + (-1) \times (0.05)$$

$$Atten(253) = -0.855 - 0.15 = -1.01 \text{ dB}$$

The following code excerpt from **mykonos_user.c** shows the implementation of the Rx gain table:

```
 /**
 * \file mykonos_user.c
 * \brief Contains the AD9371 default gain table values for Rx, ObsRx, and SnRx
 */

#include <stdint.h>
#include "t_mykonos.h"
#include "mykonos_user.h"

/**
 * \brief Default Rx gain table settings
 */
uint8_t RxGainTable [61][4] =
{
       /* Order: {FE table, External Ctl, Digital Gain/Atten, Enable Atten} */
        {0, 0, 0, 0},  /* Gain index 255 */
        {3, 0, 2, 1},  /* Gain index 254 */
        {6, 0, 3, 1},  /* Gain index 253 */
        {10, 0, 0, 0}, /* Gain index 252 */
        {13, 0, 1, 1}, /* Gain index 251 */
        {16, 0, 0, 0}, /* Gain index 250 */
        …
```

For example, when the device sets the Rx1 gain index to gain index of 254, the values in the row corresponding to Gain Index 254 are programmed into the device registers corresponding the four columns within that row. For the default Rx gain tables, this corresponds to a 0.5 dB decrease in receiver gain compared to the gain condition in Gain Index 255 (maximum gain).

Table 99. Sample Rows from the Default SnRx Gain Table

Gain Table Index	Internal Attenuator[4:0][1]	LNA Bypass[1]	Digital Gain/ Attenuation [6:0][1]	Digital Attenuation Enable[1]	Attenuation in dB (Relative to Maximum Gain)	Difference (dB) (Step Size = 1)
255	0	0	0	0	0	Not applicable
254	1	0	7	1	−0.99	0.99
253	3	0	1	0	−1.97	0.98
252	3	0	15	1	−2.97	1.00

[1] Only the elements shown in these columns are programmed to the device registers.

SnRx Gain Table

The format of the columns in the SnRx gain table rows are as follows:

- Internal attenuator word
- LNA bypass enable
- Digital gain/attenuation word
- Digital attenuation enable

A different gain table column format is used in the SnRx gain table because the variable gain elements are different in the SnRx datapath relative to the Rx/ORx datapath. The SnRx gain table includes a column representing the LNA bypass bit. LNA bypass is activated when these bits are equal to 1. The SnRx gain table does not allow external attenuator control. When not using the LNA at all during SnRx operation, set the LNA bypass column to 1 for all rows.

Custom Gain Tables

The default gain tables can be found in the **mykonos_user.c** file. The **mykonos_user.c** and **mykonos_user.h** files can be customized to modify, add, or delete the default gain table settings. Consult Analog Devices applications engineering prior to changing gain tables.

In the **mykonos_user.c** file, receiver gain tables can be modified for the intended application. In the default gain tables, the gain step size between neighboring gain indices for the Rx channel are 0.5 dB, 1 dB for the ORx channel, and 1 dB for the SnRx channel. These tables have a gain range extending to 30 dB, 18 dB, and 52 dB, respectively. In the default gain tables, the maximum gain condition for all the tables is the first index in the gain table, which corresponds to Gain Index 255.

To set a target gain for the Rx datapath, the device allows the user to configure the maximum gain index (independently for both Rx1 and Rx2 channels) such that any gain index value can be configured to be the maximum gain index. For example, if the target gain for the Rx subsystem is 40 dB, but the factory calibration returns a total gain of 42 dB, it is possible to configure the target gain appropriately by changing the maximum gain index from 255 to 251 (−2 dB).

There are two ways to change the default gain tables:

1. Modify the **mykonos_user.c** and **mykonos_user.h** files with valid settings and gain tables. Note that gain tables are programmed to device registers when the MYKONOS_ initArm(…) command is called in the initialization sequence.
2. Perform gain table programming during or after the initialization sequence provided in the **headless.c** file. After verifying the ARM is loaded properly (executing MYKONOS_verifyArmChecksum(…) without returning an error), custom gain tables can be written by using the following application programming interface (API) function:

```
mykonosErr_t
    MYKONOS_programRxGainTable(mykonosDevi
    ce_t* device, uint8_t* gainTablePtr,
    uint8_t numGainIndexesInTable,
    mykonosGainTable_t rxChannel)
```

   This function takes a pointer to a 4 × N array, the value N, and the channel with the gain table to be overwritten. The N variable specifies the number of rows of the new gain table. This function can write gain tables for the Rx1, Rx2, Rx1 channels and the Rx2, ORx, and SnRx channels. Note that Rx1 and Rx2 may have separate gain tables.

GAIN TABLE PROGRAMMING DESCRIPTION

The MYKONOS_programRxGainTable(…) command is demonstrated in the following section.

MYKONOS_programRxGainTable (…)

```
mykonosErr_t
   MYKONOS_programRxGainTable(mykonosDevice_t
   *device, uint8_t *gainTablePtr, uint8_t
   numGainIndexesInTable, mykonosGainTable_t
   rxChannel)
```

Description

This command programs the gain table settings for either the Rx1, Rx2, Rx1 and Rx2, ORx, or SnRx receiver types. This command is called during MYKONOS_initArm(…). It is not necessary to call this function after initialization.

The gain table for a receiver type is set with the parameters passed by the uint8_t gainTablePtr array. gainTablePtr is a 4 × n array, where there are four elements per index, and the array length (n) is dependent upon receiver type. The (n) value is conveyed by numGainIndexesInTable. All gain tables have a maximum index of 255 when used with this function. The minimum gain index is application dependent.

For Rx1, Rx2, and ORx (A, B, C, or D), A indicates front end Gain, B indicates external control, C indicates digital attenuation/gain, and D indicates attenuation/gain select.

For SnRx (A, B, C, or D), A indicates front end gain, B indicates LNA bypass, C indicates digital attenuation/gain, and D indicates attenuation/gain select.

The gain table starting address changes with each receiver type. This function accounts for this change as well as the difference between each byte for the Rx1, Rx2, ORx, and SnRx receiver array values and programs the correct registers.

Preconditions

The MYKONOS_programRxGainTable(…) command does not need to be called by user if following headless.c instructions.

Parameters

- *device: A pointer to the device data structure.
- *gainTablePtr: A pointer to a 4 × n array containing gain table values.
- numGainIndexesInTable: The number of n indices in a 4 × n array. A range check is performed to ensure the maximum is not exceeded.
- rxChannel: mykonosGainTable_t enumeration type to select either the Rx1, Rx2, Rx1 and Rx2, ORx, or SnRx gain table for programming. A channel check is performed to ensure a valid selection.

For all custom gain tables, note that the maximum number of rows in the gain table is limited to 255 for the Rx1 and Rx2 channels, 127 for the SnRx channels, and 47 for the ORx channels.

Rx GAIN DELAY

Within the signal path, there are a few variable gain elements, such as the RF internal attenuator, the digital gain/attenuation block, and the digital gain compensation clock. Consider the case of a gain index decrement where new values for the internal attenuator and digital gain/attenuation are set, ignoring the gain compensation and the optional external attenuator for the moment. If the new settings for the internal attenuator and digital gain/attenuation block are applied at the same time, data in the Rx datapath between the internal attenuator and the digital gain/attenuation block experience analog gain of the old gain index and digital gain of the new gain index when the data propagates through the digital gain/attenuation block. In the baseband data, this appears as a double gain step. This scenario is shown Figure 59.

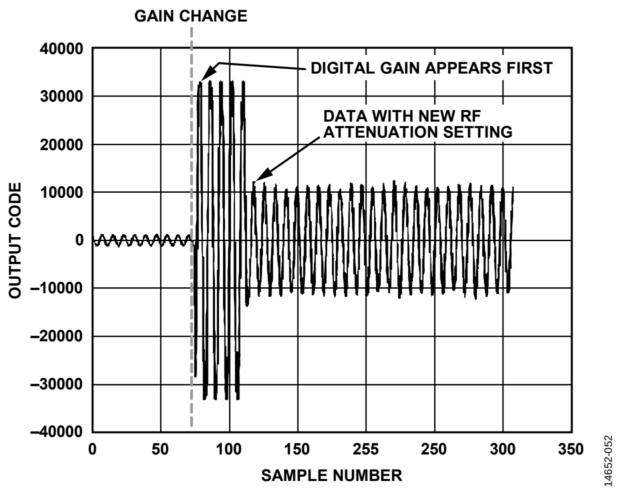


Figure 59. Double Gain Step Observed in Baseband Data Due to Insufficient Digital Gain Delay

The Rx gain delay calibration (ARM calibration) alleviates this double gain step by calculating a delay value for the onset of digital gain/attenuation. A successful calibration makes it appear to the baseband processor that only one gain change has been made, as shown in Figure 60. The calibration depends on the datapath configuration.

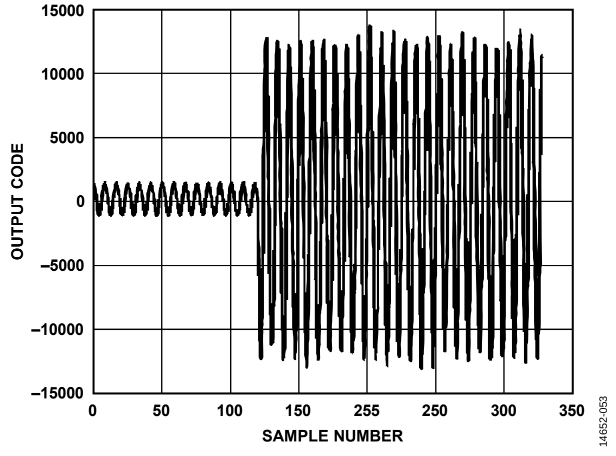


Figure 60. Single Gain Step with Proper Setting for Digital Gain Delay

Consider now the digital gain compensation block. This block is described in the Digital Gain Compensation, Slicer, and Floating Point Formatter section. The digital gain compensation block applies digital gain based on the current gain index. The timing of the onset of digital gain compensation must be delayed to prevent a double gain step. This value is calculated and programmed to the device during the Rx gain delay calibration as well.

The device is unable to delay the onset of external attenuator control word over the GPIO 3.3 V pins. Delays are only available for the digital gain/attenuator block and the digital gain compensation block.

MANUAL GAIN CONTROL, HYBRID MODE, AND AUTOMATIC GAIN CONTROL OVERVIEW

The receiver supports three modes of gain control. These modes are described in brief as follows:

- Manual gain control (MGC). MGC provides the user full control over the current gain index. In MGC mode, the gain index can be controlled by the application programming interface (API) commands and through GPIO signaling.
- Hybrid mode. When the user sends a pulse to a GPIO pin enabled as the hybrid mode gain change pin, the automatic gain control (AGC) overload counters are polled to determine if a gain increase or decrease is necessary. When the pulse is registered by the gain control block, a gain increase is made in the case of an underrange scenario, a gain decrease is made in the case of an overrange signal, or no gain change is made. This mode allows the user to determine when gain changes occur. The hybrid mode is a subset of AGC because hybrid mode uses many of the same circuits, thresholds, and counter parameters as AGC
- Automatic gain control (AGC). AGC determines when gain changes are made. There are several configurations that can be used in AGC mode. Examples include the option to reduce gain as soon as an overrange is detected (fast attack mode), to change gain (if necessary) only at the expiration of a variable length counter (AGC gain update counter, operating without fast attack), to synchronize the counter to an external pulse (AGC enable sync pulse), among others.

Selection between the three different modes of operation is performed with the MYKONOS_setRxGainControlMode(…) command for Rx1and Rx2. The ObsRx uses MYKONOS_ setObsRxGainControlMode(…) to set the gain control mode for the ORx1, ORx2, and all SnRx channels. Note that AGC for the ORx depends on the setting in device → obsRx → orxAgcCtrl → agcObsRxSelect. Descriptions of the gain control selection functions are provided in the following sections.

MYKONOS_setRxGainControlMode(…)

```
mykonosErr_t
   MYKONOS_setRxGainControlMode(mykonosDevice_
   t* device,mykonosGainMode_t mode)
```

Description

This function configures the Rx gain control mode.

Preconditions

Run this function after MYKONOS_initialize(…). It is recommended to wait until after the headless.c instructions are complete before running this function.

Parameters

- * device: This is a pointer to the device data structure.
- Mode: This is a mykonosGainMode_t enumerated data type indicating gain control mode, where manual gain is manual gain control, automatic gain control is automatic gain control, and hybrid mode is hybrid.

MYKONOS_setObsRxGainControlMode(…)

```
mykonosErr_t
   MYKONOS_setObsRxGainControlMode(mykonosDe
   vice_t* device, mykonosGainMode_t mode)
```

Description

This function configures the ORx gain control mode.

Preconditions

Run this function after MYKONOS_initialize(…).It is recommended to wait until after the headless.c instructions are complete before running this function.

Parameters

- * device: This is a pointer to the device data structure.
- Mode: a mykonosGainMode_t enumerated data type indicating gain control mode, where manual gain is manual gain control, automatic gain control is automatic gain control, and hybrid mode is hybrid.

Functions Common to All Gain Control Modes

Application programming interface (API) functions can also obtain the current gain index for a particular channel. These commands can also be used in any gain control mode. The commands to get the current gain index are as follows:

```
mykonosErr_t
   MYKONOS_getRx1Gain(mykonosDevice_t*
   device, uint8_t* rx1GainIndex)
mykonosErr_t
   MYKONOS_getRx2Gain(mykonosDevice_t*
   device, uint8_t* rx2GainIndex)
mykonosErr_t
   MYKONOS_getObsRxGain(mykonosDevice_t*
   device, uint8_t* gainIndex)
```

Whether in manual gain control, hybrid, or automatic gain control mode, the GPIO monitor outputs have several helpful status signals that assist users with determining when overloads occur in real-time. Refer to the General-Purpose Input/Output (GPIO) Configuration section for more information about the signals that can be observed on the GPIO pins.

MANUAL GAIN CONTROL

In manual gain control (MGC) mode, there are two ways to change the gain index pointer: application programming interface (API) commands and GPIO signaling. GPIO signaling is only available on Rx1 and Rx2.

API/SPI Mode MGC

The application programming interface (API) commands used to change gain in manual gain control (MGC) mode are listed in the following sections. The first two of these commands are specific to the Rx1 and Rx2 channels, respectively. The third command uses an enumerated data type to specify the ORx1, ORx2, SNRXA, SNRXB, or SNRXC channels of the ORx. Details regarding the API command mode for MGC are provided in the following section.

MYKONOS_setRx1ManualGain(…)

```
mykonosErr_t
   MYKONOS_setRx1ManualGain(mykonosDevice_
   t* device, uint8_t gainIndex)
```

Description

This function sets the Rx1 manual gain index.

If the value passed in the gainIndex parameter is within range of the gain table minimum and maximum indexes, the Rx1 gain index is updated in the device data structure and written to the transceiver. Otherwise, an error is returned. The maximum index is 255, and the minimum index is application specific.

Preconditions

Run this function after MYKONOS_initialize(…). It is recommended to wait until after headless.c instructions are complete before running this function.

Parameters

- * device: This is a pointer to the device data structure.
- gainIndex: This is the desired Rx1 gain index.

MYKONOS_setRx2ManualGain(…)

```
mykonosErr_t
   MYKONOS_setRx2ManualGain(mykonosDevice_
   t* device, uint8_t gainIndex)
```

Description

This function sets the Rx2 manual gain index.

If the value passed in the gainIndex parameter is within range of the gain table minimum and maximum indexes, the Rx2 gain index is updated in the device data structure and written to the transceiver. Otherwise, an error is returned. The maximum index is 255, and the minimum index is application specific.

Preconditions

Run this function after MYKONOS_initialize(…). It is recommended to wait until after headless.c instructions are complete before running this function.

Parameters

- * device: This is a pointer to the device data structure.
- gainIndex: This is the desired Rx2 gain index.

MYKONOS_setObsRxManualGain(…)

```
mykonosErr_t
   MYKONOS_setObsRxManualGain(mykonosDevice_
   t* device, mykonosObsRxChannels_t
   obsRxCh, uint8_t gainIndex)
```

Description

This function sets the Rx gain of the selected ORx channel by the obsRxCh parameter.

The ORx channel can have different RF inputs (ORx1/ORx2/ SNRXA, SNRXB, or SNRXC) This function sets the ORx gain index independently for ORx1/ORx2, or SnRx. SNRXA, SNRXB, and SNRXC share the same gain index. Note that ORx1/ORx2 share a gain table, as does SNRXA, SNRXB, and SNRXC. The maximum index is 255, and the minimum index is application specific.

Preconditions

Run this function after MYKONOS_initialize(…). It is recommended to wait until after headless.c instructions are complete before running this function.

Parameters

- * device: This is a pointer to the device data structure.
- obsRxCh: This is an enumeration to identify the desired RF input for gain change.
- gainIndex: the desired manual gain table index to be set.

GPIO Mode MGC

The GPIO pins can also be configured to make gain changes for Rx1 and Rx2. GPIO manual gain control (MGC) setup commands configure the gain index increment step size, gain index decrement step size, and pin selection for the increment and decrement pins. Some restrictions on the increment and decrement pin selections are as follows:

- The Rx1 increment pin must be either GPIO_0 or GPIO_10.
- The Rx1 decrement pin must be either GPIO_1 or GPIO_11.
- The Rx2 increment pin must be either GPIO_3 or GPIO_13.
- The Rx2 decrement pin must be either GPIO_4 or GPIO_14.

MYKONOS_setRx1GainCtrlPin(…)

```
mykonosErr_t
   MYKONOS_setRx1GainCtrlPin(mykonosDevice_t
   *device, uint8_t incStep, uint8_t
   decStep, mykonosGpioSelect_t
   rx1GainIncPin, mykonosGpioSelect_t
   rx1GainDecPin)
```

Description

This application programming interface (API) function configures the GPIO inputs for controlling the Rx gain, allowing the user to control the gain index in manual gain control (MGC) mode. If there is a high pulse on the rx1GainIncPin in pin control mode, it increments the gain by the value set in incStep. A high pulse on the rx1GainDecPin in pin control mode decrements the gain by the number of indices set in decStep.

Preconditions

Run this function after MYKONOS_initialize(…).

Parameters

- *device: This is a pointer to the device data structure.
- obsRxCh: This is an enumeration to identify the desired RF input for gain change.
- gainIndex: the desired manual gain table index to set.

MYKONOS_setRx2GainCtrlPin(…)

```
mykonosErr_t
   MYKONOS_setRx2GainCtrlPin(mykonosDevice_t
   *device, uint8_t incStep,  uint8_t
   decStep, mykonosGpioSelect_t
   rx2GainIncPin, mykonosGpioSelect_t
   rx2GainDecPin)
```

Description

This application programming interface (API) function configures the GPIO inputs for controlling the Rx gain, allowing the user to control the gain index in manual gain control (MGC) mode. If there is a high pulse on the rx2GainIncPin in pin control mode, it increments the gain by the value set in incStep. A high pulse on the rx2GainDecPin in pin control mode decrements the gain by the number of indices set in decStep.

Preconditions

Run this function after MYKONOS_initialize(…).

Parameters

- *device: This is a pointer to the device data structure.
- obsRxCh: This is an enumeration to identify the desired RF input for gain change.
- gainIndex: the desired manual gain table index to set.

These obsRxCh and gainIndex commands have complementary get commands for the set functions listed previously. These commands are MYKONOS_getRx1GainCtrlPin(…) and MYKONOS_getRx2GainCtrlPin(…).

HYBRID GAIN CONTROL

Hybrid gain control is a hybrid between automatic gain control (AGC) and manual gain control (MGC) modes of operation. Hybrid gain control mode allows the device to monitor the state of the overload detectors and waits for a baseband processor (BBP) signal on a GPIO to make a gain change. When an external pulse on a GPIO pin is provided, the gain control algorithm either increments the gain index if underrange conditions are detected, decrements the gain index if overrange conditions are detected, or does nothing if neither an underrange or overrange condition is detected. If fast attack for the analog peak detector (APD) or Half-Band 2 (HB2) is enabled, a gain decrease occurs immediately without the external pulse applied.

In hybrid gain control mode, the gain update counter does not run. The peak detector counters do not reset until a pulse is detected on the selected GPIO pin. For Rx1, the allowed pins for this feature are GPIO_1, GPIO_10, and GPIO_11. For Rx2, the allowed pins for this feature are GPIO_4, GPIO_10, and GPIO_13.

For proper operation of the hybrid gain control mode, the setup command for the AGC must be run. This command is MYKONOS_setupRxAgc(…). Running this command sets up all the necessary parameters for the AGC and hybrid mode. The parameters include threshold settings, how many indices to increment or decrement the gain index, and others. See the following section for an explanation of the AGC parameters.

When in hybrid mode, it is necessary to call a setup function to assign a GPIO pin per Rx channel for this feature. The function MYKONOS_setRxHybridGainChangePin(…) is described in the following section. Additionally, the GPIO pin selected must be set as an input (see the General-Purpose Input/Output (GPIO) Configuration section).

MYKONOS_setRxHybridGainChangePin(…)

```
mykonosErr_t
   MYKONOS_setRxHybridGainChangePin(mykonosD
   evice_t *device, mykonosGpioSelect_t
   rx1GainChangePin, mykonosGpioSelect_t
   rx2GainChangePin)
```

Description

This application programming interface (API) function sets the pins for hybrid gain control.

To call this function, set the gain mode to hybrid. The gain change is controlled with the selected GPIO pin. A pulse on the rx1GainChangePin in hybrid pin control enables the gain change for Rx1, and a pulse on the rx2GainChangePin in hybrid pin control enables the gain change for Rx2. A gain change is only made if deemed necessary by the automatic gain control (AGC).

Preconditions

Run this function after MYKONOS_initialize(…). Set the gain control mode to hybrid mode.

Parameters

- *device: This a pointer to the device data structure.
- rx1GainChangePin: This selects the GPIO pin to be used for hybrid gain change control. The available pins for Rx1 channel hybrid control are GPIO_1, GPIO_10, and GPIO_11. Use the MYKGPIONAN parameter for no GPIO selected.
- rx2GainChangePin: This selects the GPIO pin to be used for hybrid gain change control. The available pins for Rx2 channel hybrid control are GPIO_4, GPIO_10, and GPIO_13. Use the MYKGPIONAN parameter for no GPIO selected.

AUTOMATIC GAIN CONTROL (AGC)

When using AGC mode, there are several configurable parameters allowing the modification of how AGC mode responds to overrange or underrange conditions. This section provides details regarding configuration of the AGC by explaining the AGC theory of operation and the configuration options available in the application programming interface (API). This section explains the AGC at a high level. Parameters referenced in this section are data structure members within the following data structure types: mykonosAgcCfg_t, mykonosPeakDetAgcCfg_t, and mykonosPowerMeasAgcCfg_t.

AGC Theory of Operation

The automatic gain control (AGC) uses upper and lower thresholds from two peak detector circuits to determine if a gain index increment or decrement is necessary, depending on input signal conditions. The peak detector circuits include the analog peak detector (APD) and the Half-Band 2 (HB2) overload detector. Refer to Figure 57 or Figure 58 for the location of these peak detectors in the different receivers.

Additionally, power measurement detectors can be enabled to determine if a gain index increment or decrement is necessary. These blocks sample digital data at the HB2, programmable receiver finite impulse response (RFIR) filter, or dc correction block. Refer to Figure 57 or Figure 58 for the location of these power measurement detectors in the different receivers.

It is recommended to configure the AGC in peak threshold mode. However, the best configuration is application dependent. Peak threshold mode uses the APD/HB2 detectors to determine when to make gain changes. Peak threshold mode can also disable the power-based AGC mode by setting agcPeakThresholdMode = 1. Peak threshold mode, when configured for fast attack response to overrange signals mode (apdFastAttack, hb2FastAttack), offers the quickest response to the sudden onset of the blocking signals. Peak threshold mode with fast attack response is the recommended operation mode for the AGC.

If it is desired to use only power detector measurement AGC mode, set the gain step settings for the APD and HB2 to 0 and set agcPeakThresholdMode to 0 to enable the power measurement detectors to make gain changes.

Analog Peak Detector (APD) Basics

The APD overload detector is part of the peak threshold automatic gain control (AGC). The APD is considered the blocker overload detector because it determines if the received signal is overloading the blocks before the digital chain, including the ADC. Because the APD is located after the analog low-pass filter, attenuation increases for signals further from the Rx local oscillator (LO) frequency. APD thresholds are specified in units of voltage (peak). The ADC full-scale voltage is 707 mV peak.

The APD high threshold (apdHighThresh) determines if a gain reduction is necessary. If a detected input signal level is less than the APD high threshold but exceeds this threshold by a number of samples equal to or greater than a programmable number (apdHighThreshExceededCnt) within the duration of the AGC gain update counter (agcGainUpdateCounter), the AGC decrements the gain index according to the programmable APD gain step attack parameter (apdHighGainStepAttack) at a time depending on the APD fast attack setting (apdFastAttack). The gain decrement can occur when the overrange condition is detected by the APD or when the AGC gain update counter expires. If the potential gain index decrement places the gain index less than the AGC minimum gain index (for example, agcRx1MinGainIndex), the decrement is not made.

The APD low threshold (apdLowThresh) determines if an increase in gain is necessary. If a detected input signal level is less than the APD low threshold and does not exceed this threshold by a number of samples equal to or greater than a programmable number (apdLowThreshExceededCnt) within the duration of the AGC gain update counter (agcGainUpdateCounter), the AGC increments the gain index according to the programmable APD low gain step recovery parameter (apdLowGainStepRecovery). The gain index increment occurs at the expiration of the AGC gain update counter. If the potential gain index increment puts the gain index outside of the AGC maximum gain index (for example, agcRx1MaxGainIndex), the increment is not made.

Figure 61 shows the APD response to the presence and removal of an interferer when the APD fast attack bit is not set. Note that the gain decrement in response to the APD overrange condition occurs at the expiration of the AGC gain update counter.

Figure 62 shows the APD response to the presence and removal of an interferer when the APD fast attack bit is set. With this setting enabled, the total time when the Rx is overranged is reduced because the gain decrement occurs immediately when the APD high threshold exceeded counter overflows. All peak detector counters reset with a gain change. The AGC gain update counter does not reset with a gain change. The fast attack mode can help reduce the time the receiver ADC is overloaded, leading to smaller windows where data can be lost.

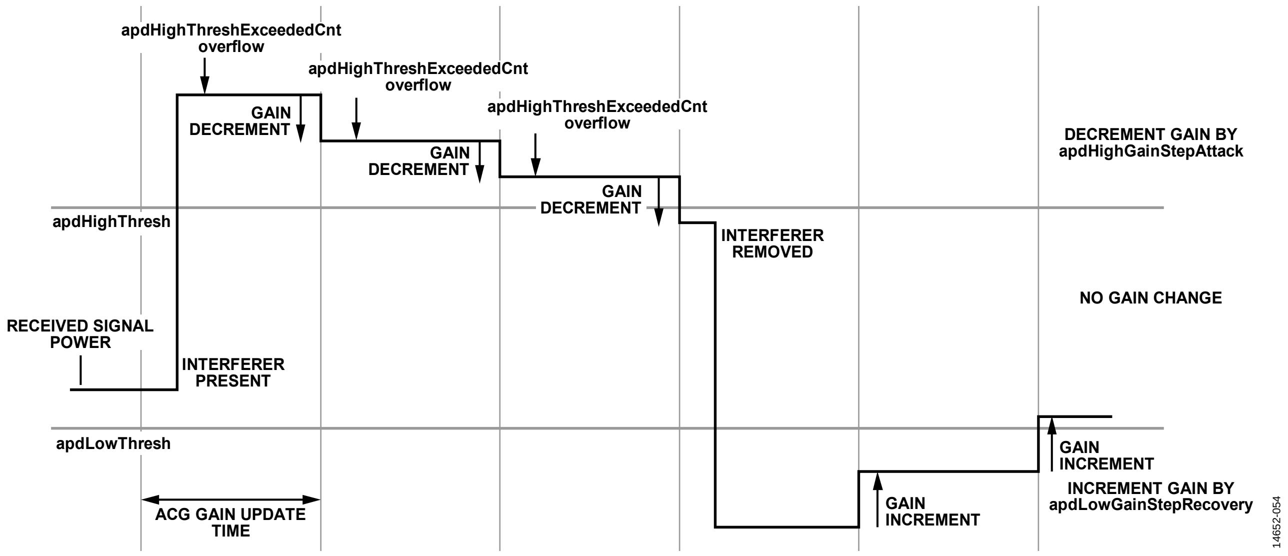


Figure 61. APD Response to Interferer Onset and Removal with APD Fast Attack Disabled

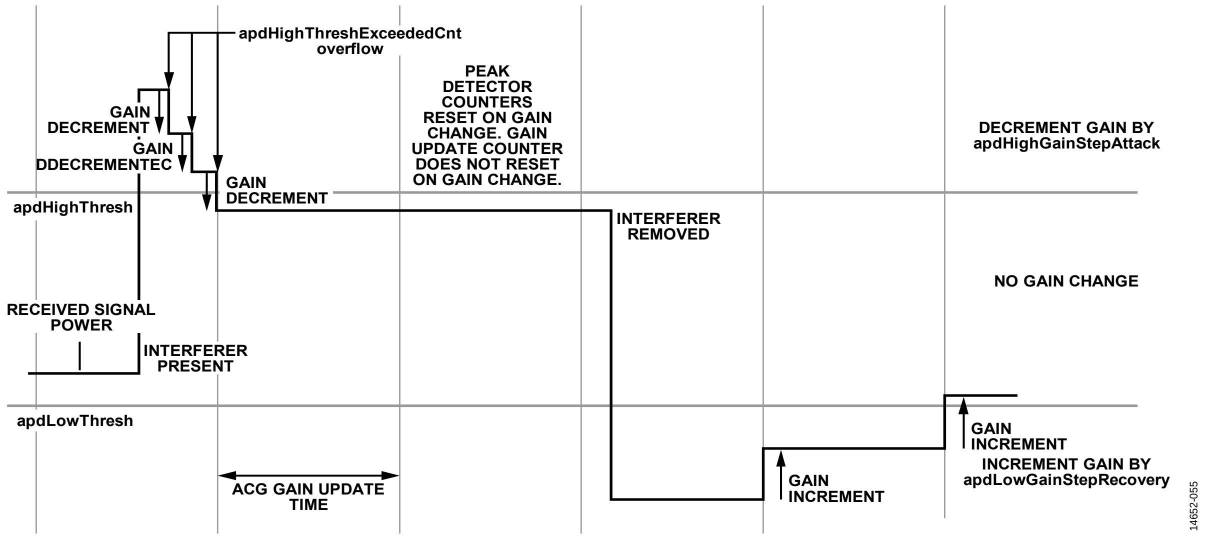

apdHighThreshExceededCnt overflow
PEAK DETECTOR COUNTERS RESET ON GAIN CHANGE. GAIN UPDATE COUNTER DOES NOT RESET ON GAIN CHANGE.
GAIN DECREMENT
GAIN DDECREMENTEC
GAIN DECREMENT
DECREMENT GAIN BY apdHighGainStepAttack
apdHighThresh
INTERFERER REMOVED
NO GAIN CHANGE
RECEIVED SIGNAL POWER
INTERFERER PRESENT
apdLowThresh
GAIN INCREMENT
INCREMENT GAIN BY apdLowGainStepRecovery
ACG GAIN UPDATE TIME
GAIN INCREMENT
14652-055

Figure 62. APD Response to Interferer Onset and Removal with APD Fast Attack Enabled

Half-Band 2 (HB2) Overload Detector Basics

The HB2 overload detector is part of the peak threshold AGC mode. The HB2 overload detector can be disabled if desired (hb2OverloadDetectEnable). The HB2 overload detector has configurable high, low, and very low thresholds. The HB2 overload detector is considered the decimated data overload detector because it uses the data output of the HB2 digital decimation filter to determine if an overload is occurring. The HB2 overload detector is similar in function to the APD. HB2 peak detector thresholds are specified in units of −dBFS relative to the ADC full-scale voltage.

The HB2 high threshold (hb2HighThresh) determines if a gain reduction is necessary. If a detected input signal level is less than the HB2 high threshold but exceeds this threshold by a number of samples equal to or greater than a programmable number (hb2HighThreshExceededCnt) within the duration of the AGC gain update counter (agcGainUpdateCounter), the AGC reduces the gain according to the programmable HB2 gain step attack parameter (hb2HighGainStepAttack) at a time depending on the HB2 fast attack setting (hb2FastAttack). The gain decrement can occur when the overrange condition is detected by the HB2 or when the AGC gain update counter expires. If the potential gain index decrement puts the gain index outside of the programmable gain index limits (for example, agcRx1MinGainIndex), the decrement is not made.

The HB2 low threshold (hb2LowThresh) determines if a gain increase is necessary. If a detected input signal level is less than the HB2 low threshold and does not exceed this threshold by a number of samples equal to or greater than a programmable number (hb2LowThreshExceededCnt) within the duration of the AGC gain update counter (agcGainUpdateCounter), the AGC increases the gain according to the programmable HB2 low threshold gain step recovery parameter (hb2LowGainStepRecovery). The gain index increment occurs at the expiration of the AGC gain update counter. If the potential gain index increment puts the gain index outside of the AGC maximum gain index (for example, agcRx1MaxGainIndex), the increment is not made.

The HB2 very low threshold (hb2VeryLowThresh) determines if a gain increase is necessary. If a detected input signal level is less than the HB2 very low threshold and does not exceed this threshold by a number of samples equal to or greater than a programmable number (hb2VeryLowThreshExceededCnt) within the duration of the AGC gain update counter (agcGainUpdateCounter), the AGC increases the gain according to the programmable HB2 very low threshold gain step recovery parameter (hb2LowGainStepRecovery). The gain index increment occurs at the expiration of the AGC gain update counter. If the potential gain index increment puts the gain index outside of the AGC maximum gain index (for example, agcRx1MaxGainIndex), the increment is not made. It is recommended to set the very low threshold gain step recovery parameter larger than the low threshold gain step recovery parameter to recovery gain quicker in very low underrange situations.

Figure 63 shows an example of HB2 operation within the AGC when the HB2 fast attack mode is not enabled. The behavior is similar to the behavior of the APD without fast attack. All peak detector counters reset upon a gain change. The AGC gain update counter does not reset upon a gain change.

The diagram in Figure 64 shows the same situation in the case where HB2 fast attack is enabled.

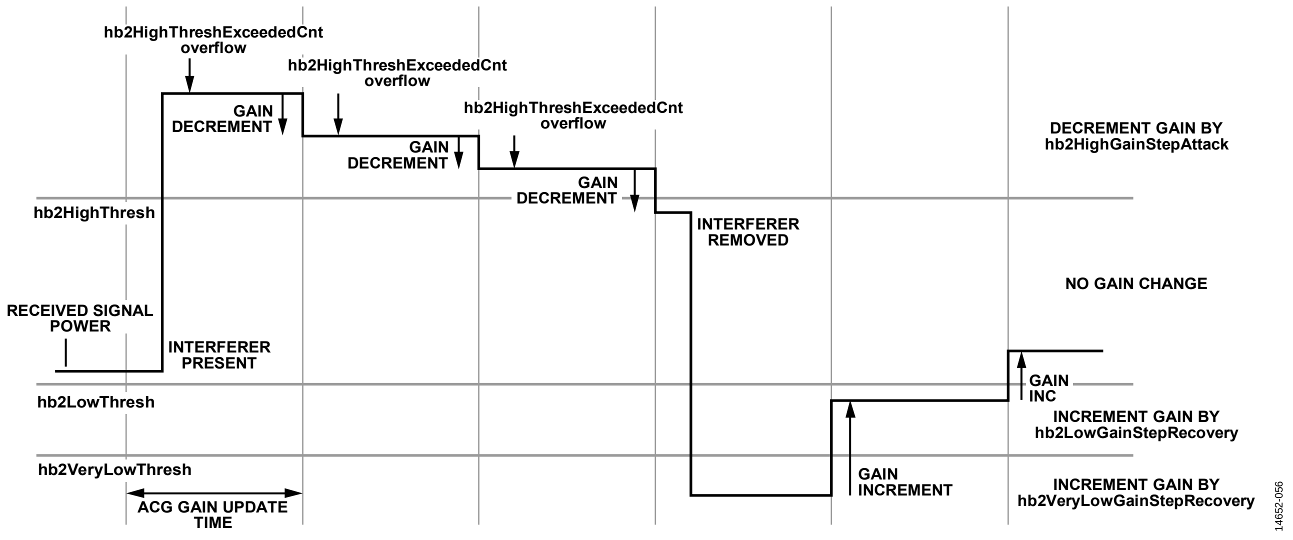

hb2HighThreshExceededCnt overflow
hb2HighThreshExceededCnt overflow
hb2HighThreshExceededCnt overflow
GAIN DECREMENT
GAIN DECREMENT
GAIN DECREMENT
DECREMENT GAIN BY hb2HighGainStepAttack
hb2HighThresh
INTERFERER REMOVED
NO GAIN CHANGE
RECEIVED SIGNAL POWER
INTERFERER PRESENT
GAIN INC
hb2LowThresh
INCREMENT GAIN BY hb2LowGainStepRecovery
hb2VeryLowThresh
GAIN INCREMENT
INCREMENT GAIN BY hb2VeryLowGainStepRecovery
ACG GAIN UPDATE TIME
14652-056

Figure 63. HB2 Thresholds and Gain Changes Associated with Underrange and Overrange Conditions with HB2 Fast Attack Mode Disabled

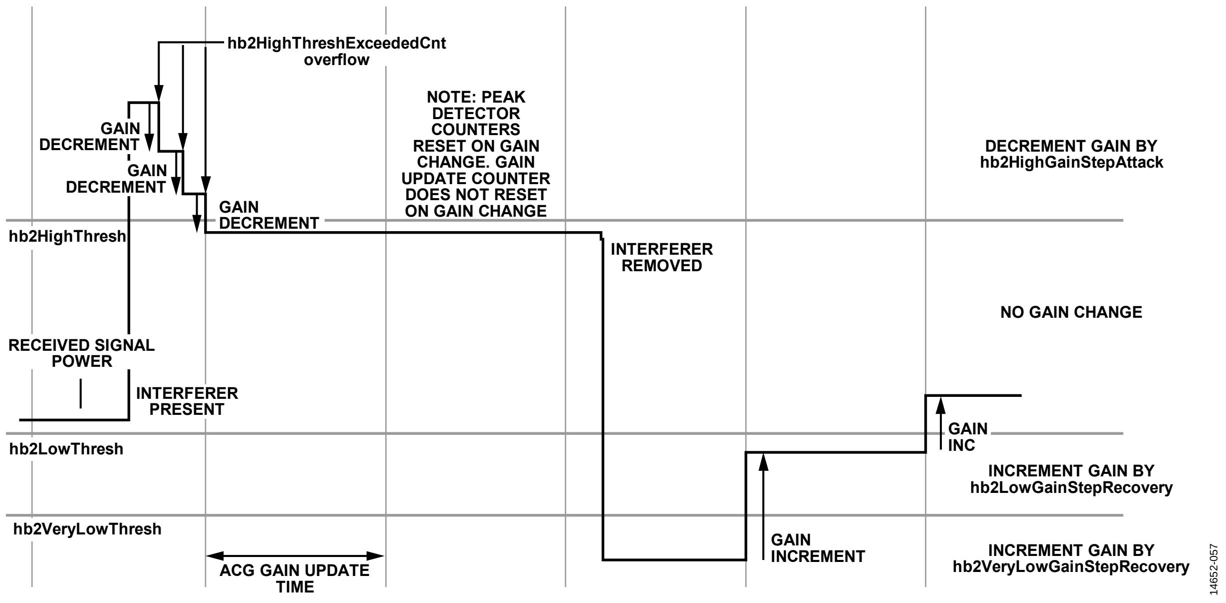

hb2HighThreshExceededCnt overflow
NOTE: PEAK DETECTOR COUNTERS RESET ON GAIN CHANGE. GAIN UPDATE COUNTER DOES NOT RESET ON GAIN CHANGE
GAIN DECREMENT
GAIN DECREMENT
GAIN DECREMENT
DECREMENT GAIN BY hb2HighGainStepAttack
hb2HighThresh
INTERFERER REMOVED
NO GAIN CHANGE
RECEIVED SIGNAL POWER
INTERFERER PRESENT
GAIN INC
hb2LowThresh
INCREMENT GAIN BY hb2LowGainStepRecovery
hb2VeryLowThresh
GAIN INCREMENT
INCREMENT GAIN BY hb2VeryLowGainStepRecovery
ACG GAIN UPDATE TIME
14652-057

Figure 64. HB2 Thresholds and Gain Changes Associated with Underrange and Overrange Conditions with HB2 Fast Attack Mode Enabled

Power Measurement Detector (PMD) Basics

The PMD obtains estimates of the digital power received at various points in the digital signal path. These locations include the HB2 output, RFIR output, and the output of the baseband dc (BBDC) correction block. The power measurement detector is a slower responding automatic gain control (AGC) mode than the peak threshold mode with fast attack enabled.

The thresholds are set such that the relative magnitude of the thresholds are pmdUpperHighThresh → pmdUpperLowThresh → pmdLowerHighThresh → pmdLowerLowThresh. It is recommended that the gain step associated with the outer thresholds (pmdUpperHighThresh, pmdLowerLowThresh) are greater than that of the inner thresholds (pmdUpperLowThresh, pmdLowerHighThresh) to speed gain attack or recovery in far overrange or far underrange conditions.

There are two PMD upper level thresholds to determine if a gain decrease is necessary. The upper thresholds are further split into high (pmdUpperHighThresh) and low (pmdUpperLowThresh) thresholds. The PMD is polled at the expiration of the AGC gain update counter (agcGainUpdateCounter) to determine if a gain index increment is necessary. If the pmdUpperHighThresh is exceeded, a gain index decrement is made according to pmdUpperHighGainStepAttack. If the pmdUpperLowThresh is exceeded, a gain index decrement is made according to pmdUpperLowGainStepAttack. Gain decrements occur only at the end of the AGC gain update counter.

There are two PMD lower level thresholds to determine if a gain increment is necessary. The lower thresholds are further split into high (pmdLowerHighThresh) and low (pmdLowerLowThresh) thresholds. The PMD is polled at the expiration of the AGC gain update counter (agcGainUpdateCounter) to determine if a gain index increment is necessary. If the pmdLowerHighThresh is exceeded, a gain index increment is made according to pmdLowerHighGainStepAttack. If the pmdLowerLowThresh is exceeded, a gain index decrement is made according to pmdLowerLowGainStepAttack. Gain decrements occur only at the end of the AGC gain update counter.

The diagram in Figure 65 shows the relative threshold levels for the PMD.

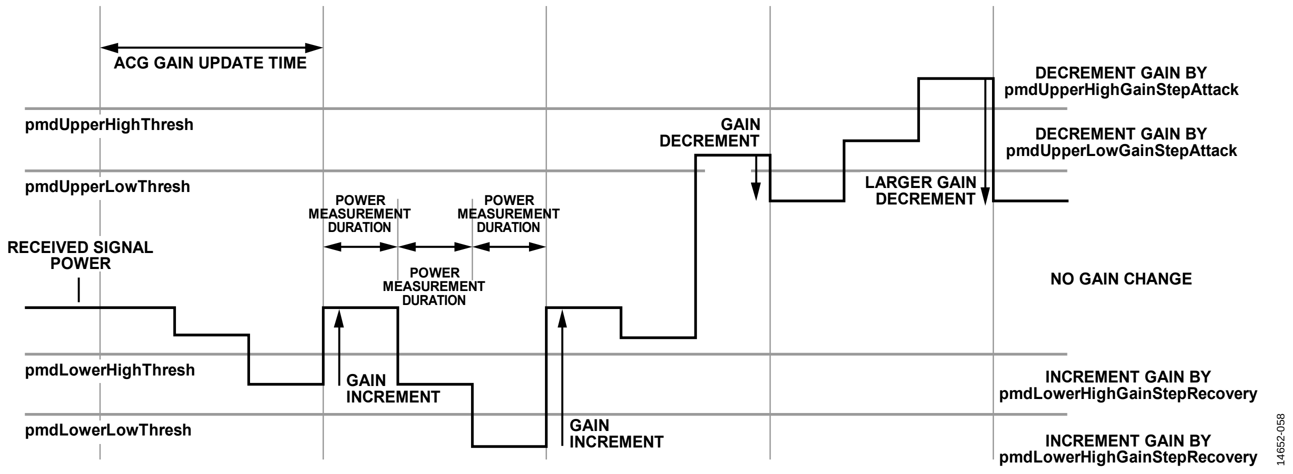

ACG GAIN UPDATE TIME
pmdUpperHighThresh
pmdUpperLowThresh
RECEIVED SIGNAL POWER
pmdLowerHighThresh
pmdLowerLowThresh
POWER MEASUREMENT DURATION
POWER MEASUREMENT DURATION
POWER MEASUREMENT DURATION
GAIN INCREMENT
GAIN INCREMENT
GAIN DECREMENT
LARGER GAIN DECREMENT
DECREMENT GAIN BY pmdUpperHighGainStepAttack
DECREMENT GAIN BY pmdUpperLowGainStepAttack
NO GAIN CHANGE
INCREMENT GAIN BY pmdLowerHighGainStepRecovery
INCREMENT GAIN BY pmdLowerHighGainStepRecovery
14652-058

Figure 65. Power Measurement Detector (PMD) Thresholds and Gain Index Changes Associated with Underrange and Overrange Conditions

Automatic Gain Control Operation

Gain decrements occur due to overrange conditions.

- The analog peak detector (APD) overrange condition occurs when the apdHighThreshExceededCnt counter overflows. Gain index decrement occurs immediately or at the end of the AGC gain update counter.
- The Half-Band 2 (HB2) overrange condition occurs when the hb2HighThreshExceededCnt counter overflows. Gain index decrement occurs immediately or at the end of the AGC gain update counter.
- Power-based overrange conditions occur when the power measurement exceeds either upper level power thresholds, pmdUpperHighThresh or pmdUpperLowThresh. Gain index decrements due to power-based overrange conditions only occur at the end of the AGC gain update counter.
- If multiple overrange conditions are detected, gain step priority is given to the APD attack gain step, then the HB2 attack gain step, then the PMD upper high threshold attack gain step, and then the PMD upper low threshold attack gain step.

Gain increments occur due to underrange conditions. All gain index increments occur at the end of the AGC gain update counter.

- The APD underrange condition occurs when the apdLowThreshExceededCnt counter does not overflow by the end of the AGC gain update counter.
- The HB2 underrange condition occurs when the hb2LowThreshExceededCnt counter does not overflow by the end of the AGC gain update counter.
- The HB2 very low underrange condition occurs when the hb2VeryLowThreshExceededCnt counter does not overflow by the end of the AGC gain update counter
- Power-based underrange conditions occur when the power measurement does not exceed either lower level power thresholds, pmdLowerHighThresh or pmdLowerLowThresh.
- If multiple underrange conditions are detected, gain step priority is given to the HB2 very low recovery gain step, then the APD low recovery gain step, then the HB2 low recovery gain step, then the PMD lower low threshold recovery gain step, and then the PMD lower high threshold recovery gain step.

When configuring the AGC, it is important to ensure that the difference between the high level thresholds and low level thresholds, in dB, is greater than the gain decrement step size and greater than the gain increment step size in dB. Setting the thresholds and step sizes in this way prevents a small overload from pushing the receiver from an overrange condition into an underrange condition when the gain decrement occurs, which may cause an underrange condition and a gain increment. This situation may lead to an undesirable gain index oscillation scenario.

A time domain depiction of the peak threshold with fast attack AGC operation is shown in Figure 66. Figure 66 also shows the receiver gain. The right side plot shows an example input waveform. Due to the presence of the large signal, the AGC reduces the receiver gain at approximately 1000 µs and 11,000 µs. The receiver gain decrements immediately after detecting the sufficient number of signal peaks, as described previously. When the large signal is removed from the input, the gain recovers. The gain index increments at the expiration of the AGC gain update counter. The gain recovery instances occur at 2000 µs and 12,000 µs. The time between gain increments is greater than the time between gain decrements because gain increments (recovery) wait until the expiration of the AGC gain update counter.

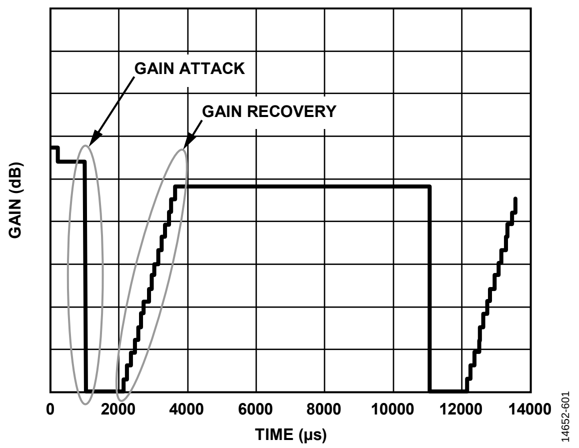


Figure 66. Typical Automatic Gain Control (AGC) Operation vs. Time

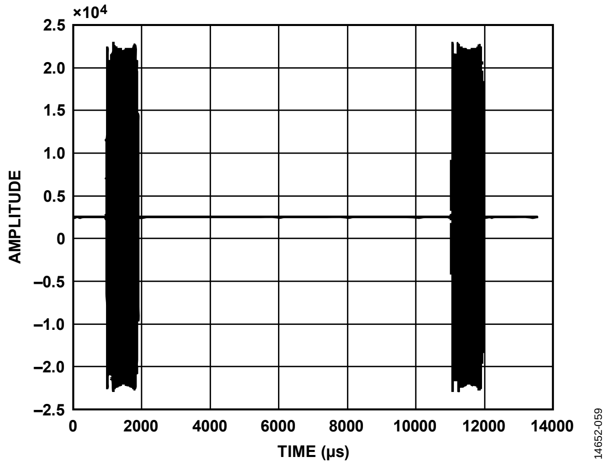


Figure 67. Typical Receiver Data Output Codes vs. Time with AGC Active

AGC API COMMANDS

The automatic gain control (AGC) data structure members must be programmed to the device registers by an application programming interface (API) command prior to putting the device into AGC mode. To program the AGC registers with the settings in the mykonosAgcCfg_t data structure, the following API commands can be used. Descriptions of the members of the mykonosAgcCfg_t data structure are provided in the following sections.

MYKONOS_setupRxAgc(…)

```
mykonosErr_t
   MYKONOS_setupRxAgc(mykonosDevice_t*
   device)
```

Description

This function sets up the device Rx automatic gain control (AGC) registers.

Three data structures (mykonosAgcCfg_t, mykonosPeakDetAgcCfg_t, and mykonosPowerMeasAgcCfg_t) must be instantiated prior to calling this function. Valid ranges for data structure members must also be provided.

Preconditions

Run this function after device initialization. It is recommended to wait until after headless.c instructions are complete before running this function.

Parameters

- *device: this is a pointer to the device data structure.

MYKONOS_setupObsRxAgc(…)

```
mykonosErr_t
   MYKONOS_setupObsRxAgc(mykonosDevice_t*
   device)
```

Description

This function sets up the device Rx AGC registers.

Three data structures (of types mykonosAgcCfg_t, mykonosPeakDetAgcCfg_t, and mykonosPowerMeasAgcCfg_t) must be instantiated prior to calling this function. Valid ranges for data structure members must also be provided.

Preconditions

Run this function after device initialization.

Parameters

- *device: This is a pointer to the device data structure.

The AGC enable sync pulse for the gain counter feature allows synchronization of the AGC gain update counter to the beginning of the time slots as signaled on the GPIO pins. The function to assign a pin for this functionality is given in the following section. The function can only be assigned to GPIO_1, GPIO_10, or GPIO_11 for Rx1, and GPIO_4, GPIO_10, or GPIO_13 for Rx2.

MYKONOS_setRxAgcEnSyncPin(…)

```
mykonosErr_t
   MYKONOS_setRxAgcEnSyncPin(mykonosDevice_t
   * device, mykonosGpioSelect_t
   rx1AgcSyncPin, mykonosGpioSelect_t
   rx2AgcSyncPin)
```

Description

This application programming interface (API) function sets the pins for sync automatic gain control (AGC).

To call this function, set the Rx gain control to AGC mode. The AGC gain sync is controlled with the selected GPIO pin. A pulse on rx1AgcSyncPin in AGC mode enables the AGC gain sync for Rx1, and a pulse on rx2AgcSyncPin in AGC enables the AGC gain sync for Rx2.

Preconditions

Run this function after device initialization in AGC mode. Set the device → rx → rxAgcCtrl → agcEnableSyncPulseForGainCounter parameter to 1 for this feature to work as intended; note that this is not a precondition for this function to be called.

Parameters

- *device: This is a pointer to the device data structure.

APPLICATION PROGRAMMING INTERFACE (API) PROGRAMMING SUMMARY

To summarize the programming requirements of the different gain control modes, a flowchart is provided in Figure 68 to assist users with setting up their desired gain control mode. The chart describes how to set up the manual gain control (MGC), hybrid, and automatic gain control (AGC) modes. For proper operation of GPIO based functions, refer to the General-Purpose Input/Output (GPIO) Configuration section. For GPIO input modes, such as hybrid mode, selected pins must be assigned as inputs in addition to the configuration described in the flowchart.

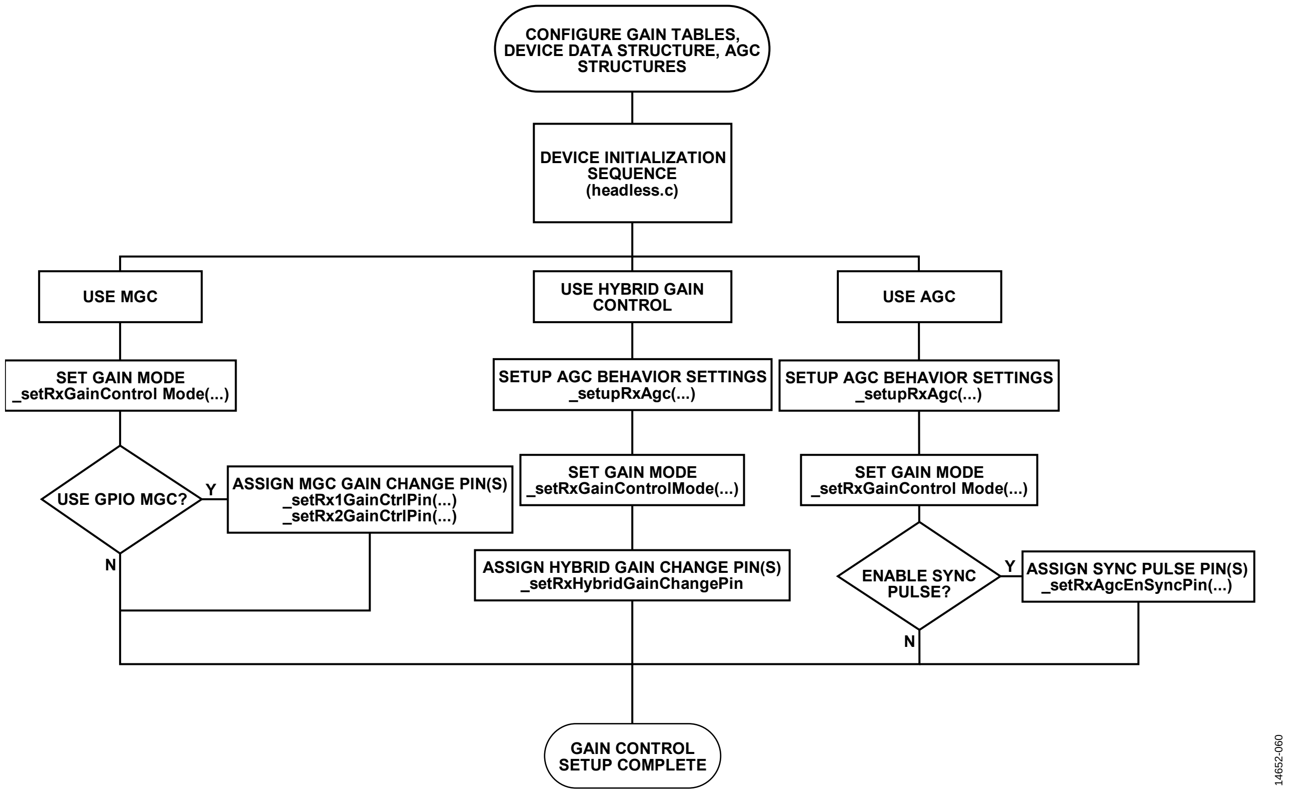

CONFIGURE GAIN TABLES, DEVICE DATA STRUCTURE, AGC STRUCTURES
DEVICE INITIALIZATION SEQUENCE (headless.c)
USE MGC
USE HYBRID GAIN CONTROL
USE AGC
SET GAIN MODE _setRxGainControl Mode(...)
SETUP AGC BEHAVIOR SETTINGS _setupRxAgc(...)
SETUP AGC BEHAVIOR SETTINGS _setupRxAgc(...)
USE GPIO MGC?
Y
N
ASSIGN MGC GAIN CHANGE PIN(S) _setRx1GainCtrlPin(...) _setRx2GainCtrlPin(...)
SET GAIN MODE _setRxGainControlMode(...)
SET GAIN MODE _setRxGainControl Mode(...)
ASSIGN HYBRID GAIN CHANGE PIN(S) _setRxHybridGainChangePin
ENABLE SYNC PULSE?
Y
N
ASSIGN SYNC PULSE PIN(S) _setRxAgcEnSyncPin(...)
GAIN CONTROL SETUP COMPLETE
14652-060

Figure 68. Gain Control Programming Flowchart

Data Structures and Parameters for AGC and Hybrid Mode

The automatic gain control (AGC) allows the automatic adjustment of receiver gain to avoid overrange conditions and underrange conditions. The AGC engine compares the received signal to programmable thresholds at various points in the signal chain. This section explains the programmable settings of the AGC by looking into the AGC configuration data structures.

The AGC uses three data structures to store configuration settings. For the Rx channels, the AGC configuration data structures are substructures to the mykonosRxSettings_t data structure. For the ORx channels, the AGC configuration data structures are substructures to the mykonosObsRxSettings_t data structure.

The AGC configuration data structures are divided into a higher level data structure, corresponding to general AGC configuration settings called mykonosAgcCfg_t, with two substructures for more specific configuration settings. The two substructures correspond to peak detector AGC settings (mykonosPeakDetAgcCfg_t) and power measurement detector settings (mykonosPowerMeasAgcCfg_t). The Rx AGC and ORx AGC use the same data structure types; however, one instance is specific for Rx and the other specific for ORx. The data structures are shown in Figure 69.

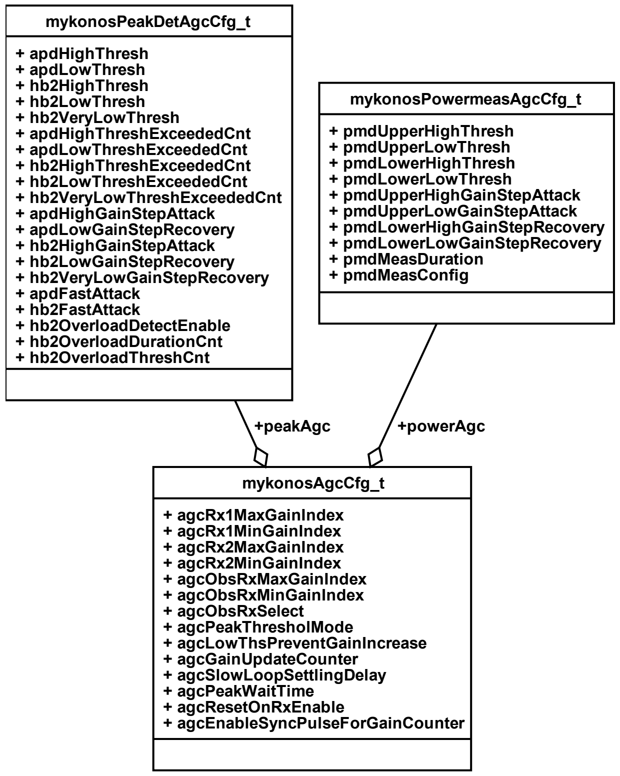


Figure 69. Member Listing of the mykonosAgcCfg_t, mykonosPeakDetAgcCfg_t, and mykonosPowerMeasAgcCfg_t Data Structures

mykonosAgcCfg_t Data Structure

The following sections describe parameters within the data structure mykonosAgcCfg_t. This data structure contains the peak detector automatic gain control (AGC) settings, power measurement AGC settings, and several other general parameters for AGC operation.

AGC Minimum and Maximum Receiver Gain Indices

The following sections outline the parameters within the three automatic gain control (AGC) configuration structures with recommended settings and minimum and maximum settings. At the end of the section is a summary of all default, minimum, and maximum settings for the AGC data structures.

The members of the mykonosAgcCfg_t structure referring to maximum and minimum gain indices for a given receiver channel are listed in Table 100. These parameters limit the AGC to make gain change decisions that result in gain indices within the minimum and maximum parameter specified for a given channel.

The current application programming interface (API) implementation dictates that agcObsRxMaxGainIndex and agcObsRxMinGainIndex are used in reference to the SnRx channels. If the user is attempting to set up the mykonosAgcCfg_t data structure for use with an ORx input, the agcObsRxSelect member must be set for SnRx usage.

Table 101 provides the receiver associated with the valid values of the agcObsRxSelect member. Only one value is supported for this data structure member.

The MYKONOS_setupRxAgc(…) function does not program the parameters specific to the ORx AGC. These ignored parameters are agcObsRxSelect, agcObsRxMaxGainIndex, and agcObsRxMinGainIndex.

The MYKONOS_setupObsRxAgc(…) function does not program parameters specific to the Rx AGC. These ignored parameters are agcRx1MaxGainIndex, agcRx1MinGainIndex, agcRx2MaxGainIndex, and agcRx2MinGainIndex.

AGC Peak Threshold Mode

The member agcPeakThresholdMode of the mykonosAgcCfg_t data structure determines if the automatic gain control (AGC) runs in peak threshold mode. This is a 1-bit field. Setting this bit disables the power measurement detector from making gain changes. Setting this bit also enables the analog peak detector (APD)/Half-Band 2 (HB2) lower thresholds to make gain increments.

Peak threshold mode is the recommended AGC configuration because it allows for fast attack response (see apdFastAttack, hb2FastAttack) in response to the sudden presence of a blocking signal.

Table 100. AGC Minimum and Maximum Gain Index Value Limits

Data Type	Parameter	Default Value	Minimum Value	Maximum Value
uint8_t	agcRx1MaxGainIndex	255	agcRx1MinGainIndex	255
uint8_t	agcRx1MinGainIndex	195	Minimum Rx1 Table Index	agcRx1MaxGainIndex
uint8_t	agcRx2MaxGainIndex	255	agcRx2MinGainIndex	255
uint8_t	agcRx2MinGainIndex	195	Minimum Rx2 Table Index	agcRx2MaxGainIndex
uint8_t	agcObsRxMaxGainIndex	255	agcObsRxMinGainIndex	255
uint8_t	agcObsRxMinGainIndex	203	Minimum ObsRx Table Index	agcObsRxMaxGainIndex

Table 101. Parameter Limits and Defaults for agcObsRxSelect

Data Type	Parameter	Default Value	Minimum Value	Maximum Value
uint8_t	agcObsRxSelect	1	1	1

Table 102. Parameter Definitions for agcPeakThresholdMode[1]

Data Type	Parameter	Value	Note
uint8_t	agcPeakThresholdMode	0	Power measurement detectors are enabled to make gain changes. Disables APD/HB2 from making gain step increments.
		1 (default)	Power measurement detectors are disabled from making gain changes. Enables APD/HB2 to make gain step increments

[1] All values greater than 1 result in an error.

Table 103. Parameter Definitions for agcLowerThreshPreventGainIncrease[1]

Data Type	Parameter	Value	Note
uint8_t	agcLowerThreshPreventGainInc	0	PMD lower thresholds increase gain; however, disregard if the APD/HB2 low threshold is exceeded.
		1 (default)	PMD cannot increase gain when the APD/HB2 low thresholds are exceeded.

[1] All values greater than 1 result in an error.

AGC Lower Threshold Prevent Gain Increase

The agcLowerThreshPreventGainInc member of the mykonosAgcCfg_t data structure prevents the gain index from incrementing in certain conditions when this bit is set to 1. This is a 1-bit field. If this bit is set, the power measurement detector (PMD) cannot initiate a gain increment when either the analog peak detector (APD) or Half-Band 2 (HB2) low thresholds are exceeded. If this bit is cleared, APD or HB2 low thresholds are ignored by the automatic gain control (AGC).

AGC Gain Update Counter

The automatic gain control (AGC) gain update time (denoted by the agcGainUpdate-Counter member) is a member that determines the length of the gain update counter. The interval between the gain update counter expiration is determined by agcGainUpdateCounter and agcSlowLoopSettlingDelay, though agcGainUpdateCounter length is typically several orders of magnitude greater than the agcSlowLoopSettling delay. Refer to Figure 70 for a depiction of agcSlowLoopSettlingDelay and agcGainUpdateCounter.

The total number of IQ data rate clock cycles between expiration of the AGC gain update counter is given by the following equation:

$$\textit{IQ Clock Cycles} = \textit{agcGainupdateCounter}[21:0] + \textit{agcSlowLoopSettingDelay}[4:0] \quad (4)$$

When the AGC gain update counter expires, the peak detectors and/or power measurement detectors are polled to determine if a gain change is necessary. Gain increments can only occur on the termination of this counter. Gain decrements can occur at the end of the counter or when the apdHighThreshExceededCnt or hb2HighThreshExceeddedCnt counters overflow (see apdFastAttack and hb2FastAttack).

The AGC gain update counter runs at the IQ data rate of the receiver. The following equation governs the AGC gain update time (the left side indicates the value of the agcGainUpdateCounter for a given gain update counter length on the right side):

$$\textit{agcGainupdateCounter}[21:0] = \textit{agcGainupdateCounterLength}\ (\mu s) \times \textit{IQ Rate}\ (\text{MHz}) \quad (5)$$

Using this equation, for an AGC gain update counter length of 250 µs with an Rx profile that has an IQ data rate of 122.88 MHz, the agcGainUpdateCounter setting is 30720 (decimal). Table 104 shows the limits of the agcGainUpdateCounter parameter.

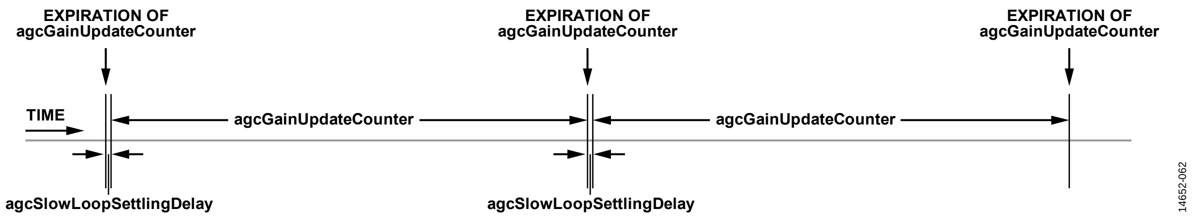

EXPIRATION OF agcGainUpdateCounter
EXPIRATION OF agcGainUpdateCounter
EXPIRATION OF agcGainUpdateCounter
TIME
agcGainUpdateCounter
agcGainUpdateCounter
agcSlowLoopSettlingDelay
agcSlowLoopSettlingDelay
14652-062

Figure 70. AGC Gain Update Counter Timing Diagram

Table 104. Parameter Limits and Defaults for agcGainUpdateCounter

Data Type	Parameter	Default Value	Minimum Value	Maximum Value
uint32_t	agcGainUpdateCounter	30720 (0x7800)	1 (0x000001)	4194303 (0x3FFFFF)

Table 105. Parameter Limits and Defaults for agcSlowLoopSettlingDelay

Data Type	Parameter	Default Value	Minimum Value	Maximum Value
uint8_t	agcSlowLoopSettlingDelay	4	0	31 (0x1F)

Table 106. Parameter Limits and Defaults for agcPeakWaitTime

Data Type	Parameter	Default Value	Minimum Value	Maximum Value
uint8_t	agcPeakWaitTime	2	2	31 (0x1F)

Table 107. Parameter Definitions for agcResetOnRxEnable[1]

Data Type	Parameter	Value	Note
uint8_t	agcResetOnRxEnable	0 (default)	AGC preserves state on falling edge of Rx enable or wait/radio off state
		1	AGC resets on falling edge of Rx enable or wait/radio_off state

[1] All values greater than 1 result in an error.

Table 108. Parameter Definitions for agcEnableSyncPulseForGainCounter[1]

Data Type	Parameter	Value	Note
uint8_t	agcEnableSyncPulseForGainCounter	0 (default)	AGC gain update counter operates as normal
		1	AGC gain update counter can be synchronized to an external source

[1] All values greater than 1 result in an error.

AGC Slow Loop Settling Delay

The automatic gain control (AGC) slow loop settling delay (denoted by member agcSlowLoopSettlingDelay) determines the number of IQ data rate clock cycles to wait after a gain change before resuming operation of the analog peak detector (APD)/Half-Band 2 (HB2) or power measurement detector (PMD). This is a 5-bit field. This parameter allows the AGC to ignore any transients associated with a gain change for the number of cycles indicated by this parameter.

If this parameter is set to 4, for example, the APD/HB2 and PMD blocks are held in reset for four IQ data rate clock cycles before resuming normal operation. Table 105 shows the limits of the agcSlowLoopSettlingDelay.

AGC Peak Wait Time

The AGC peak wait time (denoted by member agcPeakWaitTime) configures the amount of time that the gain control algorithm waits before enabling regular operation of the peak detectors after AGC is enabled or a peak is detected. This is a 5-bit field. The peak detectors in the receiver datapaths include the APD and HB2 overrange detector.

The value of the agcPeakWaitTime member of the mykonosAgcCfg_t data structure is the number of IQ data clock cycles that elapse after using enabling AGC and prior to peak detector circuits entering regular operation. This time is also the minimum time for the AGC to wait after detecting a peak.

AGC Reset on Rx Enable

The AGC reset on Rx enable (denoted by member agcResetOnRxEnable) allows for a reset of the AGC when the receiver is turned on. When this bit is set, the receiver resets the AGC to its initial state when the Rx is disabled. The gain index is reset to the maximum condition when the Rx is disabled. When this bit is set to 0, the AGC holds its current state when Rx enable is taken low. When Rx enable goes high again, AGC continues its operation.

AGC Enable Sync Pulse for Gain Counter

The AGC enable sync pulse for gain counter (denoted by the agcEnableSyncPulseForGainCounter member) allows synchronization of the AGC gain update counter to beginning of time slots as signaled on GPIO pins. It is also required to call the MYKONOS_setRxAgcEnSyncPin(…)command before synchronization can occur.

mykonosPeakDetAgcCfg_t Data Structure

The following sections describe parameters within the data structure mykonosPeakDetAgcCfg_t. This data structure contains several parameters that set the behavior of the peak detector mode AGC.

APD Thresholds

Two members of the mykonosPeakDetAgcCfg_t data structure determine the high and low threshold levels for the analog peak detector (APD) circuit. They are denoted by apdHighThresh and apdLowThresh. Each are 6-bit fields. The APD determines if received signals contain peaks greater than or less than the apdHighThresh and apdLowThresh signal threshold levels. The following equations relate the value of the apdHighThresh and apdLowThresh data structure members to the high and low peak voltage threshold levels.

$$\textit{apdHighThresh}\text{ (mV peak)} = 16\text{ mV} \times (\textit{apdHighThresh[D5:D0]} + 1) \quad (6)$$

$$\textit{apdLowThresh}\text{ (mV peak)} = 16\text{ mV} \times (\textit{apdLowThresh[D5:D0]} + 1) \quad (7)$$

The full-scale voltage of the Rx/ORx ADC is 707 mV peak. The APD thresholds can be approximated in −dBFS with the following equation, which is an approximation because some roll-off is introduced by the analog transimpedance amplifier (TIA) low-pass filter (LPF) prior to the APD circuitry.

$$\textit{Threshold}\text{ (dBFS)} = 20 \times \log((\textit{Threshold}\text{ (mV peak)}/707\text{ mV peak}) \quad (8)$$

For example, if the apdHighThresh is set as 0x1F, this corresponds to a peak voltage of 512 mV peak, which is approximately −2.8 dBFS.

If the received signal exceeds the apdHighThresh signal level for the number of times set by the apdHighThreshExceededCnt during the agcGainUpdateCounter counter duration, a gain decrease is made based on the apdGainStepAttack. The gain decrease can occur either at the expiration of the agcGainUpdateCounter or be made immediately when the apdHighThreshExceededCnt value is exceeded. The timing of the gain decrement is controlled by the apdFastAttack member.

If the received signal does not exceed the apdLowThresh for the number of times set by the apdLowerThreshExceededCnt within the agcGainUpdateCounter, a gain increase is made according to the apdGainStepRecovery member.

APD Threshold Counts

Two members determine the number of times the apdHighThresh or apdLowThresh peak thresholds must be exceeded to trigger a gain change. They are denoted by apdHighThreshExceededCnt for the apdHighThresh threshold exceeded count and apdLowThreshExceededCnt for the apdLowThresh threshold exceeded count. When the count value exceeds the apdHighThreshExceededCnt, a gain index decrement of apdGainStepAttack indices occurs at a time determined by the apdFastAttack data structure member. When the count value does not exceed the apdLowThreshExceededCnt within the duration of the agcGainUpdateCounter, a gain index increment of apdGainStepRecovery indices occurs at the expiration of the agcGainUpdate-Counter.

Table 109. Parameter Limits and Defaults for apdHighThresh/apdLowThresh[1]

Data Type	Parameter	Default Value	Minimum Value	Maximum Value
uint8_t	apdHighThresh	0x1F	0	0x3F
uint8_t	apdLowThresh	0x16	0	0x3F

[1] Do not use values less than 0x7.

Table 110. Parameter Limits and Default Values for apdHighThreshExceededCnt/apdLowThreshExceededCnt

Data Type	Parameter	Default Value	Minimum Value	Maximum Value
uint8_t	apdHighThreshExceededCnt	6	0	255
uint8_t	apdLowThreshExceededCnt	4	0	255

Table 111. Parameter Limits and Default Values for apdHighGainStepAttack

Data Type	Parameter	Default Value	Minimum Value	Maximum Value
uint8_t	apdHighGainStepAttack	2	0	31

Table 112. Parameter Limits and Default Values for apdLowGainStepRecovery

Data Type	Parameter	Default Value	Minimum Value	Maximum Value
uint8_t	apdLowGainStepRecovery	2	0	31

APD Gain Step Attack

The analog peak detector (APD) gain step attack (denoted by apdHighGainStepAttack) is a member that determines the number of gain indices decremented when the apdHighThreshExceededCnt counter is exceeded, which is the APD overrange condition. When a received signal exceeds the apdHighThresh level the number of times set by the apdHighThreshExceededCnt, the gain is decremented by the number of indices specified by the apdHighGainStepAttack. The apdFastAttack member can control the timing of the gain decrement.

Set the value of this member as the number of gain steps to decrement during APD overrange conditions. This step size depends on the application and the implemented gain table. It is recommended to make this step size the same step size as the hb2GainStepAttack parameter.

APD Gain Step Recovery

The analog peak detector (APD) gain step recovery (denoted by apdLowGainStepRecovery) is a member that determines the number of gain indices incremented when the apdLowThresh-ExceededCnt counter is not exceeded. This is the APD underrange condition. When a received signal does not exceed the apdLowThresh level the number of times set by the apdLowThreshExceededCnt, the gain is incremented by the number of indices specified by the apdLowGainStepRecovery. The gain step always occurs at the end of expiration of the agcGainUpdateCounter.

Set the value of this member as the number of gain steps to increment during APD underrange conditions. This step size depends on the application and the implemented gain table. It is recommended to make this step size the same step size as the hb2GainStepRecovery parameter.

HB2 Thresholds

Three members determine the high, low, and very low threshold levels for the Receive Half-Band 2 (HB2) overload detector. They are denoted by hb2HighThresh, hb2LowThresh, and hb2VeryLowThresh (see Table 113). The HB2 overload detector determines if the decimated data at the output of the receiver HB2 digital decimation filter exceeds the hb2HighThresh, hb2LowThresh, and hb2VeryLowThresh.

The following equations provide the threshold levels for the HB2 overload detector.

$$hb2HighThresh[7:0] = 256 \times 10^{\frac{hb2High_{dBFS}}{20}} \tag{9}$$

$$hb2LowThresh[7:0] = 256 \times 10^{\frac{hb2Low_{dBFS}}{20}} \tag{10}$$

$$hb2VeryLowThresh[7:0] = 256 \times 10^{\frac{hb2VeryLow_{dBFS}}{20}} \tag{11}$$

The hb2HighThresh sets a threshold for HB2 overrange conditions. If the received signal exceeds the hb2HighThresh the number of times specified by hb2HighThreshExceededCnt, the AGC makes a gain decrement according to the hb2HighGainStepAttack member, which is similar to the behavior of the APD high threshold detection. The timing of the gain decrement is according to the hb2FastAttack bit.

Note that the apdHighThresh overload has a higher priority than the hb2HighThresh overload. If the detected signal exceeds both the apdHighThresh and the hb2HighThresh levels for their respective counter values, the AGC makes a gain increment according to the higher priority detector gain step, namely, apdGainStepAttack.

The hb2LowThresh sets a threshold for HB2 underrange conditions. If the detected signal does not exceed the hb2LowThresh the number of times specified by hb2LowThreshExceededCnt, the AGC makes a gain increment according to the HB2 low recovery gain step member, hb2LowGainStepRecovery.

The hb2VeryLowThresh sets a lower threshold than hb2LowThresh to allow faster gain recovery. If the detected signal does not exceed the hb2VeryLowThresh the number of times specified by hb2VeryLowThreshCnt, the AGC makes a gain increase according to the HB2 very low recovery gain step member hb2VeryLowGainStepRecovery.

From Equation 9, Equation 10, and Equation 11, the thresholds must be input such that the magnitude of the following:

$$hb2HighThresh\ (\text{dBFS}) > hb2LowThresh\ (\text{dBFS}) > hb2VeryLowThresh\ (\text{dBFS}) \tag{12}$$

Table 113. Parameter Limits and Default Values for Half-Band 2 (HB2) Thresholds

Data Type	Parameter	Default Value	Minimum Value	Maximum Value
uint8_t	hb2HighThresh	0xB5	0	255
uint8_t	hb2LowThresh	0x80	0	hb2HighThresh
uint8_t	hb2VeryLowThresh	0x40	0	hb2LowThresh

HB2 Threshold Counts

Three members determine the number of times the Half-Band 2 (HB2) thresholds must be exceeded to trigger a gain change. These members are hb2HighThreshExceededCnt for the hb2HighThresh threshold exceeded count, hb2LowThreshExceeededCnt for the hb2LowThresh threshold exceeded count, and hb2VeryLowThreshExceededCnt for the hb2VeryLowThresh threshold exceeded count.

For HB2 overrange conditions, the HB2 overload detector must detect greater than hb2HighThreshExceededCnt number of overloads above the hb2HighThresh within the agcGain-UpdateCounter.

For HB2 underrange conditions, the HB2 overload detector must detect less than hb2LowThreshExceededCnt or hb2VeryLowThreshExceededCnt number of overloads above the hb2LowThresh or hb2VeryLowThresh, respectively. These overloads must be detected within the agcGainUpdateCounter.

HB2 Gain Step Attack

The HB2 gain step attack (hb2HighGainStepAttack) is a member that determines the number of gain indices decremented when the hb2HighThreshExceededCnt counter is exceeded, which is the HB2 overrange condition. When a received signal exceeds the hb2HighThresh level the number of times set by the hb2HighThreshExceededCnt, the gain is decremented by the number of indices specified by the hb2HighGainStepAttack. The timing of the gain decrement is controlled by the hb2FastAttack member.

Set the value of this member as the number of gain steps to decrement during HB2 overrange conditions. This step size depends on the application and the implemented gain table. It is recommended to make this step size the same step size as the apdGainStepAttack parameter.

HB2 Gain Step Recovery

Two members allow recovery from the HB2 underrange conditions (denoted by hb2LowGainStepRecovery and hb2VeryLowGainStepRecovery). If the underrange condition is with respect to the hb2VeryLowThresh, the gain index increment is made according to the hb2VeryLowGainStepRecovery. If the underrange condition is with respect to the hb2LowThresh, the gain index increment is made according to the hb2LowGainStepRecovery. It is recommended to set the hb2VeryLowGainStepRecovery parameter larger than the hb2LowGainStepRecovery to enable quicker recovery. The timing of the gain index step occurs at the expiration of the agcGainUpdateCounter.

APD and HB2 Fast Attack Setting

The automatic gain control (AGC), in analog peak detector (APD) and HB2 overrange conditions, can be programmed to make a gain step immediately when the overrange occurs or to make the gain step occur at the end of the agcGainUpdateCounter. It is recommended to set the fast attack bits for the APD and HB2. The fast attack setting is controlled by the apdFastAttack for APD overrange conditions, and hb2FastAttack for HB2 overrange conditions. These are both 1-bit fields.

Table 114. Parameter Limits and Default Values for HB2 Threshold Counters

Data Type	Parameter	Default Value	Minimum Value	Maximum Value
uint8_t	hb2HighThreshExceededCnt	6	0	255
uint8_t	hb2LowThreshExceededCnt	4	0	255
uint8_t	hb2VeryLowThreshExceededCnt	4	0	255

Table 115. Parameter Limits and Default Values for apdHighGainStepAttack

Data Type	Parameter	Default Value	Minimum Value	Maximum Value
uint8_t	hb2HighGainStepAttack	2	0	31

Table 116. Parameter Limits and Default Values for apdHighGainStepAttack

Data Type	Parameter	Default Value	Minimum Value	Maximum Value
uint8_t	hb2LowGainStepRecovery	2	0	31
uint8_t	hb2VeryLowGainStepRecovery	4	0	31

Table 117. Parameter Definitions for Fast Attack Settings

Data Type	Parameter	Value	Note
uint8_t	apdFastAttack	0	In response to APD overrange, gain decrement occurs at the expiration of the agcGainUpdateCounter
		1 (default)	In response to APD overrange, gain decrement occurs immediately
uint8_t	hb2FastAttack	0	In response to HB2 overrange, gain decrement occurs at the expiration of the agcGainUpdateCounter
		1 (default)	In response to HB2 overrange, gain decrement occurs immediately

HB2 Overload Detect Enable

The hb2OverloadDetectEnable parameter is a 1-bit field that enables the High-Band 2 (HB2) detector. It is recommended to set this bit.

HB2 Overload Duration Count

The HB2 overload duration count (hb2OverloadDurationCnt) specifies the size of the window of IQ data rate clock cycles to meet the overload count set in hb2OverloadThreshCnt to increment the HB2 overload detector counters. This is a 3-bit field. If the number of overload counts set in hb2Overload-ThreshCnt is exceeded within the window of IQ data rates provided by hb2OverloadThreshCnt, the counter increments, which avoids double counting overload instances.

HB2 Overload Threshold Count

The 4-bit field hb2OverloadThreshCnt specifies the number of individual overload instances within the number of samples given by hb2OverloadDurationCnt required to increment the HB2 overload detector counter. For example, if hb2Overload-DurationCnt = 1 (sample size of 4) and hb2OverloadThreshCnt is 1, then 1 overload instance must be detected within the duration of the receiver HB2 overload duration count to trigger the receiver HB2 overload detector signal. Each LSB in this field maps to a single sample. The default is 1.

mykonosPowerMeasAgcCfg_t Data Structure

The following sections describe parameters within the mykonosPowerMeasAgcCfg_t data structure. This data structure contains several parameters that set the behavior of the power measurement mode automatic gain control (AGC).

PMD Thresholds

There are four thresholds for the power measurement detector (PMD) that correspond to four gain step sizes. The PMD thresholds are located in the mykonosPowerMeasAgcCfg_t data structure. The members that set the threshold levels are, from highest signal power to lowest, pmdUpperHighThresh, pmdUpperLowThresh, pmdLowerHighThresh, and pmdLower-LowThresh. The pmdUpperLowThresh and pmdLowerHigh-Thresh are 7-bit fields that set the absolute threshold level. The pmdUpperHighThresh and pmdLowerLowThresh are 4-bit fields that set offset thresholds relative to pmdUpperLowThresh and pmdLowerHighThresh.

Each LSB of the threshold words corresponds to 1 dBFS. The threshold levels corresponding to the default PMD thresholds are −2 dBFS, −3 dBFS, −12 dBFS, and −16 dBFS for pmdUpper-HighThresh, pmdUpperLowThresh, pmdLowerHighThresh, and pmdLowerLowThresh, respectively.

The application programming interface (API) returns an error if pmdUpperLowThresh is set greater than (smaller signal) or equal to pmdLowerHighThresh.

Table 118. Parameter Definitions for hb2OverloadDetectEnable

Data Type	Parameter	Value	Note
uint8_t	hb2OverloadDetectEnable	0	HB2 overload detector disabled
		1 (default)	HB2 overload detector enabled

Table 119. Parameter Definitions for Fast Attack Settings

Data Type	Parameter	Value	Window Size for HB2 Overload
uint8_t	hb2OverloadDurationCnt	0	1
		1 (default)	4
		2	8
		3	12
		4	16
		5	24
		6	32
		>7	Invalid

Table 120. Parameter Limits and Default Values for PMD Thresholds

Data Type	Parameter	Bit Width	Default Value	Minimum Value	Maximum Value
uint8_t	pmdUpperHighThresh	4	1	0	15
uint8_t	pmdUpperLowThresh	7	3	0	127
uint8_t	pmdLowerHighThresh	7	12	0	127
uint8_t	pmdLowerLowThresh	4	4	0	15

PMD Gain Steps

Each of the power measurement detector (PMD) thresholds has a corresponding gain step. If the detected power is greater than the pmdUpperHighThresh or pmdUpperLowThresh threshold, a gain decrement is made according to pmdUpper-HighGainStepAttack or pmdUpperLowGainStepAttack. When both power thresholds are exceeded, the gain step is made according to pmdUpperHighGainStepAttack.

If the detected power is less than pmdLowerHighThresh or pmdLowerLowThresh, a gain increment is made according to pmdLowerHighGainStepRecovery or pmdLowerLow-GainStepRecovery. When both lower power thresholds are not exceeded, the gain step is made according to pmdLowerLowGainStepAttack.

Because the inner thresholds (pmdUpperLowThresh and pmdLowerHighThresh) are closer to the desired received signal power than the outer thresholds (pmdUpperHighThresh and pmdLowerLowThresh), it is recommended to make the inner gain steps (pmdUpperLowGainStepAttack and pmdLower-HighGainStepRecovery) smaller than the outer gain steps (pmdUpperHighGainStepAttack and pmdLowerLowGain-StepRecovery). This allows the gain to recover very quickly if the signal is far underrange, then slows the recovery as it approaches the desired received power.

PMD Measurement Duration

The measurement duration of the power measurement blocks (denoted by pmdMeasDuration) sets the number of samples that the power measurement blocks use to evaluate the received signal power level. This counter counts at either the IQ data rate (if receiver finite impulse response (RFIR) or Baseband DC 2 (BBDC2) is chosen as the measurement configuration) or at the RFIR input clock rate (see Figure 57 and Figure 58). The value of this member value must satisfy the following relationship with the agcGainUpdateCounter. Equation 13 assumes that the pmdMeasDuration is specified in IQ data rate clock cycles. Divide pmdMeasDuration by the RFIR decimation factor if the HB2 output is used.

$$agcGainUpdateCounter[21{:}0] > 8 \times 2^{pmdMeasDuration} \tag{13}$$

PMD Measurement Configuration

The measurement configuration of the power measurement blocks (denoted by pmdMeasConfig) determines where in the receiver signal chain the measurement takes place. There are three locations where decimated power measurements can occur: the output of the finite impulse response (RFIR), the output of the Half-Band 2 (HB2) filter, and the output of the BBDC2. This member also controls an enable bit that can disable the power measurement detector (PMD) block. See Table 123 for details.

Enabling the PMD at the RFIR output allows only the filtered data (after the RFIR) to be used to determine the power. In this mode, the measurement duration word (pmdMeasDuration) obeys Equation 13 because the sample rate is at the IQ data rate, which is the recommended setting for the PMD.

Enabling the PMD at the HB2 output increases the bandwidth of the power measurement approximately by a factor of 2, which allows more out of band blocker energy into the power measurement. When PMD is enabled, note that pmdMeasDuration is determined using the RFIR input clock rate, which is not necessarily the same as the IQ data rate. This refers to pmdMeasDuration at the input clock rate to the RFIR, not necessarily the IQ data rate.

Table 121. Parameter Limits and Default Values for PMD Gain Steps

Data Type	Parameter	Bit Width	Default Value	Minimum Value	Maximum Value
uint8_t	pmdUpperHighGainStepAttack	5	4	0	31
uint8_t	pmdUpperLowGainStepAttack	5	2	0	31
uint8_t	pmdLowerHighGainStepRecovery	5	2	0	31
uint8_t	pmdLowerLowGainStepRecovery	5	4	0	31

Table 122. Parameter Limits and Default Values for pmdMeasDuration

Data Type	Parameter	Default value	Minimum Value	Maximum Value
uint8_t	pmdMeasDuration	8	0	15

Table 123. Parameter Definitions for pmdMeasConfig

Data Type	Parameter	Value	Note
uint8_t	pmdMeasConfig	0	PMD disabled
		1	PMD enabled at the HB2 output
		2 (default)	PMD enabled at the RFIR output
		3	PMD enabled at the BBDC2 output

SUMMARY OF THE AGC PARAMETERS

This section provides a summary of all automatic gain control (AGC) data structure parameters, limits, and members. For information about the parameters and their meanings, refer to the previous sections in this user guide.

Table 124. Parameter Limits and Default Values for mykonosAgcCfg_t

Data Type	Parameter	Bit Width	Default Value	Minimum Value	Maximum Value
uint8_t	agcRx1MaxGainIndex	8	255	agcRx1MinGainIndex	255
uint8_t	agcRx1MinGainIndex	8	195	Minimum Rx1 table index	agcRx1MaxGainIndex
uint8_t	agcRx2MaxGainIndex	8	255	agcRx2MinGainIndex	255
uint8_t	agcRx2MinGainIndex	8	195	Minimum Rx2 table index	agcRx2MaxGainIndex
uint8_t	agcObsRxMaxGainIndex	8	255	agcObsRxMinGainIndex	255
uint8_t	agcObsRxMinGainIndex	8	203	Minimum ORx table index	agcObsRxMaxGainIndex
uint8_t	agcObsRxSelect	1	1	1	1
uint8_t	agcPeakThresholdMode	1	1	0	1
uint8_t	agcLowThsPreventGainIncrease	1	1	0	1
uint32_t	agcGainUpdateCounter	22	30720	1	0x3FFFFF
uint8_t	agcSlowLoopSettlingDelay	7	3	0	127
uint8_t	agcPeakWaitTime	5	2	2	31
uint8_t	agcResetOnRxEnable	1	0	0	1
uint8_t	agcEnableSyncPulseForGainCounter	1	0	0	1
	*mykonosPeakDetAgcCfg_t				
	*mykonosPowerMeasAgcCfg_t				

Table 125. Parameter Limits and Default Values for mykonosPeakDetAgcCfg_t

Data Type	Parameter	Bit Width	Default Value	Minimum Value	Maximum Value
uint8_t	apdHighThresh	6	31	apdLowThresh	63
uint8_t	apdLowThresh	6	22	0	apdHighThresh
uint8_t	hb2HighThresh	8	181	hb2LowThresh	255
uint8_t	hb2LowThresh	8	128	hb2VeryLowThresh	hb2HighThresh
uint8_t	hb2VeryLowThresh	8	64	0	hb2LowThresh
uint8_t	apdHighThreshExceededCnt	8	6	0	255
uint8_t	apdLowThreshExceededCnt	8	4	0	255
uint8_t	hb2HighThreshExceededCnt	8	6	0	255
uint8_t	hb2LowThreshExceededCnt	8	4	0	255
uint8_t	hb2VeryLowExceededCnt	8	4	0	255
uint8_t	apdHighGainStepAttack	5	4	0	31
uint8_t	apdLowGainStepRecovery	5	2	0	31
uint8_t	hb2HighGainStepAttack	5	4	0	31
uint8_t	hb2LowGainStepRecovery	5	2	0	31
uint8_t	hb2VeryLowGainStepRecovery	5	4	0	31
uint8_t	apdFastAttack	1	1	0	1
uint8_t	hb2FastAttack	1	1	0	1
uint8_t	hb2OverloadDetectEnable	1	1	0	1
uint8_t	hb2OverloadDurationCnt	3	1	0	7
uint8_t	hb2OverloadThreshCnt	4	1	0	15

Table 126. Parameter Limits and Default Values for mykonosPowerMeasAgcCfg_t

Data Type	Parameter	Bit Width	Default Value	Minimum Value	Maximum Value
uint8_t	pmdUpperHighThresh	4	1	0	15
uint8_t	pmdUpperLowThresh	7	3	0	pmdLowerHighThresh
uint8_t	pmdLowerHighThresh	7	12	pmdUpperLowThresh	127
uint8_t	pmdLowerLowThresh	4	4	0	15
uint8_t	pmdUpperHighGainStepAttack	5	4	0	31
uint8_t	pmdUpperLowGainStepAttack	5	2	0	31
uint8_t	pmdLowerHighGainStepRecovery	5	2	0	31
uint8_t	pmdLowerLowGainStepRecovery	5	4	0	31

DIGITAL GAIN COMPENSATION, SLICER, AND FLOATING POINT FORMATTER

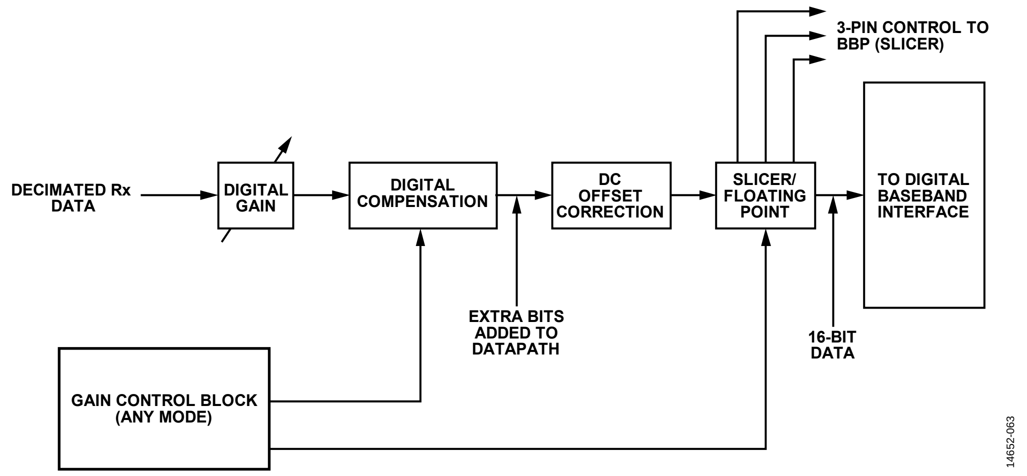

3-PIN CONTROL TO BBP (SLICER)
DECIMATED Rx DATA
DIGITAL GAIN
DIGITAL COMPENSATION
DC OFFSET CORRECTION
SLICER/ FLOATING POINT
TO DIGITAL BASEBAND INTERFACE
EXTRA BITS ADDED TO DATAPATH
16-BIT DATA
GAIN CONTROL BLOCK (ANY MODE)
14652-063

Figure 71. Gain Compensation, Floating Point Formatter, and Slicer Section of the Receiver Datapath

The device contains a digital gain compensation block that, if enabled, provides digital gain compensation to offset the gain reduction from the receiver. Gain compensation allows variation in the receiver gain to be transparent to the baseband processor (BBP), which can be useful in applications where the gain index may change quickly, leading to undesirable amplitude variations seen in the BBP that may cause problems with demodulation. Gain compensation is useful in an automatic gain control (AGC) application where the onset of an interferer can force a sharp and potentially quick reduction in the receiver gain. Without gain compensation, the BBP must request the current gain index to recover the input signal level from the device via an SPI command, which can take much longer than using gain compensation.

Gain compensation can be used in manual gain control (MGC) mode, hybrid mode, and AGC mode. The gain compensation block is capable of applying up to 31.75 dB of digital gain compensation, which is sufficient to compensate for the full attenuation range supported by the Rx datapath.

The gain compensation, slicer, and floating point formatter blocks are shown in Figure 71.

The bit width at the output of the digital gain compensation block is several bits wider than its input. This increase in bit width allows support for up to 31.75 dB of gain compensation.

The JESD204B data interface supports up to 16-bit data. Depending on the amount of compensation applied, the datapath may exceed the maximum or minimum value allowed by 16-bit signed integers. To accommodate the expanded bit width in the Rx datapath under gain compensation, two methods are available to send data to a BBP. These two methods are the slicer and the floating point formatter. The slicer requires three GPIO pins per receiver. The floating point formatter does not require GPIO pins.

Digital Gain Compensation

The digital gain compensation block is capable of fine adjustments in digital gain at a minimum resolution of 0.25 dB over a total compensation range of 31.75 dB. Gain compensation occurs only digitally. Gain compensation compensates for changes in receiver gain in MGC mode, hybrid mode, and AGC mode.

There are two gain table requirements for gain compensation to work properly:

- Gain table steps (dB) between adjacent indices must be uniform throughout the range of the table.
- The gain table step size must be one of the following options: 0.25 dB, 0.5 dB, 1 dB, 2 dB, 3 dB, 4 dB, or 6dB.

The digital gain compensation uses the programmable gain table step parameter (mykonosGainComp_t → compStep) and how many gain indices from the maximum gain index the receiver is operating at to set the digital gain compensation level. Figure 72 shows this behavior.

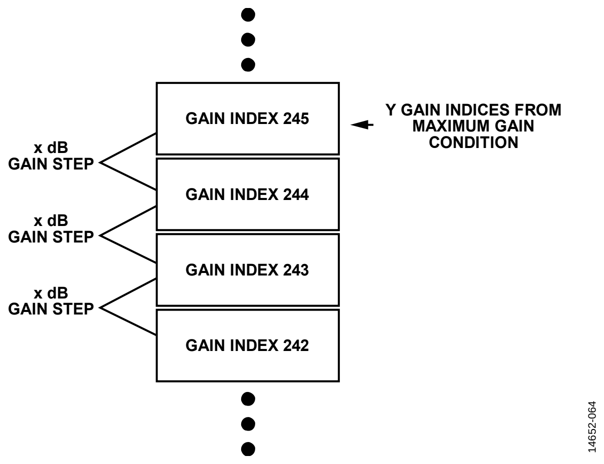


Figure 72. Gain Compensation Parameters for Setting the Digital Gain Compensation Level

In Figure 72, assume the maximum gain index is 255, and the gain table step size is 0.5 dB (the default gain table). In the case that the user selects Gain Index 245, the gain index choice results in 5 dB (10 × 0.5 dB) total attenuation. If gain compensation is enabled, the digital gain compensation adds 5 dB gain (x × y dB) in the digital datapath.

The gain compensation block is capable of compensating gain for an external attenuator; however, the table must conform to the dB step sizes previously listed.

When digital compensation is enabled, there are potentially 6 bits added into the datapath. Not all of these bits can fit into the 16-bit JESD204B datapath. The device features two methods to overcome this limitation: the slicer and the floating point formatter.

Slicer

The slicer allows 16-bit data transmission over the JESD204B interface along with a 3-bit shift value sent over three GPIO pins per receiver. Six GPIO pins are required if the application uses both Rx1 and Rx2. The slicer bit shifts data down prior to transmission over the JESD204B interface, such that the 16 MSBs of data are sent to the baseband processor (BBP).

Slicer Output Mode

The shift value, expressed over GPIO, informs the BBP how many bits the JESD204B data has shifted. The shift value depends on the amount of gain compensation applied. If the gain index is at its maximum condition, the shift value is zero because no shift in the data is necessary (no digital gain compensation is applied). If the gain index is reduced from the maximum gain condition by 1 LSB (for example, 255 to 254), the shift value is set to 1 to indicate to the BBP that the JESD204B data must be left shifted by 1 bit to recover the original value. When the slicer position is 1, 1 LSB is not sent to the BBP.

The shift position of 1 is shown in Figure 73. When the shift position is 1, bits above B17 are not used. Using the default gain table and gain compensation configuration, this scenario occurs when the gain index is set between 254 and 243, or when gain compensation is greater than 0 dB and less than 6 dB.

Table 127 provides information regarding how the amount of gain compensation affects the slicer position, the shift value, and the value expressed over the three GPIO slicer pins. This information assumes that the default gain tables are used and that the gain compensation block is programmed correctly. The slicer position indicates the reduction, in dB, of the data prior to transmission over the JESD204B interface. The shift value indicates the number of bits to shift the data in the BBP to recover the original data. Recall that the amount of gain compensation per LSB from maximum gain condition is programmable via the mykonosGainComp_t → compStep parameter.

The BBP monitors the state of the slicer pins to shift the data to the original signal level. These pins are listed as follows. Note that no other GPIO pins can be used to indicate the slicer position. The pins must be set to GPIO_SLICER_OUT_MODE, and the output must be enabled by the GPIO configuration for proper operation. The following groupings are listed from MSB to LSB:

- For Rx1, use GPIO10, GPIO9, and GPIO8.
- For Rx2, use GPIO14, GPIO13, and GPIO12.
- For ORx, use GPIO18, GPIO17, and GPIO16.

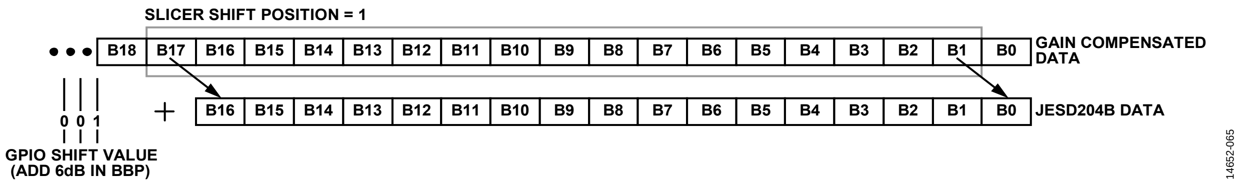

SLICER SHIFT POSITION = 1
B18 B17 B16 B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
GAIN COMPENSATED DATA
B16 B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
JESD204B DATA
0 0 1
GPIO SHIFT VALUE (ADD 6dB IN BBP)
14652-065

Figure 73. Mapping of Data from Digital Gain Compensation to JESD204B Interface with Slicer Position = 1

Table 127. GPIO Output State for Slicer Position

Gain Compensation (dB)	Slicer Position (dB)	Shift Value	Rx1/Rx2 Slicer GPIO Value (3-Bit)
0	0	0	0
0.25 to 5.75	6	1	1
6 to 11.75	12	2	2
12 to 17.75	18	3	3
18 to 23.75	24	4	4
24 to 29.75	30	5	5
30 to 31.75	36	6	6

Slicer Input Mode

Alternatively, the slicer is able to take an input shift value over the GPIO pins and shift the data as desired to allow the baseband processor (BBP) to set a desired digital gain instead of letting the slicer position and gain compensation be determined by the internal gain control block. When gain compensation is enabled, the user must call the MYKONOS_setSlicerCtrl(…) command to use this functionality and assign the GPIO pins for this purpose. The user is able to set a slicer gain step for each LSB over the three slicer input pins of 1 dB, 2 dB, 3 dB, and 4 dB.

The valid GPIO pins for the slicer input mode are configurable and include the pin groupings as follows (groupings are listed from MSB to LSB):

- For Rx1, use the GPIO2, GPIO1, GPIO0 group, the GPIO7, GPIO6, GPIO5 group, and the GPIO10, GPIO9, GPIO8 group.
- For Rx2, use the GPIO7, GPIO6, GPIO5 group and the GPIO13, GPIO12, GPIO11 group.
- For ORx, use the GPIO18, GPIO17, GPIO16 group and the GPIO16, GPIO15, GPIO14 group.

Programming information is included in the API Support for Gain Compensation, Slicer, and Floating Point Formatter section.

Floating Point Formatter

The floating point formatter offers an alternative method to reduce the digital gain compensation output into 16-bit data that can be sent over the JESD204B link. The floating point formatter is located prior to the JESD204B interface on the receivers to minimize floating point arithmetic in the receiver digital datapath. Representing the gain compensation output in a 16-bit floating point results in a slight loss of resolution. To minimize the loss of resolution, multiple modes are included in the device that are modifications to the IEEE 754 half precision binary floating point format (binary16).

Figure 74 shows the representation for the binary16 floating point number. In Figure 74, w is the bit width of the exponent, and t is the bit width of the significand. The exponent is stored after adding a bias to the actual exponent value. The user is able to switch the order from (sign, exponent, significand) to (sign, significand, exponent) if desired.

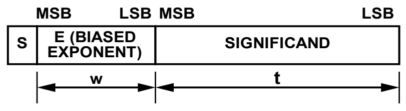


Figure 74. Floating Point Number Representation

The total precision for the significand is p = t + 1. If t = 10, that means 10 bits of the significand are stored explicitly and 1 bit is a sign bit, leading to 11 bits of significand precision.

For example, if the exponent is set as e, the stored exponent is E = e + bias. The significand precision is p and the significand itself varies between 1.0 and 1 + $2^{(1-p)}$ × t. The value of the floating point number (Value) is represented by the following equation. This formula is important to decode information on the BBP side of the JESD204B link.

$$Value = (-1)^S \times 2^{E-bias} \times (1 + 2^{1-p} \times t) \quad (14)$$

where *S* is the sign bit (1 or 0).

The numbers have an implicit leading significand of 1 unless E = 0 and t = 0. In this case, the number is a signed 0.

If E = 0 and t ≠ 0, the number is referred to as a subnormal number, and the value is instead found by the following equation:

$$Value = (-1)^S \times 2^{e\,min} \times (0 + 2^{1-p} \times t) \quad (15)$$

The device allows support for several different formats that conform to the IEEE 754 standard and other formats that do not adhere to the IEEE 754 standard, called Analog Devices modes. These modes are described in Table 128.

Note that the first column in Table 128 includes the value of the parameter leading the floating point formatter data structure, mykonosFloatPntFrmt_t. This data structure type is described in the Floating Point Formatter section. Table 128 shows that the Analog Devices modes of operation allow an increased maximum value of the exponent.

The user has direct control over the first column (leading) and the second column (bit width of exponents). Selecting a desired w value sets the bias for the exponent.

Table 128. Floating Point Formatter—IEEE 754 Modes Supported

Analog Devices/IEEE Mode (Leading)	w (Bit Width of Exponent)	t (Bit Width of Significand)	Precision (p)	Bias (E – e)	Range of e Min	Range of e Max
IEEE (1)	5	10	11	15	−14	+15
IEEE (1)	4	11	12	7	−6	+7
IEEE (1)	3	12	13	3	−2	+3
IEEE (1)	2	13	14	1	0	+1
Analog Devices (0)	5	10	11	15	−15	+16
Analog Devices (0)	4	11	12	7	−7	+8
Analog Devices (0)	3	12	13	3	−3	+4
Analog Devices (0)	2	13	14	1	−1	+2

API Support for Gain Compensation, Slicer, and Floating Point Formatter

The application programming interface (API) allows configuration of the gain compensation, slicer, and floating point formatter. Important data structures and their members are outlined in this section. API commands are also described throughout this section. These commands are necessary to configure gain compensation, the slicer, or the floating point formatter.

A programming flowchart is provided in Figure 75. This diagram begins where Figure 68 ends, at the completion of gain control setup. The diagram does not explicitly mention the configuration of the GPIO pins. However, if the user sets up the GPIO pins in the device data structure prior to the initialization sequence, the pins are still configured according to the device data structure.

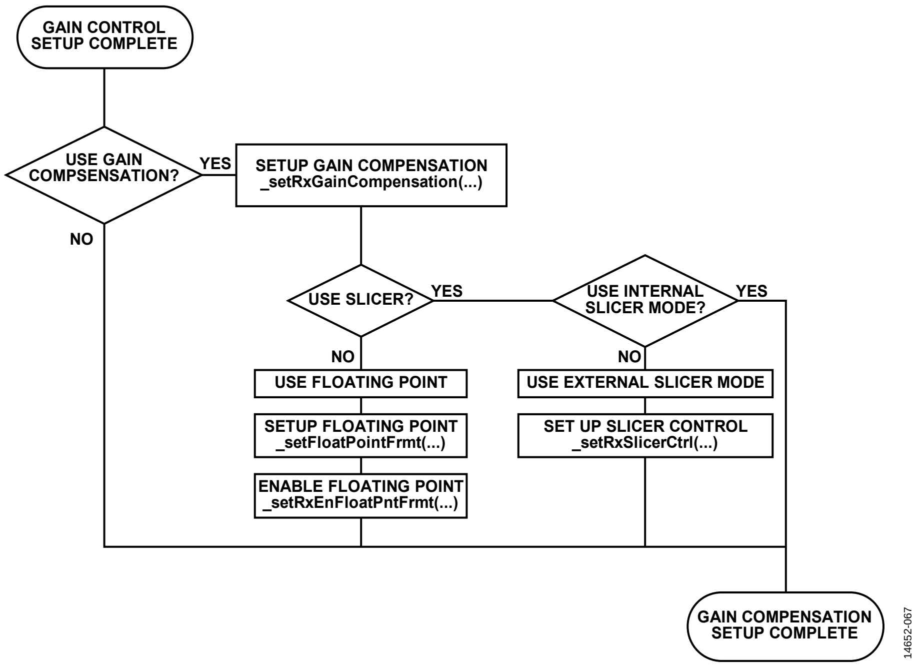

GAIN CONTROL SETUP COMPLETE
USE GAIN COMPSENSATION?
YES
NO
SETUP GAIN COMPENSATION _setRxGainCompensation(...)
USE SLICER?
YES
NO
USE INTERNAL SLICER MODE?
YES
NO
USE FLOATING POINT
SETUP FLOATING POINT _setFloatPointFrmt(...)
ENABLE FLOATING POINT _setRxEnFloatPntFrmt(...)
USE EXTERNAL SLICER MODE
SET UP SLICER CONTROL _setRxSlicerCtrl(...)
GAIN COMPENSATION SETUP COMPLETE
14652-067

Figure 75. Rx Gain Compensation Programming Flowchart

Table 129. Parameter Limits and Default Values for mykonosGainComp_t

Data Type	Parameter	Bit Width	Default Value	Comments	
uint8_t	rx1Offset	5	0	This parameter contains the Rx1 offset word used for the gain compensation when the gain index is at its maximum setting. This parameter ranges from 0 to 0x1F with a resolution of 0.5 dB/LSB.	
uint8_t	rx2Offset	5	0	This parameter contains the Rx2 offset word used for the gain compensation when the gain is at its maximum setting. This parameter ranges from 0 to 0x1F with a resolution of 0.5 dB/LSB.	
uint8_t	compStep	3	1	This parameter sets the value (in dB) that gain compensation applies to an LSB change in the gain index according to the following settings.	
				compStep	**dB Step (dB)**
				0	0.25
				1	0.5
				2	1
				3	2
				4	3
				5	4
				6	6

Gain Compensation Data Structure

The configuration parameters for the gain compensation block are set up in a data structure of type mykonosGainComp_t. The members of the data structure are described in Table 129.

The data structure mykonosGainComp_t is not a part of the device data structure and does not need to be instantiated if gain compensation is not used in the user application.

Gain Compensation API Commands

The application programming interface (API) uses the MYKONOS_setRxGainCompensation(…) command to configure internal device registers for a desired gain compensation configuration. This command does not determine whether the slicer or the floating point formatter is used. The command configures the gain compensation block for user values determined by the data structure type mykonosGainComp_t. A description of the mykonosGainComp_t data type is provided in Table 129.

MYKONOS_setRxGainCompensation(…)

```
mykonosErr_t
   MYKONOS_setRxGainCompensation(mykonosDevi
   ce_t *device, mykonosGainComp_t
   *gainComp, uint8_t enable)
```

Description

MYKONOS_setRxGainCompensation(…) is the gain compensation enable and setup function.

The gain compensation block is a function that compensates for the attenuation in the internal attenuator for the Rx channels.

Preconditions

The gain control setup must be complete.

Parameters

- *device: This is a pointer to the device data structure.
- gainComp: This is a data structure containing the gain compensation settings.
- enable: this parameter enables or disables the gain compensation block (enable = 1 and disable = 0).

MYKONOS_getRxGainCompensation(…)

```
mykonosErr_t
   MYKONOS_getRxGainCompensation(mykonosDevi
   ce_t *device, mykonosGainComp_t
   *gainComp, uint8_t enable)
```

Description

This function obtains the gain compensation setup and enabled function.

The gain compensation block is a function that compensates for the attenuation in the internal attenuator for the Rx channels. This function obtains the current setup and the enable state of the block.

Preconditions

The gain control setup must be complete.

Parameters

- *device: This is a pointer to the device data structure.
- *gainComp: This is a pointer to a mykonosGainComp_t structure which holds the current device gain compensation settings.
- enable: This is a pointer to a parameter containing the enable state of the gain compensation block (enabled = 1 and disabled = 0).

SLICER API COMMANDS AND GPIO INFORMATION

The slicer can be configured in two different ways. The first method allows the slicer to determine its own position based on the receiver gain index and output the slicer position value over GPIO pins. The second method allows the baseband processor (BBP) to control the slicer position over the GPIO pins. Both methods require the gain compensation block to be enabled for proper slicer functionality. There is no API data structure specific to the slicer setup.

- If the user desires the BBP to read the slicer position over the GPIO pin, and to use that position information to appropriately shift the data, specific GPIO pins must be set as outputs in the proper mode. The Rx1 channel uses GPIO_10 to GPIO_8 to output the slicer position. The Rx2 channel uses GPIO_14 to GPIO_12 to output the slicer position. No other GPIO pins can be used to indicate the slicer position. To configure the device to output the slicer position over the GPIO signals, ensure that the GPIO pins are set as outputs and that the mykonosGpioMode_t for GPIO_11 to GPIO_8 and GPIO_15 to GPIO_12 are set to GPIO_SLICER_OUT_MODE.
- If the user desires the BBP to control the slicer position, the desired GPIO pins must be set for input control and the command listed in the following section must be run. This command configures specific sets of GPIO pins as inputs to the slicer (rx1Pins, rx2Pins), sets the step size of the slicer when external pin control mode is enabled (slicerStep), and enables or disables the external pin control feature (enable). Refer to the following section for the valid pin configurations. Set the mykonosGpioMode_t for GPIO_BITBANG_MODE.

MYKONOS_setRxSlicerCtrl(…)

```
mykonosErr_t
   MYKONOS_setRxSlicerCtrl(mykonosDevice_t
   *device, uint8_t slicerStep,
   mykonosRxSlicer_t rx1Pins,
   mykonosRxSlicer_t rx2Pins, uint8_t
   enable);
```

Description

This function is the slicer control over the GPIO inputs.

The user can control the slicer position via three GPIO inputs per channel. There are various configurations for the GPIO pins, and these configurations are enumerated in the mykonosRxSlicer_t.

Preconditions

Configure the gain control.

Parameters

- *device: This is a pointer to the device data structure.
- slicerStep: The slicer configuration command also allows the user to set the slicer step size (slicerStep). The slicer gain is equal to the 3-bit word expressed on the GPIO inputs multiplied by the step size. Table 130 shows the relationship between the slicer step size and slicerStep parameter value.

Table 130. slicerStep Parameter Related to dB Steps in the Slicer

slicerStep	dB Step (dB)
0	1
1	2
2	3
3	4

- Rx1Pins: The value of the mykonosRxSlicer_t enumeration determines which grouping of three GPIO pins are used as inputs to the device to set the 3-bit slicer position. The valid pin groupings are listed as follows for the Rx1 channel, from MSB to LSB, with the mykonosRxSlicer_t enumeration value listed in parentheses:
  - GPIO_2, GPIO_1, and GPIO_0 (GPIO_0_1_2).
  - GPIO_7, GPIO_6, and GPIO_5 (GPIO_5_6_7).
  - GPIO_10, GPIO_9, and GPIO_8 (GPIO_8_9_10).
- Rx2Pins: The value of the mykonosRxSlicer_t enumeration determines which grouping of three GPIO pins are used as inputs to the device to set the 3-bit slicer position. The valid pin groupings are listed as follows for the Rx2 channel, from MSB to LSB, with the mykonosRxSlicer_t enumeration value listed in parentheses:
  - GPIO_7, GPIO_6, and GPIO_5 (GPIO_5_6_7).
  - GPIO_10, GPIO_9, and GPIO_8 (GPIO_8_9_10).
- enable: set enable = 1 to enable the external pin control for slicer. Set enable = 0 to disable external pin control for the slicer.

A get version of this command is described in the following section.

MYKONOS_getRxSlicerCtrl (…)

```
mykonosErr_t
   MYKONOS_getRxSlicerCtrl(mykonosDevice_t
   *device, uint8_t *slicerStep,
   mykonosRxSlicer_t *rx1Pins,
   mykonosRxSlicer_t *rx2Pins, uint8_t
   *enable);
```

Description

This function obtains the programmed slicer control for the Rx1 and Rx2 channels.

The user can control the slicer position via three GPIO inputs per channel. There are various configurations for the GPIO pins, and these configurations are enumerated in the mykonosRxSlicer_t.

Preconditions

Configure the gain control.

Parameters

- *device: This is a pointer to the device data structure.
- slicerStep: This contains the configured step size.
- rx1Pins: This contains the configured GPIO combination for Rx1.
- rx2Pins: This contains the configured GPIO combination for Rx2.
- *enable: This contains the programmed enable setting.

The slicer defaults to the first mode (internal mode) of operation when gain compensation is enabled. The slicer is disabled if gain compensation is disabled. Note that enabling gain compensation does not configure the GPIO pins.

Floating Point Data Structure

The configuration parameters for the floating point formatter are set up in a data structure of type mykonosFloatPntFrmt_t. The members of the data structure are described in Table 131.

Table 131. Parameter Limits and Default Values for mykonosFloatPntFrmt_t

Data Type	Parameter	Bit Width	Default Value	Comments	
uint8_t	roundMode	2	0	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:	
				roundMode	**Round Mode**
				0	Round ties to even
				1	Round towards positive
				2	Round towards negative
				3	Round towards 0
				4	Round ties to away
uint8_t	dataFormat	1	0	This parameter sets the format of the 16-bit output on the JESD204B interface.	
				dataFormat	**Format**
				0	MSB to LSB (sign, exponent, significand)
				1	MSB to LSB (sign, significand, exponent)

Data Type	Parameter	Bit Width	Default Value	Comments			
uint8_t	encNan	1	0	If this parameter is set to 1, then the floating point formatter reserves the highest value of exponent for NaN (not a number) to be compatible with the IEEE754 specification. Setting this parameter to 0 increases the range of the exponent by 1.			
uint8_t	expBits	2	2	This parameter is used to indicate the number of exponent bits in the floating point number according to the following settings.			
				expBits	**No. of Exponent Bits**	**No. of Significand Bits**	**No. of Sign Bits**
				0	2	13	1
				1	3	12	1
				2	4	11	1
				3	5	10	1
uint8_t	leading	1	1	Setting this parameter to 1 hides the leading one in the significand to be compatible to the IEEE754 specification (IEEE mode). Clearing this parameter causes the leading one to be at the MSB of the significand (Analog Devices mode).			

FLOATING POINT API COMMANDS

The floating point formatter uses several application programming interface (API) commands that configure the floating point formatter, enable the Rx floating point formatter, and enable the ORx floating point formatter. The floating point formatter uses the data structure of type mykonosFloatPntFrmt_t to store configuration parameters.

To set up the configuration parameters of the floating point formatter for Rx1, Rx2 and ORx, use the command described in the following section.

MYKONOS_setFloatPointFrmt (…)

```
mykonosGpioErr_t
   MYKONOS_setFloatPointFrmt(mykonosDevice
   _t *device,mykonosFloatPntFrmt_t
   *floatFrmt);
```

Description

Floating point formatter enable and setup function.

The floating point formatter block is a function that works in conjunction with the gain compensating block, as the gain compensation requires increased dynamic range which increases the bit width in the digital datapath.

Preconditions

Configure the gain control.

Parameters

- *device: This is a pointer to the device data structure.
- *floatFrmt: a mykonosFloatPntFrmt_t data structure containing floating point formatter settings.

A get version of this command is described in the following section.

MYKONOS_getFloatPointFrmt(…)

```
mykonosGpioErr_t
   MYKONOS_setFloatPointFrmt(mykonosDevice
   _t *device,mykonosFloatPntFrmt_t
   *floatFrmt);
```

Description

MYKONOS_getFloatPointFrmt(…) is the floating point formatter setup function. This command obtains the programmed floating point formatter settings.

The floating point formatter block is a function that works in conjunction with the gain compensating block, as the gain compensation requires increased dynamic range, which increases the bit width in the digital datapath.

Preconditions

Configure the gain control.

Parameters

- *device: This is a pointer to the device data structure.
- *floatFrmt: a mykonosFloatPntFrmt_t data structure containing floating point formatter settings.

The following commands enable the floating point formatter. The floating point formatter has separate enable commands for Rx1/Rx2 and the ORx.

MYKONOS_setRxEnFloatPointFrmt (…)

```
mykonosGpioErr_t
   MYKONOS_setFloatPointFrmt(mykonosDevice
   _t *device, uint8_t rx1Att, uint8_t
   rx2Att, uint8_t enable)
```

Description

MYKONOS_setRxEnFloatPointFrmt (…) is the floating point formatter enable/disable function for Rx1 and Rx2.

The floating point formatter block is a function that works in conjunction with the gain compensating block, as the gain compensation requires increased dynamic range, which increases the bit width in the digital datapath.

Preconditions

Configure the gain control.

Parameters

- *device: This is a pointer to the device data structure.
- rx1Att: This parameter sets the integer data attenuation for the Rx1 channel in 6 dB steps to enable the entire data range to be represented in the selected floating point format.
- rx2Att: This parameter sets the integer data attenuation for the Rx2 channel in 6 dB steps to enable the entire data range to be represented in the selected floating point format.
- enable: This parameter enables or disables the gain compensation block (enable = 1 and disable = 0).

A get version of this command is described in the following section.

MYKONOS_getRxEnFloatPointFrmt(…)

```
mykonosGpioErr_t
   MYKONOS_getRxEnFloatPointFrmt(mykonosDe
   vice_t *device, uint8_t *rx1Att,
   uint8_t *rx2Att, uint8_t *enable)
```

Description

MYKONOS_getRxEnFloatPointFrmt(…) is the floating point formatter readback function for Rx1 and Rx2.

The floating point formatter block is a function that works in conjunction with the gain compensating block, as the gain compensation requires increased dynamic range, which increases the bit width in the digital datapath.

Preconditions

Configure the gain control.

Parameters

- *device: This is a pointer to the device data structure.
- *rx1Att: This parameter sets the integer data attenuation for the Rx1 channel in 6 dB steps to enable the entire data range to be represented in the selected floating point format.
- *rx2Att: This parameter sets the integer data attenuation for the Rx2 channel in 6d B steps to enable the entire data range to be represented in the selected floating point format.
- *enable: This parameter enables or disables the gain compensation block (enable = 1 and disable = 0).

MYKONOS_setOrxEnFloatPointFrmt (…)

```
mykonosGpioErr_t
   MYKONOS_setOrxEnFloatPointFrmt(mykonosD
   evice_t *device, uint8_t  orxAtt,
   uint8_t enable)
```

Description

MYKONOS_setOrxEnFloatPointFrmt (…) is the floating point formatter enable/disable for the ORx channel.

The floating point formatter block is a function that works in conjunction with the gain compensating block, as the gain compensation requires increased dynamic range, which increases the bit width in the digital datapath.

Preconditions

Configure the gain control.

Parameters

- *device: This is a pointer to the device data structure.
- orxAtt: this parameter sets the integer data attenuation for the Rx1 channel in 6 dB steps to enable the entire data range to be represented in the selected floating point format.
- enable: This parameter enables or disables the gain compensation block (enable = 1 and disable = 0).

A get version of this command is noted in the following section.

MYKONOS_getOrxFloatPointFrmt(…)

```
mykonosGpioErr_t
   MYKONOS_setFloatPointFrmt(mykonosDevice
   _t *device, uint8_t rx1Att, uint8_t
   rx2Att, uint8_t enable)
```

Description

MYKONOS_getOrxFloatPointFrmt(…) is the floating point formatter enable/disable Rx1 and Rx2 function.

The floating point formatter block is a function that works in conjunction with the gain compensating block, as the gain compensation requires increased dynamic range which increases the bit width in the digital datapath.

Preconditions

Configure the gain control.

Parameters

- *device: This is a pointer to the device data structure.
- rx1Att: This parameter sets the integer data attenuation for the Rx1 channel in 6 dB steps to enable the entire data range to be represented in the selected floating point format.
- rx2Att: This parameter sets the integer data attenuation for the Rx2 channel in 6 dB steps to enable the entire data range to be represented in the selected floating point format.
- enable: This parameter enables or disables the gain compensation block (enable = 1 and disable = 0).

FILTER CONFIGURATION

This section describes the digital filters within the integrated transceiver. Descriptions of the main receivers, transmitters and the observation/sniffer receiver system filters are provided. Also described in this section is an overview of the application programming interface (API) data structures and commands necessary to configure the digital filters for proper operation.

Analog Devices uses profiles to designate different device configuration settings for the Tx, Rx, ORx, and SnRx channels. When selecting a profile, note that Rx1 and Rx2 use the same profile; Tx1 and Tx2 use the same profile; ORx1 and ORx2 use the same profile; and SnRxA, SnRxB, and SnRxC use the same profile. The profile dictates how the digital filters, analog filters, clock rates, and clock dividers are configured in the device. Some specific parameters set by the profiles include the IQ data rate, ADC clock rate, analog filter corners, FIR filter coefficients, and interpolation/decimation factors in the half-band filters.

Several profiles can be examined in the transceiver evaluation software (TES) for given device clock frequencies. If the desired profile exists in the software, it is recommended to set up the desired profile in and use the data structure values generated by the **Create Config.c** file for the Tx, Rx, ORx, and SnRx profile data structures. Custom profiles can be generated using other Analog Devices software tools that are not described in this section.

RECEIVER SIGNAL PATH

The main receivers have independent signal paths for the Rx1 and Rx2 ports. Each receiver signal path consists of an adjustable analog transimpedance low-pass filter, a Σ-Δ ADC, and digital decimation filters. The fixed coefficient decimation filters (RHB1, RHB2, RHB3, DEC5, and DEC5HR) are designed to eliminate overranging. The programmable receiver FIR filter (RFIR) in the Rx digital baseband path can overrange, depending on coefficients. However, the RFIR output code is limited to a maximum code value when overrange conditions occur.

A block diagram of the Rx1 and Rx2 datapath is shown in Figure 76. Quadrature error correction (QEC), dc offset correction, and digital gain are not described in this section. The following sections describe the functionality of the digital and analog filters and their configurations.

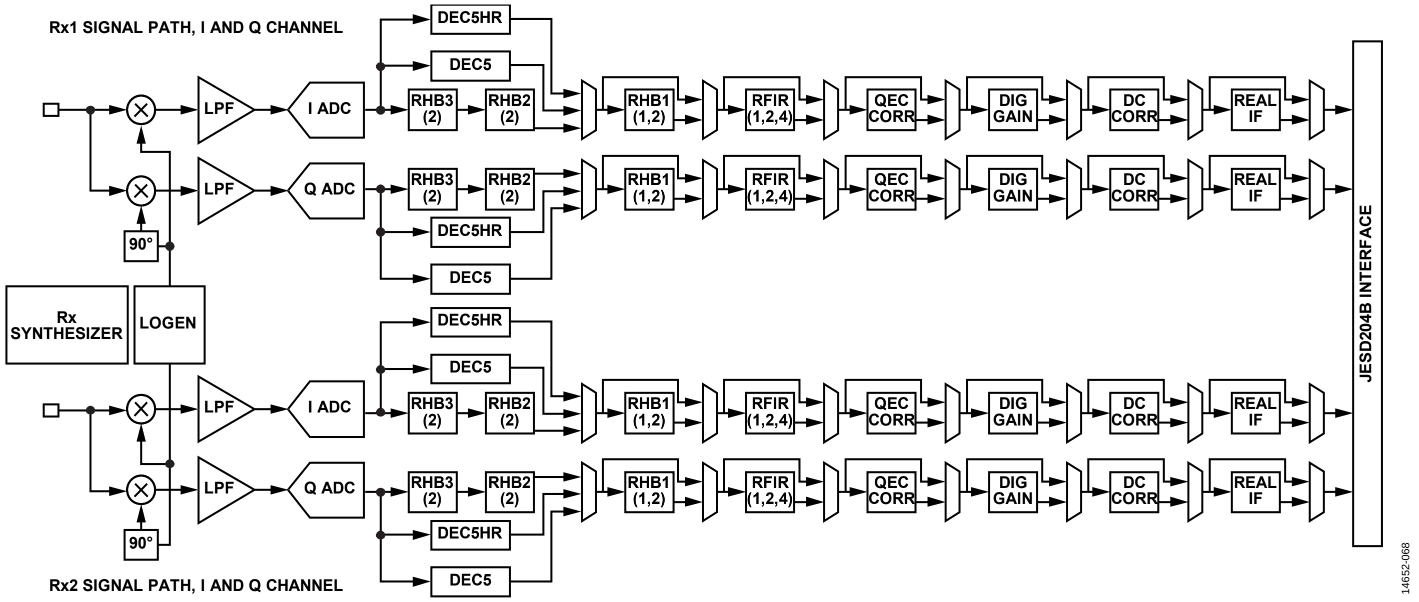


Figure 76. Rx1 and Rx2 Signal Path Diagram

Low-Pass Filter

The Rx low-pass filter (LPF) is a transimpedance amplifier (TIA) with a single, real pole frequency response. The 3 dB bandwidth of the TIA LPF ranges from 20 MHz to 100 MHz. The TIA LPF is calibrated based on the 3 dB bandwidth, resulting in 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 analog TIA LPF pass band is compensated by the RFIR to ensure a maximally flat pass-band frequency response.

The LPF bandwidth is set in the device data structure at device → rx → rxProfile → rxBbf3dBCorner_kHz.

DEC5/DEC5HR

The main Rx signal path features an option to use either the decimate by 5 (DEC5), the decimate by 5 high rejection (DEC5HR), or the series combination of the Receiver Half-Band 3 (RHB3) and Receiver Half-Band 2 (RHB2) digital decimation filters. The difference between the DEC5HR and the DEC5 digital filters is that DEC5HR features a higher stop-band rejection than DEC5. Both DEC5 and DEC5HR are fixed coefficient, decimating filters. Full scale for the DEC5 digital filter is 8192 (2^{13}). Full scale for the DEC5HR is 32768 (2^{15}).

Coefficients for the DEC5HR filter are [−64, −165, −305, −442, −499, −273, +280, +1208, +2433, +3762, +4866, +5503, +5503, +4866, +3762, +2433, +1208, +280, −273, −499, −442, −305, −165, −64].

Coefficients for the DEC5 filter are [+18, +35, +56, +72, +70, +28, −38, −126, −209, −244, −184, −20, +256, +612, +976, +1273, +1448, +1448, +1273, +976, +612, +256, −20, −184, −244, −209, −126, −38, +28, +70, +72, +56, +35, +18].

The choice between DEC5 and DEC5HR is set in the device data structure in device → rx → rxProfile → enHighRejDec5. This parameter is typically set to 1 to enable the DEC5HR filter.

The decimation factor for the first filter after the ADC is set in the device data structure in device → rx → rxProfile → rxDec5Decimation. Set this parameter as 5 to use the DEC5 or DEC5HR path.

RHB3

The Receive Half-Band 3 (RHB3) filter is a fixed coefficient decimating filter. RHB3 decimates by a factor of 2. The full scale for this filter is 16 (2^4). The RHB3 coefficients are [1, 4, 6, 4, 1].

RHB2

The Receive Half-Band 2 (RHB2) filter is a fixed coefficient decimating filter. RHB2 decimates by a factor of 2. The series combination of RHB2 and RHB3 can be bypassed using the DEC5 or DEC5HR filter. The full scale for this filter is 128 (2^7). The RHB2 coefficients are [+1, +0, −7, 0, +38, +64, +38, 0, −7, 0, +1].

The decimation factor for the first filter after the ADC is set in the device data structure in device → rx → rxProfile → rxDec5Decimation. Set this parameter as 4 to use the series combination of the RHB3 and RHB2 filters.

RHB1

The Receiver Half Band 1 (RHB1) filter is a fixed coefficient decimating filter. RHB1 can decimate by a factor of 2, or it can be bypassed. The full scale for this filter is 16384 (2^{14}). The RHB1 coefficients are [+9, 0, −41, 0, +124, 0, −304, 0, +665, 0, −1473, 0, +5074, +8108, +5074, 0, −1473, 0, +665, 0, −304, 0, +124, 0, −41, 0, +9].

The decimation factor for RHB1 is set in the device data structure in device → rx → rxProfile → rxDec5Decimation. This can be either 1 (bypass) or 2 (decimate by 2).

Programmable Receiver Finite Impulse Response (RFIR) Filter

The programmable RFIR filter acts as a decimating filter. The RFIR can decimate by a factor of 1, 2, or 4, or it can be bypassed. The RFIR can use a configurable number of taps from 24 taps to 72 taps in multiples of 24. The RFIR is typically used to compensate for the roll-off of the analog TIA LPF and decimating filters.

The maximum number of taps is limited by the FIR clock rate (data processing clock, DPCLK). The maximum DPCLK is 400 MHz. The DPCLK is the ADC clock rate divided by either 2 or 4 to limit the DPCLK below 400 MHz when the DEC5 and DEC5HR are disabled. The DPCLK is the ADC clock rate divided by 5 or 10 to limit the DPCLK below 400 MHz when DEC5 or DEC5HR are enabled.

$$\text{Maximum Number of Rx FIR Filter Taps} = (DPCLK/Rx\ IQ\ Data\ Rate) \times 24 \tag{16}$$

The RFIR coefficients are stored in the device data structure in device → rx → rxProfile → *rxFir, which is a pointer to a structure with data type mykonosFir_t.

The RFIR decimation factor is set in the device data structure in device → rx → rxProfile → rxFirDecimation, which can be either 1 (bypass), 2, or 4.

Real IF

The real IF block contains an interpolating filter and upconversion mixer used to convert complex signals centered around dc into real valued signals centered around an offset IF frequency. The real IF conversion block is typically bypassed, but it does provide the capability to operate at an IF frequency near baseband for systems that prefer to perform complex demodulation in their digital baseband. The full-scale value for the real IF interpolating filter is 16384 (2^{14}). The filter coefficients for the real IF filter are [−3, 0, +8, 0, −19, 0, +40, 0, −75, 0, +130, 0, −214, 0, +336, 0, −514, 0, +773, 0, −1169, 0, +1845, 0, −3327, 0, +10380, +16384, +10380, 0, −3327, 0, +1845, 0, −1169, 0, +773, 0, −514, 0, +336, 0, −214, 0, +130, 0, −75, 0, +40, 0, −19, 0, +8, 0, −3].

The real IF mode can be enabled in the device data structure in device → rx → realIfData, which can be either 0 (disable real IF) or 1 (enable real IF).

Rx SIGNAL PATH EXAMPLE

The transceiver evaluation software (TES) provides examples depicting how the baseband filtering stages are used for particular profiles. In this example, the Rx = 100 MHz, the IQ rate = 122.88 MHz, and the DEC5 profile is selected for the Rx channels. This profile is compatible with the other examples provided in this user guide.

Descriptions of the profile name conventions are as follows:

- Rx 100 MHz, which implies a RF (complex) receiver bandwidth of 100 MHz. In Figure 77 and Figure 78, note that the profile pass band is set to 50 MHz because the real IF mode is disabled and the received data is centered around dc. The filter responses are also symmetrical around dc. Only the positive half of the spectrum is shown. The pass band extends from −50 MHz to +50 MHz in the figure, which corresponds to RF frequencies of ±50 MHz from the Rx LO frequency.
- IQ Rate 122.88 MHz refers to the IQ data rate across the JESD204B interface.
- DEC5 string determines which profiles are compatible with each other. A given Rx (or Tx, or SnRx, or ORx) profile can only be used if all profiles (Rx, Tx, SnRx, ORx) have the same tag in the profile name string. Constraints in digital clocking prevent using DEC5 profiles with non DEC5 profiles.

Figure 79 shows the filter configuration for this profile, excluding correction stage blocks such as dc offset correction. Note that the clocking frequencies are in blue. The signal rate after the RFIR block is equal to the IQ data rate of the profile.

Also available in the transceiver evaluation software (TES) **Rx Summary** tab is the frequency response of the analog transimpedance amplifier (TIA) low-pass filter (LPF), digital filters, the ADC transfer function, and the composite response from dc to the sampling rate of the ADC (see Figure 77).

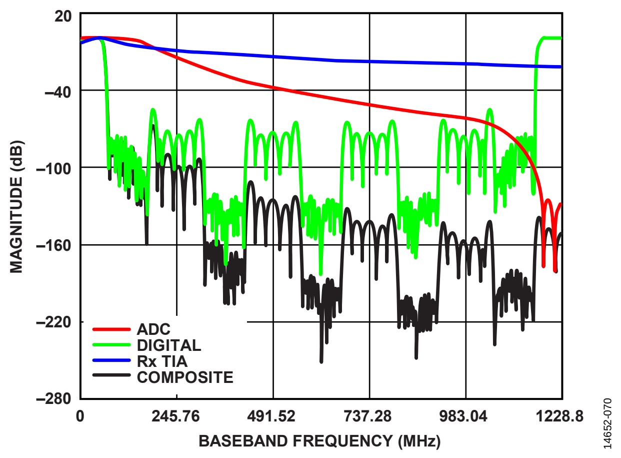


Figure 77. Main Receiver Filter Responses

An examination of the profile pass-band frequency shows that the Rx TIA 3 dB setting slightly attenuates information within the pass band. This analog attenuation is compensated by the digital filter response to obtain a maximally flat pass band for this profile. A zoom in view of the pass band is shown in Figure 78.

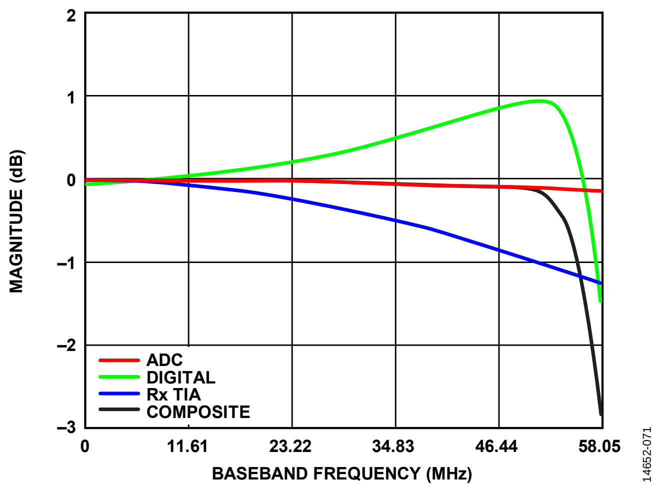


Figure 78. Pass-Band Frequency Response of the Rx = 100 MHz, 122.88 MHz, DEC5 Profile (Pass Band Zoom In View)

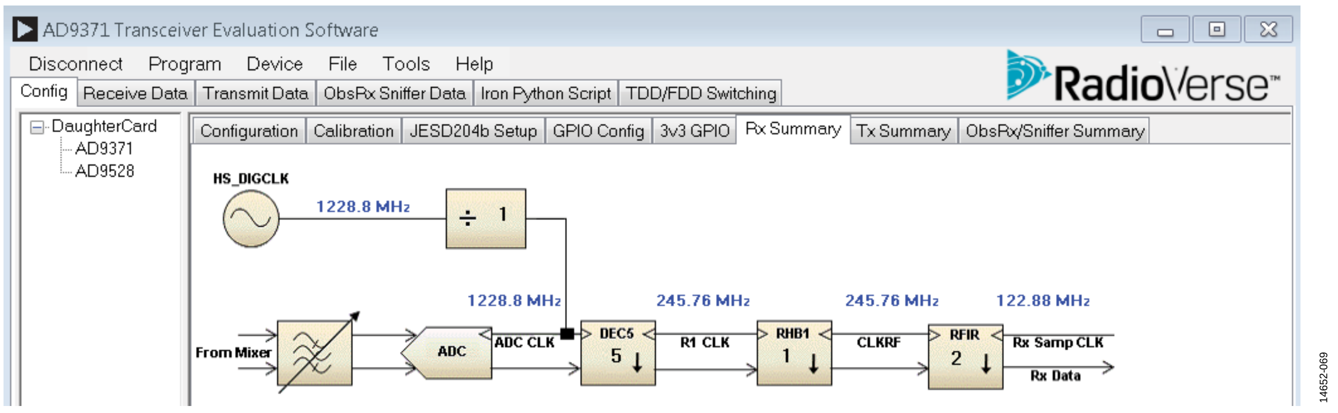


Figure 79. Filter Configuration for the Rx = 100 MHz, 122.88 MHz, DEC5 Profile

TRANSMITTER SIGNAL PATH

The transmitter has independent signal paths for the Tx1 and Tx2 ports. The Tx signal path receives data from the JESD204B interface block and sends this data through interpolating filters prior to a Σ-Δ DAC. The analog output of the DAC is low pass filtered by the Tx low-pass filter (LPF) prior to the upconversion mixer. The I and Q paths are identical to one another. Overranging is detected in the Tx digital signal path at each stage and is limited to the maximum code value to prevent data wrapping. A block diagram of the Tx1 and Tx2 signal paths is shown in Figure 80. Blocks are shown that correspond to the digital or analog filters in the Tx datapath.

LPF

The low-pass filter (LPF) is an analog, second-order Butterworth LPF with an adjustable 3 dB corner. The LPF is calibrated based on the 3 dB bandwidth, resulting in a consistent frequency corner across all devices. The transimpedance amplifier (TIA) 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.

The Tx LPF bandwidth is determined by the parameter device → tx → txProfile → txBbf3dBCorner_kHz.

THB2

The Transmit Half-Band 2 (THB2) is a fixed coefficient half-band interpolating filter. THB2 can interpolate by a factor of 2 or it can be bypassed. The full-scale range for this filter is 256 (2^8). The THB2 filter coefficients are [−17, 0, +145, +256, +145, 0, −17].

The THB2 interpolation factor is set by device → tx → txProfile → thb2Interpolation. Set this to either to 1 (bypass) or 2.

THB1

The Transmit Half-Band 1 (THB1) is a fixed coefficient half-band interpolating filter. THB1 can interpolate by a factor of 2 or it can be bypassed. The full-scale range for this filter is 8192 (2^{13}). The THB2 filter coefficients are [+21, 0, −56, 0, +108, 0, −188, 0, +319, 0, −526, 0, +876, 0, −1632, 0, +5179, +8192, +5179, 0, −1632, 0, +876, 0, −526, 0, +319, 0, −188, 0, +108, 0, −56, 0, +21].

The THB2 interpolation factor is set by device → tx → txProfile → thb1Interpolation. Set this to either 1 (bypass) or 2.

TFIR

The programmable transmitter finite impulse response (TFIR) filter acts as a interpolating filter in the Tx path. The TFIR can interpolate by a factor of 1, 2, or 4, or it can be bypassed. The TFIR is typically used to compensate for roll-off caused by the post DAC analog LPF. The TFIR can use a configurable number of taps from 16 to 96 in multiples of 16.

The maximum number of taps is limited by the TFIR clock rate (data processing clock, DPCLK). The maximum DPCLK is 400 MHz. The DPCLK is the DAC clock (DACCLK) or DACCLK/2 to ensure that the DPCLK below 400 MHz.

$$\textit{Maximum Number of Tx FIR Filter Taps} = (\textit{DPCLK}/\textit{TX\_IQDataRate}) \times 16 \quad (17)$$

The TFIR coefficients are stored in the device data structure in device → tx → txProfile → *txFir, which is a pointer to a structure with data type mykonosFir_t.

The TFIR interpolation factor is set in the device data structure in device → tx → txProfile → txFirInterpolation, which can be either 1 (bypass), 2, or 4.

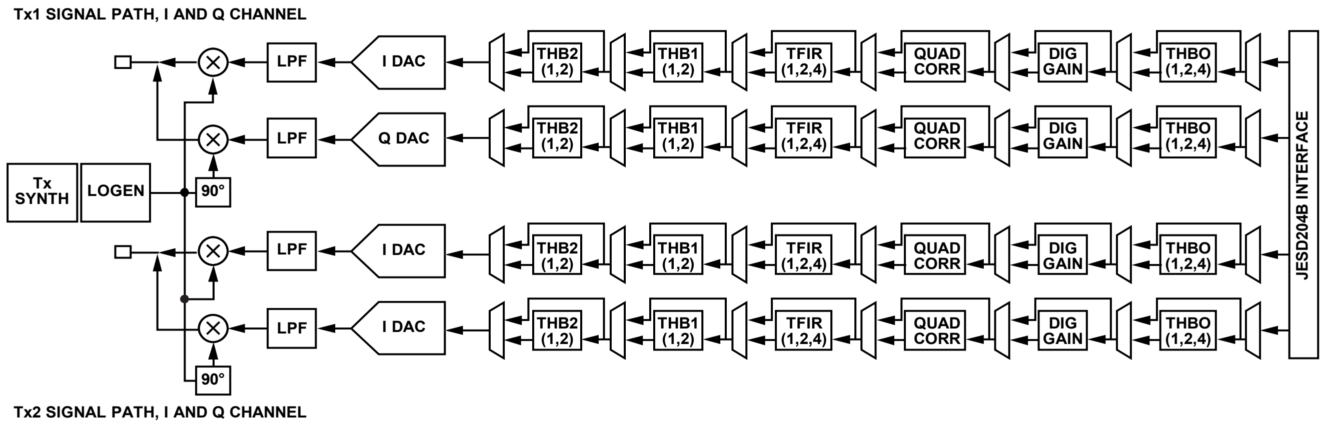


Figure 80. Tx1 and Tx2 Signal Path Diagram

Tx SIGNAL PATH EXAMPLE

The transceiver evaluation software (TES) provides an example depicting how the baseband filtering stages are used in profile configurations for a signal datapath. In this example, Tx 75 MHz/ 200 MHz, IQ rate = 245.76 MHz, and the DEC5 profile are selected for the Tx channels. This profile is compatible with the other examples provided in this user guide.

There is a difference in the naming convention for Tx profiles in the portion of the 75/200 MHz profile string. The 75 refers to the primary signal bandwidth, which is the bandwidth at which the user transmits large signal data, such as modulated carriers. The 200 refers to the digital predistortion (DPD) synthesis bandwidth.

Figure 83 shows the filter configuration for this profile. Note that the clocking frequencies are in blue. The signal rate after the TFIR block is equal to the IQ data rate of the profile.

The **Tx Summary** tab also shows the frequency response of the digital filters, the analog filters, the DAC sinc response, and the composite response of the signal chain. The response is plotted from dc to the DAC clock rate (see Figure 81).

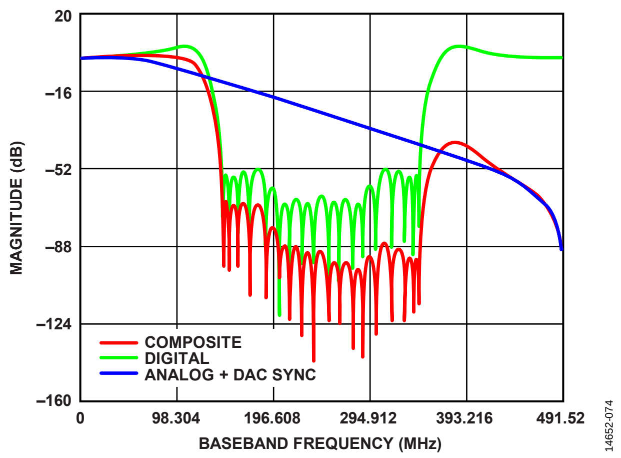


Figure 81. Transmitter Filter Responses

An examination of the profile pass band in Figure 82 shows that the analog response slightly attenuates information within the profile pass band. This analog attenuation is compensated by the digital filter response to obtain a maximally flat pass band for this profile. Recall that the primary signal bandwidth is restricted to 75 MHz. There is minimal digital gain applied to signals with baseband frequency less than 75 MHz from dc. Transmitting signals near the DAC full scale outside of this bandwidth may cause undesirable spurs.

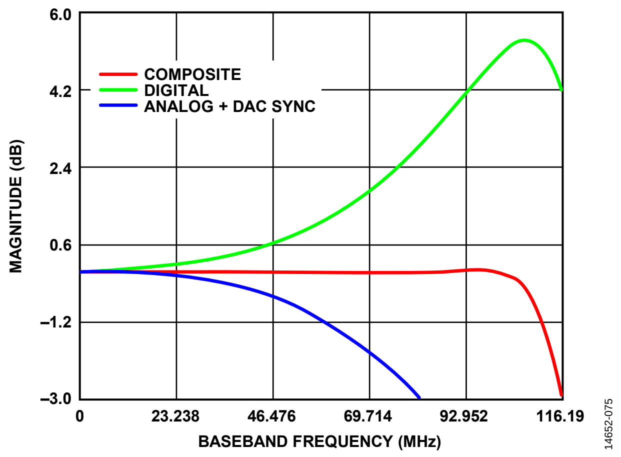


Figure 82. Examination of the Pass-Band Frequency Response of the Tx 75 MHz/200 MHz, 245.76 MHz, DEC5 Profile

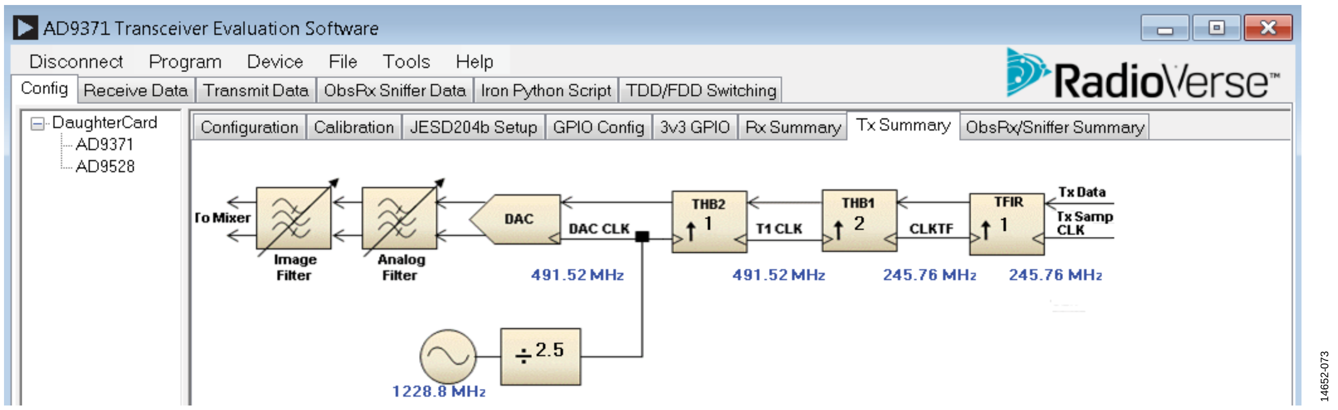


Figure 83. Filter Configuration for the Tx 75 MHz/200 MHz, 245.76 MHz, DEC5 Profile

OBSERVATION RECEIVERS SIGNAL PATH

The observation system receiver (ORx) selects one signal from five available receiver signal inputs: two observation receiver inputs (ORx1 and ORx2) and three sniffer receiver inputs (SnRxA, SnRxB, and SnRxC). It can also be used for internal calibration at initialization by connecting to one of two internal loopback paths This mode is designated as OBS_INTERNALCALS and allows the ARM to switch between ORx configurations as needed.

This path has two filter banks: one for the sniffer channels and another for the observation channels, where coefficients for the respective receiver type are loaded. The device switches from one bank to the other when switching the ORx input from one source to another. See the Observation Receiver (ORx) section for additional details about the ORx.

The ORx passes downconverted I/Q data from one of the seven channels into the ORx baseband signal path. The ORx baseband signal path consists of a programmable low-pass filter (LPF), a Σ-Δ ADC, and digital decimation stages. The fixed coefficient decimation filters (RHB1, RHB2, RHB3, and DEC5) eliminate overranging. The programmable receiver FIR filter (RFIR) in the ORx digital baseband path can overrange, depending on coefficients. However, the RFIR output code is limited to a maximum code value when overrange conditions occur.

Low-Pass Filter

The ORx LPF is a transimpedance amplifier (TIA) with a single, real pole frequency response. The 3 dB bandwidth of the TIA LPF ranges from 20 MHz to 100 MHz. The TIA LPF is calibrated based on the 3 dB bandwidth, resulting in a consistent frequency corner across all devices. The TIA 3 dB bandwidth is set within the device data structure and is profile dependent. Any roll-off within the pass band is compensated by the RFIR to ensure a maximally flat pass-band frequency response.

The LPF bandwidth is set in the device data structure in device → obsRx → orxProfile → rxBbf3dBCorner_kHz for the ORx1 and ORx2 channels, and in device → obsRx → snifferProfile → rxBbf3dBCorner_kHz for the SnRxA, SnRxB, and SnRxC channels.

DEC5

The main ORx signal path features an option to use either the decimate by 5 (DEC5) or the series combination of the decimating Receiver Half-Band 3 (RHB3) and Receive Half-Band 2 (RHB2) digital filters. The DEC5 filter is a fixed coefficient decimating filter. Full scale for the DEC5 digital filter is 8192 (2^{13}). The coefficients for the DEC5 filter are [+18, +35, +56, +72, +70, +28, −38, −126, −209, −244, −184, −20, +256, +612, +976, +1273, +1448, +1448, +1273, +976, +612, +256, −20, −184, −244, −209, −126, −38, +28, +70, +72, +56, +35, +18].

The decimation factor for the first filter after the ADC is set in the device data structure in device → obsRx → orxProfile → rxDec5Decimation for the ORx1 and ORx2 channels, and in device → obsRx → snifferProfile → rxDec5Decimation for the SnRxA, SnRxB, and SnRxC channels. Set this parameter as 5 to use the DEC5 path.

RHB3

The RHB3 filter is a fixed coefficient decimating filter. RHB3 decimates by a factor of 2. The full scale for this filter is 16 (2^4). The RHB3 coefficients are [1, 4, 6, 4, 1].

RHB2

The RHB2 filter is a fixed coefficient decimating filter. RHB2 can decimate by a factor of 2. The series combination of RHB2 and RHB3 can be bypassed using the DEC5 filter. The full scale for this filter is 128 (2^7). The RHB2 coefficients are [+1, 0, −7, 0, +38, +64, +38, 0, −7, 0, +1].

The decimation factor for the first filter after the ADC is set in the device data structure in device → obsRx → orxProfile → rxDec5Decimation for the ORx1 and ORx2 channels and device → obsRx → snifferProfile → rxDec5Decimation for the SnRxA, SnRxB, and SnRxC channels. Set this parameter as 4 to use the series combination of the RHB3 and RHB2 filters.

RHB1

The RHB1 filter is a fixed coefficient decimating filter. The RHB1 can decimate by a factor of 2, or it can be bypassed. The full scale for this filter is 16384 (2^{14}). The RHB1 coefficients are [+9, 0, −41, 0, +124, 0, −304, 0, +665, 0, −1473, 0, +5074, +8108, +5074, 0, −1473, 0, +665, 0, −304, 0, +124, 0, −41, 0, +9].

The decimation factor for RHB1 is set in the device data structure in device → obsRx → orxProfile → rhb1Decimation for the ORx1 and ORx2 channels and device → obsRx → snifferProfile → rhb1Decimation for the SnRxA, SnRxB, and SnRxC channels. This can be either 1 (bypass) or 2 (decimate by 2).

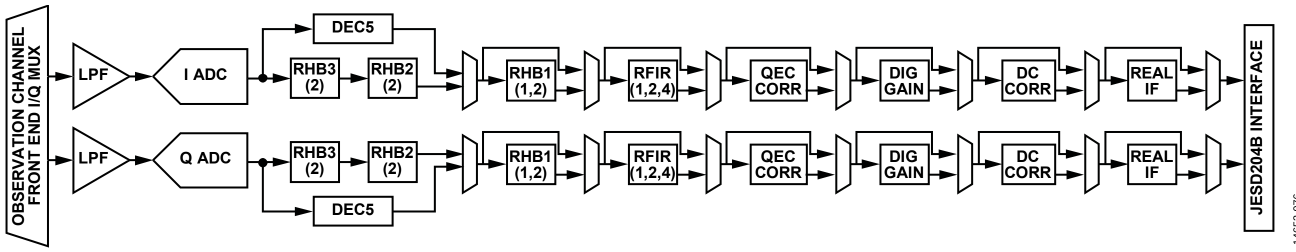


Figure 84. ORx Signal Path After the I/Q Mux Stage

RFIR

The programmable receiver finite impulse response (RFIR) filter acts as a decimating filter. The RFIR can decimate by a factor of 1, 2, or 4, or it can be bypassed. The RFIR can use a configurable number of taps from 24 taps to 72 taps in multiples of 24. The RFIR is typically used to compensate for the roll-off of the analog transimpedance amplifier (TIA) low-pass filter and decimating filters.

The maximum number of taps is limited by the FIR clock rate (data processing clock, DPCLK). The maximum DPCLK is 400 MHz. The DPCLK is the ADC clock rate divided by either 2 or 4 to limit the DPCLK below 400 MHz when the DEC5 is disabled. The DPCLK is the ADC clock rate divided by 5 or 10 to limit the DPCLK below 400 MHz when DEC5 is enabled.

$$\text{Maximum Number of Rx FIR Filter Taps} = (DPCLK/ObsRx\_IQDataRate) \times 24 \quad (18)$$

The RFIR coefficients are stored in the device data structure in device → obsRx → orxProfile → *rxFir for the ORx1 and ORx2 channels and device → obsRx → snifferProfile → *rxFir for the SnRxA, SnRxB, and SnRxC channels, which is a pointer to a structure with data type mykonosFir_t.

The RFIR decimation factor is set in the device data structure in device → obsRx → orxProfile → rxFirDecimation for the ORx1 and ORx2 channels and device → obsRx → snifferProfile → rxFirDecimation for the SnRxA, SnRxB, and SnRxC channels, which can be either 1 (bypass), 2, or 4.

Real IF

The real IF block contains an interpolating filter and mixer used to convert complex signals centered around dc into real valued signals centered around some IF frequency. The real IF conversion block is typically bypassed, but it does provide the capability to operate at an IF frequency near baseband for systems that prefer to perform complex demodulation in their digital baseband. The full-scale value for the real IF interpolating filter is 16384 (2^{14}). The coefficients for the real IF filter are [−3, 0, +8, 0, −19, 0, +40, 0, −75, 0, +130, 0, −214, 0, +336, 0, −514, 0, +773, 0, −1169, 0, +1845, 0, −3327, 0, +10380, +16384, +10380, −3327, 0, +1845, 0, −1169, 0, +773, 0, −514, 0, +336, 0, −214, 0, +130, 0, −75, 0, +40, 0, −19, 0, +8, 0, −3].

The real IF mode can be enabled in the device data structure in device → obsRx → realIfData, which can be either 0 (disable real IF) or 1 (enable real IF).

OBSERVATION RECEIVER SIGNAL PATH EXAMPLE

The transceiver evaluation software (TES) provides an example depicting how the baseband filtering stages are used in profile configurations for a signal path. In this example, the ORx 200 MHz, IQ rate 245.76 MHz, and DEC5 profile are selected for the ORx channels. This profile is compatible with the other examples provided in this user guide.

Figure 86 shows the filter configuration for this profile, excluding correction stage blocks such as dc offset correction. Note that the clocking frequencies are in blue. The signal rate after the RFIR block is equal to the IQRate of the profile.

Also available on the TES **ObsRx/Sniffer Summary** tab is the graphed frequency response of the TIA, digital filters, the ADC transfer function, and the composite response from dc to the sampling rate of the ADC, which is shown in Figure 85.

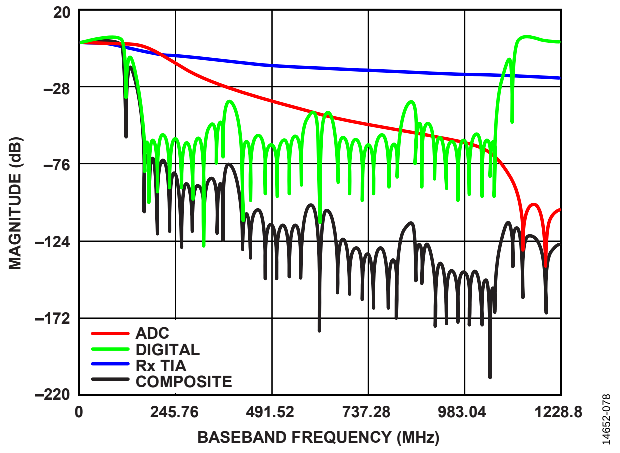


Figure 85. ORx Filter Responses

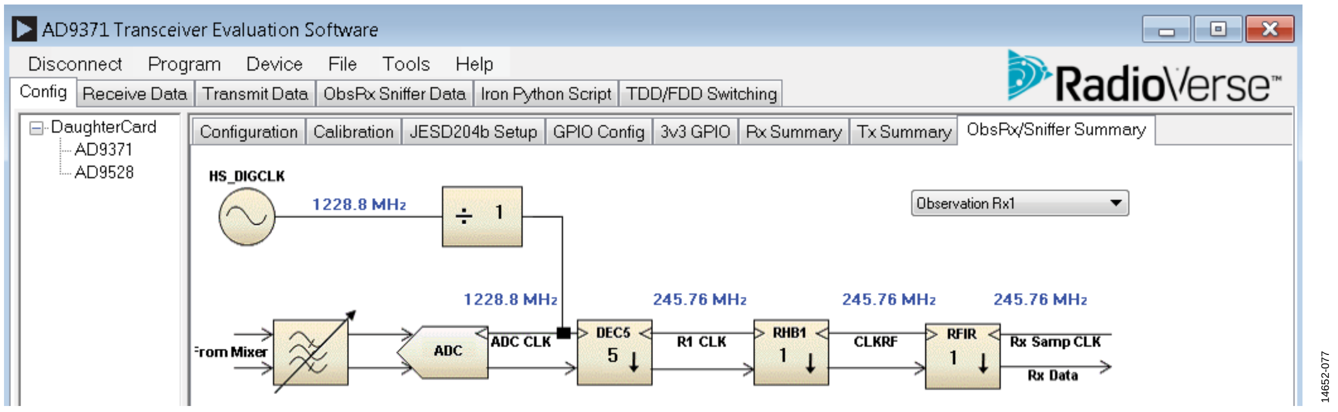

AD9371 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
Config Receive Data Transmit Data ObsRx Sniffer Data Iron Python Script TDD/FDD Switching
RadioVerse™
DaughterCard
AD9371
AD9528
Configuration Calibration JESD204b Setup GPIO Config 3v3 GPIO Rx Summary Tx Summary ObsRx/Sniffer Summary
HS_DIGCLK
1228.8 MHz
÷ 1
Observation Rx1
1228.8 MHz
245.76 MHz
245.76 MHz
245.76 MHz
From Mixer
ADC
ADC CLK
DEC5
5 ↓
R1 CLK
RHB1
1 ↓
CLKRF
RFIR
1 ↓
Rx Samp CLK
Rx Data
14652-077

Figure 86. Filter Configuration for the ORx 200 MHz, IQRate 245.76 MHz, Dec5 Profile

An examination of the profile pass-band frequency shows that the ORx transimpedance amplifier (TIA) 3 dB setting slightly attenuates information within the pass band. This analog attenuation is compensated by the digital filter response to obtain a maximally flat pass band for this profile. A zoom in view of the pass band is shown in Figure 87.

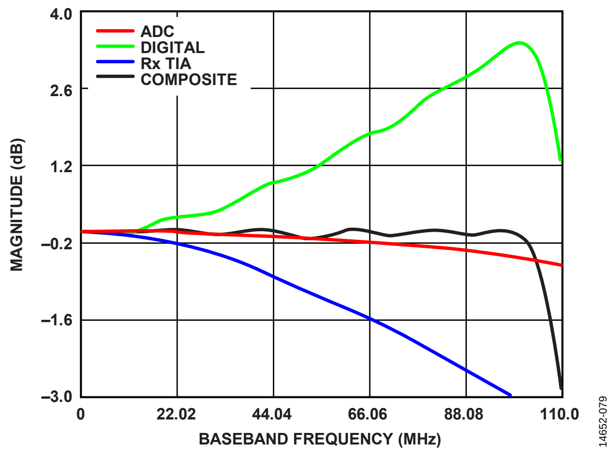


Figure 87. Examination of the Pass-Band Frequency Response of the Rx 100 MHz, 122.88 MHz, DEC5HR Profile

APPLICATION PROGRAMMING INTERFACE (API) DATA STRUCTURES AND API COMMANDS

Analog Devices software tools, such as the transceiver evaluation software (TES), populate the Tx, Rx, ORx, and SnRx profile data structures with the correct values that allow accurate pass-band flatness. If the desired profile exists in the software, it is recommended to set up the desired profile and use the data structure values generated by the **Create Config.c** file for the Tx, Rx, ORx, and SnRx profile data structures.

The following are subsets of the data structures and a partial listing of data structure members related to the configuration of the digital filters. Many of these data structure members have been covered in previous sections. Consult the device **API.chm** file for additional reference.

- mykonosRxSettings_t. This data structure contains members such as realIFData, the data structure rxProfile of the mykonosRxProfile_t type.
- mykonosRxProfile_t. This data structure contains members such as adcDiv, rxFirDecimation, enHighRejDec5, rhb1Decimation, iqRate_kHz, rfBandwidth_Hz, rxBbf3dBCorner_kHz, and the data structure rxFir of type mykonosFir_t.
- mykonosFir_t. This data structure contains gain_dB, numFirCoefs, and coefs members.
- mykonosObsRxSettings_t. This data structure contains members such as realIFData, and the data structures for orxProfile and snifferProfile. These data structures are of type mykonosRxProfile_t.
- mykonosTxSettings_t. This data structure contains members such as the data structure txProfile of type mykonosTxProfile_t.
- mykonosTxProfile_t. This data structure contains members such as dacDiv, txFirinterpolation, thb1Interpolation, thb2Interpolation, iqRate_kHz, primarySigBandwidth_Hz, rfBandwidth_Hz, txDac3dBCorner_kHz, and the data structure txFir of type mykonosFir_t.

mykonosFir_t Data Structure

The mykonosFir_t data structure is a structure that contains the finite impulse response (FIR) settings for the configurable transmitter finite impulse response (TFIR) and receiver finite impulse response (RFIR) filters. These filters are programmable to ensure digital compensation for analog filter roll-off to ensure a maximally flat pass band for any valid profile. Include an instance of the mykonosFir_t data structure for each profile used, for example, the Rx channels have a mykonosFir_t structure populated with the desired RFIR configuration, the Tx channels have a mykonosFir_t data structure populated with the TFIR configuration, and so on.

The data structure prototype is as follows:

```
/**
 *  \brief Data structure to hold FIR
filter settings */
typedef struct
{
    int8_t gain_dB;
/*!< Filter gain in dB*/
    uint8_t numFirCoefs;
/*!< Number of coefficients in the FIR
filter */
    int16_t *coefs;
/*!< A pointer to an array of filter
coefficients */
} mykonosFir_t;
```

The parameter descriptions are as follows:

- gain_dB. For Rx, ORx, and SnRx profiles, the valid gain settings are +6 dB, 0 dB, −6 dB, and −12 dB. For Tx profiles, the valid gain settings are 6 dB and 0 dB.
- numFirCoefs. For Rx, ORx, and SnRx profiles, the valid number of FIR coefficients are 24, 48, or 72. For Tx profiles, the valid number of FIR coefficients are 16, 32, 48, 64, 80 or 96.
- *coefs. This parameter points to an array of filter coefficients. The range of these coefficients can span the range of 16-bit signed integers.

These parameters are typically generated by Analog Devices or Analog Devices filter tools but must be loaded into data structures in the user application for desired operation.

Programming Filter Settings via API

Assuming that the device data structure, specifically the Tx, Rx, ORx, and SnRx profiles, are loaded properly, the procedure described in **headless.c/headless.h** sets up the digital and analog filters correctly. Some commands of note are listed as follows, with descriptions related to the digital filter settings:

- MYKONOS_initialize(…). This command initializes the device based on the desired device settings. This command calls MYKONOS_initDigitalClocks(…), which allows the filters to be clocked at the appropriate data rates. The MYKONOS_initDigitalClocks(…) command does not typically need to be called outside of MYKONOS_initialize(…).
- MYKONOS_initArm(…). This command resets the ARM processor and performs initialization. This command calls MYKONOS_initSubRegisterTables(…), which calls MYKONOS_programFir(…). These commands do not need to be called outside of the MYKONOS_initArm(…) command.

OBSERVATION RECEIVER (ORx)

This section describes the configuration and operation of the ORx. There are five receiver front-end inputs available in the ORx. The ORx also features a calibration mode, allowing Tx tracking calibration updates. This section includes a description of the signal chain of the ORx as well as references to application programming interface (API) commands and data structures used to configure the ORx.

The five front-end inputs include two observation receiver channels (ORx1 and ORx2) suitable for transmitter output observation and three sniffer receiver channels (SnRxA, SnRxB, and SnRxC) to monitor radioactivity at frequencies of interest. There is also an internal ORx input for Tx loopback calibration signals, OBS_INTERNALCALS. In OBS_INTERNALCALS mode, the ARM microprocessor controls the ORx.

The signal flow through the ORx is as follows: frequency downconverted I/Q data from one of the five receiver inputs passes through the I/Q mux switch into to the I/Q transimpedance amplifier (TIA) low-pass filter (LPF), which feeds the signal to the ADC and then to the digital signal path. The digital signal path consists of digital filters and signal conditioning stages. Refer to the Filter Configuration section for more information on the ORx digital filters. Note that only one input channel of the ORx inputs can be active at any time.

A high level block diagram of the ORx datapath is shown in Figure 88. Note that the digital filtering and signal conditioning between the ADC and the JESD204B interface is omitted from this diagram.

Proper functionality of the ORx requires power supplied to the 3.3 V and 1.3 V analog supply pins.

To achieve optimal device performance levels from the ORx ports, the receiver input pins likely need to be impedance matched. This matching typically involves a single-ended to differential signal conversion. Proper impedance matching facilitates ORx performance comparable to levels indicated in the data sheet. Information regarding RF port impedance matching is found in the RF Port Interface section.

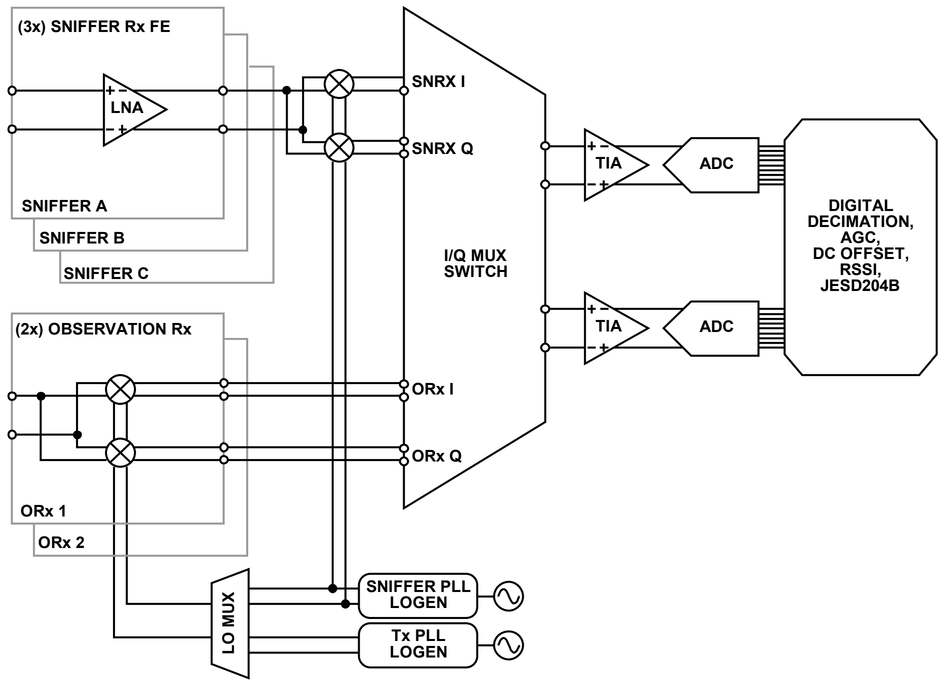


Figure 88. Simplified Block Diagram of the ORx Front End

OBSERVATION SYSTEM RECEIVER DETAILS

This section outlines the different front-end selections available in the ORx, explains how to select the active ORx channel, and other details regarding operation of the ORx.

Observation Receiver (ORx) Overview

The two independent observation receiver inputs have programmable bandwidths up to 250 MHz. The local oscillator (LO) source for the ORx1 and ORx2 mixers can come from the Tx LO or the SnRx LO. The SnRx LO facilitates the use of an LO frequency different than that of the Tx LO to monitor the Tx output. Each ORx channel has an independent analog front end up to the I/Q mux switch that is common to all ORx analog front ends.

The ORx inputs run a quadrature error correction (QEC) tracking calibration and dc offset calibration when the channel is enabled to improve the ORx image rejection and dc offset performance. These calibrations use ORx data to calculate calibration update words. The dc offset calibration runs any time the ORx channels are used. The QEC tracking calibration only runs if it is included in the tracking calibration mask set by the MYKONOS_enable-TrackingCals(…) command. ORx1 and ORx2 have independent bit masks in the tracking calibration mask. See the Quadrature Error Correction, Calibration, and ARM Configuration section for more information.

An external Tx LO signal can also be used in place of the integrated Tx LO. In this case, the external LO signal frequency must be twice that of the desired LO frequency of operation. When using the external LO, it is still necessary to program device data structures to represent the desired LO frequency, which allows the ARM calibration algorithms to function properly with respect to the LO frequency. There is no external LO option when the SnRx LO is selected.

Sniffer Receiver (SnRx)

Three SnRx inputs with maximum bandwidth of 20 MHz are also available on the ORx channel. Each SnRx has its own integrated low noise amplifier (LNA). The sniffer receivers share an inphase and quadrature mixer for the SnRxA, SnRxB, and SnRxC inputs. The sniffer mixer LO is generated by the SnRx LO.

Because there are three SnRx channels, it is useful to design matching networks with different center frequencies such that each SnRx channel can sniff in a different frequency band with optimum signal quality. Note that only one input can be activated at any given time.

The SnRx inputs run a dc offset calibration when the channel is enabled to improve the SnRx dc offset performance. These calibrations use SnRx data to calculate calibration update words. These calibrations only execute when the channel is active.

OBS_INTERNALCALS

The ORx must be set to OBS_INTERNALCALS to allow Tx tracking calibrations to update properly. This ORx input selection allows the ARM to control the ORx to generate update information for the active Tx tracking calibrations. The tracking calibrations that can update when in OBS_INTERNALCALS mode are the Tx LO leakage tracking calibration and the Tx QEC tracking calibration. Baseband processor (BBP) access to the ORx is not possible when this input is selected. It is recommended to set the ORx to OBS_INTERNALCALS when ORx data is not required by the BBP.

Refer to the Quadrature Error Correction, Calibration, and ARM Configuration section for more information.

Single ORx Mode

The single ORx mode is a mode that allows reduction of the number of ORx channels that must be used to maintain Tx LO leakage calibration performance. Typical modes of operation involve looping back the output of Tx1 into ORx1 and Tx2 into ORx2. With the single ORx mode, both Tx1 and Tx2 can be looped back into a switch that sends data from one of the transmitters into an ORx port.

See the Initial ARM Calibrations section for additional information.

Selecting the ORx Front End

Table 132 lists all front-end inputs available in the ORx.

Table 132. ORx Front-End Input Selections

Enumeration Name	ORx Front End	Enumeration Value	ORX_MODE[2:0]
OBS_RXOFF	None	0	000
OBS_RX1_TXLO	ORx1	1	001
OBS_RX2_TXLO	ORx2	2	010
OBS_INTERNALCALS	Dependent on calibration scheduling	3	011
OBS_SNIFFER	Sniffer (select channel)	4	100
OBS_RX1_SNIFFERLO	ORx1	5	101
OBS_RX2_SNIFFERLO	ORx2	6	110
OBS_SNIFFER_A	SnRxA	0x14	100
OBS_SNIFFER_B	SnRxB	0x24	100
OBS_SNIFFER_C	SnRxC	0x34	100

There are two control modes that select the ORx front-end channel: ORx ARM command mode or ORx ARM pin control mode. Selection between these two modes is determined in mykonosArmGpioConfig_t, which is in the orxPinMode data structure.

Set the orxPinMode member to 0 for ORx ARM command mode, or set it to 1 for ORx ARM pin control mode. Assuming the desired mode is set in the mykonosArmGpioConfig_t structure, the ORx control mode updates with successful execution of the MYKONOS_setRadioControlPinMode(…) API command. The MYKONOS_setRadioControlPinMode(…) command is called during MYKONOS_loadArmFromBinary(…).

The ORx does not need to be under the same control mode as the Tx/Rx (txRxPinMode).

ORx ARM Command Mode

The ORx ARM command mode method uses application programming interface (API) commands to switch between the various ORx inputs. The API command to switch between the ORx inputs is MYKONOS_setObsRx-PathSource(…). This command passes an argument of the mykonosObsRxChannels_t type. The enumerators corresponding to the various front ends of the ORx are listed in Table 132.

When calling the MYKONOS_setObsRxPathSource(…) command, the device must be set into the radio on mode. The MYKONOS_setObsRxPathSource(…) command can only be used when the ARM is in ORx command mode; otherwise, an error is returned.

ORx ARM Pin Control Mode

Alternatively, the ARM can be set into ORx ARM pin control mode. ORx ARM pin control mode is helpful in cases requiring precise control over the state of the ORx, such as in time division duplexed (TDD) applications.

When the device is set in ORx ARM pin control mode, the orxTriggerPin, orxMode2Pin, orxMode1Pin, and orxMode0Pin members of the mykonosArmGpioConfig_t data structure assign the GPIO pins for the 3-bit word, ORX_MODE[2:0] and for ORX_TRIGGER. These four signals are ARM inputs and control the active channel of the ORx when in ORx pin control mode.

The ORX_MODE[2:0] is a 3-bit value, set by three GPIO pins that are assigned to the orxMode2Pin, orxMode1Pin, and orxMode0Pin functions. These functions set the active channel of the ORx. At most, one of the ORx channels is connected to the shared baseband datapath at any given time. The available ORx modes are indicated in Table 132. The ORX_MODE[2:0] pins are sampled after the rising edge of the ORX_TRIGGER signal. Refer to the right column of Table 132 to correlate the ORX_MODE[2:0] word to the ORx front end.

Note the following specifics about the GPIO pin assignments for ORX_MODE[2:0] and ORX_TRIGGER:

- The ARM ignores the ORX_MODE[2:0] and ORX_TRIGGER pin inputs if the ORx is set to command mode, even if the ORX_MODE[2:0] and ORX_TRIGGER signals are assigned to GPIO input pins.
- If the ORX_MODE[2:0] signals are assigned to GPIO_18 to GPIO_16, the ARM assumes ORX_MODE[0] maps to GPIO_16, ORX_MODE[1] maps to GPIO_17, and ORX_MODE[2] maps to GPIO_18.
- Configure all three ORX_MODE[2:0] pins to GPIO pins within the pin ranges of GPIO_3 to GPIO_0, GPIO_15 to GPIO_4, or GPIO_18 to GPIO_16.

The ARM GPIO pins are configured to the settings within the mykonosArmGpioConfig_t data structure with the API command MYKONOS_setArmGpioPins(…).

When the ARM is set in ORx pin control mode, SnRxA, SnRxB, and SnRxC are not specified by ORX_MODE[2:0]. To select a sniffer channel, use the MYKONOS_setSnifferChannel(…) API command.

Note that the ARM can also send output signals indicating the status of the ORx (ARM acknowledge signals). These outputs are specified by the orx1EnableAck, orx2EnableAck, and srxEnableAck members of mykonosArmGpioConfig_t. These ARM acknowledge signals can be assigned to any GPIO pin from GPIO_0 to GPIO_15. For the pin assignment members, Bit 4 of the assignment 8-bit word must be set to enable the output. Figure 89 shows the relationship between ORX_MODE[2:0], ORX_TRIGGER, and the ORx ARM acknowledge signals. Note that, while the ORX_MODE[2:0] signals may toggle, they are not sampled until the ORX_TRIGGER satisfies the $t_{ORX\_TRIGGER\_HOLD}$ timing constraint.

The timing characteristics of Figure 89 are described in Table 133. Note that the minimum times for ORx switching depend on if the user is enabling tracking calibrations. The ORx must be set to OBS_INTERNALCALS to update for the minimum time for the Tx tracking calibrations (for example, Tx quadrature error correction (QEC) tracking and Tx local oscillator leakage (LOL) tracking) to update.

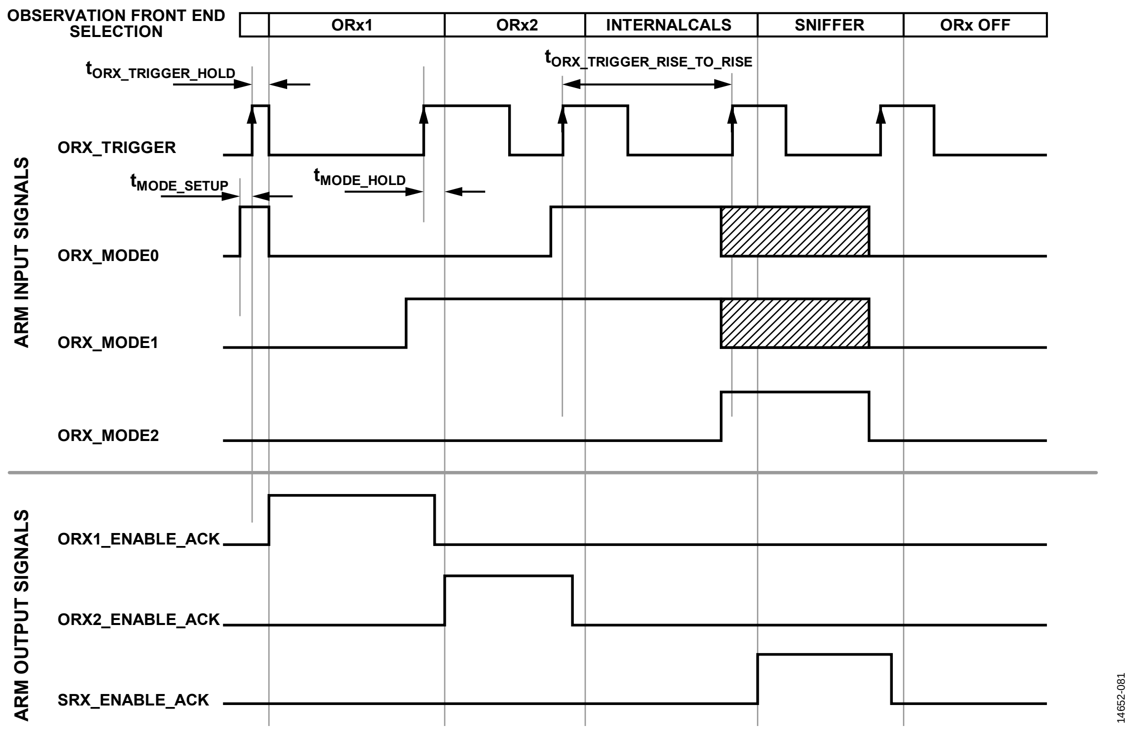


Figure 89. ORx Pin Control Mode Timing Diagram

Table 133. ORx Pin Control Timing Characteristics

Symbol	Description	Absolute Minimum Time	Minimum Time for Tx Tracking Calibrations	Maximum
$t_{ENABLE\_RISE\_TO\_FALL}$	Tx/Rx enable rising edge to enable falling edge—ENABLE signal width high	10 µs	800 µs	Not applicable
$t_{ENABLE\_FALL\_TO\_RISE}$	Tx/Rx enable falling edge to enable rising edge—ENABLE signal width low	10 µs	800 µs	Not applicable
$t_{ENABLE\_FALL\_TO\_ACK}$	Tx/Rx enable falling edge to acknowledge signal, to BBP going low	Not applicable	Not applicable	3 µs
$t_{ENABLE\_RISE\_TO\_ACK}$	Tx/Rx enable rising edge to acknowledge signal, to BBP going high	Not applicable	Not applicable	3 µs

ORx AGC, HYBRID, AND MGC

The device supports manual gain control (MGC) for all channels in the ORx path. The SnRx has the added capability of supporting hybrid gain control mode and automatic gain control (AGC) mode as well. In MGC, the baseband processor (BBP) can control the gain index of the channel via the application programming interface (API) commands to set the gain. The gain control block adjusts the gain of the ORx or SnRx receiver based on settings provided in the corresponding gain table. Gain settings and gain control only affect the active input of the ORx because only one input can be active at any given time.

To change the gain mode of the ORx channel, use the MYKONOS_setObsRxGainControlMode(…) API command, using the proper enumerated value for the gain control mode. The enumerated data type is mykonosGainMode_t.

To change the gain of the active channel, use the MYKONOS_ setObsRxManualGain(…) API command. The ORx must be in MGC mode to use this function. This API command returns an exception if the argument passed is out of range for the gain index of the active channel. For example, if the SnRx gain table is defined for Gain Index 255 to Gain Index 203, values outside of that range cause an error.

Readback of the current gain index for the active channel is available using MYKONOS_getObsRxGain(…). This function is valid in the MGC, hybrid, and AGC modes of operation.

Custom gain tables can be created in the **mykonos_user.c** and the mykonos_user.h files. The gain tables provide a means to vary the internal radio frequency (RF) attenuation, the external RF attenuation, the digital attenuation, and the digital gain. The data structure that sets up the AGC operation parameters is of the mykonosAgcCfg_t. type.

Additional details regarding the implementation of gain control schemes are provided in the Gain Control section.

OBSERVATION SYSTEM RECEIVER FRONT-END PROGRAMMING

Programming Prior to Device Initialization

This section provides a brief explanation of the data structures and application programming interface (API) commands used to configure the ORx channel.

Several data structures must be initialized and configured prior to initializing the device. Some of these data structures are members of other data structures, such as the mykonosObsRxSettings_t data structure, which is one of several members of the mykonosDevice_t data structure.

Assuming all the device data structures are valid, when the MYKONOS_initialize(…) API command is executed, the settings contained within the members of the device data structures are programmed to device registers.

Refer to the device **API.chm** file for clarification on data structure definitions and their member data types. This file is located under **Files/File List/t_mykonos.h**.

Important data structures for the ORx are listed as follows, with hierarchy details and short descriptions:

- mykonosDevice_t (device). This data structure type contains all device settings. The members of the structure relevant to the ORx setup include the mykonosObsRxSettings_t and mykonosTxSettings_t data structures. This data structure type also includes the mykonosDigClocks_t and profilesValid members.
- mykonosObsRxSettings_t (device → obsRx). This data structure type contains all ORx profile settings, JESD204B interface settings, and all other parameters specific to the operation of the available receivers of the ORx. This data structure sets up the SnRx LO frequency.
- mykonosOrxGainControl_t (device → obsRx → orxGainCtrl). This data structure type contains general gain control settings related to the gain mode of the ORx1 and ORx2 channels, the gain setting of the ORx1 and ORx2 channels when active, and their minimum and maximum gain indices.
- mykonosAgcCfg_t (device → obsRx → orxAgcCtrl). This data structure type contains gain control settings specific to the AGC for a specific ObsRx channel.
- mykonosSnifferGainControl (device → obsRx → snifferGainCtrl). This data structure type contains general gain control settings related to the gain mode of the sniffer channels, the gain setting of the sniffer channel when active, and the minimum and maximum gain indices of the sniffer channel. The SnRxA, SnRxB, and SnRxC inputs use the same gain index.
- mykonosRxProfile_t (device → obsRx → orxProfile and device → obsRx → snifferProfile). This data structure type contains profile settings used to configure the main receivers or observation receivers. This data structure type is used to define the profile for the SnRx and ORx channels.
- mykonosJesd204bFramerConfig_t (device → obsRx → framer). This data structure type contains configuration settings for the JESD204B digital interface. Information regarding the data structure members and how to set up the interface are provided in the JESD204B Interface section.
- mykonosFir_t (device → obsRx → orxProfile → rxFir and device → obsRx → snifferProfile → rxFir). This data structure type contains members required to program the programmable finite impulse response (PFIR) digital filter in the ORx channel.

Programming After Initialization

After initialization, some settings can be altered using other API commands. A list of the API commands relevant to the ORx are listed in this section. Refer to the section of the device **API.chm** file under **Files/File List/mykonos.c** for more information about the parameters to pass into these functions.

- MYKONOS_enableObsRxFramerPrbs(…). This function selects the pseudorandom bit sequence (PRBS) type and enables or disables ORx framer PRBS20 generation.
- MYKONOS_enableSysrefToObsRxFramer(…). This function enables or disables the SYSREF signal to the ORx framer of the transceiver.
- MYKONOS_getObsRxGain(…). This function obtains the gain index of the currently enabled ORx channel. The ORx datapath can have multiple RF sources. This function reads back the gain index of the currently enabled RF source. If the ORx datapath is disabled, an error is returned. If the uint8_t *gainIndex parameter is a valid pointer, the gain index is returned at the pointers address. Or, if the uint8_t *gainIndex pointer is null, the gainIndex readback is stored in the device data structure.
- MYKONOS_obsRxInjectPrbsError(…). This function initiates a PRBS error injection into the ORx datapath.
- MYKONOS_radioOff(…). This function instructs the ARM processor to move the radio state to the off state. When the ARM moves from the radio on state to the radio off (idle) state, the ARM tracking calibrations are stopped, and the Tx enable, Rx enable, and GPIO control pins (among others) are ignored. This stoppage also keeps the receive and transmit chains powered down until the MYKONOS_radioOn() function is called again.
- MYKONOS_radioOn(…). This function instructs the ARM processor to move the radio state to the radio on state. When the ARM moves to the radio on state, the enabled Rx and Tx signal chains power up, and the ARM tracking calibrations begin. To exit this state back to a low power, offline state (MYKONOS_radioOff(…) function).
- MYKONOS_readOrxFramerStatus(…). This function reads the status of the transceiver ORx framer.
- MYKONOS_setObsRxGaincontrolMode(…). This function configures the ORx gain control mode.

- MYKONOS_setObsRxManualGain(…). This function sets the Rx gain of the ORx channel. The ORx channel can have different RF inputs, for example, ORx1, ORx2, SnRxA, SnRxB, or SnRxC. This function sets the ORx gain index independently for ORx1 and ORx2, or SnRx. SnRxA, SnRxB, and SnRxC share the same gain index. Note that ORx1 and ORx2 share a gain table, as do SnRxA, SnRxB, and SnRxC. The maximum index is 255 and the minimum index is application specific.
- MYKONOS_setObsRxPathSource(…). This function powers up or powers down the observation Rx signal chain. When the ARM radio control is in ARM command mode, this function allows the user to selectively power up or power down the desired ORx datapath. If this function is called when the ARM is expecting GPIO pin control of the ORx path source, an error is returned.
- MYKONOS_setRfPllFrequency(…). This function sets the RF PLL local oscillator frequency (RF carrier frequency). This function must be called in the radio off state.
- MYKONOS_setSnifferChannel(…). This function selects the sniffer RF input to use for the observation receiver when in ORx pin mode (ORX_MODE = Sniffer 4). This function is only valid when using ORx pin mode. In pin mode, three GPIO pins select an observation Rx source for mykonos-ObsRxChannels_t enumerator values less than 7. When the ORX_MODE GPIO pins are set to 4 for sniffer mode, Sniffer Input A, Sniffer Input B, and Sniffer Input C can be chosen by calling this function. This function can be called any time after the ARM is loaded and running, and it can be called in the radio on or radio off state.
- MYKONOS_setupObsRxAgc(…). This function sets up the ORx AGC registers. Due to the dependencies, the instantiated AGC setting structure (mykonosAgcCfg_t) must be initialized with valid settings before this function can be used
- MYKONOS_setupJesd204bObsRxFramer(…). This function sets up the JESD204B ORx framer.
- MYKONOS_setArmGpioPins(…). This function programs register values to the device based on the current values stored in the mykonosArmGpioConfig_t data structure. This function does not write the entirety of the structure, only the orxTriggerPin, orxMode2Pin, orxMode1Pin, orxMode0Pin, rx1EnableAck, rx2EnableAck, tx1EnablePin, tx2EnableAck, orx1EnableAck, orx2EnableAck, srxEnableAck, or txObsSelect data structure members.
- MYKONOS_setRadioControlPinMode(…). This function programs register values to the device based on the current values stored in the mykonosArmGpioConfig_t data structure. This function does not write the entirety of the structure, only the useRx2EnablePin, useTx2EnablePin, txRxPinMode, or orxPinMode data structure members.

TDD CONFIGURATION AND SETUP

This section describes how to configure the device for time division duplexed (TDD) use cases. This section describes how to program the ZC706 (Zynq) evaluation platform and the evaluation board, the timing requirements for operating the device in ARM pin control mode, and relevant application programming interface (API) and **.dll** commands that configure the system for TDD operation. The guidelines in this section enable users to set up programming environments for TDD performance evaluation without the graphical user interface (GUI).

The transceiver evaluation software (TES) can configure the device and field programmable gate array (FPGA) into a TDD use case. The TDD page in the GUI configures the FPGA for Tx/Rx/ORx enable/disable timing and data timing, and it configures the device for pin control mode. To determine what commands are executed when the GUI configures TDD, see the log (found under **File** → **View Log Files**) in the GUI. This log is a helpful reference to configure TDD functionality with the evaluation system outside of the GUI. See the TES Interface for TDD Mode section for more information.

In this document, the **.dll** or the **.dll** layer refers to the **AdiCmdServerClient.dll** library, which is a **.NET** framework library allowing interaction with the device API from **.NET**-compatible languages such as C#, MATLAB, or IronPython. The **.dll** is typically run in a PC environment and facilitates communication between a PC and the ZC706 motherboard. Example scripts in this document are provided in IronPython and can be used in the **Iron Python Script** tab in TES.

TDD IN THE MYKONOS EVALUATION SYSTEM

The evaluation system allows users to implement TDD use cases. For TDD operation, the device must be set into ARM pin control mode for the Tx/Rx. The ORx can be set to either ARM pin control mode or ARM command mode. The setting that determines ARM pin control mode vs. ARM command mode is contained within the mykonosArmGpioConfig_t data structure (txRxPinMode, orxPinMode) in the API, and the procedure to modify this data structure is described in the ARM Pin Mode Configuration section. Pin control mode allows strict timing control over the state of the transceiver required by TDD use cases.

In the evaluation system, FPGA output signals can be programmed to drive the GPIO pins. The voltage on these pins can be monitored on the GPIO headers on the evaluation card. In ARM pin control mode, these pins act as inputs to the ARM to control the enable or disable state of the Tx/Rx/ORx. The FPGA, which can be implemented as a baseband processor (BBP) in user systems, controls the state of the transceiver through the GPIO pins.

Refer to the TddFsmParameters_us Class Details section (in **.dll** layer) and the mykonosArmGpioConfig_t Data Structure Details section (in the Mykonos API) to become familiar with the configurable parameters available for TDD implementation. TddFsmParameters_us configures the FPGA, while mykonosArmGpioConfig_t is used to configure the device. Also note that although there are descriptions of several API/**.dll** commands in this document, they are not related to the ARM command mode unless otherwise specified.

Note that the prefix MYKONOS_ implies that the function described exists in the Mykonos API. The prefix FpgaMykonos or Mykonos implies that the command exists in the **AdiCommandServerClient.dll**. All functions in the API have a **.dll** counterpart.

ARM Input and Output Signals

A BBP controls the Tx/Rx/ORx state of the device by interfacing through the GPIO pins to the ARM microcontroller in pin control mode. The ARM core has two main responsibilities with respect to TDD use cases: the real-time control of the Tx/Rx/ORx and the scheduling and execution of device calibrations. This section focuses on the real-time control of Tx/Rx/ORx data paths with respect to ARM pin control mode.

Refer to the mykonosArmGpioConfig_t Data Structure Details section for a brief overview of the GPIO controls available for the ARM.

ARM Pin Mode Configuration

The ADRV9371-N/PCBZ and ADRV9371-W/PCBZ evaluation boards have pin headers for the TX1_ENABLE, TX2_ENABLE, RX1_ENABLE, and RX2_ENABLE signals. The silkscreen indicators adjacent to the enable header pins are highlighted in Figure 90. The channel enable header pins are routed to corresponding balls (M6, M8, M5, and M9, respectively). The FPGA outputs enable signals to drive these pins. These signals are ignored unless the device is set to pin mode.

Pin control mode offers real-time control of the Tx/Rx/ORx channels. In pin control mode, the enable signals determine if the Tx/Rx/ORx signal chain is powered up or down. The API allows the Tx/Rx control mode to be set independently of the ORx control mode. The members that determine pin mode vs. command mode are txRxPinMode and orxPinMode within the mykonosArmGpioConfig_t data structure. If these parameters are set to 0, the device is in command mode. If set to 1, the device is set to pin mode. The user is able to configure the Tx/Rx in pin mode and ORx in command mode, if desired.

The Rx1/Rx2 and Tx1/Tx2 channels can be powered up together or independently. The member useTx2EnablePin of the data structure mykonosArmGpioConfig_t, if set to 0, allows the TX1_ENABLE pin voltage level to control the state of Tx1 and Tx2 simultaneously. If useTx2EnablePin is set to 1, the TX1_ENABLE pin only controls the enable signal for Tx1 while the TX2_ENABLE pin controls the enable signal for Tx2. Similarly, setting the useRx2EnablePin in the mykonosArmGpioConfig_t allows independent control of Rx1 and Rx2.

Note that updating the mykonosArmGpioConfig_t data structure does not change device settings. Two commands exist in the API to write ARM pin control settings (denoted useRx2EnablePin, useTx2EnablePin, txRxPinMode, orxPinMode) or ARM GPIO configuration settings (denoted orxTriggerPin, orxMode2Pin, orxMode1Pin, orxMode0Pin, rx1EnableAck, rx2EnableAck, tx2EnableAck, tx2EnableAck, orx1EnableAck, srxEnableAck, txObsSelect). MYKONOS_setRadioControlMode(…) and MYKONOS_setArmGpioPins(…) are these commands, respectively. The equivalent commands at the delay-locked loop (DLL) level are Mykonos.setRadioPinControlMode(…) and Mykonos.setArmGpioPins(…).

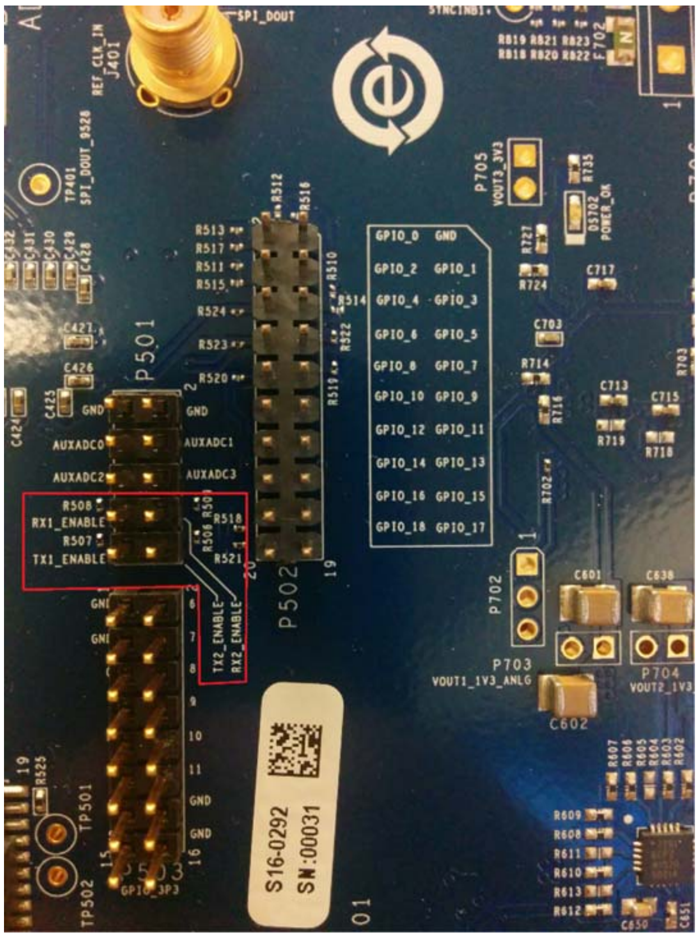


Figure 90. Tx1/Tx2 and Rx1/Rx2 Enable Pins

ARM Acknowledge Signals

The ARM microcontroller provides real-time control of Tx and Rx channels in the device when in Tx/Rx pin control mode. The ARM features several output signals that can be configured to output at user defined GPIO pins. The ARM acknowledgement (ARM ACK) signals go high (or low) to indicate that the ARM has powered up (or down) a Tx/Rx/ORx channel. For example, when the TX1_ENABLE signal goes high, the ARM outputs a signal when it enables the Tx1 channel. Table 134 shows the available ARM ACK signals.

Table 134. Available ARM Acknowledgement Signals

ARM Output Signal	Description
RX1_ENABLE_ACK	Indicates that Rx1 (or Rx2)[1] is/are enabled
RX2_ENABLE_ACK	Indicates that Rx2 is enabled
TX1_ENABLE_ACK	Indicates that Tx1 (or Tx2)[2] is/are enabled
TX2_ENABLE_ACK	Indicates that Tx2 is enabled
ORX1_ENABLE_ACK	Indicates that ORx1 is enabled for BBIC use
ORX2_ENABLE_ACK	Indicates that ORx2 is enabled for BBIC use
SNRX_ENABLE_ACK	Indicates that SnRxA, B, or C is enabled for BBIC use

[1] Determined by member useRx2EnablePin in mykonosArmGpioConfig_t. 0 = RX1_ENABLE controls Rx1 and Rx2, 1 = separate RX1_ENABLE/RX2_ENABLE pins.
[2] Determined by member useTx2EnablePin in mykonosArmGpioConfig_t. 0 = TX1_ENABLE controls Tx1 and Tx2, 1 = separate TX1_ENABLE/TX2_ENABLE pins

The ARM ACK signals are valid even when not in pin control mode; however, they must be assigned and enabled in the mykonosArmGpioConfig_t data structure.

Assigning the GPIO pin for an ARM acknowledge signal is a two step process in the device, and this procedure is specific to the ARM ACK output GPIO pin assignments. This two step process follows:

1. The device must be told what GPIO pin to output an ARM ACK signal. Pin assignment is accomplished by the rx1EnableAck, rx2EnableAck, tx1EnableAck, tx2EnableAck, orx1EnableAck, orx2EnableAck, and srxEnableAck members of the mykonosArmGpioConfig_t data structure in the API. Set these members to user desired configuration in the data structure instance. The data structure members listed previously are of Type uint8_t. The valid pin assignments for these signals are GPIO_0 through GPIO_15. In the 8-bit word, Bits[3:0] designate the pin, and Bit 4 enables the signal. The ARM ACK signals are not sent to the GPIO without the enable bit (Bit 4) set to 1.
2. Write the mykonosArmGpioConfig_t data structure values to the device by calling the MYKONOS_setArmGpioPins(…) API command, which can be done through the **.dll** command in the Mykonos class, Mykonos.setArmGpioPins(…).

Additionally, the ARM ACK signals can be used as a data path trigger in the evaluation system for the Tx and Rx. See the Datapath Trigger Modes section for more information about how to use the ARM ACK signals as trigger sources for Tx and Rx datapaths.

Figure 91 illustrates the timing relationship between the Tx and Rx enable signals and the Tx and Rx ARM ACK signals. Note that the minimum time period for any enable or disable state is 800 µs.

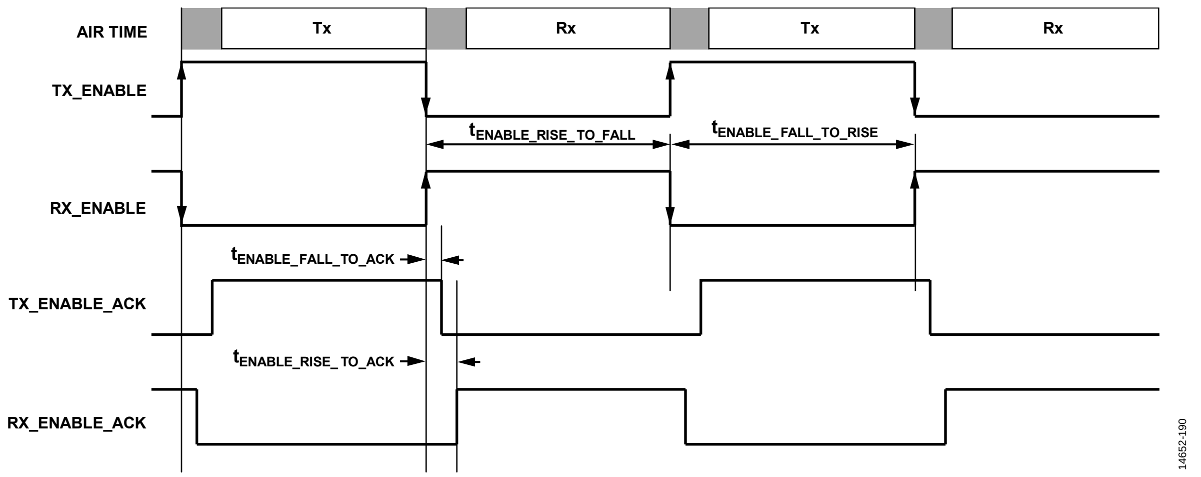

AIR TIME
Tx
Rx
Tx
Rx
TX_ENABLE
$t_{ENABLE\_RISE\_TO\_FALL}$
$t_{ENABLE\_FALL\_TO\_RISE}$
RX_ENABLE
$t_{ENABLE\_FALL\_TO\_ACK}$
TX_ENABLE_ACK
$t_{ENABLE\_RISE\_TO\_ACK}$
RX_ENABLE_ACK
14652-190

Figure 91. Tx and Rx Enable and Enable Acknowledge Signal Timing Diagram

Table 135. Tx and Rx Timing Characteristics

Symbol	Description	Absolute Minimum (µs)	Minimum Time for Tracking Calibrations (µs)	Maximum (µs)
$t_{ENABLE\_RISE\_TO\_FALL}$	Tx/Rx enable rising edge to enable falling edge—enable signal width high	10	800	
$t_{ENABLE\_FALL\_TO\_RISE}$	Tx/Rx enable falling edge to enable rising edge—enable signal width low	10	800	
$t_{ENABLE\_FALL\_TO\_ACK}$	Tx/Rx enable falling edge to ACK signal to BBP going low			2
$t_{ENABLE\_RISE\_TO\_ACK}$	Tx/Rx enable rising edge to ACK signal to BBP going high			2

Regarding calibrations, the internal tracking calibrations in the device are scheduled by the ARM microprocessor. The calibrations perform data captures in batches, with each batch guaranteed to have a duration greater than the value listed in the Minimum Time for Tracking Calibrations column of Table 135. Each calibration needs to capture a certain number of batches of data per second to keep up with device parameter (temperature, voltage supply variation) drift. If the transceiver state switches before a batch of data can be collected, the corresponding data is unusable by the calibration and thrown out.

Assuming that all tracking calibrations are active, Rx tracking calibrations update when the Rx is enabled. Tx tracking calibrations only update when the Tx channel is enabled and the ObsRx path source is set to INTERNAL_CALS mode. When in INTERNAL_CALS mode, the ARM has access to the ObsRx, temporarily preventing the baseband processor (BBP) access to the ORx datapath.

ORX_MODE and ORX_TRIGGER

When the ARM is set in pin control mode for the ORx, the members orxTriggerPin, orxMode2Pin, orxMode1Pin, and orxMode0Pin of the mykonosArmGpioConfig_t data structure assign GPIO pins for the ORX_MODE[2:0] 3-bit word and the ORX_TRIGGER. These four signals are ARM inputs and control the active channel of the ORx when in ORx pin control mode.

The ORX_MODE[2:0] is a 3-bit value (set by three GPIO pins: orxMode2Pin, orxMode1Pin, and orxMode0Pin) that sets the active channel of the ORx. At most, one of the ORx channels is connected to the shared baseband datapath at any given time. The ORX_MODEs available are detailed in Table 136. The ORX_MODE[2:0] pins are sampled after the rising edge of the ORX_TRIGGER signal.

Table 136. ORX_MODE[2:0] Word Definitions

ORX_MODE[2:0]	ORx Front End
000	ORx off
001	ORx1 with Tx local oscillator (LO)
010	ORx2 with Tx LO
011	Internal calibrations (OBS_INTERNALCALS)
100	Sniffer channel
101	ORx1 with sniffer LO
110	ORx2 with sniffer LO
111	Reserved

Figure 92 shows these relationships between the ORX_TRIGGER, ORX_MODE[2:0], and the ObsRx ARM ACK signals.

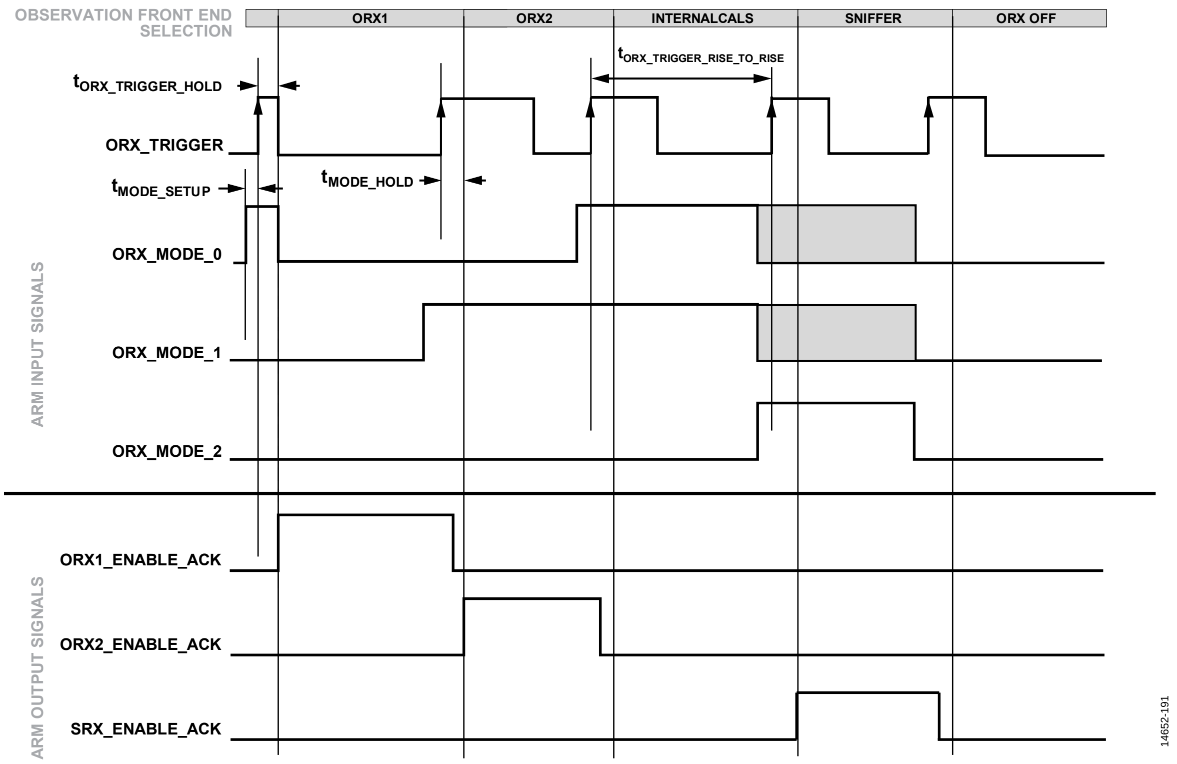


Figure 92. Observation Receiver (ORx) Signal Timing in TDD Mode

Table 137. Observation Receiver (ORx) Timing Characteristics

Symbol	Description	Absolute Minimum (μs)	Minimum Time for Tracking Calibrations (μs)
$t_{ORX\_TRIGGER\_RISE\_TO\_RISE}$	ORX_TRIGGER frequency—how often ORX_MODE[2:0] can be changed in the device.	10	800
$t_{ORX\_TRIGGER\_HOLD}$	ORX_TRIGGER hold time	1	
t_{MODE_SETUP}	ORX_MODE[2:0] setup time before ORX_TRIGGER rising edge	1	
t_{MODE_HOLD}	ORX_MODE[2:0] hold time to be sample by ARM	2	

Figure 92 illustrates that the ORX_MODE[2:0] is setup before the rising edge of the ORX_TRIGGER. This setup time is given by t_{MODE_SETUP}. The ORX_MODE[2:0] is not sampled until the ORX_TRIGGER has been high for $t_{ORX\_TRIGER\_HOLD}$. The ORX_MODE[2:0] must have a hold time relative to ORX_TRIGGER going high of t_{MODE_HOLD}. The ORX_MODE[2:0] can be changed at a minimum period of $t_{ORX\_TRIGGER\_RISE\_TO\_RISE}$. This time is dependent on whether tracking calibrations are operational for the Tx, because the Tx tracking calibrations use the ObsRx in OBS_INTERNALCALS mode. This timing is summarized in Table 137.

Timing constraints for the ORX_MODE[2:0] and ORX_TRIGGER signals are summarized in Table 138. Additionally, it is recommended that the ORX_TRIGGER signal stay low for at least 10 μs prior to the ORX_TRIGGER signal going high (high to low to high transition) to properly detect the rising edge.

To change the ORX_MODE[2:0] and ORX_TRIGGER GPIO pin assignments, set the members orxTriggerPin, orxMode2Pin, orxMode1Pin, and orxMode0Pin to the desired GPI O pins. Valid pin assignments for these signals range from GPIO Pin 0 to GPIO Pin 18. Be sure that orxPinMode is set to 1 to set the device into orxPinMode; otherwise, the pin assignments for ORX_MODE[2:0] and ORX_TRIGGER are ignored. Write the mykonosArmGpioConfig_t data structure values to the device by calling the MYKONOS_setArmGpioPins(…) API command, through the **.dll** command in the Mykonos.setArmGpioPins(…) class. This procedure is a similar procedure to the one outlined in the ARM Acknowledge Signals section.

Note the following details about the GPIO pin assignments for ORX_MODE[2:0] and ORX_TRIGGER:

- The ARM ignores the ORX_MODE[2:0] and ORX_TRIGGER pin inputs if the ORx is set to command mode, even if the ORX_MODE[2:0] and ORX_TRIGGER signals are assigned to GPIO input pins.
- If the ORX_MODE[2:0] signals are assigned to GPIO[18:16], ARM assumes ORX_MODE[0] maps to GPIO[16], ORX_MODE[1] maps to GPIO[17], and ORX_MODE[2] maps to GPIO[18].
- Configure all three ORX_MODE[2:0] pins to GPIO pins within the pin ranges of GPIO[3:0], GPIO[15:4], or GPIO[18:16].

The ORx can also run in ARM command mode if desired. This mode does not allow as precise timing control over the state of the ORx mode; however, it can free up GPIO pins. See the Observation Receiver (ORx) section for more details.

FPGA Output Signals

In the evaluation system, the field programmable gate array (FPGA) is capable of outputting the TX1_ENABLE/ TX2_ENABLE, RX1_ENABLE/RX2_ENABLE, ORX_MODE[2:0], and ORX_TRIGGER signals to the GPIO pins. The ARM responds to some or all of these FPGA output signals depending on the value of txRxPinMode and orxPinMode. This section describes how to program the FPGA enable and trigger signals to setup a variety of time division duplexed (TDD) use cases. A major software class in the evaluation system is the TddFsmParameters_us, which offers a highly configurable class to set TDD configuration parameters, and Tx/Rx/ORx enable and disable timing in the FPGA.

Datapath Trigger Modes

The FPGA used with the evaluation system provides several trigger sources that initiate either data captures in the case of the receivers or data transmission in case of the transmitters.

The trigger sources for data captures include an IMMEDIATE capture mode, a TDD_SM_PULSE mode, an EXT_SMA mode, and an ARM_ACK mode. These modes and some associated details are listed in Table 138.

Table 138. Rx Trigger Modes and Descriptions

RXTRIGGER Enumeration Name	Description
IMMEDIATE	Data in the Rx datapath is loaded into the FPGA memory immediately and stops when the capture is complete.
TDD_SM_PULSE	Data from the Rx datapath is loaded into the FPGA memory at the beginning of the frame and continues until the end of the capture. The sample captures are fixed to start at the beginning of the frame, given by TDD_SM_PULSE, but the number of samples to capture can be changed. This is the recommended data capture trigger.
EXT_SMA	Data from the Rx datapath is loaded into the FPGA at the rising edge of an external signal applied to the J68 surface-mount SMA on the ZC706 platform. The external trigger pin is configured with the FpgaMykonos.setupRxExtTrigPin(…) .dll command.
ARM_ACK	Data from the Rx datapath is loaded into the FPGA memory at the onset of the RX_ENABLE_ACK signal from the ARM. The FPGA must know the GPIO pin assignment for the Rx1, Rx2, and/or ObsRx ARM ACK pins, as configured in mykonosArmGpioConfig_t, for proper data capture. This setup is performed with the following commands in the .dll: FpgaMykonos.setupRx1ArmAckGpio(…), FpgaMykonos.setupRx2ArmAckGpio(…), and FpgaMykonos.setupOrxArmAckGpio(…).

For the evaluation system, the field programmable gate array (FPGA) programming mandates that the GPIO pin assignment for ARM acknowledge outputs to be the same pin (that is, pin assignments for orx1EnableAck = orx2EnableAck = srxEnableAck).

The trigger sources for data transmission include an IMMEDIATE transmit mode, a TDD_SM_PULSE mode, EXT_SMA mode, and an ARM_ACK mode. These modes and some associated details are listed in Table 139.

Table 139. Tx Trigger Modes and Descriptions

TXTRIGGER Enumeration Name	Description
IMMEDIATE	The FPGA transmits data from memory immediately into the Tx datapath.
TDD_SM_PULSE	The FPGA transmits data from memory into the Tx datapath at the beginning of the frame and continues until all samples have been transmitted. It is important to keep the duration of the transmitted samples equal to one frame length because the data loops to the beginning when the FPGA has transmitted the last sample of the frame. The wrong number of samples can lead to looping unsynchronized to the beginning of the frame.
EXT_SMA	The FPGA transmits data from memory into the Tx datapath at the rising edge of an external signal applied to the J67 throughhole SMA on the ZC706 platform. The external trigger pin is configured with the FpgaMykonos.setupRxExtTrigPin(…) .dll command.
ARM_ACK	Data from the FPGA is loaded into the Tx datapath at the onset of the Tx enable acknowledge signal from the ARM on the device. The FPGA must know the GPIO pin assignment for the Tx1 and/or Tx2 ARM acknowledge pins, as configured in mykonosArmGpioConfig_t, for proper data transmission. This setup is performed with the following commands in the .dll: FpgaMykonos.setupTx1ArmAckGpio(…) and FpgaMykonos.setupTx2ArmAckGpio(…).

TDD Finite State Machine Class

The **.dll** offers a mechanism to program configuration settings, enable timings, disable timings, and datapath delays for the Tx, Rx, and ORx channels in the field programmable gate array (FPGA). The TddFsmParameters_us class is a **.dll** features that sets up several variables that control the timing settings, allowing the user to program a variety of time division duplexed (TDD) uplink/downlink configuration timing on the evaluation system. The TDD finite state machine (FSM) is implemented on the FPGA to provide the Tx/Rx enable, ORX_MODE[2:0], ORX_TRIGGER, and EXT TRIG signals.

A full listing of the parameters in the TddFsmParameters_us class and a short description is provided in the TddFsmParameters_us Class Details section. The five major member categories of the TddFsmParameters_us class are as follows:

- Configuration settings: includes members such as TddSecondPtrEnable, TddLoopCount, TddFrameCount_us, and more. The length of the frame is defined in this region by the TddFrameCount_us member.
- Primary region pointers: includes the start time and stop time pointers for a primary region within a frame. In this case, primary refers to the first event within a frame when a Tx/Rx/ObsRx channel is enabled. If a UL/DL frame configuration has two separate Tx/Rx/ObsRx enable events, the primary region pointers set up the first event, and the secondary region pointers set up the second event. These values must be positive and less than the TddFrameCount_us duration.
- Secondary region pointers: includes the start and stop time pointers for a secondary region within a frame. These values must be positive and less than the TddFrameCount_us.
- Delay values: These are not included in the FPGA Version 0x46000030 or earlier. The option to delay the datapaths relative to the Tx/Rx/ObsRx enable events will be included in a future release of FPGA code. Setting a delay value has no effect in FPGA Version 0x46000030 or earlier.
- Other: includes pointers for the start and stop time for the EXT TRIG signal. The EXT TRIG can be further configured with the FpgaMykonos.setupTxExtTri(…) and/or FpgaMykonos.setupRxExtTrig(…) commands in the **.dll**.

Writing new values into this class does not update the FPGA registers with new values. After creating an instance and setting TDD FSM parameters to the desired values, the FpgaMykonos.initTdd(…) command passes an instance of the class to the cmd_server to update the FPGA.

Quick Help for Programming FPGA/Mykonos

To summarize the setup and configuration concepts described in this document thus far, general strategies on how to program the evaluation system are provided as follows.

- To configure the ARM pin control mode setup, GPIO pin assignments for ARM inputs (enable/ORX_MODE/ ORX_TRIGGER[2:0]) and ARM outputs (ARM ACKs):
  - From the application programming interface (API), first, write the desired values into the Mykonos-ArmGpioConfig_t type data structure. Two commands exist to update the ARM. Table 140 depicts which parameters are updated using specific API commands.
  - From the delay-locked loop (DLL), first, write the desired values into the mykonosArmGpioConfig_t type data structure by using the Mykonos.init_ armGpioStructure(…) command. Two commands exist to update the ARM. Table 140 depicts which parameters are updated using specific DLL commands.

Table 140. Commands to Program mykonosArmGpioConfig_t Data Structure Members to the ARM

mykonosArmGpioConfig_t Parameter	How to Update the ARM from the API	How to Update the ARM from the DLL
useRx2EnablePin	MYKONOS_setRadioControlPinMode(…)	Mykonos.setRadioControlPinMode()
useTx2EnablePin	MYKONOS_setRadioControlPinMode(…)	Mykonos.setRadioControlPinMode()
txRxPinMode	MYKONOS_setRadioControlPinMode(…)	Mykonos.setRadioControlPinMode()
orxPinMode	MYKONOS_setRadioControlPinMode(…)	Mykonos.setRadioControlPinMode()
orxTriggerPin	MYKONOS_setArmGpioPins(…)	Mykonos.setArmGpioPins(…)
orxMode2Pin	MYKONOS_setArmGpioPins(…)	Mykonos.setArmGpioPins(…)
orxMode1Pin	MYKONOS_setArmGpioPins(…)	Mykonos.setArmGpioPins(…)
orxMode0Pin	MYKONOS_setArmGpioPins(…)	Mykonos.setArmGpioPins(…)
rx1EnableAck	MYKONOS_setArmGpioPins(…)	Mykonos.setArmGpioPins(…)
rx2EnableAck	MYKONOS_setArmGpioPins(…)	Mykonos.setArmGpioPins(…)
tx1EnableAck	MYKONOS_setArmGpioPins(…)	Mykonos.setArmGpioPins(…)
tx2EnableAck	MYKONOS_setArmGpioPins(…)	Mykonos.setArmGpioPins(…)
orx1EnableAck	MYKONOS_setArmGpioPins(…)	Mykonos.setArmGpioPins(…)
srxEnableAck	MYKONOS_setArmGpioPins(…)	Mykonos.setArmGpioPins(…)
txObsSelect	MYKONOS_setArmGpioPins(…)	Mykonos.setArmGpioPins(…)

- To configure the field programmable gate array (FPGA) in the Mykonos evaluation system, including Tx/Rx/ORx start and stop primary and secondary regions, time division duplexed (TDD) configuration parameters, delay variables, and external trigger setup:
  - From the application programming interface (API), it is not applicable.
  - From the delay-locked loop (DLL), first, write the desired values into an instance of the TddFsmParameters_us class. Second, program the values to device with FpgaMykonos.initTdd_us(…) and passing the TddFsmParameters_us instance as the argument.
- Set up the Tx trigger source as follows:
  - From the API, it is not applicable.
  - From the DLL, pass the desired trigger source into the FpgaMykonos.setTxTrigger(…) command.
- Set up the Rx trigger source as follows:
  - From the API, it is not applicable.
  - From the DLL, pass the desired trigger source into the FpgaMykonos.setRxTrigger(…) command.
- Set up the external trigger sources from the DLL as follows:
  - For the Tx external trigger, set up with the FpgaMykonos.setupTxExtTrigPin(…) command, which is the J67 SMA connector on the ZC706 motherboard.
  - For the Rx external trigger, set up with the FpgaMykonos.setupRxExtTrigPin(…) command, which is the J68 SMA connector on the ZC706 motherboard.

The ORX_MODE[2:0] and ORX_TRIGGER Seen by the FPGA

The ORX_MODE[2:0] and ORX_TRIGGER are ARM input signals driven by the FPGA. The FPGA must send commands to the appropriate GPIO pins configured in the device. The following two commands set up the FPGA to drive the ORX_MODE[2:0] and ORX_TRIGGER signals:

- FpgaMykonos.setupFpgaGpio(…)
- FpgaMykonos.setupFpgaOrxGpio(…)

The first command sets the FPGA to enable the ORX_MODE[2:0] and ORX_TRIGGER signals from the FPGA. The second command assigns the pin that these signals are sent to. It is important that these assignment match with the Mykonos GPIO pin assignments for the ORX_MODE[2:0] and ORX_TRIGGER set in the mykonosArmGpioConfig_t data structure. These commands are explained in the API/DLL Commands for TDD Configuration section.

Refer to the ORX_MODE and ORX_TRIGGER section for GPIO pin assignment constraints.

Example TDD Script in IronPython

The following script is an example used to configure the device for TDD operation in LTE UL/DL Configuration 3. This script does not demonstrate the steps necessary to capture Rx/ObsRx samples or to transmit a Tx data file. This functionality can be performed in the transceiver evaluation software (TES) or in the scripting tab itself.

Commands related to TDD that have not been explained to this point are described in the API/DLL Commands for TDD Configuration section.

Note that details such as file paths and TCPIP addresses may need to be modified to properly connect to the device. This script is set up for LTE Uplink/Downlink Configuration 0 (D, S, U, U, U, D, S, U, U, U). To achieve best performance, the timing of the enable pointers may need to be adjusted. Note that for the TddFsmParamters_us pointers, Time 0 corresponds to the beginning of the frame in the FPGA counter.

Run this script after a successful program device execution.

```
########################
#ADI Demo Python Script
########################

#Import Reference to the DLL
import clr
clr.AddReferenceToFileAndPath("C:\\Program Files (x86)\\Analog Devices\\Mykonos Transceiver
Evaluation Software\\AdiCmdServerClient.dll")
from AdiCmdServerClient import AdiCommandServerClient
from AdiCmdServerClient import TddFsmParameters_us

#Create an Instance of the Command server client, FpgaMykonos, Mykonos, and
TddFsmParameters_us classes
Link = AdiCommandServerClient.Instance
FPGA = Link.FpgaMykonos.Instance
MYK  = Link.Mykonos.Instance

#Connect to the Zynq Platform
if(Link.hw.Connected == 1):
    Connect = 0
else:
    Connect = 1
    Link.hw.Connect("192.168.1.10", 55555)

#Read the Version
print Link.version()

DISABLE = 0
ENABLE = 1
OUTPUT = 0
INPUT = 1


################################
###        ENABLE TDD        ###
################################

MYK.radioOff()
Link.hw.ReceiveTimeout = 0

##############
# FPGA SETUP #
##############
FPGA.disableTdd() #Disable TDD FSM in FPGA
FPGA.gateDataTdd(ENABLE) #Enable data gating for Tx

#Set FPGAs to output ORX_MODE/ORX_TRIG on GPIO pins
FPGA.setupFpgaGpio(FPGA.GPIO_FEATURES.TDDORXMODE, OUTPUT)

#Assign GPIO pins for ORX_MODE/ORX_TRIG signals
FPGA.setupFpgaOrxGpio(FPGA.ORXGPIO_SELECT.GPIO_15, #ORX_TRIGGER
                      FPGA.ORXGPIO_SELECT.GPIO_16, #ORX_MODE[0]
                      FPGA.ORXGPIO_SELECT.GPIO_17, #ORX_MODE[1]
                      FPGA.ORXGPIO_SELECT.GPIO_18) #ORX_MODE[2]

#Setup ARM ACK Output Signals on FPGA side
FPGA.setupTx1ArmAckGpio(0x80)   #GPIO 7
FPGA.setupTx2ArmAckGpio(0x100)  #GPIO 8
FPGA.setupRx1ArmAckGpio(0x200)  #GPIO 9
FPGA.setupRx2ArmAckGpio(0x400)  #GPIO 10
FPGA.setupOrxArmAckGpio(0x800)  #GPIO 11
```

```
#Enable ARM ACK Output Signals on FPGA
FPGA.enableArmAckOutputs(ENABLE)

#Setup FPGA TX/RX Data Path Trigger to TDD_SM_PULSE signals
FPGA.setTxTrigger(FPGA.TXTRIGGER.TDD_SM_PULSE)
FPGA.setRxTrigger(FPGA.RXTRIGGER.TDD_SM_PULSE)

#Create instance of TDD FSM Class
tddFsm = TddFsmParameters_us()

#Set Miscellaneous TDD Settings
tddFsm.TddLoopCount = 0
tddFsm.TddFrameCount_us = 10000
tddFsm.TddSecondPtrEnable = 1
tddFsm.TddExtTrigOnPtr_us = 9900
tddFsm.TddExtTrigOffPtr_us = 0

#Set Tx Timings
tddFsm.TddTx1OnPtr_us = 9950
tddFsm.TddTx1OffPtr_us = 1800
tddFsm.Tdd2ndTx1OnPtr_us = 4950
tddFsm.Tdd2ndTx1OffPtr_us = 6800
tddFsm.TddTx2OnPtr_us = 9950
tddFsm.TddTx2OffPtr_us = 1800
tddFsm.Tdd2ndTx2OnPtr_us = 4950
tddFsm.Tdd2ndTx2OffPtr_us = 6800
tddFsm.TddIntCalsOnPtr_us = 50
tddFsm.TddIntCalsOffPtr_us = 1700
tddFsm.Tdd2ndIntCalsOnPtr_us = 5050
tddFsm.Tdd2ndIntCalsOffPtr_us = 6700

#tddFsm.TddTx1DataPathDel_us = 40
#tddFsm.TddTx2DataPathDel_us = 40
#tddFsm.Tdd2ndTx1DataPathDel_us = 40
#tddFsm.Tdd2ndTx2DataPathDel_us = 40

#Set Rx Timings
tddFsm.TddRx1OnPtr_us = 1800
tddFsm.TddRx1OffPtr_us = 4950
tddFsm.Tdd2ndRx1OnPtr_us = 6800
tddFsm.Tdd2ndRx1OffPtr_us = 9950
tddFsm.TddRx2OnPtr_us = 1800
tddFsm.TddRx2OffPtr_us = 4950
tddFsm.Tdd2ndRx2OnPtr_us = 6800
tddFsm.Tdd2ndRx2OffPtr_us = 9950

#tddFsm.TddRx1DataPathDel_us = 40
#tddFsm.TddRx2DataPathDel_us = 40
#tddFsm.Tdd2ndRx1DataPathDel_us = 40
#tddFsm.Tdd2ndRx2DataPathDel_us = 40

#Load Tdd Fsm parameters into FPGA
FPGA.initTdd_us(tddFsm)

#Setup the Tx/Rx External Trigger Output (J67/J68 SMA on Zynq)
FPGA.setupTxExtTrigPin(FPGA.FPGA_TRIGGER_DIRECTION.FPGA_OUTPUT,
                       FPGA.FPGA_EXT_TRIGGER_SOURCE.TDD_FSM_TRIG)
FPGA.setupRxExtTrigPin(FPGA.FPGA_TRIGGER_DIRECTION.FPGA_OUTPUT,
                       FPGA.FPGA_EXT_TRIGGER_SOURCE.TDD_FSM_TRIG)

#############
# ARM SETUP #
```

```
#############
useRx2EnablePin = ENABLE
useTx2EnablePin = ENABLE
txRxPinMode = ENABLE
orxPinMode = ENABLE
orxTriggerPin = 15
orxMode2Pin = 18
orxMode1Pin = 17
orxMode0Pin = 16
rx1EnableAck = (9 | 0x10) #OR the pin assignment with 0x10 to make the pin an output pin.
rx2EnableAck = (10 | 0x10)
tx1EnableAck = (7 | 0x10)
tx2EnableAck = (8 | 0x10)
orx1EnableAck = (11 | 0x10)
orx2EnableAck = (11 | 0x10)
srxEnableAck = (11 | 0x10)
txObsSelect = 0

MYK.init_armGpioStructure(useRx2EnablePin, useTx2EnablePin,
    txRxPinMode, orxPinMode, orxTriggerPin, orxMode2Pin, orxMode1Pin,
    orxMode0Pin, rx1EnableAck, rx2EnableAck, tx1EnableAck, tx2EnableAck,
    orx1EnableAck, orx2EnableAck, srxEnableAck, txObsSelect)

FPGA.enableArmAckOutputs(ENABLE)
MYK.setArmGpioPins()
MYK.setRadioControlPinMode()


##########################################
##         Write Tx Data To RAM         ##
##########################################

FPGA.stopTxData()
FPGA.setTxTransmitMode(1) #Set Tx into continuous data transmit mode in data path
FPGA.enableTxDataPaths(FPGA.TX_DATAPATH.TX1_TX2)
FPGA.startTxData()
MYK.radioOn()
FPGA.enableTdd()

#Disconnect from the Zynq Platform
if(Connect == 1):
    Link.hw.Disconnect()
```

API/DLL COMMANDS FOR TDD CONFIGURATION

This section provides a list of functions that can assist in configuration of time division duplexed (TDD) mode. Note that the MYKONOS_ prefix implies that the function described exists in the Mykonos application programming interface (API). The FpgaMykonos. or Mykonos. prefix implies that the command exists in the **AdiCommandServerClient.dll**. All functions in the API have a **.dll** counterpart.

TDD Specific Commands in the FPGA

FpgaMykonos.disableTdd()

- Purpose: This command disables the field programmable gate array (FPGA) TDD finite state machine (FSM).
- Arguments: None.

FpgaMykonos.gateDataTdd(byte enable)

- Purpose: This command gates zeros into the datapath to the JESD204B framer until the TDD module is enabled, if the Tx RAM is connected to the Tx datapath. This command is useful before and after the TDD module is enabled.
- Argument enable: set enable to 1 to enable TDD data gating, and set enable to 0 to disable TDD data gating.

FpgaMykonos.initTdd_us(byte rw, ref TddFsmParameters_us tddParameters)

- Purpose: This command sets up the FPGA registers with the timing information set by the TddFsmParameters_us class. This class contains the start time and stop time for the primary and secondary pointers of a TDD frame, secondary pointer enable, continuous capture and transmit, and several other TDD FSM parameters.
- Argument rw: this is a read/write (0 = read, 1 = write) control. If read is selected, the FPGA registers values corresponding to the TDD FSM parameters are loaded into the tddFsmParameters_us structure. If write is selected, the FPGA registers are written to with the values in the tddFsmParameters_us structure.
- Argument tddParameters: see the TddFsmParameters_us Class Details section for more information.

GPIO Specific Commands

FpgaMykonos.setupFpgaGpio(GPIO_FEATURES configSetting, UInt32 gpioDirection)

- Purpose: This command sets the pin direction (input or output) of the GPIO pins.
- Argument gpioDirection: This argument uses the lower 19 bits to individually map to the GPIO pins, 0 through 18. Bit 0 corresponds to GPIO_0, Bit 1 corresponds to GPIO_1, and so on. Setting a bit sets the corresponding GPIO pin as an input.
- Argument configSetting: Several configuration settings are available. These modes can be enabled simultaneously so that the enumeration type GPIO_FEATURES provides combinations of the modes listed as follows. In the **.dll** layer, the available GPIO modes are as follows:
  - SPI2 is over GPIO[4:0].
  - ORX_TRIGGER and ORX_MODE interface enable. The user can define GPIO pins indicating the ORX_MODE over three pins set by the setupFpgaOrxGpio(…) command. The setupFpgaOrxGpio(…) also sets a GPIO pin for the ORX_TRIGGER signal.
  - HSCP mode over GPIO[17:12] and GPIO[8:5].
  - TESTPIN mode.

FpgaMykonos.setupFpgaOrxGpio(ORXGPIO_SELECT gpioOrxTrig, ORXGPIO_SELECT gpioOrxMode_0, ORXGPIO_SELECT gpioOrxMode_1, ORXGPIO_SELECT gpioOrxMode_2)

- Purpose: Set GPIO pins for the ORX_MODE and ORX_TRIGGER signals.
- Argument gpioOrxTrig: Designates a GPIO pin for the ORX_TRIGGER signal. ORX_TRIGGER goes high for ~1 μs at the start and stop time of an ORX_MODE.
- Arguments gpioOrxMode_0, gpioOrxMode_1, gpioOrxMode_2: Designates GPIO pins for the 3-bit word describing the active ObsRx mode. See Table 136 for ORX_MODES available and The ORX_MODE[2:0] and ORX_TRIGGER Seen by the FPGA section for GPIO restrictions.

FpgaMykonos.enableArmAckOutputs(byte enableArmAckOutput)

- Purpose: This command enables or disables the ARM acknowledge signal output sent to the GPIO pins specified by the setup commands (setupTx1ArmAckGpio(…), setupTx2ArmAckGpio(…), and so on).
- Argument enableArmAckOutput: 0 = disable, and 1 = enable.

FpgaMykonos.setupTx1ArmAckGpio(UInt32 tx1ArmAckGpio)

- Purpose: This command sets up a GPIO pin to output the ARM acknowledge signal for TX1_ENABLE_ACK. The signal does not appear on GPIO until the enableArmAckOutputs(…) command runs.
- Argument tx1ArmAckGpio: Sets the GPIO output pin for the TX1_ENABLE_ACK signal. The argument is a 19-bit [18:0] bit mask aligned for each GPIO pin.

FpgaMykonos.setupTx2ArmAckGpio(UInt32 tx2ArmAckGpio)

- Purpose: This command sets up a GPIO pin to output the ARM acknowledge signal for TX2_ENABLE_ACK. The signal does not appear on GPIO until the enableArmAckOutputs(…) command is run.
- Argument tx2ArmAckGpio: Sets the GPIO output pin for the TX2_ENABLE_ACK signal. The argument is a 19-bit [18:0] bit mask aligned for each GPIO pin.

FpgaMykonos.setupRx1ArmAckGpio(UInt32 rx1ArmAckGpio)

- Purpose: This command sets up a GPIO pin to output the ARM acknowledge signal for RX1_ENABLE_ACK. The signal does not appear on GPIO until the enableArmAckOutputs(…) command runs.
- Argument rx1ArmAckGpio: Sets the GPIO output pin for the RX1_ENABLE_ACK signal. The argument is a 19-bit [18:0] bit mask aligned for each GPIO pin.

FpgaMykonos.setupRx2ArmAckGpio(UInt32 rx2ArmAckGpio)

- Purpose: This command sets up a GPIO pin to output the ARM acknowledge signal for RX2_ENABLE_ACK. The signal does not appear on GPIO until the enableArmAckOutputs(…) command runs.
- Argument rx2ArmAckGpio: Sets the GPIO output pin for the RX2_ENABLE_ACK signal. Argument is a 19-bit [18:0] bit mask aligned for each GPIO pin.

FpgaMykonos.setupOrxArmAckGpio(UInt32 orxArmAckGpio)

- Purpose: This command sets up a GPIO pin to output the ARM acknowledge signal for ORX_ENABLE_ACK. The signal does not appear on GPIO until the enableArmAckOutputs(…) command runs. Note that all ObsRx channels acknowledge signal (ORx1, ORx2, SnRx) share an acknowledge pin.
- Argument orxArmAckGpio: Sets the GPIO output pin for the TX1_ENABLE_ACK signal. Argument is a 19-bit [18:0] bit mask aligned for each GPIO pin.

Mykonos.init_armGpioStructure(byte useRx2EnablePin, byte useTx2EnablePin, byte txRxPinMode, byte orxPinMode, byte orxTriggerPin, byte orxMode2Pin, byte orxMode1Pin, byte orxMode0Pin, byte rx1EnableAck, byte rx2EnableAck, byte tx1EnableAck, byte tx2EnableAck, byte orx1EnableAck, byte orx2EnableAck, byte srxEnableAck)

- Purpose: This command writes to the mykonos-ArmGpioConfig_t data structure in the Mykonos application programming interface (API). This command also has an overload function providing read/write access to the mykonosArmGpioConfig_t data structure.
- Arguments: See Mykonos API **.chm** file on mykonosArmGpioConfig_t.

Mykonos.setArmGpioPins(…)

- Purpose: Program register values to Mykonos based on the current values stored in the mykonosArmGpioConfig_t data structure. This does not write the entirety of the structure, only the following data structure members: orxTriggerPin, orxMode2Pin, orxMode1Pin, orxMode0Pin, rx1EnableAck, rx2EnableAck, tx1EnablePin, tx2EnableAck, orx1EnableAck, orx2EnableAck, srxEnableAck, and txObsSelect. This function also exists in the API as MYKONOS_setArmGpioPins(…).
- Arguments: none.

Mykonos.setRadioControlPinMode(…)

- Purpose: Program register values to Mykonos based on the current values stored in the mykonosArmGpioConfig_t data structure. This does not write the entirety of the structure, only the following data structure members: useRx2EnablePin, useTx2EnablePin, txRxPinMode, and orxPinMode. This function also exists in the API MYKONOS_setRadioControlPinMode(…).
- Arguments: none.

Trigger Specific Commands

FpgaMykonos.setRxTrigger(RXTRIGGER rxTrig)

- Purpose: The Rx trigger determines what event triggers the receiver datapath to move data into the field programmable gate array (FPGA). Table 138 contains descriptions of the available RXTRIGGER enumerations in the FpgaMykonos class.
- Argument rxTrig: Refer to Table 138.

FpgaMykonos.setTxTrigger(TXTRIGGER txTrig)

- Purpose: The Tx trigger determines what event triggers the beginning of Tx data transmission.
- Argument txTrig: Refer to Table 139.

TddFsmParameters_us Class Details

This section provides an explanation for the members of the TddFsmParameters_us class. Primary and secondary region timing are relative to the beginning of a frame.

Configuration Options

The configuration options are as follows:

- TddSecondPtrEnable: enables the second pointer regions.
- TddLoopCount: loop count determines the number of frames. 0 = continuous loop mode, 1 = one total frame, 2 = two total frames, and so on, 4-bit value.
- TddContRxCapture: This value is ignored. Continuous sample capture is default.
- TddContTxTransmit: This value is ignored. Continuous sample transmit is default.
- TddSyncExtTrig: sync frames to the external sync trigger.

- TddEnableSnRxPtr: enables pointers for SnRxA, SnRxB, and SnRxC regions.
- TddEnableOrx1SnifferLoPtrs: enables pointers for ORx1 with SnRx local oscillator (LO) input to the mixer.
- TddEnableOrx2SnifferLoPtrs: enables pointers for ORx2 with SnRx LO input to the mixer.

Primary Region Pointers

The primary region pointers include the following:

- TddFrameCount_us: duration of the frame.
- TddTx1OnPtr_us: start time of primary Tx1 transmit region. Must be less than the TddFrameCount_us value.
- TddTx1OffPtr_us: stop time of primary Tx1 transmit region. Must be less than the TddFrameCount_us value.
- TddTx2OnPtr_us: start time of primary Tx2 transmit region. Must be less than the TddFrameCount_us value.
- TddTx2OffPtr_us: stop time of primary Tx2 transmit region. Must be less than the TddFrameCount_us value.
- TddRx1OnPtr_us: start time of primary Rx1 receive region. Must be less than the TddFrameCount_us value.
- TddRx1OffPtr_us: start time of primary Rx1 receive region. Must be less than the TddFrameCount_us value.
- TddRx2OnPtr_us: start time of primary Rx2 receive region. Must be less than the TddFrameCount_us value.
- TddRx2OffPtr_us: start time of primary Rx2 receive region. Must be less than the TddFrameCount_us value.
- TddOrx1TxLoOnPtr_us: start time of primary ORx1 with Tx LO receive region. Must be less than the TddFrameCount_us value.
- TddOrx1TxLoOffPtr_us: stop time of primary ORx1 with Tx LO receive region. Must be less than the TddFrameCount_us value.
- TddOrx2TxLoOnPtr_us: start time of primary ORx2 with Tx LO receive region. Must be less than the TddFrameCount_us value.
- TddOrx2TxLoOffPtr_us: stop time of primary ORx2 with Tx LO receive region. Must be less than the TddFrameCount_us value.
- TddIntCalsOnPtr_us: start time of the primary ORx internal calibrations receive region. Must be less than the TddFrameCount_us value.
- TddIntCalsOffPtr_us: start time for the primary ObsRx internal calibrations receive region. Must be less than the TddFrameCount_us value.
- TddSnRxOnPtr_us: start time for the SnRx (A, B, or C) receive region. Must be less than the TddFrameCount_us value.
- TddSnRxOffPtr_us: stop time for the SnRx (A, B, or C) receive region. Must be less than the TddFrameCount_us value.
- TddOrx1SnifferLoOnPtr_us: start time for the ORx1 with SnRx LO receive region. Must be less than the TddFrameCount_us value.
- TddOrx1SnifferLoOffPtr_us: stop time for the ORx1 with SnRx LO receive region. Must be less than the TddFrameCount_us value.
- TddOrx2SnifferLoOnPtr_us: Start time for the ORx2 with SnRx LO receive region. Must be less than the TddFrameCount_us value.
- TddOrx2SnifferLoOffPtr_us: Stop time for the ORx2 with SnRx LO receive region. Must be less than the TddFrameCount_us value.

Secondary Region Pointers

The secondary region pointers include the following:

- Tdd2ndTx1OnPtr_us: start time of the secondary Tx1 transmit region. Must be less than the TddFrameCount_us value.
- Tdd2ndTx1OffPtr_us: stop time of the secondary Tx1 transmit region. Must be less than the TddFrameCount_us value.
- Tdd2ndTx2OnPtr_us: start time of the secondary Tx2 transmit region. Must be less than the TddFrameCount_us value.
- Tdd2ndTx2OffPtr_us: stop time of the secondary Tx2 transmit region. Must be less than the TddFrameCount_us value.
- Tdd2ndRx1OnPtr_us: start time of the secondary Rx1 receive region. Must be less than the TddFrameCount_us value.
- Tdd2ndRx1OffPtr_us: stop time of the secondary Rx1 receive region. Must be less than the TddFrameCount_us value.
- Tdd2ndRx2OnPtr_us: start time of the secondary Rx2 receive region. Must be less than the TddFrameCount_us value.
- Tdd2ndRx2OffPtr_us: stop time of the secondary Rx2 receive region. Must be less than the TddFrameCount_us value.
- Tdd2ndOrx1TxLoOnPtr_us: start time of the secondary ORx1 with Tx LO receive region. Must be less than the TddFrameCount_us value.
- Tdd2ndOrx1TxLoOffPtr_us: stop time of the secondary ORx1 with Tx LO receive region. Must be less than the TddFrameCount_us value.
- Tdd2ndOrx2TxLoOnPtr_us: start time of the secondary ORx2 with Tx LO receive region. Must be less than the TddFrameCount_us value.
- Tdd2ndOrx2TxLoOffPtr_us: Stop time of the secondary ORx2 with Tx LO receive region. Must be less than the TddFrameCount_us value.
- Tdd2ndIntCalsOnPtr_us: start time for the secondary ObsRx internal calibrations receive region. Must be less than the TddFrameCount_us value.

- Tdd2ndIntCalsOffPtr_us: stop time for the secondary ObsRx internal calibrations receive region. Must be less than the TddFrameCount_us value.

Delay Values (Secondary Datapath Delays (in Gray Blocks) are Ignored in the TDD FSM)

The delay values (secondary datapath delays are ignored in the time division duplexed, TDD, finite state machine, FSM) include the following:

- TddTx1DataPathDel_us: Tx1 primary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- TddTx2DataPathDel_us: Tx2 primary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- TddRx1DataPathDel_us: Rx1 primary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- TddRx2DataPathDel_us: Rx2 primary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- TddOrx1TxLoDataPathDel_us: ORx1 with Tx local oscillator (LO) primary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- TddOrx2TxLoDataPathDel_us: ORx2 with Tx LO primary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- Tdd2ndTx1DataPathDel_us: Tx1 secondary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- Tdd2ndTx2DataPathDel_us: Tx2 secondary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- Tdd2ndRx1DataPathDel_us: Rx1 secondary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- Tdd2ndRx2DataPathDel_us: Rx2 secondary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- Tdd2ndOrx1TxLoDataPathDel_us: ORx1 with Tx LO secondary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- Tdd2ndOrx2TxLoDataPathDel_us: ORx2 with Tx LO secondary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- TddSnRxDataPathDel_us: SnRx (A, B, or C) secondary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- TddOrx1SnifferLoPathDel_us: ORx1 with SnRx LO secondary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).
- TddOrx2SnifferLoPathDel_us: ORx2 with SnRx LO secondary start and stop time delay. Can be positive or negative. Maximum/minimum delay is abs(215/(TxLaneRate_MHz/40)).

Other Values

Other values include the following:

- TddExtTrigOnPtr_us: start time for the external trigger signal. The rising edge of the external trigger signal can indicate the beginning of a frame if set to 0.
- TddExtTrigOffPtr_us: stop time for the external trigger signal.

mykonosArmGpioConfig_t Data Structure Details

The mykonosArmGpioConfig_t data structure contains the following members. They are all data type uint8_t.

Configuration Modes

Configuration modes include the following:

- useRx2EnablePin: 0 = RX1_ENABLE controls Rx1 and Rx2. 1 = separate RX1_ENABLE/RX2_ENABLE pins.
- useTx2EnablePin: 0 = TX1_ENABLE controls TX1 and TX2. 1 = separate TX1_ENABLE/TX2_ENABLE pins.
- txRxPinMode: 0 = ARM command mode. 1 = pin mode to power up Tx/Rx chains.
- orxPinMode: 0 = ARM command mode. 1 = pin mode to power up ORx receiver.

Mykonos ARM Input GPIO Pins (Only Valid if orxPinMode = 1)

Mykonos ARM input GPIO pins (only valid if orxPinMode = 1) includes the following:

- orxTriggerPin: Select desired GPIO pin (valid 0 to 18).
- orxMode2Pin: Select desired GPIO pin (valid 0 to 18).
- orxMode1Pin: Select desired GPIO pin (valid 0 to 18).
- orxMode0Pin: Select desired GPIO pin (valid 0 to 18).

Mykonos ARM Output GPIO Pins (Always Available, Even When Pin Mode Not Enabled)

Mykonos ARM output GPIO pins (always available, even when pin mode is not available) include the following:

- rx1EnableAck: select desired GPIO pin (0 to 15), [4] = output enable.
- rx2EnableAck: select desired GPIO pin (0 to 15), [4] = output enable.
- tx1EnableAck: select desired GPIO pin (0 to 15), [4] = output enable.
- tx2EnableAck: select desired GPIO pin (0 to 15), [4] = output enable.
- orx1EnableAck: select desired GPIO pin (0 to 15), [4] = output enable.
- orx2EnableAck: select desired GPIO pin (0 to 15), [4] = output enable.
- srxEnableAck: select desired GPIO pin (0 to 15), [4] = output enable.
- txObsSelect: select desired GPIO pin (0 to 15), [4] = output enable. When two Tx are used with only one ORx input, this GPIO tells the baseband processor (BBP) which Tx channel is active for calibrations, so that the BBP can route the correct radio frequency (RF) Tx path into the single ORx input.

GENERAL-PURPOSE INPUT/OUTPUT (GPIO) CONFIGURATION

The device provides 19 general-purpose input/output (GPIO) capable signals that can be used for a variety of control functions. These signals can be configured using the transceiver evaluation software (TES) to demonstrate the available configurations. This user guide refers to these signals as GPIO_x, where x is the GPIO number 0 through 18. All 19 signals have output buffers powered from the VDD_IF domain. The voltage range for these outputs is the same as the VDD_IF supply: 1.8 V to 2.5 V.

In addition to being used as general-purpose control signals, certain GPIO pins can be used as real-time control signals that provide operational details from the device to the baseband processor (BBP) when configured as outputs, enabling transceiver performance monitoring in different situations. The application programming interface (API) functions created to manage this block allow the user to configure pins as inputs or outputs for specific functions. This section describes the GPIO signals and their behavior in detail, while also describing how to program the device using the API functions so that the desired signals are available on the appropriate pins.

Figure 93 outlines different functionalities that can be enabled in the device and then controlled using the GPIO interface. Not all functionalities can be enabled at the same time. Use the TES to preconfigure API structures, ensuring that the desired combination is possible, and that there are no conflicts on the GPIO interface.

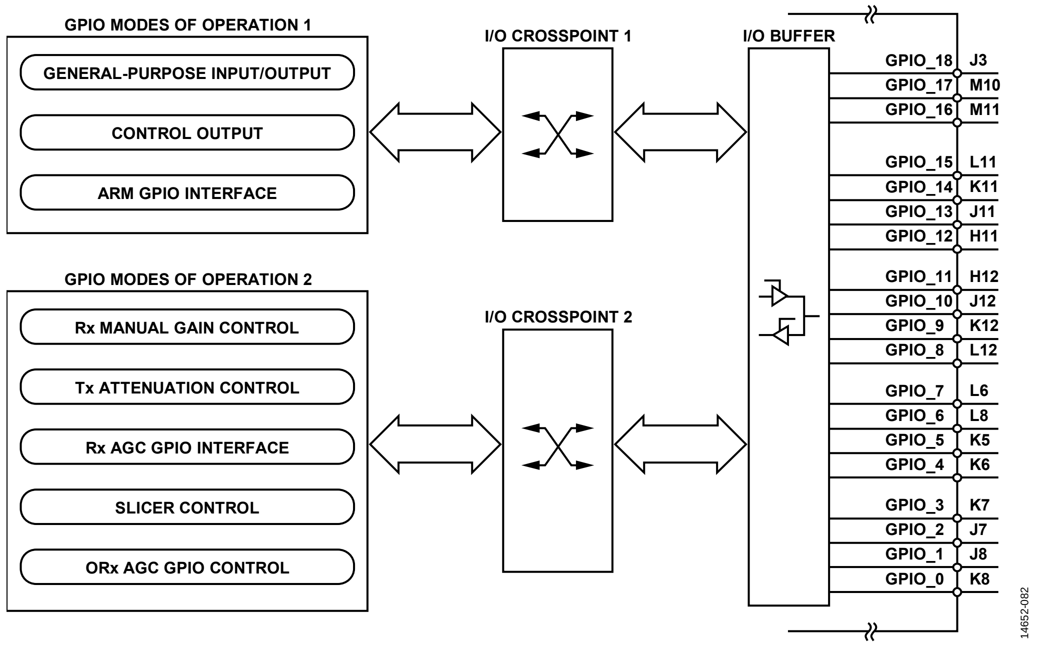


Figure 93. Overview of the GPIO Modes of Operation

GPIO OPERATION

Figure 94 outlines the flow of events to follow while configuring the GPIO in a desired configuration. In general, all general-purpose input/output (GPIO) configuring can be performed in the radio on or radio off state, except ARM GPIO mode. ARM GPIO mode can only be configured in the radio off state. Use the transceiver evaluation software (TES) to set member values for all GPIO structures, ensuring that there is no conflict among the different GPIO functionalities.

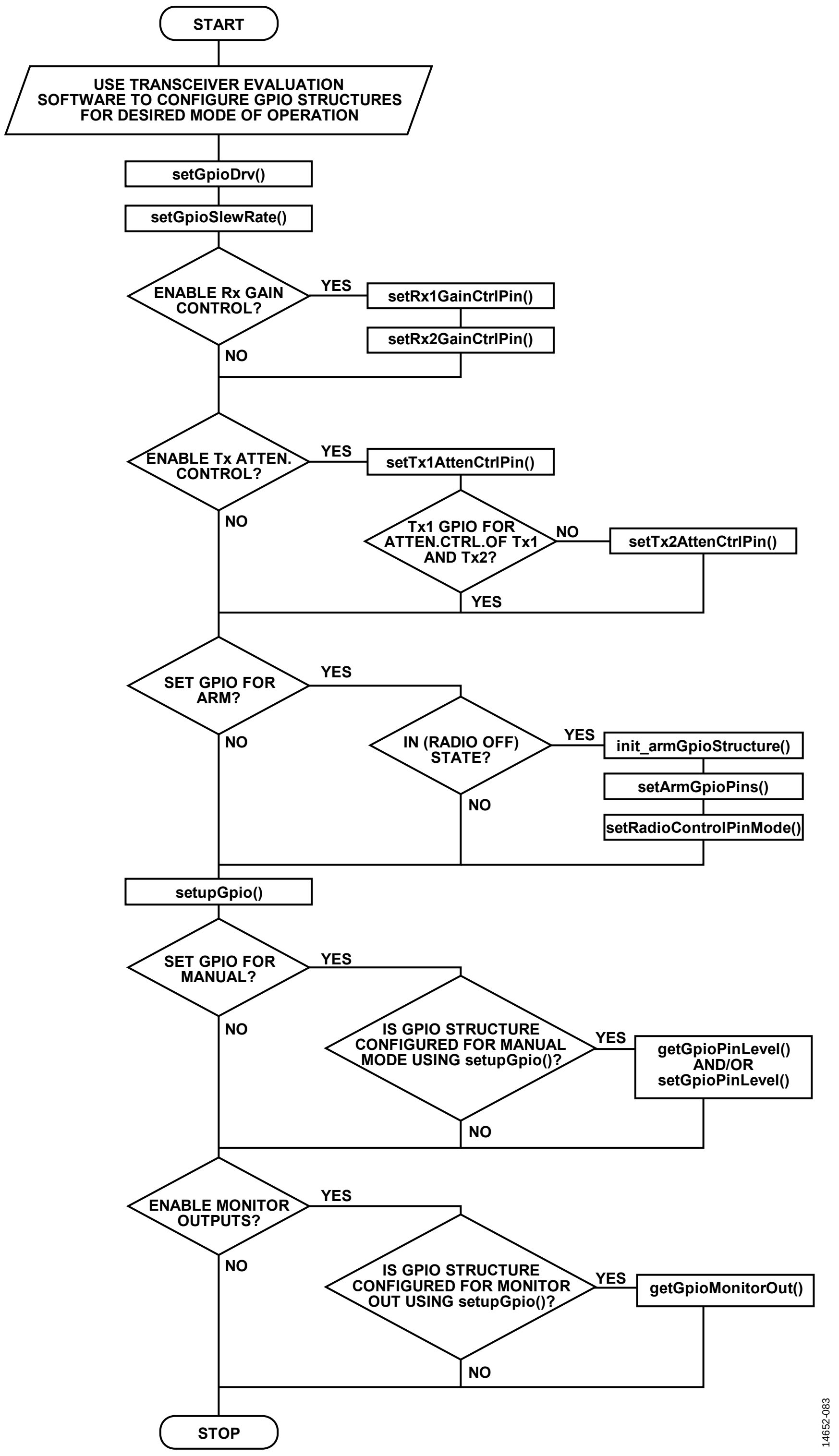


Figure 94. GPIO Interface Configuration and Control Flowchart

API Description

As shown in Figure 93, there are many functionalities available that can be enabled and to interact with the baseband processor (BBP) over the general-purpose input/output (GPIO) interface. Two separate crosspoint switches and one buffer must be configured properly to enable a particular functionality. The application programming interface (API) package includes functions that can configure all blocks to desired states. The following sections contain general API functions and short descriptions of their functionalities.

MYKONOS_setupGpio

```
mykonosGpioErr_t
MYKONOS_setupGpio(mykonosDevice_t *device)
```

This function sets the GPIO configuration registers. It configures the pin direction for each GPIO, as well as the crosspoints. This function relies on the correct settings of the device → auxIo → gpio structure members. An overview of this structure follows:

```
typedef struct
{
    uint32_t  gpioOe;                      /*!< Output Enable per low voltage GPIO pin
(1=output, 0=input) */
    mykonosGpioMode_t  gpioSrcCtrl3_0;    /*!< Mode for low voltage GPIO[3:0] pins */
    mykonosGpioMode_t  gpioSrcCtrl7_4;    /*!< Mode for low voltage GPIO[7:4] pins */
    mykonosGpioMode_t  gpioSrcCtrl11_8;   /*!< Mode for low voltage GPIO[11:8] pins */
    mykonosGpioMode_t  gpioSrcCtrl15_12;  /*!< Mode for low voltage GPIO[15:12] pins */
    mykonosGpioMode_t  gpioSrcCtrl18_16;  /*!< Mode for low voltage GPIO[18:16] pins */
} mykonosGpioLowVoltage_t;
```

The gpioOe function controls configuration of the I/O buffer shown in Figure 93. The gpioSrcCtrlNN_nn function controls the configuration of I/O Crosspoint 1, as shown in Figure 93. The values for gpioSrcCtrlNN_nn are as follows:

- GPIO_MONITOR_MODE. This value allows a choice of debug signals to output from the device to monitor the state of the device. See the Monitor Output section for more details.
- GPIO_BITBANG_MODE—(manual mode). This application programming interface (API) function sets the output pin levels and reads the input pin levels. See the GPIO Manual Mode section for more details.
- GPIO_ARM_OUT_MODE. This value allows the internal ARM processor to output on the GPIO pins. See the ARM GPIO Interface section for more details.
- GPIO_SLICER_OUT_MODE. This value assigns the slicer active configuration to the GPIO output pins. See the ARM GPIO Interface section for more details.

Use the transceiver evaluation software (TES) to set member values of this structure, which ensures that there are no conflicts with other GPIO functions.

MYKONOS_setGpioOe

```
mykonosGpioErr_t
   MYKONOS_setGpioOe(mykonosDevice_t
   *device, uint32_t gpioOutEn, uint32_t
   gpioUsedMask)
```

This is a helper function called by the MYKONOS_setupGpio() function (this helper function must not be called by the user).

This function sets the GPIO direction given by the passed parameter. This direction can be either output or input. The gpioUsedMask parameter allows the function to affect only the GPIO pins of interest.

Parameters

- gpioOutEn: The valid range for this variable is from 0x0 to 0x07FFFF. Each bit represents the corresponding GPIO pin (Bit 0 represents GPIO_0, Bit 1 represents GPIO_1, and so on). The direction of the input buffer is set by each bit value: 0 indicates input, and 1 indicates output.
- gpioUsedMask: This parameter is the mask used to control which bits are set/cleared. If a mask bit = 1, that bit is modified by the value in GPIOOUTEN.

MYKONOS_getGpioOe

```
mykonosGpioErr_t
   MYKONOS_getGpioOe(mykonosDevice_t
   *device, uint32_t *gpioOutEn)
```

This function reads back the current status of the GPIO direction set in the device. The direction can be either output or input. The function parameter returns a bit per GPIO pin where 1 indicates output and 0 indicates input

Parameters

- *gpioOutEn: This parameter is a pointer to the data to be returned with the output enable GPIO pins.

MYKONOS_setGpioSourceCtrl

```
mykonosGpioErr_t
   MYKONOS_setGpioSourceCtrl(mykonosDevice_t
   *device, uint32_t gpioSrcCtrl)
```

This is a helper function called by the MYKONOS_setupGpio() function (the helper function must not be called by the user).

This function configures crosspoints for different GPIO functionality. This function only affects the GPIO pins that have their output enable direction set by the MYKONOS_getGpioOe() function as outputs.

Parameters

- gpioSrcCtrl: This parameter is a nibble-based source control, and it is a 32-bit value containing five nibbles that set the source control. Use the TES to generate correct settings for a desired configuration.

MYKONOS_getGpioSourceCtrl

```
mykonosGpioErr_t
   MYKONOS_getGpioSourceCtrl(mykonosDevice_t
   *device, uint32_t *gpioSrcCtrl)
```

This function reads back the current status of the GPIO source control for different GPIO functionality.

Parameters

- *gpioSrcCtrl: This parameter is a nibble-based source control, and it is a 32-bit value containing five nibbles that represent the current settings of the source control (crosspoint configuration).

MYKONOS_setGpioDrv

```
mykonosGpioErr_t
   MYKONOS_setGpioDrv(mykonosDevice_t
   *device, mykonosGpioSelect_t gpioDrv)
```

This function configures the drive strength of each GPIO. This function only affects the GPIO pins that have their OE direction set by the MYKONOS_getGpioOe() function as outputs.

Parameters

- gpioDrv: This parameter indicates which GPIO is to be selected to set its drive strength. If the bit in gpioDrv corresponds to particular a GPIO is set to 0, the drive strengths of this GPIO is set as described in the data sheet. If a bit in gpioDrv corresponds to a particular GPIO is set to 1, the drive strengths of this GPIO is doubled, compare to the description in the data sheet. The valid range of this parameter is from 0x00000 to 0x7FFFF. There are limitations for the way drive strength can be configured. Table 141 outlines these limitations in the configuration flexibility.

The range of GPIO pins from GPIO_8 to GPIO_17 can only be selected in pairs. Setting one of the corresponding bits automatically selects the other one, meaning, for example, if it is required to double the drive strength for GPIO_17, the drive strength is also doubled for GPIO_16, and so on.

Table 141. GPIO Configuration Limitations—gpioDrv

GPIO Pin	Corresponding Bits in the gpioDrv Parameter
GPIO_0	xxx xxxx xxxx xxxx xxxN
GPIO_1	xxx xxxx xxxx xxxx xxNx
…	xxx xxxx xxxx xNNN NNxx
GPIO_7	xxx xxxx xxxx Nxxx xxxx
GPIO_8	xxx Nxxx xxxN xxxx xxxx
GPIO_9	xxx xxxx xNNx xxxx xxxx
GPIO_10	xxx xxxx xNNx xxxx xxxx
GPIO_11	xxx xxxN Nxxx xxxx xxxx
GPIO_12	xxx xxxN Nxxx xxxx xxxx
GPIO_13	xxx xNNx xxxx xxxx xxxx
GPIO_14	xxx xNNx xxxx xxxx xxxx
GPIO_15	xxx Nxxx xxxN xxxx xxxx
GPIO_16	xNN xxxx xxxx xxxx xxxx
GPIO_17	xNN xxxx xxxx xxxx xxxx
GPIO_18	Nxx xxxx xxxx xxxx xxxx

MYKONOS_getGpioDrv

```
mykonosGpioErr_t
   MYKONOS_getGpioDrv(mykonosDevice_t
   *device, mykonosGpioSelect_t *gpioDrv)
```

This function reads back the current status of the GPIO drive strength setting, set by the MYKONOS_setGpioDrv() function.

Parameters

- *gpioDrv: This parameter is a pointer to the data to be returned with the current GPIOs drive strength setting. Refer to the MYKONOS_setGpioDrv() function description for bit field interpretation.

MYKONOS_setGpioSlewRate

```
mykonosGpioErr_t
   MYKONOS_setGpioSlewRate(mykonosDevice_t
   *device, mykonosGpioSelect_t gpioSelect,
   mykonosGpioSlewRate_t slewRate)
```

This function configures the slew rate of the selected GPIO. This function only affects the GPIO pins that have their output enable direction set by the MYKONOS_getGpioOe() function as outputs.

Parameters

- gpioSelect: This parameter indicates which GPIO is to be selected to set its slew rate. If the bit in gpioSelect that corresponds to a particular GPIO is set to 0, its slew rate does not change. If the bit in gpioSelect that corresponds to a particular GPIO is set to 1, its slew rate changes to the value selected by the slewRate parameter. The valid range for this parameter is from 0x00000 to 0x7FFFF. There are limitations in the way that GPIO can be selected for its slew rate settings. Table 142 outlines these limitations in the configuration flexibility.
- slewRate: This parameter contains information of slew rate that to be applied to the GPIO selected using the gpioSelect parameter.The valid slew rate settings are given by the enumeration type mykonosGpioSlewRate_t.
  - MYK_SLEWRATE_NONE—lower slew rate for the selected GPIO.
  - MYK_SLEWRATE_LOW—low slew rate for the selected GPIO.
  - MYK_SLEWRATE_MEDIUM—medium slew rate for the selected GPIO.
  - MYK_SLEWRATE_HIGH—high slew rate for the selected GPIO

The GPIO pins from GPIO_8 to GPIO_17 can only be selected in pairs. Setting one bit automatically selects the other one. For example, if it is required to set the slew rate for GPIO_17, the same slew rate is also applied for GPIO_16, and so on.

MYKONOS_getGpioSlewRate

```
mykonosGpioErr_t
   MYKONOS_getGpioSlewRate(mykonosDevice_t
   *device, mykonosGpioSelect_t gpioSelect,
   mykonosGpioSlewRate_t *slewRate)
```

This function reads back the current status of the GPIO slew rate setting set by the MYKONOS_setGpioSlewRate() function.

Parameters

- gpioSelect: This parameter indicates which GPIO is selected to read back its programmed slew rate settings Each bit in gpioSelect corresponds to a particular GPIO. GPIO_0 is selected by Bit 0, GPIO_1 is selected by Bit 1, and so on. Setting a bit to 1 selects the corresponding GPIO. Only a single GPIO can be selected at one time. The valid range on this parameter is from 0x00000 to 0x7FFFF.
- *slewRate: This parameter is a pointer to the data to be returned with the current slew rate programmed for the GPIO selected by the gpioSelect parameter. Refer to the MYKONOS_setGpioSlewRate() function for a description of the mykonosGpioSlewRate_t enumeration type.

Table 142. GPIO Configuration Limitations—gpioSelect

GPIO Pin	Corresponding Bits in the gpioDrv Parameter
GPIO_0	xxx xxxx xxxx xxxx xxxN
GPIO_1	xxx xxxx xxxx xxxx xxNx
…	xxx xxxx xxxx xNNN NNxx
GPIO_7	xxx xxxx xxxx Nxxx xxxx
GPIO_8	xxx Nxxx xxxN xxxx xxxx
GPIO_9	xxx xxxx xNNx xxxx xxxx
GPIO_10	xxx xxxx xNNx xxxx xxxx
GPIO_11	xxx xxxN Nxxx xxxx xxxx
GPIO_12	xxx xxxN Nxxx xxxx xxxx
GPIO_13	xxx xNNx xxxx xxxx xxxx
GPIO_14	xxx xNNx xxxx xxxx xxxx
GPIO_15	xxx Nxxx xxxN xxxx xxxx
GPIO_16	xNN xxxx xxxx xxxx xxxx
GPIO_17	xNN xxxx xxxx xxxx xxxx
GPIO_18	Nxx xxxx xxxx xxxx xxxx

GPIO MANUAL MODE

Figure 95 shows the GPIO_x signals in GPIO manual mode. In this mode, the user can configure GPIOs as outputs or inputs.

If a GPIO pin is configured as an output, the user can also control its logic level. If a GPIO pin is configured as an input, the user can read back the voltage level present at the input.

In both cases, low (or 0) corresponds to the ground level and high (or 1) corresponds to the VDD_IF voltage level.

Note that two I/O crosspoints (Crosspoint 1 and Crosspoint 2) access a single I/O buffer (see Figure 93). Crosspoint 2 has a higher priority when accessing the I/O buffer. To configure the device for GPIO manual mode only, or for manual mode when working with any other GPIO mode in parallel, use the transceiver evaluation software (TES) to preconfigure the application programming interface (API) auxIo/gpio structure with valid configurations prior to writing the code to accomplish this function.

Manual mode uses Crosspoint 1. This crosspoint operates using nibbles (4 bits); the GPIOs are controlled in groups of four. The exception to this is that the three most significant GPIOs operate in a block of three (GPIO_16, GPIO_17, and GPIO_18).

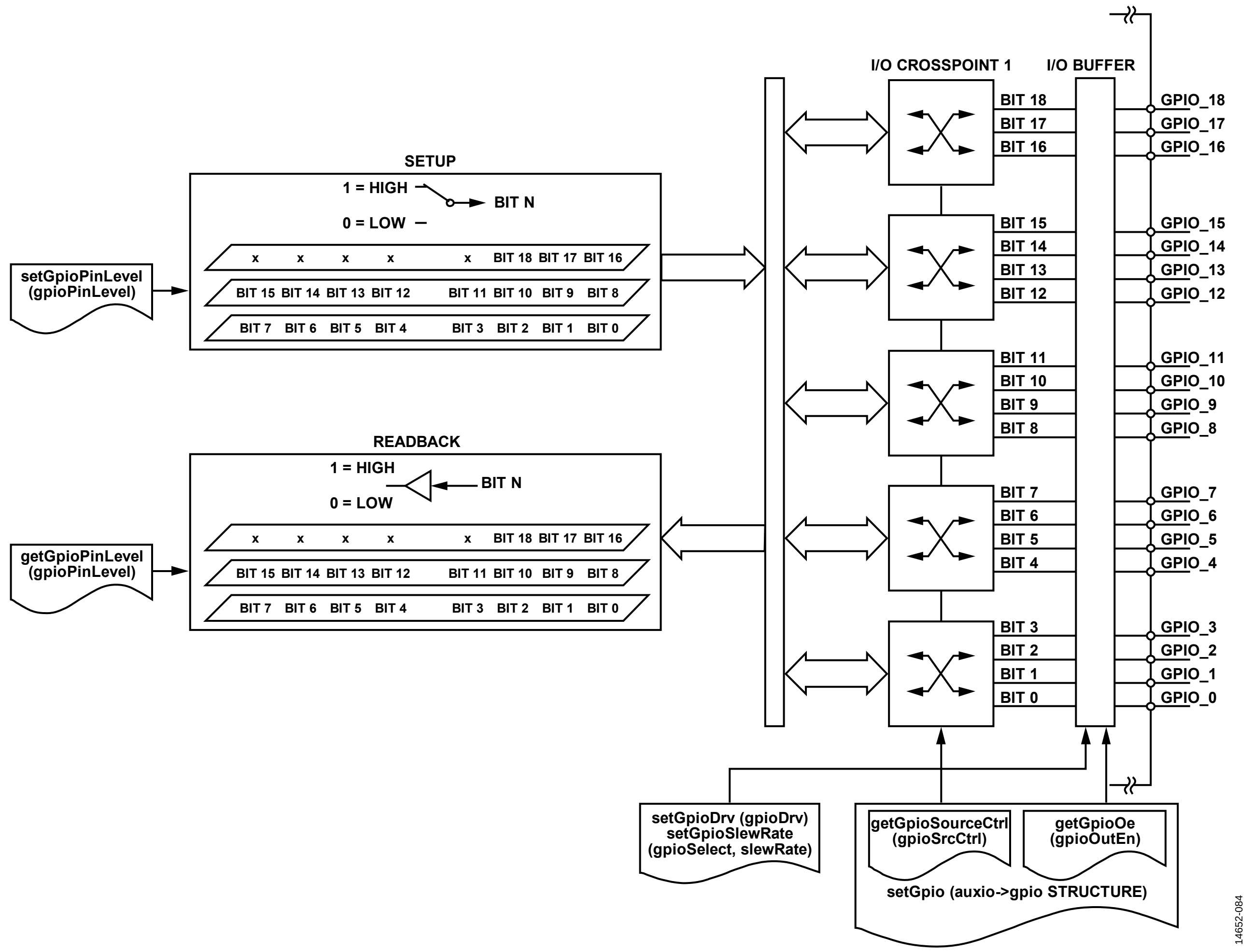

I/O CROSSPOINT 1
I/O BUFFER
SETUP
1 = HIGH
0 = LOW
BIT N
x x x x x BIT 18 BIT 17 BIT 16
BIT 15 BIT 14 BIT 13 BIT 12 BIT 11 BIT 10 BIT 9 BIT 8
BIT 7 BIT 6 BIT 5 BIT 4 BIT 3 BIT 2 BIT 1 BIT 0
setGpioPinLevel (gpioPinLevel)
READBACK
getGpioPinLevel (gpioPinLevel)
GPIO_18
GPIO_17
GPIO_16
GPIO_15
GPIO_14
GPIO_13
GPIO_12
GPIO_11
GPIO_10
GPIO_9
GPIO_8
GPIO_7
GPIO_6
GPIO_5
GPIO_4
GPIO_3
GPIO_2
GPIO_1
GPIO_0
setGpioDrv (gpioDrv)
setGpioSlewRate (gpioSelect, slewRate)
getGpioSourceCtrl (gpioSrcCtrl)
getGpioOe (gpioOutEn)
setGpio (auxio->gpio STRUCTURE)
14652-084

Figure 95. GPIO Manual Mode

API Description

The application programming interface (API) provides functions designed to operate the GPIO in manual mode. Before the user can employ manual mode, the crosspoints and the I/O buffer must be configured. Use the MYKONOS_setupGpio() function to properly configure the crosspoints and the I/O buffer. This function relies on correct configuration of device → auxIo → gpio structure members. Use the transceiver evaluation software (TES) to generate correct settings for this structure. After the crosspoints and I/O buffer are configured, the user can start operating the GPIOs in manual mode. The following sections list the manual mode API functions, along with short descriptions of their functionalities.

MYKONOS_getGpioPinLevel

```
mykonosGpioErr_t
   MYKONOS_getGpioPinLevel(mykonosDevice_t
   *device, uint32_t *gpioPinLevel)
```

This function reads the GPIO pins that are set as inputs in manual mode. Pin levels are returned in the form of a single 32-bit word. The return value is 1 bit per pin. The GPIO_0 level returns on Bit 0 of the gpioPinLevel parameter, the GPIO_1 level returns on Bit 1, and so on. A logic low level returns a 0, and a logic high level returns a 1.

Parameters

- *gpioPinLevel: the input GPIO pin levels read back on the pins assigned as inputs (1 bit per pin).

MYKONOS_setGpioPinLevel

```
mykonosGpioErr_t
   MYKONOS_setGpioPinLevel(mykonosDevice_t
   *device, uint32_t gpioPinLevel)
```

This function sets the GPIO output pin levels. This function only affects the GPIO pins that are set as outputs in manual mode. GPIO_0 reflects the Bit 0 status of the gpioPinLevel parameter, GPIO_1 reflects the Bit 1 status, and so on. A logic low is set at the GPIO output if its corresponding bit is set to 0. A logic high is set at the GPIO output if its corresponding bit is set to 1.

Parameters

- gpioPinLevel: This parameter describes the output level for each bit per GPIO pin (0 = low output, and 1 = high output).

MYKONOS_getGpioSetLevel

```
mykonosGpioErr_t
   MYKONOS_getGpioSetLevel(mykonosDevice_t
   *device, uint32_t *gpioPinSetLevel)
```

This function provides readback of the value describing how GPIO output pins are set in the manual mode.

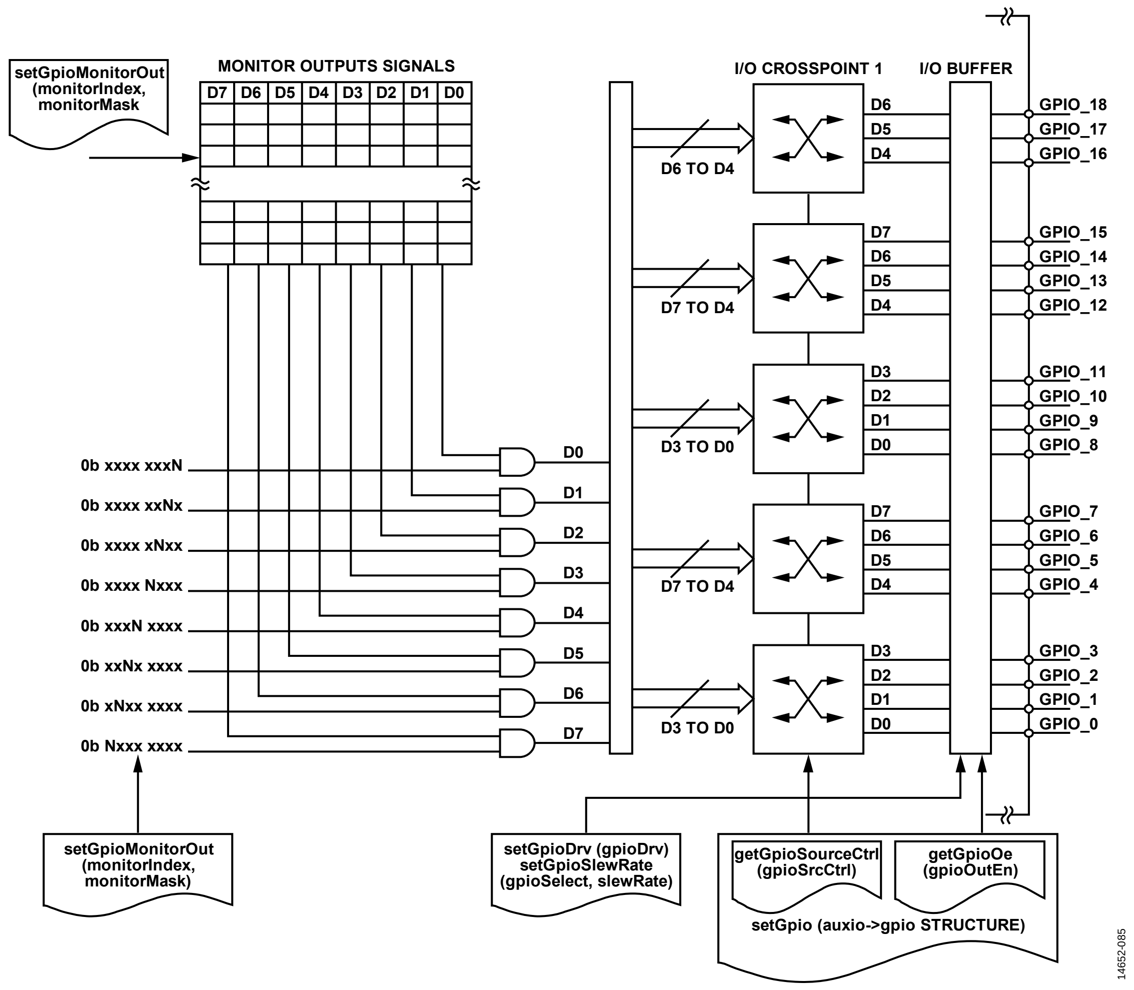
Parameters

- *gpioPinSetLevel: This parameter is a pointer to the 32-bit variable that contains the level of each GPIO pin, 1 bit per pin (0 = sets output to low level, and 1 = sets output to high level).

MONITOR OUTPUT

The device offers the capability to output a real-time status for a variety of internal conditions or states through the GPIO interface to the baseband processor (BBP). Information, such as the state of the overload detectors in the receiver signal path or automatic gain control (AGC) states, are just a few of the many options available. This section describes these signals and also how to use API functions to make the desired signals available on the appropriate GPIO pins. Figure 96 shows the possible connections between the monitor output signals and the GPIO pins. Note that the logic block format of the figure shows the functional operation of the GPIO signals, but it does not necessarily represent the method of implementation inside the device.

The monitor output signals allow the user to monitor selected internal device functions by outputting a single row from the monitor output signal table. Table 143 lists the available signal combinations that can be routed to the selected GPIO pins. The user can only monitor the signals in one row at a time. Selection of one signal over another depends on which other signals the BBP must monitor simultaneously. Some internal signals are available on more than one table row using different GPIO assignments. Some of the signals are helpful in a production system, while others are useful for debugging purposes. In either case, Analog Devices recommends connecting the device monitor outputs to the BBP inputs so that the signals can be monitored under real-time conditions.



setGpioMonitorOut (monitorIndex, monitorMask
MONITOR OUTPUTS SIGNALS
D7 D6 D5 D4 D3 D2 D1 D0
0b xxxx xxxN
0b xxxx xxNx
0b xxxx xNxx
0b xxxx Nxxx
0b xxxN xxxx
0b xxNx xxxx
0b xNxx xxxx
0b Nxxx xxxx
D0 D1 D2 D3 D4 D5 D6 D7
I/O CROSSPOINT 1
I/O BUFFER
D6 TO D4
D7 TO D4
D3 TO D0
D7 TO D4
D3 TO D0
GPIO_18 GPIO_17 GPIO_16
GPIO_15 GPIO_14 GPIO_13 GPIO_12
GPIO_11 GPIO_10 GPIO_9 GPIO_8
GPIO_7 GPIO_6 GPIO_5 GPIO_4
GPIO_3 GPIO_2 GPIO_1 GPIO_0
setGpioMonitorOut (monitorIndex, monitorMask)
setGpioDrv (gpioDrv) setGpioSlewRate (gpioSelect, slewRate)
getGpioSourceCtrl (gpioSrcCtrl)
getGpioOe (gpioOutEn)
setGpio (auxio->gpio STRUCTURE)
14652-085

Figure 96. Monitor Output Signal Routing

Table 143. Monitor Output Signals

Monitor Index	Bits							
	D7	D6	D5	D4	D3	D2	D1	D0
0x01	Rx1: gain change (OR of gain increment/ gain decrement)	Rx1: APD upper threshold counter exceeded (slow loop)	Rx1: HB2 upper threshold counter exceeded	Rx1: HB2 lower threshold counter exceeded	Rx2: gain change (OR of gain increment/ gain decrement)	Rx2: APD upper threshold counter exceeded	Rx2: HB2 upper threshold counter exceeded	Rx2: HB2 lower threshold counter exceeded
0x02	Rx1: gain change (OR of gain increment/ gain decrement)	Rx1: APD upper threshold exceeded	Rx1: HB2 upper threshold exceeded	Rx1: digital saturation	Rx2: gain change (OR of gain increment/ gain decrement)	Rx2: APD upper threshold exceeded	Rx2: HB2 upper threshold exceeded	Rx2: digital saturation
0x03	Rx1: gain lock	Rx1: APD upper threshold exceeded	Rx1: HB2 upper threshold exceeded	Rx1: energy lost	Rx2: gain lock	Rx2: APD upper threshold exceeded	Rx2: HB2 upper threshold exceeded	Rx2: energy lost
0x04	Rx1: low threshold exceeded	Rx1: high threshold exceeded	Rx1: gain update counter expired	Reserved		Rx1: gain change	Rx1: gain change increment	Rx1: gain change decrement
0x05	Rx2: low threshold exceeded	Rx2: high threshold exceeded	Rx2: gain update counter expired	Reserved		Rx2: gain change	Rx2: gain change increment	Rx2: gain change decrement
0x06	Rx1: gain change increment	Rx1: gain change decrement	Rx2: gain change increment	Rx2: gain change decrement	Reserved			

Monitor Index	Bits							
	D7	D6	D5	D4	D3	D2	D1	D0
0x07	Rx1: APD upper threshold counter exceeded (slow loop)	Rx1: HB2 upper threshold counter exceeded	Rx1: gain update counter expired	Rx1: gain change	Rx2: APD upper threshold counter exceeded	Rx2: HB2 upper threshold counter exceeded	Rx2: gain update counter expired	Rx2: gain change
0x08	Rx1: decrement power ready	Rx1: gain update counter expired	Rx1: APD upper threshold counter exceeded (slow loop)	Rx1: HB2 upper threshold counter exceeded	Rx2: decimate power ready	Rx2: gain update counter expired	Rx2: APD upper threshold counter exceeded	Rx2: HB2 upper threshold counter exceeded
0x09	Rx1: gain index (Bit D7)	Rx1: gain index (Bit D6)	Rx1: gain index (Bit D5)	Rx1: gain index (Bit D4)	Rx1: gain index (Bit D3)	Rx1: gain index (Bit D2)	Rx1: gain index (Bit D1)	Rx1: gain index (Bit D0)
0x0A	Rx2: gain index (Bit D7)	Rx2: gain index (Bit D6)	Rx2: gain index (Bit D5)	Rx2: gain index (Bit D4)	Rx2: gain index (Bit D3)	Rx2: gain index (Bit D2)	Rx2: gain index (Bit D1)	Rx2: gain index (Bit D0)
0x0C	Rx1: gain index (Bit D3)	Rx1: gain index (Bit D2)	Rx1: gain index (Bit D1)	Rx1: gain index (Bit D0)	Rx2: gain index (Bit D3)	Rx2: gain index (Bit D2)	Rx2: gain index (Bit D1)	Rx2: gain index (Bit D0)
0x22	Tx2: RFIR overflow	Tx2: Tx HB2 overflow	Tx2: Tx HB1 overflow	Tx2: TFIR overflow	Tx1: RFIR overflow	Tx1: Tx HB2 overflow	Tx1: Tx HB1 overflow	Tx1: TFIR overflow
0x23	Reserved	Rx1: Signal of interest (SOI) present	Rx1: correction word above threshold	Rx1: update counter expired	Rx1: gain change	Rx1: update dc offset in the RF section	Rx1: measure dc offset in the RF section	Rx1: RF dc count reached
0x24	Reserved	Rx2: SOI present	Rx2: correction word above threshold	Rx2: update counter expired	Rx2: gain change	Rx2: update dc offset in the RF section	Rx2: measure dc offset in the RF section	Rx2: RF dc count reached
0x42	Rx2 overrange very low counter exceeded	Rx2 overrange low counter exceeded	Rx2 overrange high counter exceeded	Rx2 gain update counter expired	Rx1 overrange very low counter exceeded	Rx1 overrange low counter exceeded	Rx1 overrange high counter exceeded	Rx1 gain update counter expired
0x43	Reserved				ORx overrange very low counter exceeded	ORx overrange low counter exceeded	ORx overrange high counter exceeded	ORx gain update counter expired

Table 144 through Table 159 provide the detailed descriptions of the signals available for each monitor index address listed in Table 143.

Table 144. Monitor Index: 0x01 (Name: AGC Monitor 1)

Bits	Description	Reset
D7	Rx1: gain change. The gain index (in automatic gain control (AGC) and manual gain control (MGC) mode) is used to select an entry in the gain table. This signal is active (output level toggle between high and low) only if there is a difference in entries in the gain table when the AGC or MGC index changes. If the entries of the two indices are identical, the Rx gain change signal is not generated (toggled).	0x0
D6	Rx1 APD upper threshold counter exceeded. The Rx1 analog peak detector (APD) is set when the upper threshold counter is exceeded. This bit operates in real-time.	0x0
D5	Rx1 HB2 upper threshold counter exceeded. Set this bit when the Rx1 ADC/High-Band 2 (HB2) upper threshold overflow counter is exceeded.	0x0
D4	Rx1 HB2 lower threshold counter exceeded. Set this bit when the Rx1 ADC/HB2 lower threshold overflow counter is exceeded.	0x0
D3	Rx2: gain change. The gain index (in AGC and MGC mode) selects an entry in the gain table. This signal is active (the output level toggles between high and low) only if there is a difference in entries in the gain table when the AGC or MGC index changes. If the entries of the two indices are identical, the Rx gain change signal is not generated (toggled).	0x0
D2	Rx2 APD upper threshold counter exceeded. The Rx2 APD set when the upper threshold counter is exceeded. This bit operates in real-time.	0x0
D1	Rx2 HB2 upper threshold counter exceeded. Set this bit when the Rx2 ADC/HB2 upper threshold overflow counter is exceeded.	0x0
D0	Rx2 HB2 lower threshold counter exceeded. Set this bit when the Rx2 ADC/HB2 lower threshold overflow counter is exceeded.	0x0

Table 145. Monitor Index: 0x02 (Name: AGC Monitor 2)

Bits	Description	Reset
D7	Rx1: gain change. The gain index (in automatic gain control (AGC) and manual gain control (MGC) mode) is used to select an entry in the gain table. This signal is active (the output level toggles between high and low) only if there is a difference in entries in the gain table when the AGC or MGC index changes. If the entries of the two indices are identical, the Rx gain change signal is not generated (toggled).	0x0
D6	Rx1: analog peak detector (APD) upper threshold exceeded. Set this bit when the upper threshold is exceeded. This bit operates in real-time.	0x0
D5	Rx1: High-Band 2 (HB2) upper threshold. Set this bit when high threshold detector overflow occurs.	0x0
D4	Rx1: digital saturation. Set this bit when the signal is saturated after digital gain is applied.	0x0
D3	Rx2: gain change. The gain index (in AGC and MGC mode) is used to select an entry in the gain table. This signal is active (the output level toggles between high and low) only if there is a difference in entries in the gain table when the AGC or MGC index changes. If the entries of the two indices are identical, the Rx gain change signal is not generated (toggled).	0x0
D2	Rx2: APD upper threshold exceeded. Set this bit when the upper threshold is exceeded. This bit operates in real-time.	0x0
D1	Rx2: HB2 upper threshold exceeded. Set this bit when high threshold detector overflow occurs.	0x0
D0	Rx2 digital saturation. Set this bit when the signal is saturated after digital gain is applied.	0x0

Table 146. Monitor Index: 0x03 (Name: AGC Monitor 3)

Bits	Description	Reset
D7	Rx1: gain lock. Set this bit when the gain locks.	0x0
D6	Rx1: APD upper threshold exceeded. Set this bit when the upper threshold is exceeded. This bit operates in real-time.	0x0
D5	Rx1: HB2 upper threshold. Set this bit when high threshold detector overflow occurs.	0x0
D4	Rx1: energy lost. Set this bit when the signal power change exceeds the lower threshold.	0x0
D3	Rx2: gain lock. Set this bit when the gain locks.	0x0
D2	Rx2: APD upper threshold exceeded. Set this bit when the upper threshold is exceeded. This bit operates in real-time.	0x0
D1	Rx2: HB2 upper threshold. Set this bit when high threshold detector overflow occurs.	0x0
D0	Rx2: energy lost. Set this bit when the signal power change exceeds the threshold.	0x0

Table 147. Monitor Index: 0x04 (Name: AGC Monitor 4)

Bits	Description	Reset
D7	Rx1: low threshold exceeded. Set this bit when the low power threshold is exceeded.	0x0
D6	Rx1: high threshold exceeded. Set this bit when the high power threshold is exceeded.	0x0
D5	Rx1: gain update counter expired. This bit signals when the gain update counter expires.	0x0
[D4:D3]	Reserved.	0x0
D2	Rx1: gain change. The gain index (in AGC and MGC mode) is used to select an entry in the gain table. This signal is active (the output level toggles between high and low) only if there is a difference in entries in the gain table when the AGC or MGC index changes. If the entries of the two indices are identical, the Rx gain change signal is not generated (toggled).	0x0
D1	Rx1: gain change increment. This bit toggles when the gain increases.	0x0
D0	Rx1: gain change decrement. This bit toggles when the gain decreases.	0x0

Table 148. Monitor Index: 0x05 (Name: AGC Monitor 5)

Bits	Description	Reset
D7	Rx2: low threshold exceeded. Set this bit when the low power threshold is exceeded.	0x0
D6	Rx2: high threshold exceeded. Set this bit when the high power threshold is exceeded.	0x0
D5	Rx2: gain update counter expired. This bit signals when the gain update counter expires.	0x0
[D4:D3]	Reserved.	0x0
D2	Rx2: gain change. This bit toggles when the gain changes value.	0x0
D1	Rx2: gain change increment. This bit toggles when the gain increases.	0x0
D0	Rx2: gain change decrement. This bit toggles when the gain decreases.	0x0

Table 149. Monitor Index: 0x06 (Name: AGC Monitor 6)

Bits	Description	Reset
D7	Rx1: gain change increment. This bit toggles when the gain increases.	0x0
D6	Rx1: gain change decrement. This bit toggles when the gain decreases.	0x0
D5	Rx2: gain change increment. This bit toggles when the gain increases.	0x0
D4	Rx2: gain change decrement. This bit toggles when the gain decreases.	0x0
[D3:D0]	Reserved.	0x0

Table 150. Monitor Index: 0x07 (Name: AGC Monitor 7)

Bits	Description	Reset
D7	Rx1 APD upper threshold counter exceeded. The Rx1 analog peak detector (APD) is set when the upper threshold counter is exceeded. This bit operates real-time.	0x0
D6	Rx1 HB2 upper threshold counter exceeded. Set this bit when the Rx1 ADC/High-Band 2 (HB2) upper threshold overflow counter is exceeded.	0x0
D5	Rx1: gain update counter expired. This bit signals when the gain update counter expires.	0x0
D4	Rx1: gain change. The gain index (in automatic gain control (AGC) and manual gain control (MGC) mode) is used to select an entry in the gain table. This signal is active (the output level toggles between high and low) only if there is a difference in entries in the gain table when the AGC or MGC index changes. If the entries of the two indices are identical, the Rx gain change signal is not generated (toggled).	0x0
D3	Rx2 APD upper threshold counter exceeded. The Rx2 analog peak detector (APD) is set when the upper threshold counter is exceeded. This bit operates real-time.	0x0
D2	Rx2 HB2 upper threshold counter exceeded. Set this bit when the Rx2 ADC/HB2 upper threshold overflow counter is exceeded.	0x0
D1	Rx2: gain update counter expired. This bit signals when the gain update counter expires.	0x0
D0	Rx2: gain change. The gain index (in AGC and MGC mode) is used to select an entry in the gain table. This signal is active (the output level toggles between high and low) only if there is a difference in entries in the gain table when the AGC or MGC index changes. If the entries of the two indices are identical, the Rx Gain change signal is not generated (toggled).	0x0

Table 151. Monitor Index: 0x08 (Name: AGC Monitor 8)

Bits	Description	Reset
D7	Rx1: decimated power ready. The received signal strength indicator (RSSI) power word is ready.	0x0
D6	Rx1: gain update counter expired. Signals when the gain update counter expires.	0x0
D5	Rx1 APD upper threshold counter exceeded. The Rx1 APD is set when the upper threshold counter is exceeded. This bit operates real-time.	0x0
D4	Rx1 HB2 upper threshold counter exceeded. Set this bit when the Rx1 ADC/HB2 upper threshold overflow counter is exceeded.	0x0
D3	Rx2: decimated power ready. RSSI power word is ready.	0x0
D2	Rx2: gain update counter expired. This bit signals when the gain update counter expires.	0x0
D1	Rx2 APD upper threshold counter exceeded. The Rx2 APD is set when the upper threshold counter is exceeded. This bit operates real-time.	0x0
D0	Rx2 HB2 upper threshold counter exceeded. Set this bit when the Rx2 ADC/HB2 upper threshold overflow counter is exceeded.	0x0

Table 152. Monitor Index: 0x09 (Name: AGC Monitor 9)

Bits	Description	Reset
[D7:D0]	AGC—Rx1 gain index	0x0

Table 153. Monitor Index: 0x0A (Name: AGC Monitor 10)

Bits	Description	Reset
[D7:D0]	AGC—Rx2 gain index	0x0

Table 154. Monitor Index: 0x0C (Name: AGC Monitor 11)

Bits	Description	Reset
[D7:D4]	AGC—Rx1 gain index (4 LSBs only)	0x0
[D3:D0]	AGC—Rx2 gain index (4 LSBs only)	0x0

Table 155. Monitor Index: 0x22 (Name: Datapath Overflow)

Bits	Description	Reset
D7	Tx2 datapath—RFIR overflow error	0x0
D6	Tx2 datapath—HB2 overflow	0x0
D5	Tx2 datapath—HB1 overflow	0x0
D4	Tx2 datapath—TFIR overflow	0x0
D3	Tx1 datapath—RFIR overflow error	0x0
D2	Tx1 datapath—HB2 overflow	0x0
D1	Tx1 datapath—HB1 overflow	0x0
D0	Tx1 datapath—TFIR overflow	0x0

Table 156. Monitor Index: 0x23 (Name: RF DC Offset Correction Tracking Signals, Channel 1)

Bits	Description	Reset
D7	Reserved.	0x0
D6	Rx1: Signal of interest (SOI) present. The SOI is calculated in the decimated power block.	0x0
D5	Rx1: correction word above threshold. The calculated radio frequency (RF) DC offset word is above the threshold.	0x0
D4	Rx1: update counter expired. The RF DC update counter is expired.	0x0
D3	Rx1: gain change. The gain index (in automatic gain control (AGC) and manual gain control (MGC) is used to select an entry in the gain table. This signal is active (the output level toggles between high and low) only if there is a difference in entries in the gain table when the AGC or MGC index changes. If the entries of the two indices are identical, the Rx gain change signal is not generated (toggled).	0x0
D2	Rx1: update dc offset in the RF section. Updates the RF dc offset word.	0x0
D1	Rx1: measure dc offset in the RF section. Calibration and tracking is in measurement mode.	0x0
D0	Rx1: RF dc count reached. Calibration and tracking measurement counter expired.	0x0

Table 157. Monitor Index: 0x24 (Name: RF DC Offset Correction Tracking Signals, Channel 2)

Bits	Description	Reset
D7	Reserved.	0x0
D6	Rx2: SOI present. The SOI is calculated in the decimated power block.	0x0
D5	Rx2: correction word above threshold. The calculated RF dc offset word is above threshold.	0x0
D4	Rx2: update counter expired. The RF dc update counter is expired.	0x0
D3	Rx2: gain change. The gain index (in AGC and MGC mode) is used to select an entry in the gain table. This signal is active (the output level toggles between high and low) only if there is a difference in entries in the gain table when the AGC or MGC index changes. If the entries of the two indices are identical, the Rx gain change signal is not generated (toggled).	0x0
D2	Rx2: update dc offset in the RF section. Updates the RF dc offset word.	0x0
D1	Rx2: measure dc offset in the RF section. Calibration and tracking is in measurement mode.	0x0
D0	Rx2: RFDC count reached. Calibration/tracking measurement counter expired.	0x0

Table 158. Monitor Index: 0x42 (Name: AGC Monitor 12)

Bits	Description	Reset
D7	Rx2: overrange very low counter exceeded. This pin goes high when High-Band 2 (HB2) detects a number of peaks greater than the counter value (hb2VeryLowThreshExceededCnt—refer to Automatic Gain Control section for more details). These peaks must be greater than the hb2VeryLowThresh threshold to increment the count. If this signal is high at the end of the AGC gain update time counter, no gain decrement is made. This signal resets at the expiration of the AGC gain update counter.	0x0
D6	Rx2: overrange low counter exceeded. This pin goes high when HB2 detects a number of peaks greater than the counter value (hb2LowThreshExceededCnt—refer to Automatic Gain Control section for more details). These peaks must be greater than the hb2LowThresh threshold to increment the count. If this signal is high at the end of the AGC gain update time counter, no gain decrement is made. This signal resets at the expiration of the AGC gain update counter.	0x0
D5	Rx2: overrange high counter exceeded. This pin goes high when HB2 detects a number of peaks greater than the counter value (hb2HighThreshExceededCnt—refer to Automatic Gain Control section for more details). These peaks must be greater than the hb2HighThresh threshold to increment the count. This signal going high initiates a gain decrement immediately if hb2FastAttack mode is enabled or at the expiration of the AGC gain update counter if hb2FastAttack mode is disabled. This signal resets at the expiration of the AGC gain update counter.	0x0
D4	Rx2: gain update counter expired. This bit signals when the gain update counter expires.	0x0

Bits	Description	Reset
D3	Rx1: overrange very low counter exceeded. This pin goes high when High-Band 2 (HB2) detects a number of peaks greater than the counter value (hb2VeryLowThreshExceededCnt—refer to Automatic Gain Control section for more details). These peaks must be greater than the hb2VeryLowThresh threshold to increment the count. If this is high at the end of the automatic gain control (AGC) gain update time counter, no gain decrement is made. This signal resets at the expiration of the AGC gain update counter.	0x0
D2	Rx1: overrange low counter exceeded. This pin goes high when HB2 detects a number of peaks greater than the counter value (hb2LowThreshExceededCnt —refer to Automatic Gain Control section for more details). These peaks must be greater than the hb2LowThresh threshold to increment the count. If this signal is high at the end of the AGC gain update time counter, no gain decrement is made. This signal resets at the expiration of the AGC gain update counter.	0x0
D1	Rx1: overrange high counter exceeded. This pin goes high when HB2 detects a number of peaks greater than the counter value (hb2HighThreshExceededCnt—refer to Automatic Gain Control section for more details). These peaks must be greater than the hb2HighThresh threshold to increment the count. This signal going high initiates a gain decrement immediately if hb2FastAttack mode is enabled or at the expiration of the AGC gain update counter if hb2FastAttack mode is disabled. This signal resets at the expiration of the AGC gain update counter.	0x0
D0	Rx1: gain update counter expired. This bit signals when the gain update counter expires.	0x0

Table 159. Monitor Index: 0x43 (Name: AGC Monitor 13)

Bits	Description	Reset
[7:4]	Reserved.	0x0
3	ORx: overrange very low counter exceeded. This pin goes high when HB2 detects a number of peaks greater than the counter value (hb2VeryLowThreshExceededCnt—refer to the Automatic Gain Control section for more details). These peaks must be greater than the hb2VeryLowThresh threshold to increment the count. If this is high at the end of the AGC gain update time counter, no gain decrement is made. This signal resets at the expiration of the AGC gain update counter.	0x0
2	ORx: overrange low counter exceeded. This pin goes high when HB2 detects a number of peaks greater than the counter value (hb2LowThreshExceededCnt—refer to the Automatic Gain Control section for more details). These peaks must be greater than the hb2LowThresh threshold to increment the count. If this signal is high at the end of the AGC gain update time counter, no gain decrement is made. This signal resets at the expiration of the AGC gain update counter.	0x0
1	ORx: overrange high counter exceeded. This pin goes high when HB2 detects a number of peaks greater than the counter value (hb2HighThreshExceededCnt —refer to the Automatic Gain Control section for more details). These peaks must be greater than the hb2HighThresh threshold to increment the count. This signal going high initiates a gain decrement immediately if hb2FastAttack mode is enabled or at the expiration of the AGC gain update counter if hb2FastAttack mode is disabled. This signal resets at the expiration of the AGC gain update counter.	0x0
0	ORx: gain update counter expired. Signals when the gain update counter has expired.	0x0

API Description

The application programming interface (API) package provides functions that allow users to operate GPIO in monitor output mode. Before the user can use this mode, the crosspoints and the input/output buffer must be configured. Use the MYKONOS_setupGpio() function to properly configure the crosspoints and input/output buffers. This function relies on correct configuration of the device → auxIo → gpio structure members.

For an example, see the following configuration:

- device → auxIo → gpio → gpioOe = 0xXXXFF, the first eight GPIOs (Bits[D7:D0]) have the output enabled
- device → auxIo → gpio → gpioSrcCtrl3_0 = GPIO_MONITOR_MODE
- device → auxIo → gpio → gpioSrcCtrl4_7 = GPIO_MONITOR_MODE

These commands enable monitor Output D3 to Output D0 on GPIO_3 through GPIO_0, and Output D7 to Output D4 on GPIO_7 through GPIO_4. Refer to the API Description section in the GPIO Operation section for more details. In general, use the transceiver evaluation software (TES) to generate correct settings for this structure. After the crosspoint and input/output buffers are configured, the user can start to operate GPIOs in monitor output mode. A list of API functions dedicated for monitor output configuration, with short description of their functionalities, is in the following section.

As described previously, the control output signals are mapped as a table. Use the API MYKONOS_setGpioMonitorOut() function to select a particular row in the monitor output in Table 143, as well as to enable particular bits from the selected row. The device holds low any pins that are not enabled.

MYKONOS_setGpioMonitorOut

```
mykonosGpioErr_t
   MYKONOS_setGpioMonitorOut(mykonosDevice_t
   *device, uint8_t monitorIndex, uint8_t
   monitorMask)
```

This API function configures the monitor output function for the GPIOs. The monitor outputs are grouped in sets of nibbles (4 bits). The user can set individual nibbles for having the monitor output function across the available GPIO. Use the TES to generate correct settings for this structure.

Parameters

- monitorIndex: This parameter selects a row from Table 143, which in turn selects the desired monitor output signal assignments for the GPIO outputs.
- monitorMask: This parameter enables or disables active monitor output bits in the selected word. Setting Bit 0 of that word enables the D0 monitor output, setting Bit 1 of that word enables the D1 monitor output, and so on. The device holds low any pins that are not enabled.

MYKONOS_getGpioMonitorOut

```
mykonosGpioErr_t
   MYKONOS_getGpioMonitorOut(mykonosDevice_t
   *device, uint8_t *monitorIndex, uint8_t
   *monitorMask)
```

This API function reads back the current monitor index and monitor mask used to control monitor outputs.

Parameters

- *monitorIndex: Pointer to a variable storing the current monitor index to which outputs are set.
- *monitorMask: Pointer to a variable storing current monitor output bits in the word selected by *monitorIndex. Bit 0 of that word outlines the status of the D0 monitor output, Bit 1 of that word outlines status of the D1 monitor output, and so on. If a bit is set to 1, then that monitor output signal is enabled. If a bit is set to 0, then that monitor output signal is disabled.

ARM GPIO INTERFACE

The ARM microcontroller in the device can communicate with external devices using the GPIO interface. Various ARM input and output signals can be configured to map to specific GPIO pins. The specific pin location for ARM microcontroller GPIO functionalities can be configured using API commands.

Several external (GPIO) pins are used for controlling timing critical functionality provided by the ARM. When planning the routing of ARM microcontroller signals to the GPIO interface, the user must adhere to some routing limitations. Figure 97 and Table 160 outline the restrictions that apply to configuring the GPIOs as an ARM interface.

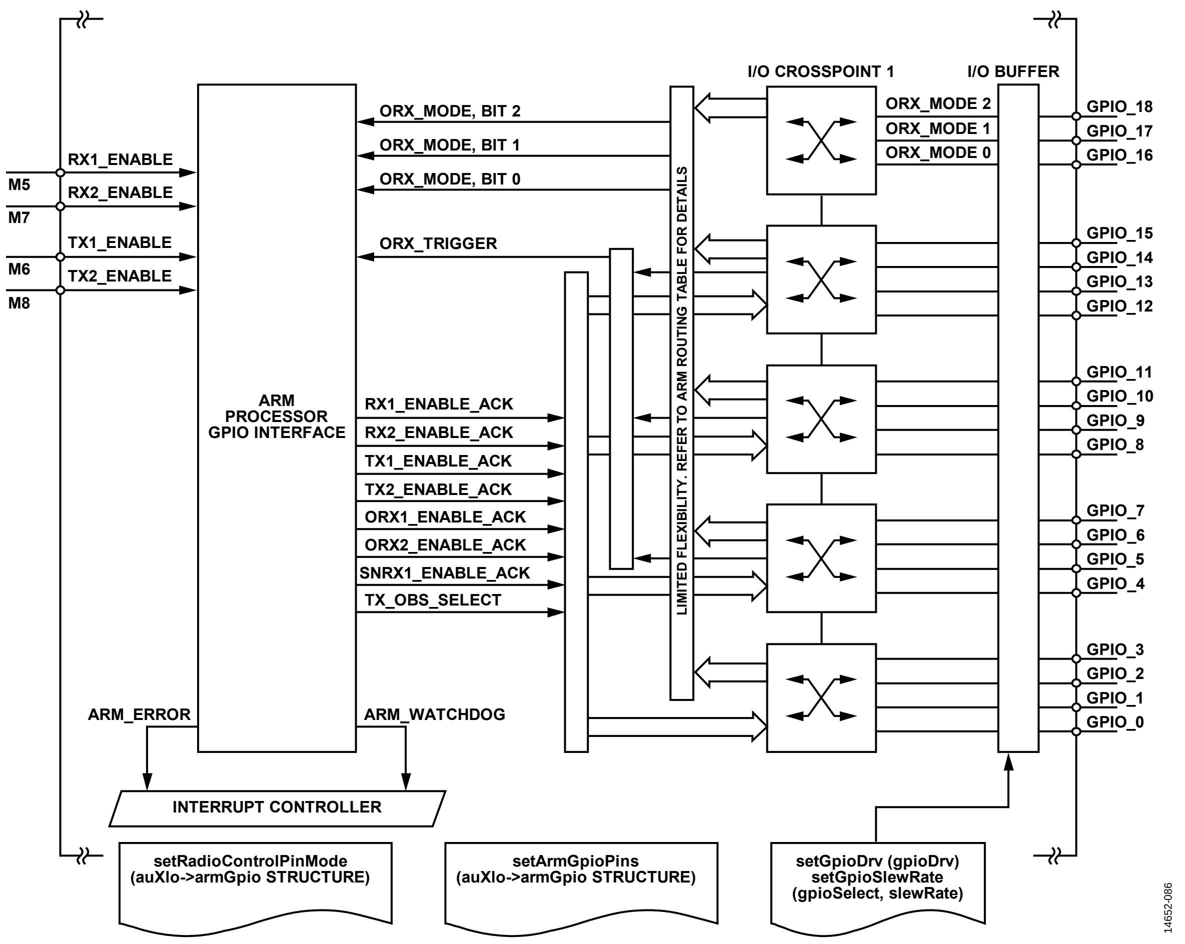


Figure 97. ARM Microcontroller to GPIO Interface

Table 160. GPIO Routing Options for ARM Signals

Signal Name	Description	Possible GPIO pin
RX1_ENABLE	This pin can be configured to enable/disable Rx1 by itself, or to enable/disable both Rx1 and Rx2 simultaneously.	RX1_ENABLE pin
RX2_ENABLE	Optionally, this pin can be configured to enable/disable Rx2.	RX2_ENABLE pin
TX1_ENABLE	This pin can be configured to enable/disable Tx1 by itself, or to enable/disable both Tx1 and Tx2 simultaneously.	TX1_ENABLE pin
TX2_ENABLE	Optionally, this pin can be configured to enable/disable Tx2.	TX2_ENABLE pin
ORX_TRIGGER	A rising edge on this signal triggers a change in the configuration of the ORx receiver. On the rising edge, the ORX_MODE[2:0] pins are sampled to determine the target ORx configuration.	Any pin from GPIO_15 to GPIO_4
ORX_MODE[2:0]	Selects the ORx mode. Sampled on rising edge of ORX_TRIGGER. 000 = ORx powered off. 001 = ORx1 (with Tx local oscillator (LO)). 010 = ORx2 (with Tx LO). 011 = no baseband processor (BBP) access. ORx available to internal Tx calibrations. 100 = sniffer Rx. 101 = ORx1 (with sniffer LO). 110 = ORx2 (with sniffer LO). 111 = reserved (trigger ignored). Note that all three ORX_MODE[2:0] pins must come from the same bank of GPIOs. Banks are defined as GPIO_3 to GPIO_0, or GPIO_15 to GPIO_4, or GPIO_18 to GPIO_16. When GPIO_18 to GPIO_16 are used, the only valid assignments are ORX_MODE[0] on GPIO_16, ORX_MODE[1] on GPIO_17, and ORX_MODE[2] on GPIO_18.	Any three pins from GPIO_3 to GPIO_0 or any three pins from GPIO_15 to GPIO_4; or, ORX_MODE[0] on GPIO_16, ORX_MODE[1] on GPIO_17, or ORX_MODE[2] on GPIO_18.

Signal Name	Description	Possible GPIO pin
RX1_ENABLE_ACK	Indicates that Rx1 is (or both Rx1 and Rx2 are) enabled.	Any pin from GPIO_15 to GPIO_0
RX2_ENABLE_ACK	Indicates that Rx2 is enabled.	Any pin from GPIO_15 to GPIO_0
TX1_ENABLE_ACK	Indicates that Tx1 is (or both Tx1 and Tx2 are) enabled.	Any pin from GPIO_15 to GPIO_0
TX2_ENABLE_ACK	Indicates that Tx2 is enabled.	Any pin from GPIO_15 to GPIO_0
ORX1_ENABLE_ACK	Indicates that ORx1 is enabled for BBP use.	Any pin from GPIO_15 to GPIO_0
ORX2_ENABLE_ACK	Indicates that ORx2 is enabled for BBP use.	Any pin from GPIO_15 to GPIO_0
SNRX_ENABLE_ACK	Indicates that SnRx is enabled for BBP use.	Any pin from GPIO_15 to GPIO_0
TX_OBS_SELECT	When two transmitters are used with only one ORx input, this GPIO tells the BBP which transmitter channel is active for calibrations. BBP controls an RF switch that routes the desired RF Tx path into the single ORx input.	Any pin from GPIO_15 to GPIO_0
ARM_ERROR	Indicates that the ARM must be rebooted because of some error. The ARM was able to detect a problem and record some diagnostic information before setting this flag.	GP_INTERRUPT pin
ARM_WATCHDOG	Indicates that the watchdog has expired because of some error that the ARM was unable to respond to (for example, the ARM was unable to set the ARM_ERROR flag). The ARM must be rebooted.	GP_INTERRUPT pin

API Description

The application programming interface (API) package provides functions that allow the user to operate the GPIO in ARM GPIO interface mode. The ARM GPIO input and output signals that are desired for Rx/Tx control in pin mode must be set up before the device enters radio on mode.

The ARM GPIO interface configurations rely on correct values set to the device → auxIo → armGpio structure members. A quick overview of this structure is as follows (use the transceiver evaluation software (TES) to generate correct settings for this structure):

```
typedef struct
{
    uint8_t useRx2EnablePin;         /*!< 0= RX1_ENABLE controls RX1 and RX2, 1 = separate
                                     RX1_ENABLE/RX2_ENABLE pins */
    uint8_t useTx2EnablePin;         /*!< 0= TX1_ENABLE controls TX1 and TX2, 1 = separate
                                     TX1_ENABLE/TX2_ENABLE pins */
    uint8_t txRxPinMode;             /*!< 0= ARM command mode, 1 = Pin mode to power up Tx/Rx
                                     chains */
    uint8_t orxPinMode;              /*!< 0= ARM command mode, 1 = Pin mode to power up ObsRx
                                     receiver*/

    /* the AD9371 ARM input GPIO pins -- Only valid if orxPinMode = 1 */
    uint8_t orxTriggerPin;       /*!< Select desired GPIO pin (valid 4-15) */
    uint8_t orxMode2Pin;         /*!< Select desired GPIO pin (valid 0-18  - limited combin.) */
    uint8_t orxMode1Pin;         /*!< Select desired GPIO pin (valid 0-18) - limited combin.) */
    uint8_t orxMode0Pin;         /*!< Select desired GPIO pin (valid 0-18) - limited combin.) */

    /* the AD9371 ARM output GPIO pins  --  always available, even when pin mode not enabled*/
    uint8_t rx1EnableAck;        /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    uint8_t rx2EnableAck;        /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    uint8_t tx1EnableAck;        /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    uint8_t tx2EnableAck;        /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    uint8_t orx1EnableAck;       /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    uint8_t orx2EnableAck;       /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    uint8_t srxEnableAck;        /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
    uint8_t txObsSelect;         /*!< Select desired GPIO pin (0-15), [4] = Output Enable */
                                 /* When 2Tx are used with only 1 ORx input, this GPIO tells the
                                 BBP which Tx channel is active for calibrations, so BBP can
                                 route correct RF Tx path into the single ORx input */
} mykonosArmGpioConfig_t;
```

The previously described structure controls all sections of the ARM GPIO interface hardware shown in Figure 97, including the input/output (I/O) buffer, I/O Crosspoint 1, and the ARM processor. The previous description outlines the structure members values and their interpretations. For the orxMode2Pin, orxMode1Pin, and orxMode0Pin structure members, refer to Table 160 for limitations in possible routing combinations. Note that the ORX_MODE and ORX_TRIGGER signals are ignored if ORx control orxPinMode is set to command mode, even if these signals are mapped to GPIO pins.

MYKONOS_setArmGpioPins

```
mykonosGpioErr_t
    MYKONOS_setArmGpioPins(mykonosDevice_t
    *device)
```

This function sets the input and output GPIO pin selections for ARM related signals. The baseband processor (BBP) does not have to call this function because it is automatically set up during the MYKONOS_loadArmFromBinary() function call. If the BBP wishes to change the GPIO assignments, this function can be called again to change the configuration while the ARM is in the radio off state. This function relies on correct settings in the armGpio structure. Use the transceiver evaluation software (TES) to generate correct settings for this structure.

MYKONOS_setRadioControlPinMode

```
mykonosGpioErr_t
    MYKONOS_setRadioControlPinMode(mykonosDev
    ice_t *device)
```

This function configures the radio power-up/power-down control for the Rx and Tx paths to be controlled by pins (TX1_ENABLE, TX2_ENABLE, RX1_ENABLE, RX2_ENABLE, and the GPIO pins) or an application programming interface (API) function call. The GPIO setup and configuration can be performed in both the ready and idle state; it can be performed as many times as desired. The BBP does not have to call this function because it is automatically set up at the end of the MYKONOS_loadArm-FromBinary() function call. If the BBP wishes to change the GPIO assignments, this function can be recalled to change the configuration while the ARM is in the radio off state.

This function relies on correct settings in the armGpio structure. Use the TES to generate correct settings for this structure.

Tx ATTENUATION CONTROL

The device uses an accurate and efficient method of transmit power control (Tx attenuation control) that involves minimum interaction with the BBP. The power control in the transmit chain is implemented with two variable attenuations: one in the digital domain and one in the analog domain. The digital attenuator is programmable from 0 dB to −6 dB in steps of 0.05 dB. The analog RF attenuator is programmable from 0 dB to −36.12 dB. There are 64 possible settings for the analog attenuator. An internal table is provided within the device that adjusts both analog and digital attenuators simultaneously. Each row in this table provides a unique combination of analog and digital gain. The table is arranged in increasing attenuation, with Row 0 being the lowest attenuation setting (0 dB) and Row 839 being the largest attenuation setting (41.95 dB). A consistent attenuation step size of 0.05 dB is maintained between each consecutive row of the table.

Figure 98 shows the location of the analog and digital attenuation blocks within the Tx chain, as well as the GPIO interface to control it. The attenuation table is controlled by a pointer. By moving the pointer to the required row of the table, the corresponding analog and digital attenuation settings of this row are applied.

The pointer to the attenuation table can be controlled through the GPIO pins. In this mode, four GPIO pins control the Tx attenuation values for Tx1 and Tx2: two pins for Tx1 (one to increase pointer index, one to decrease), and two pins for Tx2. There is also an option to use just two GPIO pins to control Tx attenuation for both Tx1 and Tx2 at the same time. Minimum lengths of pulse are present at the GPIO input to be latched is 2 clock radio frequency (RF) cycles. The clock RF frequency can be found in the **Rx Summary** tab of the TES.

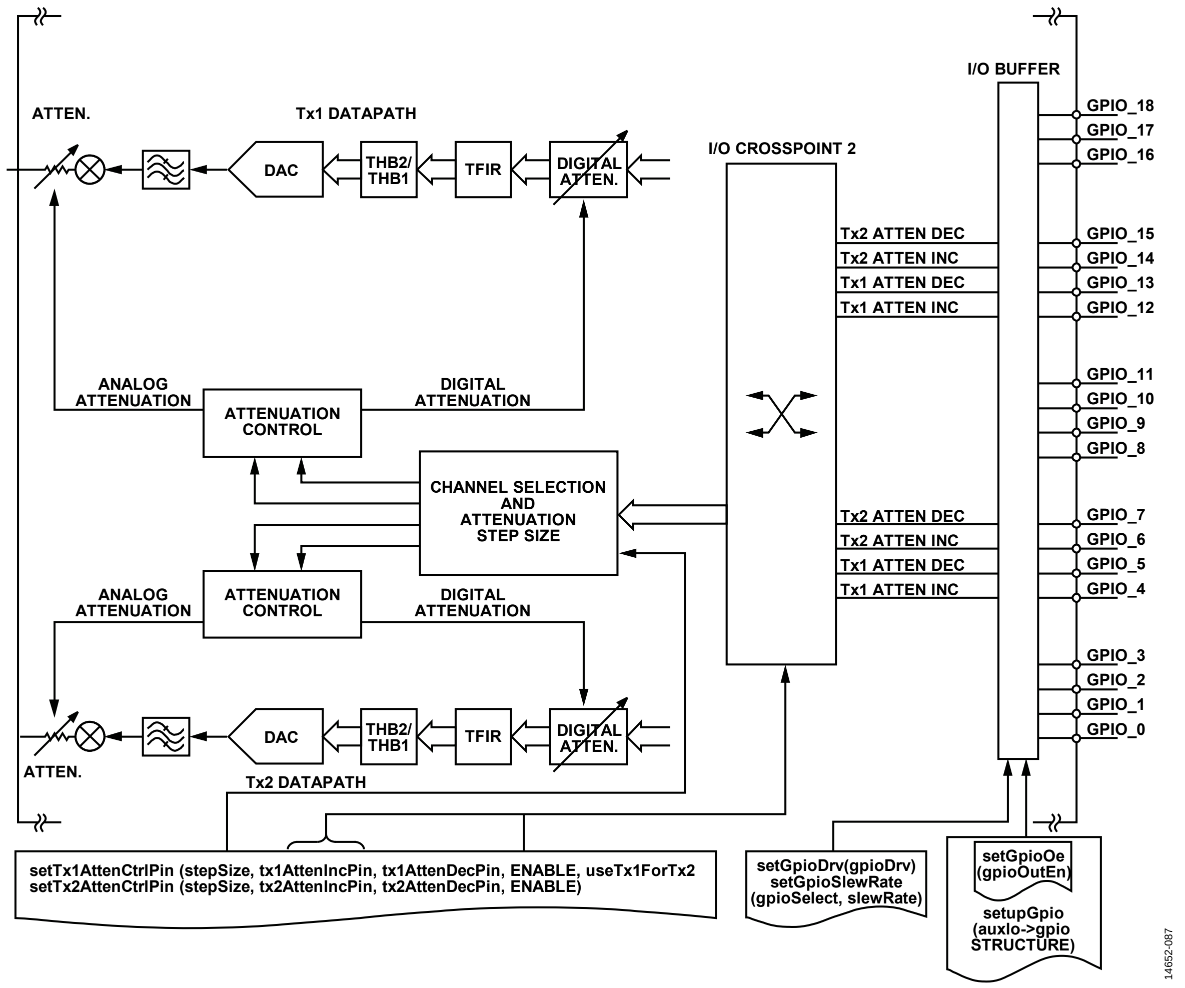


Figure 98. GPIO Interface for Tx Attenuation Control

API Description

The application programming interface (API) package provides functions that allow users to operate the GPIO in Tx attenuation control mode. Before users can use the device in Tx attenuation control mode, Crosspoint 2 and the input/output (I/O) buffer must be configured. Use the MYKONOS_setupGpio() function to properly configure the crosspoint and I/O buffers. This function relies on correct configuration of device → auxIo → gpio structure members. Use the transceiver evaluation software (TES) to generate correct settings for this structure. After a crosspoint and I/O buffer are configured, the user can start to operate GPIOs in Tx attenuation control mode. The following sections list Tx attenuation control mode API functions and short descriptions of their functions.

MYKONOS_setTx1AttenCtrlPin

```
mykonosGpioErr_t
    MYKONOS_setTx1AttenCtrlPin(mykonosDevice_
    t *device, uint8_t stepSize,
    mykonosGpioSelect_t tx1AttenIncPin,
    mykonosGpioSelect_t tx1AttenDecPin,
    uint8_t enable, uint8_t useTx1ForTx2)
```

This API function allows the user to control the Tx1 attenuation using GPIO inputs. When a low to high transition is applied to configure the GPIO input, the attenuation changes by the desired step.

Parameters

- stepSize: This parameter is the step that increases or decreases the Tx1 channel attenuation. This parameter sets the change in Tx attenuation for each increment or decrement signal received in increment/decrement mode. A step of 1 changes attenuation by 0.05 dB.
- tx1AttenIncPin: This parameter is the GPIO pin configuration that controls the increment of Tx1 attenuation. The available pins are MYKGPIO4 and MYKGPIO12.
- tx1AttenDecPin: This parameter is the GPIO pin configuration that controls the decrement of Tx1 attenuation. The available pins are MYKGPIO5 and MYKGPIO13.

- enable: This parameter enables or disables Tx attenuation pin control mode for Tx1.
  - 0 = disables the attenuation pin control for Tx1.
  - 1 = enables the attenuation pin control for Tx1.
- useTx1ForTx2: This parameter enables use of Tx1 GPIOs to control attenuation for both the Tx1 and Tx2 channels.
  - 0 = disables use of Tx1 GPIOs to control attenuation for both the Tx1 and Tx2 channels.
  - 1 = enables use of Tx1 GPIOs to control attenuation for both the Tx1 and Tx2 channels.

MYKONOS_setTx2AttenCtrlPin

```
mykonosGpioErr_t
   MYKONOS_setTx2AttenCtrlPin(mykonosDevice_
   t *device, uint8_t stepSize,
   mykonosGpioSelect_t tx2AttenIncPin,
   mykonosGpioSelect_t tx2AttenDecPin,
   uint8_t enable)
```

This application programming interface (API) function allows the user to control the Tx2 attenuation using GPIO inputs. When a low to high transition is applied to configure the GPIO input, the attenuation changes by the desired step.

Parameters

- stepSize: This parameter is the step that increases or decreases the Tx2 channel attenuation. This parameter sets the change in Tx attenuation for each increment or decrement signal received in increment/decrement mode. A step of 1 changes attenuation by 0.05 dB.
- tx2AttenIncPin: This parameter is the GPIO pin configuration that controls the increment of Tx2 attenuation. The available pins are MYKGPIO6 and MYKGPIO14.
- tx2AttenDecPin: This parameter is the GPIO pin configuration that controls the decrement of Tx2 attenuation. The available pins are MYKGPIO7 and MYKGPIO15.
- enable: This parameter enables or disables Tx attenuation pin control mode for Tx2.
  - 0 = disables the attenuation pin control for Tx2.
  - 1 = enables the attenuation pin control for Tx2.

MYKONOS_getTx1AttenCtrlPin

```
mykonosGpioErr_t
   MYKONOS_getTx1AttenCtrlPin(mykonosDevice_
   t *device, uint8_t *stepSize,
   mykonosGpioSelect_t *tx1AttenIncPin,
   mykonosGpioSelect_t *tx1AttenDecPin,
   uint8_t *enable, uint8_t *useTx1ForTx2)
```

This API function returns the current configuration of Tx1 attenuation in pin control mode.

Parameters

- *stepSize: This is a pointer to the variable that contains the step used for increment and decrement of Tx1 attenuation.
- *tx1AttenIncPin: This is a pointer to the variable that stores information about the pin used for Tx1 attenuation increment.
- *tx1AttenDecPin: This is a pointer to the variable that stores information about the pin used for Tx1 attenuation decrement.
- *enable: This is a pointer to the variable that contains the enable status for this channel. If it is set to 1, then this function is enabled for this channel, and if it is 0, it is not enabled.
- *useTx1ForTx2. This is a pointer to the variable that contains the parameter indicating if Tx1 settings are used to control attenuation in the Tx2 channel.

MYKONOS_getTx2AttenCtrlPin

```
mykonosGpioErr_t
   MYKONOS_getTx2AttenCtrlPin(mykonosDevice_
   t *device, uint8_t *stepSize,
   mykonosGpioSelect_t *tx2AttenIncPin,
   mykonosGpioSelect_t *tx2AttenDecPin,
   uint8_t *enable, uint8_t *useTx1ForTx2)
```

This API function returns the current configuration of Tx2 attenuation in pin control mode.

Parameters

- *stepSize: This is a pointer to the variable that contains the step used for increment and decrement of Tx2 attenuation.
- *tx2AttenIncPin: This is a pointer to the variable that stores information about the pin used for Tx2 attenuation increment.
- *tx2AttenDecPin: This is a pointer to the variable that stores information about the pin used for Tx2 attenuation decrement.
- *enable: This is a pointer to the variable that contains the enable status for this channel. If it is set to 1, then this function is enabled for this channel, and if it is 0, it is not enabled.
- *useTx1ForTx2: This is a pointer to the variable that contains parameter indicating if Tx1 settings are used to control attenuation in the Tx2 channel.

SECONDARY SERIAL PERIPHERAL INTERFACE (SPI2)

The transmitter (Tx) features an optional dedicated SPI, or SPI2, interface that can adjust Tx attenuation. The SPI2 interface is included exclusively for the control of Tx attenuation on both channels. It can only be used to write to the 8 registers mentioned in this section; it does not offer write access to any other registers.

The SPI2 interface allows users to program two distinct attenuation states (S1 and S2) per transmitter and use an additional pin to toggle between these states. This interface can be particularly useful for solutions that need to adjust the Tx attenuation at a precise instance in time but need a wider attenuation range than what is available over the GPIO based Tx attenuation increment/decrement control. Note that when SPI2 is enabled, the primary SPI port cannot be used to adjust Tx attenuation.

A block diagram depicting the use of the SPI2 interface is shown in Figure 99.

Figure 100 shows another way to look at the use case where the Tx1 and Tx2 attenuation are changed to accommodate a special slot in time.

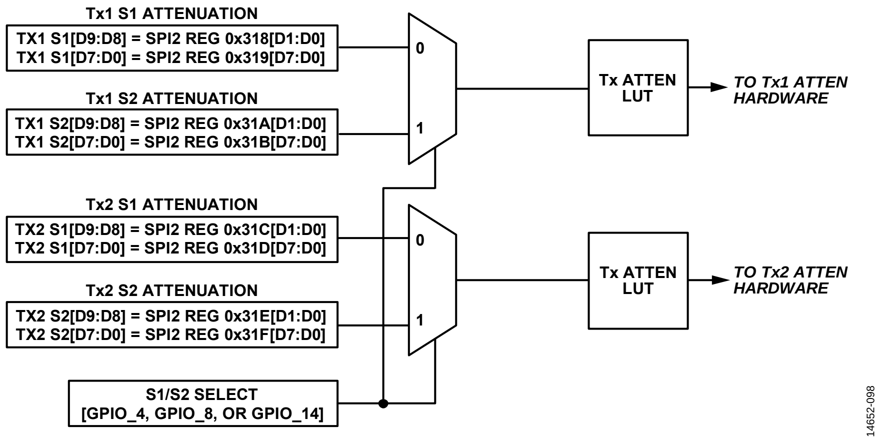

Tx1 S1 ATTENUATION
TX1 S1[D9:D8] = SPI2 REG 0x318[D1:D0]
TX1 S1[D7:D0] = SPI2 REG 0x319[D7:D0]
Tx1 S2 ATTENUATION
TX1 S2[D9:D8] = SPI2 REG 0x31A[D1:D0]
TX1 S2[D7:D0] = SPI2 REG 0x31B[D7:D0]
Tx2 S1 ATTENUATION
TX2 S1[D9:D8] = SPI2 REG 0x31C[D1:D0]
TX2 S1[D7:D0] = SPI2 REG 0x31D[D7:D0]
Tx2 S2 ATTENUATION
TX2 S2[D9:D8] = SPI2 REG 0x31E[D1:D0]
TX2 S2[D7:D0] = SPI2 REG 0x31F[D7:D0]
S1/S2 SELECT [GPIO_4, GPIO_8, OR GPIO_14]
0
1
0
1
Tx ATTEN LUT
TO Tx1 ATTEN HARDWARE
Tx ATTEN LUT
TO Tx2 ATTEN HARDWARE
14652-098

Figure 99. Block Diagram of SPI2 Operation

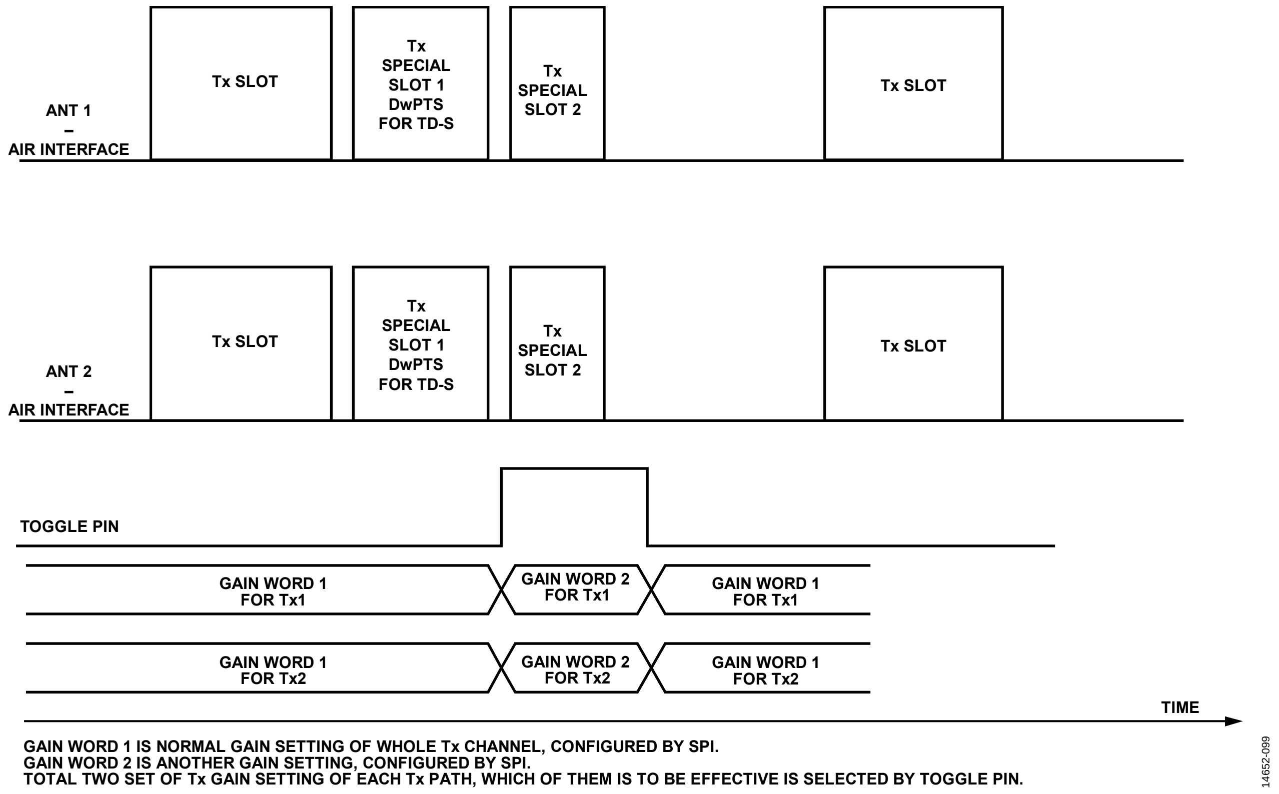

ANT 1 – AIR INTERFACE
Tx SLOT
Tx SPECIAL SLOT 1 DwPTS FOR TD-S
Tx SPECIAL SLOT 2
Tx SLOT
ANT 2 – AIR INTERFACE
Tx SLOT
Tx SPECIAL SLOT 1 DwPTS FOR TD-S
Tx SPECIAL SLOT 2
Tx SLOT
TOGGLE PIN
GAIN WORD 1 FOR Tx1
GAIN WORD 2 FOR Tx1
GAIN WORD 1 FOR Tx1
GAIN WORD 1 FOR Tx2
GAIN WORD 2 FOR Tx2
GAIN WORD 1 FOR Tx2
TIME
GAIN WORD 1 IS NORMAL GAIN SETTING OF WHOLE Tx CHANNEL, CONFIGURED BY SPI.
GAIN WORD 2 IS ANOTHER GAIN SETTING, CONFIGURED BY SPI.
TOTAL TWO SET OF Tx GAIN SETTING OF EACH Tx PATH, WHICH OF THEM IS TO BE EFFECTIVE IS SELECTED BY TOGGLE PIN.
14652-099

Figure 100. Timing Diagram of SPI2 Operation

The SPI2 interface uses the GPIO_3 to GPIO_0 pins for the SPI control pins and an additional pin that toggle between Tx attenuation states, S1 and S2. The additional pin, S1/S2 select, can be designated as one of the following pins: GPIO_4, GPIO_8, and GPIO_14. The pin mapping is described in Table 161.

The SPI2 port uses the same SPI protocol as the primary SPI port in terms of MSB/LSB first and descending/ascending order. Note that the GPIO output enables must be set correctly for SPI2 to work properly.

Table 161. GPIO Pin Mappings for SPI2

GPIO Pin	I/O	Function	Description
GPIO_0	Input	SDIO2	SPI2 port data input
GPIO_1	Output	SDO2	SPI2 port data output
GPIO_2	Input	SCLK2	SPI2 port data clock
GPIO_3	Input	CSB2	SPI2 port chip select bar (negative logic)
GPIO_4/GPIO_8/ GPIO_14	Input	S1/S2 Select	Selection of Tx attenuation word (0 = S1, 1 = S2)

SPI2 Register Map

The SPI2 interface is restricted to 8 addresses. These addresses cannot be accessed via the main SPI interface. The SPI2 registers correspond to 10-bit words for Tx1 attenuation S1, Tx1 attenuation S2, Tx2 attenuation S1, Tx2 attenuation S2 as detailed in the SPI2 register map. Because the Tx attenuation is set as a 10-bit word, it takes two SPI writes to write a single Tx attenuation word.

Note that the LSB in the 10-bit words are programmable. The step size can be set to 0.05dB (default), 0.1dB, 0.2dB, or 0.4dB per LSB. Refer to the parameter in the device data structure device → tx → txAttenStepSize. The txAttenStepSize is written over the primary SPI interface during MYKONOS_initialize(…).

The attenuation readback registers (0x2E8 to 0x2EB) can be read over the SPI port or the SPI2 port. These allow the user to read the currently applied Tx attenuation settings. The SPI2 register map is shown in Table 162.

API Description

The SPI2 is set up with the application programming interface (API) command as described as follows.

MYKONOS_spi2GpioSetup

```
mykonosGpioErr_t
   MYKONOS_spi2GpioSetup(mykonosDevice_t
   *device, uint8_t enable, uint8_t
   updateTxAttenPinSelect)
```

This API function configures and enables the secondary SPI port. This port allows control compatibility with baseband processors (BBPs) that employ dual SPI ports. The GPIO mapping for SPI2 is fixed, excluding a configurable select pin that selects Tx attenuation between Attenuation State 1 (S1) and Attenuation State 2 (S2).

Parameters

- enable: This is the parameter that enables the secondary SPI port. 1 = enable, and 0 = disable.
- updateTxAttenPinSelect: This parameter sets the GPIO pin to be toggled for determining the S1 or S2 state. Configuration options include the following:
    - GPIO_4 → updateTxAttenPinSelect = 0x00
    - GPIO_8 → updateTxAttenPinSelect = 0x01
    - GPIO_14 → updateTxAttenPinSelect = 0x02

Table 162. SPI2 Register Map

Register Address	Name	D7	D6	D5	D4	D3	D2	D1	D0	Default	R/W	I/O Scope
0x2E8	Tx1 atten index readback LSB	Tx1 attenuation index readback[7:0]								0x00	R	Digital
0x2E9	Tx1 atten index readback MSB	Unused						Tx1 attenuation index readback[9:8]		0x00	R	Digital
0x2EA	Tx2 atten index readback LSB	Tx2 attenuation index readback[7:0]								0x00	R	Digital
0x2EB	Tx2 atten index readback MSB	Unused						Tx2 attenuation index readback[9:8]		0x00	R	Digital
0x318	Tx1 attenuation S1 MSB	Unused						Tx1 attenuation S1[9:8]		0x00	R/W	Digital
0x319	Tx1 attenuation S1 LSB	Tx1 attenuation S1[7:0] (update happens when this byte is written, write this byte after S1[9:8] to see update)								0x00	R/W	Digital
0x31A	Tx1 attenuation S2 MSB	Unused						Tx1 attenuation S2[9:8]		0x00	R/W	Digital
0x31B	Tx1 attenuation S2 LSB	Tx1 attenuation S2[7:0] (update happens when this byte is written, write this byte after S2[9:8] to see update)								0x00	R/W	Digital
0x31C	Tx2 attenuation S1 MSB	Unused						Tx2 attenuation S1[9:8]		0x00	R/W	Digital
0x31D	Tx2 attenuation S1 LSB	Tx2 attenuation S1[7:0] (update happens when this byte is written, write this byte after S1[9:8] to see update)								0x00	R/W	Digital
0x31E	Tx2 attenuation S2 MSB	Unused						Tx2 attenuation S2[9:8]		0x00	R/W	Digital
0x31F	Tx2 attenuation S2 LSB	Tx2 attenuation S2[7:0] (update happens when this byte is written, write this byte after S2[9:8] to see update)								0x00	R/W	Digital

Rx MANUAL GAIN CONTROL

The device provides gain control blocks that are externally controllable through GPIO pins, allowing the user complete control over the applied attenuation. This section describes the GPIO configuration for and the operation of manual gain control (MGC). Refer to the Gain Control section for more information about the gain control block.

Figure 101 shows a block diagram of the receiver paths together with the gain control blocks and the GPIO interface connection. There are two variable gain elements in the receive path: the internal RF attenuator and the digital gain/attenuator block. The MGC blocks control the gains of both these components simultaneously; the Gain Control block outputs in Figure 101 indicate this control.

Note that this device has two receiver chains. Each receiver has its own gain table that simultaneously controls each of the variable gain blocks in Figure 101. Each row of this table has a unique combination of gain settings. A pointer to the table determines which settings from this table are used. The internal radio frequency (RF) attenuator has 64 different attenuation settings, or indices (0 to 63), which range from 0 dB to 36.12 dB of attenuation. The device also has a digital gain block that can provide higher resolution than provided by the internal RF attenuator. The digital gain has 128 indices (0 to 127) that corresponds to a gain range of 0 dB to 31.75 dB in 0.25 dB steps. The device also has functionality allowing the user to control a digital stepped attenuator with GPIO_3P3_x pins. There are up to four bits (on GPIO_3P3 interface) available.

In MGC mode, the baseband processor (BBP) controls the gain index pointer(s), which is the pointer used to select the required row of the gain table. In MGC mode, the gain index pointer can be controlled either by using the SPI interface or by using the GPIO interface. The GPIO interface method is implemented by toggling the GPIO pins to initiate gain changes according to the following process. The gain control GPIO pins are driven high. A transition from a logic low to a logic high, held high for at least 2 clock RF cycles, initiates a gain change in the device (clock RF is the clock at the input to the Rx finite impulse response (FIR), refer to transceiver evaluation software (TES) for more information). Similarly, a logic low must be maintained for at least 2 clock RF cycles. The number of gain indices that an increase or decrease corresponds to is user programmable.

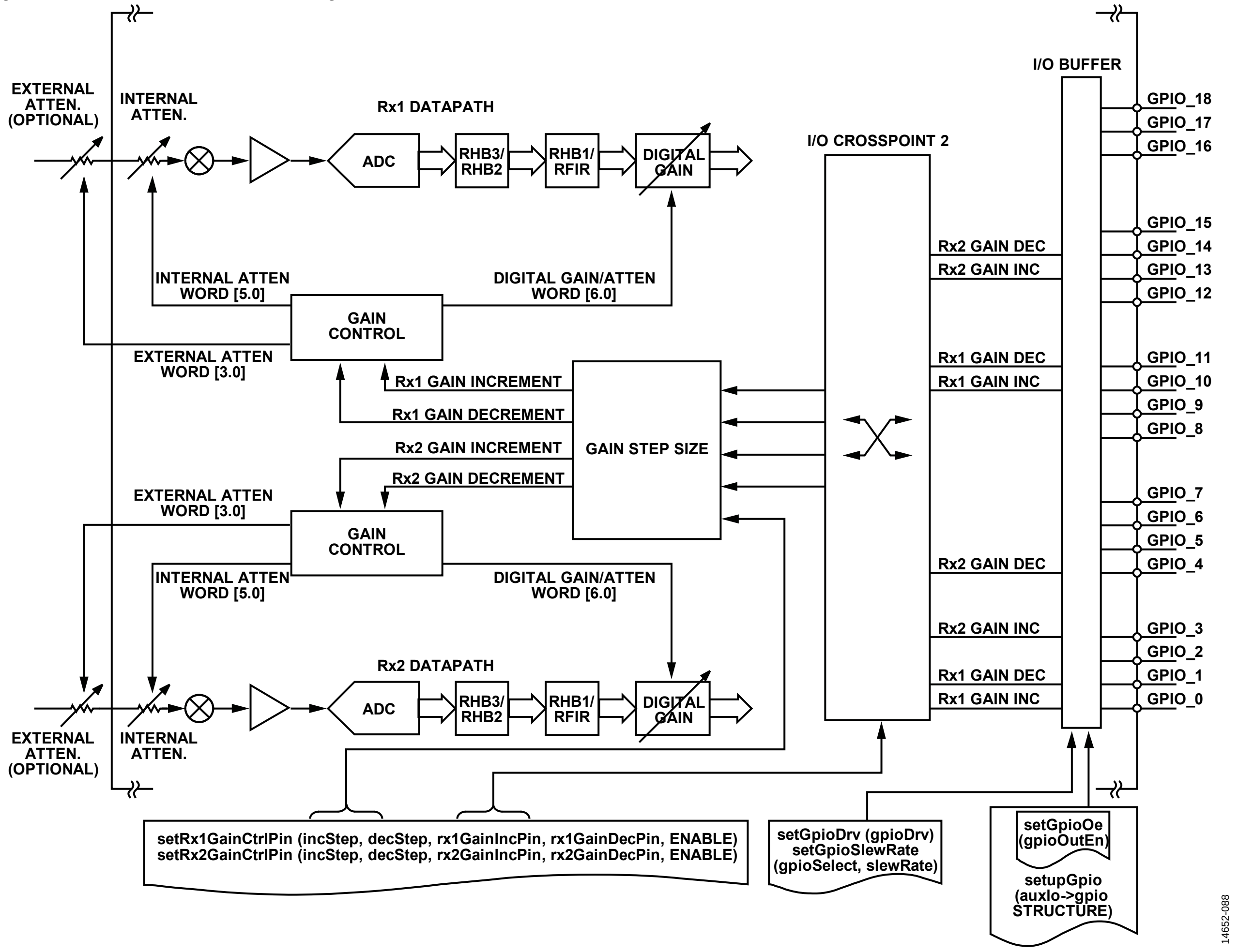


Figure 101. GPIO Configuration for Manual Gain Control Mode

API Description

The application programming interface (API) package provides functions that allow users to operate the GPIO in Rx manual gain control mode. Before the user can use the device in Rx manual gain control mode, Crosspoint 2 and the input/output (I/O) buffers must be configured. Use the MYKONOS_setupGpio() function to properly configure the crosspoint and I/O buffers. This function relies on correct configuration of device → auxIo → gpio structure members. Use the transceiver evaluation software (TES) to generate correct settings for this structure. After the crosspoint and I/O buffers are configured, the user can start to operate GPIOs in Rx manual gain control mode. The following sections list the Rx manual gain control mode API functions and short descriptions of their functionalities.

MYKONOS_setRx1GainCtrlPin

```
mykonosGpioErr_t
   MYKONOS_setRx1GainCtrlPin(mykonosDevice_t
   *device, uint8_t incStep, uint8_t
   decStep, mykonosGpioSelect_t
   rx1GainIncPin, mykonosGpioSelect_t
   rx1GainDecPin, uint8_t enable)
```

This API function configures the GPIO input pin and step size to allow the baseband processor (BBP) to control gain changes in Rx1 signal chain. A high pulse on the GPIO pin set by rx1GainIncPin increments the gain by the value set in incStep. A high pulse on the GPIO pin set by rx1GainDecPin decrements the gain by the value set in decStep.

Parameters

- incStep: This parameter sets the change (increase) in the gain index that is applied when the increment gain pin (in manual gain control (MGC) pin control mode) is pulsed.
- decStep: This parameter sets the change (decrement) in gain index to be applied when the decrement gain pin (in MGC pin control mode) is pulsed high and none of the peak detector signals trigger. If any combination of the peak detector signals are high, the gain step that corresponds to that combination of peak detector signals is used as the decrement step size.
- rx1GainIncPin: This parameter selects the GPIO used for the Rx1 manual gain increment input. The available pins are MYKGPIO0 and MYKGPIO10.
- rx1GainDecPin: This parameter selects the GPIO used for the Rx1 manual gain decrement input. The available pins are MYKGPIO1 and MYKGPIO11.
- enable: This parameter enables or disables manual gain control mode for Rx1.
  - 0 = disables the manual gain control pin mode for Rx1.
  - 1 = enables the manual gain control pin mode for Rx1.

MYKONOS_setRx2GainCtrlPin

```
mykonosGpioErr_t
   MYKONOS_setRx2GainCtrlPin(mykonosDevice_t
   *device, uint8_t incStep, uint8_t
   decStep, mykonosGpioSelect_t
   rx2GainIncPin, mykonosGpioSelect_t
   rx2GainDecPin, uint8_t enable)
```

This API function configures the GPIO input pin and step size to allow the BBP to control gain changes in the Rx2 signal chain. A high pulse on the GPIO pin set by rx2GainIncPin increments the gain by the value set in incStep. A high pulse on the GPIO pin set by rx2GainDecPin decrements the gain by the value set in decStep.

Parameters

- incStep: This parameter sets the change (increase) in gain index that is applied when the increment gain pin (in MGC pin control mode) is pulsed.
- decStep: This parameter sets the change (decrement) in gain index to be applied when the decrement gain pin (in MGC pin control mode) is pulsed high and none of the peak detector signals have triggered. If any combination of the peak detector signals are high, the gain step that corresponds to that combination of peak detector signals is used as the decrement step size.
- rx2GainIncPin: This parameter selects the GPIO used for the Rx2 manual gain increment input. The available pins are MYKGPIO3 and MYKGPIO13.
- rx2GainDecPin: This parameter selects the GPIO used for the Rx2 manual gain decrement input. The available pins are MYKGPIO4 and MYKGPIO14.
- enable: This parameter enables or disables manual gain control mode for Rx2.
  - 0 = disables the manual gain control pin mode for Rx2.
  - 1 = enables the manual gain control pin mode for Rx2.

MYKONOS_getRx1GainCtrlPin

```
mykonosGpioErr_t
   MYKONOS_getRx1GainCtrlPin(mykonosDevice_t
   *device, uint8_t *incStep, uint8_t
   *decStep, mykonosGpioSelect_t
   *rx1GainIncPin, mykonosGpioSelect_t
   *rx1GainDecPin, uint8_t *enable)
```

This API function returns the configuration for the GPIO inputs and step sizes used to control the gain index in MGC input pin control mode for the Rx1 signal chain.

Parameters

- *incStep: A pointer to a variable that contains the step used for gain increment.
- *decStep: A pointer to a variable that contains the step used for gain decrement.
- *rx1GainIncPin: A pointer to a variable that has the pin used for gain increment.
- *rx1GainDecPin: A pointer to a variable that has the pin used for gain decrement.

- *enable: A pointer to a variable that contains the enable status for the Rx1 channel.
    - 1 = function is enabled for the Rx1 channel.
    - 0 = function is not enabled for the Rx1 channel.

MYKONOS_getRx2GainCtrlPin

```
mykonosGpioErr_t
   MYKONOS_getRx2GainCtrlPin(mykonosDevice_t
   *device, uint8_t *incStep, uint8_t
   *decStep, mykonosGpioSelect_t
   *rx2GainIncPin, mykonosGpioSelect_t
   *rx2GainDecPin, uint8_t *enable)
```

This application programming interface (API) function returns the configuration for the GPIO inputs and step sizes used to control the gain index in manual gain control (MGC) input pin control mode for the Rx2 signal chain.

Parameters

- *incStep: A pointer to a variable that contains the step used for gain increment.
- *decStep: A pointer to a variable that contains the step used for gain decrement.
- *rx2GainIncPin: A pointer to a variable that has the pin used for gain increment.
- *rx2GainDecPin: A pointer to a variable that has the pin used for gain decrement.
- *enable: A pointer to a variable that contains the enable status for the Rx2 channel.
    - 1 = function is enabled for the Rx2 channel.
    - 0 = function is not enabled for the Rx2 channel.

3.3 V GENERAL-PURPOSE INPUT/OUTPUT OVERVIEW

The device provides 12, 3.3 V capable (GPIO_3P3_x) general-purpose input/output signals that can be configured for numerous functions. The GPIO_3P3_x pins control or monitor external devices, and several features are included to facilitate this control/ monitor function. In this user guide, the naming convention used to describe these pins is GPIO_3P3_x, where x is the number of the port.

Some of the GPIO_3P3_x pins can also be configured as outputs for the auxiliary DACs. To prevent any conflict on the GPIO_3P3_x pins, the auxiliary DAC has priority; that is, if the auxiliary DAC is powered up using the application programming interface (API) function, the pin takes on the auxiliary DAC function and the GPIO output buffer is tristated. Refer to the Auxiliary DACs section for more details on auxiliary DAC configuration and operation.

This section describes control of the GPIO_3P3_x signals and their behavior in detail. It also outlines how to program the GPIO_3P3_x API structure parameters and use API functions so that the desired signals are available on the appropriate pins.

API Description

As shown in Figure 102, there are a number of functionalities available in this block that can be enabled and then interacted with over the GPIO_3P3_x interface. There are crosspoints and an I/O buffer that must be configured properly to enable particular functionality. The API package provides functions that allow users to configure those blocks to their desired states. The following sections provides a list of general API functions and short descriptions of their functionalities.

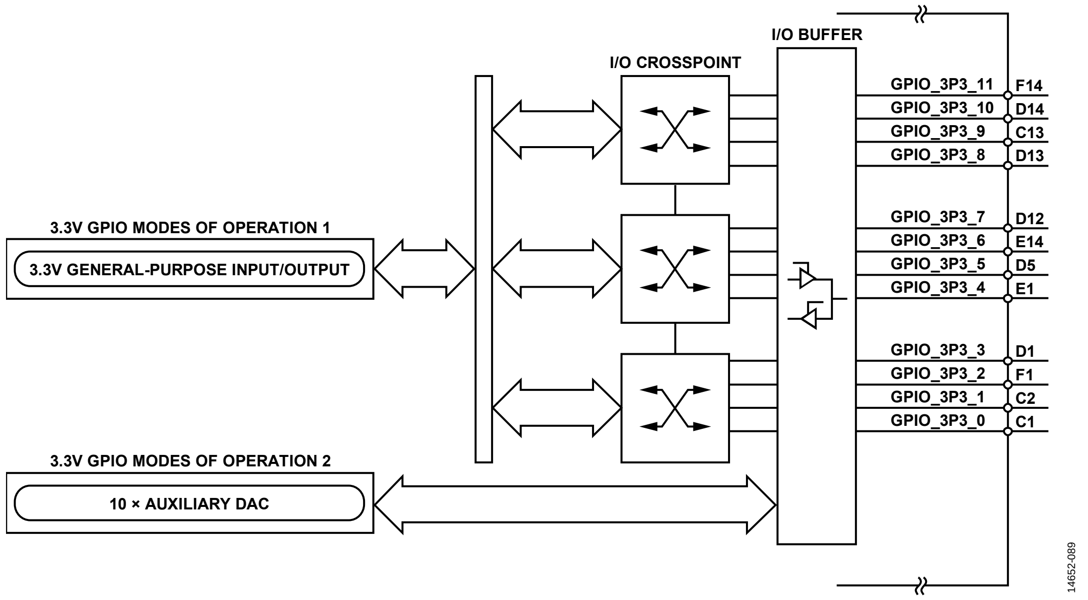


Figure 102. High Level Overview of the GPIO_3V3_x Interface

MYKONOS_setupGpio3v3

```
mykonosGpioErr_t
   MYKONOS_setupGpio3v3(mykonosDevice_t
   *device)
```

This function sets the GPIO_3P3_x configuration registers. It configures the pin direction for each GPIO_3P3_x pin, as well as the crosspoint. This function relies on correct settings of the device → auxIo → gpio3v3 structure members. A quick overview of this structure follows:

```
typedef struct
{
    uint16_t gpio3v3Oe;
/*!< Pin direction: bit per 3.3v GPIO,
0=Input, 1=Output from the AD9371 device
*/
    mykonosGpio3v3Mode_t
gpio3v3SrcCtrl3_0;     /*!< Mode for
GPIO3v3[3:0] pins */
    mykonosGpio3v3Mode_t
gpio3v3SrcCtrl7_4;     /*!< Mode for
GPIO3v3[7:4] pins */
    mykonosGpio3v3Mode_t
gpio3v3SrcCtrl11_8;    /*!< Mode for
GPIO3v3[11:8] pins */
} mykonosGpio3v3_t;
```

The gpio3v3Oe member controls the configuration of the input/output (I/O) buffer shown in Figure 102. The gpio3v3SrcCtrlNN_N structure member control configuration of the I/O crosspoint is shown in Figure 102. Options for gpio3v3SrcCtrlNN_N are as follows:

- GPIO3V3_LEVELTRANSLATE_MODE—level translate mode, signal level on low voltage GPIO output on the GPIO_3P3_x pins.
- GPIO3V3_INVLEVELTRANSLATE_MODE—inverted level translate mode, inverse of signal level on low voltage GPIO output on GPIO_3P3_x pins.
- GPIO3V3_BITBANG_MODE—manual mode; the API function sets the output pin levels and reads the input pin levels.
- GPIO3V3_EXTATTEN_LUT_MODE—GPIO_3P3_x output level follows the Rx1/Rx2 gain table external control 4-bit field.

Use the transceiver evaluation software (TES) for setting the values for the members of this structure, ensuring that there is no conflict with the other GPIO functionalities.

MYKONOS_setGpio3v3Oe

```
mykonosGpioErr_t
   MYKONOS_setGpio3v3Oe(mykonosDevice_t
   *device, uint16_t gpio3v3OutEn)
```

This is a helper function that is called by the MYKONOS_ setupGpio3v3() function.

This function sets the GPIO_3P3_x direction given by the gpio3v3OutEn parameter. The direction can be either output or input. A bit set to 1 indicates that the pin is configured as an output. A bit set to 0 indicates that the pin is configured as an input.

For example, setting gpio3v3OutEn = 0x02 configures GPIO_3P3_2 as an output and the rest of the GPIO_3P3_x pins as inputs.

The function parameter is as follows:

- gpio3v3OutEn. The valid range for this variable is from 0x0 to 0x0FFF. Each bit represents the corresponding GPIO_3P3_x pin (Bit 0 represents GPIO_3P3_0, Bit 1 represents GPIO_3P3_1, and so on.). The direction of the input buffer is set by each bit value: 0 = input, and 1 = output.

MYKONOS_getGpio3v3Oe

```
mykonosGpioErr_t
   MYKONOS_getGpio3v3Oe(mykonosDevice_t
   *device, uint16_t *gpio3v3OutEn)
```

This function reads back the current status of the GPIO_3P3_x direction set in the device. The direction can be either output or input. The function parameter returns a bit per the GPIO_3P3_x pin, where 1 outputs from the device and 0 inputs to the device.

Parameters

- *gpio3v3OutEn: a pointer to the data to be returned with the output to enable the GPIO_3P3_x pins in a bit field format.

MYKONOS_setGpio3v3SourceCtrl

```
mykonosGpioErr_t
   MYKONOS_setGpio3v3SourceCtrl(mykonosDevic
   e_t *device, uint16_t gpio3v3SrcCtrl)
```

This is a helper function that is called by the MYKONOS_ setupGpio3v3() function.

This function configures the crosspoint for different GPIO_3P3_x functionality. This function only affects the GPIO_3P3_x pins that have their output enable direction set by the MYKONOS_getGpio3v3Oe() function as outputs.

Parameters

- gpioSrcCtrl (nibble-based source control): This parameter is a 12-bit number that contains three nibbles that set the source control. Use the TES to generate correct settings for the desired configuration.

MYKONOS_getGpio3v3SourceCtrl

```
mykonosGpioErr_t
   MYKONOS_getGpio3v3SourceCtrl(mykonosDev
   ice_t *device, uint16_t
   *gpio3v3SrcCtrl)
```

This function reads back the current status of the GPIO_3P3 source control for different GPIO functionality.

Parameters

- gpio3v3SrcCtrl (nibble-based source control): this is a 12-bit number that contains three nibbles that represent the current settings of the source control (crosspoint configuration).

3.3 V GENERAL-PURPOSE INPUT/OUTPUT CONTROL

Figure 103 shows a graphical representation of the GPIO_3P3_x signals in manual mode. In this mode, the user can configure the GPIO_3P3_x inputs/outputs as outputs or inputs. If a GPIO_3P3_x pin is configured as an output, the user can control its logic level. If a GPIO_3P3_x pin is configured as an input, the user can read back the voltage level present at the pin. In both cases, a logic low or 0 corresponds to the ground voltage level. A logic high or 1 corresponds to the 3.3 V voltage level present at the VDDA_3P3 pin.

Some of the GPIO_3P3_x pins can also be configured to provide access to the auxiliary DAC outputs (as per Figure 102). To prevent any conflict on the GPIO_3P3_x pins, the auxiliary DAC has priority. If the auxiliary DAC is powered up using its application programming interface (API) function, the pin takes on the auxiliary DAC function and the GPIO output buffer is tristated. Refer to the Auxiliary DACs section for more details on auxiliary DAC configuration and operation.

When using GPIO manual mode only or when working with any other GPIO mode in parallel, use the transceiver evaluation software (TES) to set up the API auxIo/gpio3v3 structure with a valid configuration.

The GPIO_3P3_x interface in manual mode uses the input/output crosspoint. This crosspoint operates using nibbles (4 bits), meaning that the GPIO_3P3_x pins are controlled in groups of four.

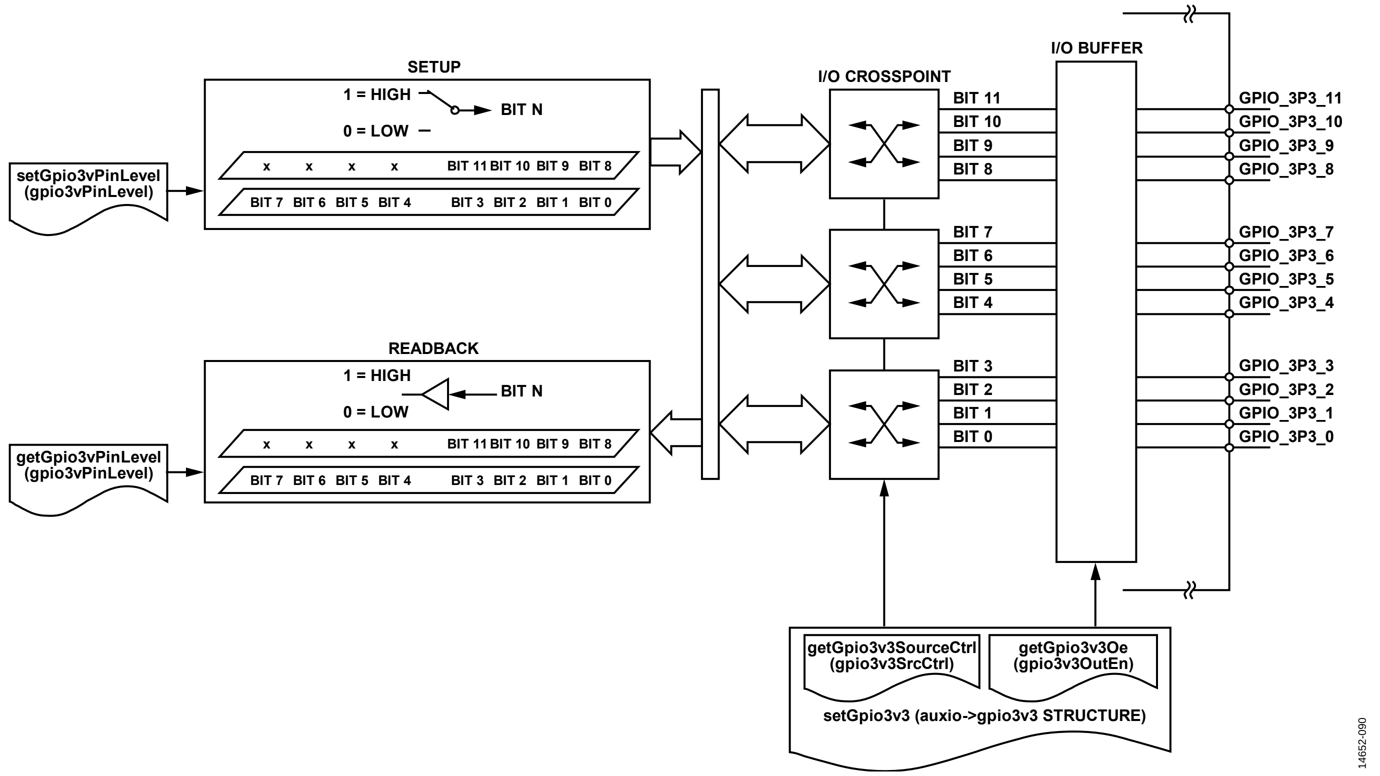


Figure 103. GPIO_3P3_x in Manual Mode of Operation

API Description

The application programming interface (API) package provides functions that allow users to operate the GPIO_3P3_x signals in manual mode. Before the user can set the GPIO_3P3_x signals in manual mode, the crosspoint and input/output (I/O) buffers must be configured. Use the MYKONOS_setupGpio3v3() function to properly configure the crosspoints and I/O buffers. This function relies on correct configuration of device → auxIo → gpio3v3 structure members. Use the transceiver evaluation software (TES) to generate correct settings for this structure. After the crosspoints and I/O buffers are configured, the user can operate GPIO_3P3_x signals in manual mode. The following sections describe the manual mode API functions and provide short descriptions of their functionalities.

MYKONOS_getGpio3v3PinLevel

```
mykonosGpioErr_t
   MYKONOS_getGpio3v3PinLevel(mykonosDevice_
   t *device, uint16_t *gpio3v3PinLevel)
```

This function reads the GPIO_3P3_x pins that are set to be inputs in manual mode. Any pin set to be an output reads back as zero. The pin level is returned in the form of a single 16-bit word. The returned value is a bit per pin. GPIO_3P3_0 returns on Bit 0 of the gpio3v3PinLevel parameter, GPIO_3P3_1 returns on Bit 1 of the gpio3v3PinLevel, and so on. A logic low level returns 0, and a logic high level returns 1.

Parameters

- *gpio3v3PinLevel: input GPIO pin levels are read back on the pins assigned as inputs (1 bit per pin).

MYKONOS_setGpio3v3PinLevel

```
mykonosGpioErr_t
   MYKONOS_setGpio3v3PinLevel(mykonosDevice_
   t *device, uint16_t gpio3v3PinLevel)
```

This function sets the GPIO_3P3_x output pin level. This function only affects the GPIO_3P3_x pins that are set to be outputs in manual mode. GPIO_3P3_0 reflects the status of Bit 0 of the gpio3v3PinLevel parameter, GPIO_3P3_1 reflects the status of Bit 1 of the gpio3v3PinLevel, and so on. A logic low is set at the GPIO output and its corresponding bit is set to 0. A logic high is set at the GPIO_3P3_x output (the corresponding bit is set to 1).

Parameters

- gpio3v3PinLevel: This parameter describes the level output for each GPIO_3P3_x pin (0 = low output, and 1 = high output).

MYKONOS_getGpio3v3SetLevel

```
mykonosGpioErr_t
   MYKONOS_getGpio3v3SetLevel(mykonosDevice_
   t *device, uint16_t *gpio3v3SetLevel)
```

This function allows the user to read the value of each GPIO_3P3_x output pin set to output in manual mode.

Parameters

- *gpio3v3PinSetLevel: a pointer to the 16-bit variable that contains the level of each GPIO_3P3_x pin (1 bit per pin). 0 sets the output to a low level, and 1 sets the output to a high level.

GENERAL-PURPOSE INTERRUPT OVERVIEW

The device provides the user with an interrupt signal in the form of a single, general-purpose interrupt output pin GP_INTERRUPT. This pin asserts to a logic high level when interrupt events occur. When the baseband processor (BBP) detects a rising edge on the GP_INTERRUPT pin, the BBP uses an application programming interface (API) function to determine the source of the interrupt.

The user can control (enable or disable) assertion of the GP_INTERRUPT pin by certain events. Figure 104 shows interrupt sources for the device and outlines control blocks for them. The ARM error interrupt cannot be ignored and can always assert the GP_INTERRUPT pin. Note that the logic block format in Figure 104 is intended to show the functional operation of the GP_INTERRUPT signal but does not necessarily represent the method of implementation inside the device.

API Description

MYKONOS_configGpInterrupt

```
mykonosGpioErr_t
   MYKONOS_configGpInterrupt(mykonosDevice_t
   *device, uint16_t gpMask)
```

This function sets the general-purpose (GP) interrupt register bit mask to enable interrupt sources that assert the GP_INTERRUPT pin. The GP_INTERRUPT pin only asserts for the enabled sources. The events that cause the GP_INTERRUPT pin to assert are user selectable by setting the gpMask parameter in this function. The device default is gpMask = x1FF, which means ignore all events. The ARM error interrupt cannot be ignored and can always assert the GP_INTERRUPT pin. Table 163 outlines possible interrupt sources for this device.

Parameters

- gpMask: This value is passed to enable one or more general purpose interrupt sources (1 = ignores the source, and 0 = enables the source interrupt to the GP_INTERRUPT pin).

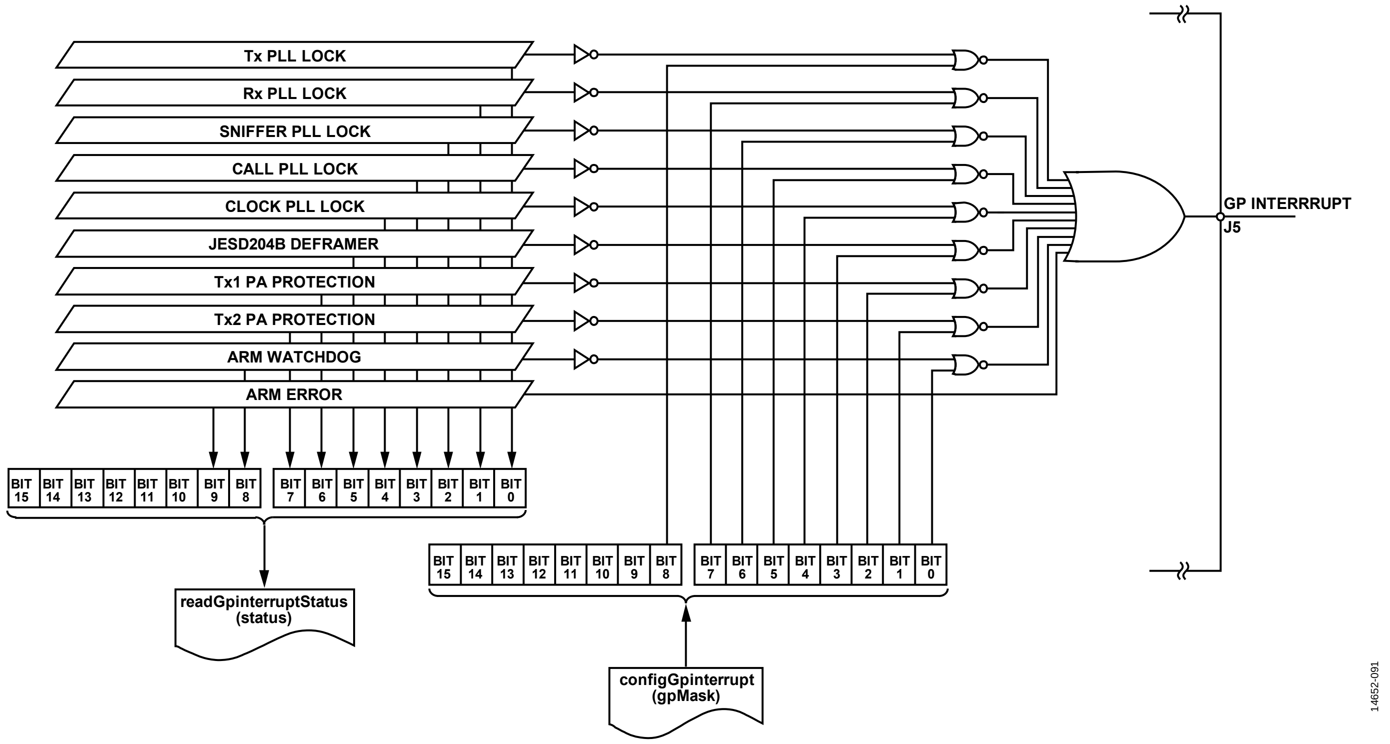


Figure 104. General-Purpose Interrupt Structure

Table 163. GP_Interrupt Configurations

gpMask Bits	Description	Reset
0	Tx PLL lock 0 = allows phased-locked loop (PLL) unlocking to assert the GP_INTERRUPT pin 1 = ignores Tx PLL lock	0x1
1	Rx PLL lock 0 = allows PLL unlocking to assert the GP_INTERRUPT pin 1 = ignores Rx PLL lock	0x1

gpMask Bits	Description	Reset
2	Sniffer PLL lock 0 = allows PLL unlocking to assert the GP_INTERRUPT pin 1 = ignores sniffer PLL lock	0x1
3	Calibration PLL lock 0 = allows PLL unlocking to assert the GP_INTERRUPT pin 1 = ignores calibration PLL lock	0x1
4	Clock PLL lock 0 = allows PLL unlocking to assert the GP_INTERRUPT pin 1 = ignores clock PLL lock	0x1
5	JESD204 deframer interrupt 0 = allows the JESD204B deframer interrupt to assert the GP_INTERRUPT pin 1 = ignores JESD204 deframer interrupt	0x1
6	Tx1 PA protection 0 = allows the Tx1 PA protection event to assert the GP_INTERRUPT pin 1 = ignores Tx1 PA protection	0x1
7	Tx2 PA protection 0 = allows the Tx2 PA protection event to assert the GP_INTERRUPT pin 1 = ignores Tx2 PA protection	0x1
8	ARM watchdog 0 = allows the ARM watchdog timeout to assert the GP_INTERRUPT pin 1 = ignores the watchdog timeout event	0x1
9	ARM error. 0 = allows the ARM to assert the GP_INTERRUPT pin when an error occurs. 1 = ignores the ARM error event.	0x1
[15:10]	Reserved for future use	0x0

MYKONOS_readGpInterruptStatus

```
mykonosGpioErr_t
   MYKONOS_readGpInterruptStatus(mykonosDevi
   ce_t *device, uint16_t *status)
```

This function reads the GP interrupt status to determine what caused the GP_INTERRUPT pin to assert. When the baseband processor (BBP) detects a rising edge on the GP_INTERRUPT pin, this function allows the BBP to determine the source of the interrupt. The value returned in the status parameter shows one or more sources for the interrupt, as shown in Table 164.

Note that the phase-locked loop (PLL) unlock bits are not sticky. These bits follow the current status of the PLLs. If the PLL relocks, the status bit clears as well. The GP_INTERRUPT pin is the logical OR of all the sources. When all the status bits are low, the GP_INTERRUPT pin is low. The status word readback shows the current value for all interrupt sources, even if they are disabled by the mask using the MYKONOS_configGpInterrupt() function. However, the GP_INTERRUPT pin only asserts for the enabled sources.

Parameters

- status: This parameter returns the IRQ source(s) that caused the GP_INTERRUPT pin to assert.

Table 164. GP_INTERRUPT Status

Status Bit(s)	Description
0	1 = Tx PLL unlock
1	1 = Rx PLL unlock
2	1 = sniffer PLL unlock
3	1 = calibration PLL unlock
4	1 = clock PLL unlock
5	1 = JESD204 deframer interrupt occurred
6	1 = Tx1 PA protection event
7	1 = Tx2 PA protection event
8	1 = ARM watchdog timeout
9	1 = ARM interrupt occurred
[15:10]	Reserved for future use

AUXILIARY CONVERTERS—AUXDAC_x, AUXADC_x, AND TEMPERATURE SENSOR

This section describes the setup and operation of the auxiliary data converters in the device. These features are included to simplify control tasks, take static measurements during normal operation, and provide flexibility that can be used across multiple applications without adding external components. The following sections provide details needed to set up each block and the application programming interface (API) functions required to control operation.

AUXILIARY DACs

The 10 auxiliary digital-to-analog converters (DACs) are 10-bit, general-purpose DACs. An auxiliary DAC is a segmented 10-bit current source array. Two additional bits of dynamic range are created through a reference voltage selection. The aggregate auxiliary DAC spans approximately 12 bits of dynamic range. The auxiliary DAC output range spans from 0.5 V to 3.0 V (Changed to 3.0 V to match d/s). Each auxiliary DAC is capable of sourcing 10 mA. The auxiliary DACs and the 12 GPIO_3P3_x ports are multiplexed onto the same pins. Figure 105 shows the auxiliary DAC block. Note that, for stability, a 100 nF bypass capacitor must be placed at each active auxiliary DAC output.

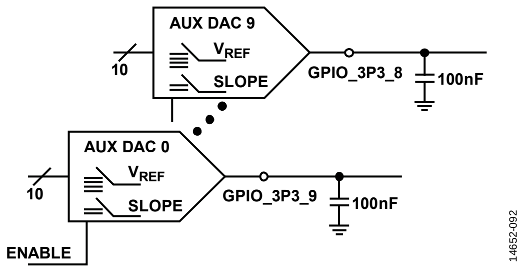


Figure 105. Auxiliary DACs Block Diagram

Hardware Configuration

The auxiliary DACs have priority use of the pins when a given auxiliary DAC is enabled. The pin takes on the auxiliary DAC function and the corresponding GPIO_3P3_x output driver is tristated. See Table 166 for auxiliary DAC to GPIO_3P3_x pin mapping.

The auxiliary DACs are designed to be used in feedback loop operations. For example, one can generate a voltage supply used to control a voltage controlled crystal oscillator (VCXO) input. For such control system uses, the absolute value of the voltage output is not critical, but it is recommended that the voltage steps be 12-bit accurate and monotonic. The feedback of the servo loop renders the absolute level unimportant.

Note that when using an auxiliary DAC as a controlled voltage reference, take care regarding the tolerances on the 3.3 V domain that serve as the supplies for the auxiliary DACs.

Auxiliary DAC Control Software Control Procedure

The flowchart shown in Figure 106 illustrates the process required to properly control the auxiliary DACs when using them to generate control voltage outputs.

Rise and Fall Times

Table 165 provides an example of auxiliary DAC rise and fall times.

Table 165. Example Auxiliary DAC Rise and Fall Times

Voltage Change (V)	Bit Change	Rise Time (µs)	Fall Time (µs)
1.325	768	2.86	11.86

PSRR

The auxiliary DAC power supply rejection ratio (PSRR) is measured to 100 kHz (20 mV ripple injection). Worst case PSRR is approximately 6 dB and occurs near the maximum output levels.

Table 166. Auxiliary DAC to GPIO_3P3_x Pin Mapping

Pin Number	Type	Mnemonic	Description
C13	Output	GPIO_3P3_9	Auxiliary DAC 0 output pin.
D12	Output	GPIO_3P3_7	Auxiliary DAC 1 output pin
E14	Output	GPIO_3P3_6	Auxiliary DAC 2 output pin
D14	Output	GPIO_3P3_10	Auxiliary DAC 3 output pin
C1	Output	GPIO_3P3_0	Auxiliary DAC 4 output pin
C2	Output	GPIO_3P3_1	Auxiliary DAC 5 output pin
D1	Output	GPIO_3P3_3	Auxiliary DAC 6 output pin
E1	Output	GPIO_3P3_4	Auxiliary DAC 7 output pin
D5	Output	GPIO_3P3_5	Auxiliary DAC 8 output pin
D13	Output	GPIO_3P3_8	Auxiliary DAC 9 output pin

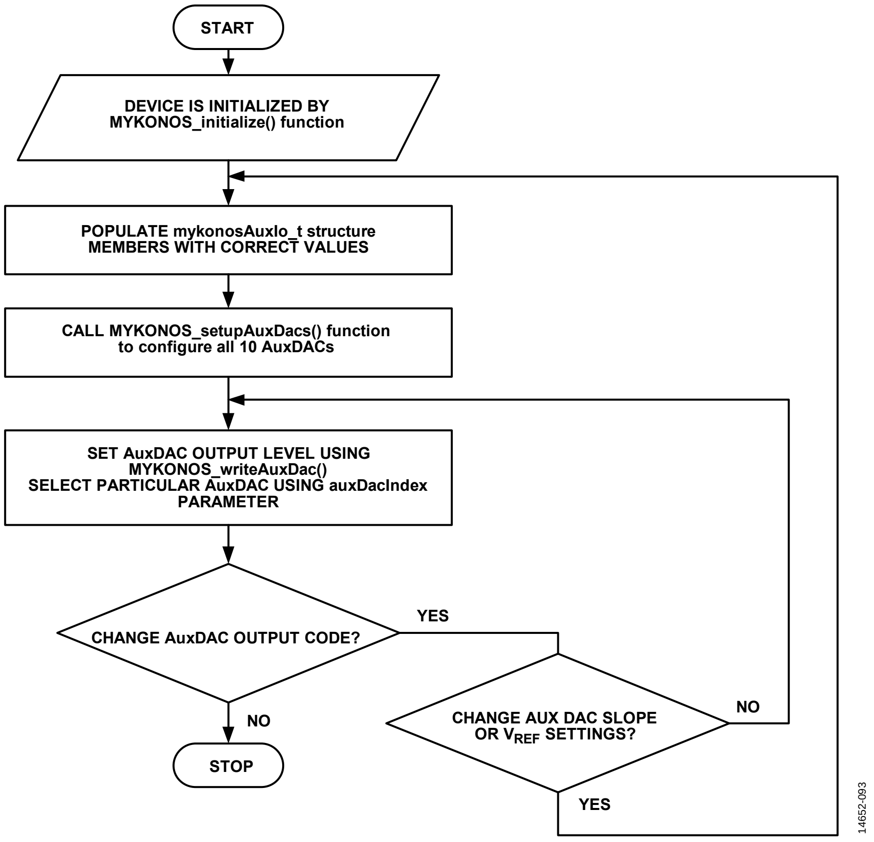

START
DEVICE IS INITIALIZED BY MYKONOS_initialize() function
POPULATE mykonosAuxIo_t structure MEMBERS WITH CORRECT VALUES
CALL MYKONOS_setupAuxDacs() function to configure all 10 AuxDACs
SET AuxDAC OUTPUT LEVEL USING MYKONOS_writeAuxDac() SELECT PARTICULAR AuxDAC USING auxDacIndex PARAMETER
CHANGE AuxDAC OUTPUT CODE?
YES
NO
STOP
CHANGE AUX DAC SLOPE OR V_{REF} SETTINGS?
NO
YES
14652-093

Figure 106. Auxiliary DAC Control Procedure

Auxiliary DACs Software Configuration

The process required to enable the auxiliary DACs requires the user to select values for members in the mykonosAuxIo_t substructure associated with the auxiliary DAC. The following descriptions outline the possible values for the mykonosAuxIo_t structure members and the interpretation of these values. Each auxiliary DAC can be independently configured.

The device → auxIo → auxDacEnable structure member settings follow:

- When set to 0, this structure member disables the corresponding auxiliary DAC (Bit 0 corresponds to Auxiliary DAC 0, Bit 1 corresponds to Auxiliary DAC 1, and so on).
- When set to 1, this structure member enables the corresponding auxiliary DAC (Bit 0 corresponds to Auxiliary DAC 0, Bit 1 corresponds to Auxiliary DAC 1, and so on).

The device → auxIo → auxDacSlope[i] structure member settings follow:

- When set to 0, this structure member sets the corresponding auxiliary DAC voltage output (V_{OUT}) codes to be 1.404 mV/code (see Figure 107).
- When set to 1, this structure member sets the corresponding auxiliary DAC voltage output (V_{OUT}) codes to be 0.702 mV/code (see Figure 108).

The device → auxIo → auxDacVref[i] structure member settings follow:

- When set to 0, this structure member sets the corresponding auxiliary DAC output midpoint to 1 V (see Figure 107 and Figure 108).
- When set to 1, this structure member sets the corresponding auxiliary DAC output midpoint to 1.5 V (see Figure 107 and Figure 108).
- When set to 2, this structure member sets the corresponding auxiliary DAC output midpoint to 2 V (see Figure 107 and Figure 108).
- When set to 3, this structure member sets the corresponding auxiliary DAC output midpoint to 2.5 V (see Figure 107 and Figure 108).

The device → auxIo → auxDacValue[i] structure member settings follow:

- The value programmed to this structure member is loaded to the corresponding auxiliary DAC and is output as the corresponding analog voltage. The value programmed to this member must be in range between 0 and 1023 (10-bit DAC code).

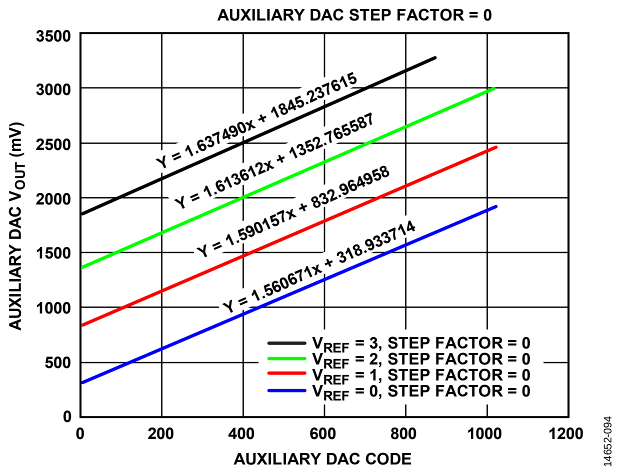


Figure 107. Auxiliary DAC Output Voltage (VOUT) vs. Auxiliary DAC Code for the Different auxDacVref Values (auxDacSlope = 0)

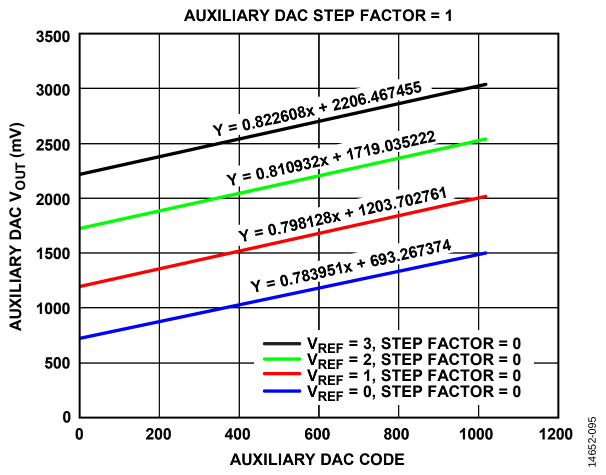


Figure 108. Auxiliary DAC Output Voltage (V_{OUT}) vs. Auxiliary DAC Codes for Different auxDacVref Values (auxDacSlope = 1)

MYKONOS_setupAuxDacs

After correctly configuring all structure members mentioned in the Auxiliary DACs Software Configuration section, execute the following application programming interface (API) function:

```
mykonosErr_t
   MYKONOS_setupAuxDacs(mykonosDevice_t
   *device)
```

This function reads data from the device auxiliary input/output substructure and then loads this data into the device. This function programs all configuration parameters for 10 auxiliary DACs at the same time, including the enable/disable, slope, V_{REF} (midpoint), and the initial auxiliary DAC code.

This function can be called any time after MYKONOS_initialize() to reconfigure, enable, or disable the different auxiliary DAC outputs. The auxiliary DACs are used in manual control mode.

MYKONOS_writeAuxDac

After calling this setup function, it is possible to change a particular auxiliary DAC code by calling the following function:

```
mykonosErr_t
   MYKONOS_writeAuxDac(mykonosDevice_t
   *device, uint8_t auxDacIndex, uint16_t
   auxDacCode)
```

This function updates the 10-bit code that controls the auxiliary DAC output voltage. The auxiliary DAC code is updated for the specified auxiliary DAC and is written to the device data structure for future reference.

Parameters

- *device: This is a pointer to the device settings structure.
- auxDacIndex: an index that selects which auxiliary DAC to set the new DAC code. The allowable values are from 0 to 9.
- auxDacCode: the DAC code to update the auxiliary DAC; sets the output voltage of selected DAC. The programmed value must be in range between 0 and 1023 (10-bit DAC code).

This function can be called any time after MYKONOS_ initialize() and MYKONOS_setupAuxDacs().

AUXILIARY ADC

The auxiliary analog-to-digital converter (ADC) is a single 12-bit auxiliary converter with four multiplexed inputs that cover an input level range from 0.05 V to 3.25 V. The auxiliary ADC allows monitoring of the desired voltages, such as a power amplifier (PA) power detector or an external temperature sensor. Figure 109 shows the general auxiliary ADC input connection scheme.

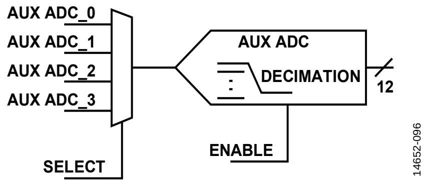


Figure 109. Auxiliary ADC Input Block Diagram

Table 167 outlines the hardware connections on the device for the auxiliary ADC inputs. A small value capacitor (680 pF) can be placed on the auxiliary ADC input pins to improve noise performance.

Auxiliary (AUX) ADC Readback Software Control Procedure

The flowchart shown in Figure 111 illustrates the process required to properly control the auxiliary ADCs when using them to read voltage levels. A typical plot of output code vs. input voltage is shown in Figure 110.

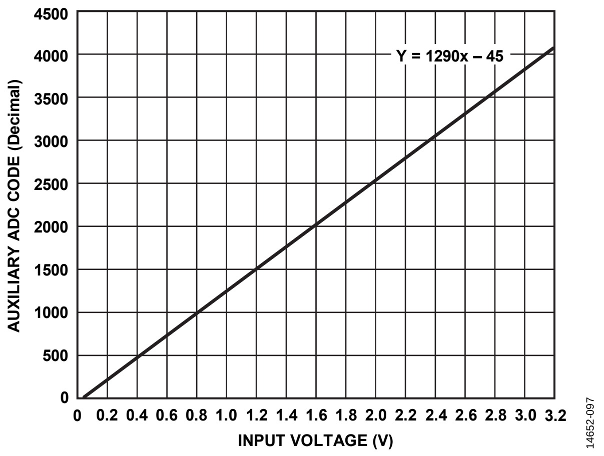


Figure 110. Auxiliary ADC Code vs. Input Voltage

Table 167. Auxiliary ADC Input Mapping

Pin Number	Type	Mnemonic	Description
E13	Input	AUXADC_0	Auxiliary ADC 0 input pin
C11	Input	AUXADC_1	Auxiliary ADC 1 input pin
C12	Input	AUXADC_2	Auxiliary ADC 2 input pin
D11	Input	AUXADC_3	Auxiliary ADC 3 input pin

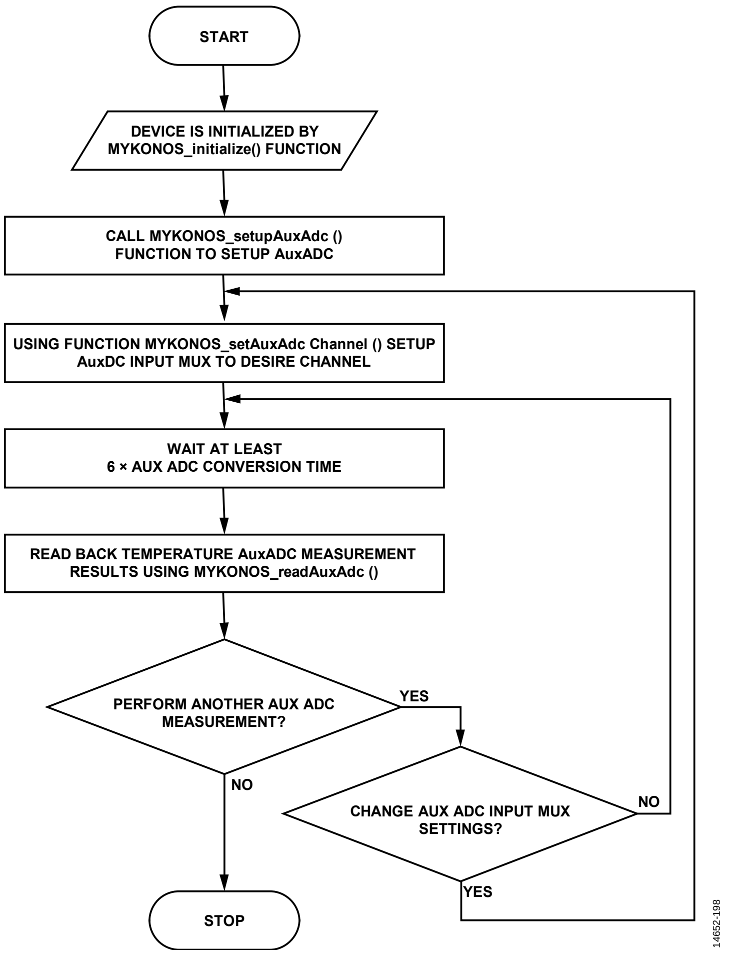

START
DEVICE IS INITIALIZED BY MYKONOS_initialize() FUNCTION
CALL MYKONOS_setupAuxAdc () FUNCTION TO SETUP AuxADC
USING FUNCTION MYKONOS_setAuxAdc Channel () SETUP AuxDC INPUT MUX TO DESIRE CHANNEL
WAIT AT LEAST 6 × AUX ADC CONVERSION TIME
READ BACK TEMPERATURE AuxADC MEASUREMENT RESULTS USING MYKONOS_readAuxAdc ()
PERFORM ANOTHER AUX ADC MEASUREMENT?
YES
NO
CHANGE AUX ADC INPUT MUX SETTINGS?
NO
YES
STOP
14652-198

Figure 111. Auxiliary (AUX) ADC Readback Procedure

Auxiliary ADCs Software Configuration

The user must use an application programming interface (API) command to read back the ADC code for a selected input. A list of API commands with instructions detailing how to use them is outlined in this section.

MYKONOS_setupAuxAdcs

The following function configures an auxiliary ADC with the requested decimation factor; excecute this command first.

```
mykonosErr_t MYKONOS_setupAuxAdcs
    (mykonosDevice_t *device, uint8_t
    adcDecimation,uint8_t  enable)
```

The auxiliary ADC clock is automatically set as close as possible to 40 MHz. The auxiliary ADC conversion time = (1/40 MHz) × decimation, where the decimation ranges from 256 auxiliary ADC clock cycles to 32,768 auxiliary ADC clock cycles.

Parameters

- *device: This is a pointer to the device settings structure.
- adcDecimation: ADC decimation factor. The allowable values are in the 0 to 7 range.

$$Decimation = 256 \times 2^{adcDecimation} \quad (19)$$

$$Conversion\ Time = ADC\ Clock\ Cycles \times Decimation \quad (20)$$

where *ADC Clock Cycles* = (1/40 MHz) = 25 ns.

For example, if adcDecimation = 4, the ADC conversion time is approximately 0.1 ms

- enable: When set to 0, this disables the auxiliary ADC. When set to 1, this enables the auxiliary ADC.

This function can be called any time after MYKONOS_ initializer().

MYKONOS_setAuxAdcChannel

The next API function to be used with the auxiliary ADC is as follows:

```
mykonosErr_t MYKONOS_setAuxAdcChannel
    (mykonosDevice_t *device,  uint8_t
    auxAdcChannel )
```

This function sets the selected input channel of the auxiliary ADC. After setting the auxiliary ADC channel, wait at least 1 auxiliary ADC conversion time before reading back the auxiliary ADC value.

Parameters

- *device: A pointer to the device settings structure
- auxAdcChannel: this parameter selects the auxiliary ADC inputs and internal temperature sensor as follows:
    - When set to 0, this parameter selects Auxiliary ADC Input 0. Refer to Table 167 for hardware configuration.
    - When set to 1, this parameter selects Auxiliary ADC Input 1. Refer to Table 167 for hardware configuration.
    - When set to 2, this parameter selects Auxiliary ADC Input 2. Refer to Table 167 for hardware configuration.
    - When set to 3, this parameter selects Auxiliary ADC Input 3. Refer to Table 167 for hardware configuration.
    - When set to 16, this parameter selects the internal temperature sensor.

This function can be called any time after MYKONOS_initialize() and MYKONOS_setupAuxAdcs().

MYKONOS_readAuxAdc

To read the auxiliary ADC data for the selected auxiliary ADC input, use the following application programming interface (API) function:

```
mykonosErr_t MYKONOS_readAuxAdc
    (mykonosDevice_t *device,  uint16_t
    *adcCode )
```

Before using this function to read back the output value of the selected auxiliary ADC, ensure that at least one ADC conversion time passes after setting the auxiliary ADC channel.

Parameters

- *device: This is a pointer to the device settings structure.
- *adcCode: This is a pointer for the 12-bit ADC read value.

This function can be called any time after MYKONOS_initialize(). First, configure the auxiliary ADC using the MYKONOS_setupAuxAdcs() and MYKONOS_setAuxAdcChannel() functions.

TEMPERATURE SENSOR

The transceiver provides the user with an option to use the auxiliary ADC to measure the transceiver die temperature. This temperature sensor is on the die; therefore, it cannot be used to measure the ambient device temperature.

Figure 112 shows the temperature sensor output vs. the temperature measured at the surface of the device.

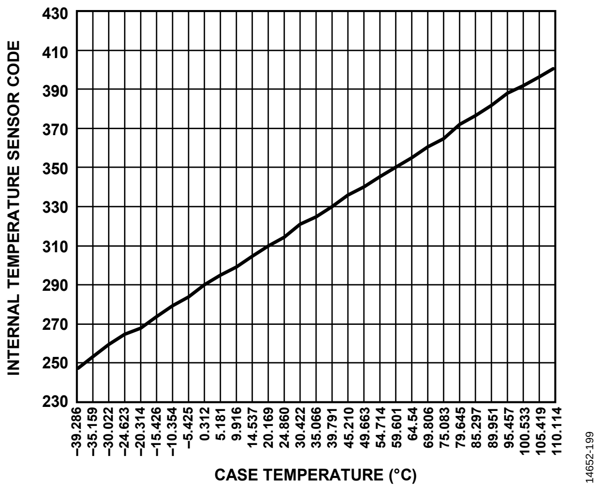


Figure 112. Internal Temperature Sensor Code vs. Case Temperature

Software Configuration

The API provides functions to read back the internal die temperature sensor output. Figure 113 outlines the procedure that must be followed to read back the internal temperature sensor. A list of API commands with instructions detailing how to use them is outlined in the following section.

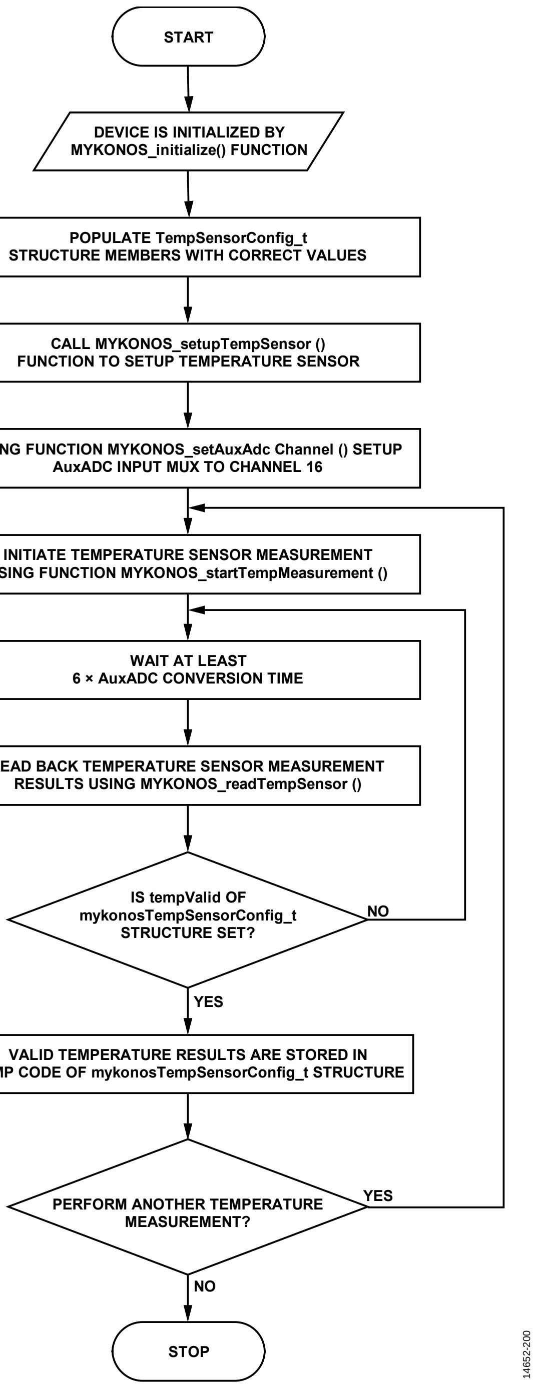

START
DEVICE IS INITIALIZED BY MYKONOS_initialize() FUNCTION
POPULATE TempSensorConfig_t STRUCTURE MEMBERS WITH CORRECT VALUES
CALL MYKONOS_setupTempSensor () FUNCTION TO SETUP TEMPERATURE SENSOR
USING FUNCTION MYKONOS_setAuxAdc Channel () SETUP AuxADC INPUT MUX TO CHANNEL 16
INITIATE TEMPERATURE SENSOR MEASUREMENT USING FUNCTION MYKONOS_startTempMeasurement ()
WAIT AT LEAST 6 × AuxADC CONVERSION TIME
READ BACK TEMPERATURE SENSOR MEASUREMENT RESULTS USING MYKONOS_readTempSensor ()
IS tempValid OF mykonosTempSensorConfig_t STRUCTURE SET?
NO
YES
VALID TEMPERATURE RESULTS ARE STORED IN TEMP CODE OF mykonosTempSensorConfig_t STRUCTURE
PERFORM ANOTHER TEMPERATURE MEASUREMENT?
YES
NO
STOP
14652-200

Figure 113. Temperature Sensor Readback Procedure

The user must use API commands to read back the ADC code for a selected input. A list of API commands with instructions detailing how to use them is outlined in this section.

MYKONOS_setupTempSensor

The following function sets up the operation of the internal temperature sensor:

```
mykonosGpioErr_t
   MYKONOS_setupTempSensor(mykonosDevice_t *
   device, mykonosTempSensorConfig_t
   *tempSensor)
```

Before using this function, ensure that the MYKONOS_ setupAuxADC() function executes and MYKONOS_ setAuxAdcChannel() is configured to read back from the internal temperature sensor. Also ensure that the mykonosTempSensorConfig_t structure is populated with correct values.

Parameters

- * device: This is a pointer to the device settings structure.
- *tempSensor: this is a pointer to the mykonos-TempSensorConfig_t structure that holds the configuration settings for the temperature sensor.

The following members are the mykonosTempSensorConfig_t structure members:

- uint8_t tempDecimation—a 3-bit value that controls the auxiliary ADC decimation factor when used for temperature sensor calculations, according to the following:

  $$Auxiliary\ ADC\ decimation = 256 \times 2^{tempDecimation} \quad (21)$$

- uint8_t offset—an 8-bit offset added to the temperature sensor code internally.
- uint8_t overrideFusedOffset—a bit that overrides the factory calibrated offset value; uses the value stored in the offset member.
- uint8_t tempWindow—a 4-bit code with a resolution of 1°C/LSB. Each time a temperature measurement is performed, the device compares the current temperature against the previous value. If the value exceeds tempWindow, the windowExceeded member of the mykonosTempSensorStatus_t structure is set.

In the scenario where structure member values are outside of range, the MYKONOS_setupTempSensor() returns an error. Refer to the description in the application programming interface (API) help file for more details.

This function can be called any time after MYKONOS_ initialize().

MYKONOS_getTempSensorConfig

The next API function reads data from the temperature sensor registers and populates the *tempSensor data structure. After this function is executed, the *tempSensor parameter holds updated values from the device registers. This parameter is used as follows:

```
mykonosGpioErr_t
   MYKONOS_getTempSensorConfig(mykonosDevice
   _t *device,mykonosTempSensorConfig_t
   *tempSensor)
```

Parameters

- *device: This is a pointer to the device setting structure.
- *tempSensor: This is a pointer to the mykonosTemp-SensorConfig_t structure, which holds the configuration settings for the temperature sensor.

This function can be called any time after MYKONOS_ initialize().

MYKONOS_startTempMeasurement

This API function initiates a temperature sensor measurement.

```
mykonosGpioErr_t
   MYKONOS_startTempMeasurement(mykonosDevic
   e_t *device)
```

Before this function can be executed, the user must do the following:

- Set up the temperature sensor using the MYKONOS_ setupTempSensor() function.
- Connect the auxiliary ADC input mux to the internal temperature sensor (Channel 16 = 0x10) using the MYKONOS_setAuxAdcChannel() function.

After this function is executed, the internal temperature sensor block performs a measurement and updates the register values. The user can read back the temperature sensor information using the MYKONOS_readTempSensor() function.

Parameters

- *device: This is a pointer to the device settings structure.

This function can be called any time after MYKONOS_ initialize().

MYKONOS_readTempSensor

To read temperatures from the internal on die temperature sensor and update temperature sensor status information, use the following API command:

```
mykonosGpioErr_t
   MYKONOS_readTempSensor(mykonosDevice_t
   *device, mykonosTempSensorStatus_t
   *tempStatus)
```

Before using this function to read back the temperature sensor information, the user must do the following:

- Set up the temperature sensor using the MYKONOS_ setupTempSensor() function.
- Set up the auxiliary ADC input mux to the internal temperature sensor (Channel 16 = 0x10) using the MYKONOS_setAuxAdcChannel() function.
- Initiate temperature sensor measurements using the MYKONOS_startTempMeasurement() function. The user must call this function every time temperature readings must be performed.

The user must allow at least one computation period to elapse between the temperature measurement request function call, MYKONOS_startTempMeasurement(), and the temperature readback using the MYKONOS_readTempSensor() function.

The temperature sensor computation period equals six times the auxiliary ADC conversion time, which is determined by

$$Auxiliary\ ADC\ Conversion\ Time = \frac{256 \times 2^{tempDecimation}}{40 \times 10^6} \quad (22)$$

The tempDecimation parameter is stored in the mykonos-TempSensorConfig_t structure.

The results of a temperature measurement function call are stored in the mykonosTempSensorStatus_t structure. If a valid measurement is achieved, the tempValid member of the mykonos-TempSensorStatus_t structure is set, and the tempCode of the mykonosTempSensorStatus_t structure contains the actual reading.

Parameters

- *device: This is a pointer to the device settings structure.
- *tempStatus: a pointer to the mykonosTempSensorStatus_t structure that is updated with the temperature sensor readings.

The description of the mykonosTempSensorStatus_t structure members is as follows:

- int16_t tempCode. This structure member contains a 16-bit signed temperature value. This value is reported in degrees Celsius.
- uint8_t windowExceeded. This flag is set if the absolute value of the difference between the previous and current temperature measurement is greater than the value stored in the temperature configuration tempWindow, a member of mykonosTempSensorConfig_t structure.
- uint8_t windowHiLo. When windowExceeded flag is set, this bit is set to 1 if the current value is greater than the previous value. This bit is set to 0 if the current value is less than or equal to the previous value.
- uint8_t tempValid. This structure member indicates valid measurement results when the temperature reading is complete, and a valid temperature value is stored in the tempCode structure member.

RF PORT INTERFACE

This section provides the recommended RF transmitter and receiver interfaces to obtain optimal device performance. This section includes data regarding the expected RF port impedance values, potential impedance matching network techniques, and examples of impedance matching networks. Some reference material is also provided regarding board layout techniques and balun selection guidelines.

The device is a highly integrated transceiver with two transmitters, two main receivers, and an observation channel with two observation receiver inputs and three sniffer receiver inputs. All input and output ports are differential; therefore, external impedance matching networks are required on transmitter and receiver ports to convert them from single-ended to differential as well as to achieve performance levels indicated on the data sheet.

SERIES AND PARALLEL IMPEDANCE MODELS

In describing differential impedances, two equivalent models are commonly used: series equivalent differential impedance (SEDZ) and parallel equivalent differential impedance (PEDZ). Both formats are used throughout this user guide, and descriptions of the models and the conversion between the two formats are provided in the following sections.

Series Equivalent Differential Impedance (SEDZ) Models

The SEDZ model is depicted in Figure 114. Note that series refers to the fact that the resistance is in series with the reactance formed by the parallel combination of the capacitor and inductor (see Figure 114).

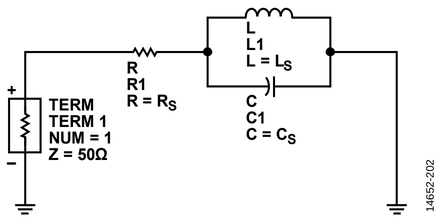


Figure 114. SEDZ Definition

Mathematically, the differential series impedance is represented by

$$SEDZ = R_S + jX_S \tag{23}$$

The + sign implies that the elements are in series. The jX_S can be positive (inductive) or negative (capacitive).

Parallel Equivalent Differential Impedance (PEDZ) Model

The PEDZ model is depicted in Figure 115. In contrast to the SEDZ model, the resistance is in parallel with the reactance formed by the parallel combination of the capacitor and inductor.

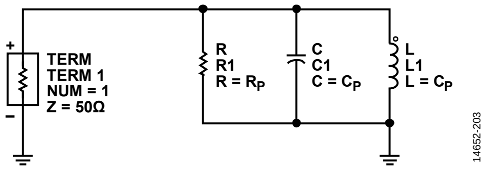


Figure 115. PEDZ Definition

Mathematically, the PEDZ model is represented as

$$PEDZ = R_P \| jX_P \tag{24}$$

The notation, a || b is short for a in parallel with b. The term, jX_P, can be positive (inductive) or negative (capacitive).

Conversions Between SEDZ and PEDZ

From a network theory perspective, the SEDZ model and the PEDZ model are equivalent. The following equations provide a means to calculate a PEDZ model from a SEDZ model. Note that SEDY is the series equivalent differential admittance, which is the reciprocal of SEDZ. Similarly, PEDY is the parallel equivalent differential admittance, which is the reciprocal of PEDZ.

$$R_P = \left[Re\left\{\frac{1}{SEDZ}\right\}\right]^{-1} = \left[Re\left\{\frac{1}{R_S + jX_S}\right\}\right]^{-1} \tag{25}$$

$$jX_P = -\left[Im\left\{\frac{1}{SEDZ}\right\}\right]^{-1} = -\left[Im\left\{\frac{1}{R_S + jX_S}\right\}\right]^{-1} \tag{26}$$

Conversion from and to the SEDZ model from the PEDZ model is accomplished using the following equations:

$$R_S = Re\left\{\frac{1}{\frac{1}{R_P} - \frac{j}{X_P}}\right\} \tag{27}$$

$$jX_S = Im\left\{\frac{1}{\frac{1}{R_P} - \frac{j}{X_P}}\right\} \tag{28}$$

Example 1 and Example 2 that follow illustrate the simplicity of the conversion process.

Example 1: SEDZ to PEDZ Conversion

Given SEDZ = 100 – j20 Ω, the PEDZ model calculation uses Equation 25 and Equation 26 as follows:

$$R_P = \left[Re\left\{\frac{1}{SEDZ}\right\}\right]^{-1} = \left[Re\left\{\frac{1}{R_S + jX_S}\right\}\right]^{-1} = [0.0096]^{-1} = 104\ \Omega \tag{29}$$

$$jX_P = -\left[Im\left\{\frac{1}{SEDZ}\right\}\right]^{-1} = -\left[Im\left\{\frac{1}{100 - j20}\right\}\right]^{-1} = -[0.0019]^{-1} = -j520\ \Omega \tag{30}$$

The preceding calculations show that PEDZ = R_P || jX_P = 104 || (−j520 Ω).

Example 2: PEDZ to SEDZ Conversion

Given PEDZ = 104 || −j520 Ω, the SEDZ model calculation uses Equation 27 and Equation 28 as follows:

$$R_S = Re\left\{\frac{1}{\frac{1}{R_P} - \frac{j}{X_P}}\right\} = Re\left\{\frac{1}{\frac{1}{104} - \frac{j}{-520}}\right\} = 100\ \Omega \qquad (31)$$

$$jX_S = Im\left\{\frac{1}{\frac{1}{R_P} - \frac{j}{X_P}}\right\} = Im\left\{\frac{1}{\frac{1}{104} - \frac{j}{-520}}\right\} = -j20\ \Omega \qquad (32)$$

The preceding calculations show SEDZ = $R_S + jX_S$ = 100 – j20 Ω.

Thus, Example 1 and Example 2 illustrate that conversion between the two models is straightforward.

The following two major factors must be kept in mind:

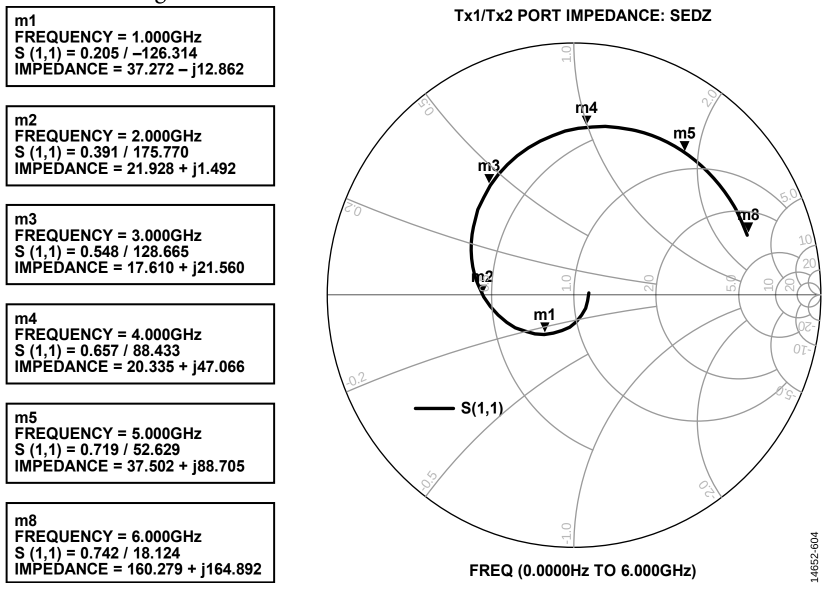
- The SEDZ model is equivalent to the PEDZ model. The conversion between the models is straightforward.
- Understanding the differential impedance models is critical when interpreting the data within this user guide.

RF PORT IMPEDANCE DATA

This section provides the port impedance data for all transmitters and receivers in the device. The following points of consideration are important for this section:

- Z_O is defined as 50 Ω.
- The reference plane for this data is the device ball pads.

Figure 116, Figure 118, Figure 120, Figure 122, Figure 124, Figure 126, Figure 128, and Figure 130 show the SEDZ vs. frequency in the Smith chart. Figure 117, Figure 119, Figure 121, Figure 123, Figure 125, Figure 127, Figure 129, and Figure 131 show PEDZ vs. frequency. The X STATUS parameter in the PEDZ plot indicates if the L or C PE parameter represents a capacitance measured in pF (X STATUS = 0) or an inductance measured in nH (X STATUS = 1).


m1
FREQUENCY = 1.000GHz
S (1,1) = 0.205 / –126.314
IMPEDANCE = 37.272 – j12.862

m2
FREQUENCY = 2.000GHz
S (1,1) = 0.391 / 175.770
IMPEDANCE = 21.928 + j1.492

m3
FREQUENCY = 3.000GHz
S (1,1) = 0.548 / 128.665
IMPEDANCE = 17.610 + j21.560

m4
FREQUENCY = 4.000GHz
S (1,1) = 0.657 / 88.433
IMPEDANCE = 20.335 + j47.066

m5
FREQUENCY = 5.000GHz
S (1,1) = 0.719 / 52.629
IMPEDANCE = 37.502 + j88.705

m8
FREQUENCY = 6.000GHz
S (1,1) = 0.742 / 18.124
IMPEDANCE = 160.279 + j164.892

Figure 116. Tx1 and Tx2 Series Equivalent Differential Port Impedance

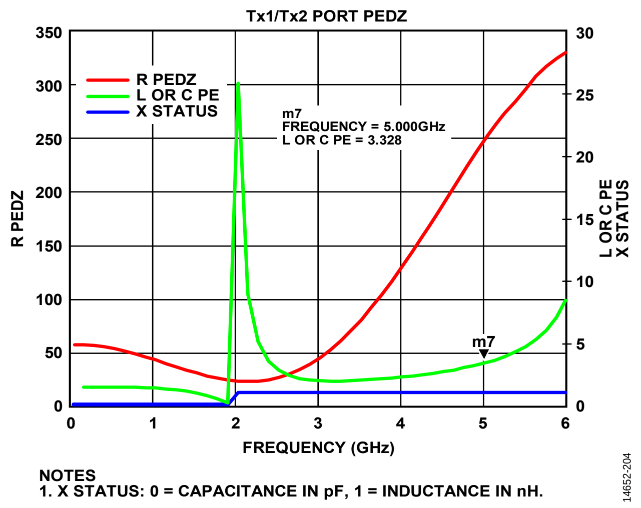


Figure 117. Tx1 and Tx2 Parallel Equivalent Differential Port Impedance

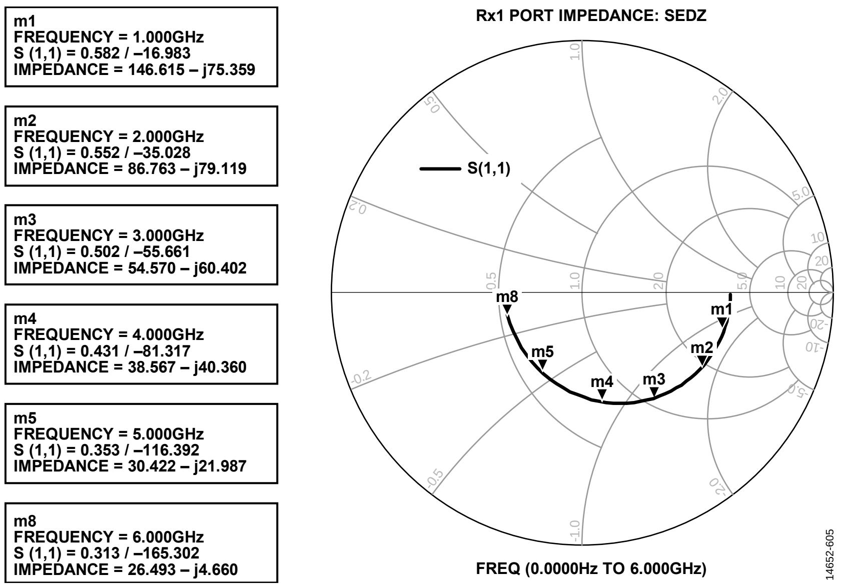


Figure 118. Rx1 Series Equivalent Differential Port Impedance

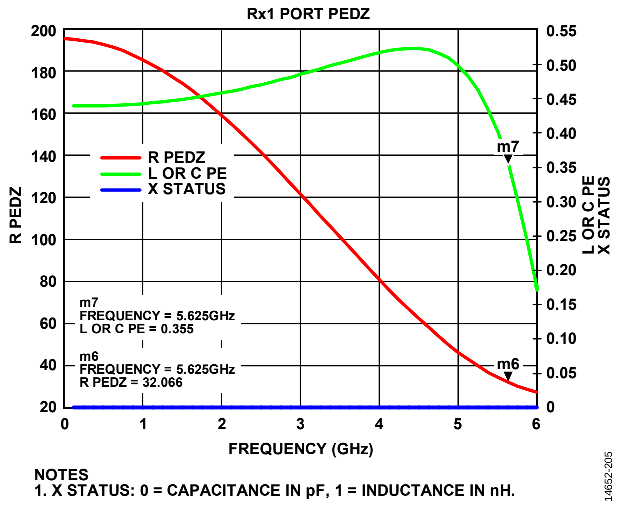
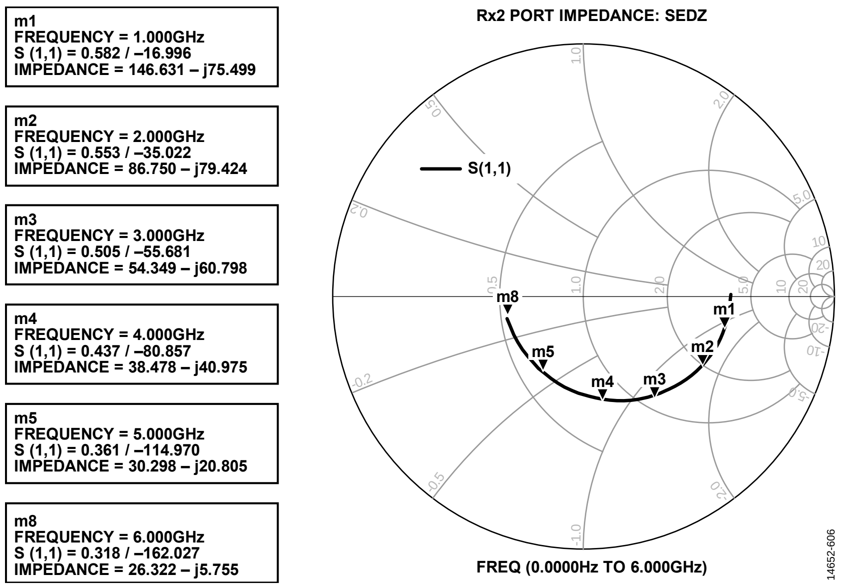


Figure 119. Rx1 Parallel Equivalent Differential Port Impedance



m1
FREQUENCY = 1.000GHz
S (1,1) = 0.582 / –16.996
IMPEDANCE = 146.631 – j75.499

m2
FREQUENCY = 2.000GHz
S (1,1) = 0.553 / –35.022
IMPEDANCE = 86.750 – j79.424

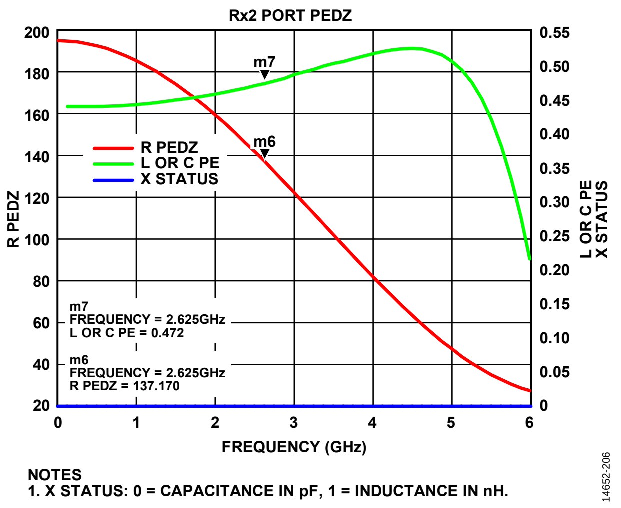
m3
FREQUENCY = 3.000GHz
S (1,1) = 0.505 / –55.681
IMPEDANCE = 54.349 – j60.798

m4
FREQUENCY = 4.000GHz
S (1,1) = 0.437 / –80.857
IMPEDANCE = 38.478 – j40.975

m5
FREQUENCY = 5.000GHz
S (1,1) = 0.361 / –114.970
IMPEDANCE = 30.298 – j20.805

m8
FREQUENCY = 6.000GHz
S (1,1) = 0.318 / –162.027
IMPEDANCE = 26.322 – j5.755

Figure 120. Rx2 Series Equivalent Differential Port Impedance



m7
FREQUENCY = 2.625GHz
L OR C PE = 0.472

m6
FREQUENCY = 2.625GHz
R PEDZ = 137.170

NOTES
1. X STATUS: 0 = CAPACITANCE IN pF, 1 = INDUCTANCE IN nH.

Figure 121. Rx2 Parallel Equivalent Differential Port Impedance

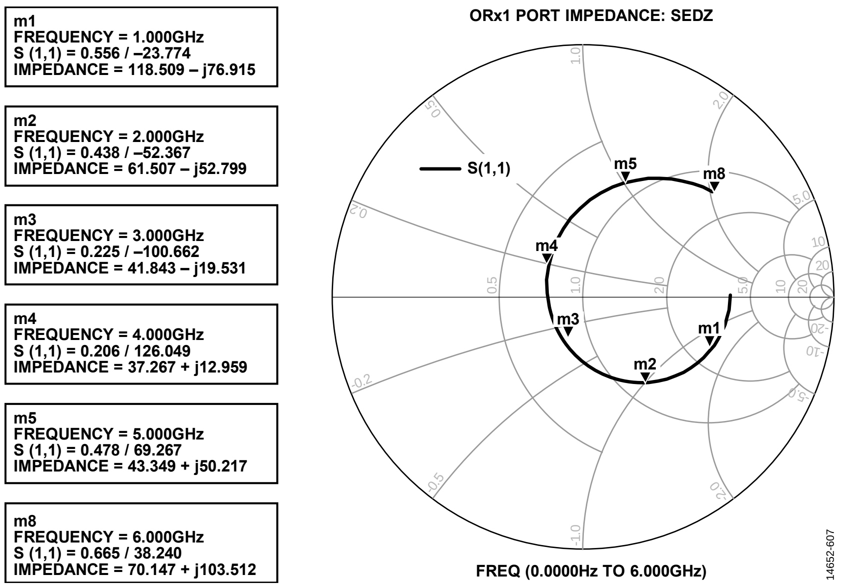


Figure 122. ORx1 Series Equivalent Differential Port Impedance

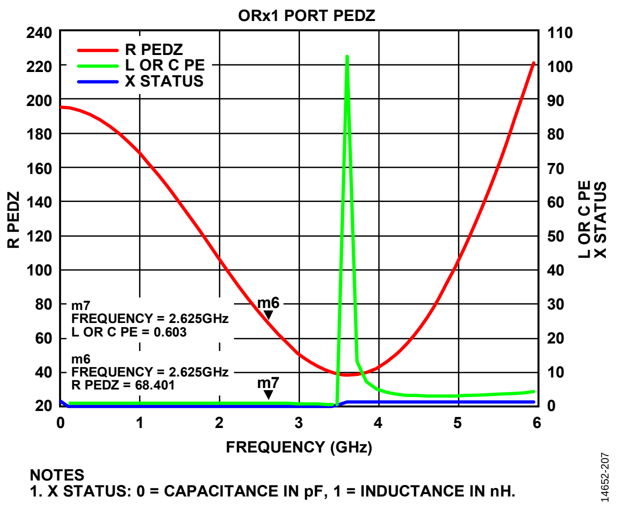


Figure 123. ORx1 Parallel Equivalent Differential Port Impedance

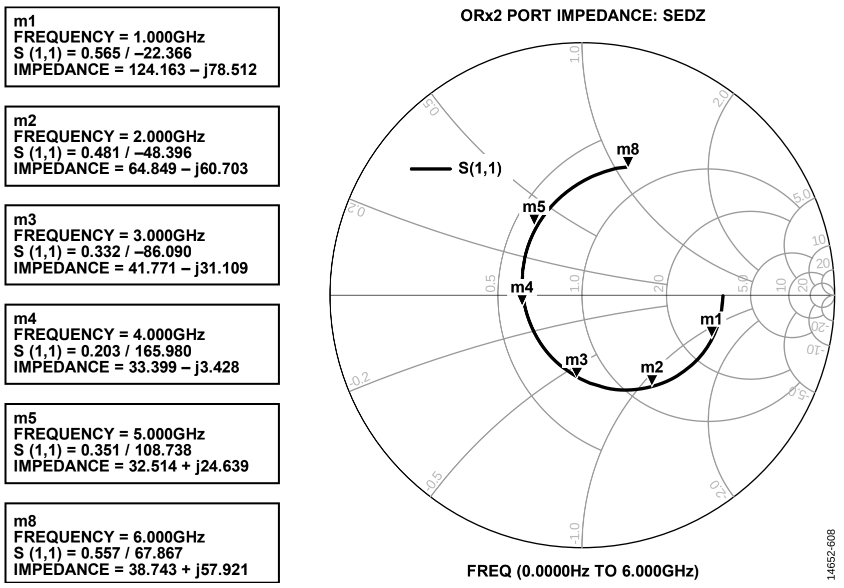


Figure 124. ORx2 Series Equivalent Differential Port Impedance

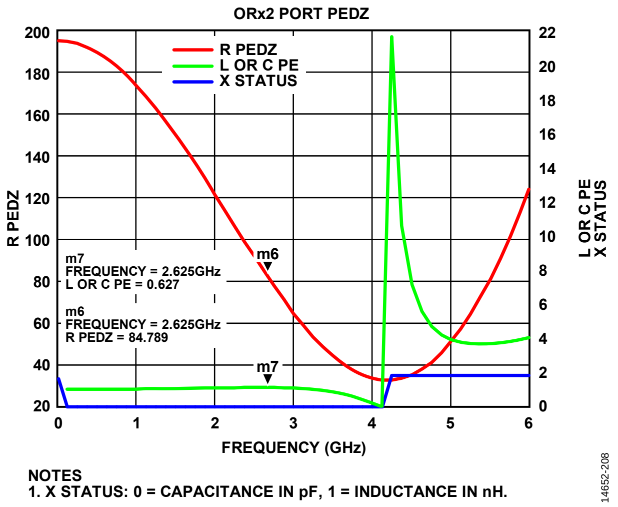


Figure 125. ORx2 Parallel Equivalent Differential Port Impedance

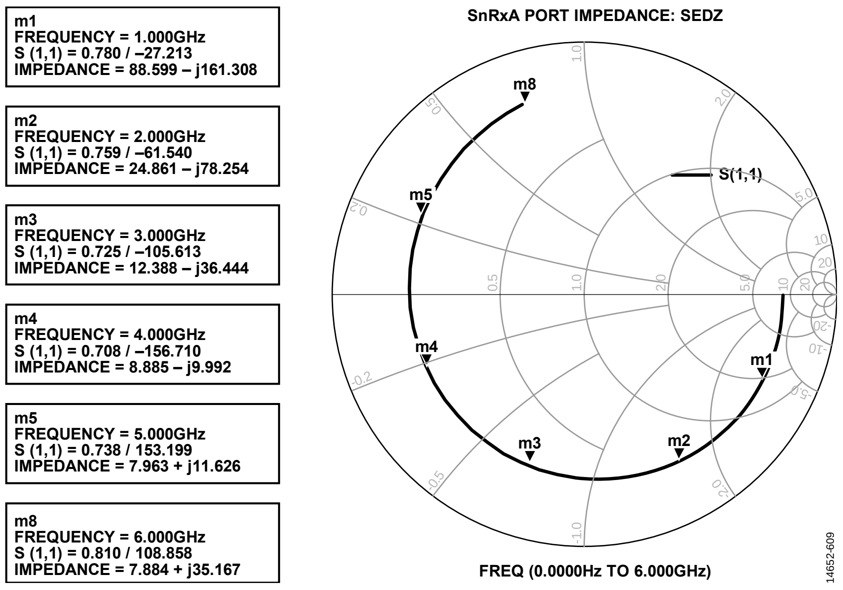


Figure 126. SnRxA Series Equivalent Differential Port Impedance

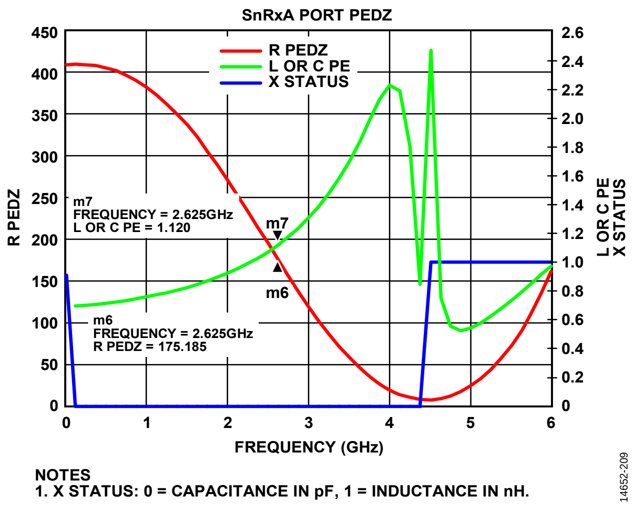


Figure 127. SnRxA Parallel Equivalent Differential Port Impedance

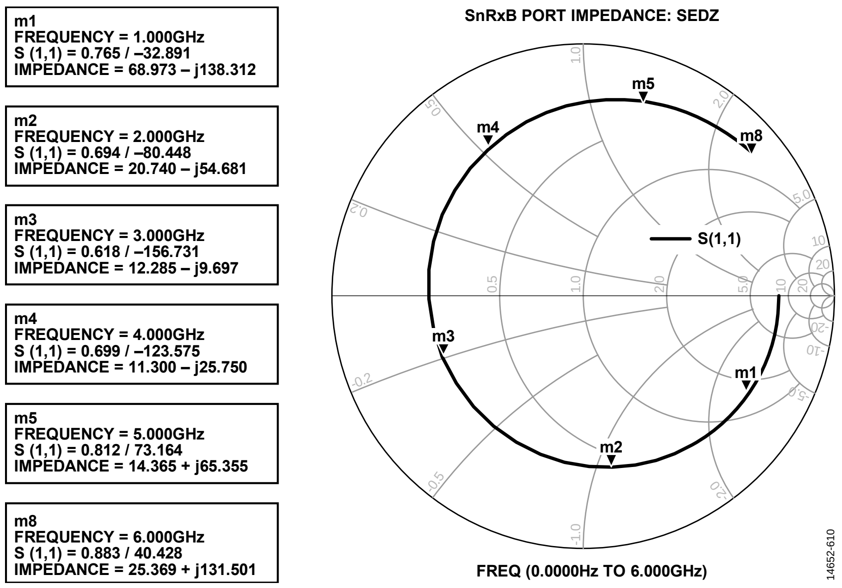


Figure 128. SnRxB Series Equivalent Differential Port Impedance

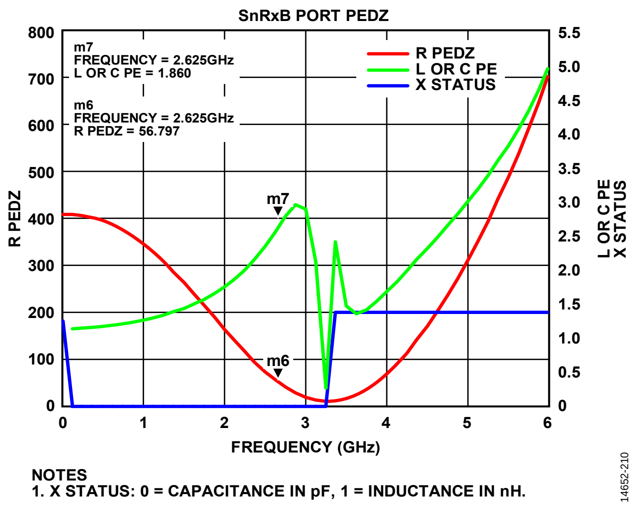


Figure 129. SnRxB Parallel Equivalent Differential Port Impedance

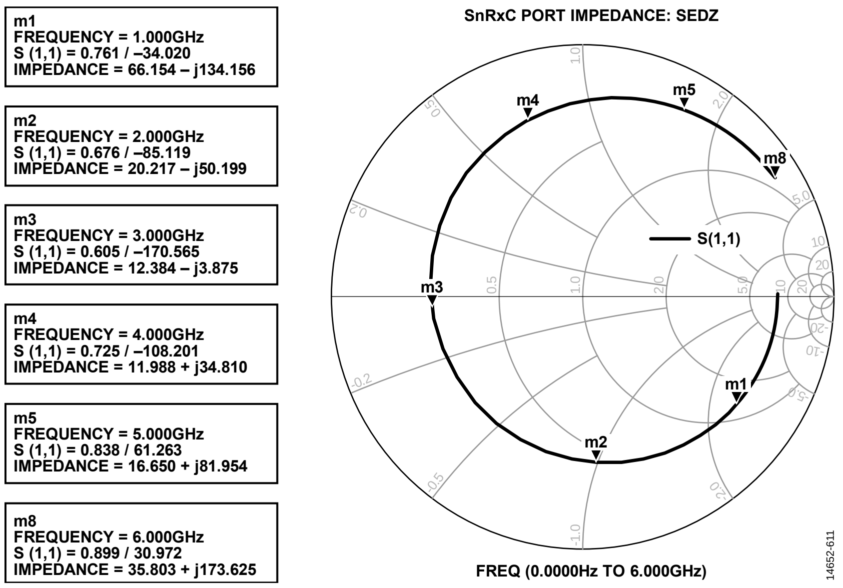


Figure 130. SnRxC Series Equivalent Differential Port Impedance (X STATUS Transitions from 0 to 1 at 3.1 GHz)

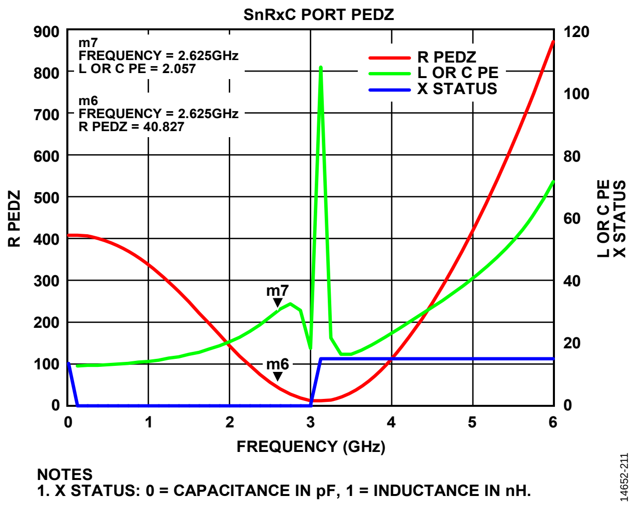


Figure 131. SnRxC Parallel Equivalent Differential Port Impedance (X STATUS Transitions from 0 to 1 at 3.1 GHz)

TRANSMITTER BIAS AND PORT INTERFACE

This section explains the dc biasing of the transmitter (Tx) outputs and how to interface to each Tx port. The transmitters operate over a wide range of frequencies. The Tx outputs are dc biased to a 1.8 V supply voltage using either RF chokes (wire wound inductors) or a transformer center tap connection. At full output power, each differential output side draws approximately 100 mA of dc bias current. For a differential Tx port application, the total Tx current consumption is approximately 400 mA.

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 shown in Figure 132 and Figure 133. The R_{DCR} resistors 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 (that is, $P_{O, 1dB}$, $P_{O, MAX}$, and so forth) to degrade. Select a high enough choke inductance (L_C) relative to the load impedance to avoid degrading the output power (see Figure 132).

The recommended dc bias network is shown in Figure 133. This network has fewer parasitics and fewer total components.

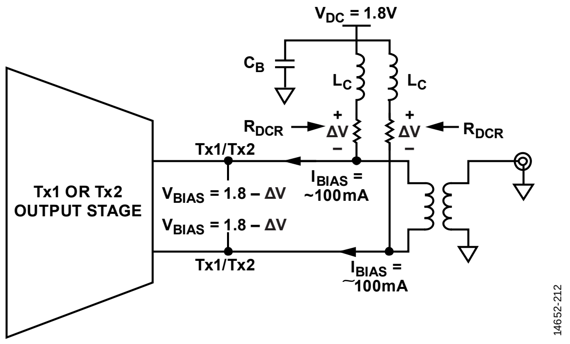


Figure 132. RF DC Bias Configuration: Parasitic Losses Due to Wire Wound Chokes

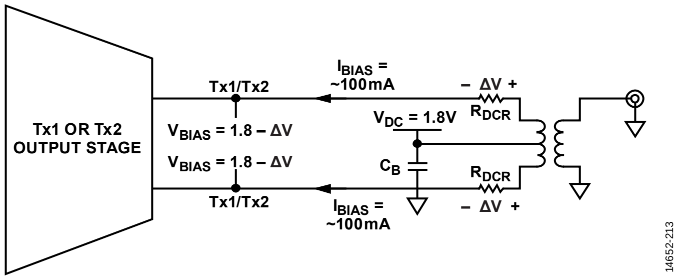


Figure 133. RF DC Bias Configuration: Parasitic Losses Due to Center Tapped Transformers

Figure 134 through Figure 137 identify four basic differential transmitter output configurations. Impedance matching networks (balun single-ended ports) are likely to be required to achieve optimum device performance from the device. In addition, the transmitter outputs must be ac-coupled in most applications due to the dc bias voltage applied to the differential output lines of the transmitter.

RF Interface Options

Figure 134 shows the recommended RF transmitter interface. It features a center tapped balun and offers the lowest component count of all of the options.

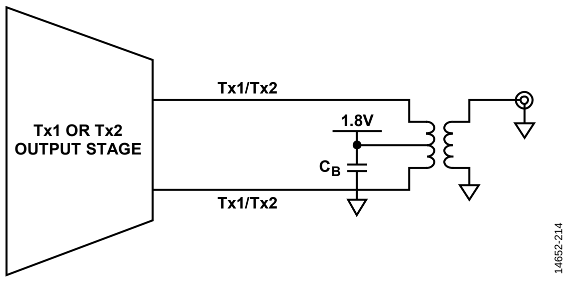


Figure 134. Recommended RF Transmitter Interface (Center Tapped Balun)

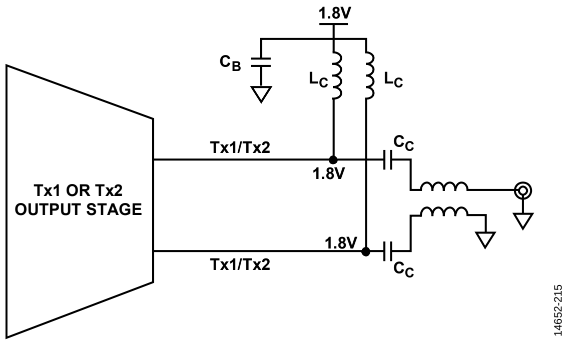


Figure 135. RF Transmitter Interface (RF Chokes Bias Differential Tx Output Lines with Additional Coupling Capacitors Creating a Transmission Line Balun)

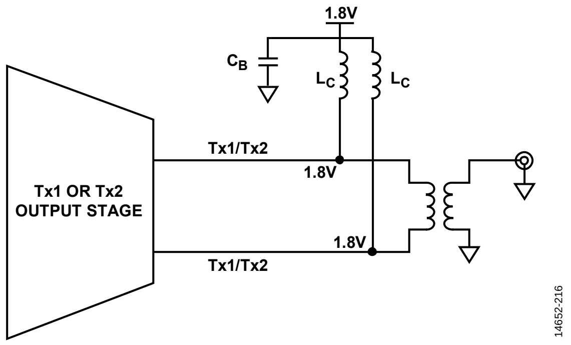


Figure 136. RF Transmitter Interface (RF Chokes Bias Differential Tx Output Lines and Connect to Transformer, No Additional Capacitors)

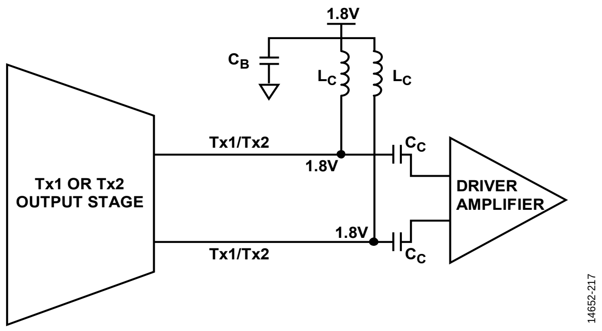


Figure 137. RF Transmitter Interface (RF Chokes Bias the Differential Output Lines That Are AC-Coupled Into the Input of a Driver Amplifier)

If a Tx balun is selected that requires a set of external dc bias chokes, careful planning is required. It is necessary to find the optimum compromise between the choke physical size, choke dc resistance (R_{DCR}), and the balun low frequency insertion loss. In commercially available dc bias chokes, resistance decreases as size increases. However, as choke inductance increases, resistance increases. Therefore, it is undesirable to use physically small chokes with high inductance because they exhibit the greatest resistance. For example, the voltage drop of 500 nH on a 0603 package size choke at 100 mA is roughly 50 mV.

Table 168. Sample Wire Wound DC Bias Choke Resistance vs. Inductance (0603 and 1206 Package Sizes)

	Resistance (Ω)	
Inductance (nH)	**Size 0603**	**Size 1206**
100	0.10	0.08
200	0.15	0.10
300	0.16	0.12
400	0.28	0.14
500	0.45	0.15
600	0.52	0.20

Transmitter Impedance Matching Network Design Methodology

The transmitter differential output port is viewed as a medium signal device. Therefore, impedance matching is based on load-pull style matching techniques as opposed to a small signal design used on the receivers, which is a similar methodology to power amplifier impedance matching. The goal is to provide a transmitter output differential load impedance that represents the best compromise between the maximum output power delivered (P_{OUT}) and the highest possible third-order linearity (P_{OIP3}).

Load-pull is a general term that defines the power delivered into a specific load impedance. Typically, this term is applied to systems where, at a given load impedance, the power delivered is limited by either the dc power supply voltage or the maximum current through the device. If enough sample points are taken, contours of delivered power (or other performance parameters such as P_{OIP3}) can be plotted on the Smith chart vs. load impedance and frequency.

Load-pull style matching is straightforward. Determine the frequency of interest and provide the load impedance that represents the desired compromise between the output power and linearity. The focus is on developing the preferred load impedance at the Tx output ball pads. A quick contrast between load-pull and small signal matching is instructive and follows:

- Load-pull: design the matching network for the preferred load impedance at the transmitter output pads.
- Small signal: design the matching network for the maximum power transfer based on the transducer gain.

From the empirical data, the preferred transmitter output differential load impedance is 50 Ω. Note the following:

- The reference plane is the transmitter output evaluation board ball pads.
- The fundamental power (P_{OUT}) is inversely proportional to the real portion of the load impedance.
- The output third-order intercept point (P_{OIP3}) is inversely proportional to the real portion of the load impedance.
- The P_{OIP3} is higher for capacitive loads compared to inductive loads. Therefore, if matching errors persist throughout the design, it is preferable to err to the side of capacitive rather than inductive to avoid voltage peaking effects.
- The optimum transmitter differential output load impedance is subject to change.

One negative issue associated with the load-pull style matching is that the transmitter port, S11, may degrade compared to the small signal matching technique. However, sometimes it is not possible to make a small signal style match, which is the case when the transmitter output impedance of the packaged device is capacitive. A conjugate match provides an inductive residual reactance; this is potentially harmful to device output third-order intercept (IP3) and harmonic distortion perspectives.

GENERAL RECEIVER PATH INTERFACE

The device has three types of receivers. These include two main receive pathways (Rx1 and Rx2), two observation receivers (ORx1 and ORx2), and three sniffer receivers (SnRxA, SnRxB, and SnRxC). The Rx path can support up to 100 MHz bandwidth, the ORx path can support up to 250 MHz bandwidth, and the SnRx path can support up to 20 MHz bandwidth. The ORx and Rx channels are designed for differential use only. The SnRx path supports both differential and single-ended usage, but differential configurations are recommended.

The receivers support a wide range of operation frequencies. In the case of the Rx and ORx channels, the differential signals interface to an integrated mixer. The mixer input pins have a dc bias of ~0.7 V and may need to be ac-coupled depending on the common-mode voltage level of the external circuit.

For the SnRx channel, the input pins interface directly to an integrated low noise amplifier (LNA). The LNA input pins have a dc bias of ~0.6 V. These inputs may need to be ac-coupled, depending on the common-mode voltage of the external circuit. To achieve best noise figure and even-order distortion (IP2) performance, use the SnRx ports in differential mode.

Important considerations for the receiver port interface are as follows:

- The device being interfaced to the transceiver. Options for this consideration include, but are not limited to, filters, baluns transmit/receive switches, external LNAs, and external power amplifiers (PAs). It is important to determine if the interfaced device presents a short to ground at dc.
- Rx and ORx maximum input power is 23 dBm (peak). The SnRx maximum safe input power is 2 dBm (peak).
- Rx and ORx optimum dc bias voltage is 0.7 V_{BIAS} to ground. The SnRx optimum dc bias voltage is 0.6 V_{BIAS} to ground.
- Board design: reference planes, transmission lines, impedance matching, including careful attention to low noise layout techniques, placement of transmission lines, and accurate impedance matching are essential for optimal performance.

Single-Ended and Differential Receiver Input Interface Circuits

Figure 138 through Figure 141 show possible single-ended and differential receiver port interface circuits. The options presented in Figure 138 and Figure 139 are valid only for the SnRx channels in single-ended mode. The options in Figure 140 and Figure 141 are valid for all receiver inputs operating in differential mode, though only the Rx1 signal names are indicated. Differential inputs with impedance matching may be necessary to obtain data sheet performance levels.

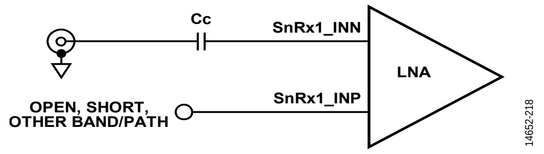


Figure 138. Single-Ended Input Interface Circuit, SnRx Only, Negative Side of Differential Input

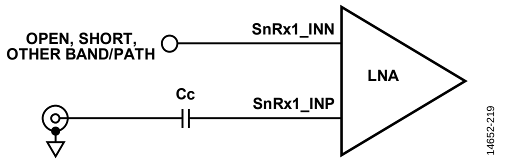


Figure 139. Single-Ended Input Interface Circuit, SnRx Only, Positive Side of Differential Input

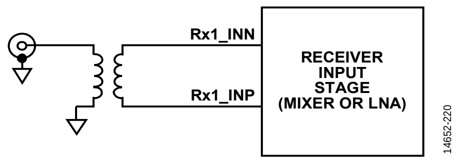


Figure 140. Differential Receiver Interface Using a Transformer, All Receiver Inputs

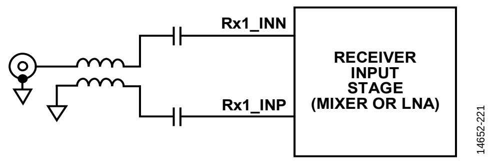


Figure 141. Differential Receiver Interface Using a Transmission Line Balun, All Receiver Inputs

Given wide RF bandwidth applications, surface-mount device (SMD) balun devices function well. Decent loss and differential balance are available in relatively small (0603, 0805) packages.

For receiver applications, the transmission line balun referenced in Figure 141 may be configured in multiple ways. Most configurations are based on a Marchand planer design derivative like the one shown in Figure 142.

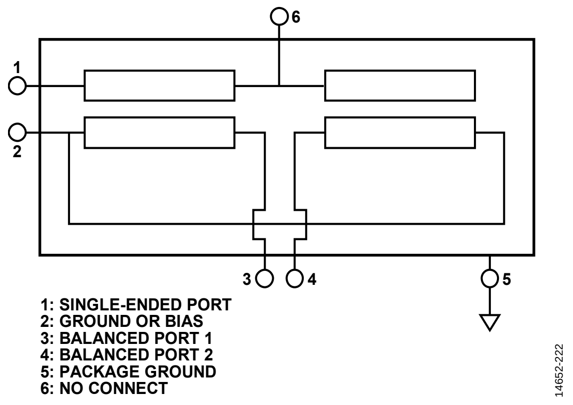


Figure 142. Marchand Planer Balun Schematic

The termination applied to the balun, Pin 2 (ground or bias), may be implemented in at least two ways. For applications where a dc short to ground (through the balun) is tolerated at the differential ports, connecting the Balun Pin 2 pad to ground is the best approach. However, if the application does not allow a short to ground, termination of Balun Pin 2 with a decoupling capacitor creates a simultaneous dc open and RF short. The decoupling capacitor value may be tuned to set the RF bandwidth low frequency corner.

If the impedance matching network component count or layout size is a critical parameter, a good choice is to terminate Balun Pin 2 with a decoupling capacitor.

General Receiver Impedance Matching Network Design Methodology

The device application determines the best receiver input port impedance matching methodology. The recommendations within this section are general-purpose only.

Low Noise Matching Network Design

If noise figure is a critical parameter in the application, low loss impedance matching between the receiver ports and the rest of the system is required.

A low loss generic impedance matching topology is defined in Figure 143. Because this topology is generic, it may be simplified by removing unused SMD component pads to save board layout area. The receiver input ports exhibit a dc bias voltage to ground. Avoid a dc short to ground on the input pins.

Note that in Figure 143, the single-ended match is a Π topology. The differential side shows a differential T network in the small box on the left and a dc block PI network in the large box on the right.

Given a three port device, such as a balun, both single-ended impedance matching and differential impedance matching may be required to obtain the lowest possible system power transmission loss. Implementation of an impedance matching network on only one side rarely results in optimum small signal power transfer.

In terms of design methodology, optimum impedance matching network performance is not guaranteed by monitoring only S11, the single-ended port parameter. Baluns and filters exhibit dissipative loss. When the dissipative loss is severe, it prevents the S11 single-ended side measurement from detecting differential side impedance matching issues.

For single-ended impedance matching, the PI topology is the most flexible option. Referring to Figure 143, the S1P7, S1P8, and S1P9 blocks form the PI shape. It is possible to reduce this network to an L topology by eliminating one of the shunt components (S1P7 or S1P9).

For differential side impedance matching, the network may be implemented as a T network (see Figure 143) or a dc block PI network (see Figure 143). Realizing that the single-ended T shape is formed by S1P2, S1P4, and S1P5, the differential T is formed by inclusion of the S1P3 and S1P6 blocks. The differential T is horizontally symmetrical.

Similarly, the components S1P1, S1P2, and S1P4 form a single-ended PI network. The differential PI network is formed by the inclusion of the S1P3 blocks. The addition of the S1P5 and S1P6 blocks complete the topology to form a dc block PI differential impedance matching network.

Both the T and dc block PI impedance matching topologies can be reduced to L networks, if desired. The L network may enable the widest possible impedance matching with the lowest number of SMD matching components.

A summary of the differential T and dc blocked differential PI topology is listed in Table 169.

If the three port device does not exhibit a dc short from each differential side to ground, the dc block PI topology may be simplified to a standard differential PI topology (removing components S1P5 and S1P6). This change results in reduced SMD component count (four) and reduced board layout area.

Note that many devices within the system such as baluns, filters, or switches heavily influence the optimum impedance matching topology. It is best to simulate the impedance matching options and then implement the best overall solution.

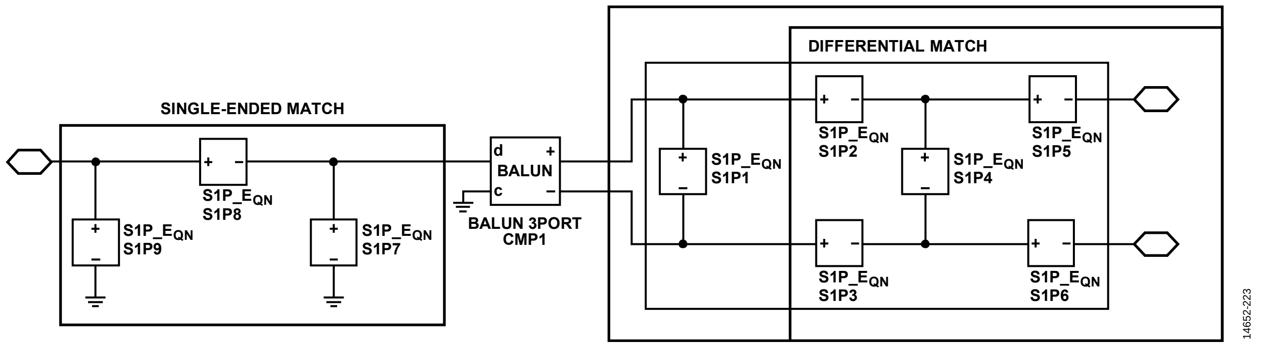


Figure 143. Generic Single-Ended to Differential Matching Topology

Table 169. Topology Advantage/Disadvantage Summary

Topology	Advantage	Disadvantage
Differential T	Lower SMD component count (5)	Impedance matching bandwidth may be up to 5% smaller than dc blocked differential PI topology
DC Blocked Differential PI	Relatively wide impedance matching bandwidth and impedance matching topology implementation flexibility	Higher SMD component count (6)

Simplified Matching Design

For applications that use an external low noise amplifier (LNA) before the input receiver ports, the device noise figure may not be very important. An external LNA allows a simplified impedance matching topology to potentially achieve wider RF bandwidth. From an RF bandwidth viewpoint, the limiting factor becomes the three-port device. In certain cases, RF bandwidths approaching 1.5 GHz are possible. Figure 144 illustrates the simplified schematic.

The impedance matching network in Figure 144 adds some insertion loss to the channel. However, in applications that utilize an external LNA before the receiver input ports, the loss associated with this simple impedance matching network may not be significant to system performance. When this network is implemented, the noise figure of the device is expected to degrade somewhat with respect to an application that utilizes a low loss impedance matching methodology. However, careful LNA selection mitigates such differences in overall system noise figure.

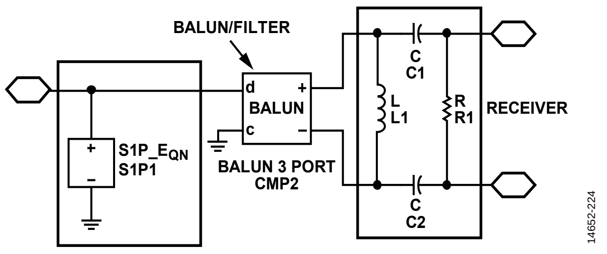


Figure 144. Simplified Impedance Matching Network

Note the following regarding the proposed topology in Figure 144:

- The R1 SMD component sets the real portion of the impedance set by the balun or filter.
- L1 resonates with the receiver input port capacitance, which results in the highest possible RF bandwidth.
- C1 and C2 are dc block capacitors. Depending on balun/filter selection, these SMD components may be replaced with 0 Ω resistors (that is, if the differential output lines do not exhibit a dc short to ground).
- The shunt SMD device (S1P1) on the single-ended port of the balun/filter is intended to resonate with either the inductance or capacitance seen at the balun/filter input port. This component may not be necessary for particular applications.

IMPEDANCE MATCHING NETWORK

Impedance matching networks are required to achieve the performance levels that are noted in the data sheet. This section provides example topologies and components used on the evaluation boards.

The S parameter models of the devices, board, balun, and SMD components are required to build an accurate system level simulation. The board layout model may be obtained from an electromagnetic momentum simulator. The balun and SMD component models may be obtained from the balun and SMD vendors or built from empirical data.

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 impedance matching networks and the component designators in the following diagrams are specific to the ADRV9371-N/PCBZ evaluation board. The board revision is indicated on the silkscreen under the RadioVerse™ logo.

The schematics show three elements in parallel; however, only one set of SMD component pads are placed on the board. For example, R201, L201, and C201 in Figure 145 have only one set of SMD pads for one SMD component. The schematic shows that in a generic port impedance matching network, the shunt or series elements may be a resistor, inductor, or a capacitor. Figure 145 through Figure 149 show the schematic blocks for the Rx1, Rx2, SnRxA, ORx1, and ORx2 channels, respectively.

In the transmitter (Tx) matching networks shown in Figure 150 and Figure 151, the C307, L307, L308, and C308 components for Tx1 (see Figure 150) and the C315, L315, L316, and C316 for components for Tx2 (see Figure 151) form dc bias feeds into the differential port of the transmitter. For baluns that do not supply dc to the differential side of the balun, this is an example of an external feed topology that can be used to supply proper voltage to the Tx output. For the matching networks listed in Table 175 and Table 176, these components are all DNI.

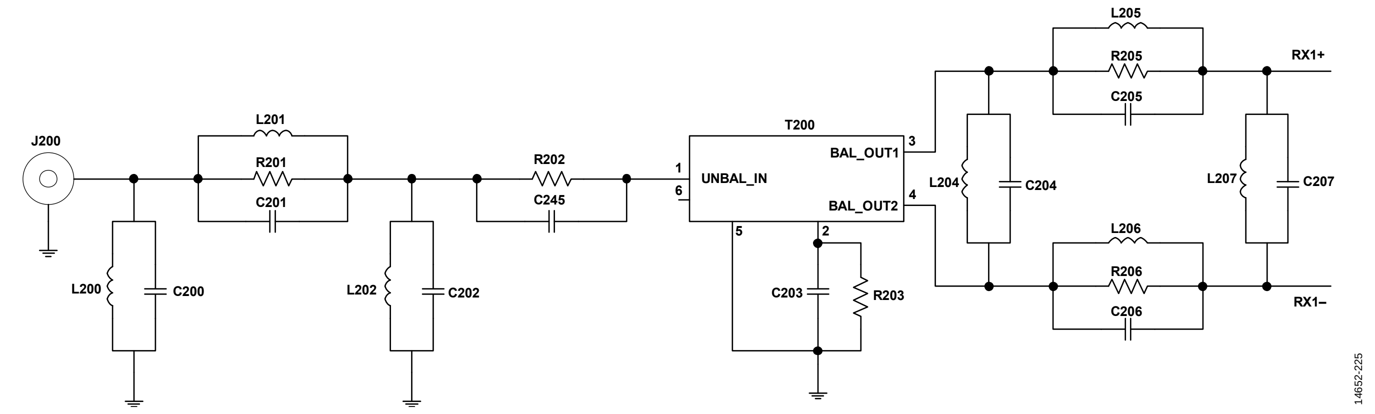


Figure 145. Rx1 Generic Matching Network Topology from the ADRV9371-N/PCBZ Evaluation Board

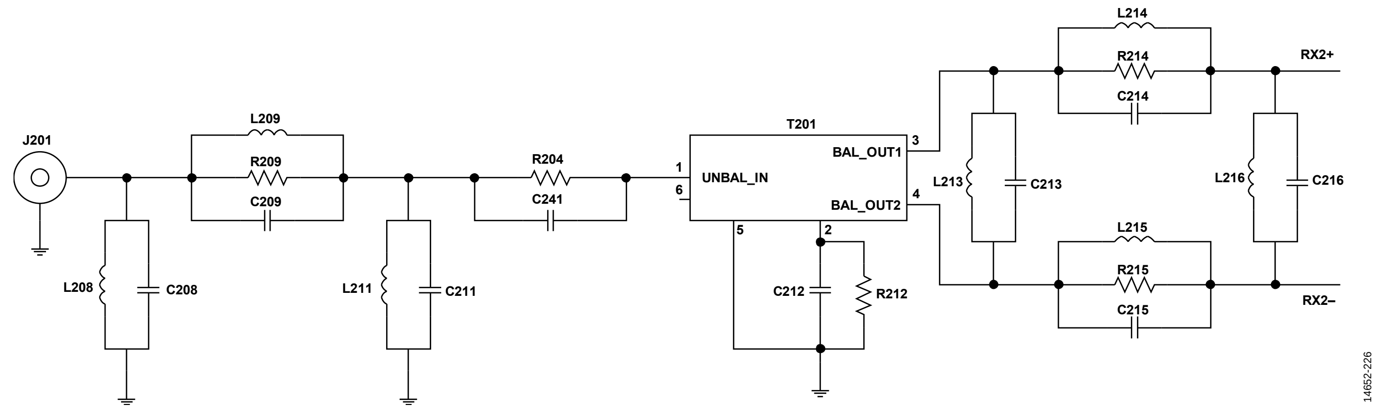


Figure 146. Rx2 Generic Matching Network Topology from the ADRV9371-N/PCBZ Evaluation Board

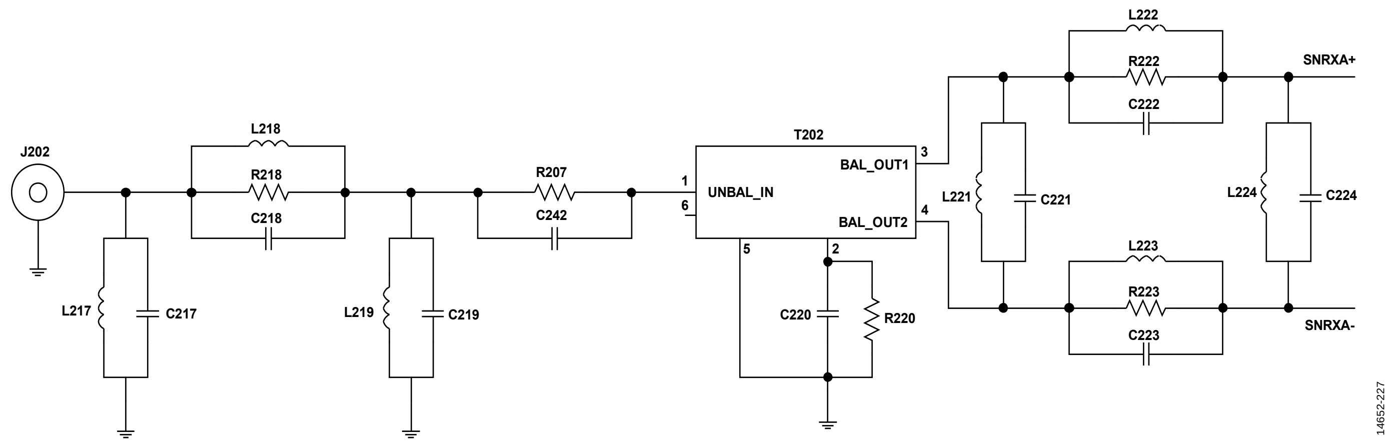


Figure 147. SnRxA Generic Matching Network Topology from ADRV9371-N/PCBZ Evaluation Board

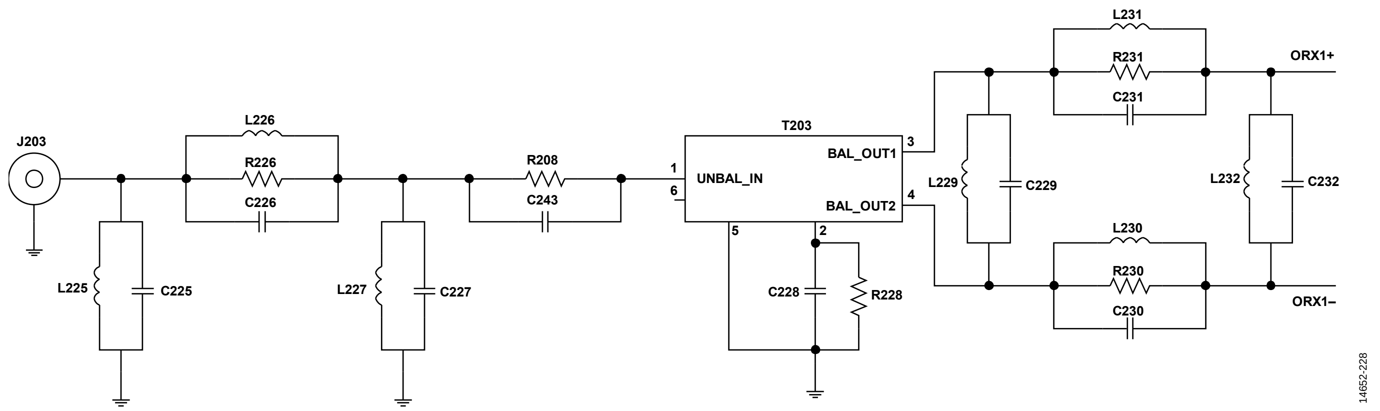


Figure 148. ORx1 Generic Matching Network Topology from the ADRV9371-N/PCBZ Evaluation Board

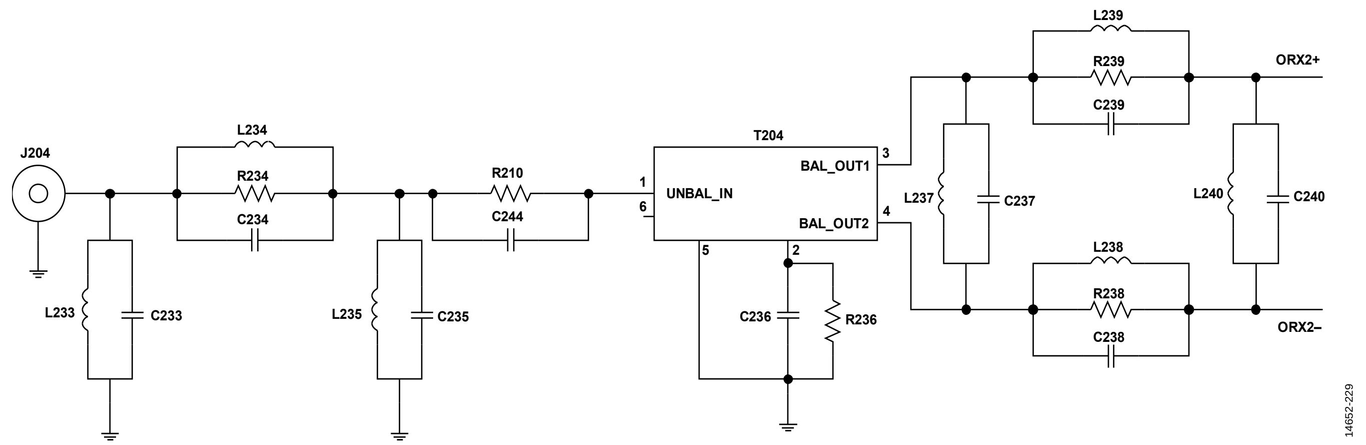


Figure 149. ORx2 Generic Matching Network Topology from the ADRV9371-N/PCBZ Evaluation Board

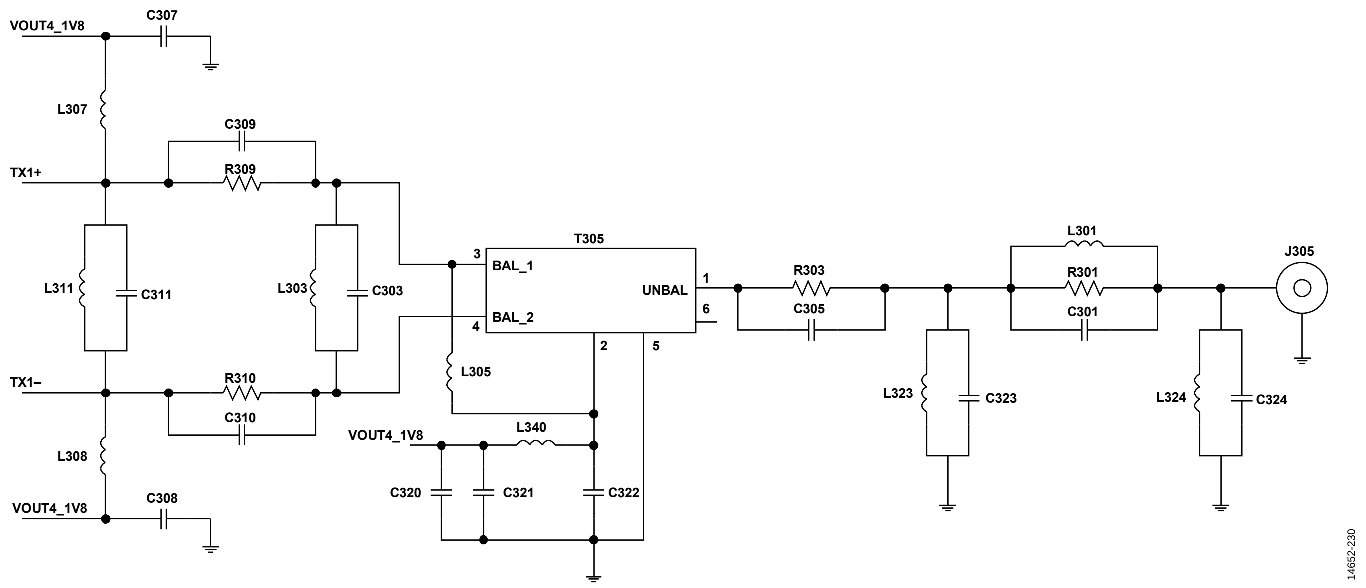


Figure 150. Tx1 Generic Matching Network Topology from ADRV9371-N/PCBZ Evaluation Board

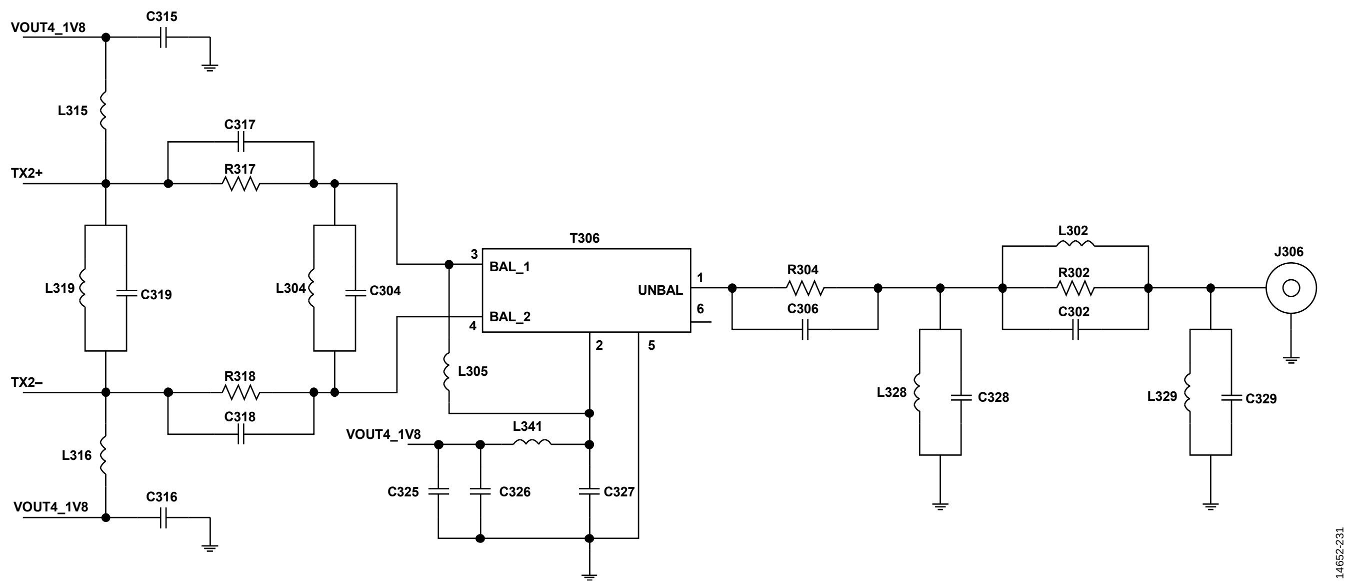


Figure 151. Tx2 Generic Matching Network Topology from ADRV9371-N/PCBZ Evaluation Board

Selected Balun and Component Values

Table 170 through Table 176 show the selected balun and component values used in theADRV9371-N/PCBZ evaluation board for four matching network sets. DNI stands for do not install (leave open). Component tolerances are also indicated. Note that all tolerances are at ±5%, unless noted in parentheses. Tolerance notations are either in percent (%) or units.

The Suffix column in Table 170 through Table 176 indicates the reference used to describe each matching network frequency band. Note that circuit values for the SnRxB and SnRxC inputs are not included here because the ADRV9371-N/PCBZ evaluation board only provides one sniffer receiver input.

Table 170. Main Receiver Rx1

		Component Location on PCB (All Tolerances at ±5% Unless Noted)									
Frequency Band (MHz)	**Suffix (G)**	**C200, L200**	**C201, L201, R201**	**C202, L202**	**C245, R202**	**C203, R203**	**C204, L204**	**C205, L205, R205**	**C206, L206, R206**	**C207, L207**	**T200**
300 to 1000	−0.7	DNI	10 pF (±1%)	27 nH (±3%)	220 pF	180 pF	1.0 pF (±0.1 pF)	9.1 nH (±3%)	9.1 nH (±3%)	0.6 pF (±0.1 pF)	Anaren B0310J50100AHF
1800 to 2800	−2.6	10 nH	0 Ω	DNI	0 Ω	0 Ω	0.5 pF (±0.1 pF)	100 pF	100 pF	3.9 nH	Anaren BD0826J50200AHF
3300 to 3800	−3.5	DNI	0 Ω	0.2 pF (±0.1 pF)	0 Ω	0 Ω	0.3 pF (±0.1 pF)	10 pF	10 pF	0.4 pF (±0.1 pF)	Johanson 3700BL15B050
5300 to 5900	−5.5	0.3 pF (±0.1 pF)	1.2 nH (±0.1 nH)	DNI	0 Ω	0 Ω	DNI	0.4 pF (±0.1 pF)	0.4 pF (±0.1 pF)	5.1 nH	Johanson 5400BL15B200

Table 171. Main Receiver Rx2

		Component Location on PCB (All Tolerances at ±5% Unless Noted)									
Frequency Band (MHz)	**Suffix (G)**	**C208, L208**	**C209, L209, R209**	**C211, L211**	**C241, R204**	**C212, R212**	**C213, L213**	**C214, L214, R214**	**C215, L215, R215**	**C216, L216**	**T201**
300 to 1000	−0.7	DNI	10 pF (±1%)	27 nH (±3%)	220 pF	180 pF	1.0 pF (±0.1 pF)	9.1 nH (±3%)	9.1 nH (±3%)	0.6 pF (±0.1 pF)	Anaren B0310J50100AHF
1800 to 2800	−2.6	10 nH	0 Ω	DNI	0 Ω	0 Ω	0.5 pF (±0.1 pF)	100 pF	100 pF	3.9 nH	Anaren BD0826J50200AHF
3300 to 3800	−3.5	DNI	0 Ω	0.2 pF (±0.1 pF)	0 Ω	0 Ω	0.3 pF (±0.1 pF)	10 pF	10 pF	0.4 pF (±0.1 pF)	Johanson 3700BL15B050
5300 to 5900	−5.5	0.3 pF (±0.1 pF)	1.2 nH (±0.1 nH)	DNI	0 Ω	0 Ω	DNI	0.4 pF (±0.1 pF)	0.4 pF (±0.1 μF)	5.1 nH	Johanson 5400BL15B200

Table 172. Observation Receiver ORx1

		Component Location on PCB (All Tolerances at ±5% Unless Noted)									
Frequency Band (MHz)	**Suffix (G)**	**C225, L225**	**C226, L226, R226**	**C227, L227**	**C243, R208**	**C228, R228**	**C229, L229**	**C230, L230, R230**	**C231, L231, R231**	**C232, L232**	**T203**
300 to 1000	−0.7	DNI	15 pF (±2%)	33 nH (±3%)	220 pF	180 pF	1.0 pF (±0.1 pF)	8.2 nH (±3%)	8.2 nH (±3%)	0.4 pF (±0.1 pF)	Anaren B0310J50100AHF
1800 to 2800	−2.6	27 nH	5.6 pF (±0.1 pF)	DNI	0 Ω	0 Ω	6.2 nH	6.0 pF (±0.1 pF)	6.0 pF (±0.1 pF)	1.0 pF (±0.1 pF)	Anaren B0322J5050AHF
3300 to 3800	−3.5	DNI	0 Ω	DNI	0 Ω	0 Ω	DNI	2.0 pF (±0.1 pF)	2.0 pF (±0.1 pF)	7.5 nH (±0.1 nH)	Johanson 3700BL15B200
5300 to 5900	−5.5	10 nH (±0.1 nH)	100 pF	4.7 nH (±3%)	0 Ω	0 Ω	DNI	0.6 pF (±0.1 pF)	0.6 pF (±0.1 pF)	1.5 nH	Johanson 5400BL15B200

Table 173. Observation Receiver ORx2

Frequency Band (MHz)	Suffix (G)	Component Location on PCB (All Tolerances at ±5% Unless Noted)									
		C233, L233	C234, L234, R234	C235, L235	C244, R210	C236, R236	C237, L237	C238, L238, R238	C239, L239, R239	C240, L240	T204
300 to 1000	−0.7	DNI	15 pF (±2%)	33 nH (±3%)	220 pF	180 pF	1.0 pF (±0.1 pF)	8.2 nH (±3%)	8.2 nH (±3%)	0.4 pF (±0.1 pF)	Anaren B0310J50100AHF
1800 to 2800	−2.6	27 nH	5.6 pF	DNI	0 Ω	0 Ω	6.2 nH	6.0 pF (±0.1 pF)	6.0 pF (±0.1 pF)	1.0 pF (±0.1 pF)	Anaren B0322J5050AHF
3300 to 3800	−3.5	DNI	0 Ω	DNI	0 Ω	0 Ω	DNI	2.0 pF (±0.1 pF)	2.0 pF (±0.1 pF)	7.5 nH (±0.1 nH)	Johanson 3700BL15B200
5300 to 5900	−5.5	DNI	100 pF	2.7 nH (±0.1 nH)	0 Ω	0 Ω	DNI	0.6 pF (±0.1 pF)	0.6 pF (±0.1 pF)	1.8 nH	Johanson 5400BL15B200

Table 174. Sniffer Receiver SnRxA

Frequency Band (MHz)	Suffix (G)	Component Location on PCB (All Tolerances at ±5% Unless Noted)									
		C217, L217	C218, L218, R218	C219, L219	C242, R207	C220, R220	C221, L221	C222, L222, R222	C223, L223, R223	C224, L224	T202
300 to 1000	−0.7	75 nH (±3%)	15 nH (±3%)	0.7 pF (±0.1 pF)	220 pF	180 pF	0.9 pF (±0.1 pF)	27 nH (±3%)	27 nH (±3%)	DNI	Anaren B0310J50100AHF
1800 to 2800	−2.6	0.5 pF (±0.1 pF)	22 pF (±0.1 pF)	3.0 nH (±0.1 nH)	0 Ω	100 pF	DNI	5.6 nH	5.6 nH	0.6 pF (±0.05 pF)	Johanson 2450BL15B200E
3300 to 3800	−3.5	100 nH	1.8 pF (±0.1 pF)	100 nH	0 Ω	8 pF	DNI	2.5 nH (±0.1 nH)	2.5 nH (±0.1 nH)	4.7 nH (±0.1 nH)	Johanson 3700BL15B050
5300 to 5900	−5.5	0.5 pF (±0.05 pF)	1.0 nH (±0.1 nH)	0.3 pF (±0.1 pF)	0 Ω	0 Ω	3.6 nH (±0.1 nH)	0.2 pF (±0.1 pF)	0.2 pF (±0.1 pF)	8.2 nH	Johanson 5400BL15B200

Table 175. Transmitter Tx1

Frequency Band (MHz)	Suffix (G)	Component Location on PCB (All Tolerances at ±5% Unless Noted)											
		C311, L311	C309, R309,	C310, R310	C303, L303	L305	C305, R303	C323, L323	C301, L301, R301	C324, L324	L340	C322	T305
300 to 1000	−0.7	DNI	0 Ω	0 Ω	DNI	39 nH	0 Ω	1.0 pF (±0.1 pF)	5.6 nH (±3%)	DNI	27 nH	180 pF	Anaren B0322J5050AHF
1800 to 2800	−2.6	1 pF (±0.1 pF)	0 Ω	0 Ω	DNI	DNI	0 Ω	DNI	1.5 nH	0.75 pF (±0.1 pF)	27 nH	0.1 µF	Mini-Circuits NCS1-292+
3300 to 3800	−3.5	0.6 pF (±0.1 pF)	0 Ω	0 Ω	DNI	DNI	0 Ω	DNI	0 Ω	DNI	27 nH	0.1 µF	Johanson 3700BL15B100
5300 to 5900	−5.5	5.1 nH (±3%)	1.5 nH (±1 nH)	1.5 nH (±1 nH)	5.1 nH (±3%)	DNI	0 Ω	DNI	0 Ω	DNI	0 Ω	1.2 pF (±0.1 pF)	TDK HHM1752A2

Table 176. Transmitter Tx2

Frequency Band (MHz)	Suffix (G)	Component Location on PCB (All Tolerances at ±5% Unless Noted)											
		C319, L319	C317, R317,	C318, R318	C304, L304	L306	C306, R304	C328, L328	C302, L302, R302	C329, L329	L341	C327	T306
300 to 1000	−0.7	DNI	0 Ω	0 Ω	DNI	39 nH	0 Ω	1.0 pF (±0.1 pF)	5.6 nH (±3%)	DNI	27 nH	180 pF	Anaren B0322J5050AHF
1800 to 2800	−2.6	1 pF (±0.1 pF)	0 Ω	0 Ω	DNI	DNI	0 Ω	DNI	1.5 nH	0.75 pF (±0.1 pF)	27 nH	0.1 µF	Mini-Circuits NCS1-292+
3300 to 3800	−3.5	0.6 pF (±0.1 pF)	0 Ω	0 Ω	DNI	DNI	0 Ω	DNI	0 Ω	DNI	27 nH	0.1 µF	Johanson 3700BL15B100
5300 to 5900	−5.5	5.1 nH (±3%)	1.5 nH (±1 nH)	1.5 nH (±1 nH)	5.1 nH (±3%)	DNI	0 Ω	DNI	0 Ω	DNI	0 Ω	1.2 pF (±0.1 pF)	TDK HHM1752A2

Mykonos Tx1 and Tx2 Port Impedance

The equivalent parallel equivalent differential impedances and the corresponding S11 values for the Tx1 and Tx2 output ports are shown in Table 177 for a series of operating frequencies ranging from 1.000 GHz to 6.000 GHz. The relationship between these points is illustrated in the Smith chart in Figure 152 and the frequency response plot in Figure 153.

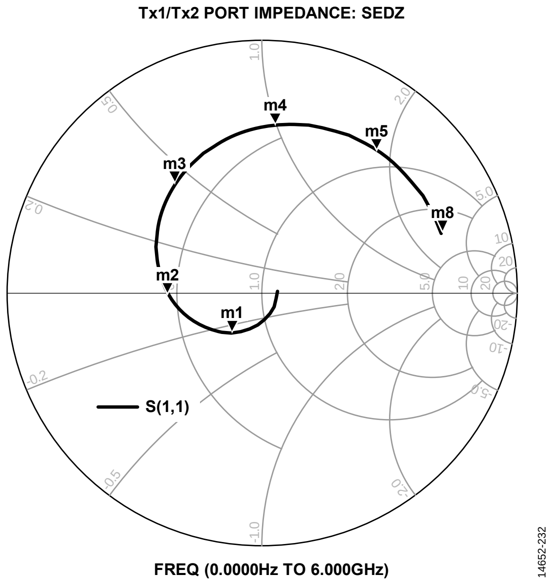


Figure 152. SEDZ vs. Frequency, Tx1 and Tx2 Port Impedance

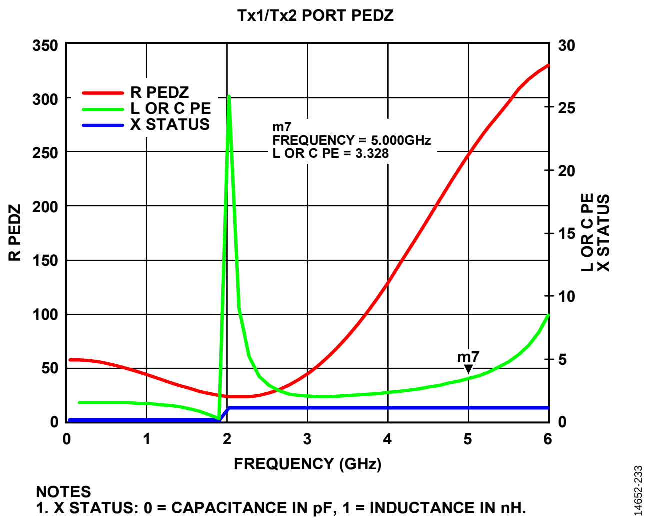


Figure 153. PEDZ vs. Frequency, Tx1 and Tx2 Ports

BOARD LAYOUT DESIGN RECOMMENDATIONS

Circuit board layout is a critical part of the impedance matching process. Design trade-offs are often required to balance material cost, performance, and bandwidth. Refer to the RF and JESD204B Transmission Line Layout section for details on how to design the circuit board traces that make matching circuits more accurate.

Table 177. PEDZ vs. Frequency

Parameter	M1	M2	M3	M4	M5	M8
Frequency (GHz)	1.000	2.000	3.000	4.000	5.000	6.000
S11	+0.205/−126.314	0.391/175.770	0.548/128.665	0.657/88.433	0.719/52.629	0.742/18.124
Impedance (Ω)	37.272 − j12.862	21.298 + j1.492	17.610 + j21.560	30.335 + j47.066	37.502 + j88.705	160.279 + j164.892

PRINTED CIRCUIT BOARD LAYOUT GUIDELINES

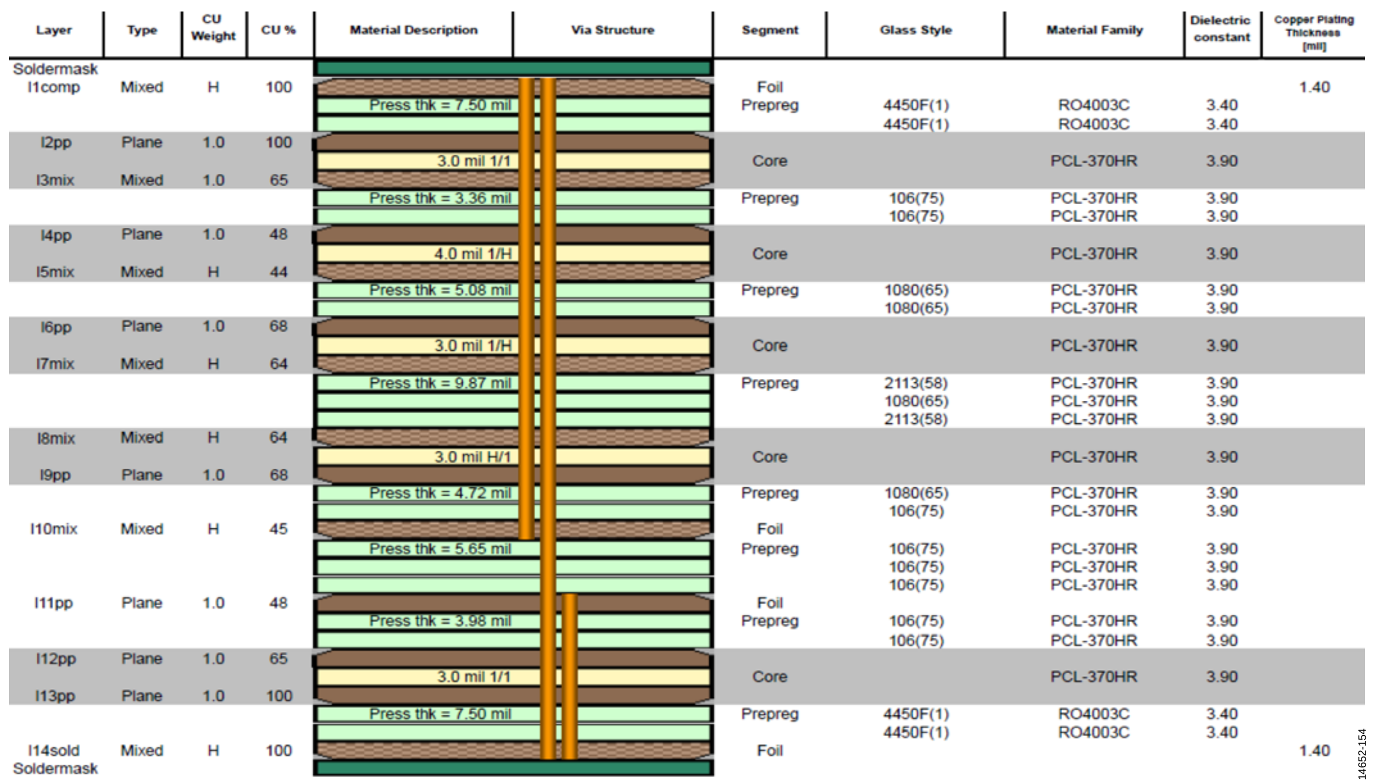
Because of the integration complexity of the device and its high pin count, careful printed circuit board (PCB) layout is important to optimize performance. This section provides a checklist of issues to look for and guidelines on how to optimize the PCB to mitigate performance issues. The goal of this document is to help achieve the best possible performance from the device while reducing board layout effort. It is assumed that the reader is an experienced analog/RF engineer who understands RF PCB layout and has an understanding of RF transmission lines as well as low noise analog design techniques. The ADRV9371-N/PCBZ evaluation board is used as the reference for this information, but all guidelines are best practices that can be applied to other reference designs. This document provides guidelines for system designers and discusses the following issues relative to layout and power management.

- PCB material and stack up selection
- Fanout and layout guidelines relative to trace widths and spacing
- Component placement and routing guidelines
- RF and JESD204B transmission line layout
- Isolation techniques used on the ADRV9371-N/PCBZ evaluation board
- Power management considerations—how to maximize performance without using linear low dropout (LDO) regulators
- Instructions for what to do with unused pins

PCB MATERIAL AND STACK UP SELECTION

Figure 154 shows the PCB stackup used for the evaluation board. The board employs 14 layers to achieve proper routing and isolation to best demonstrate all device functionality. The dielectric material used on the top and the bottom layers is Rogers 4003C with a thickness of 7.50 mil. The remaining dielectric layers are FR4-370 HR. The board design uses the Rogers laminate for the top and the bottom layers for its low loss tangent at high frequencies. The ground planes under the Rogers laminate (Layer 2 and Layer 13) are the reference planes for the transmission lines routed on the outer surfaces. These layers are solid copper planes under the RF traces with no discontinuities. Layer 2 and Layer 13 are crucial to maintaining the RF signal integrity. Layer 3 and Layer 12 are used to route power supply domains. To keep the RF area isolated from the fast transients of the digital area, the JESD204B interface lines are routed on Layer 5 and Layer 10. Those layers have an impedance control set to 100 Ω differential for the differential JESD204B pairs. The remaining digital signals are routed on inner Layer 7 and Layer 8. Table 178 describes details of the trace impedance controls used on different layers.

RF traces on the outer layers must be a controlled impedance to achieve the best performance. These outer layers use 1.5 oz copper so that the RF traces are less prone to pealing. One ounce copper is used for all the inner layers in this board. All ground planes on this board are full copper floods with no splits except for vias, throughhole components, and isolation structures. Note that it is important to route ground planes entirely to the edge of the PCB under the SMA connectors to maintain signal launch integrity. Power planes can be pulled back from the board edge to decrease the risk of developing shorts with the ground plane.



Layer	Type	CU Weight	CU %	Material Description	Via Structure	Segment	Glass Style	Material Family	Dielectric constant	Copper Plating Thickness [mil]
Soldermask										
I1comp	Mixed	H	100			Foil				1.40
				Press thk = 7.50 mil		Prepreg	4450F(1)	RO4003C	3.40	
							4450F(1)	RO4003C	3.40	
I2pp	Plane	1.0	100							
				3.0 mil 1/1		Core		PCL-370HR	3.90	
I3mix	Mixed	1.0	65							
				Press thk = 3.36 mil		Prepreg	106(75)	PCL-370HR	3.90	
							106(75)	PCL-370HR	3.90	
I4pp	Plane	1.0	48							
				4.0 mil 1/H		Core		PCL-370HR	3.90	
I5mix	Mixed	H	44							
				Press thk = 5.08 mil		Prepreg	1080(65)	PCL-370HR	3.90	
							1080(65)	PCL-370HR	3.90	
I6pp	Plane	1.0	68							
				3.0 mil 1/H		Core		PCL-370HR	3.90	
I7mix	Mixed	H	64							
				Press thk = 9.87 mil		Prepreg	2113(58)	PCL-370HR	3.90	
							1080(65)	PCL-370HR	3.90	
							2113(58)	PCL-370HR	3.90	
I8mix	Mixed	H	64							
				3.0 mil H/1		Core		PCL-370HR	3.90	
I9pp	Plane	1.0	68							
				Press thk = 4.72 mil		Prepreg	1080(65)	PCL-370HR	3.90	
							106(75)	PCL-370HR	3.90	
I10mix	Mixed	H	45			Foil				
				Press thk = 5.65 mil		Prepreg	106(75)	PCL-370HR	3.90	
							106(75)	PCL-370HR	3.90	
							106(75)	PCL-370HR	3.90	
I11pp	Plane	1.0	48			Foil				
				Press thk = 3.98 mil		Prepreg	106(75)	PCL-370HR	3.90	
							106(75)	PCL-370HR	3.90	
I12pp	Plane	1.0	65							
				3.0 mil 1/1		Core		PCL-370HR	3.90	
I13pp	Plane	1.0	100							
				Press thk = 7.50 mil		Prepreg	4450F(1)	RO4003C	3.40	
							4450F(1)	RO4003C	3.40	
I14sold	Mixed	H	100			Foil				1.40
Soldermask										

Figure 154. ADRV9371-N/PCBZ Evaluation Board Stackup

Table 178. ADRV9371-N/PCBZ Evaluation Board Trace Impedance Table

Layer	Impedance Requirement (Ω)	Tolerance (Ω)		Type	Reference		Line Width (mil)		Spacing (mil)		Finished Line Width (mil)	Finished Spacing (mil)	Impedance Simulation (Ω)
		Pos (+)	Neg (–)		Upper	Lower	Designed	Plotted	Designed	Coplaner			
I1comp	50	5.0	5.0	Surface single-ended coplaner		I2pp	15.50	15.00		20.00	14.50		50.2
I1comp	100	10.0	10.0	Surface microstrip differential		I2pp	8.00	9.00	6.00		8.50	5.50	100.1
I1comp	50	5.0	5.0	Coated single-ended coplaner		I2pp	15.50	13.75		20.00	13.25		49.7
I1comp	100	10.0	10.0	Coated microstrip differential		I2pp	8.00	7.50	6.00		7.00	7.00	100.6
I5mix	50	5.0	5.0	Single-ended	I6pp	I4pp	4.50	4.50			4.00		50.9
I5mix	100	10.0	10.0	Differential	I6pp	I4pp	3.60	4.25	6.40		3.75	6.25	98.9
I7mix	50	5.0	5.0	Single-ended	I8pp	I6pp	4.50	4.75			4.25		48.2
I7mix	100	10.0	10.0	Differential	I8pp	I6pp	3.60	4.00	6.40		3.50	6.50	100.6
I8mix	50	5.0	5.0	Single-ended	I7pp	I9pp	4.50	4.75			4.25		48.2
I8mix	100	10.0	10.0	Differential	I7pp	I9pp	3.60	4.00	6.40		3.50	6.50	100.6
I10mix	50	5.0	5.0	Single-ended	I11pp	I9pp	4.50	5.00			4.50		49.8
I10mix	100	10.0	10.0	Differential	I11pp	I9pp	3.60	4.50	6.40		4.00	6.00	100.8
I14sold	50	5.0	5.0	Surface single-ended coplaner		I13pp	15.50	15.00		20.00	14.50		50.2
I14sold	100	10.0	10.0	Surface microstrip differential		I13pp	8.00	9.00	6.00		8.50	5.50	100.1
I14sold	50	5.0	5.0	Coated single-ended coplaner		I13pp	15.50	13.75		20.00	13.25		49.7
I14sold	100	10.0	10.0	Coated microstrip differential		I13pp	8.00	7.50	6.00		7.00	7.00	100.6

FANOUT AND TRACE SPACE GUIDELINES

The device uses a 12 mm × 12 mm, 196-ball CSP_BGA package. The recommended BGA land pad size is 14 mils. The pitch between the pins is 0.8 mm, which makes it impractical to route all signals from the balls away from the device on a single PCB layer. RF pins have been placed on the outer edges of the package, making it easier to route the critical signals on a single PCB layer. Each digital signal is routed from the BGA pad using a 10 mil trace at a 45° angle to begin the route. The trace is connected to a fanout via that is centered between four BGA pads to maximize separation from each signal. The recommended via size is 6 mil with a 12 mil keepout. The signals are then routed to internal layers where they are routed to other parts of the system.

The JESD204B interface signals must be treated differently than other general-purpose digital signals. These traces are routed on two signal layers that utilizes impedance control (Layer 5 and Layer 10 on the ADRV9371-N/PCBZ board). The spacing between the BGA pads and each escape via is 22 mil. When the signal is connected to the inner layers, a 3.6 mil trace (50 Ω) is used to route the JESD204B signal to the FMC connector. Figure 155 shows the fanout scheme of the ADRV9371-N/PCBZ evaluation card. Another option not used on this evaluation board is the via in the pad technique. Putting the fanout vias directly inside the ball pads increases separation distance from other signals compared to placing the vias between four balls. This routing approach was not used on the ADRV9371-N/PCBZ PCB because of added cost; however, it can be used if there are no issues with manufacturing capabilities.

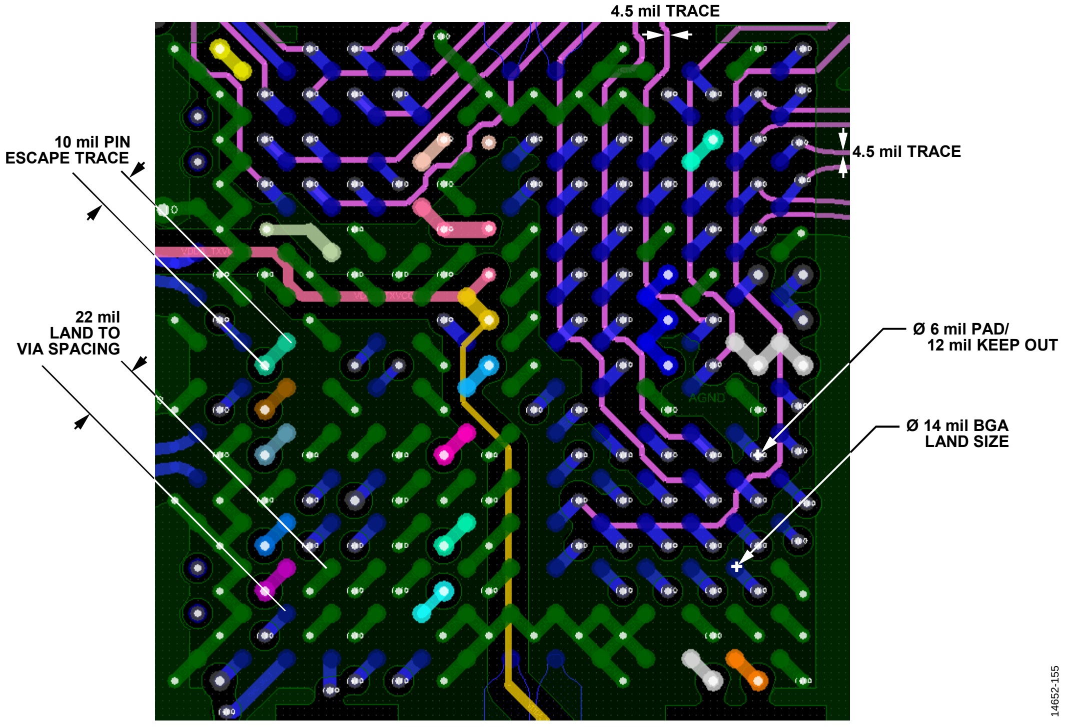


Figure 155. ADRV9371-N/PCBZ Trace Fanout Scheme

COMPONENT PLACEMENT AND ROUTING PRIORITIES

The device requires few external components to function; however, those that are needed require careful placement and routing to optimize performance. The following sections provide a priority order and checklist for properly placing and routing critical signals and components as well as those whose location and isolation are not as critical.

Signals with Highest Routing Priority

RF lines and JESD204B interface signals are the signals that are most critical and must be routed with highest priority. Figure 156 shows the general directions in which each of the signals must be routed so that they can be properly isolated from noisy signals.

RF baluns are typically used to interface single-ended signals to the differential receiver and transmitter ports. These baluns and their associated matching circuits affect overall RF performance. Every effort must be made to optimize the component selection and placement to avoid performance degradation. Refer to the RF Port Interface for more information.

RF signal path isolation is critical to achieving the level of isolation specified in the data sheet. More details on proper isolation are provided in the Isolation Techniques Used on the ADRV9371-N/PCBZ Evaluation section.

For each RF Tx output, install a 10 μF capacitor near the balun power supply pin connected to the VDDA_1P8 supply. If baluns with no dc supply connection are used, power must be supplied to the Tx outputs using RF chokes connected between the VDDA_1P8 supply and each Tx output. In both cases, the 10 μF capacitor acts as a reservoir for Tx supply current. The Tx Balun DC Supply Options section describes the Tx output power supply configuration in more detail.

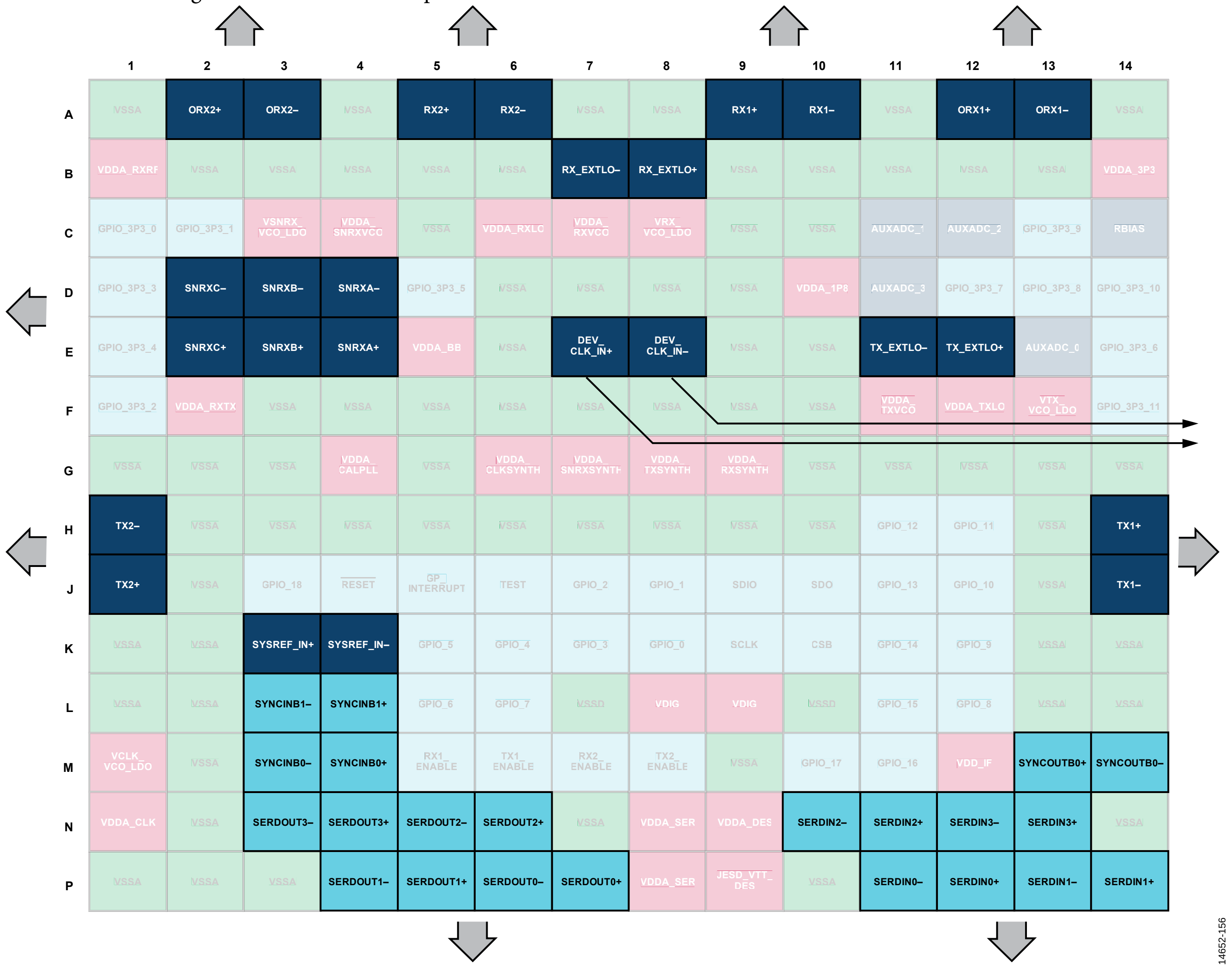

1 2 3 4 5 6 7 8 9 10 11 12 13 14
A VSSA ORX2+ ORX2– VSSA RX2+ RX2– VSSA VSSA RX1+ RX1– VSSA ORX1+ ORX1– VSSA
B VDDA_RXRF VSSA VSSA VSSA VSSA VSSA RX_EXTLO– RX_EXTLO+ VSSA VSSA VSSA VSSA VSSA VDDA_3P3
C GPIO_3P3_0 GPIO_3P3_1 VSNRX_VCO_LDO VDDA_SNRXVCO VSSA VDDA_RXLO VDDA_RXVCO VRX_VCO_LDO VSSA VSSA AUXADC_1 AUXADC_2 GPIO_3P3_9 RBIAS
D GPIO_3P3_3 SNRXC– SNRXB– SNRXA– GPIO_3P3_5 VSSA VSSA VSSA VSSA VDDA_1P8 AUXADC_3 GPIO_3P3_7 GPIO_3P3_8 GPIO_3P3_10
E GPIO_3P3_4 SNRXC+ SNRXB+ SNRXA+ VDDA_BB VSSA DEV_CLK_IN+ DEV_CLK_IN– VSSA VSSA TX_EXTLO– TX_EXTLO+ AUXADC_0 GPIO_3P3_6
F GPIO_3P3_2 VDDA_RXTX VSSA VSSA VSSA VSSA VSSA VSSA VSSA VSSA VDDA_TXVCO VDDA_TXLO VTX_VCO_LDO GPIO_3P3_11
G VSSA VSSA VSSA VDDA_CALPLL VSSA VDDA_CLKSYNTH VDDA_SNRXSYNTH VDDA_TXSYNTH VDDA_RXSYNTH VSSA VSSA VSSA VSSA VSSA
H TX2– VSSA VSSA VSSA VSSA VSSA VSSA VSSA VSSA VSSA GPIO_12 GPIO_11 VSSA TX1+
J TX2+ VSSA GPIO_18 RESET GP_INTERRUPT TEST GPIO_2 GPIO_1 SDIO SDO GPIO_13 GPIO_10 VSSA TX1–
K VSSA VSSA SYSREF_IN+ SYSREF_IN– GPIO_5 GPIO_4 GPIO_3 GPIO_0 SCLK CSB GPIO_14 GPIO_9 VSSA VSSA
L VSSA VSSA SYNCINB1– SYNCINB1+ GPIO_6 GPIO_7 VSSD VDIG VDIG VSSD GPIO_15 GPIO_8 VSSA VSSA
M VCLK_VCO_LDO VSSA SYNCINB0– SYNCINB0+ RX1_ENABLE TX1_ENABLE RX2_ENABLE TX2_ENABLE VSSA GPIO_17 GPIO_16 VDD_IF SYNCOUTB0+ SYNCOUTB0–
N VDDA_CLK VSSA SERDOUT3– SERDOUT3+ SERDOUT2– SERDOUT2+ VSSA VDDA_SER VDDA_DES SERDIN2– SERDIN2+ SERDIN3– SERDIN3+ VSSA
P VSSA VSSA VSSA SERDOUT1– SERDOUT1+ SERDOUT0– SERDOUT0+ VDDA_SER JESD_VTT_DES VSSA SERDIN0– SERDIN0+ SERDIN1– SERDIN1+
14652-156

Figure 156. RF Input/Output, DEV_CLK, and JESD204B Signal Routing Guidelines

Connect the external clock inputs to the DEV_CLK_IN+ (E7) and DEV_CLK_IN− (E8) balls using ac coupling capacitors. Use a 100 Ω termination at the input to the device. Figure 157 illustrates the recommended placement for these components near the DEV_CLK_IN± balls. Traces must be shielded by surrounding ground with vias staggered along the edge of the differential trace pair. This arrangement creates a shielded channel that prevents the reference clock from any interference from other signals. Refer to the ADRV9371-N/PCBZ evaluation card layout for exact details.

It is recommended that the JESD204B interface be routed at the beginning of the PCB design and with the same priority as RF signals. The JESD204B Trace Routing Recommendations section outlines recommendations for JESD204B interface routing. Ensure that appropriate isolation between these differential pairs are provided. The Isolation Between JESD204B Lines section provides guidelines for optimizing isolation.

The RX_EXTLO− (B7), RX_EXTLO+ (B8), TX_EXTLO− (E11), TX_EXTLO+ (E12) balls are internally dc biased. If an external local oscillator (LO) is used, connect it via ac coupling capacitors.

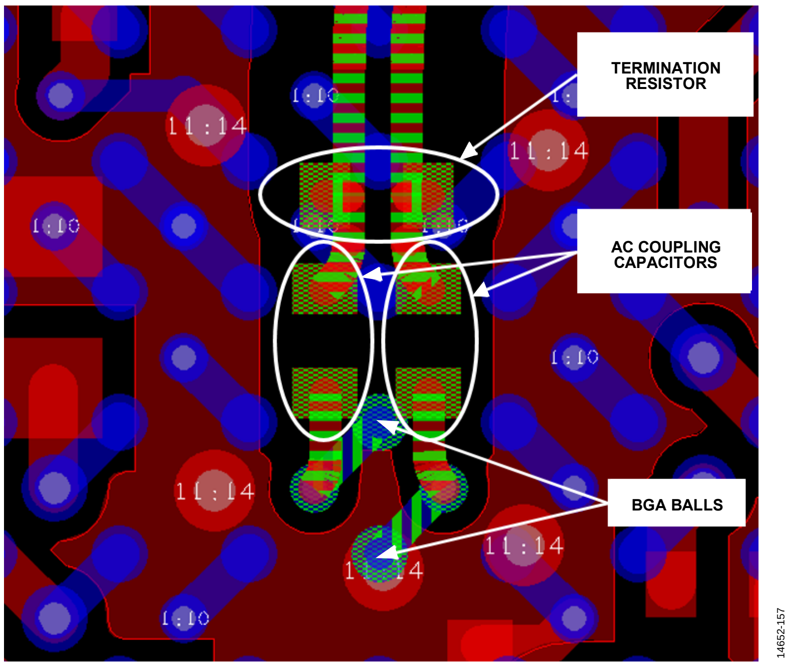


Figure 157. DEV_CLK Signal Routing Recommendations

Signals with Second Routing Priority

Power supply quality has direct impact on overall system performance. To achieve optimal performance, follow the recommendations regarding power supply routing. The following recommendations outline how different power domains can be routed and which supplies can be tied to the same supply but separated by a ferrite bead or 0 Ω resistor.

A general recommendation for power supply routing is to follow the star methodology in which each power domain is deliver by a separate trace from the source supply. Ensure that each power trace is surrounded by ground. Figure 158 shows an example of such traces routed on the evaluation card on Layer 12. Each trace is separated from any other signal by ground plane fill and vias. This approach is essential to providing necessary isolation between power domains.

Each power supply requires a 0.1 µF bypass capacitor near the ball at a minimum. Place the ground side of the bypass capacitor so that ground currents flow away from other power balls and their bypass capacitors.

For those domains shown in Figure 159 that are powered through a ferrite bead (FB), place the ferrite beads near the supply pins. It is recommended to space the ferrite beads to ensure their electric fields do not influence each other. Figure 160 shows an example of how to place the ferrite beads, reservoir capacitors, and decoupling capacitors. The ferrite bead must supply a trace with a reservoir capacitor connected to it. It is recommended to shield this trace with ground and to provide power to the input power ball. Place a 100 nF capacitor near the power supply ball with the ground side of the bypass capacitor placed so that ground currents flow away from other power balls and their bypass capacitors.

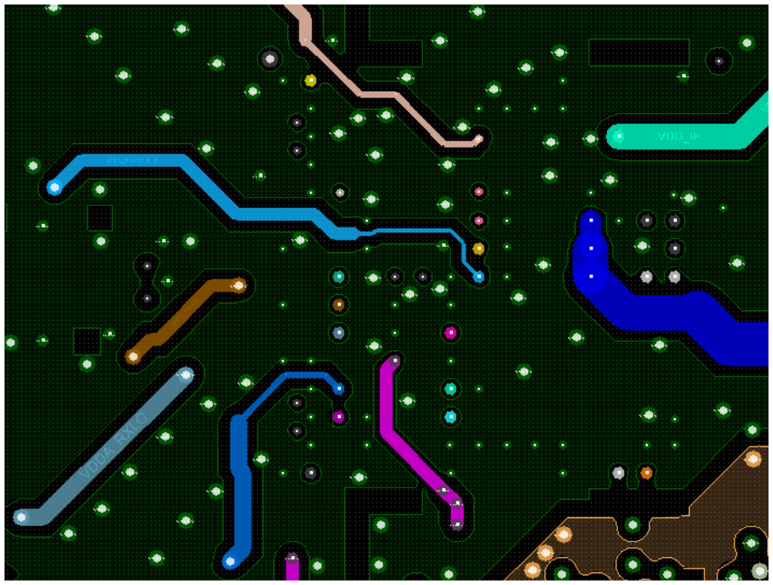


Figure 158. Layout Example of Power Supply Connections Routed with Ground Shielding (Layer 12)

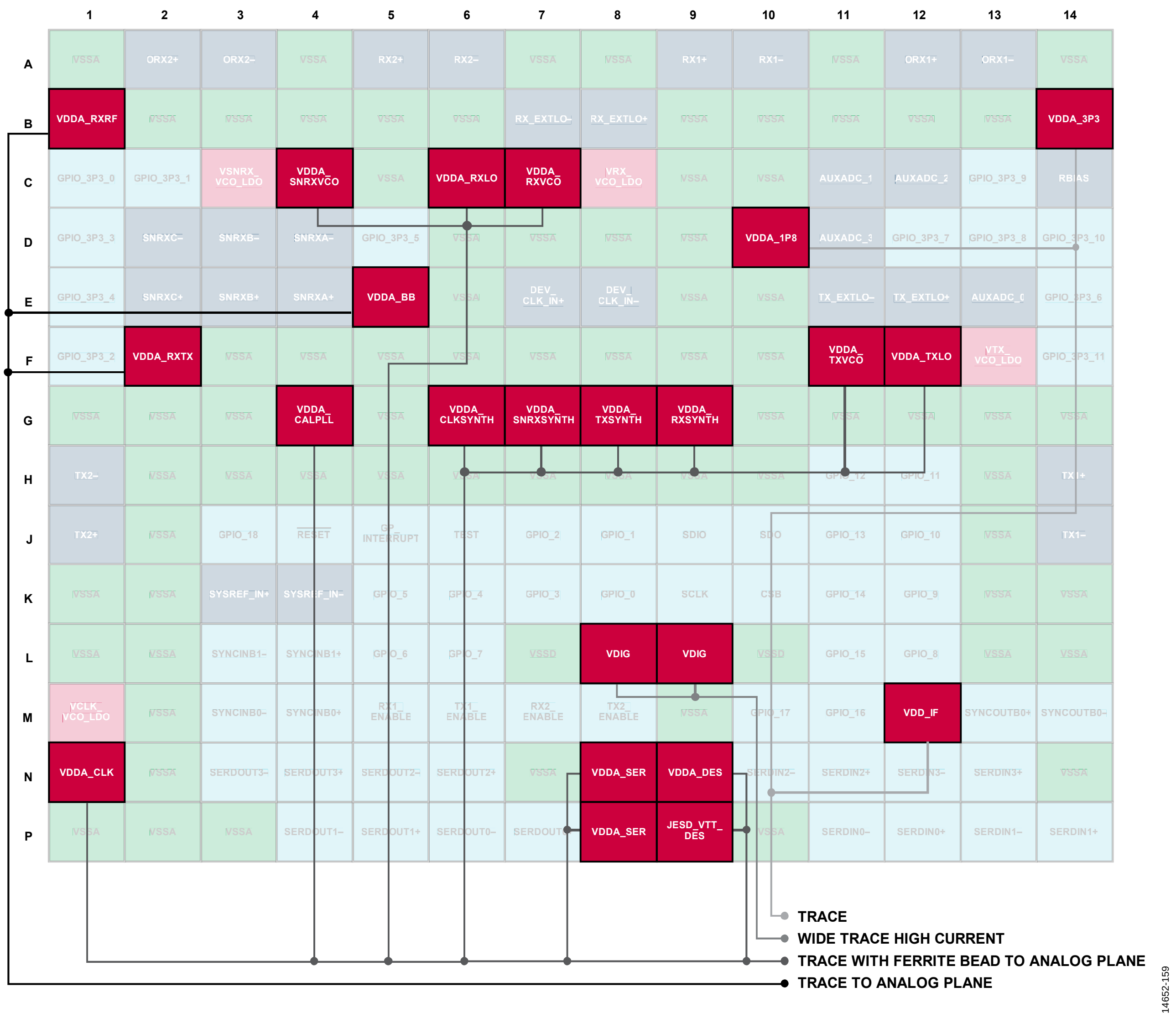


Figure 159. Power Supply Domains with Connection Guidelines

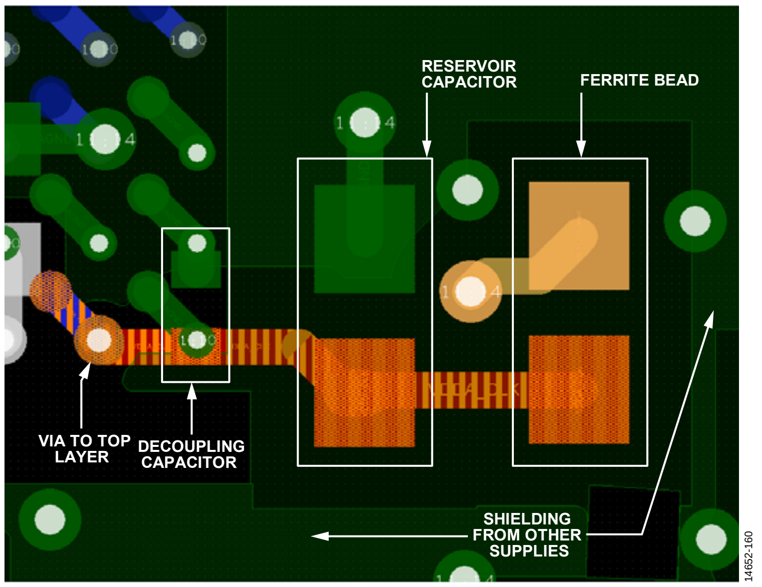


Figure 160. Placement Example for Ferrite Bead, Reservoir Capacitor, and Decoupling Capacitor on the ADRV9371-N/PCBZ Evaluation Card

Signals with Lowest Routing Priority

The following guidelines govern those signals that are the lowest signal routing priority. These signals can be routed after all critical signal routes have been completed so that they do not interfere with the critical component placement and routing. The signals shown in Figure 161 can be routed with the lowest priority.

- Ceramic 1 µF bypass capacitors must be placed at the VRX_VCO_LDO, VTX_VCO_LDO, VSNRX_VCO_LDO and VCLK_VCO_LDO balls. Place these capacitors as close as possible to the device with the ground side of the bypass capacitor placed so that ground currents flow away from other power balls and their bypass capacitors if at all possible.
- Connect a 14.3 kΩ resistor to RBIAS pin (C14). This resistor must have a 1% tolerance or better.
- The device has support for JTAG boundary scan, and the TEST ball is used to access the function. Connect the TEST ball (J6) to ground for normal operation. Refer to the data sheet for JTAG boundary scan information.
- Connect the $\overline{\text{RESET}}$ pin (J4) to VDD_IF with a 10 kΩ resistor for normal operation. The device can be reset by driving this pin low.
- When routing digital signals from Row H and under, it is important to route them away from the analog section (Row A through Row G). It is recommended that digital signal routing not pass before the dashed line highlighted in Figure 161.
- The GPIO_3P3_N signals can be routed using inner PCB layers. Those signals control analog blocks such as power amplifiers or low noise amplifiers. They can also be used as general-purpose analog outputs when muxed to the internal auxiliary DAC outputs. To prevent noise coupling into those signals, route them away from the digital region (before the dashed line highlighted in Figure 161).
- The AUXADC_N signals can be routed using inner PCB layers. Those signals sense analog voltage levels such as temperature sensors. To prevent noise coupling into those signals, route them away from the digital region (before the dashed line highlighted in Figure 161).

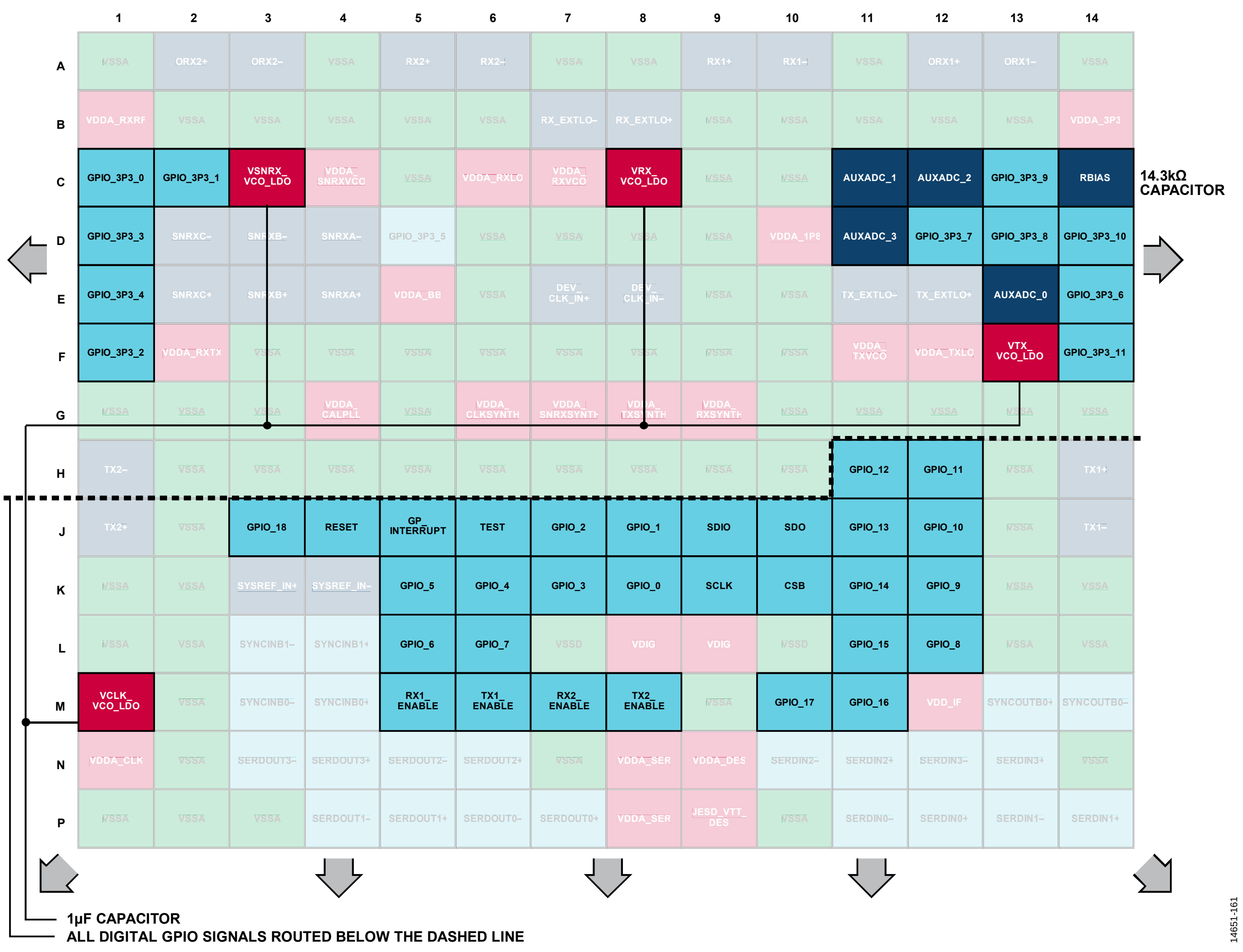

1 2 3 4 5 6 7 8 9 10 11 12 13 14
A VSSA ORX2+ ORX2– VSSA RX2+ RX2– VSSA VSSA RX1+ RX1– VSSA ORX1+ ORX1– VSSA
B VDDA_RXRF VSSA VSSA VSSA VSSA VSSA RX_EXTLO– RX_EXTLO+ VSSA VSSA VSSA VSSA VSSA VDDA_3P3
C GPIO_3P3_0 GPIO_3P3_1 VSNRX_VCO_LDO VDDA_SNRXVCO VSSA VDDA_RXLO VDDA_RXVCO VRX_VCO_LDO VSSA VSSA AUXADC_1 AUXADC_2 GPIO_3P3_9 RBIAS
D GPIO_3P3_3 SNRXC– SNRXB– SNRXA– GPIO_3P3_5 VSSA VSSA VSSA VSSA VDDA_1P8 AUXADC_3 GPIO_3P3_7 GPIO_3P3_8 GPIO_3P3_10
E GPIO_3P3_4 SNRXC+ SNRXB+ SNRXA+ VDDA_BB VSSA DEV_CLK_IN+ DEV_CLK_IN– VSSA VSSA TX_EXTLO– TX_EXTLO+ AUXADC_0 GPIO_3P3_6
F GPIO_3P3_2 VDDA_RXTX VSSA VSSA VSSA VSSA VSSA VSSA VSSA VSSA VDDA_TXVCO VDDA_TXLO VTX_VCO_LDO GPIO_3P3_11
G VSSA VSSA VSSA VDDA_CALPLL VSSA VDDA_CLKSYNTH VDDA_SNRXSYNTH VDDA_TXSYNTH VDDA_RXSYNTH VSSA VSSA VSSA VSSA VSSA
H TX2– VSSA VSSA VSSA VSSA VSSA VSSA VSSA VSSA VSSA GPIO_12 GPIO_11 VSSA TX1+
J TX2+ VSSA GPIO_18 RESET GP_INTERRUPT TEST GPIO_2 GPIO_1 SDIO SDO GPIO_13 GPIO_10 VSSA TX1–
K VSSA VSSA SYSREF_IN+ SYSREF_IN– GPIO_5 GPIO_4 GPIO_3 GPIO_0 SCLK CSB GPIO_14 GPIO_9 VSSA VSSA
L VSSA VSSA SYNCINB1– SYNCINB1+ GPIO_6 GPIO_7 VSSD VDIG VDIG VSSD GPIO_15 GPIO_8 VSSA VSSA
M VCLK_VCO_LDO VSSA SYNCINB0– SYNCINB0+ RX1_ENABLE TX1_ENABLE RX2_ENABLE TX2_ENABLE VSSA GPIO_17 GPIO_16 VDD_IF SYNCOUTB0+ SYNCOUTB0–
N VDDA_CLK VSSA SERDOUT3– SERDOUT3+ SERDOUT2– SERDOUT2+ VSSA VDDA_SER VDDA_DES SERDIN2– SERDIN2+ SERDIN3– SERDIN3+ VSSA
P VSSA VSSA VSSA SERDOUT1– SERDOUT1+ SERDOUT0– SERDOUT0+ VDDA_SER JESD_VTT_DES VSSA SERDIN0– SERDIN0+ SERDIN1– SERDIN1+
14.3kΩ CAPACITOR
1µF CAPACITOR
ALL DIGITAL GPIO SIGNALS ROUTED BELOW THE DASHED LINE
14651-161

Figure 161. Auxiliary ADC, SPI, Analog GPIO/Auxiliary DAC, and Digital GPIO Signal Routing Guidelines

RF AND JESD204B TRANSMISSION LINE LAYOUT

Board layout design involves compromise. The recommendations within this user guide are intended for wide RF bandwidth applications. For narrow RF bandwidth applications, the board line impedance parameters within this document may not be optimal.

The following list provides general suggestions for board design:

- Match the evaluation board design as close as possible to the board design files available on the product page.
- Be attentive to power distribution and power ground return methodology.
- Do not run high speed digital lines in close proximity to dc power distribution routes or RF line routes.
- Use microstrip or coplanar waveguides (CPWG) for transmission lines. These structures do not require via structures that cause additional impedance discontinuities that vary across frequency. For ports such as the Sniffer receivers, which do not have balls on the perimeter of the BGA, a via structure such as stripline may be necessary.

Design the RF line systems between the device ball pad reference plane and the balun/filter reference plane for a differential impedance (Z_{DIFF}) of 100 Ω for the receivers and 50 Ω for the transmitters. This design is a compromise impedance with respect to frequency and a good starting point for design. The Z_{DIFF} can be optimized to fit a narrower frequency range. It is desirable to design the lines for reasonable coupling (−10 dB to −15 dB) to promote adequate electromagnetic interface (EMI) suppression performance.

In most cases, the required board artwork stackup is different than the ADRV9371-N/PCBZ evaluation board stackup. Optimization of RF transmission lines specific to the desired board environment is essential to the design and layout process.

The ADRV9371-N/PCBZ evaluation board uses microstrip lines for Rx, ORx and Tx RF traces. The SnRx signal is routed using a combination of microstrip lines on the bottom of the PCB and stripline traces on internal layers due to board complexity. In general, it is not recommended to use vias with RF traces unless a direct line route is not possible.

Differential lines from the balun to the Rx, ORx, SnRx and Tx balls must be as short as possible. It is also recommended that the length of the single-ended transmission lines be short to minimize the effects of parasitic coupling.

System designers can optimize RF performance with the proper selection of balun, matching components, and ac coupling capacitors. The external local oscillator (LO) traces and the DEV_CLK_IN traces may require matching components as well to ensure optimal performance. For additional information on matching network design see the RF Port Interface section.

Differential Line Design Equations

Some high level differential line design equations follow; these are valid for reasonably low loss transmission lines.

Odd Impedance Mode

Odd impedance mode is represented by the variable, Z_{ODD}.

$$Z_{ODD} = Z_{DIFF} \div 2 \tag{33}$$

where Z_{DIFF} is 100 Ω for receivers and 50 Ω for trasmitters.

Even Impedance Mode

Even impedance is represented by the variable, Z_{EVEN}.

$$Z_{EVEN} = Z_{CM} \times 2 \tag{34}$$

where:
Z_{CM} is the common-mode impedance from each pin to ground.

Differential Mode Characteristic Impedance

Differential mode impedance is represented by the variable, Z_0.

$$Z_0 = \sqrt{Z_{ODD} Z_{EVEN}} \tag{35}$$

where:
Z_{EVEN} is a function of line coupling. As the line coupling increases, both Z_{EVEN} and Z_0 increase. Given the ball pad diameter of 17.7 mil, 0.45 mm, and the array pitch of 31.5 mil, 0.8 mm coupled, microstrip differential lines are the preferred design choice.

Single-Ended Impedance

Single-ended impedance is represented by the variable, Z_{SE}.

$$Z_{SE} = (Z_{ODD} + Z_{EVEN}) \div 2 \tag{36}$$

Inductance per Unit Length

Inductance per unit length is represented by the variable, L′.

$$L' = \frac{Z_0\sqrt{\varepsilon_r}}{c} \tag{37}$$

where:
ε_r is the media relative dielectric constant.
c is the speed of light (1.1803×10^{13} mils/sec).

Capacitance per Unit Length

Capacitance per unit length is represented by the variable, C′.

$$C' = \frac{\sqrt{\varepsilon_r}}{Z_0 c} \tag{38}$$

Alternative Characteristic Differential Impedance

The alternative characteristic differential impedance is represented by the variable, Z_0.

$$Z_0 = \sqrt{\frac{L'}{C'}} \tag{39}$$

Line Design Examples

The following sections provide examples of transmission line design. These design examples frequently use Equation 33 through Equation 39 and electromagnetic simulation tools to calculate the impedance parameters.

Example 1: Microstrip Line System, Receiver Only, Z_{DIFF} = 100 Ω

The line system described by the parameters in Table 179 is a well suited line system for the receiver system. The line coupling is adequate and electromagnetic interface (EMI) issues are not expected.

This design is not acceptable for a JESD204B application. The Z_{DIFF} requirement is met, but the Z_{CM} requirement of 25 Ω is not met. This line system is only acceptable for receiver lines.

Table 179. Example 1 Microstrip Parametric Calculated Results

Parameter	Value
ε_r (Rogers 4003C)	3.55
Dissipation Loss Tangent (TanD)	0.0021
Height	8.0 mil
Width	8.5 mil
Spacing	5.0 mil
Z_{EVEN}	85.1 Ω
Z_{ODD}	50.2 Ω
Z_0	65.4 Ω
Coupling	−11.8 dB
Inductance per Unit Length (L/UL)	10.44 pH/mil
Capacitance per Unit Length (C/UL)	2.44 fF/mil
$Z_{DIFF} = 2 \times Z_{ODD}$	100.4 Ω
$Z_{CM} = Z_{EVEN} \div 2$	42.6 Ω

Example 2: Microstrip Line System, Receiver and JESD204B Traces, Z_{DIFF} = 100 Ω

From a line impedance perspective, this is a good system for both the receiver system and JESD204B system (see Table 180). However, the lines are weakly coupled. Exercise care during the board layout phase to reduce EMI risk.

Another advantage of Example 2 over Example 1 is that the line width is closer to the board ball diameter of 17.7 mil thereby reducing discontinuities between the line and pad structures.

Table 180. Example 2 Microstrip Parametric Calculated Results

Parameter	Value
ε_r (Rogers 4003C)	3.55
Dissipation Loss Tangent (TanD)	0.0021
Height	8.0 mil
Width	15.0 mil
Spacing	30.0 mil
Z_{EVEN}	54.3 Ω
Z_{ODD}	50.2 Ω
Z_0	52.2 Ω
Coupling	−27.9 dB
Inductance per Unit Length (L/UL)	8.33 pH/mil
Capacitance per Unit Length (C/UL)	3.06 fF/mil
$Z_{DIFF} = 2 \times Z_{ODD}$	100.4 Ω
$Z_{CM} = Z_{EVEN} \div 2$	27.2 Ω

Example 3: Microstrip Line System, Transmitter Traces Only, Z_{DIFF} = 50 Ω

Generally, this is a good system for transmitter systems (see Table 181). The line coupling is adequate, and EMI issues are not expected. These lines are wide relative to the transmitter board ball pads; they must be tapered down to create minimal discontinuity between the lines and pads. Some amount of line width tuning may be required to obtain an adequate broadband system impedance.

Table 181. Example 3 Microstrip Parametric Calculated Results

Parameter	Value
ε_r (Rogers 4003C)	3.55
Dissipation Loss Tangent (TanD)	0.0021
Height	8.0 mil
Width	34.0 mil
Spacing	4.0 mil
Z_{EVEN}	35.2 Ω
Z_{ODD}	25.2 Ω
Z_0	29.8 Ω
Coupling	−15.6 dB
Inductance per Unit Length (L/UL)	4.76 pH/mil
Capacitance per Unit Length (C/UL)	5.36 fF/mil
$Z_{DIFF} = 2 \times Z_{ODD}$	50.4 Ω
$Z_{CM} = Z_{EVEN} \div 2$	17.6 Ω

Example 4: Microstrip Line System, Receiver and JESD204B Traces, Typical Production PCB, Z_{DIFF} = 100 Ω

This line design represents a typical production circuit board scenario (see Table 182). The low stackup height represents a challenge to generating a Z_{DIFF} of 100 Ω. Generally, this is an adequate line for the receiver and JESD204B systems; however, exercise care during board layout to minimize the line to ball pad discontinuities and potential EMI risk.

Table 182. Example 4 Microstrip Parametric Calculated Results

Parameter	Value
ε_r (Rogers 4003C)	3.55
Dissipation Loss Tangent (TanD)	0.0021
Height	8.0 mil
Width	34.0 mil
Spacing	4.0 mil
Z_{EVEN}	35.2 Ω
Z_{ODD}	25.2 Ω
Z_0	29.8 Ω
Coupling	−15.6 dB
Inductance per Unit Length (L/UL)	4.76 pH/mil
Capacitance per Unit Length (C/UL)	5.36 fF/mil
$Z_{DIFF} = 2 \times Z_{ODD}$	50.4 Ω
$Z_{CM} = Z_{EVEN} \div 2$	17.6 Ω

Example 5: Microstrip Line System, Transmitter Traces Only, Z_{DIFF} = 100 Ω

Generally, this is a good line design for the transmitter system. The line coupling is marginal. Exercise care during board layout to reduce the EMI risk. Note this example assumes FR4 for the circuit board material, so the dielectric constant is substantially higher than the other examples.

Table 183. Example 5 Microstrip Parametric Calculated

Parameter	Value
ε_r (Rogers 4003C)	4.6
Dissipation Loss Tangent (TanD)	0.025
Height	3.0 mil
Width	12.0 mil
Spacing	4.0 mil
Z_{EVEN}	30.8 Ω
Z_{ODD}	25.2 Ω
Z_0	27.9 Ω
Coupling	−19.9 dB
Inductance per Unit Length (L/UL)	5.07 pH/mil
Capacitance per Unit Length (C/UL)	6.51 fF/mil
$Z_{DIFF} = 2 \times Z_{ODD}$	50.4 Ω
$Z_{CM} = Z_{EVEN} \div 2$	15.4 Ω

RF Line Design Summary

As evident in Example 1 through Example 5, the RF line design is a compromise between many variables. Line impedance, line to line coupling, and physical size represent the parameters subject to compromise.

Smallest physical size is in direct opposition to the Z_{CM} of the line, which is directly opposed to the line electromagnetic interface (EMI) performance. In addition, the interface between the RF line width and the device ball pad diameter on the PCB represents a potential discontinuity. As the RF line width approaches the ball pad diameter, the risk associated with potential interface discontinuity reduces.

The circuit shown in Figure 162 shows the layout topology for the chosen receiver matching network. Note the location and orientation of each component; placement is critical to achieve expected performance. Similarly, the circuit in Figure 163 shows the layout topology used for the transmitter matching network (see the RF Port Interface section for circuit details).

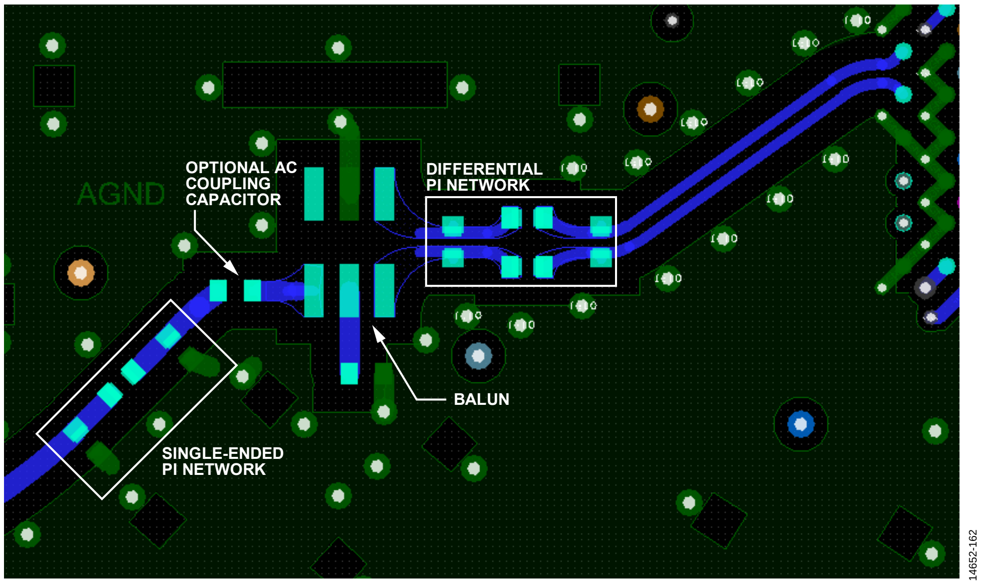


Figure 162. Receiver Matching Network on ADRV9371-N/PCBZ Evaluation Board

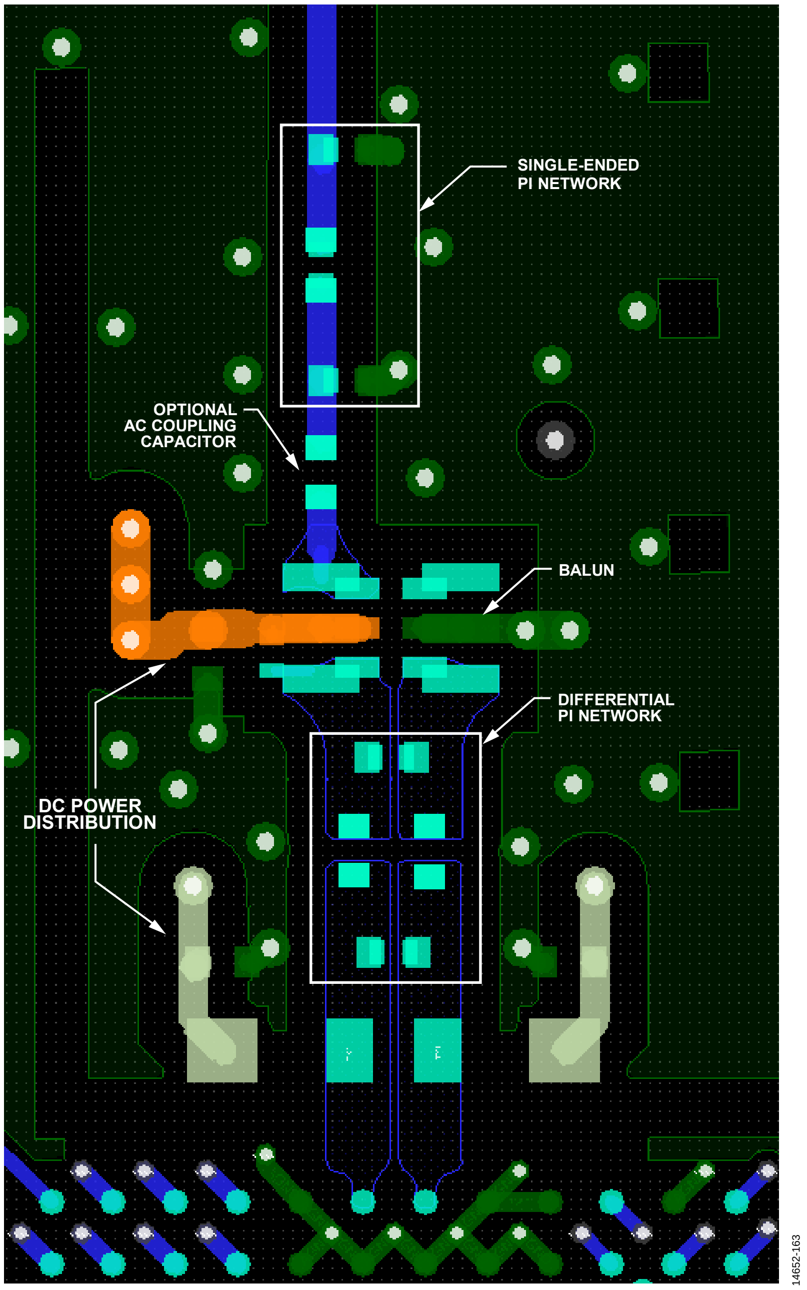


Figure 163. Transmitter Matching Network on ADRV9371-N/PCBZ Evaluation Board

Transmitter Bias Design Considerations

This section considers the dc biasing of the device transmitter (Tx) outputs and how to interface to each Tx port. At full output power, each differential output side draws approximately 100 mA of dc bias current. The Tx outputs are dc biased to a 1.8 V supply voltage using either RF chokes (wire wound inductors) or a transformer (balun) 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 dc bias schemes suggested in Figure 164 and Figure 165. The resistors (R_{DCR}) indicate the parasitic elements. As the impedance of the parasitics increase, the voltage drop (ΔV) across the parasitic element increases which causes the transmitter RF performance to degrade. The choke inductance (L_C) must be selected high enough relative to the load impedance such that it does not degrade the output power. If chokes are used, they must be very well matched (including PCB traces). Uneven matching of chokes design can cause unwanted emission of spikes at the Tx output. This emission can affect components connected to the Tx output.

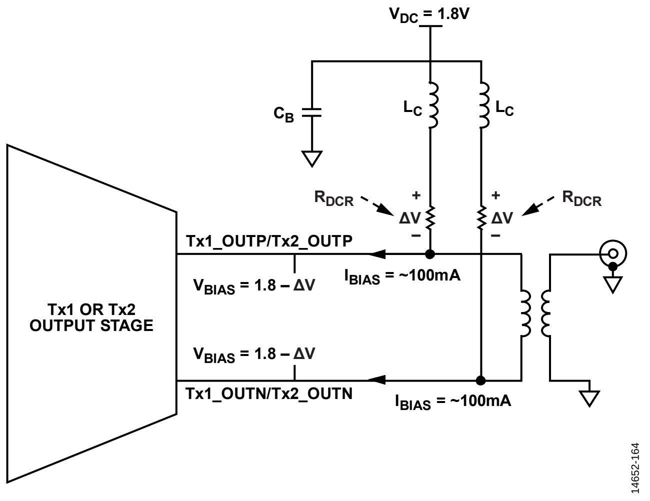


Figure 164. ADRV9371-N/PCBZ DC Bias Configuration for the Transmitter Output Using Wire Wound Chokes

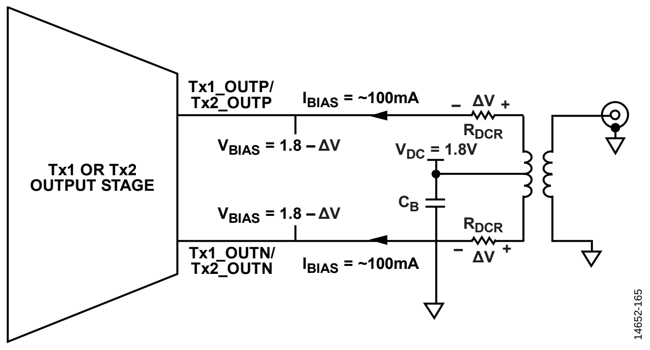


Figure 165. ADRV9371-N/PCBZ DC Bias Configuration for the Transmitter Output Using a Center Tapped Transformer

The recommended dc bias network is the one using the center tap balun shown in Figure 165. This network has fewer parasitics and fewer total components.

The ADRV9371-N/PCBZ evaluation board provides flexibility to configure each Tx output to either work with a center tapped transformer (balun) or a set of two closely matched wire wounded chokes. The center tapped transformer passes the bias voltage directly to the transmitter outputs through each differential input. This configuration offers the lowest component count.

In some cases, the desired balun does not provide a dc connection to the transmitter output lines. To support this situation, the ADRV9371-N/PCBZ evaluation board provides the placeholders for RF chokes tied to the VDDA_1P8 (1.8 V) supply. It also provides the placeholders for ac coupling capacitors to prevent creating a dc short through the balun to ground.

Impedance matching networks on the balun single-ended port are usually required to achieve optimum performance. In addition, ac coupling is often required on the single-ended side if the balun contains a dc path from one of the differential outputs transmitter to the single-ended port.

Careful planning is required for the Tx balun selection. If a Tx balun is selected that requires a set of external dc bias chokes, it is necessary to find the optimum compromise between the choke physical size, choke dc resistance (R_{DCR}), and the balun pass-band insertion loss. Refer to the RF Port Interface section for more information on Tx output balun and RF choke selection as well as matching circuit recommendations.

Tx Balun DC Supply Options

Each transmitter requires approximately 200 mA supplied through an external connection. The PCB layout of the ADRV9371-N/PCBZ allows use of external chokes to provide the 1.8 V power domain to the device outputs to allow users to try different baluns that may not have a dc center tap pin to supply the bias voltage to the transmitter outputs.

To reduce switching transients when attenuation settings change, it is recommended to power the balun dc feed directly by the 1.8 V plane. Design the geometry of the 1.8 V plane so that each balun or each pair of chokes is isolated from those of the other transmitter. If careful layout and isolation of the dc supply is not followed, it can adversely affect Tx to Tx isolation.

Figure 166 shows the power supply layout configuration used on the ADRV9371-N/PCBZ board to achieve the desired Tx to Tx isolation performance. This image illustrates how a trade-off was used when a direct star connection was not possible. To improve isolation, one transmitter is fed from the power plane, and the plane is continued until it can be connected to the other transmitter. This process keeps the impedance for the second transmitter feed as low as possible and separates the current paths enough to avoid intermingling of supply currents.

An example of the balun feed supply designed to achieve the channel isolation in the evaluation board is shown in Figure 167 and Figure 168.

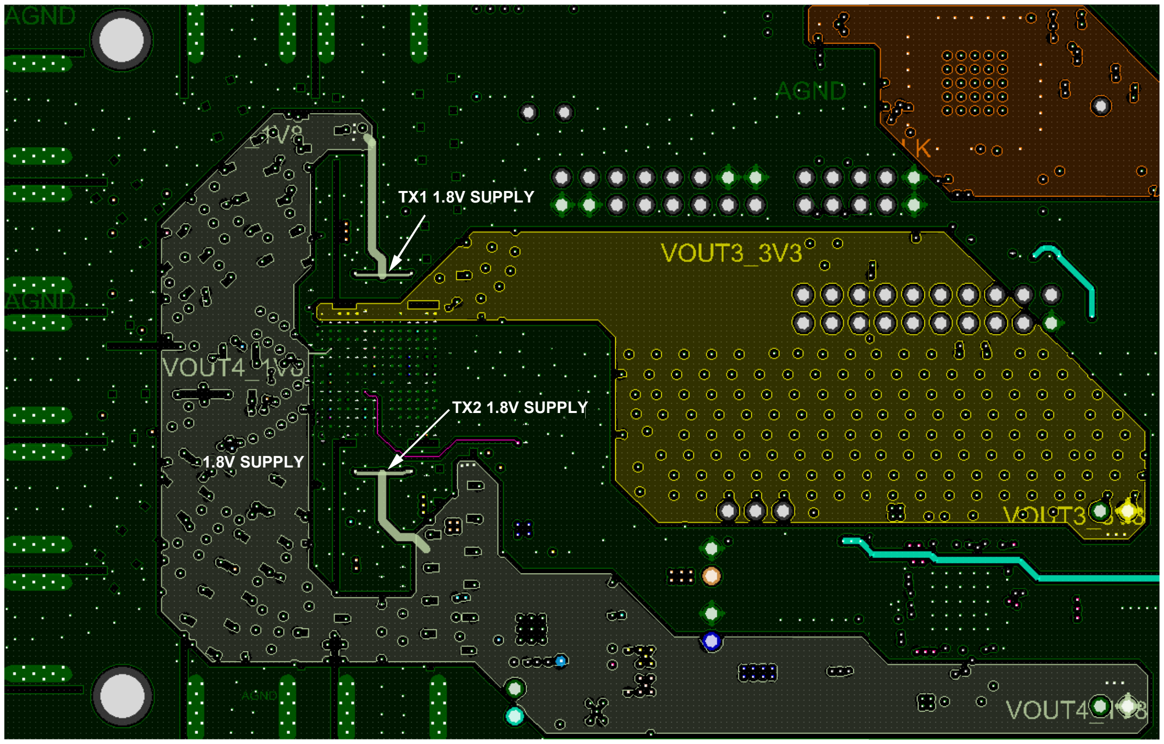


Figure 166. 1.8 V Tx Power Supply Routing on the ADRV9371-N/PCBZ Evaluation Board

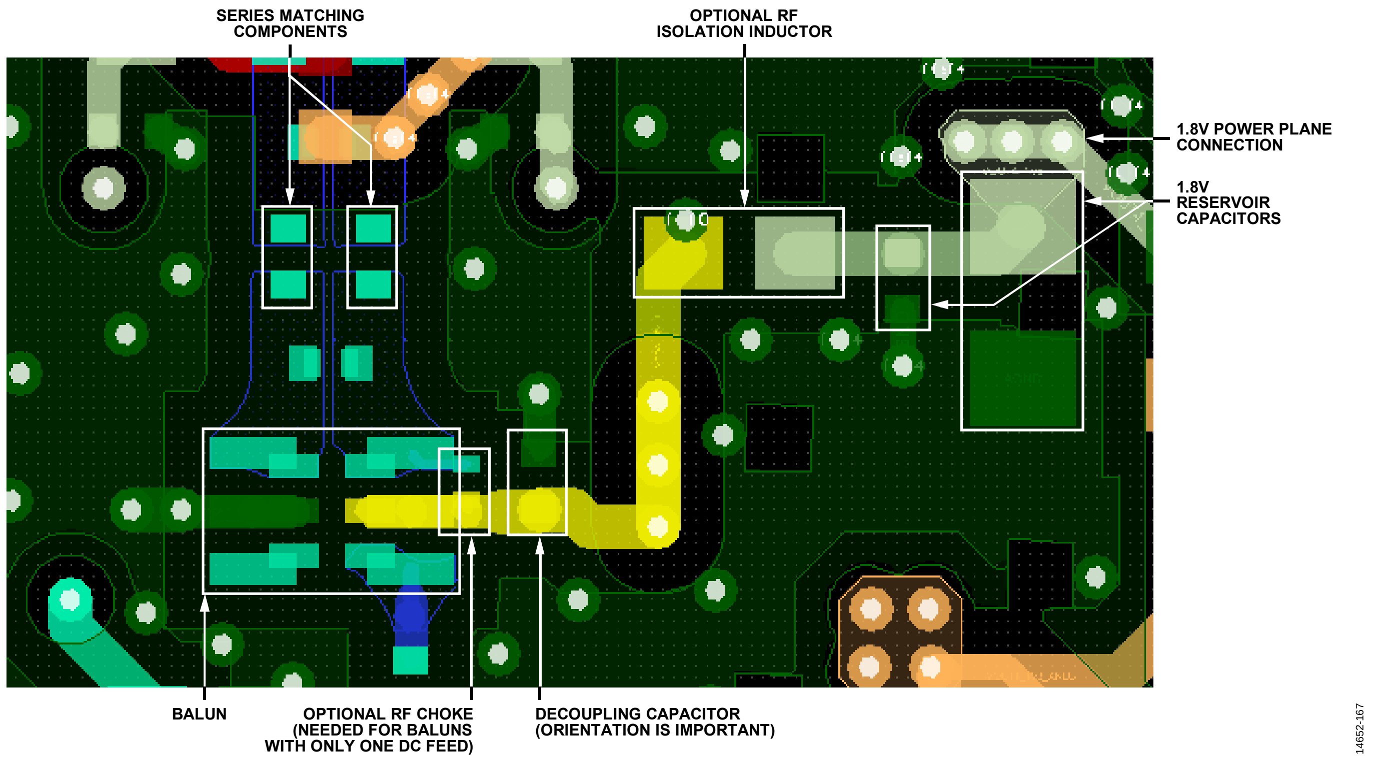

SERIES MATCHING COMPONENTS
OPTIONAL RF ISOLATION INDUCTOR
1.8V POWER PLANE CONNECTION
1.8V RESERVOIR CAPACITORS
BALUN
OPTIONAL RF CHOKE (NEEDED FOR BALUNS WITH ONLY ONE DC FEED)
DECOUPLING CAPACITOR (ORIENTATION IS IMPORTANT)
14652-167

Figure 167. Transmitter Power Supply for a Balun with a Center Tap

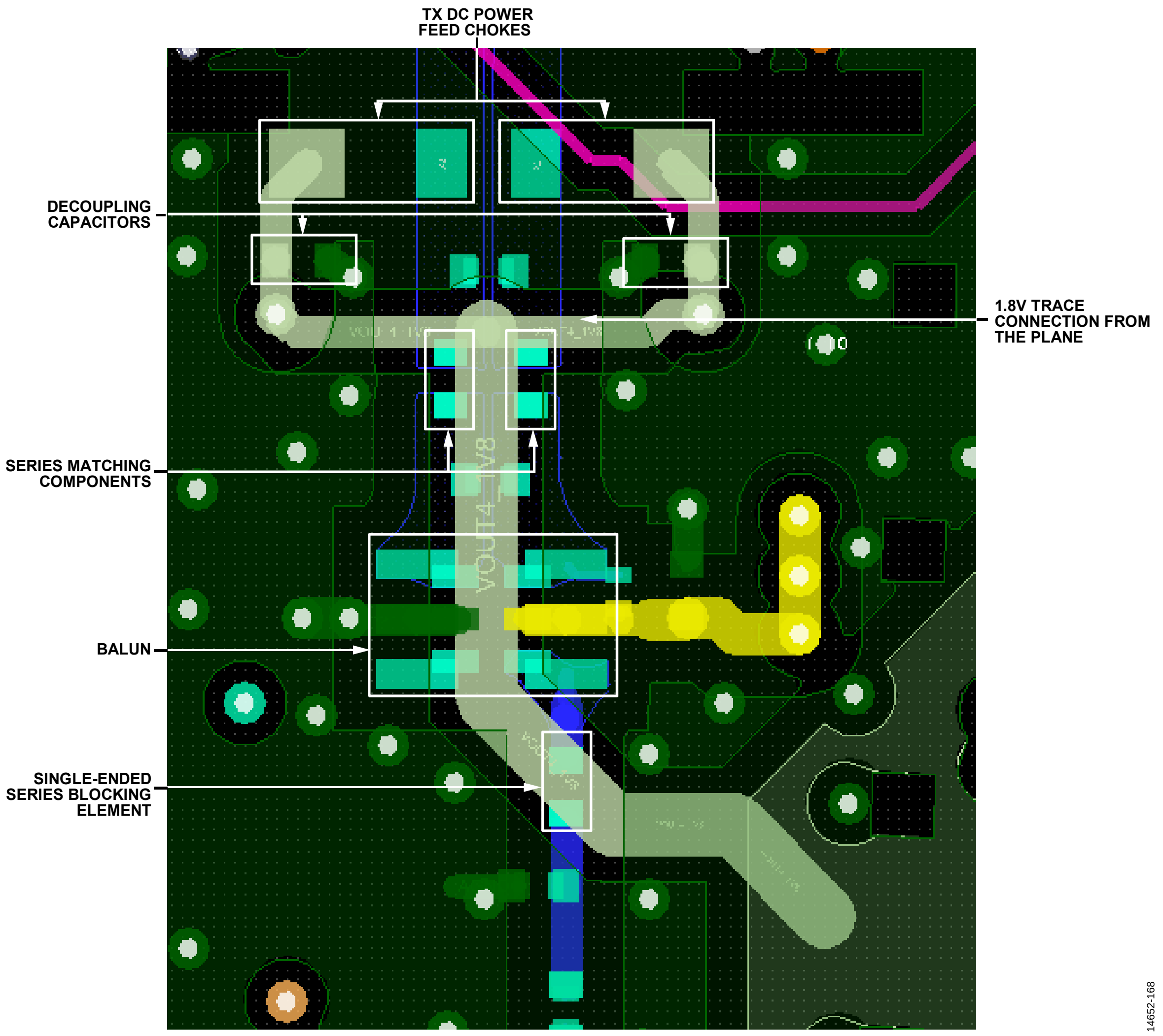

TX DC POWER FEED CHOKES
DECOUPLING CAPACITORS
1.8V TRACE CONNECTION FROM THE PLANE
SERIES MATCHING COMPONENTS
BALUN
SINGLE-ENDED SERIES BLOCKING ELEMENT
VOUT4_1V8
14652-168

Figure 168. Transmitter Power Supply Using RF Chokes

DC Balun

When a Tx balun that is able to conduct dc is used, use the system shown in Figure 167. Place the decoupling capacitor near the Tx balun as close as possible to the dc feed pin of the balun. Its orientation must be perpendicular to the device so that the return current avoids a ground loop with the ground pins surrounding the Tx input. The evaluation board provides an option to install an RF isolation inductor, which can provide extra isolation between the Tx1 and Tx2 balun supply feeds. A 10 µF capacitor and a 0.1 µF capacitor are helpful on the dc feed pin to eliminate Tx spectrum spurs and dampen the transients. Note that when this supply approach is used, the series matching components must be dc shorts. It is recommended to use 0 Ω if an inductor is not needed to match the balun impedance to the Tx output impedance.

Chokes

The ADRV9371-N/PCBZ evaluation board provides flexibility to use a Tx balun that is not capable of conducting dc current. In such a scenario, the user must install dc chokes as well as their decoupling capacitors as highlighted in Figure 168. Care must be taken to match both chokes to avoid potential current spikes. Differences in parameters between both chokes can cause unwanted emission at Tx outputs. Note that, if the differential input to the balun can form a dc short to ground through the balun, the series matching components must be capacitors. If a short can form on the single-ended side, the single-end series blocking element must be a capacitor.

JESD204B Trace Routing Recommendations

Routing the JESD204B data lines requires techniques similar to routing differential RF traces. 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 (BBP) and route the traces as directly as possible between the devices. Using a PCB material with a low dielectric constant (<4) to minimize loss is also strongly recommended. For distances greater than 6 inches, it is recommended to use a premium PCB material such as Rogers 4003C.

Routing Recommendations

Route the differential pairs on a single plane using a solid ground plane as a reference on the layers above and/or below these traces

All JESD204B lane traces must be impedance controlled to achieve 50 Ω to ground. It is recommended that the differential pair be coplanar and loosely coupled (a typical configuration is 5 mil trace width, 15 mil edge to edge spacing) with the trace width maximized.

It is recommended that trace widths match pin/ball widths as closely as possible while maintaining impedance control. Trace widths of at least 8 mils using 1 oz. copper are recommended. It is recommended that coupling capacitor pad size match JESD204B lane trace widths as closely as possible.

Pad area for all connector and passive component choices must be minimized as much as possible due to a capacitive plate effect that can lead to problems with signal integrity.

Reference planes for impedance controlled signals must not be segmented or broken for the entire length of a trace.

The DEV_CLK_IN and SYSREF signal traces must be impedance controlled for Z_0 = 50 Ω.

Stripline vs. Microstrip

When routing the PCB layout for JESD204B data lines, the designer must decide to route the signals using stripline or microstrip traces. There are positives and negatives for each that must be carefully considered.

- Stripline has less loss and emits less electromagnetic interface (EMI) than microstrip lines, but stripline traces require the use of vias that can add complexity to the task of controlling the impedance by adding line inductance.
- Microstrip is easier to implement if the component placement and density allow for routing on the top layer, simplifying the task of controlling the impedance.

If using the top layer of the PCB is problematic or the advantages of stripline are desirable, follow these recommendations:

- Minimize the number of vias.
- Use blind vias wherever possible to eliminate via stub effects, and use microvias to minimize via inductance.
- If using standard vias, use maximum via length to minimize the stub size. For example, on an 8-layer board, use Layer 7 for the stripline pair.
- For each via pair, place a pair of ground vias in close proximity to them to minimize the impedance discontinuity.
- For the JESD204B lines, the recommendation is to route them on the top side of the board as a differential 100 Ω pair (microstrip). In the case of the ADRV9371-N/PCBZ evaluation board, the JESD204B differential signals are routed on inner layers of the board (Layer 5 and Layer 10) as differential 100 Ω pairs (stripline). To minimize potential coupling, these signals are placed on an inner layer using a via embedded in the component footprint pad where the ball connects to the PCB. AC coupling capacitors (100 nF) on these signals are placed at the connector, away from the chip, to minimize coupling. The JESD204B interface can operate at frequencies up to 6.4 GHz. Care must be exercised to maintain signal integrity from the chip to the connector.

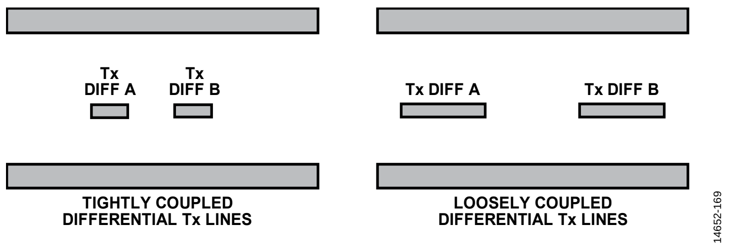

Tx DIFF A
Tx DIFF B
TIGHTLY COUPLED DIFFERENTIAL Tx LINES
Tx DIFF A
Tx DIFF B
LOOSELY COUPLED DIFFERENTIAL Tx LINES
14652-169

Figure 169. JESD204B Differential Pair Routing Example

ISOLATION TECHNIQUES USED ON THE ADRV9371-N/PCBZ EVALUATION BOARD

The device was designed to provide extremely good channel isolation. Significant isolation challenges must be overcome while designing the ADRV9371-N/PCBZ evaluation board. The following isolation requirements were followed to accurately evaluate transceiver performance:

- Tx to Tx: 80 dB out to 6 GHz
- Tx to Rx: 80 dB out to 6 GHz
- Rx to Rx: 60 dB out to 6 GHz
- ORx to ORx: 60 dB out to 6 GHz

To meet those goals with significant margin, isolation structures were introduced.

Figure 170 shows the isolation structures used on the ADRV9371-N/PCBZ evaluation card. These structures consist of a combination of slots and square apertures. Both structures are present on every copper layer of the PCB stack. The advantage of using square apertures is that signals can be routed between the openings without disturbing the isolation benefits that the array of apertures provides.

When utilizing the proposed isolating structures, it is important to place ground vias around the slots and apertures.

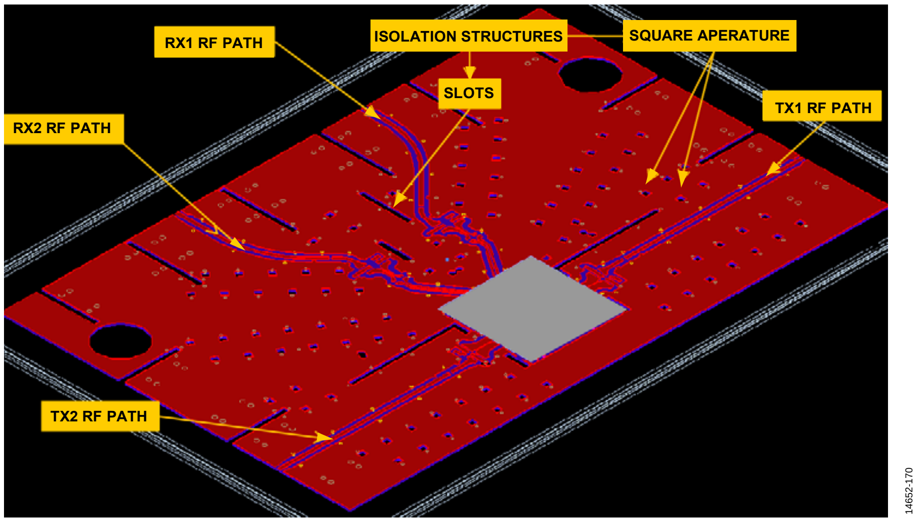


Figure 170. Isolation Structures on the ADRV9371-N/PCBZ Evaluation Board

The methodology used on the ADRV9371-N/PCBZ evaluation board is shown in Figure 171. When slots are used, place ground vias at each end of the slots and along each side. When square apertures are used, place at least one single ground via adjacent to each square. It is recommended that these vias be throughhole vias connecting the top to the bottom layer and all layers in between. The function of these vias is to steer return current to the ground planes near the apertures.

It is recommended to use electromagnetic simulation software to develop accurate slot spacing and square aperture layout when designing a PCB for an AD9371 family transceiver. Ensure that spacing between square apertures is not more than 1/10 of the shortest wavelength supported.

The wavelength can be calculated using Equation 40.

$$wavelength(\text{m}) = \frac{300}{frequency(\text{MHz}) \times \sqrt{\varepsilon_r}} \quad (40)$$

where ε_r is the dielectric constant of the isolator material.

For Roger 4003C material, microstrip structure (and taking in account air as an insulator), $\varepsilon_r = 3.55$.

For FR4-370 HR material, stripline structure, $\varepsilon_r = 4.6$.

For example, the following:

- Maximum RF signals frequency is 6 GHz.
- For Rogers 4003C material (and taking in account air as an insulator), using microstrip structures, and $\varepsilon_r = 3.55$, the minimum wavelength is approximately 26.5 mm.

To fulfill the 1/10 of a wavelength rule, square aperture spacing must be at a distance of 2.65 mm or closer.

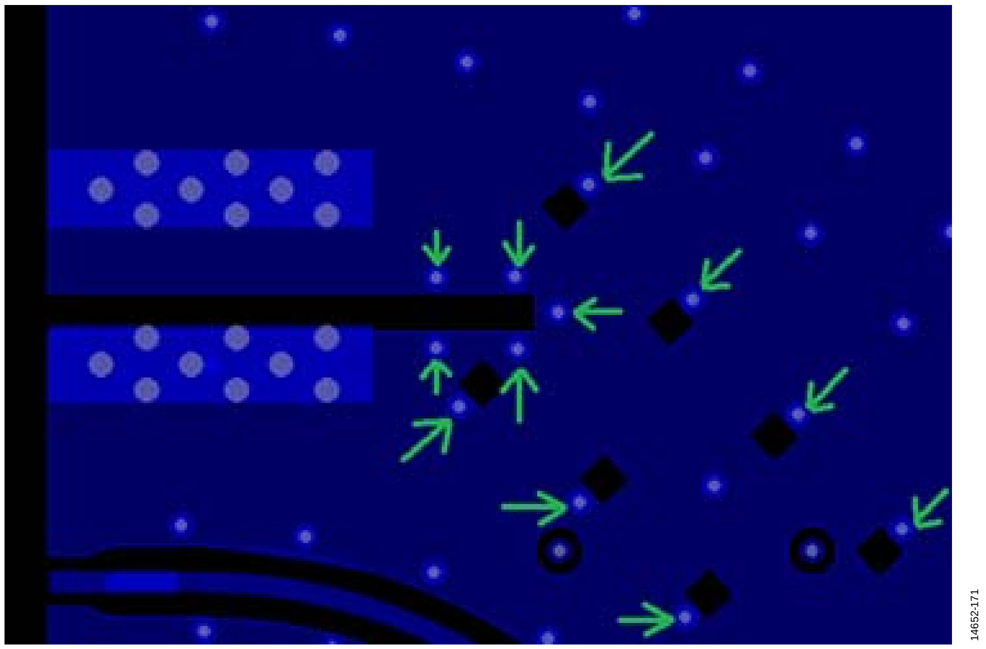


Figure 171. Current Steering Vias Placed Near Isolation Slots and Apertures

Isolation Between JESD204B Lines

The JESD204B interface uses eight line pairs that can operate at speeds of up to 6.4 GHz. Care must be taken when doing PCB layout to ensure those lines are routed following the rules described in the JESD204B Trace Routing Recommendations section. In addition, use isolation techniques to prevent crosstalk between different JESD204B lane pairs. A technique used on the ADRV9371-N/PCBZ evaluation board uses via fencing. Figure 172 illustrates this technique. Ground vias placed around each JESD204B pair provide isolation and decrease crosstalk. Spacing between vias follows the rule provided in Equation 40. JESD204B lines are routed on Layer 5 and Layer 10 so that they utilizes stripline structures. The dielectric material used in the inner layers of the ADRV9371-N/PCBZ evaluation board PCB is FR4-370 HR.

For accurate spacing of JESD204B fencing vias, use layout simulation software. Use Equation 40 with the following outlined details:

- Maximum JESD204B signal frequency is around 6.4 GHz.
- For FR4-370 HR material, stripline structure, and ε_r = 4.6, the minimum wavelength is approximately 21.9 mm.

To fulfill the 1/10 wavelength spacing rule, use vias spaced at a distance of 2.19 mm or closer.

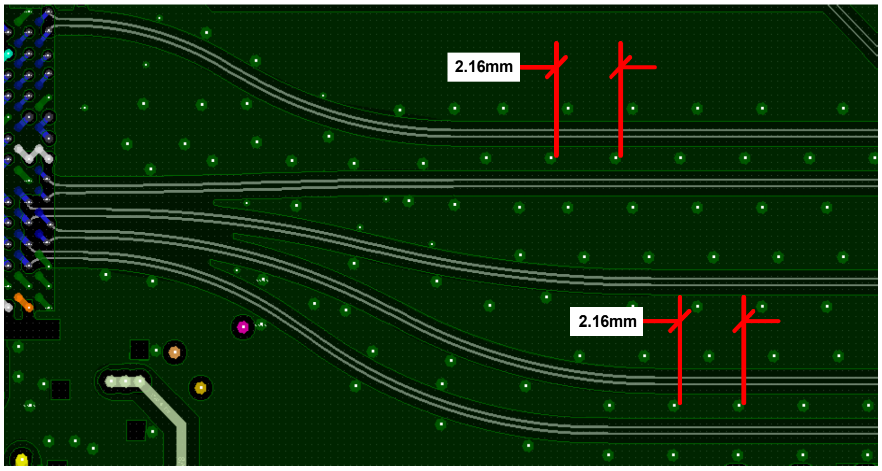


Figure 172. Via Fencing Shield Around JESD204B Lines (Layer 10 of the ADRV9371-N/PCBZ Shown)

UNUSED BALLS

In some end applications, the user may decide not to use all available inputs or outputs. In these cases, ensure that unused pins follow the recommendations outlined in Table 184.

Table 184. Recommendations for Unused Balls

Pin No.	Type	Mnemonic	When Pins Are Not Used
A9, A10, A5, A6	I	RX1+, RX1−, RX2+, RX2−	Do not connect. When active, there is a 0.7 V_{BIAS}. When disabled, the internal protection diodes protect the inputs.
A12, A13, A2, A3	I	ORX1+, ORX1−, ORX2+, ORX2−	Do not connect. When active, there is a 0.7 V_{BIAS}. When disabled, the internal protection diodes protect the inputs.
D4, E4, D3, E3, D2, E2	I	SNRXA−, SNRXA+, SNRXB−, SNRXB+, SNRXC−, SNRXC+	Connect to GND with a 1 kΩ pull-down resistor or directly to GND. Note that when active there is a bias voltage on those inputs.
H14, J14, H1, J1	O	TX1+, TX1−, TX2−, TX2+	Do not connect.
B7, B8	I/O	RX_EXTLO−, RX_EXTLO+	Do not connect.
E11, E12	I/O	TX_EXTLO−, TX_EXTLO+	Do not connect.
M5, M7, M6, M8	I	RX1_ENABLE, RX2_ENABLE, TX1_ENABLE, TX2_ENABLE	Connect to GND with a 1 kΩ pull-down resistor or directly to GND.
E13, C11, C12, D11	I	AUXADC_0, AUXADC_1, AUXADC_2, AUXADC_3	Connect to GND with a 1 kΩ pull-down resistor or directly to GND.
H11, H12, J3, J7, J8, J11, J12, K5 to K8, K11, K12, L5, L6, L11, L12, M10, M11	I/O	GPIO_0 to GPIO_18	Because these pins contain an input stage, the voltage on the pin must be controlled. They can be tied to ground through a 1 kΩ resistor (to safeguard against misconfiguration), or they can be left floating, programmed as outputs, and driven low.
C1, C2, C13, D1, D5, D12, D13, D14, E1, E14, F1, F14	I/O	GPIO_3P3_0 to GPIO_3P3_11	Because these pins contain an input stage, the voltage on the pin must be controlled. They can be tied to ground through a 1 kΩ resistor (to safeguard against misconfiguration), or they can be left floating, programmed as outputs, and driven low.
J5	O	GP_INTERRUPT	Do not connect.
J6	I	TEST	Connect to GND.
J10	O	SDO	In SPI, 3-wire mode, do not connect.
M3, M4, L3, L4	I	SYNCINB0−, SYNCINB0+, SYNCINB1−, SYNCINB1+	Connect to GND with a 1 kΩ pull-down resistor or directly to GND.
M13, M14	O	SYNCOUTB0+, SYNCOUTB0−	Do not connect.
P11, P12, P13, P14, N10, N11, N12, N13	I	SERDIN0−, SERDIN0+, SERDIN1−, SERDIN1+, SERDIN2−, SERDIN2+, SERDIN3−, SERDIN3+	Do not connect. These pins have a bias on them; therefore, they must not be tied to power or GND.
P6, P7, P4, P5, N5, N6, N3, N4	O	SERDOUT0−, SERDOUT0+, SERDOUT1−, SERDOUT1+, SERDOUT2−, SERDOUT2+, SERDOUT3−, SERDOUT3+	Do not connect. These pins have a bias on them; therefore, they must not be tied to power or GND.

POWER MANAGEMENT CONSIDERATIONS

The device has six different power supply domains:

- 1.3 V is the main analog domain that powers the major part of the chip, which is divided into various 1.3 V domains, all with a tolerance of ±2.5%.
- The JESD_VTT_DES and VDDA_SER supplies are 1.3 V. These can be powered from the main analog core voltage, if desired, as long as appropriate isolation is used. It is suggested that for best performance the JESD_VTT_DES and VDDA_SER supplies come from a separate regulator so that they can be varied from 1.3 V to 1.2 V to adhere to the JESD204B specification. Both require a tolerance of ±5%.
- The VDIG supply is the main digital power supply. It must be kept separate from the main 1.3 V analog supply to minimize digital noise coupling into analog circuits. The tolerance for this supply is ±2.5%.
- The VDD_IF supply is a separate power domain. The nominal input voltage on the VDD_IF can range from 1.8 V to 2.5 V. This voltage controls the voltage levels of the digital interface (SPI and control signals). It has a tolerance of ±5%.
- The VDDA_3P3 supply is a 3.3 V domain. This supply provides a higher voltage rail for GPIO_3P3s, receiver mixer switches, auxiliary DACs, and the auxiliary ADC; therefore, it is required whether the GPIO_3P3s are used or not. It has a tolerance of ±5%.
- The VDDA_1P8 supplies the Tx output section and is applied to the balun center taps. In the scenario where baluns do not have the dc feed capability, RF chokes can be used. They must be connected from this supply to each Tx output. This domain has a tolerance of ±5%.

POWER SUPPLY SEQUENCE

The device requires a specific power-up sequence to avoid undesired power-up currents. In the optimal power-up sequence, the VDIG and the VDDA supplies (all 1.3 V domains) come up first and simultaneously. If they cannot be brought up simultaneously, the VDIG supply must come up first. It is recommended to bring up the VDDA_3P3, VDDA1P8, VDD_JESD_VTT, and VDDA_SER supplies after the 1P3 supplies. The VDD_IF supply can be brought up at any time. Note that no device damage occurs if this sequence is not observed; however, this can result in higher than expected power-up currents. It is also recommended to toggle the $\overline{\text{RESET}}$ signal after power has stabilized prior to configuration. The power-down sequence is not critical. If a power-down sequence is followed, the VDIG supply must be removed last to avoid any back biasing of the digital control lines.

POWER DISTRIBUTION FOR DIFFERENT POWER SUPPLY DOMAINS

One key aspect to ensuring good PCB performance is careful low noise power management design. Table 185 lists the pin number, the pin name, the recommended routing technique for that pin from the main 1.3 V analog supply (if applicable), and a brief description of the block it powers in the chip. When routing power traces to the device, follow a star configuration where a separate trace from a common power plane is used to power each 1.3 V power supply pins.

The information listed in Table 185 shows which power supply pins must be powered by designated traces with ferrite bead and which pins are tied together to the power plain using 0 Ω resistors.

The VDDA_SER and JESD_VTT_DES power domains can be connected together and driven by a separate regulator. Noise from this supply can affect the JESD204B link performance directly.

Although the recommendation for VDDA1P3_DES is to keep it separate from the other JESD204B supplies using a separate trace, it is acceptable to power this input from the other 1.3 V analog supply to simplify layout.

Table 185. Power Supply Layout Recommendations (N/A Means Not Applicable)

Pin Name	Pin No.	Type	Voltage (V)	Maximum Current (mA)[1]	Recommended Routing/Notes	Description
Tx Balun or RF Choke DC Feed	N/A	Analog	1.8	240	1.8 V plane, separate trace to common supply point.	1.8 V supply for Tx1
Tx Balun or RF Choke DC Feed	N/A	Analog	1.8	240	1.8 V plane, separate trace to common supply point.	1.8 V supply for Tx2
VDDA_3P3	B14	Analog	3.3	200	To 3.3 V supply (routing typically not critical).	GPIO 3.3 V, auxiliary DAC, auxiliary ADC, RF bias, supply voltage
VDD_IF	M12	Analog	1.8 to 2.5	60	CMOS/LVDS interface supply (routing typically not critical).	Interface pull-up voltage (1.8 V to 2.5 V)
VDDA_1P8	D10	Analog	1.8	20	1.8 V plane, separate trace to common supply point.	1.8 V supply for Tx

Pin Name	Pin No.	Type	Voltage (V)	Maximum Current (mA)[1]	Recommended Routing/Notes	Description
VDIG	L8, L9	Digital	1.3	1700	1.3 V separate supply domain. Use thick trace to separate power domain. Use reservoir capacitors close to the chip.	1.3 V digital core high current
VDDA_RXRF	B1	Analog	1.3	20	Separate trace (using 0 Ω resistor) to 1.3 V analog power plane. Use reservoir capacitors close to the chip.	Sniffer front end only
VDDA_RXTX	F2	Analog	1.3	560	Separate trace (using 0 Ω resistor) to1.3 V analog power plane. Use reservoir capacitors close to the chip.	1.3 V supply for Tx/ORx baseband circuits, TIA/ Tx GM/baseband filters
VDDA_BB	E5	Analog	1.3	670	Separate trace (using 0 Ω resistor) to1.3.V analog power plane. Use reservoir capacitors close to the chip.	Rx ADC, ORx ADC, Tx DAC, auxiliary ADC, REF_CLK
VDDA_RXLO	C6	Analog	1.3	270	1.3 V separate trace (using FB) to common supply point. Very sensitive to aggressors.	1.3 V LO generator for Rx synthesizer, external LO
VDDA_TXLO	F12	Analog	1.3	400	1.3 V separate trace (using FB) to common supply point. Very sensitive to aggressors.	1.3 V LO generator for Tx synthesizer, buffers, external LO
VDDA_CALLPLL	G4	Analog	1.3	230	1.3 V separate trace (using FB) to common supply point. Very sensitive to aggressors.	1.3 V LO generator for calibration PLL synthesizer
VDDA_RXSYNTH	G9	Analog	1.3	12	1.3 V separate trace (using FB) to common supply point. Very sensitive to aggressors.	Rx synthesizer supply
VDDA_TXSYNTH	G8	Analog	1.3	12	1.3 V separate trace (using FB) to common supply point. Very sensitive to aggressors.	Tx synthesizer supply
VDDA_SNRXSYNTH	G7	Analog	1.3	12	1.3 V separate trace (using FB) to common supply point. Very sensitive to aggressors.	ORx synthesizer supply
VDDA_CLKSYNTH	G6	Analog	1.3	12	1.3 V separate trace (using FB) to common supply point. Very sensitive to aggressors.	Clock synthesizer supply
VDDA_SNRXVCO	C4	Analog	1.3	340	1.3 V separate trace (using FB) to common supply point. Very sensitive to aggressors.	SnRx PLL LDO, LO, buffers
VDDA_CLK	N1	Analog	1.3	270	1.3 V separate trace (using FB) to common supply point. Very sensitive to aggressors.	Clock LDO
VRX_VCO_LDO	C8	Analog	1.1	N/A	1 µF bypass close to chip.	1.1 V VCO supply, decouple with 1 µF
VTX_VCO_LDO	F13	Analog	1.1	N/A	1 µF bypass close to chip.	1.1 V VCO supply, decouple with 1 µF
VSNRX_VCO_LDO	C3	Analog	1.1	N/A	1 µF bypass close to chip.	1.1 V VCO supply, decouple with 1 µF
VCLK_VCO_LDO	M1	Analog	1.1	N/A	1 µF bypass close to chip.	1.1 V VCO supply, decouple with 1 µF
VDDA_RXVCO	C7	Analog	1.3	85	1.3 V separate trace (using FB) to common supply point. Very sensitive to aggressors.	Rx PLL LDO
VDDA_TXVCO	F11	Analog	1.3	85	1.3 V separate trace (using FB) to common supply point. Very sensitive to aggressors.	Tx PLL LDO
VDDA_SER	N8	Analog	1.2 to 1.3	120	Connect to P8, P9 and to 1.3 V. Use separate trace (using FB) to common supply point. Use reservoir capacitors close to the chip.	JESD204B VTT signal for serializer
VDDA_SER	P8	Analog	1.2 to 1.3	120	Connect to N8, P9 and to 1.3 V. Use separate trace (using FB) to common supply point. Use reservoir capacitors close to the chip.	JESD204B VTT signal for serializer
JESD_VTT_DES	P9	Analog	1.2 to 1.3	60	Connect to N8, P8 and to 1.3 V. Use separate trace (using FB) to common supply point. Use reservoir capacitors close to the chip.	JESD204B VTT signal for deserializer
VDDA_DES	N9	Analog	1.3	240	1.3 V separate trace (using FB) to common supply point. Use reservoir capacitors close to the chip.	1.3 V supply for JESD204B deserializer

[1] Maximum current is used for sizing voltage regulators not for calculating power consumption, which is heavily dependent on operating conditions.

ADRV9371-N/PCBZ EVALUATION BOARD POWER SUPPLY BLOCK DIAGRAM

The diagram in Figure 173 outlines the power supply configuration used on the ADRV9371-N/PCBZ evaluation board. This configuration follows recommendations outlined in Table 185. The ADRV9371-N/PCBZ evaluation board supports the recommended power-up sequence. The open drain input/output (I/O)[1], open drain I/O[2], and open drain I/O[3] signals allow the user to implement external control over power-up and power-down sequencing as described in the Power Supply Sequence section.

The ADP5054 contains four switch mode, step down regulators. Each of those regulators produces a different power domain that supplies power to the device. Power signals to the device are further isolates using high current ferrite beads. The device uses sense line to monitor the voltage output after the ferrite bead. This approach ensures that the voltage drop resulting from the FB resistance is taken into account, and that the voltage level delivered is in line with expected accuracy.

The power trace connections to the device shown in Figure 173 are made using four different devices:

- High current ferrite beads (FB1)
- Medium current ferrite beads with better RF rejection (FB2)
- Low current ferrite beads with high dc resistance, best RF rejection (FB3)
- 0 Ω resistors

The use of each 0 Ω resistor accomplishes two goals:

- It serves as a placeholder for a ferrite bead in cases where the user encounters noise problems and more isolation is required. For more details regarding ferrite bead selection, refer to the RF and Clock Synthesizer Supplies section.
- It ensures that the layout engineer follows the power routing advice outlined in the Signals with Second Routing Priority section. Resistor placeholders in series force the use of separate traces to deliver different power domains to the device.

For more details on exact power supply implementation, refer to the schematic of the ADRV9371-N/PCBZ evaluation board.

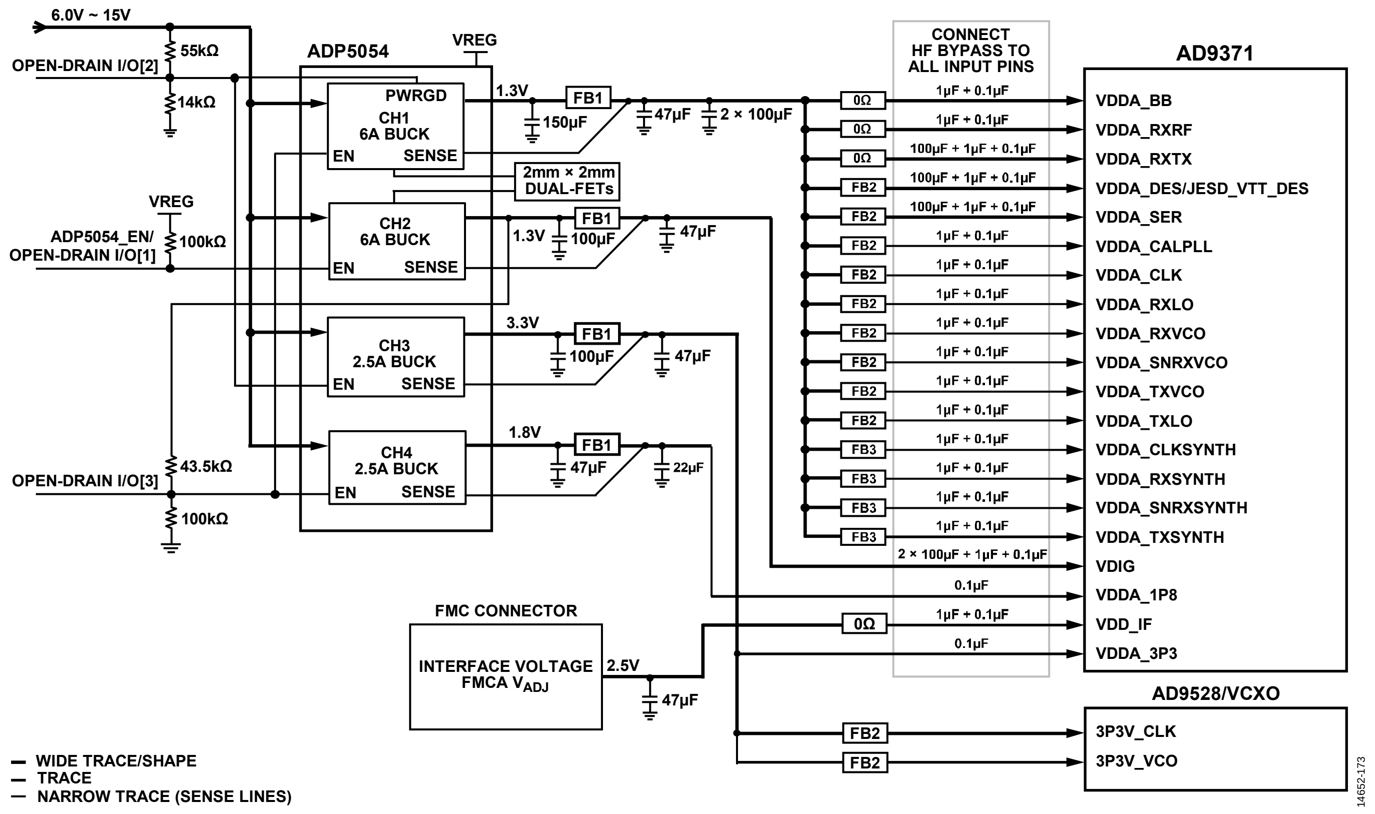


Figure 173. Power Supply Connection Block Diagram of ADRV9371-N/PCBZ Evaluation Card

RF AND CLOCK SYNTHESIZER SUPPLIES

The noise performance of the power domain used to power the RF blocks directly affects the phase noise. The following pins can be powered from a single power supply using a star configuration where each domain is separated using 0 Ω resistors:

- VDDA_RXRF
- VDDA_RXTX
- VDDA_BB

Those domains must have a minimum 100 µF capacitor placed near the device to help mitigate effects of transients on the 1.3 V analog supply.

It is recommended that the following power domains be powered using separate traces with extra isolation using a low DCR ferrite bead, such as the Murata BLM15AX300SN1D or a similar device:

- VDDA_DES
- VDDA_SER
- VDDA_CALPLL
- VDDA_CLK
- VDDA_RXLO
- VDDA_RXVCO
- VDDA_SNRXVCO
- VDDA_TXVCO
- VDDA_TXLO

The power supply noise rejection on the synthesizer power input pins is very low, meaning that any noise ripple on these pins affects the synthesizer performance. The RF synthesizer requires more critical supply decoupling because any noise or variation in voltage that occurs during operation is directly imposed on the RF channel. Refer to Figure 173 for an example of how the power supply connections are made on the ADRV9371-N/PCBZ evaluation board.

The synthesizers are more susceptible to low frequency noise than other supplies because they have programmable loop filters. The loop filter bandwidth defaults are 50 kHz for the Rx, Tx, and ORx synthesizers, and 350 kHz for the clock synthesizer. The loop filter bandwidth directly affects the supply noise rejection on the synthesizers. For example, if the loop filter bandwidth is 50 kHz, any noise on the supply less than 50 kHz is not filtered. The roll-off of the loop filter provides a noise rejection of more than 50 kHz.

For each of the following power domains listed, a ferrite bead with high isolation at the frequency of operation is recommended to help isolate the pin from the supply source for best performance, which is especially important when operating in time division duplexed (TDD) mode. Such high isolation ferrite beads tend to also have high dc resistance. This trade-off is acceptable for the synthesizer power inputs because their low current draws result in relatively small voltage drops that are well within the supply tolerance range.

It is recommended that the following power domains be powered using a separate trace with extra isolation using a high rejection ferrite bead such as the Taiyo Yuden BK1005LL470-T or a similar device:

- VDDA_CLKSYNTH
- VDDA_RXSYNTH
- VDDA_SNRXSYNTH
- VDDA_TXSYNTH

DEMONSTRATION SYSTEM OVERVIEW

The demonstration system enables users to evaluate the device without having to develop custom software or hardware. The system is comprised of a radio daughtercard, a Xilinx® Zynq® field programmable gate array (FPGA) evaluation platform, an SanDisk® (SD) card with an operating system, a power supply, and a C#-based evaluation software application. The evaluation system uses an Ethernet interface to communicate with the PC.

INITIAL SETUP

The transceiver evaluation software (TES) can run with or without evaluation hardware. When TES runs without the hardware connected, it can be configured for a particular operating mode. If the evaluation hardware is connected, the desired operating parameters can be setup with TES and then the software programs the device evaluation hardware. Once the device is configured, use the evaluation software to transmit waveforms, observe received waveforms, and initiate correction algorithms. A sequence of application programming interface (API) commands in the form of an IronPython script can be generated and executed using the TES.

HARDWARE AND SOFTWARE REQUIREMENTS

The hardware and software require the following:

- The Xilinx ZC706 ZYNQ evaluation platform (not included in the demonstration kit). Both Xilinx platforms, EK-Z7-ZC706 Rev. 1.2 and AES-Z7-JESD3-G Rev. 1.2, are compatible with the device demonstration system kit.
- The device demonstration system kit.
- The operating system on the controlling PC must be Windows® Vista SP2 (x86 or x64) or Windows 7 SP1 (x86 or x64).
- The PC must have a free Ethernet port with the following constraints:
    - If the Ethernet port is occupied by another LAN connection, use a USB to Ethernet adapter.
    - Using a dedicated Ethernet connection, the PC can access the following ports:
        - 22—SSH protocol
        - 55555—access to the evaluation software on the ZYNQ platform
        - TES—available on the RadioVerse landing page (users must have PC administrative privileges).

Hardware Kit

The device demonstration system hardware kit contains the following:

- The evaluation board is a daughter card. There are two versions: the ADRV9371-N/PCBZ for a narrow tuning range, and the ADRV9371-N/PCBZ for tuning across the entire range. Both operate with no differences with the TES. The ADRV9371-N/PCBZ is used for all descriptions in this user guide.
- Two SD cards containing the files for the Xilinx ZC706 motherboard. The SD card is 8 GB, Type 4.
- One SD card with the Linux operating system with required Linux-based evaluation software (see the RadioVerse landing page for further information on this).
- One SD card with the Linux operating system and interface needed to operate with the Windows-based TES.

Hardware Setup

The ZYNQ platform setup (see Figure 174) requires the following steps:

1. Place all jumpers in the positions shown in Figure 174.
2. Place the SW11 toggle switches in the positions as shown in Figure 174 (Toggle Switch 1, Toggle Switch 2, and Toggle Switch 5 = Position A.
3. Place the SD card (included with the evaluation kit) in the J30 slot of the ZYNQ board.

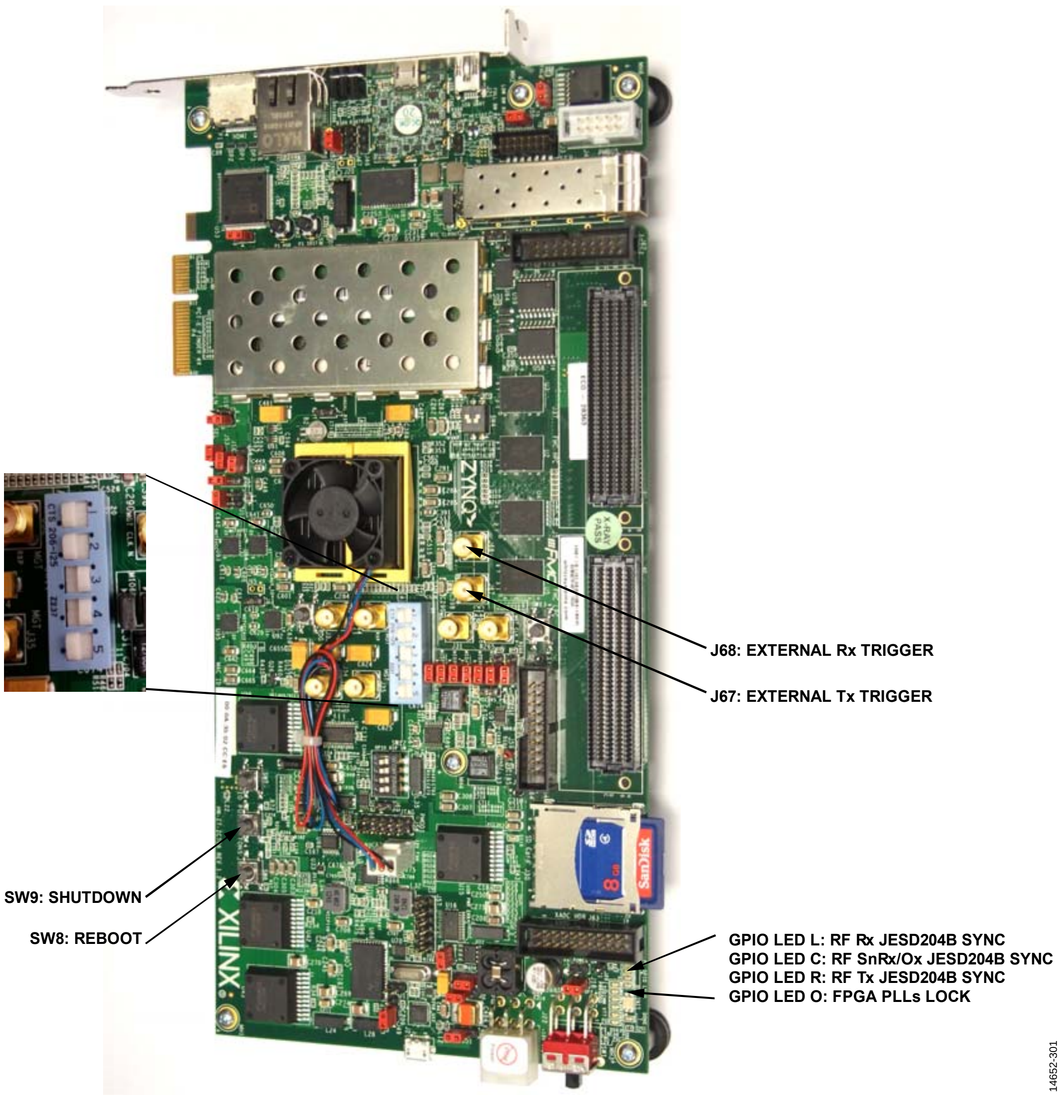

J68: EXTERNAL Rx TRIGGER
J67: EXTERNAL Tx TRIGGER
SW9: SHUTDOWN
SW8: REBOOT
GPIO LED L: RF Rx JESD204B SYNC
GPIO LED C: RF SnRx/Ox JESD204B SYNC
GPIO LED R: RF Tx JESD204B SYNC
GPIO LED O: FPGA PLLs LOCK
14652-301

Figure 174. Xilinx EK-Z7-ZC706 ZYNQ Motherboard with Jumper Settings and Switch Position Configured to Work with the Device Evaluation Board

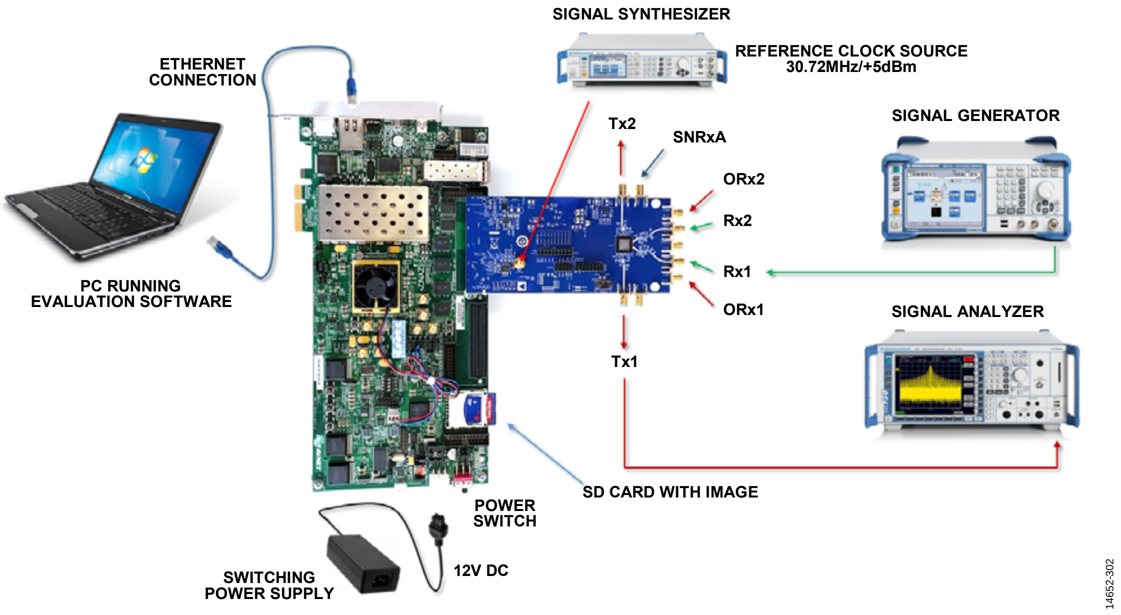


Figure 175. Evaluation Board and Xilinx EK-Z7-ZC706 ZYNQ Motherboard with Connections Required for Channel 1 Transmit and Receive Testing

Do the following to set up the evaluation board for testing:

1. Connect the evaluation board and the ZYNQ evaluation platform together, as shown in Figure 175. Use the HPC FMC connector (J37). Ensure proper alignment of the connectors.
2. Ensure that all jumpers on the ZYNQ motherboard, as well as the SW11 position, match the settings shown in Figure 174 (1, 2, 5 = Position A).
3. Insert the for use with the Windows-based transceiver evaluation software (TES) SD card that came with the device evaluation kit into the ZYNQ SD card slot (J30).
4. Provide a 30.72 MHz clock source (or frequency that matches the setting selected on the AD9528 configuration (**Config**) tab, see Figure 202), at a 5 dBm power level to the J401 connector on the daughter card. This signal drives the reference clock into the AD9528 clock generation chip on the daughter card. Note that the REFA/$\overline{\text{REFA}}$ pins of AD9528 generate the DEV_CLK signal for the device and the REF_CLK signal for the field programmable gate array (FPGA) on the ZYNQ platform.
5. Connect a 12 V, 5 A power supply to the ZYNQ evaluation platform at the J22 header.
6. Connect the ZYNQ evaluation platform to the PC with an Ethernet cable (connect to P3). No driver installation is required. Note the following:
   a. In cases where the Ethernet port is already occupied by another connection, use a USB to Ethernet adapter.
   b. With an Ethernet connection dedicated to the ZYNQ platform, manually set the IPv4 address to 192.168.1.2 and set the IPv4 subnet mask to 255.255.255. See the Instructions to Set the IPv4 Addresses section and Figure 176 to Figure 178 for instructions on setting the IPv4 addresses.
7. Ensure that the following ports on the PC are not blocked by firewall software:

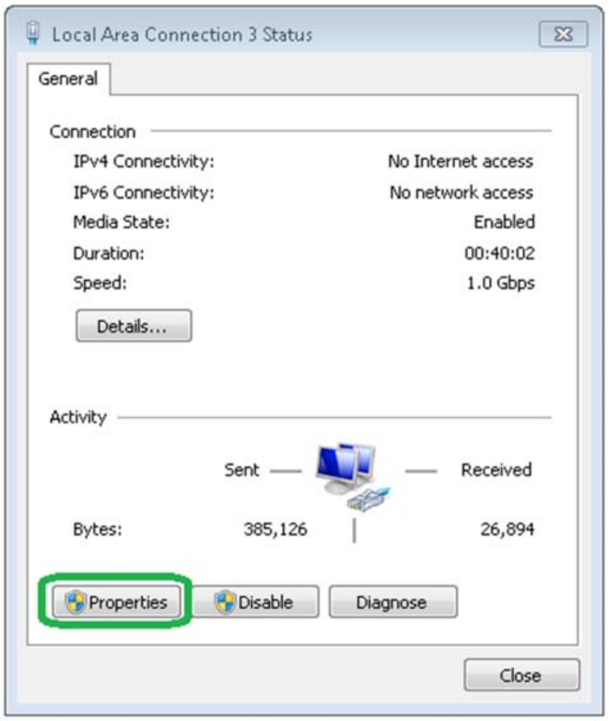
   - 22: SSH protocol
   - 55555: access to the evaluation software on the ZYNQ platform

Note that the ZYNQ IP address is set by default to: 192.168.1.10.

Instructions to Set the IPv4 Addresses

To set the IPv4 addresses as instructed in the evaluation board testing setup (Step 6b), take the following steps:

1. In the Windows **Control Panel**, navigate to **Network Connections** > **Local Area Connection** > **General**, and click **Properties** (see Figure 176).



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

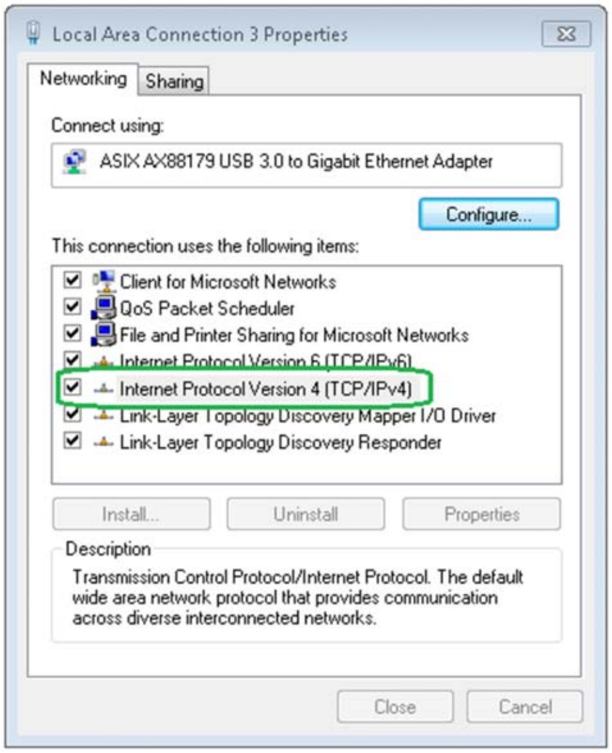
Properties Disable Diagnose

Close

14652-303

*Figure 176. **General** Tab Properties*

2. Select **Networking** and check **Internet Protocol Version 4 (TCP/IPv4)**, as shown in Figure 177.



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

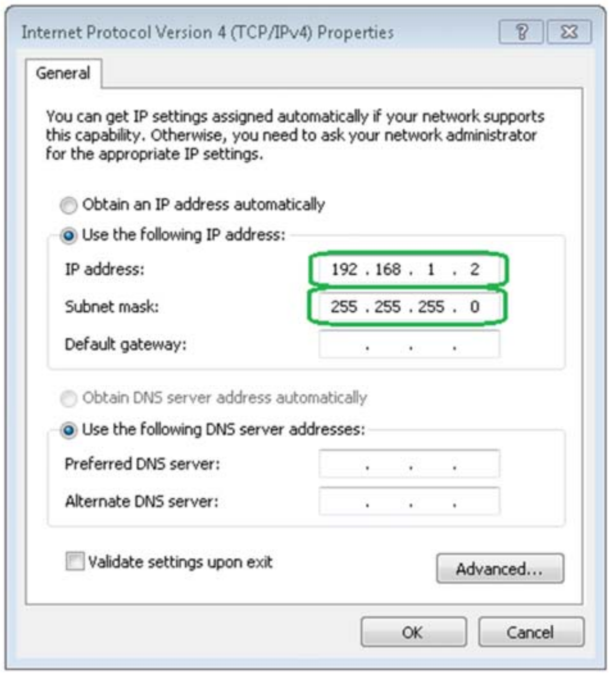
Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

Close Cancel

14652-304

*Figure 177. Select **Internet Protocol Version 4 (TCP/IPv4)***

3. Verify that the appropriate addresses (Step 6b of the evaluation system setup) are listed for the fields, **IP address** and **Subnet mask**, as shown in Figure 178.



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...

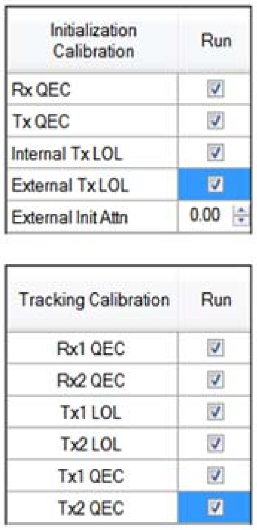
OK Cancel

14652-305

Figure 178. Select the Proper IP Address and Subnet Mask

HARDWARE SETUP FOR EXTERNAL Tx LO LEAKAGE CALIBRATION

The device is a direct conversion transceiver. The Tx baseband dc offset and direct coupling of the local oscillator (LO) to the Tx output can cause an undesired continuous wave (CW) emission at the Tx LO frequency. The purpose of the transmitter local oscillator leakage (LOL) calibration is to minimize this emission. The evaluation system supports two types of Tx LOL calibration algorithms, internal and external. Make Tx LOL calibration algorithm selections in the calibration boxes located in the transceiver evaluation software (TES), see Figure 179 (the full screen capture of this graphical user interface (GUI) is shown in Figure 184).



Initialization Calibration	Run
Rx QEC	✓
Tx QEC	✓
Internal Tx LOL	✓
External Tx LOL	✓
External Init Attn	0.00

Tracking Calibration	Run
Rx1 QEC	✓
Rx2 QEC	✓
Tx1 LOL	✓
Tx2 LOL	✓
Tx1 QEC	✓
Tx2 QEC	✓

Figure 179. Tx LOL Calibration Options

When user selects **Internal Tx LOL**, external hardware is not necessary. Note that, in this case, there is no Tx LOL tracking calibration available.

When the user selects **External Tx LOL** in the **Initial Calibration** list and **External Tx LOL** in the **Tracking Calibration** list as shown in Figure 179, external components must be connected to the evaluation platform for proper operation. Figure 180 shows the proper configuration to demonstrate performance of the Tx LOL calibration algorithm.

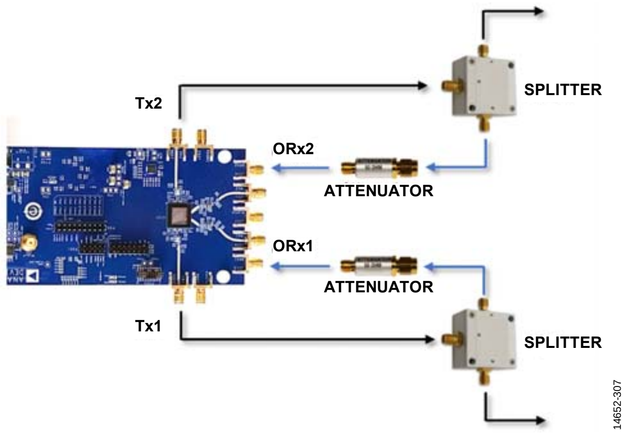


Figure 180. Demonstration Operation of External Tx LOL Calibration on the Device Evaluation System

The user must connect an RF splitter at the Tx output. Connect one of the splitter outputs to the corresponding observation (ORx) inputs through an RF attenuator (Tx1 → ORx1, Tx2 → ORx2). The amount of attenuation needed depends on the attenuation introduced by the splitter. For maximum ORx gain, do not exceed a −16 dBm signal level at the ORx input. Note that for external Tx LOL tracking calibration, both transmitters must loop back to both observation receivers through splitters and attenuators.

For example, for maximum ORx gain with a maximum Tx output signal of 7 dBm (CW), the RF splitter attenuation = 3 dB and the RF attenuator = 20 dB (7 dBm – 3 dB – 20 dB = −16 dBm).

To use the device evaluation system with a power amplifier (PA), either split or couple the signal after the PA and then connect it to the corresponding ORx channel through an appropriate attenuator. Do not exceed −16 dBm maximum signal level at the ORx input. Calculate the amount of attenuation required based on the power level after the PA and RF coupler (see Figure 181).

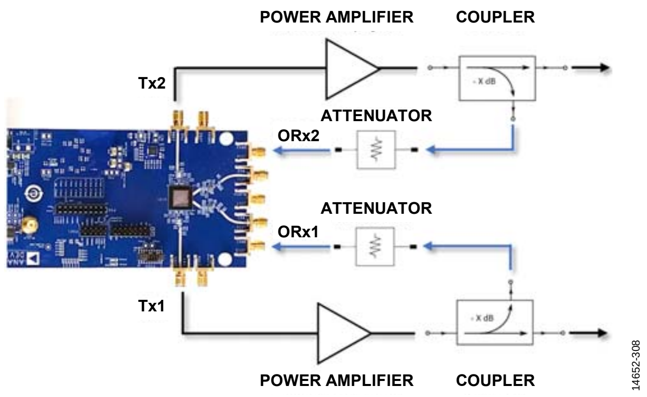


Figure 181. End User Application of External Tx LOL Calibration Using the Device Evaluation System

HARDWARE OPERATION

Turn on the evaluation system by switching the ZYNQ board power switch (SW1) to the on position. Two LEDs (Blue DS701 and Green DS702) on the device evaluation board turn on. If the LEDs do not turn on, it indicates improper hardware connection.

The ZYNQ evaluation system uses a Linux operating system. It takes approximately 30 seconds before the system is ready for operation and can accept commands from PC software. Boot status can be observed on the ZYNQ GPIO LEDs (L, C, R, O) (see Figure 174 for LED locations). Take the following steps for proper operating sequence:

1. Wait approximately 15 sec after SW1 turns on and all four LEDs turn on to allow the image to copy from the SD card into the field programmable gate array (FPGA) memory.
2. Allow another approximately 15 sec for the ZYNQ system to boot up, which is indicated by flashing LEDs. Note that the flashing sequence occurs one LED at a time. When the LEDs cease flashing, the system is ready for normal operation.
3. After the LEDs stop flashing, establish a connection with the PC over the Ethernet (using the transceiver evaluation software (TES)).

   During normal operation, the LED lights indicate system status as defined in Table 186.

Table 186. GPIO LED Status Light Indicators

GPIO LED	Description
L	RF Rx JESD204B sync
C	RF SnRx/ORx JESD204B sync
R	RF Tx JESD204B sync
O	FPGA phase-locked loops (PLLs) lock

4. Connect the reference clock signal (30.72 MHz continuous wave (CW) tone, 5 dBm maximum) to J401.
   a. After using the TES to program the system, the two LEDs on the evaluation board (D401 and D402) are on.
   b. Active LEDs indicate that the correct reference clock is provided and the PLLs in the AD9528 are locked.
5. For receiver testing on the device evaluation board, use a clean signal generator with low phase noise to provide an input signal to the selected RF input. Use a shielded RG-58, 50 Ω coaxial cable (1 m or shorter) to connect the signal generator to the desired RF input. To set the input level, do the following:
   a. To set the input level near full scale of the Rx receiver, set the generator level (for a single-tone signal) to approximately −15 dBm. This level depends on the input frequency and the gain settings through the path. Do not apply the input signal to the Rx input when performing an initialization calibration.
   b. To set the input level near the full scale of the ORx receiver, set the generator level (for a single-tone signal) to approximately −15 dBm. This level depends on the input frequency and the gain settings through the path.
   c. To set the input level near the full scale of the SnRx receiver, set the generator level (for a single-tone signal) to approximately −16 dBm. This level depends on the input frequency and the gain settings through the path.
6. For transmitter testing, connect a spectrum analyzer to either Tx output on the device evaluation board. Use a shielded RG-58, 50 Ω coaxial cable (1 m or shorter) to connect the spectrum analyzer. Terminate both Tx paths into the spectrum analyzers or, if only one Tx is being checked, terminate one Tx path into a spectrum analyzer and terminate the other Tx into 50 Ω. Note that, initial calibrations run on both channels and can take an extended time to complete if a Tx channel is not correctly terminated.
7. Power off must be executed using the TES or the user must power down the ZYNQ system using the SW9 push button (see Figure 174) before the user powers down the evaluation system by switching SW1 off.
8. When shutdown is executed using the TES, the ZYNQ operating system begins the power-down procedure. It takes a few seconds to finish. All four LEDs blinking together indicates that the user can safely power off the system using the SW1 switch on the ZYNQ platform.

Shutdown Caution

The device evaluation system utilizes a Linux operating system. Linux requires time to boot up as well as time for the software to shut down before hardware power-off. To safely shut down the system use the power-off feature in the TES, or press the SW9 button on the ZYNQ platform before physically switching power off by using the SW1 switch. Using any other shutdown procedure risks corrupting the file system on the SD card and causing the evaluation system to stop operating.

To shut down the system, execute one of these options

- Close the TES application (Windows X button) and then select **Switch Zynq Off**.
- Select **Device** > **Shutdown Zynq Platform** in the TES.

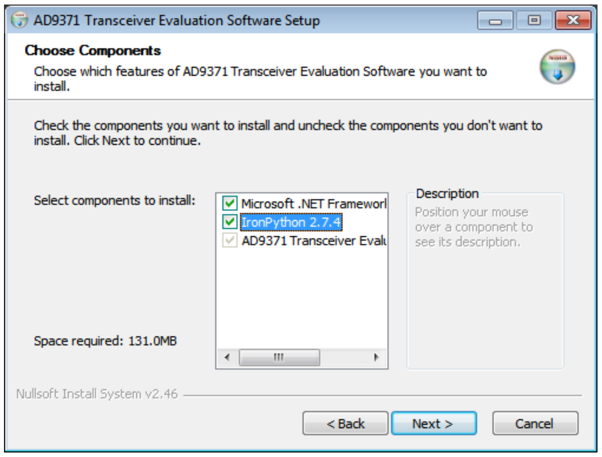
After several seconds, when all four GPIO LEDs on the ZYNQ platform blink together, the user can safely power off the system using the SW1 switch on the ZYNQ platform.

TRANSCEIVER EVALUATION SOFTWARE

INSTALLATION

Download the transceiver evaluation software (TES) directly from the design center landing page. After the initial software download, copy the software to the target system and unzip the files (if not already unzipped). The downloaded zip container contains an executable file, **AD9371 Transceiver Evaluation Software.exe**. Note that the same process is followed when using the AD9375 evaluation board (or any other compatible variant).

PC administrator privileges are required to install TES. After running an executable file, the standard installation process follows. Portions 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 182 shows the recommended configuration for installation.


AD9371 Transceiver Evaluation Software Setup

Choose Components

Choose which features of AD9371 Transceiver Evaluation Software 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 Framewor
- [✓] IronPython 2.7.4
- [✓] AD9371 Transceiver Evalu

Description
Position your mouse over a component to see its description.

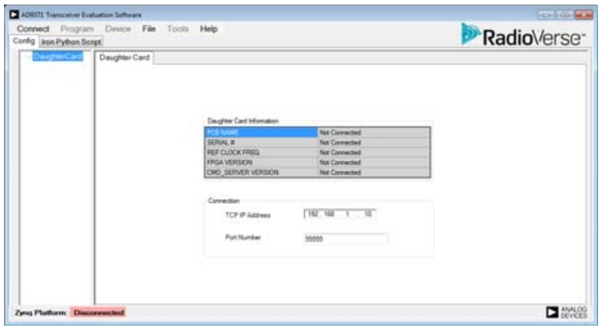
Space required: 131.0MB

Nullsoft Install System v2.46

Figure 182. Software Installation Components

STARTING THE TRANSCEIVER EVALUATION SOFTWARE (TES)

Click **Start** > **All Programs** > **Analog Devices** > **AD9371 Transceiver Evaluation Software** > **AD9371 Transceiver Evaluation Software** to start the software. Figure 183 shows the opening page of the TES after it activates.


AD9371 Transceiver Evaluation Software

Daughter Card Information

PCB NAME	Not Connected
SERIAL #	Not Connected
REF CLOCK FREQ	Not Connected
FPGA VERSION	Not Connected
CMD_SERVER VERSION	Not Connected

Connection
TCP IP Address: 192 . 168 . 1 . 10
Port Number: 55555

Zynq Platform: Disconnected

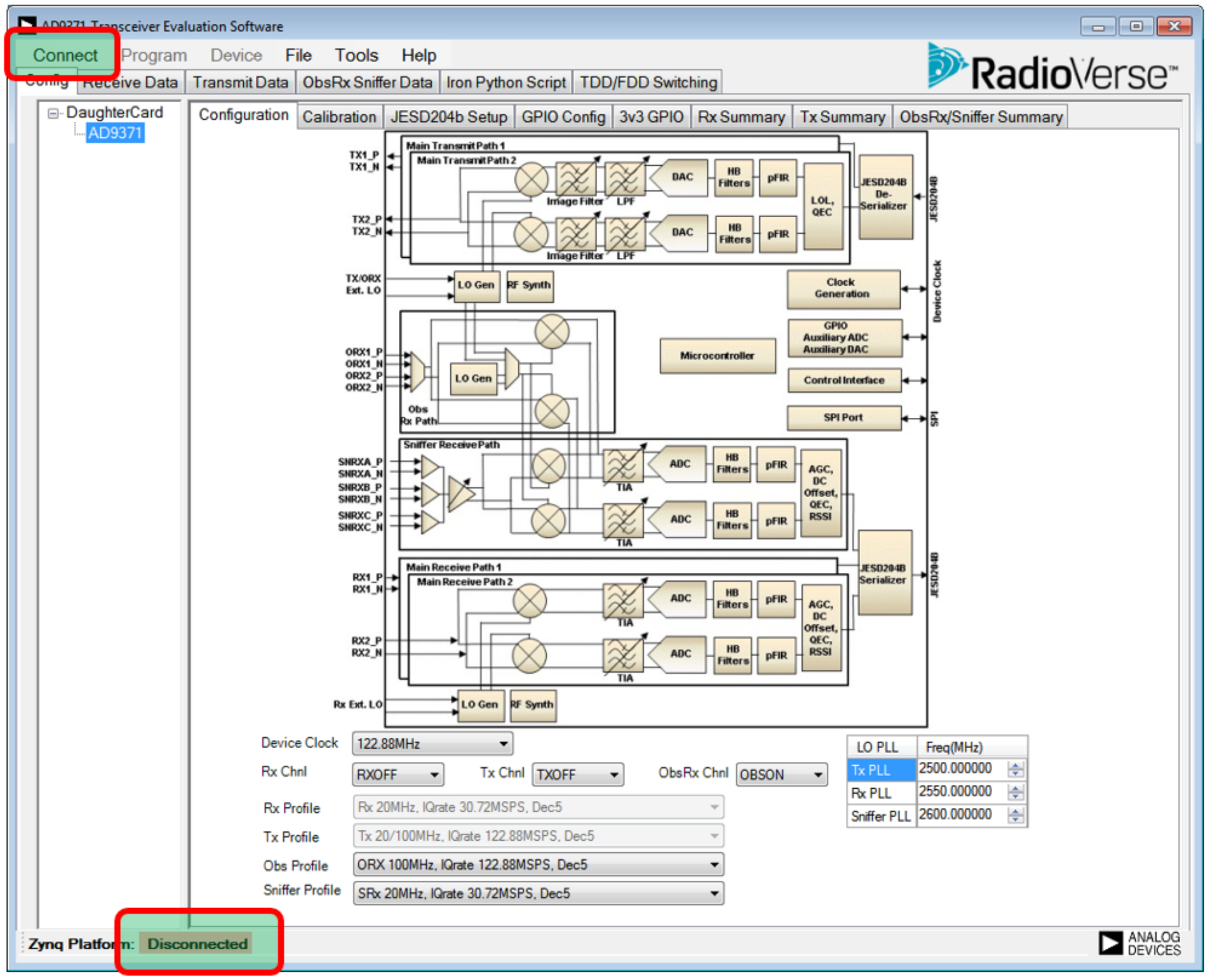
Figure 183. TES Interface

When the evaluation hardware is not connected, the software can still be used in demonstration mode by following these steps:

1. Click **Connect** (see Figure 184) in the TES.
2. **The Zynq board is disconnected** message will appear, then click **OK**.

After clicking OK, the software automatically enters demonstration mode in which a subset of all the features display.

Indication of the connection status is shown at the bottom of the **Zynq Platform** software window. When **Disconnected** is the status display, the TES operates in demonstration mode.



Device Clock 122.88MHz

Rx Chnl RXOFF Tx Chnl TXOFF ObsRx Chnl OBSON

Rx Profile Rx 20MHz, IQrate 30.72MSPS, Dec5

Tx Profile Tx 20/100MHz, IQrate 122.88MSPS, Dec5

Obs Profile ORX 100MHz, IQrate 122.88MSPS, Dec5

Sniffer Profile SRx 20MHz, IQrate 30.72MSPS, Dec5

LO PLL	Freq(MHz)
Tx PLL	2500.000000
Rx PLL	2550.000000
Sniffer PLL	2600.000000

Figure 184. Transceiver Evaluation Software (TES) Project Setup Page

NORMAL OPERATION

When the hardware is connected to the PC and the user wants to use the complete evaluation system, the transceiver evaluation software (TES) will establish a connection with the ZYNQ system via the Ethernet cable after the **Connect** button is clicked. When proper connection is established, click **DaughterCard** in the device tree on the left side of the window (see Figure 185).

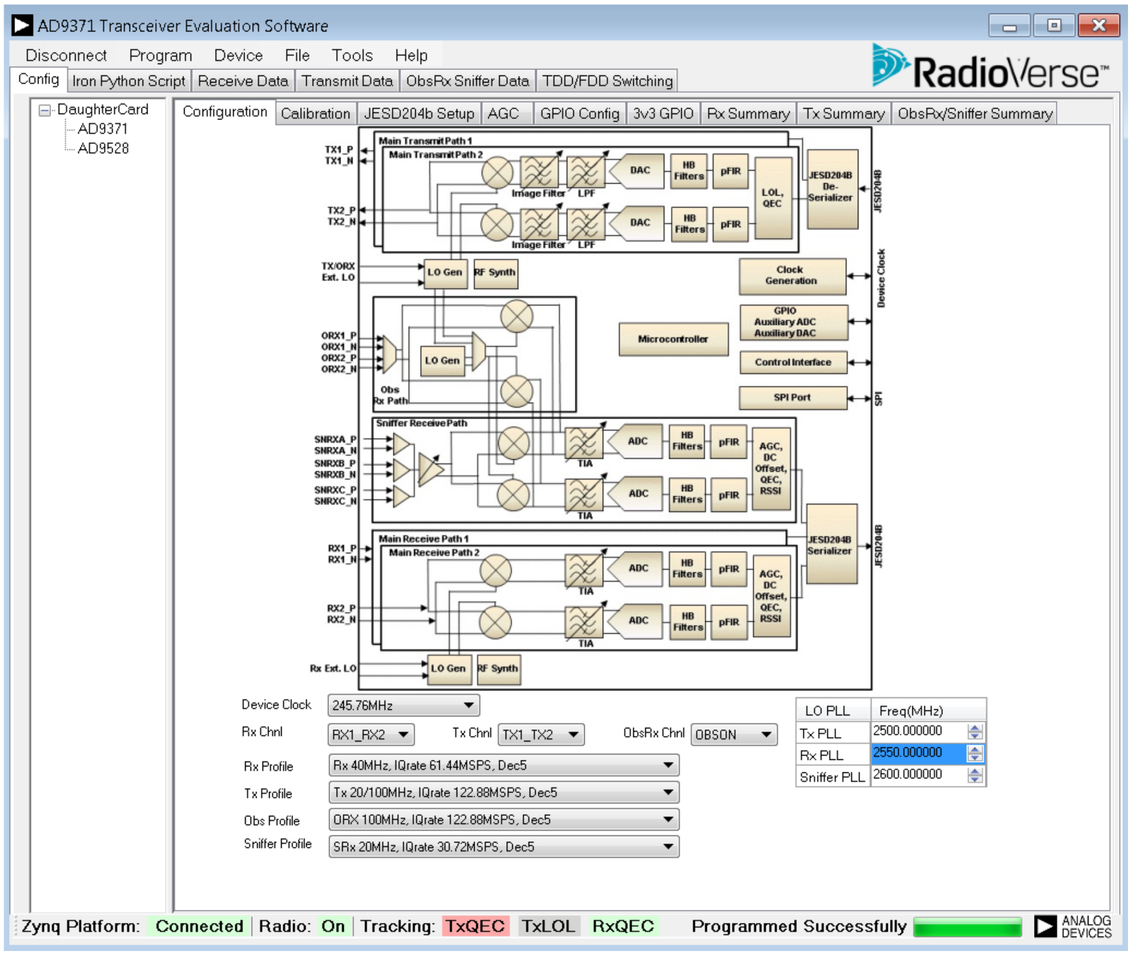
Once **DaughterCard** is selected, information about the revisions of the different setup blocks appear in the main window. This window shows the TCP IP address set to 192.168.1.10, and the port number set to 55555. Figure 185 shows an example of the correct connection between a PC and the ZYNQ system with a daughter card connected to them.

Software Update

Before continuing, determine if the latest version of software is installed by checking for updates on the landing page of the design center at www.analog.com/ad9371-evaluation-software.

Typically, when performing a TES update, platform files must also be updated. To perform a platform files update, select **Device** > **Update** > **Platform Files**. The TES automatically updates files on the ZYNQ SD card and reboots the evaluation system.

After installation of all updates, the system is ready for normal operation.



Device Clock: 245.76MHz

Rx Chnl: RX1_RX2 | Tx Chnl: TX1_TX2 | ObsRx Chnl: OBSON

Rx Profile: Rx 40MHz, IQrate 61.44MSPS, Dec5

Tx Profile: Tx 20/100MHz, IQrate 122.88MSPS, Dec5

Obs Profile: ORX 100MHz, IQrate 122.88MSPS, Dec5

Sniffer Profile: SRx 20MHz, IQrate 30.72MSPS, Dec5

LO PLL	Freq(MHz)
Tx PLL	2500.000000
Rx PLL	2550.000000
Sniffer PLL	2600.000000

Zynq Platform: Connected | Radio: On | Tracking: TxQEC TxLOL RxQEC | Programmed Successfully

Figure 185. Setup Revision Information

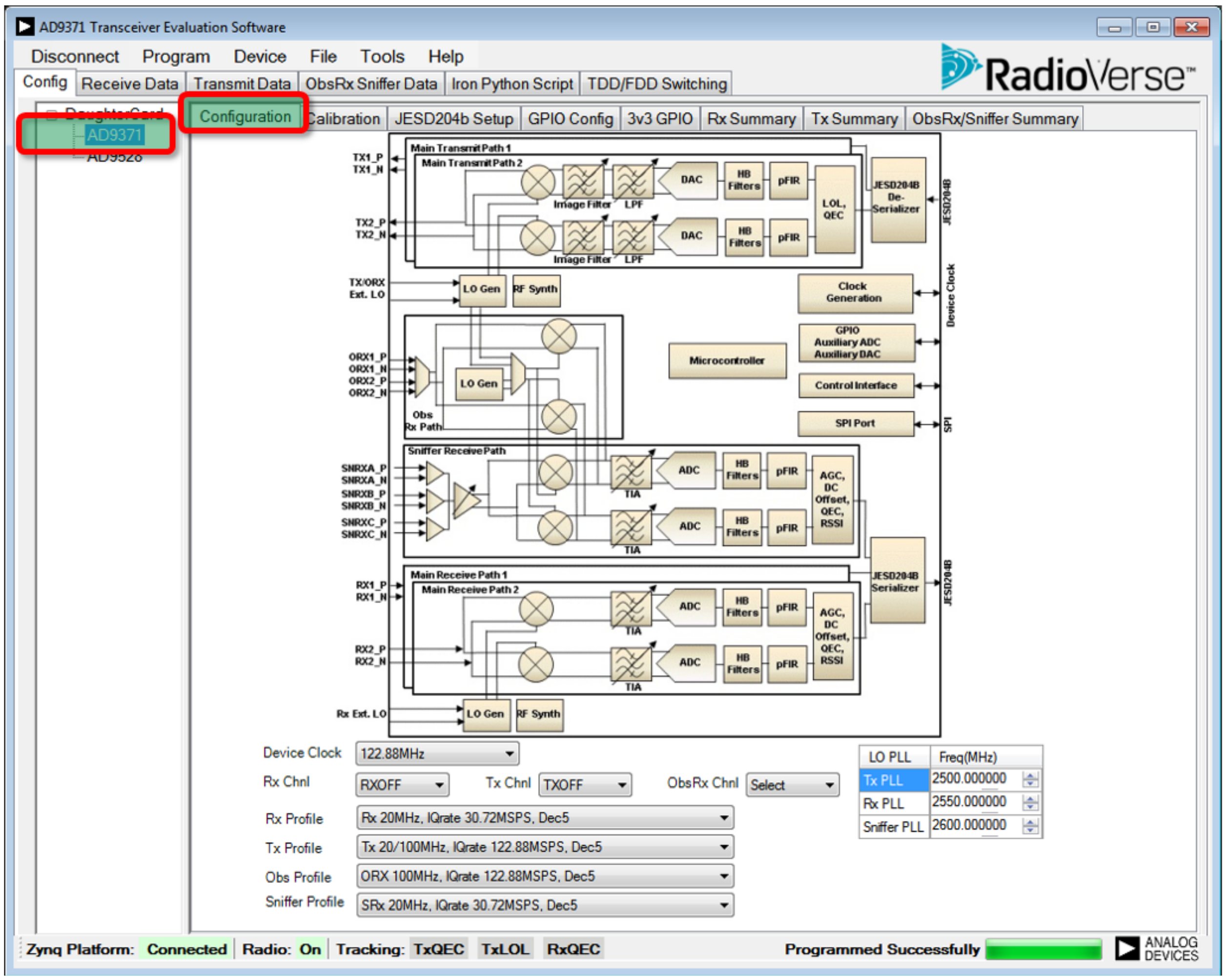
CONFIGURING THE AD9371

The transceiver evaluation software (TES) contains four main user configurable pages (see Figure 186 through Figure 188 and Figure 202). After the user selects the AD9371 in the device tree, the **Config** tab activates. Contained within this tab are eight subtabs that contain setup options for the device.

Configuration Tab

The **Configuration** tab is the first tab within the **Config** tab (see Figure 186). The following selections are available within the **Configuration** tab:

- Device clock frequency
- Number of active Rx channels
- Number of active Tx channels
- Observation/sniffer input
- Rx, Tx, ORx (Obs), and sniffer profiles
- Rx, Tx, and SnRx/ORx LO PLL



Device Clock: 122.88MHz

Rx Chnl: RXOFF Tx Chnl: TXOFF ObsRx Chnl: Select

Rx Profile: Rx 20MHz, IQrate 30.72MSPS, Dec5

Tx Profile: Tx 20/100MHz, IQrate 122.88MSPS, Dec5

Obs Profile: ORX 100MHz, IQrate 122.88MSPS, Dec5

Sniffer Profile: SRx 20MHz, IQrate 30.72MSPS, Dec5

LO PLL	Freq(MHz)
Tx PLL	2500.000000
Rx PLL	2550.000000
Sniffer PLL	2600.000000

Zynq Platform: Connected | Radio: On | Tracking: TxQEC TxLOL RxQEC   Programmed Successfully

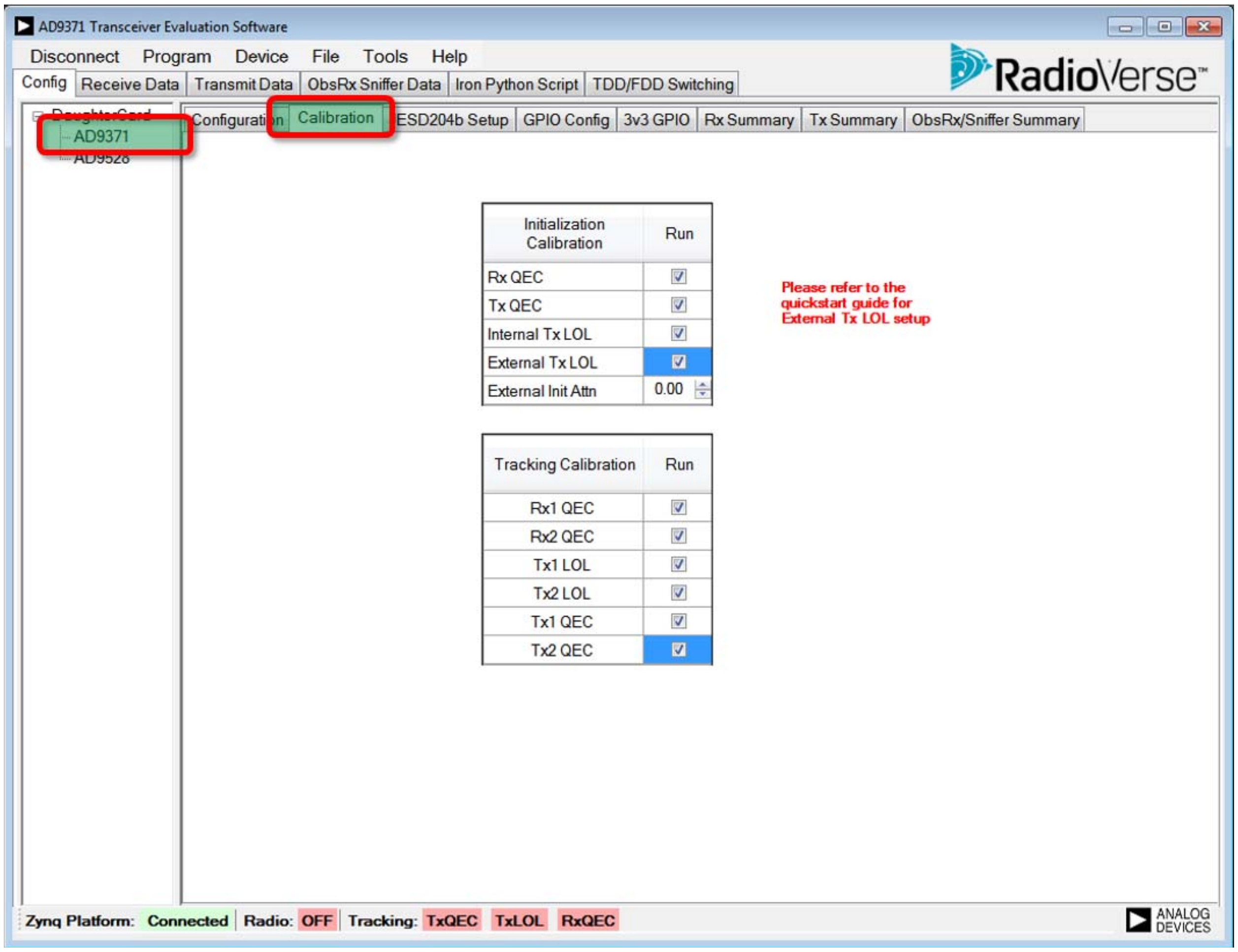
14652-313

Figure 186. Main **Configuration** Tab

Calibration Tab

The second user configurable tab within the **Config** tab is **Calibration**. Within this tab, users can enable initialization and tracking of Rx/Tx quadrature error correction (QEC) and Tx local oscillator leakage (LOL) calibrations. Figure 187 shows a calibration configuration example. The user can enable or disable initialization calibrations as well as tracking calibrations.

Use external circuitry for **External Tx LOL** initialization calibration as well as **Tx1 LOL** and **Tx2 LOL** tracking calibrations. The Hardware Setup for External Tx LO Leakage Calibration section explains the external hardware configuration. The **External Init Atten** option, located in the **Initialization Calibration** section of the **Calibration** tab (see Figure 187), allows the user to control the level of attenuation applied internally at both Tx outputs simultaneously.



Initialization Calibration	Run
Rx QEC	✓
Tx QEC	✓
Internal Tx LOL	✓
External Tx LOL	✓
External Init Attn	0.00

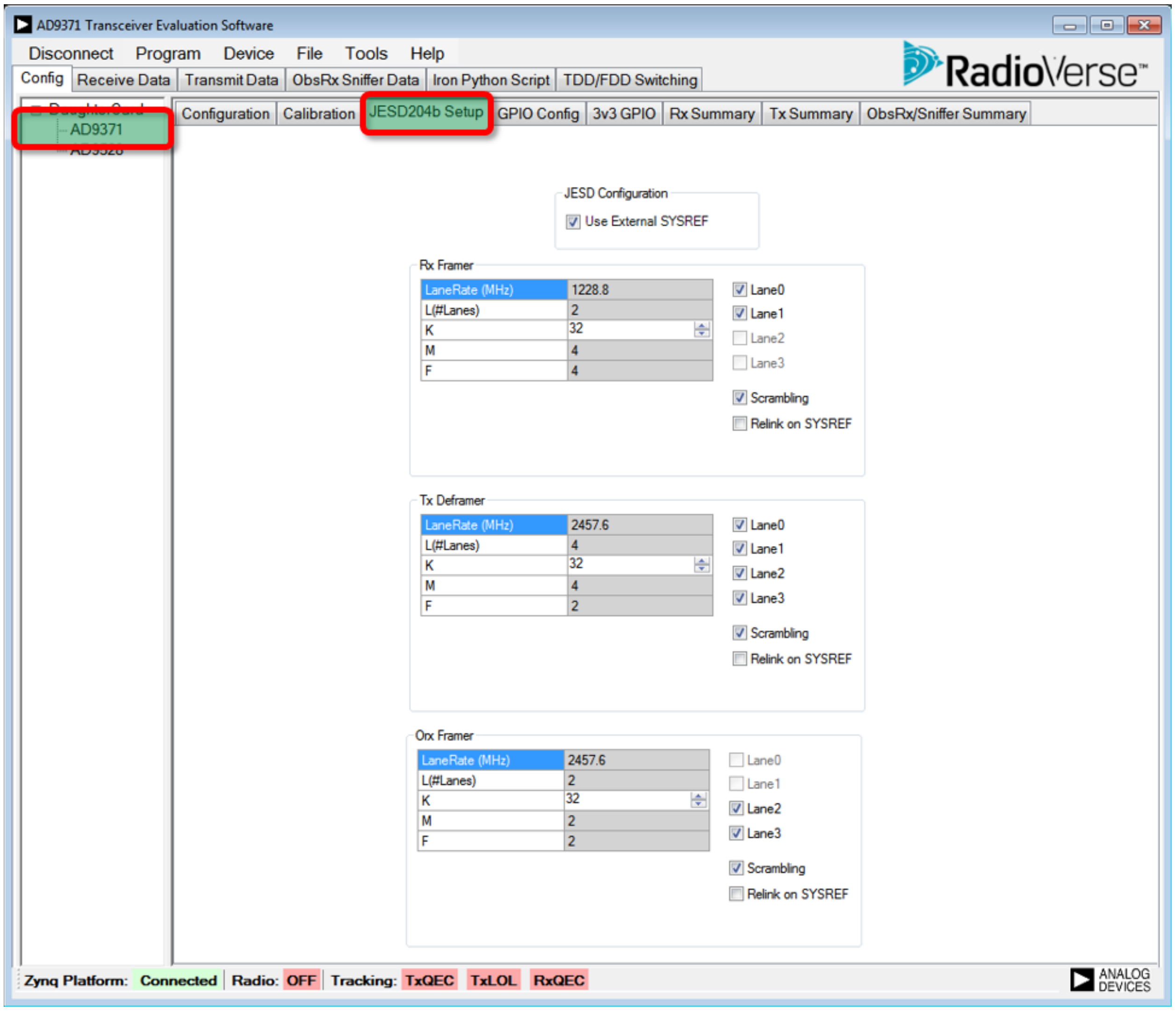
Tracking Calibration	Run
Rx1 QEC	✓
Rx2 QEC	✓
Tx1 LOL	✓
Tx2 LOL	✓
Tx1 QEC	✓
Tx2 QEC	✓

Please refer to the quickstart guide for External Tx LOL setup

Figure 187. ***Calibration*** *Configuration Tab*

JESD204B Setup Tab

The third user configurable tab within the **Config** tab is the **JESD204b Setup** tab. Users can set the characteristics of the digital data interface within this tab. Figure 188 shows a JESD204B setup configuration example. The user can set the desired JESD204B lane configuration, select scrambling, and choose whether the selected framer/deframer relinks on SYSREF. The user can also select either an internal (free running) or external (provided by the AD9528) SYSREF to synchronize the JESD204B links.


AD9371 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
RadioVerse™
Config | Receive Data | Transmit Data | ObsRx Sniffer Data | Iron Python Script | TDD/FDD Switching
DaughterCard
AD9371
AD9528
Configuration | Calibration | JESD204b Setup | GPIO Config | 3v3 GPIO | Rx Summary | Tx Summary | ObsRx/Sniffer Summary
Zynq Platform: Connected Radio: OFF Tracking: TxQEC TxLOL RxQEC
ANALOG DEVICES

JESD Configuration

[✓] Use External SYSREF

Rx Framer

LaneRate (MHz)	1228.8
L(#Lanes)	2
K	32
M	4
F	4

[✓] Lane0
[✓] Lane1
[ ] Lane2
[ ] Lane3
[✓] Scrambling
[ ] Relink on SYSREF

Tx Deframer

LaneRate (MHz)	2457.6
L(#Lanes)	4
K	32
M	4
F	2

[✓] Lane0
[✓] Lane1
[✓] Lane2
[✓] Lane3
[✓] Scrambling
[ ] Relink on SYSREF

Orx Framer

LaneRate (MHz)	2457.6
L(#Lanes)	2
K	32
M	2
F	2

[ ] Lane0
[ ] Lane1
[✓] Lane2
[✓] Lane3
[✓] Scrambling
[ ] Relink on SYSREF

Figure 188. ***JESD204b Setup*** *Tab*

GPIO Configuration Tab

The fourth user configurable tab within the **Config** tab is the general-purpose input/output configuration (**GPIO Config**) tab. Users can set the characteristics of the general-purpose input/output (GPIO) interface. It allows the user to program behavior of the GPIO interface (powered from the VDD_INTERFACE power domain).

Use the GPIO **Config** tab, to do the following:

- Monitor the status of the GPIO pins using the **GPIO ACTIVE** controls. When modifying GPIO settings, click the **Check GPIO** button to check the new settings, and click the **Program GPIO** to program these settings to the transceiver (see Figure 190 and Figure 191).
- Check the **Rx MGC Pin control** box (see Figure 190 and Figure 189), to select the GPIO pins used by Rx manual gain control (MGC) mode.

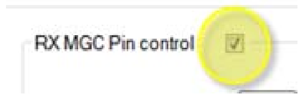
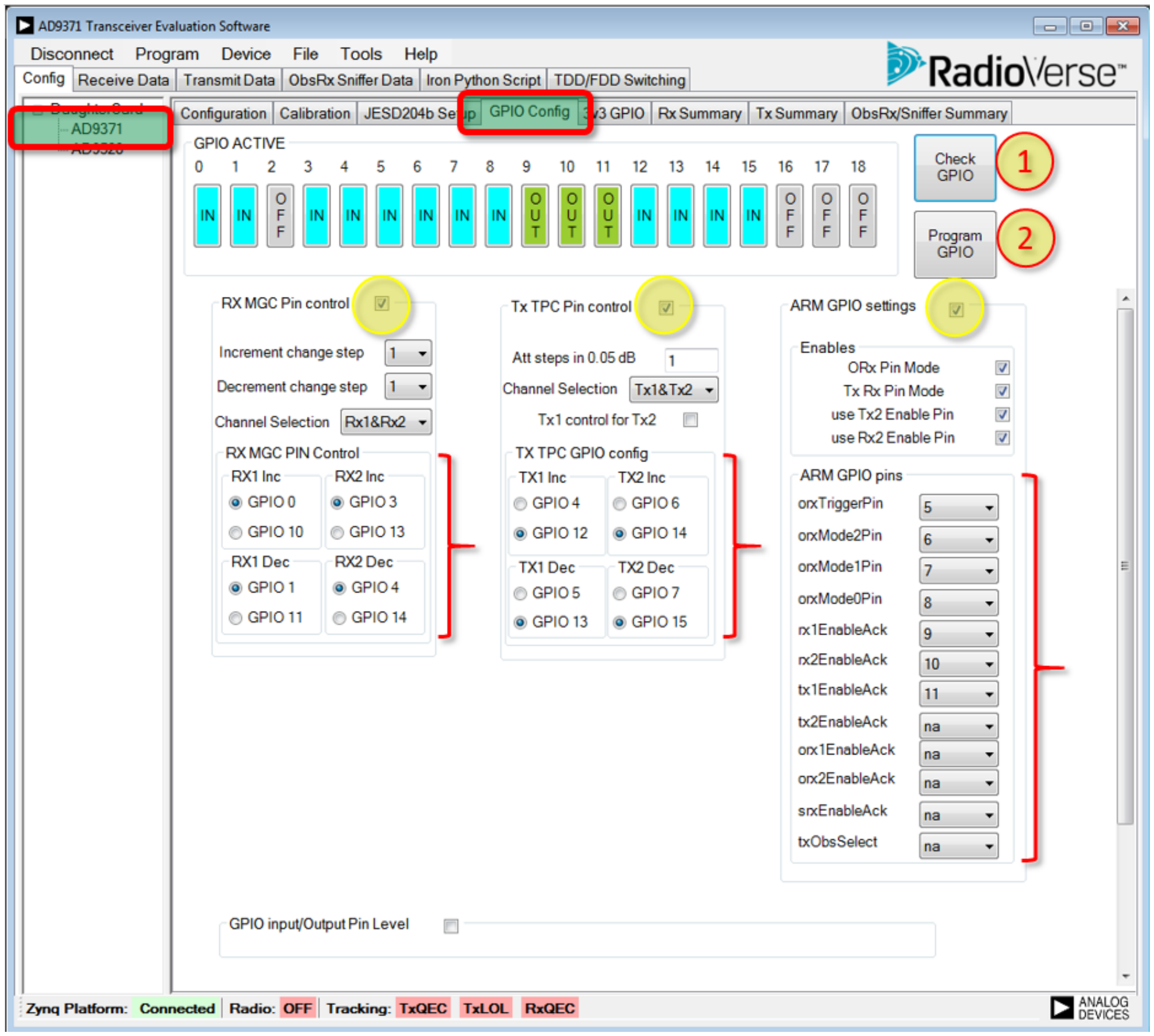


*Figure 189. **Rx MCG Pin control Box** (Circle Highlight for Emphasis Only)*

- The user can select increment or decrement gain steps as well as assign particular GPIO pins to perform selected actions. Figure 190 shows a GPIO receive configuration example.



GPIO ACTIVE

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
IN	IN	OFF	IN	IN	IN	IN	IN	IN	OUT	OUT	OUT	IN	IN	IN	IN	OFF	OFF	OFF

RX MGC Pin control ✓

Increment change step: 1
Decrement change step: 1
Channel Selection: Rx1&Rx2

RX MGC PIN Control
RX1 Inc: [● GPIO 0] ○ GPIO 10
RX2 Inc: [● GPIO 3] ○ GPIO 13
RX1 Dec: [● GPIO 1] ○ GPIO 11
RX2 Dec: [● GPIO 4] ○ GPIO 14

Tx TPC Pin control ✓

Att steps in 0.05 dB: 1
Channel Selection: Tx1&Tx2
Tx1 control for Tx2 ☐

TX TPC GPIO config
TX1 Inc: ○ GPIO 4 [● GPIO 12]
TX2 Inc: ○ GPIO 6 [● GPIO 14]
TX1 Dec: ○ GPIO 5 [● GPIO 13]
TX2 Dec: ○ GPIO 7 [● GPIO 15]

ARM GPIO settings ✓

Enables
ORx Pin Mode ✓
Tx Rx Pin Mode ✓
use Tx2 Enable Pin ✓
use Rx2 Enable Pin ✓

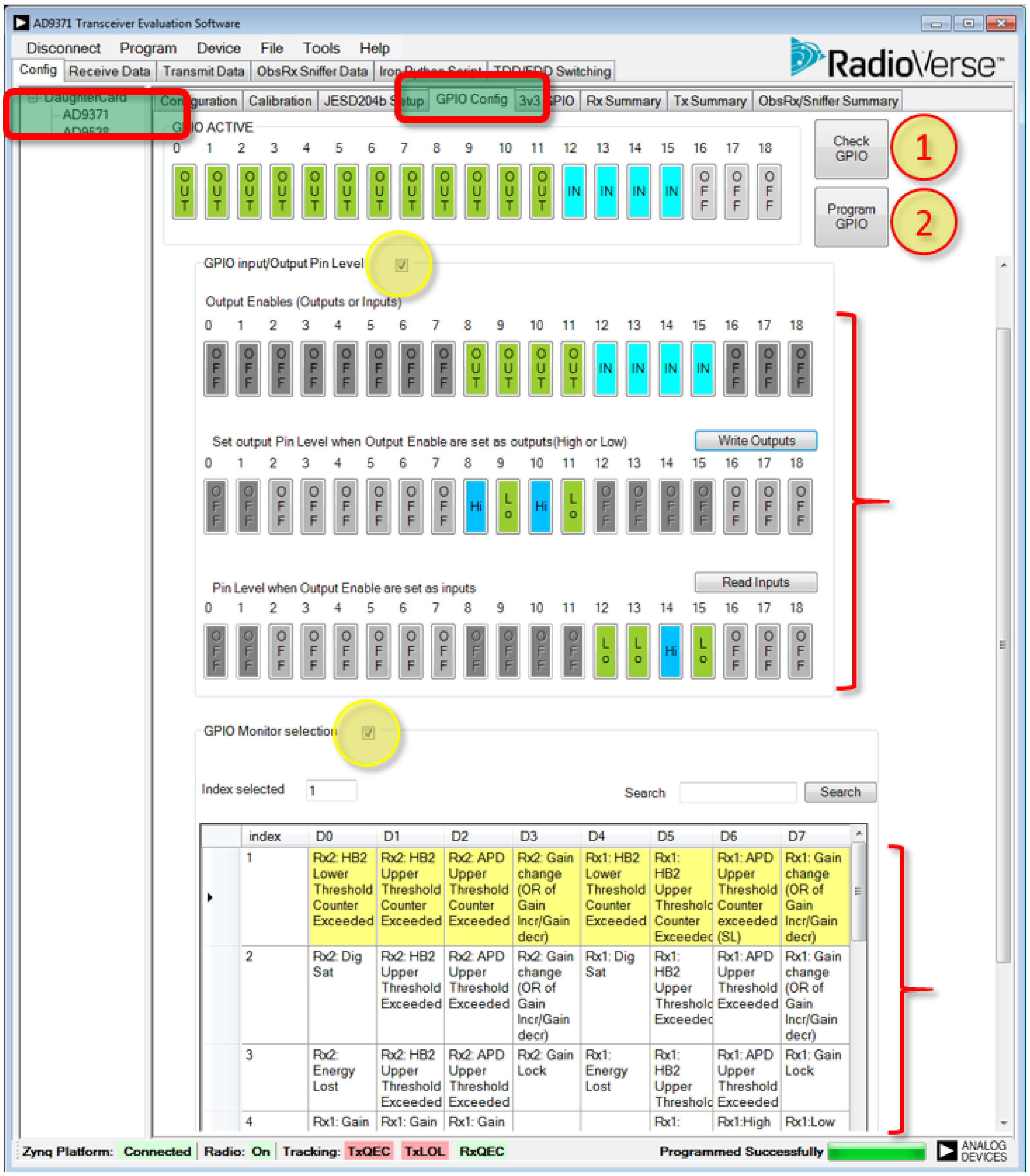
ARM GPIO pins

orxTriggerPin	5
orxMode2Pin	6
orxMode1Pin	7
orxMode0Pin	8
rx1EnableAck	9
rx2EnableAck	10
tx1EnableAck	11
tx2EnableAck	na
orx1EnableAck	na
orx2EnableAck	na
srxEnableAck	na
txObsSelect	na

GPIO input/Output Pin Level ☐

Zynq Platform: Connected | Radio: OFF | Tracking: TxQEC TxLOL RxQEC

Figure 190. GPIO Tab Setup: Rx MGC, Tx TPC, and ARM GPIO Settings


AD9371 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
Config Receive Data Transmit Data ObsRx Sniffer Data Iron Python Script TDD/FDD Switching
RadioVerse
DaughterCard
AD9371
AD9528
Configuration Calibration JESD204b Setup GPIO Config 3v3 GPIO Rx Summary Tx Summary ObsRx/Sniffer Summary
Check GPIO
Program GPIO
1
2
Write Outputs
Read Inputs
Search

GPIO ACTIVE

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	IN	IN	IN	IN	OFF	OFF	OFF

GPIO input/Output Pin Level ✓

Output Enables (Outputs or Inputs)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OUT	OUT	OUT	OUT	IN	IN	IN	IN	OFF	OFF	OFF

Set output Pin Level when Output Enable are set as outputs(High or Low)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	Hi	Lo	Hi	Lo	OFF	OFF	OFF	OFF	OFF	OFF	OFF

Pin Level when Output Enable are set as inputs

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	Lo	Lo	Hi	Lo	OFF	OFF	OFF

GPIO Monitor selection ✓

Index selected: 1

index	D0	D1	D2	D3	D4	D5	D6	D7
1	Rx2: HB2 Lower Threshold Counter Exceeded	Rx2: HB2 Upper Threshold Counter Exceeded	Rx2: APD Upper Threshold Counter Exceeded	Rx2: Gain change (OR of Gain Incr/Gain decr)	Rx1: HB2 Lower Threshold Counter Exceeded	Rx1: HB2 Upper Threshold Counter Exceeded	Rx1: APD Upper Threshold Counter exceeded (SL)	Rx1: Gain change (OR of Gain Incr/Gain decr)
2	Rx2: Dig Sat	Rx2: HB2 Upper Threshold Exceeded	Rx2: APD Upper Threshold Exceeded	Rx2: Gain change (OR of Gain Incr/Gain decr)	Rx1: Dig Sat	Rx1: HB2 Upper Threshold Exceeded	Rx1: APD Upper Threshold Exceeded	Rx1: Gain change (OR of Gain Incr/Gain decr)
3	Rx2: Energy Lost	Rx2: HB2 Upper Threshold Exceeded	Rx2: APD Upper Threshold Exceeded	Rx2: Gain Lock	Rx1: Energy Lost	Rx1: HB2 Upper Threshold	Rx1: APD Upper Threshold Exceeded	Rx1: Gain Lock
4	Rx1: Gain	Rx1: Gain	Rx1: Gain			Rx1:	Rx1:High	Rx1:Low

Zynq Platform: Connected | Radio: On | Tracking: TxQEC TxLOL RxQEC   Programmed Successfully

14652-317

Figure 191. GPIO Tab Setup, Input/Output Manual Control and Monitor Output Mode

- Check the **Tx TPC Pin control** box (see Figure 190 and Figure 192) to select the GPIO pins used by Tx transmit power control mode.

Figure 192. Tx TPC Pin Control Checkbox (Circle Highlight for Emphasis Only)

- The user can select attenuation step size as well as assign particular GPIO pins to perform selected actions. Figure 190 shows a GPIO transmit configuration example.
- Check the **ARM GPIO settings** box (see Figure 190 and Figure 193) to selecting the GPIO pins assigned to interface with the on-board ARM microcontroller.

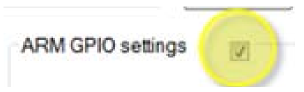
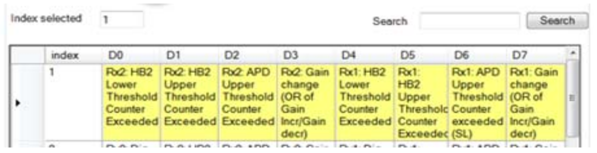


Figure 193. ARM GPIO Checkbox (Circle Highlight for Emphasis Only)

- The user can select GPIO pin or SPI interface mode to control the on-board ARM mode operation. Figure 190 shows a GPIO ARM configuration example.
- Check the **GPIO Monitor selection** box (see Figure 191) to select the GPIO pins used by the internal monitoring block. Press the **Ctrl** key while clicking in the cells within the shown table to select the various options for selecting the desired signals (see Figure 194). Note that only cells from a single row can be selected at one time. The user can select from a single row the entire set of monitoring output signals or a subset of them. Figure 191 shows a GPIO monitor configuration example where all signals from the Index 1 row are selected. The **Search** function helps the user navigate in the control output signals table.



Index selected 1 Search

index	D0	D1	D2	D3	D4	D5	D6	D7
1	Rx2: HB2 Lower Threshold Counter Exceeded	Rx2: HB2 Upper Threshold Counter Exceeded	Rx2: APD Upper Threshold Counter Exceeded	Rx2: Gain change (OR of Gain Incr/Gain decr)	Rx1: HB2 Lower Threshold Counter Exceeded	Rx1: HB2 Upper Thresholc Counter Exceedec	Rx1: APD Upper Threshold Counter exceeded (SL)	Rx1: Gain change (OR of Gain Incr/Gain decr)

Figure 194. Index 1, GPIO Monitor Selection Table

- Selecting the GPIO pins used by the GPIO operating in manual input/output mode are enabled by Check the **GPIO input/Output Pin Level** box (see Figure 191 and Figure 195).

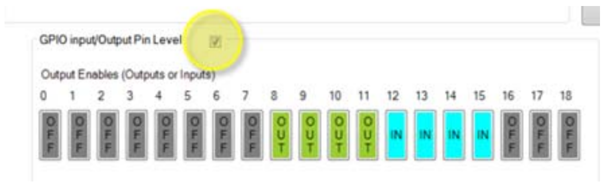

GPIO input/Output Pin Level
Output Enables (Outputs or Inputs)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
OFF OFF OFF OFF OFF OFF OFF OFF OUT OUT OUT OUT IN IN IN IN OFF OFF OFF
14652-322

*Figure 195. **GPIO input/Output Pin Level** Box Location (Yellow Circle for Emphasis)*

- Click the **Output Enables (Outputs or Inputs)** box to select the GPIO pins to be used in manual mode (see Figure 191). Choosing whether a GPIO pin will operate as an output, an input, or be disabled is determined by the number of clicks made to the selection. For example, to select a pin (such as GPIO Pin 8 in Figure 196) to operate as an output, place your mouse cursor over the **8** box and click once. Each mouse click on an individual box toggles the operational status of that pin between three options: output (one click), input (two clicks), and disabled (three clicks). Disabling the selected GPIO pin in manual mode allows that GPIO pin to be used by different GPIO modes.

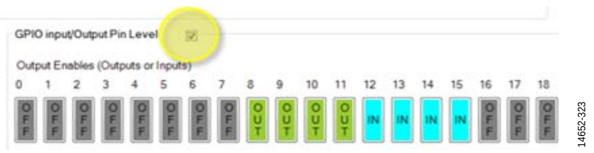

GPIO input/Output Pin Level
Output Enables (Outputs or Inputs)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
OFF OFF OFF OFF OFF OFF OFF OFF OUT OUT OUT OUT IN IN IN IN OFF OFF OFF
14652-323

Figure 196. Selecting GPIO Pin Status: Output, Input, or Disabled

- Select the logic level (high or low) of the GPIO pins in the **Set output Pin Level when Output Enable are set as outputs (High or Low)** section (see Figure 191). Click either once (for high) or twice (for low) on its respective numbered icon to set the desired logic level of each GPIO pin. To apply the logic selections, click **Write Outputs** (see Figure 197).

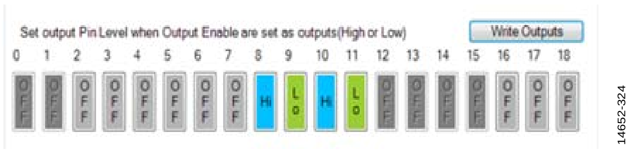

Set output Pin Level when Output Enable are set as outputs(High or Low)
Write Outputs
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
OFF OFF OFF OFF OFF OFF OFF OFF Hi Lo Hi Lo OFF OFF OFF OFF OFF OFF OFF
14652-324

Figure 197. Setting Logic Level of GPIO Pins

- Read the logic level for the GPIO input only pins in the **Pin Level when Output Enable are set as inputs** section. Click **Read Inputs** to populate the logic levels for the inputs, high (**Hi**), low (**Lo**), or off (**OFF**) (see Figure 198).

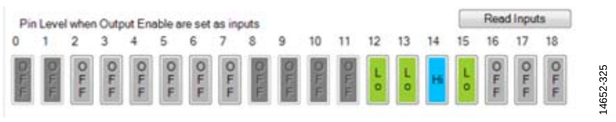

Pin Level when Output Enable are set as inputs
Read Inputs
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
OFF OFF OFF OFF OFF OFF OFF OFF OFF OFF OFF OFF Lo Lo Hi Lo OFF OFF OFF
14652-325

Figure 198. Setting GPIO Inputs to Logic High or Logic Low

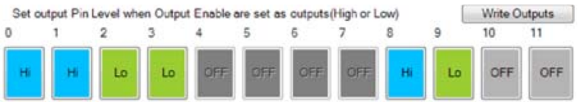
Figure 191 shows a configuration example with GPIO Pin 8 to GPIO Pin 11 set to operate as outputs and GPIO Pin 12 to GPIO Pin 15 set to operate as inputs. GPIO Pin 8 and GPIO Pin 10 are set to logic high. GPIO Pin 9 and GPIO Pin 11 are set to logic low. GPIO Pin 12, GPIO Pin 13, and GPIO Pin 15 read back logic low, and GPIO Pin 14 reads back logic high.

3V3 GPIO Tab

The fifth user configurable tab within the **Config** tab is the **3v3 GPIO** tab (see Figure 200). This tab sets the characteristics of the 3.3 V general-purpose input/output interface. It allows users to program the behavior of the GPIO interface (powered from a 3.3 V power domain). Within the **3V3 GPIO** tab, the user can select and enable operation of the 3.3 V GPIO pins in manual input/output mode. Check the **GPIO 3v3 Input/Output Pin Level** box to enable manual mode.

Within the **3v3 GPIO** tab, users can perform the following:

- Click the GPIO numbered boxes in the **Output Enables (Outputs or Inputs)** section (see Figure 200) to select which 3.3 V GPIO pin is used in manual mode. Click once on a numbered box to set the 3.3 V GPIO pin to operate as an output, click twice to set it to operate as an input, and click it a third time to disable manual mode for that 3.3 V GPIO pin. For example, in Figure 200, click once in the **0** box to set the 3.3 V GPIO 0 pin as an output.
- Click on the numbered boxes in the **Set output Pin Level when Output Enable are set as outputs (High or Low)** section to select the logic level for the 3.3 V GPIO output only pins. Click once on the box to set that 3.3 V GPIO pin to logic high and click twice on the box to set the pin to logic low. Click **Write Outputs** to selected the levels applied to the 3.3 V GPIO pins (see Figure 199).

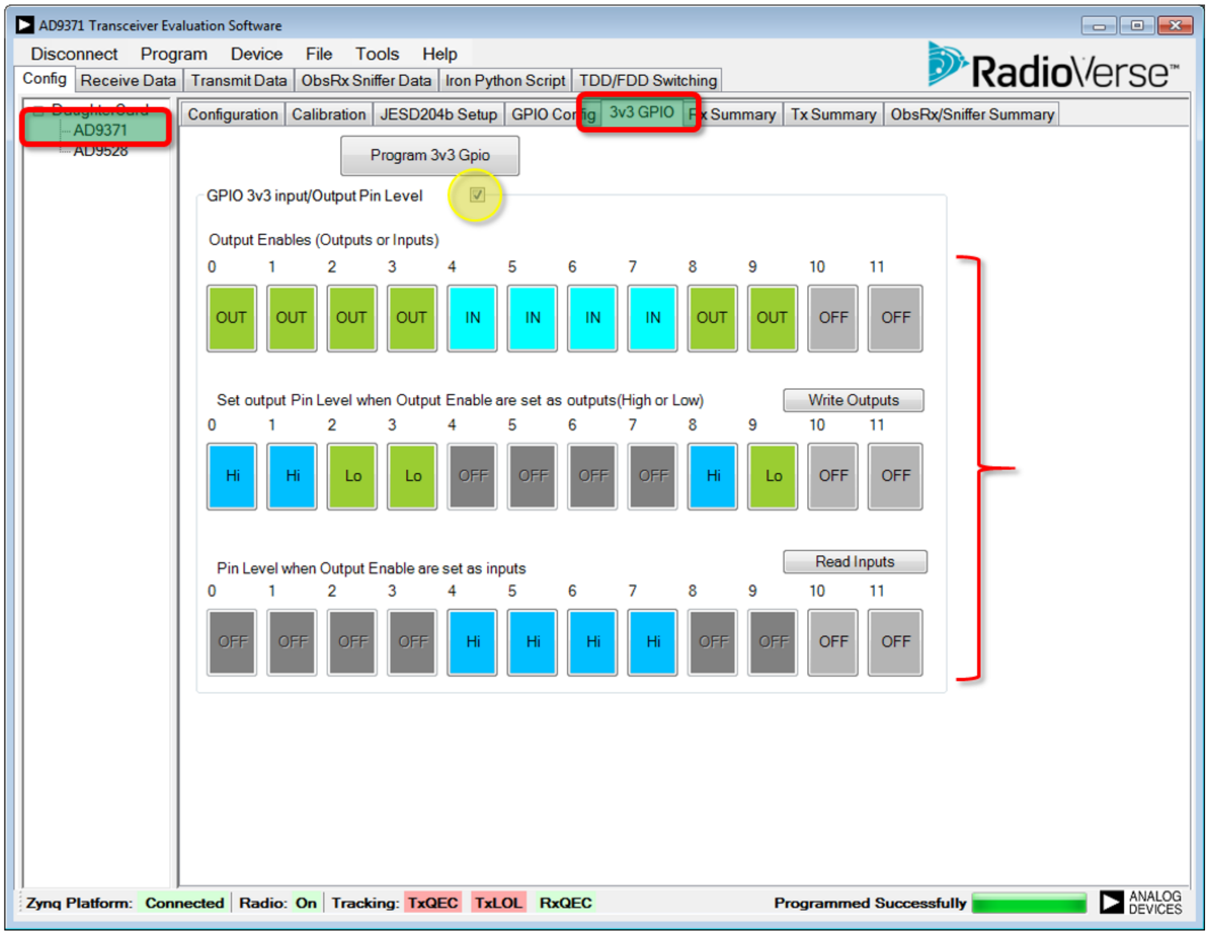

Set output Pin Level when Output Enable are set as outputs(High or Low) — Write Outputs

0	1	2	3	4	5	6	7	8	9	10	11
Hi	Hi	Lo	Lo	OFF	OFF	OFF	OFF	Hi	Lo	OFF	OFF

Figure 199. Setting Logic Levels for the 3.3 V GPIO Pins

- Read the logic level for the 3.3 V GPIO input only pins in the **Pin Level when Output Enable are set as inputs** section. Click **Read Inputs** to populate the logic levels for the inputs, high (**Hi**), low (**Lo**), or off (**OFF**).

Figure 200 shows a configuration example with 3.3 V GPIO Pin 0 to Pin 3, Pin 8, and Pin 9 set to operate as outputs, and 3.3 V GPIO Pin 4 to Pin 7 set to operate as inputs. The 3.3 V GPIO Pin 0, Pin 1, and Pin 8 are set to logic high, and the 3.3 V GPIO Pin 2, Pin 3, and Pin 9 are set to logic low. The 3.3 V GPIO Pin 4 to Pin 7 read back logic high.


AD9371 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
RadioVerse
Config | Receive Data | Transmit Data | ObsRx Sniffer Data | Iron Python Script | TDD/FDD Switching
DaughterCard
AD9371
AD9528
Configuration | Calibration | JESD204b Setup | GPIO Config | 3v3 GPIO | Rx Summary | Tx Summary | ObsRx/Sniffer Summary
Program 3v3 Gpio
Write Outputs
Read Inputs
ANALOG DEVICES
14652-326

GPIO 3v3 input/Output Pin Level ✓

Output Enables (Outputs or Inputs)

0	1	2	3	4	5	6	7	8	9	10	11
OUT	OUT	OUT	OUT	IN	IN	IN	IN	OUT	OUT	OFF	OFF

Set output Pin Level when Output Enable are set as outputs(High or Low)

0	1	2	3	4	5	6	7	8	9	10	11
Hi	Hi	Lo	Lo	OFF	OFF	OFF	OFF	Hi	Lo	OFF	OFF

Pin Level when Output Enable are set as inputs

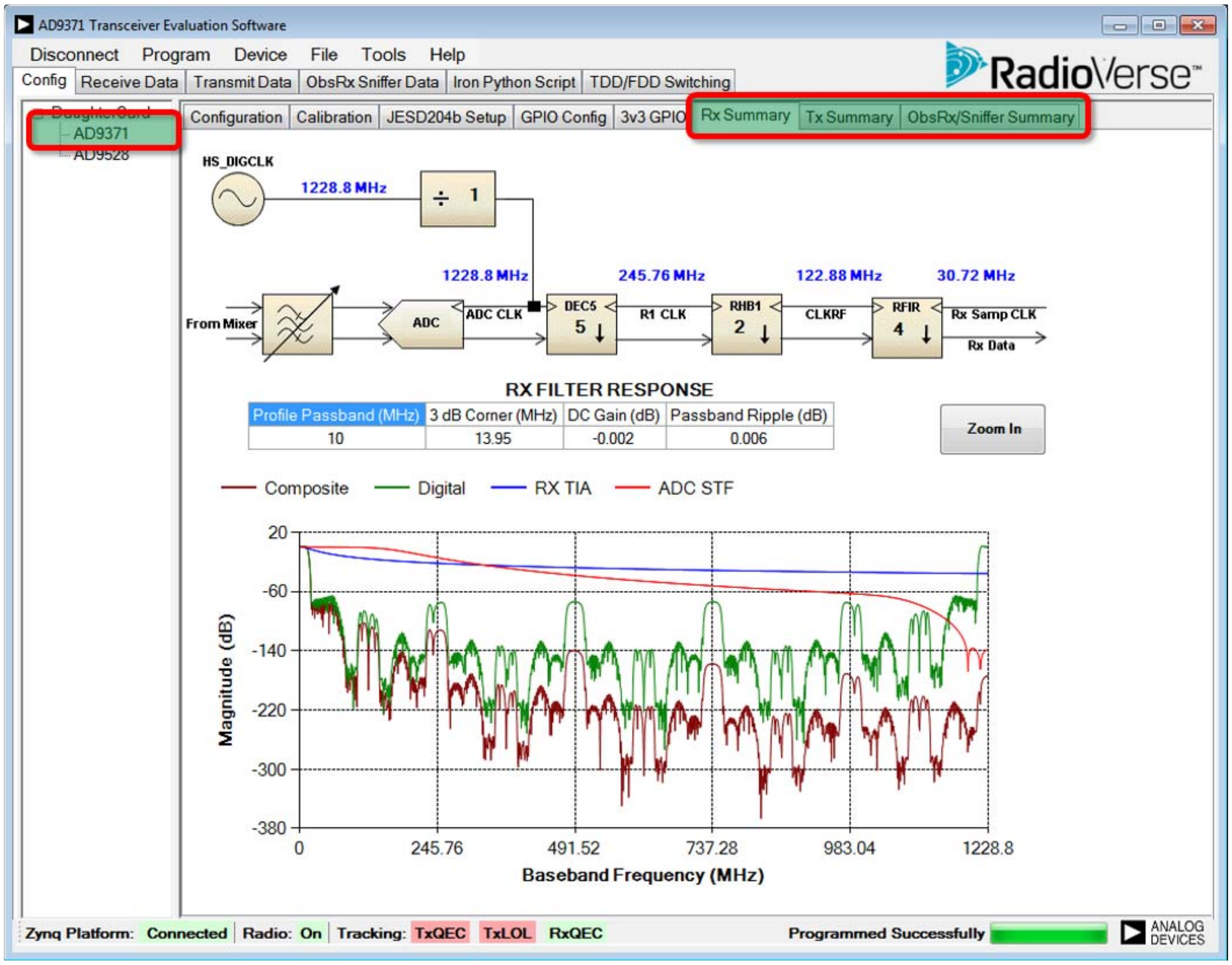
0	1	2	3	4	5	6	7	8	9	10	11
OFF	OFF	OFF	OFF	Hi	Hi	Hi	Hi	OFF	OFF	OFF	OFF

Zynq Platform: Connected | Radio: On | Tracking: TxQEC TxLOL RxQEC | Programmed Successfully

Figure 200. **3v3 GPIO** Tab Setup

Rx, Tx and ObsRx/Sniffer Summary Tabs

The **Rx Summary**, **Tx Summary**, and **ObsRx/Sniffer Summary** tabs are primarily informative and are based on the profile selection in the **Configuration** tab (see Figure 186). In each of these tabs, the user can check clock rates at each filter node as well as filter characteristics and their pass-band flatness. Quick zooming capabilities allow zooming of the pass-band response using the mouse cursor as well as restoring to the full-scale plot. Right-click on the graph area and select **Export Data to File** within the transceiver evaluation software (TES) to export the data plotted on the graphs to an external file. Data can then be saved to a file for later analyses. Figure 201 shows an example of the **Rx Summary** tab with the resulting composite filter response for the chosen profile.


AD9371 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
RadioVerse™
Config | Receive Data | Transmit Data | ObsRx Sniffer Data | Iron Python Script | TDD/FDD Switching
DaughterCard
AD9371
AD9528
Configuration | Calibration | JESD204b Setup | GPIO Config | 3v3 GPIO | Rx Summary | Tx Summary | ObsRx/Sniffer Summary
HS_DIGCLK
1228.8 MHz
÷ 1
1228.8 MHz
245.76 MHz
122.88 MHz
30.72 MHz
From Mixer
ADC
ADC CLK
DEC5 5 ↓
R1 CLK
RHB1 2 ↓
CLKRF
RFIR 4 ↓
Rx Samp CLK
Rx Data
Zoom In
Composite
Digital
RX TIA
ADC STF
Magnitude (dB)
Baseband Frequency (MHz)
20
-60
-140
-220
-300
-380
0
245.76
491.52
737.28
983.04
1228.8
Zynq Platform: Connected | Radio: On | Tracking: TxQEC TxLOL RxQEC
Programmed Successfully
ANALOG DEVICES
14652-328

RX FILTER RESPONSE

Profile Passband (MHz)	3 dB Corner (MHz)	DC Gain (dB)	Passband Ripple (dB)
10	13.95	-0.002	0.006

*Figure 201. **Rx Summary** Tab*

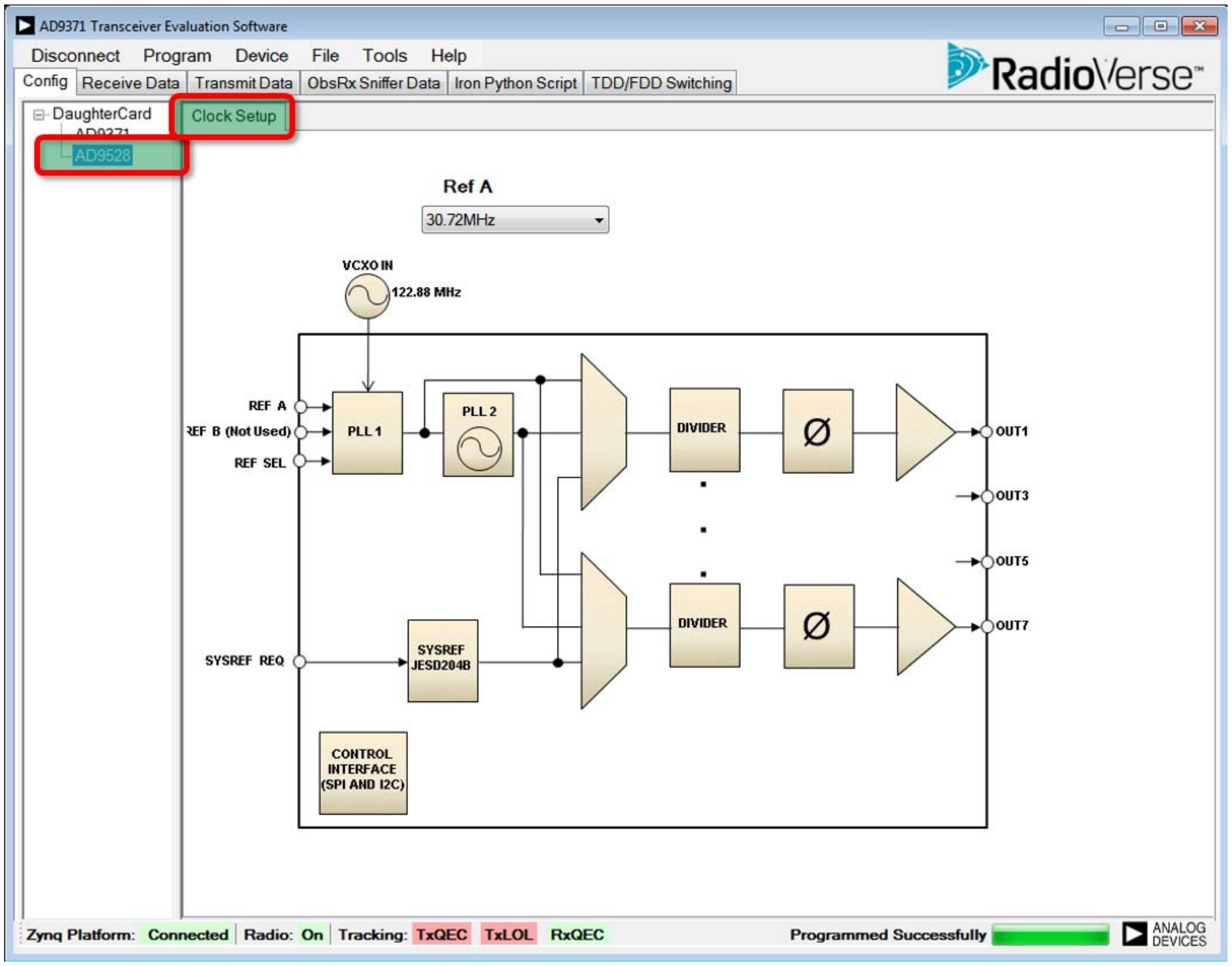
Configuring the AD9528 Clock Chip

The daughter card utilizes the AD9528 clock chip to provide the reference clock, DEV_CLK, as well as a SYSREF pulse to the device and the field programmable gate array (FPGA) on the ZYNQ platform via the FMC connector. Configure the AD9528 using the **Clock Setup** tab (see Figure 202). Use the **Ref A** dropdown menu within the **Clock Setup** tab to select the input reference frequency. Note that an external reference clock must be connected to the J401 SMA connector that matches the frequency selected in the dropdown menu. The signal amplitude must not exceed 5 dBm.

On-Board VCXO Modification

The device evaluation board contains an on-board voltage controlled crystal oscillator (VCXO) as well as the AD9528 chip responsible for the device clock and SYSREF signal generation and distribution. With the hardware configuration provided on the evaluation board, a user can generate device clock frequencies such as 122.88 MHz, 153.6 MHz, 184.32 MHz, 245.76 MHz, and 307.2 MHz.

There are limitations with the default hardware configuration in the scenario where user desired device frequencies are not related to the on-board 122.88 MHz VCXO by a rational fraction. Examples of such device clock frequencies are: 125 MHz, 133.33 MHz, 250 MHz, and 266.66 MHz. The following section outlines these limitations and explains how to overcome them with evaluation board hardware modifications.



Ref A: 30.72MHz

Zynq Platform: Connected | Radio: On | Tracking: TxQEC TxLOL RxQEC

Programmed Successfully

Figure 202. AD9528 ***Clock Setup*** *Configuration Page*

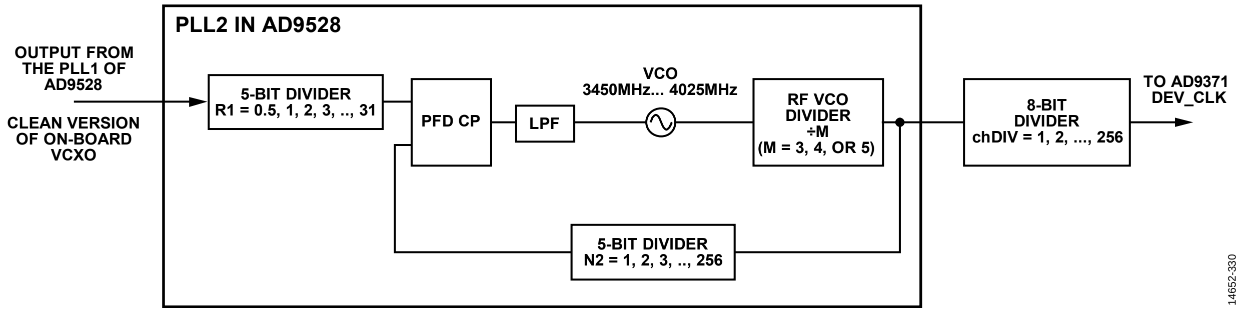

PLL2 IN AD9528
OUTPUT FROM THE PLL1 OF AD9528
CLEAN VERSION OF ON-BOARD VCXO
5-BIT DIVIDER R1 = 0.5, 1, 2, 3, .., 31
PFD CP
LPF
VCO 3450MHz... 4025MHz
RF VCO DIVIDER ÷M (M = 3, 4, OR 5)
8-BIT DIVIDER chDIV = 1, 2, ..., 256
TO AD9371 DEV_CLK
5-BIT DIVIDER N2 = 1, 2, 3, .., 256
14652-330

Figure 203. AD9528 PLL2 Block Diagram

AD9528 Description

The AD9528 contains two cascaded phase-locked loop (PLL) stages. The first PLL stage (PLL1) works with a narrow loop filter bandwidth. PLL1 provides jitter clean up of the input reference signal to provide a clean clock for the input stage of PLL2. The configuration of PLL2 is described in Figure 203.

PLL2 blocks are programmable and the following values can be selected:

- M = 3, 4, or 5
- N2 = 1, 2, 3, …, 256
- R1 = 0.5, 1, 2, …, 31
- chDIV = 1, 2, 3, …, 256

Note that the PLL2 voltage controlled oscillator (VCO) frequency operates from 3450 MHz to 4025 MHz.

To calculate the DEV_CLK frequency, use the following equations:

$$\textit{VCO Frequency} = (\textit{VCXO Frequency} \times M \times N2)/R1 \quad (41)$$

Note that the VCO can operate with a frequency range from 3450 MHz to 4025 MHz.

$$\textit{DEVCLK Frequency} = \textit{VCXO Frequency}/(M \times \textit{Channel Division}) \quad (42)$$

AD9528 Operation Examples

The AD9528 can only generate different DEV_CLK frequencies from the VCXO frequency when the ratio of the two frequencies is a rational fraction. If the result of the DEV_CLK division and VCXO frequency does not create a rational fraction, the AD9528 cannot precisely generate the desired DEV_CLK.

The following are some examples of how DEV_CLK is calculated based on an on-board VCXO with a frequency of 122.88 MHz.

Example 1: This example targets a DEV_CLK of 245.76 MHz. Plugging in values to Equation 41 produces Equation 43 and, likewise, using Equation 42 generates the results shown in Equation 44.

$$(122.88 \text{ MHz} \times 3 \times 30)/3 = 3686.4 \text{ MHz} \quad (43)$$

$$3686.4 \text{ MHz}/(3 \times 5) = 245.76 \text{ MHz} \quad (44)$$

Therefore, Equation 43 and Equation 44 demonstrate that a DEV_CLK = 245.76 MHz can generate using the on-board VCXO.

Example 2: This example targets a 153.6 MHz DEV_CLK. Using Equation 41 and Equation 42, the following results:

$$(122.88 \text{ MHz} \times 3 \times 30)/3 = 3686.4 \text{ MHz} \quad (45)$$

$$3686.4 \text{ MHz}/(3 \times 8) = 153.6 \text{ MHz} \quad (46)$$

Equation 45 and Equation 46 demonstrate that a DEV_CLK = 153.6 MHz can be generated using the on-board VCXO.

Example 3: This example targets a DEV_CLK of 125 MHz. Following the same process as the previous examples, the following results are obtained:

$$(122.88 \text{ MHz} \times 3 \times 30)/3 = 3686.4 \text{ MHz} \quad (47)$$

$$3686.4 \text{ MHz}/(3 \times 10) = 122.88 \text{ MHz} \neq 125 \text{ MHz} \quad (48)$$

In the scenario presented in Equation 47 and Equation 48, the on-board VCXO cannot generate a DEV_CLK of 125 MHz. Likewise, Equation 49 and Equation 50 produce the same result wherein a DEV_CLK of 125 MHz cannot be generated using the on-board VCXO.

$$(122.88 \text{ MHz} \times 3 \times 31)/3 = 3809.28 \text{ MHz} \quad (49)$$

$$3809.28 \text{ MHz}/(3 \times 10) = 126.976 \text{ MHz} \neq 125 \text{ MHz} \quad (50)$$

Example 4: This example uses a VCXO of 125 MHz rather than the 128.88 MHz used in the previous equations to achieve a DEV_CLK of 125 MHz. By modifying the hardware to use a VCXO of 125 MHz, a 125 MHz DEV_CLK can now be selected.

$$(125 \text{ MHz} \times 3 \times 30)/3 = 3750 \text{ MHz} \quad (51)$$

$$3750 \text{ MHz}/(3 \times 8) = 125 \text{ MHz} \quad (52)$$

Example 5: This example returns to using a VCXO of 122.88 MHz and targets a DEV_CLK of 266.66 MHz.

$$(122.88 \text{ MHz} \times 3 \times 32)/3 = 3932.16 \text{ MHz} \quad (53)$$

$$3932.16 \text{ MHz}/(3 \times 5) = 262.144 \text{ MHz} \neq 266 \text{ MHz} \quad (54)$$

Equation 53 and Equation 54 demonstrate that a DEV_CLK of 266.66 MHz cannot be generated using the on-board VCXO.

Example 6: Like Example 5, this example attempts to generate a 266.66 MHz DEV_CLK. However, similar to achieving a 125 MHz DEV_CLK through hardware modification, in this example, the 266.66 MHz DEV_CLK can be achieved by modifying the hardware to use a VCXO of 133.33 MHz rather than 122.88 MHz, as demonstrated in Equation 55 and Equation 56.

achieves a DEV_CLK of 266.66 MHz by modifying the hardware with a VCXO of 133.33 MHz.

$$(133.33\ \text{MHz} \times 3 \times 30)/3 = 3999.9\ \text{MHz} \quad (55)$$

$$3999.9\ \text{MHz}/(3 \times 5) = 266.66\ \text{MHz} \quad (56)$$

On-Board VCXO Hardware Replacement

The evaluation board supports two different footprints for the on-board voltage controlled crystal oscillator (VCXO). Figure 204 outlines two different VCXO symbols present in the evaluation board schematic. Both footprint details are outlined in Figure 205 and Figure 206.

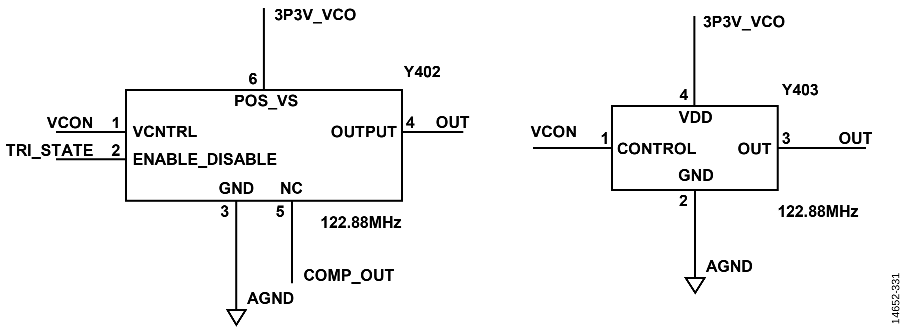


Figure 204. Double Footprint for the On-Board VCXO

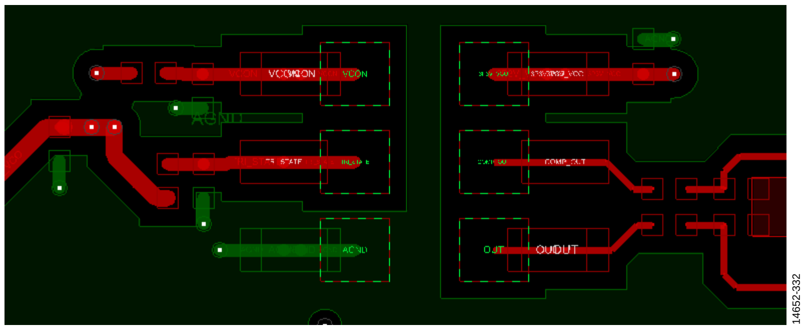


Figure 205. VCXO Footprint: SMD, 5.0 mm × 9.0 mm

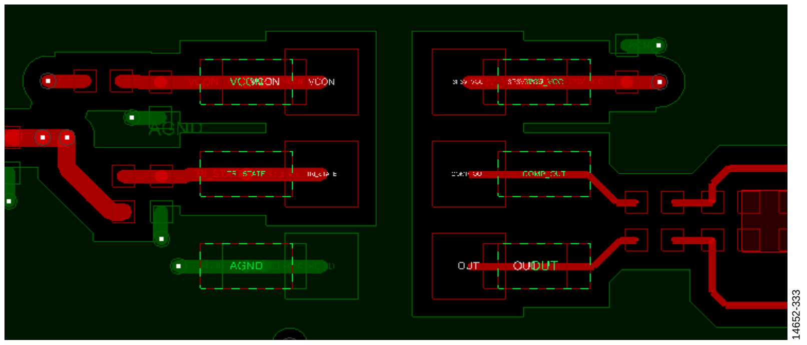


Figure 206. VCXO Footprint: SMD, 9.0 mm × 14.0 mm

VCXO Recommendations

The evaluation board utilizes a voltage controlled crystal oscillator (VCXO) manufactured by Crystek Corporation. All frequency variants recommended for the evaluation board are based on the Crystek Corporation VCXO CVHD-950 series.

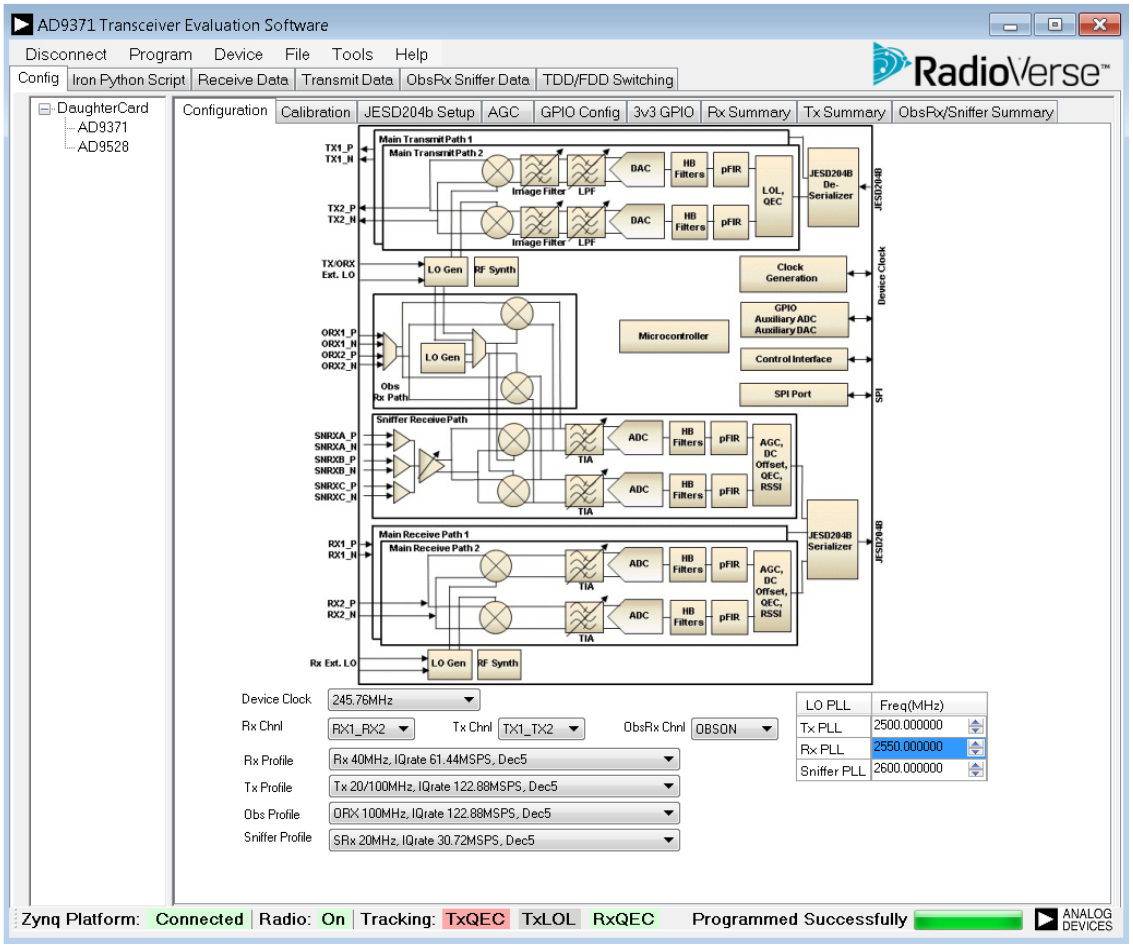
Table 187 lists the typical characteristics of the CVHD-950 series VCXO.

Table 187. Typical CVHD-950 Series Parameters

Parameter	Value
Frequency Range	40 MHz to 130 MHz
Input Voltage	3.3 V ± 0.3 V
Input Current	15 mA typical; 25 mA maximum
Control Voltage	1.65 V ± 1.65 V
Frequency Pulling	±20 ppm APR minimum
Typical Phase Noise (100 MHz)	
1 kHz	−140 dBc/Hz
10 kHz	−155 dBc/Hz
100 kHz	−164 dBc/Hz
1 MHz	−166 dBc/Hz
Phase Noise Floor	−166 dBc/Hz typical, −162 dBc/Hz maximum

PROGRAMMING THE EVALUATION SYSTEM

After all tabs are configured, click **Program** to send (via the transceiver evaluation software (TES)) a series of application programming interface (API) commands that are executed by a dedicated application running on the ZYNQ platform. A progress bar is shown at the bottom of the window and, upon programming completion, the system is ready to operate (see Figure 207).



Device Clock: 245.76MHz

Rx Chnl: RX1_RX2 Tx Chnl: TX1_TX2 ObsRx Chnl: OBSON

LO PLL	Freq(MHz)
Tx PLL	2500.000000
Rx PLL	2550.000000
Sniffer PLL	2600.000000

Rx Profile: Rx 40MHz, IQrate 61.44MSPS, Dec5

Tx Profile: Tx 20/100MHz, IQrate 122.88MSPS, Dec5

Obs Profile: ORX 100MHz, IQrate 122.88MSPS, Dec5

Sniffer Profile: SRx 20MHz, IQrate 30.72MSPS, Dec5

Zynq Platform: Connected | Radio: On | Tracking: TxQEC TxLOL RxQEC Programmed Successfully

Figure 207. Program Device Window

OTHER TES FEATURES

The transceiver evaluation software (TES) provides the user with multiple options to store and load the TES and also hardware configurations. Figure 208 outlines all dropdown menu options provided by TES.

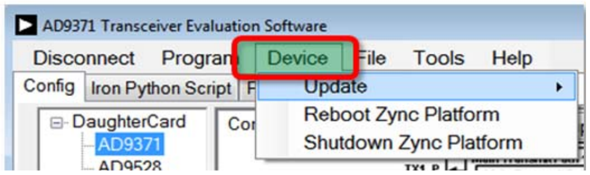


*Figure 208. TES **Device** Dropdown Menu*

Device Dropdown Menu

The following option selections are available in the **Device** dropdown menu:

- **Update** > **Platform Files** is for updating files on the ZYNQ SD card after installing a new version of the software. See the Software Update section for more details.
- **Reboot Zynq Platform** is for when a soft restart of the evaluation system is needed.
- **Shutdown Zynq Platform** is for powering down the evaluation system. The user must use this option or power down the system by closing the TES application and by selecting **Switch Zynq Off** to correctly execute the power-down sequence. If this process is not followed, the file system on the SD card can be corrupted, and the evaluation system may stop operating.

File Dropdown Menu

The following selections are available in the **File** dropdown menu:

- **Save GUI Setup** stores all TES configuration settings. TES generates an XML format file with all software settings recorded. Click **Load Setup** and select the **saved setup** file to load software settings.
- **Load GUI Setup** loads all TES configuration settings stored in XML format using the **Save GUI Setup** option.
- **Load Custom Profile** allows the user to load a custom version of the TES profile using the Filter Wizard software available at the RadioVerse landing page.
- **Clear Custom Profile** restores the TES software to the state before a custom profile was loaded.

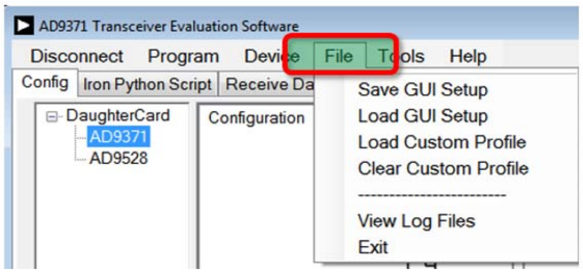
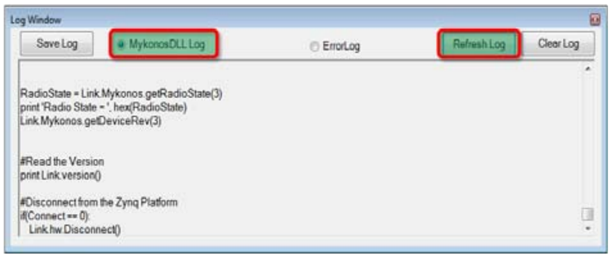


Figure 209. TES File Dropdown Menu

- **View Log Files** monitors application programming interface (API) activities. It opens a window where the following options are available (see Figure 210):
  - Monitoring all API activates. The output is displayed in an IronPython script form.
  - Monitoring error log only. To observe error messages reported by the API software layer, click **Refresh Log** and content of the log window updates. **Log Window** allows the user to store log messages as text files for further analysis. Click **Clear Log** to clear the **Log Window**.



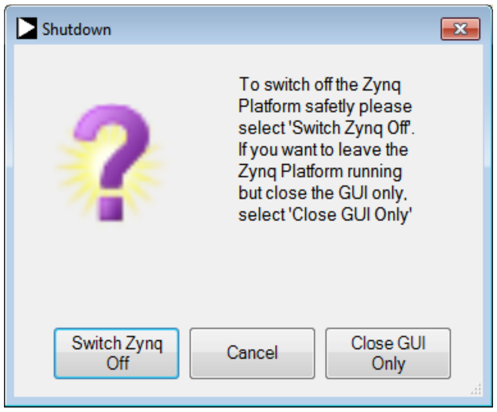
```
RadioState = Link.Mykonos.getRadioState(3)
print 'Radio State = ', hex(RadioState)
Link.Mykonos.getDeviceRev(3)


#Read the Version
print Link.version()

#Disconnect from the Zynq Platform
if(Connect == 0):
   Link.hw.Disconnect()
```

*Figure 210. TES **Log Window***

- **Exit** opens the **Shutdown** window (see Figure 211) where the following options are available:
  - **Switch Zynq Off** powers down the entire system and closes TES.
  - **Close GUI Only** closes only the TES software, and the ZYNQ system remains active
  - **Cancel** closes the **Shutdown** window.



To switch off the Zynq Platform safely please select 'Switch Zynq Off'. If you want to leave the Zynq Platform running but close the GUI only, select 'Close GUI Only'

*Figure 211. **Shutdown** Window*

Tools Dropdown Menu

Figure 212 shows the **Tools** dropdown menu.

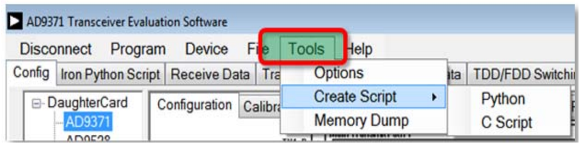
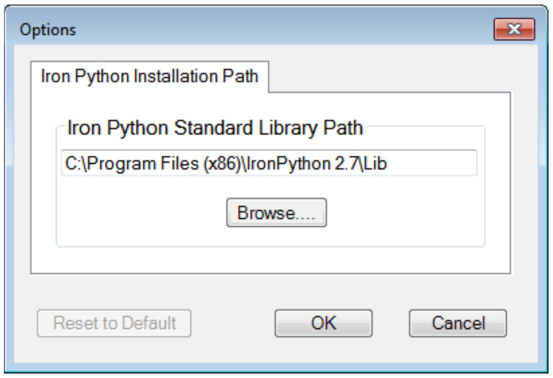


Figure 212. Transceiver Evaluation Software (TES) Tools Dropdown Menu

This menu allows selection of the following options:

- **Options** allows the user to configure a path to the Iron Python library folder (see Figure 213). This setting is automatically populated with the path set during the installation process.



Iron Python Standard Library Path: C:\Program Files (x86)\IronPython 2.7\Lib

Figure 213. TES Options Window

- **Create Script** allows the storing of the initialization script. The TES allows the creation of a script in the following forms:
  - **Python**. When this function executes, the user is asked for the script file name and place where it can be stored. TES generates the new_name.py file with all API initialization calls in the form of IronPython functions. This file can then execute using the **Iron Python Script** tab shown in Figure 226.
  - **C Script**. This action opens the **Save as** window, requires the user to name the file and specify its location for storage. Based on configuration settings outlined in the Configuring the AD9371 section, the TES sets up structure members values that are then used by application programming interface (API) commands. The TES allows the user to create a *.c file that contains all initial values. This file can be imported into the system of the user that utilizes APIs. The TES generates five separate files the include headless.c, headless.h, user_name.c, user_name.h, and user_name_ad9528init.c (see Table 188).

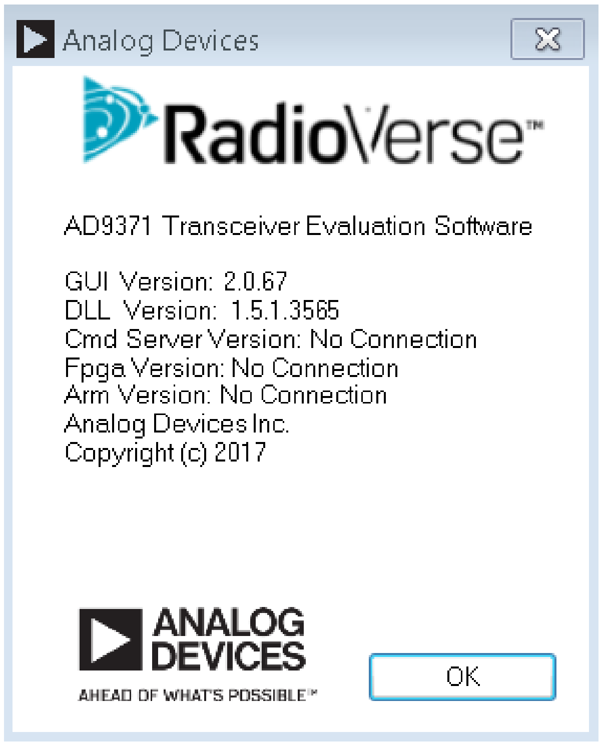
Table 188. TES Files

File	Description
headless.c	Provides an example file that calls into the API to initialize the device.
headless.h	Header file for headless.c.
user_name.c	Contains all initialization values for the structures members used by APIs.
user_name.h	Header file for user_name.c.
user_name_ad9528init.c	Contains all initialization values for the structures used by the AD9528 (clock IC) APIs
Memory Dump	Provides users with the ability to store register values from the internal ARM processor, the register map, and the ZYNQ field programmable gate array (FPGA) register map. When the user clicks **Memory Dump**, the user must enter a file name for the file and select a location where those files are to be stored. The TES then reads internal register values and stores them in three separate files: **user_name.bin**, **user_name_MykonosReg.txt**, and **user_name_FpgaReg.txt**
user_name_bin	For internal ARM processor dump.
user_name_MykonosReg.txt	For register dump.
user_name_FpgaReg.txt	For ZYNQ FPGA register dump.

Help Dropdown Menu

The **Help** dropdown menu (see Figure 217) includes the following:

- **API Help File** opens the **Mykonos Device API** file in Windows help format (*.chm). Refer to the software detail maual that comes with the software when looking for detailed information about the API commands.
- **DLL Help File** opens **ADI ZC706 TCPIP Client DLL** file in Windows help format (*.chm). Refer to this document when looking for detailed information about functions to control the device that use the Xilinx ZC706 FPGA platform.
- **About** opens an information window about the transceiver evaluation software (TES) and delay-locked loop (DLL) versions installed on the PC as well as for the software and firmware versions installed on the ZYNQ SD card. It also displays information about the internal ARM firmware version of the device (see Figure 214).



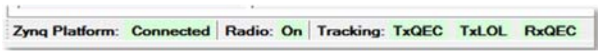
AD9371 Transceiver Evaluation Software

GUI Version: 2.0.67
DLL Version: 1.5.1.3565
Cmd Server Version: No Connection
Fpga Version: No Connection
Arm Version: No Connection
Analog Devices Inc.
Copyright (c) 2017

*Figure 214. TES Help **About** Window*

System Status Bar

The TES provides the user with visual information about the current state of the evaluation system (see Figure 215).



Zynq Platform: Connected | Radio: On | Tracking: TxQEC TxLOL RxQEC

Figure 215. TES Status Bar

The status bar information can be interpreted as follows:

- **Zynq Platform:** connected or disconnected.
  - **Connected**. PC established connection with the ZYNQ evaluation system,
  - **Disconnected**. No connection between PC and the ZYNQ evaluation system.
- **Radio:** on or off.
  - **On**. The device is enabled and ready to transmit/receive.
  - **Off**. The device must be initialized and moved into the radio on state before data can be transmitted or received.
- **Tracking:** TxQEC, TxLOL, or RxQEC.
- **TxQEC**, **TxLOL**, **RxQEC**. Those controls display the status of the tracking calibrations used by the device.
  - Green control indicates calibration is enabled and active.
  - Red control indicates calibration is enabled but not active.
  - Grey control indicates calibration is disabled (using the calibration tab described in the Calibration Tab section of this user guide).
- **Programmed Successfully** indicates the progress when programming the evaluation system (see Figure 216).

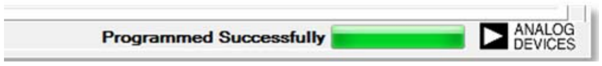


Figure 216. Program Status Bar

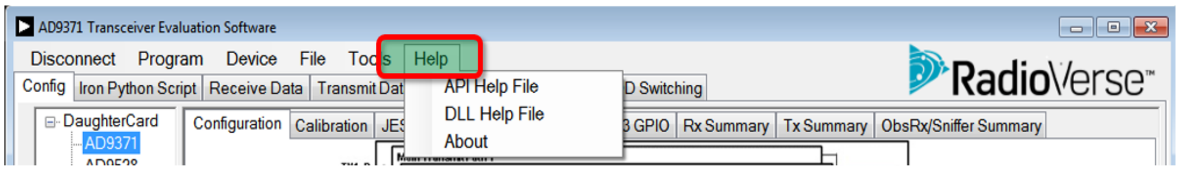


Figure 217. TES Help Dropdown Menu

RECEIVER SETUP

Rx Signal Chain

After configuring the TES using the **Config** tab and programming the system by executing the **Program** function, the system is ready for normal operation. Select **Receive Data** to open the **RxDataPlot** function, as shown in Figure 218. From the **Receive Data** tab, the user can enter the radio frequency (RF) Rx center frequency in megahertz, and set the Rx gain by entering the desired gain index for each Rx channel. The gain index refers to the value in the programmable gain index table. Refer to the Gain Control section for details on implementing the gain index table. The user can also enable or disable Rx1/Rx2 quadrature error correction (QEC) tracing calibrations as well as rerun Rx initialization calibrations from this window.

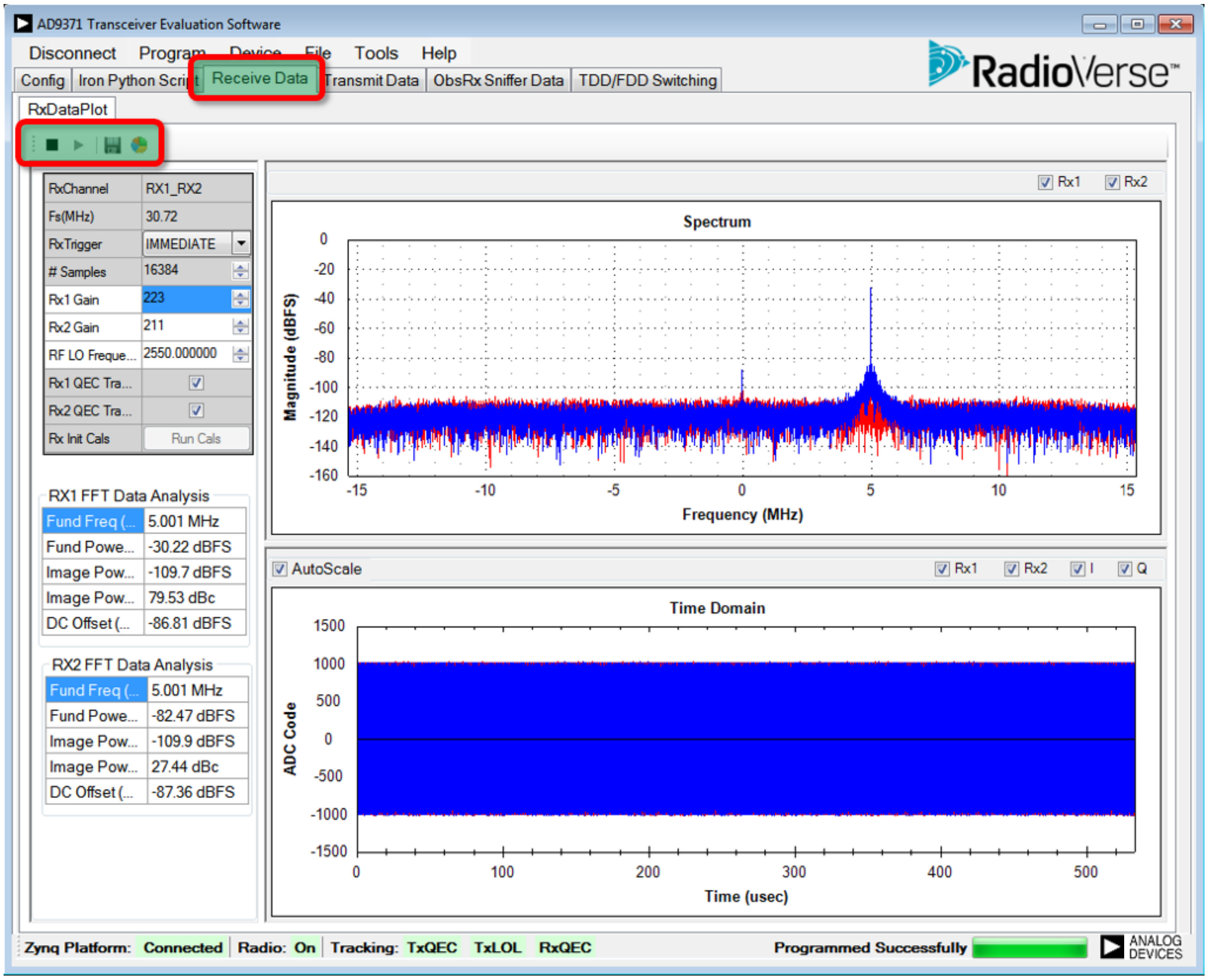
Click the play symbol in the **Receiver Data** tab, to move the device to the receive state and to generate graphs for the received data in both frequency and time domains. An example of a captured waveform is shown in Figure 218.

The upper plot displays the fast Fourier transform (FFT) result. Check the corresponding boxes to select if both Rx1 and Rx2 data are displayed in this window, or only one of type of data is displayed.

The lower plot shows the time domain waveform.

Check the corresponding boxes to select if both Rx1 and Rx2 data are displayed in this window, or only one of type of data is displayed. The user can also select if only I or only Q data is displayed in the same manner.

The time domain waveform display supports zoom function by selecting the region of the time plot to zoom in to. Right-click on the **Time Domain** window and select **Undo All Zoom/Pan** to return the time domain plot to its original scale. Check the **AutoScale** box to enable automatic scale in the time domain plot.



RxChannel	RX1_RX2
Fs(MHz)	30.72
RxTrigger	IMMEDIATE
# Samples	16384
Rx1 Gain	223
Rx2 Gain	211
RF LO Freque...	2550.000000
Rx1 QEC Tra...	✓
Rx2 QEC Tra...	✓
Rx Init Cals	Run Cals

RX1 FFT Data Analysis

Fund Freq (...	5.001 MHz
Fund Powe...	-30.22 dBFS
Image Pow...	-109.7 dBFS
Image Pow...	79.53 dBc
DC Offset (...	-86.81 dBFS

RX2 FFT Data Analysis

Fund Freq (...	5.001 MHz
Fund Powe...	-82.47 dBFS
Image Pow...	-109.9 dBFS
Image Pow...	27.44 dBc
DC Offset (...	-87.36 dBFS

Zynq Platform: Connected | Radio: On | Tracking: TxQEC TxLOL RxQEC | Programmed Successfully

*Figure 218. Rx **Receiver Data** Tab*

If the fast Fourier transform (FFT) analysis is selected (multicolored pie chart symbol), basic analysis information from the FFT is displayed on the left side of the screen. The FFT results are displayed separately for each Rx channel.

Select among the following in the **RxTrigger** dropdown box:

- **IMMEDIATE** starts the capture as soon as the SPI command is received to initiate capture.
- **EXT_SMA** starts the capture when a high level is present at Connector J68 on the ZYNQ platform.
- **TDD_SM_PULSE** hands over control of the Rx datapaths to the state machine that is implemented in the ZYNQ on-board field programmable gate array (FPGA). Use this option when the device operates in the time division duplexed (TDD) mode.

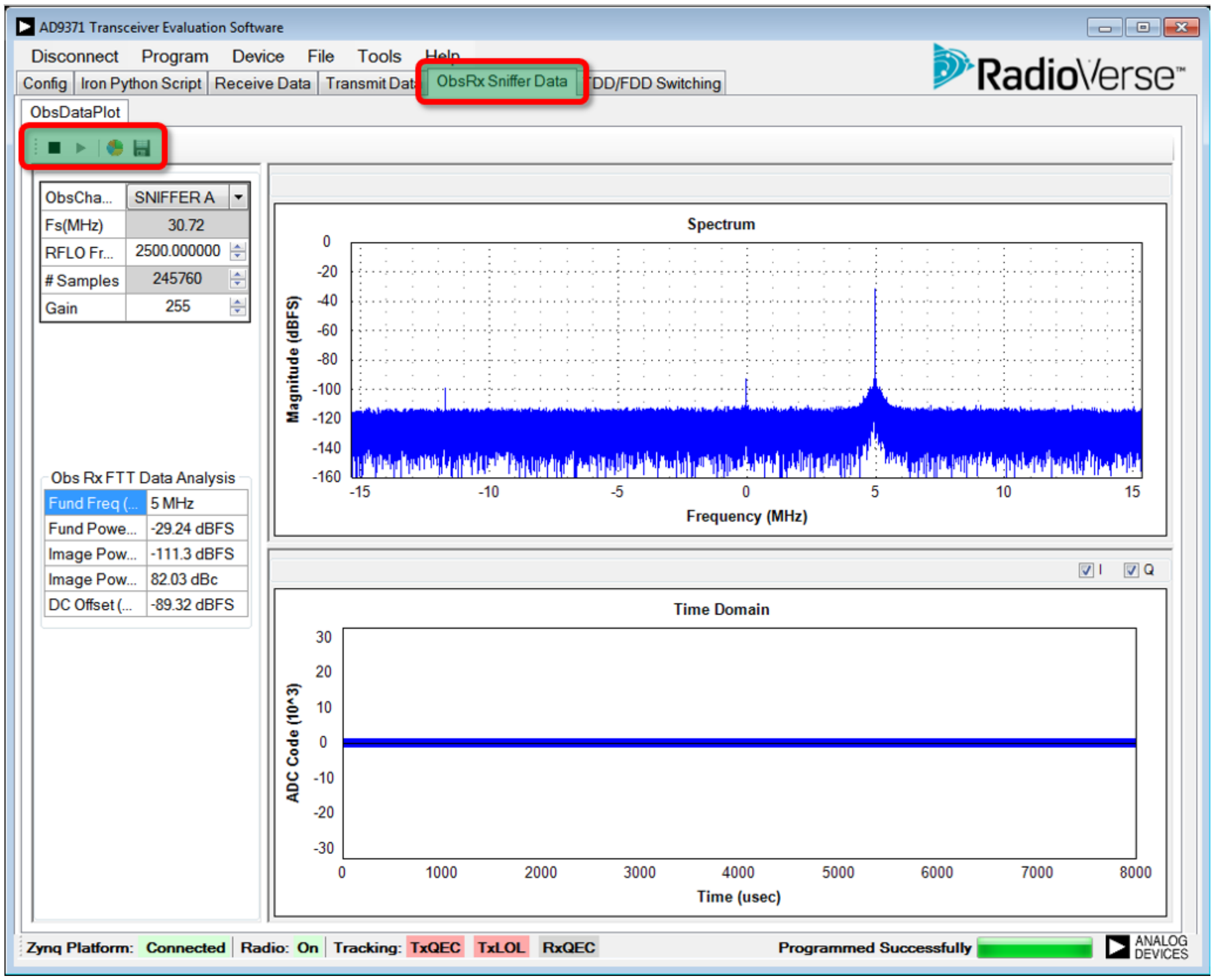
The floppy disk icon saves the receive data to a file. Selecting this option opens a window allowing selection of the format for the exported data. The file type can be specified as one of the following:

- **Agilent Data.** The TES adds a header to the saved file that Agilent VSA software can read and use to demodulate the data. The header is followed by data stored in I <TAB> Q [new_line] format.
- **No Header (Tab delimited)**. The TES saves data as a text file where I data is separated by <TAB> from Q data. Each data record is finished with a [new_line] character. There is no header information stored in this file format.
- **No Header (Comma delimited)**. The TES saves data as a text file where I data is separated by a comma [,] from Q data. Each data line is finished with a [new_line] character. There is no header information stored in this file format.

Additional settings in the **Receive Data** tab include the following:

- **# Samples**: the number of points saved to the file is determined by the number of samples selected in this box (see Figure 218).
- **Rx Init Cals**: click **Rx Init Cals** to rerun initial Rx calibrations. When calibrations execute, the button changes its appearance to running. Do not apply an input signal to the Rx input when performing an initialization calibration.
- **Rx1 QEC Tracking** and **Rx2 QEC Tracking**: enable or disable Rx1/Rx2 QEC tracking calibrations. Check **Rx1 QEC Tracking** to enable tracking calibration for the Rx1 path and check **Rx2 QEC Tracking** to enable tracking calibration for the Rx2 path. Tracking calibrations operate when an Rx signal path receives data.
- **ObsRx Sniffer Data**: opens the **ObsDataPlot** page (see Figure 219). When this tab is open, the user can enter the observation receive (ObsRx) radio frequency (RF) center frequency in megahertz, and set the ObsRx gain by entering the desired gain index. The gain index refers to the value in the programmable gain index table. See the Gain Control section for details on implementing the gain index table.
- **ObsChannel**: This dropdown menu (see Figure 219) offers the following input choices:
  - **Internal path** allows the ObsRx path to be used by internal calibrations. See the Gain Control section for details on calibration requirements.
  - **SNIFFER A**—Sniffer A input is the only sniffer input accessible on the device evaluation system.
  - **SNIFFER B**—It is not used and it is not available.
  - **SNIFFER C**—It is not used and it is not available.
  - **ORx1 with TxLO**—Used to select the ORx1 channel and the Tx LO PLL.
  - **ORx2 with TXLO**— Used to select the ORx2 channel and the Tx LO PLL.
  - **ORx1 with SNIFFERLO**— Used to select the ORx1 channel and the SNIFFERLO PLL.
  - **ORx2 with SNIFFERLO**— Used to select the ORx2 channel and the Tx LO PLL.

When clicking the play symbol in the **ObsRx Sniffer Data** tab, the device moves to the receive state and graphs the output data. An example of a captured waveform is shown in Figure 219.


AD9371 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
Config Iron Python Script Receive Data Transmit Data ObsRx Sniffer Data TDD/FDD Switching
RadioVerse
ObsDataPlot
Spectrum
Magnitude (dBFS)
Frequency (MHz)
Time Domain
ADC Code (10^3)
Time (usec)
I Q
Zynq Platform: Connected Radio: On Tracking: TxQEC TxLOL RxQEC
Programmed Successfully
ANALOG DEVICES
14652-346

ObsCha...	SNIFFER A
Fs(MHz)	30.72
RFLO Fr...	2500.000000
# Samples	245760
Gain	255

Obs Rx FTT Data Analysis

Fund Freq (...	5 MHz
Fund Powe...	-29.24 dBFS
Image Pow...	-111.3 dBFS
Image Pow...	82.03 dBc
DC Offset (...	-89.32 dBFS

Figure 219. Observation Rx Receive Tab

The upper plot displays the fast Fourier transform (FFT) result and the lower plot shows the time domain waveform. The user can select if only I or only Q data is displayed. The time domain waveform display supports a zoom function by selecting the region of the time plot to zoom in to. Right-click the **Time Domain** window and select**Undo All Zoom/Pan** to return the time domain plot to its original scale.

If the FFT analysis is selected (multicolored pie chart symbol), basic analysis information from the FFT displays on the left side of the screen.

Click the floppy disk icon to save the data received by the observation receive (**ObsRx**) channel. By selecting this window, users can select the format for the exported data.

When a file type chosen, the following happens within the file chosen:

- **Agilent Data**. The TES adds a header to the saved file that Agilent VSA software can read and use to demodulate the data. The header is followed by data stored in I <TAB> Q [new_line] format.
- **No Header (Tab delimited)**. Saves data as a text file where I data is separated by <TAB> from Q data. Each data record is finished with [new_line] character. There is no header information in stored this file format.
- **No Header (Comma delimited)**. Saves data as a text file where I data is separated by comma [,] from Q data. Each data line is finished with [new_line] character. There is no header information stored in this file format.
- **# Samples**. The number of points saved to the file is determined by the number of samples selected in the **# Samples** box.

TRANSMITTER SETUP

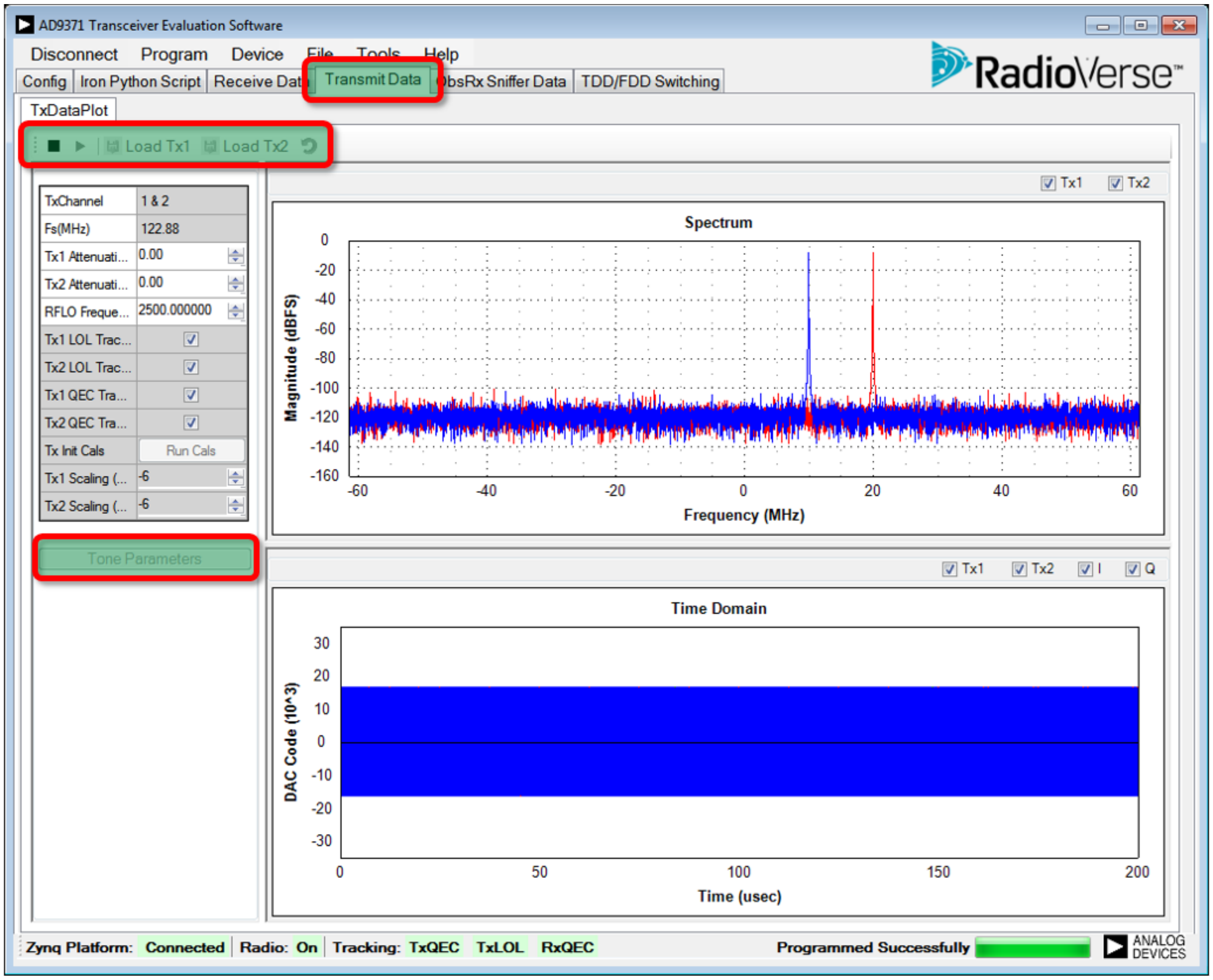
Selecting the **Transmit Data** tab opens the page shown in Figure 220.

The upper plot displays the fast Fourier transform (FFT) result. Check the corresponding box(es) to select if both Tx1 and Tx2 data are displayed in this window, or if only one type of data is displayed.

The lower plot shows the time domain waveform. Check the corresponding box(es) to select if both Tx1 and Tx2 data are displayed in this window, or if only one type of data is displayed. The user can also select if only I or only Q data is displayed.

The time domain waveform display supports zoom function by selecting the region of the time plot to zoom in to. To return the plot to its original scale, right-click **Time Domain** window and select **Undo All Zoom/Pan**. To enable automatic scaling in the time domain plot, check the **AutoScale** box.

From the **Transmit Data** tab, the user can enter the radio frequency (RF) Tx center frequency in megahertz, change the attenuation level independently for each Tx output, enable/disable various calibrations, control data scaling, and transmit continuous wave (CW) tones or a desired Tx data file.



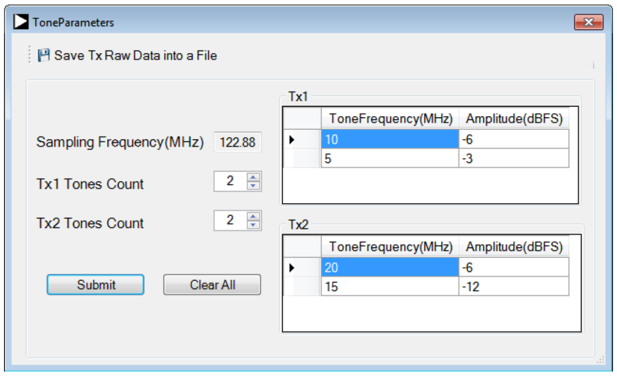
TxChannel	1 & 2
Fs(MHz)	122.88
Tx1 Attenuati...	0.00
Tx2 Attenuati...	0.00
RFLO Freque...	2500.000000
Tx1 LOL Trac...	✓
Tx2 LOL Trac...	✓
Tx1 QEC Tra...	✓
Tx2 QEC Tra...	✓
Tx Init Cals	Run Cals
Tx1 Scaling (...	-6
Tx2 Scaling (...	-6

Figure 220. ***Transmit Data*** *Tab*

Transmitter Data Options

The transceiver evaluation software (TES) provides the following options for inputting transmitter data:

- A single tone or two tones can be generated by the evaluation system using the **Tone Parameters** menu (see Figure 221).



Sampling Frequency(MHz) 122.88

Tx1 Tones Count 2

Tx2 Tones Count 2

Tx1

ToneFrequency(MHz)	Amplitude(dBFS)
10	-6
5	-3

Tx2

ToneFrequency(MHz)	Amplitude(dBFS)
20	-6
15	-12

*Figure 221. Tx **ToneParameters** Setup Menu*

  Within **Tone Parameters**, the user can select the number of tones (1 or 2) for transmission on the selected Tx output. The user has control over the tone frequency offset with respect to the local oscillator (LO) frequency as well as tone amplitude in dBFS. Select **Save Tx Raw Data into a File** to store those signals in the form of text files. Before data can populate into those files, **Play** must be clicked.

- To select user generated data files, use the **Load Tx1** and **Load Tx2** buttons as follows:
  - Format these files as: I sample <tab> Q sample <new_line> per line. Each I or Q sample must be in a range between +32768 and −32767.
  - If values of I and Q samples are smaller, check the **Scaling Required** box in the **Load file** menu to scale the values up to numbers in the correct range.
  - File size is limited to 4 megasamples for each channel (I data = 4 megasamples maximum, and Q data = 4 megasamples maximum). The ZYNQ platform, allocates 134,217,728 MB for each buffer. Rx1, Rx2, ORx, Tx1, and Tx2 have separate buffers. Each datapath uses four bytes (16-bit for I sample and 16-bit for Q sample); each datapath has 33,554,432, 16-bit I/Q pairs dedicated for sample collection. At a 122.88 MSPS I/Q data rate, this translates to 273 ms of capture time for each channel.
- Press **Play** to enable device data transmitted on the Tx1/Tx2 outputs. It starts a process where the generated continuous wave (CW) data or the I/Q data in the Tx1 and Tx2 files are sent to the device. The data is stored on the ZYNQ motherboard RAM, and the RAM pointer loops through the data continuously until **Stop** is clicked.
- The **Tx1 Attenuation (dB)** input controls analog attenuation in the Tx1 channel. It provides 0.05 dB of attenuation control accuracy. **Tx2 Attenuation (dB)** performs the same operation on the Tx2 channel.
- **Tx1 Scaling (dBFS)** input controls digital scaling of data sent over the Tx1 channel. It can be varied in 1 dB steps. It is only available for Tx data loaded using the **Load Tx1/ Load Tx2** buttons. **Tx2 Scaling (dBFS)** performs the same operation on the data sent over the Tx2 channel.
- Check the **Tx1 LOL Tracking** box to enable Tx local oscillator leakage (LOL) tracking calibration. Calibration improves the LOL performance on the Tx1 channel. Check the **Tx2 LOL Tracking** box to perform the same operation on the Tx2 channel. To perform Tx LOL tracking calibrations, external circuitry is required to route the Tx signals back through an ORx receiver input. For more details, see the Hardware Setup for External Tx LO Leakage Calibration section. Note that for external Tx LOL tracking calibration, both transmitters must loop back to both observation receivers through splitters and attenuators.
- Check the **Tx1 QEC Tracking** box to enable a Tx quadrature error correction (QEC) tracking calibration on the Tx1 channel. Calibration improves the QEC performance. Check the **Tx2 QEC Tracking** box to perform the same operation on the Tx2 channel.
  Note that Tx1/Tx2 LOL and QEC tracking calibrations can only operate when the observational receive path is configured for use by internal calibrations. When the user enables Tx outputs, the TES automatically reconfigures the observational receive path to the internal calibration mode. The user can change the observational receive path at any time using the **ObsRx Sniffer Data** tab.
- Click **Tx Init Cals** to run Tx initialization QEC and LO leakage calibrations. Run these calculations first before transmitting real data. Terminate both Tx paths into spectrum analyzers unless only one Tx is monitored; in that case, terminate the unused Tx into a 50 Ω termination to avoid extended initial calibration times. Longer initial calibration times occur when they are running on both channels, and when a Tx channel is improperly terminated.

TDD MODE

The transceiver evaluation hardware together with the transceiver evaluation software (TES) provide capabilities to demonstrate time division duplex (TDD) operation. The following sections explain the setup and operation in the TES to observe TDD operation and characterize performance.

LTE TDD Frame Structure

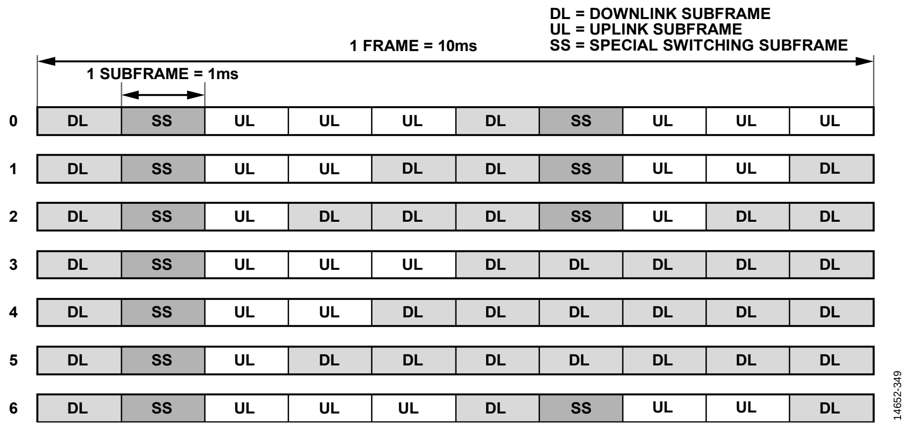
Preset configurations provided in the TES follow 3GPP™ specifications (TS 36.211 version 10.0.0 Release 10), in which Frame Structure Type 2 is utilized for TDD operation. In this configuration, each 10 ms radio frame consists of two, 5 ms half frames. Each half frame consists of five 1 ms subframes. The supported uplink and downlink configurations are listed in Table 189 where the following parameters are described for each subframe in a radio frame:

- D denotes a subframe reserved for downlink transmissions
- U denotes a subframe reserved for uplink transmissions
- S denotes a special subframe with the three fields: DwPTS (downlink pilot time slot), GP (guard period), and UpPTS (uplink pilot time slot)

The TES provides a preset configuration for all uplink and downlink configurations with both a 5 ms and 10 ms downlink to uplink switch point periodicity. All preset configurations are shown in Figure 222. In the case of the 5 ms downlink to uplink switch point periodicity, the special subframe exists in both half frames. In case of 10 ms downlink to uplink switch point periodicity, the special subframe only exists in the first half frame.

Table 189. Uplink and Downlink Configurations (Source: Table 4.2-2; 3GPP TS 36.211, Version 10.0.0, Release 10)

		Subframe Number									
Uplink and Downlink Configuration	**Downlink to Uplink Switchpoint Timing (ms)**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	5	D	S	U	U	U	D	S	U	U	U
1	5	D	S	U	U	D	D	S	U	U	D
2	5	D	S	U	D	D	D	S	U	D	D
3	10	D	S	U	U	U	D	D	D	D	D
4	10	D	S	U	U	D	D	D	D	D	D
5	10	D	S	U	D	D	D	D	D	D	D
6	5	D	S	U	U	U	D	S	U	U	D



DL = DOWNLINK SUBFRAME
UL = UPLINK SUBFRAME
SS = SPECIAL SWITCHING SUBFRAME
1 FRAME = 10ms
1 SUBFRAME = 1ms
0 DL SS UL UL UL DL SS UL UL UL
1 DL SS UL UL DL DL SS UL UL DL
2 DL SS UL DL DL DL SS UL DL DL
3 DL SS UL UL UL DL DL DL DL DL
4 DL SS UL UL DL DL DL DL DL DL
5 DL SS UL DL DL DL DL DL DL DL
6 DL SS UL UL UL DL SS UL UL DL
14652-349

Figure 222. Graphical Representation of Uplink and Downlink Configurations in the TDD Frame

Evaluation Hardware in TDD Mode

For time division duplex (TDD) operation, the initialization calibrations are run just as they are for frequency division duplex (FDD) mode. After the initialization calibrations are complete, the TDD command is used to configure the device into TDD mode.

The field programmable gate array (FPGA) on the ZYNQ platform contains a configurable TDD state machine to control the Tx enable, Rx enable, and GPIO hardware signals provided to the device. The **TDD/FDD Switching** tab in transceiver evaluation software (TES) allows enabling and disabling of the TDD state machine and configuring of the Tx/Rx regions in the TDD frame pulse to either a preset LTE™ TDD configuration or a user defined configuration.

The ARM processor inside the device uses the Rx enable, Tx enable, and GPIO signals controlled by the ZYNQ FPGA to determine when the device is in the Rx state, Tx state, ORx state, and so forth. Figure 223 is a timing diagram of the Tx enable and Rx enable signals during the LTE Configuration 0 type frame. The device responds based on the signal levels of TX_ENABLE and RX_ENABLE. Note that, for proper calibration operation, the minimum required duration for the Rx enable or Tx enable signal is 800 μs.

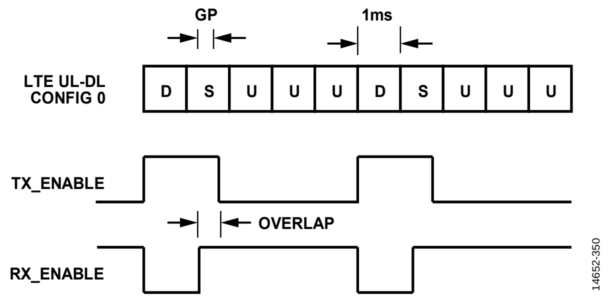
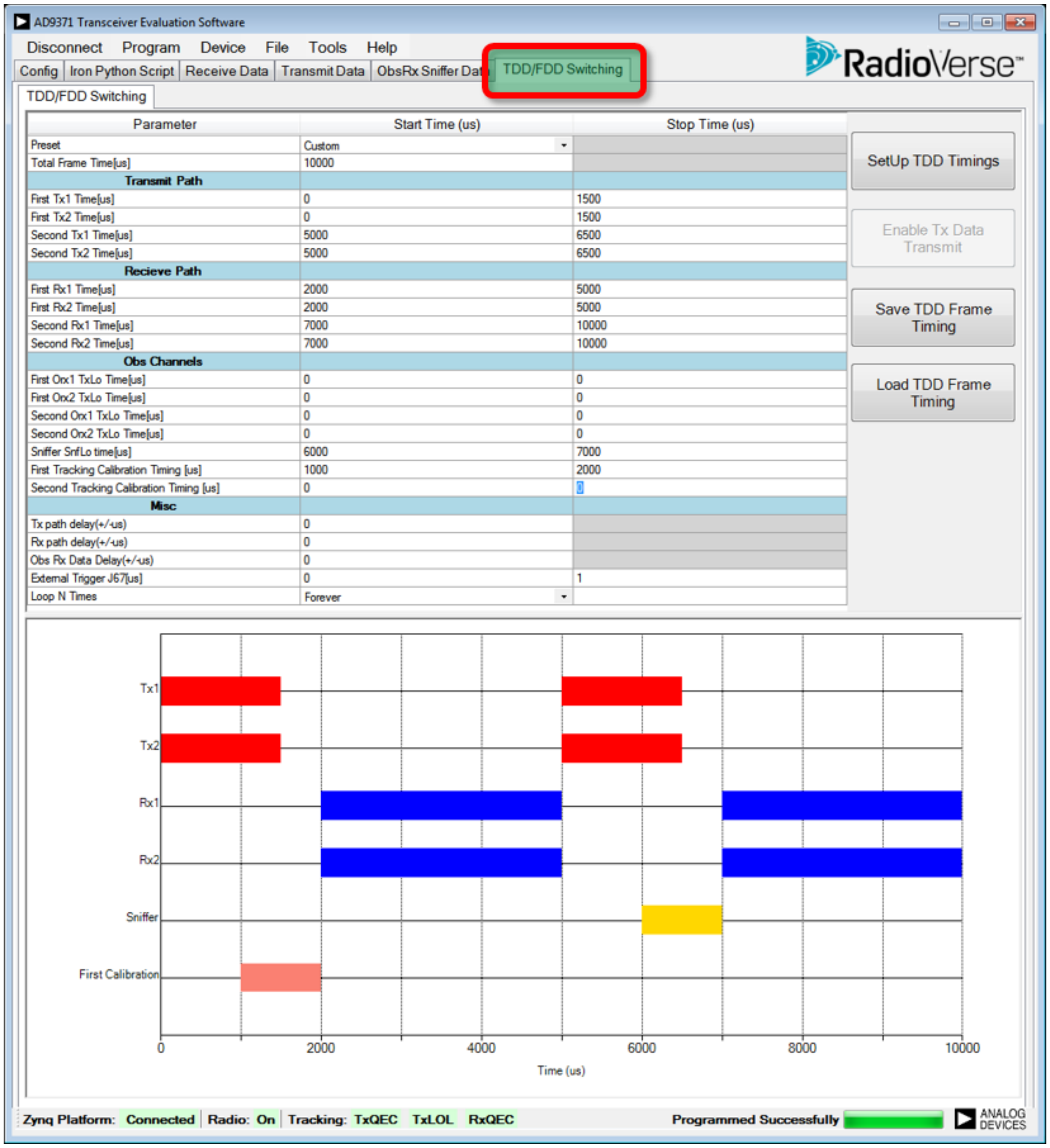


Figure 223. Timing Diagram Showing Example RX_ENABLE/TX_ENABLE Signaling for LTE Uplink and Downlink Configuration 0

TES Interface for TDD Mode

Figure 224 shows the transceiver evaluation software (TES) time division duplex (TDD) interface tab. The parameters available within are as follows:

- **Preset** allows the selection of one of eight LTE TDD Type 2 frame structures (described in the LTE TDD Frame Structure section) as well as provides options for user specific TDD frame timing, such as custom mode or custom LTETDD0 mode.
  - In custom mode, all parameters described in this section can configure the desired radio frequency (RF) paths with user specific timing.
  - The TES also provides a special mode, custom LTETDD0 mode that configures the hardware and software with LTE TDD 0 frame timing, optimized specifically to suit the evaluation system. The TES also provides Tx data files with timing optimized for this particular mode. The **Resources** subfolder inside the TES installation folder is where the **TDD0_245.76_ Downlink_20MHz_TM3p1.txt** file is located.
- The **Total Frame Time[µs]** field determines the total length of a single TDD frame in microseconds.



Parameter	Start Time (us)	Stop Time (us)
Preset	Custom	
Total Frame Time[us]	10000	
Transmit Path		
First Tx1 Time[us]	0	1500
First Tx2 Time[us]	0	1500
Second Tx1 Time[us]	5000	6500
Second Tx2 Time[us]	5000	6500
Recieve Path		
First Rx1 Time[us]	2000	5000
First Rx2 Time[us]	2000	5000
Second Rx1 Time[us]	7000	10000
Second Rx2 Time[us]	7000	10000
Obs Channels		
First Orx1 TxLo Time[us]	0	0
First Orx2 TxLo Time[us]	0	0
Second Orx1 TxLo Time[us]	0	0
Second Orx2 TxLo Time[us]	0	0
Sniffer SnfLo time[us]	6000	7000
First Tracking Calibration Timing [us]	1000	2000
Second Tracking Calibration Timing [us]	0	0
Misc		
Tx path delay(+/-us)	0	
Rx path delay(+/-us)	0	
Obs Rx Data Delay(+/-us)	0	
External Trigger J67[us]	0	1
Loop N Times	Forever	

Zynq Platform: Connected | Radio: On | Tracking: TxQEC TxLOL RxQEC Programmed Successfully

Figure 224. The TES TDD Interface Tab

The **Transmit Path** section of the time division duplex (TDD) parameters allows the control the following fields:

- The **First Tx1 Time[µs]** field determines the beginning and end of the first Tx1 subframes (or group of subframes).
- The **Start Time [µs]** field determines the beginning of a subframe (or group of subframes) in a single frame.
- The **Stop Time [µs]** field determines the end of a subframe (or group of subframes) in a single frame.

  The transceiver evaluation software (TES) TDD interface follows the convention where a subframe (or group of subframes) is enabled at the end and loops back to the beginning of a frame border. The subframe (or group of subframes) beginning is marked at the end of a single frame. Figure 225 shows the naming conventions used in the TES.

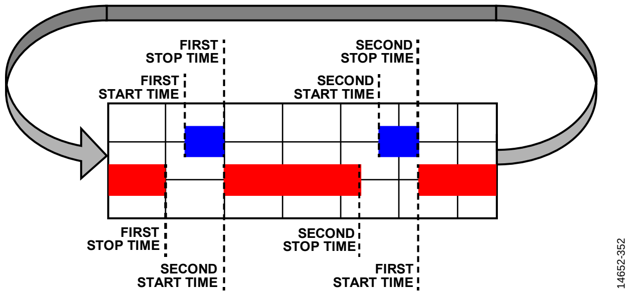


Figure 225. Naming Convention Used for TDD Start/Stop Description in the TES

- The **First Tx2 Time[µs]** field determines the beginning and end of the first Tx2 subframes (or group of subframes).
- If more than one Tx1 subframe (or group of subframes) is used, the **Second Tx1 Time[µs]** field determines the beginning and end of the second Tx1 subframes (or group of subframes).
- If more than one Tx2 subframe (or group of subframes) is used, the **Second Tx2 Time[µs]** field determines the beginning and end of the second Tx2 subframes (or group of subframes).

The **Receive Path** section of the TDD parameters allows control of the following fields:

- The **First Rx1 Time[µs]** field determines the beginning and end of the first Rx1 subframes (or group of subframes). It follows the same convention as described in Figure 225.
- The **First Rx2 Time[µs]** field determines the beginning and end of the first Rx2 subframes (or group of subframes).
- If more than one Rx1 subframe (or group of subframes) is used, the **Second Rx1 Time[µs]** field determines the beginning and end of the second Rx1 subframes (or group of subframes).
- If more than one Rx2 subframe (or group of subframes) is used, the **Second Rx2 Time[µs]** field determines the beginning and end of the second Rx2 subframes (or group of subframes).

The number of parameters available in the **Obs Channels** section determines operation details for the internal ORx/sniffer path. Note that there is only one internal observation/sniffer path; therefore, only one function mentioned as follows can be active at any given time (see Figure 224):

- The **First ORx1 TxLO Time[µs]** field determines the beginning and end of the first ORx1 subframes (or group of subframes).
- The **First ORx2 TxLO Time[µs]** field determines the beginning and end of the first ORx2 subframes (or group of subframes).
- If more than one ORx1 subframe (or group of subframes) is used, the **Second ORx1 TxLO Time[µs]** field determines the beginning and end of the second ORx1 subframes (or group of subframes).
- If more than one ORx1 subframe (or group of subframes) is used, the **Second ORx2 TxLO Time[µs]** field determines the beginning and end of the second ORx2 subframes (or group of subframes).
- The **Sniffer SnfLO time[µs]** field determines the beginning and end of the sniffer subframes (or group of subframes).
- The **First Internal Calibration Timing[µs]** field determines the beginning and end of the first internal calibration window. Use the minimum duration of 800 µs for the internal calibration window.
- The **Second Internal Calibration Timing[µs]** field determines the beginning and end of the second internal calibration window. Use the minimum duration of 800 µs for the internal calibration window.

The **Misc** section of the TDD parameters allows control of the following fields (see Figure 224):

- The **Tx path delay (+/-µs)** field allows the user to delay data sent to the Tx path over the JESD204B interface in reference to the TX_ENABLE signal.
- The **Rx path delay (+/-µs)** field allows the user to delay data received from the Rx path over the JESD204B interface in reference to the RX_ENABLE signal.
- The **Obs Rx Path Delay (+/-µs)** field allows delaying of data received from the ORx path over the JESD204B interface in reference to the ORx_ENABLE signal.
- In TDD mode, the device evaluation hardware generates a pulse on SMA Connector J67, located on the ZYNQ platform. The **External Trigger J67(µs)** parameter controls the position and the width of that pulse in reference to the start of the TDD frame.
- The **Loop N Times** option allows control of the number of loop repetitions. The allowable range is from 1 to 15, or the repetitions loop until stopped by the user.

The bottom part of the **TDD/FDD Switching** tab in the TES provides a diagram of the timing parameters entered in the table shown in Figure 224. This feature allows the user to visually check activities on the Rx, Tx, and ORx datapaths.

The time division duplex (TDD) page (see Figure 224) also contains four buttons to interact with the user. A description of the functionalities provided by these buttons follows:

- Press **SetUp TDD Timing** to cause the current TDD configuration stop/start parameters from the table to be written into the field programmable gate array (FPGA) and sets up the state machine for operation. This button also zeroes the Tx datapath, resets the Tx RAM pointer to the start address of the data, and then reconnects the Tx RAM in the ZYNQ FPGA to the Tx datapath. Finally, it enables the TDD state machine and starts the data. After the evaluation system is in the TDD state, this button changes its name to **Disable TDD**. Press **Disable TDD** to stop the TDD state machine.
- After the user sets up TDD mode and presses the **SetUp TDD Timing** button, the device evaluation system enables the TDD state machine, and the TDD mode becomes operational. There is no data present at the Tx output until the user presses the **Enable Tx Data Transmit** button. This button enables the data transfer in the FPGA. The Tx datapath is zeroed until the **Enable Tx Data Transmit** button is pressed (data does not start until the **Enable Tx Data Transmit** button is pressed). After being placed in TDD mode, the Tx data is sent continuously to the device through the JESD204B link, which is not gated by the TX_ENABLE signal. Therefore, the content of the TDD Tx data files must be properly time aligned to the TDD state machine signals.
- The **Save TDD Frame Timing** button saves TDD timing to the file in a text readable format.
- The **Load TDD Frame Timing** button loads TDD timing from the previously saved TDD timing file using the save TDD frame timing option.

Setting Up TDD Functionality

Perform the following steps to operate the device evaluation system and the transceiver evaluation software (TES) in TDD mode.

Hardware Configuration

Follow the hardware configuration described instructions found in the Hardware Setup section. If external local oscillator (LO) calibrations are used, setup must contain external hardware connections between the Tx outputs and ORx inputs, as described in the Hardware Setup for External Tx LO Leakage Calibration section.

The device evaluation system provides a synchronization pulse on the ZYNQ motherboard SMA Connector J67. Use this pulse to synchronize external measurement equipment. Fine tuning of this signal can be applied using the TES interface described in the TES Interface for TDD Mode section. After all hardware is connected properly, the user can start configuring the software.

TES Configuration

A number of steps must be performed before enabling TDD mode using the TES. The following list provides guidelines for completing these steps:

1. Using the TES interface described in the Configuring the AD9371Configuring the section, perform the following steps:
   a. Select profiles for the Rx channels, Tx channels, and ORx or SnRx channels (if used).
   b. Set the same frequency for the Tx PLL and the Rx PLL. In TDD mode, both the Rx and Tx operate at the same frequency; therefore, select the same carrier frequency for both the Rx and Tx radio frequency (RF) phase-locked loops (PLLs).
2. Use the calibration page described in the Configuring the AD9371 section, to enable the desired calibrations.
3. After all configurations are complete, click **Program** to program the device evaluation system (see Figure 207).
4. After the device evaluation system is programmed, move to the **Receiver Data** tab shown in Figure 218. In this tab, perform the following actions:
   a. Set **RxTrigger** to **TDD_SM_PULSE** value.
   b. Set the number of samples in **# Samples** to at least 1 frame length (10 ms for standard LTE TDD Type 2 frame structures described in the LTE TDD Frame Structure section).
   c. Click **Play**.
5. Click the **Transmit Data** tab shown in Figure 220. In this tab, perform the following actions:
   a. Load the data files that are time aligned with the desired LTE TDD Type 2 frame structure. The TES provides an example data file that can be used with LTE TDD 0 Type 2 frame structure. See the TES Interface for TDD Mode section for more information.
   b. Click the **Play** button.
6. Select the desired TDD timing profile using the **TES TDD/FDD Switching** tab shown in Figure 224. A detailed description of this tab is provided in the TES Interface for TDD Mode section. After all timing settings are configured, perform the following actions:
   a. Click **SetUp TDD Timings**.
   b. Click **Enable Tx Data Transmit**.

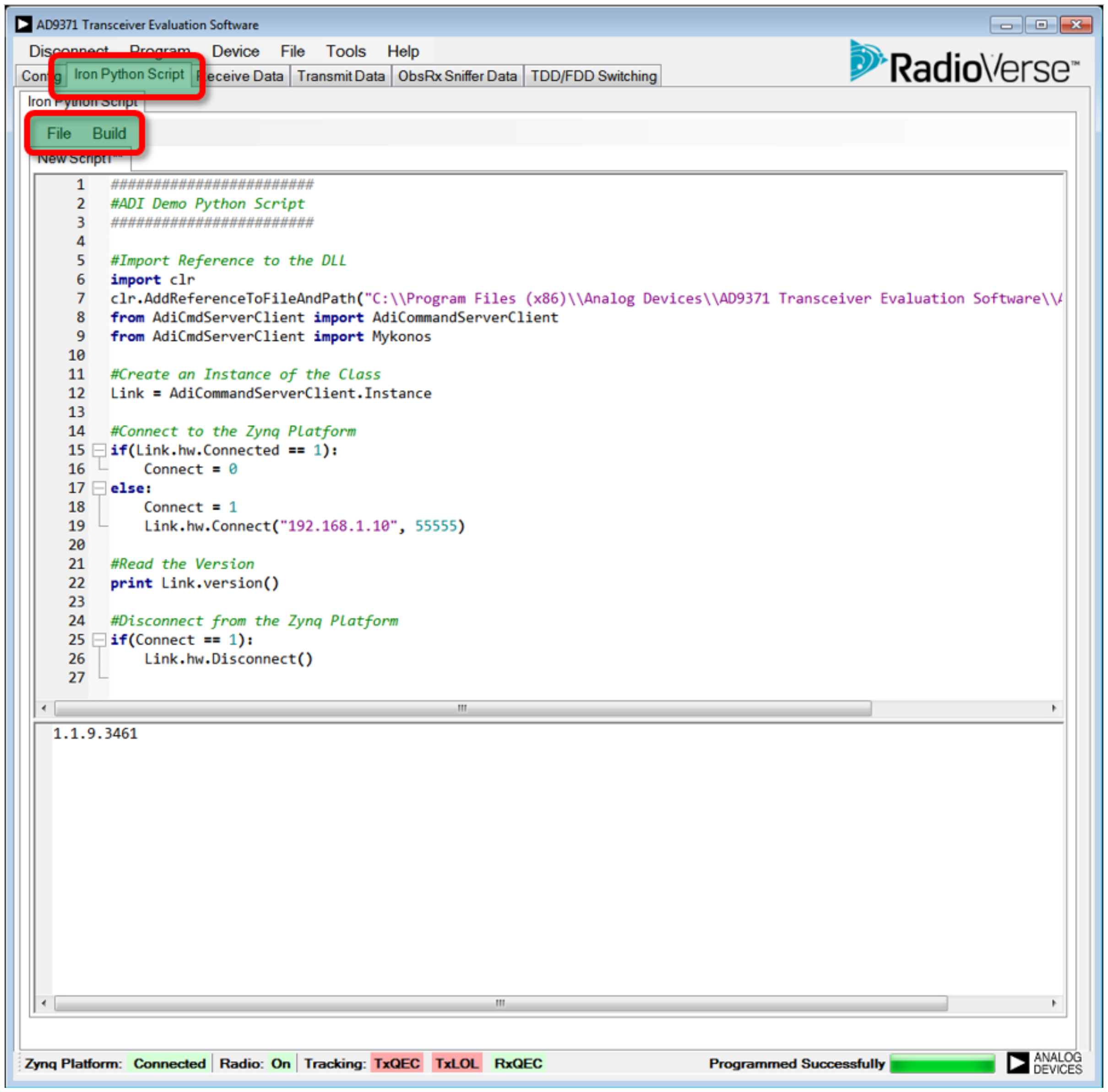
If the user does not follow this sequence, the TES software provides real-time pop-up warning messages. These messages inform the user about possible misconfigured settings.

SCRIPTING

After the user configures the device to the desired profile, a script can generate with all application programming interface (API) initialization calls in the form of IronPython functions. The **Tools** > **Create Script** > **Python** function can accomplish this task. Refer to the Tools Dropdown Menu section for more details.

The **Iron Python Script** tab allows the user to use IronPython to write a unique sequence of events and then execute them using the device evaluation system.

Scripts generated using the **Tools** > **Create Script** > **Python** function can load, modify if needed, and run in the **IronPython Script** tab. Figure 226 shows the **Iron Python Script** tab after executing the **File** > **New** function in the **Iron Python Script** tab. The top portion of the window contains IronPython script commands, and the bottom part of the window displays the script output.


AD9371 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
Config Iron Python Script Receive Data Transmit Data ObsRx Sniffer Data TDD/FDD Switching
RadioVerse™
Iron Python Script
File Build
New Script1
Zynq Platform: Connected Radio: On Tracking: TxQEC TxLOL RxQEC Programmed Successfully
ANALOG DEVICES
14652-353

```
########################
#ADI Demo Python Script
########################

#Import Reference to the DLL
import clr
clr.AddReferenceToFileAndPath("C:\\Program Files (x86)\\Analog Devices\\AD9371 Transceiver Evaluation Software\\
from AdiCmdServerClient import AdiCommandServerClient
from AdiCmdServerClient import Mykonos

#Create an Instance of the Class
Link = AdiCommandServerClient.Instance

#Connect to the Zynq Platform
if(Link.hw.Connected == 1):
    Connect = 0
else:
    Connect = 1
    Link.hw.Connect("192.168.1.10", 55555)

#Read the Version
print Link.version()

#Disconnect from the Zynq Platform
if(Connect == 1):
    Link.hw.Disconnect()
```

```
1.1.9.3461
```

*Figure 226. **Iron Python Script** Window*

The Iron Python Script tab offers a number of options to manipulate the editing and execution of the Iron Python scripts. The **File** drop-down menu in the **Iron Python Script** tab, shown in Figure 227, offers the user following options:

- **New**. This function creates a new Iron Python script that connects to the device evaluation system and checks the API version operating on the ZYNQ hardware.
- **Load**. This function allows the user to load previously stored Iron Python scripts.
- **Save** and **Save As**. These functions allow the user to store Iron Python scripts.
- **Close**. This function closes the currently active Iron Python script tab.

The **Build** dropdown menu in the **Iron Python Script** tab, shown in Figure 228, offers the user following options:

- **Run**. This function executes the Iron Python script open in the active script tab using the device evaluation hardware. Script output is displayed in bottom side of the Iron Python script tab.
- **Clear Script**. This function clears the Iron Python script editing window.
- **Clear Output**. This function clears the Iron Python script output window.

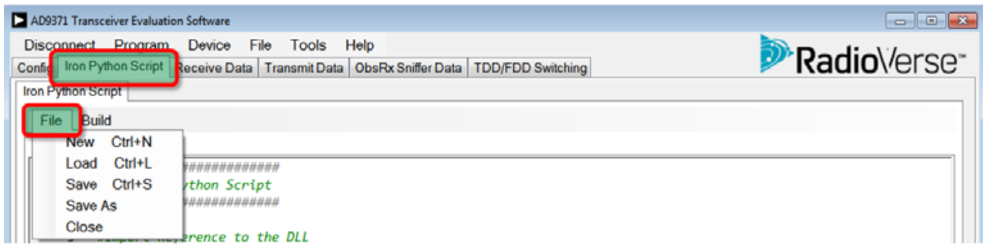

AD9371 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
RadioVerse
Config Iron Python Script Receive Data Transmit Data ObsRx Sniffer Data TDD/FDD Switching
Iron Python Script
File Build
New Ctrl+N
Load Ctrl+L
Save Ctrl+S
Save As
Close
14652-354

Figure 227. **File** Menu in the **Iron Python Script** Window

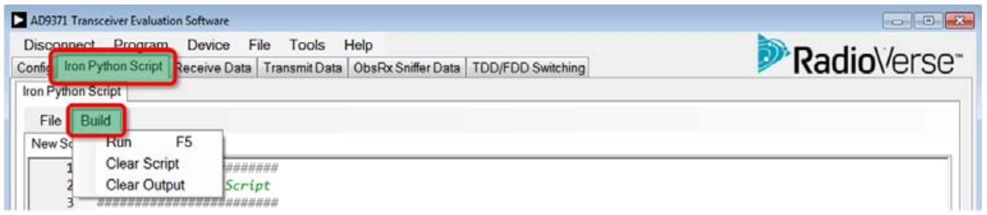

AD9371 Transceiver Evaluation Software
Disconnect Program Device File Tools Help
RadioVerse
Config Iron Python Script Receive Data Transmit Data ObsRx Sniffer Data TDD/FDD Switching
Iron Python Script
File Build
Run F5
Clear Script
Clear Output
14652-355

Figure 228. **Build** Menu in the **Iron Python Script** Window

IronPython Script Example

The following example, which is generated after executing the **File** > **New** function in the **IronPython Script** tab, connects to the device evaluation system and then checks and displays the application programming interface (API) version operating on the ZYNQ hardware.

```
########################
#ADI Demo Python Script
########################

#Import Reference to the DLL
import clr
clr.AddReferenceToFileAndPath("C:\\Program Files (x86)\\Analog Devices\\AD9371
Transceiver Evaluation Software\\AdiCmdServerClient.dll")
from AdiCmdServerClient import AdiCommandServerClient
from AdiCmdServerClient import Mykonos

#Create an Instance of the Class
Link = AdiCommandServerClient.Instance

#Connect to the Zynq Platform
if(Link.hw.Connected == 1):
    Connect = 0
else:
    Connect = 1
    Link.hw.Connect("192.168.1.10", 55555)

#Read the Version
print Link.version()

#Disconnect from the Zynq Platform
if(Connect == 1):
    Link.hw.Disconnect()
```

When using the Iron Python window, the user can execute any application programming interface (API) command.

The list of all API commands is provided by the transceiver evaluation software (TES). Review this list by executing **Help** > **DLL Help File.** When called in the **IronPython** window, rename all API functions to reflect the Iron Python mnemonic:

```
MYKONOS_ → Mykonos.
```

Add a header with a new class instance for a new connection. For example, after calling

```
#Create an Instance of the Class
Link = AdiCommandServerClient.Instance
```

The new class instance for device evaluation hardware is

```
Link.
```

An example of an API function called using Iron Python is as follows. If checking the gain index for Rx1 signal chain use the following API function:

```
MYKONOS_getRx1Gain()
```

The user calls the following Iron Python function (assuming that the platform was initialized using example code described previously):

```
print Link.Mykonos.getRx1Gain()
```

Troubleshooting

This section provides a quick help guide if the system is not operational. This guide assumes that the user followed all instructions and that the hardware configuration matches that described in this user guide.

Startup

No LED Activity

If there is no LED activity at startup, perform the following actions:

1. Check if the board is powered properly (12 V must be present at the J22 input). After powering on the ZYNQ platform (SW1 is turned on), the following is true:
   a. The fan on the ZYNQ platform is activated.
   b. A number of green LEDs on the ZYNQ platform near SW1 are on with no red LEDs active on the ZYNQ platform.
   c. The ZYNQ GPIO LEDs follow the sequence described in the Hardware Operation section.
2. If the LED sequence is not as described, check the jumper settings and the SW11 positions on the ZYNQ platform. If these are correct, check if the SD card is correct and properly inserted in the J30 socket. Use the SD card provided with the evaluation system.

If there is still a problem, and the user is certain that the ZYNQ platform is operational, contact an Analog Devices representative at www.analog.com/en/landing-pages/001/sdr-radioverse-pavilion/support.html.

LED Active, TES Reports That Hardware Not Connected

At startup, if the LEDs are active but the TES reports that hardware is not connected, perform the following actions:

1. Check if the Ethernet cable is properly connected between the PC used to run the TES and the ZYNQ platform. The LEDs on the ZYNQ platform next to the Ethernet socket flash when the connection is active.
2. If the cable is properly connected, check if Windows is able to communicate over the Ethernet port with the ZYNQ platform. Check if the IP number and open ports for the Ethernet connection used to communicate with the ZYNQ platform follow what is described in the Hardware Setup section.
3. Run **cmd.exe** on the Window operating system and then type ping 192.168.1.10. The user then sees a reply from the ZYNQ platform. If no reply is received, connection with the ZYNQ platform must be reexamined.
4. If connection with the ZYNQ platform is established but the TES still reports that hardware is not available, ensure that Port 22 (SSH) and Port 55555 (evaluation software) are not blocked by firewall software on the Ethernet connection used to communicate with the ZYNQ platform. Both ports must be open for normal operation. Refer to the Hardware Setup section for more details.

Error Handling

The TES provides the user with a number of error messages in case there are problems with hardware or software configuration. The error messages the TES displays provide a description of the problem encountered by the software. If an error description refers to the delayed-locked loop (DLL) command, refer to the API and DLL help files supplied with the TES.

DPD, CLGC, AND VSWR MEASUREMENT (AD9375 ONLY)

The AD9375 device variant 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: digital predistortion (DPD), closed-loop gain control (CLGC), and voltage standing wave ratio (VSWR) measurement.

This section describes the hardware setup and application programming interface (API) commands used to control these transmitter features. While the API descriptions are intended for customer software developers, the paragraphs describing the DPD graphical user interface (GUI) can guide the evaluation of the DPD algorithm performance by the systems designers of the customer. The combination of this information can develop system designs using these algorithms and to develop the code to integrate control into the baseband processor (BBP).

DPD OVERVIEW

The DPD is a feature available on the AD9375 that enables users to achieve higher power amplifier (PA) efficiency while still meeting adjacent channel leakage ratio (ACLR) requirements in the Tx signal chain for compliance with 3GPP and European Telecommunications Standards Institute (ETSI) standards for LTE and other technologies. The DPD works on the principle of predistorting the Tx data to cancel distortion caused by PA compression. The DPD engine in the AD9375 is based on a pruned implementation of generalized memory polynomials (GMP) that are a generalized subset of the well known Volterra series. The simplified polynomial used in the AD9375 models a large number of PA characteristics such as weak nonlinearities, temperature variation, and memory effects. Integration of the DPD into the transceiver chip results in significant system level cost, space, and power savings when compared to conventional external implementations. The DPD implementation on the AD9375 is especially well suited for use in a small cell application (typically 0.1 W to 10 W at the antenna), where significant cost savings can be achieved at conventional performance levels.

The DPD algorithm runs on the ARM processor of the AD9375 and calculates the coefficients and terms of the inverse PA model. This model predistorts the digital baseband signal before digital-to-analog conversion and transmission of samples to the Tx upconverter (this output becomes the radio frequency (RF) input to the PA). This computation is performed along with other transceiver operations as specified by the priorities of the ARM scheduler. The PA output is sampled using an external loopback to an observation receiver (ORx) port on the AD9375. A simplified representation of the implementation of DPD in AD9375 is shown in Figure 229.

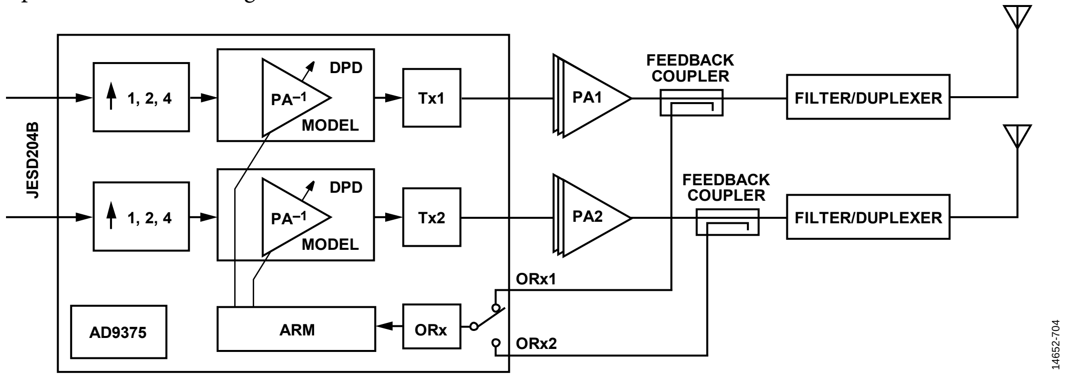


Figure 229. Representation of the Integrated DPD Implementation in AD9375

Figure 229 illustrates that the Tx1 and Tx2 digital datastreams are first upsampled by a factor of 1, 2, or 4 depending on the active Tx profile. The digital datastreams are sent via JESD204B interfaces. Upsampling allows the baseband processor (BBP) to transmit a lower rate on the JESD204B link than is needed for the full digital predistortion (DPD) bandwidth, saving valuable JESD204B resources, which directly translates to power savings and lower data rates in the digital front end (DFE). This upsampling is done to achieve a wide enough bandwidth expansion for the DPD algorithm to obtain optimal results. DPD algorithms in general require a bandwidth 3× to 5× larger than the signal bandwidth to correct for power amplifier (PA) nonlinearities that cause higher adjacent channel leakage ration (ACLR) levels. The ORx input is sparsely time sampled and fed to the ARM for DPD processing. The DPD engine then correlates the ORx and transceiver (Tx) samples to calculate the latest coefficients. The DPD engine performs a brief check on model error before updating the lookup tables (LUTs) that feed the correction coefficients into the DPD actuator hardware. Due to the relatively simple implementation of this algorithm, the overall time taken to react to sudden changes in Tx waveforms is relatively short and is typically less than 1 second (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 PA due to large model errors.

Sample Capture

Samples used to learn the latest model are captured by the ARM processor using a sparse sampling technique. Whenever the processor is available, it captures a time aligned sample set of seven Tx samples x(n), x(n − 1) … x(n − 6) and seven consecutive ORx samples y(n), y(n − 1) … , y(n − 6).

Time alignment is performed during the device initialization by generating a pseudonoise (PN) sequence and maximizing Tx to ORx correlation to within 1/16 of a sample period. Delay for alignment is implemented with a first in, first out (FIFO) and a 1/16 sample fractional delay filter. Additional fractional sample offsets from the initialized delay can be introduced with an application programming interface (API) command to tune modeling performance (see additionalDelayOffset in Table 193). Alignment accuracy and tuning is most important when performing DPD on wider bandwidth signals.

For a given x(n) and a set of three time aligned y(n) samples, a correlation computation involving 22 generalized memory polynomials (GMP) nonlinear functions of the y samples is performed. Sampling is random and on the fly; it can be interrupted and resumed at any time. Samples are spread out and tend to be less correlated so that the features are more independent and their correlation converges faster than with blocks of consecutive samples. In practice, less than 2048 sparse captures are required for a full adaptation update. Using only 22 features reduces the number of required samples because the number of features limits the degrees of freedom that need to be learned in the PA model.

DPD Actuator Model Configurations

A few configurable adjustments of the DPD actuator datapath are also supported. These are Model 0 through Model 3 and involve multiplexing various LUT outputs differently (see Figure 230 to Figure 233). As shown in accompanying figures, the magnitude squared data is put through a compander that has been optimized for LTE signals. The companded input is then used to address each of the four LUTs which contains the DPD coefficients. The output of the LUTs is then multiplexed depending on the model configuration being used (see modelVersion in Model 2 (see Figure 232 and Table 193) is recommended as a starting point for most power amplifiers (PAs), especially for gallium arsenide (GaAs) PAs, and gives good wideband performance, as does Model 3. Model 0 and Model 1 may achieve superior narrow-band performance in some cases. The absence of deeper delay terms in Model 0 prevents the DPD from overfitting on narrow-band signals. Model 1 has also been shown to give marginally better performance for some lateral diffused metal-oxide semiconductors (LDMOS) PAs.

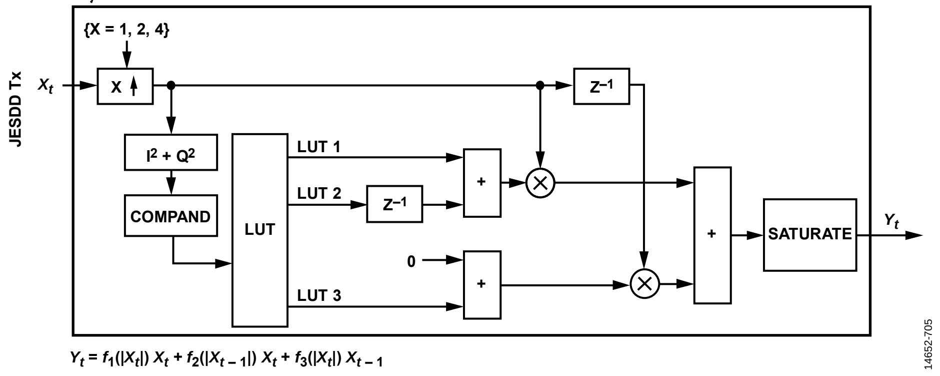
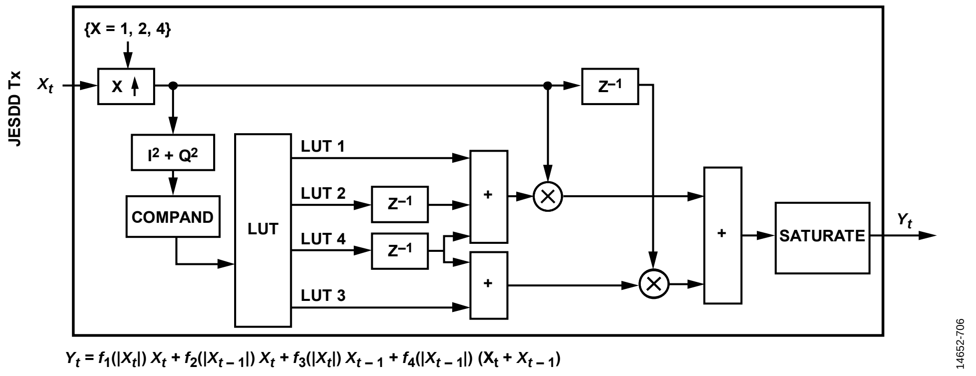
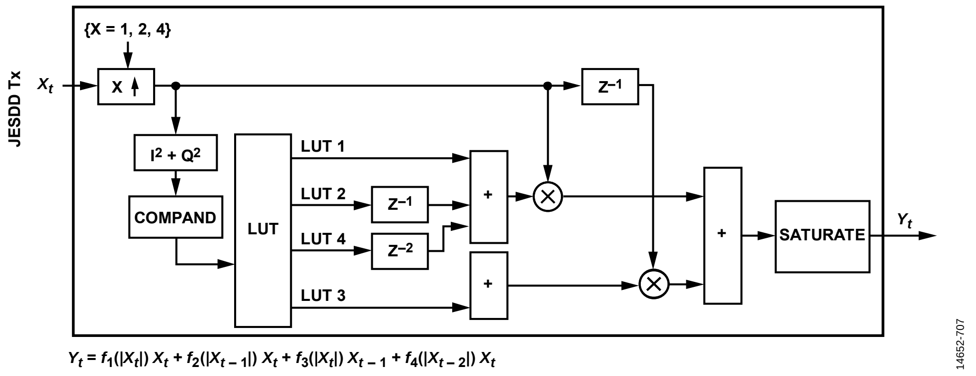


Figure 230. Model 0



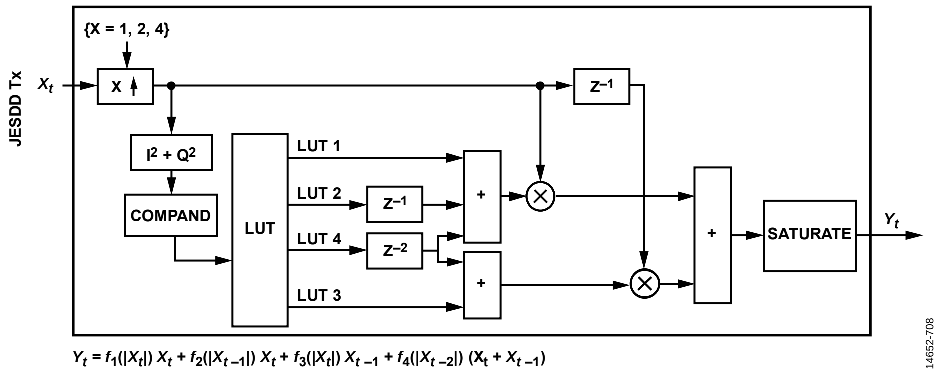
$Y_t = f_1(|X_t|)\,X_t + f_2(|X_{t-1}|)\,X_t + f_3(|X_t|)\,X_{t-1} + f_4(|X_{t-1}|)\,(X_t + X_{t-1})$

Figure 231. Model 1



$Y_t = f_1(|X_t|)\,X_t + f_2(|X_{t-1}|)\,X_t + f_3(|X_t|)\,X_{t-1} + f_4(|X_{t-2}|)\,X_t$

Figure 232. Model 2



$Y_t = f_1(|X_t|)\,X_t + f_2(|X_{t-1}|)\,X_t + f_3(|X_t|)\,X_{t-1} + f_4(|X_{t-2}|)\,(X_t + X_{t-1})$

Figure 233. Model 3

High Amplitude Model Priors

One of the challenges in a digital predistortion (DPD) system is how to handle large swings in input signal power. Immediately after a transition event (low to high or high to low power), the power amplifier (PA) amplifies in a different region of its operating curve than where the training samples for the current model were taken. At this moment, the model may be inaccurate and adjacent channel leakage ratio (ACLR) performance may degrade. Another model update may not occur for a few milliseconds, and the emissions are higher during this time.

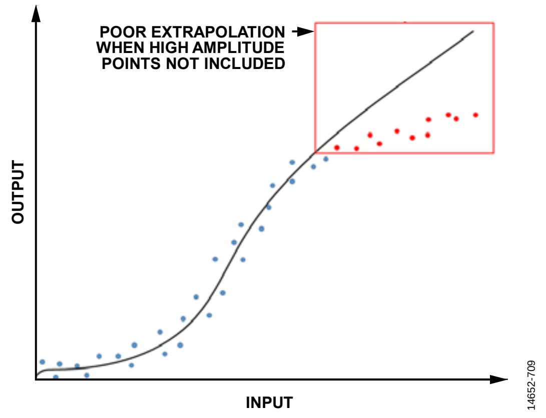


Figure 234. Model Accuracy Deviation with High Amplitude Swings

Typically, the challenging situations are low to high power transitions, where the model is first fit to small signal measurements, then the PA operates on high power data. At high amplitudes, the model is an extrapolation of polynomial fits at lower amplitudes.

If operating at low power for a long duration, leaky correlation averaging eventually diminishes high amplitude PA model information. One mitigating approach is to keep some high power measurement data in the correlation matrix at all times, even if the data is outdated. Stored high amplitude samples can be stratified across fixed amplitude bins. In every model regression, these samples can be included so that the polynomials fit these high power sample points as well. This technique is adequate to keep performance stable before a full update with fresh high amplitude training samples. However, this technique generally requires a large number of samples to work well, and it is hard to store enough samples.

Instead, the AD9375 DPD incorporates a probabilistic prior model on $\vec{\alpha}$ values (DPD actuator terms) when solving for new model coefficients, which provides information about what the higher order coefficients should be if they are not well defined based on the current low amplitude data.

When the prior model is Gaussian distributed (for example, some guess vector $\vec{\alpha}_0$ values and the associated precision matrix (P) are an inverse covariance matrix), the combined estimate of prior and new data follows Equation 57. The matrix (F) denotes the correlation feature matrix used by the DPD.

$$\vec{\alpha} = \left(F^H F + P\right)^{-1}\left(F^H \vec{y} + P\vec{\alpha}_0\right) \tag{57}$$

where:

$\vec{\alpha}$ values are the DPD model coefficient vector.
$F^H F$ is the autocorrelation of the feature matrix
P is the precision matrix.
y is ORx sample vector.
$\vec{\alpha}_0$ is the prior model vector.

In the AD9375 DPD, $\vec{\alpha}_0$ prior coefficients can be programmed at startup or during operation (if it is known that the operating condition of the PA is about to change) with the save and restore DPD model application programming interface (API). The prior model precision matrix P is scaled by a modelPriorWeight API parameter that scales the strength in the final solution (see modelPriorWeight in Table 193). Setting the weight higher causes the prior model precision matrix (P) to have more influence than the current data when determining the final coefficients used for the lookup tables (LUTs).

Optionally, the prior model coefficients can update automatically during high power data (by default, within 1 dB of the highest Tx rms power observed by the DPD) by setting this option in the API or GUI default: enabled. When automatically updated, P is the diagonal portion of the correlation matrix [$C_{YY} = F^H F$] and $\vec{\alpha}_0$ is the solution vector from the previous iteration of DPD that included high power rms data.

CLGC OVERVIEW

The closed-loop gain control (CLGC) feature in the AD9375 enables a constant gain level to be maintained at the ORx input (total gain from Tx output to ORx input) which translates to a constant output power (P_{OUT}) at the power amplifier (PA) for a given digital input level. The gain level is controlled by modifying the Tx attenuation and by specifying a desired loop gain value (here, gain means the net loop gain and attenuation combined, including all transceiver gain and attenuation blocks on-chip). Note that the CLGC must be enabled to establish the total desired loop gain in the system. The CLGC feature in the AD9375 reacts to changes in the overall system loopback, which includes PA and channel gain as well as ORx gain variations. ORx gain variations over time, temperature, and bandwidth are typically minimal; however, the user must refer to the AD9375 data sheet to verify the tolerance over these factors in the user-specific use case. Therefore, the CLGC primarily compensates for any PA gain drift over time and temperature by periodically monitoring and modifying the Tx attenuation to achieve a constant target gain. Note that the CLGC does not track when power at the ORx input results in a digital signal less than −39 dBFS to avoid damage to the PA by either increasing the Tx power too much, or by causing CLGC instability issues due to the low ORx power available (see the Other Considerations section for more information). This compensation protects the PA if the ORx gets disconnected, or if some other component in the loopback fails. A minimum Tx attenuation can also be configured to protect the PA during CLGC tracking (see tx1AttenLimit/tx2AttenLimit parameter in Table 195).

VOLTAGE STANDING WAVE RATIO MEASUREMENT OVERVIEW

The voltage standing wave ratio (VSWR) on a transmission line is defined as the ratio of the voltage maxima to the voltage minima along the line. The VSWR measurement feature in the AD9375 facilitates the computation of this quantity by means of using the ORx path that is time multiplexed to measure both the forward and reflected voltages. An example block diagram is shown in Figure 235.

Formally,

$$VSWR = \frac{1+\Gamma}{1-\Gamma} \tag{58}$$

where:
VSWR is the voltage standing wave ratio.

$$\Gamma = \frac{R/Tx}{F/Tx}$$

where:
R is the reflected power.
F is the forward power measured at the ORx.
Tx is the transmit power measured within the device.

Note that R and F must be calculated at different time intervals; therefore, the accuracy of the VSWR measurement can be impacted by signal statistics (VSWR performance with fast changing and dynamic signals may vary).

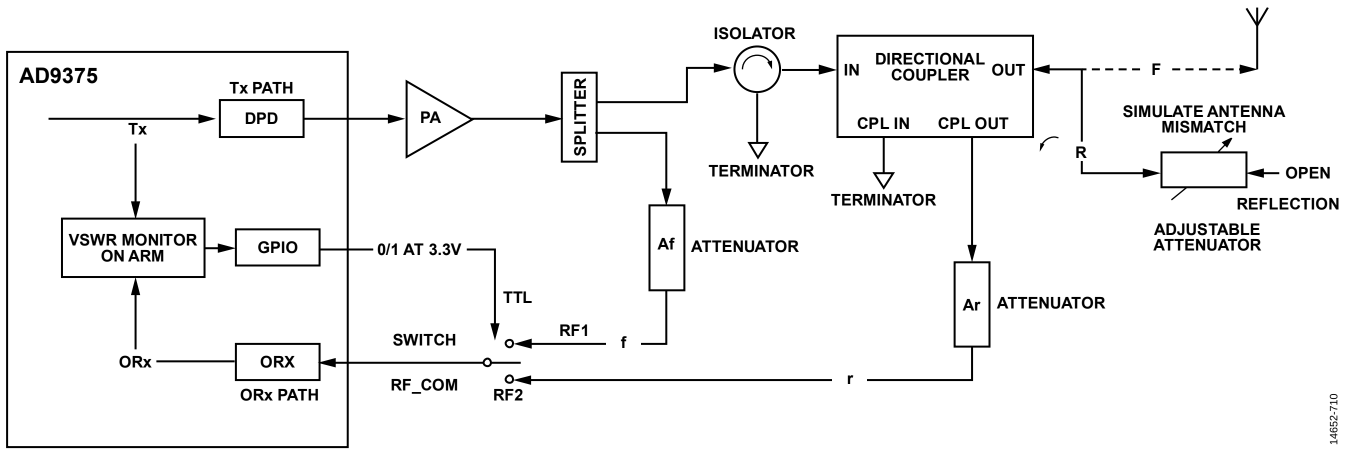
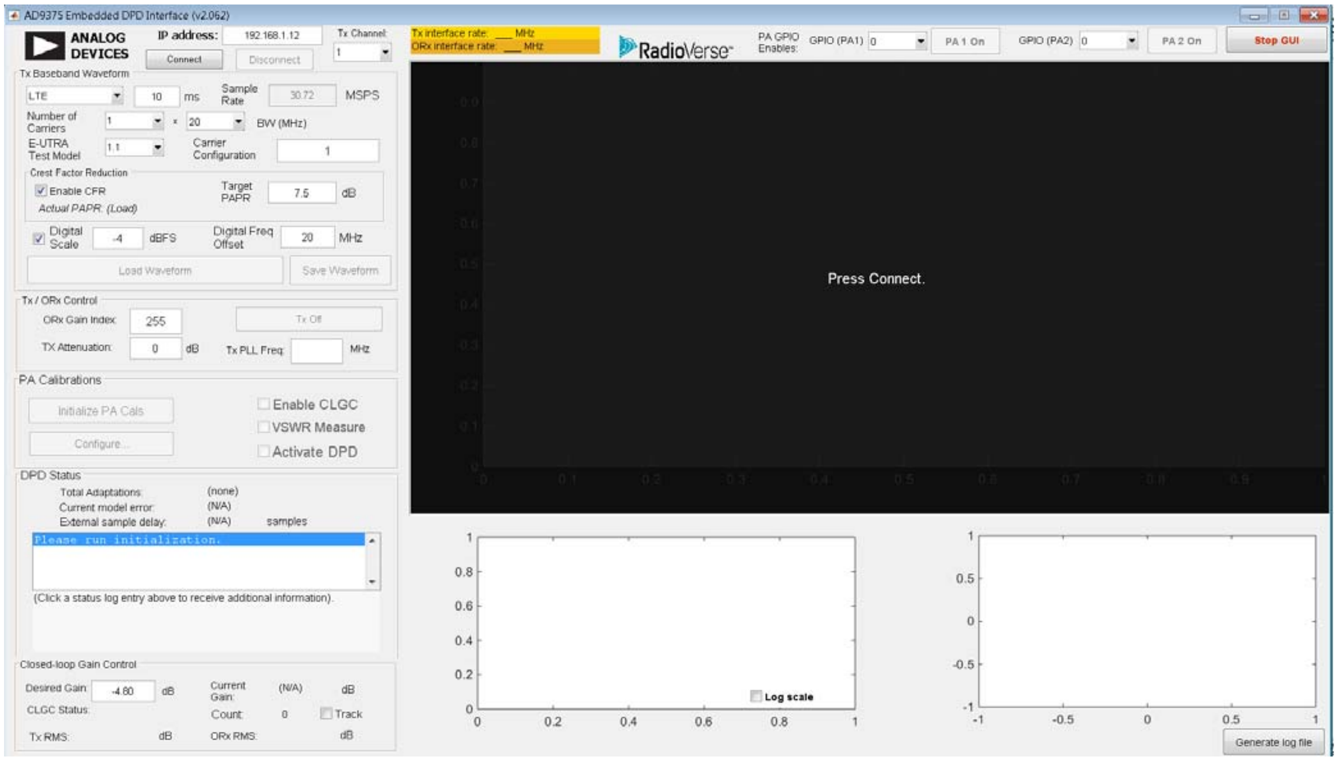


Figure 235. VSWR Measurement Setup Example

DPD GUI

The digital predistortion (DPD) graphical user interface (GUI) is the primary evaluation tool for the DPD, closed-loop gain control (CLGC), and voltage standing wave ratio (VSWR) features. Figure 236 shows the initial DPD GUI, and Figure 237 shows the GUI when it is connected to the command server. All DPD functionality can be controlled from this DPD GUI. It also incorporates waveform generation, CLGC, and VSWR control.

These features are described in subsequent sections of this user guide. In addition, the DPD, application programming interface (API), and delay-locked loop (DLL) may be used to interact and control the DPD via Python or C#. The transceiver evaluation software (TES) GUI supports an **IronPython** tab that may be used for scripting purposes.


AD9375 Embedded DPD Interface (v2.062)

IP address: 192.168.1.12
Tx Channel: 1
Tx interface rate: ___ MHz
ORx interface rate: ___ MHz
PA GPIO Enables: GPIO (PA1) 0 GPIO (PA2) 0

Tx Baseband Waveform
LTE 10 ms Sample Rate 30.72 MSPS
Number of Carriers 1 × 20 BW (MHz)
E-UTRA Test Model 1.1 Carrier Configuration 1
Crest Factor Reduction
☑ Enable CFR Target PAPR 7.5 dB
Actual PAPR: (Load)
☑ Digital Scale -4 dBFS Digital Freq Offset 20 MHz

Tx / ORx Control
ORx Gain Index: 255
TX Attenuation: 0 dB Tx PLL Freq: MHz

PA Calibrations
☐ Enable CLGC
☐ VSWR Measure
☐ Activate DPD

DPD Status
Total Adaptations: (none)
Current model error: (N/A)
External sample delay: (N/A) samples

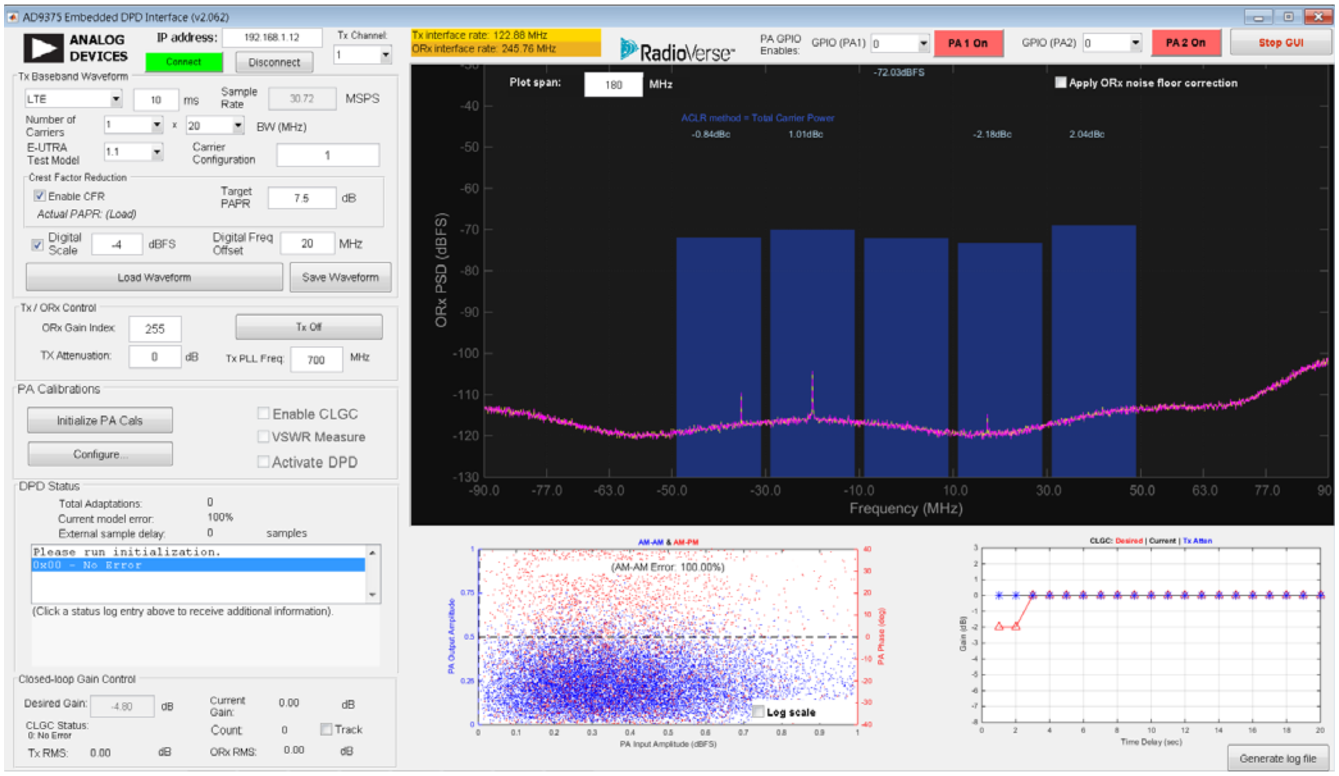
```
Please run initialization.
```

(Click a status log entry above to receive additional information).

Closed-loop Gain Control
Desired Gain: -4.80 dB Current Gain: (N/A) dB
CLGC Status: Count: 0 ☐ Track
Tx RMS: dB ORx RMS: dB

Press Connect.

Figure 236. Initial DPD GUI Interface—Not Connected to Command Server


AD9375 Embedded DPD Interface (v2.062)

IP address: 192.168.1.12 Connect Disconnect Tx Channel: 1

Tx interface rate: 122.88 MHz
ORx interface rate: 245.76 MHz

PA GPIO Enables: GPIO (PA1) 0 PA 1 On GPIO (PA2) 0 PA 2 On Stop GUI

Tx Baseband Waveform: LTE 10 ms Sample Rate 30.72 MSPS
Number of Carriers 1 x 20 BW (MHz)
E-UTRA Test Model 1.1 Carrier Configuration 1
Crest Factor Reduction: ☑ Enable CFR Target PAPR 7.5 dB
Actual PAPR: (Load)
☑ Digital Scale -4 dBFS Digital Freq Offset 20 MHz
Load Waveform Save Waveform

Tx / ORx Control: ORx Gain Index: 255 Tx Off
TX Attenuation: 0 dB Tx PLL Freq: 700 MHz

PA Calibrations: Initialize PA Cals Configure... ☐ Enable CLGC ☐ VSWR Measure ☐ Activate DPD

DPD Status
Total Adaptations: 0
Current model error: 100%
External sample delay: 0 samples

```
Please run initialization.
0x00 - No Error
```

(Click a status log entry above to receive additional information.)

Closed-loop Gain Control
Desired Gain: -4.80 dB Current Gain: 0.00 dB
CLGC Status: 0: No Error Count: 0 ☐ Track
Tx RMS: 0.00 dB ORx RMS: 0.00 dB

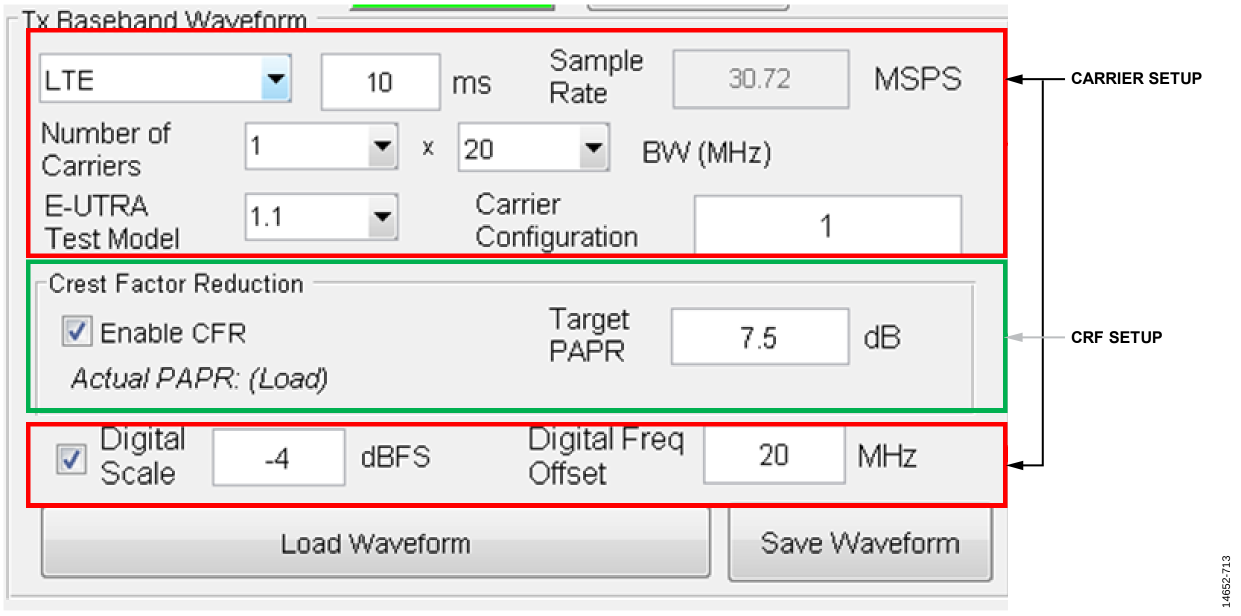
Plot span: 180 MHz ☐ Apply ORx noise floor correction

ACLR method = Total Carrier Power
-72.03dBFS
-0.84dBc 1.01dBc -2.18dBc 2.04dBc

Figure 237. Initial DPD GUI Interface—After Connecting to Command Server

WAVEFORM SETUP

The baseband waveform characteristics can be manipulated using the **Tx Baseband Waveform** section of the GUI (see Figure 238).


Tx Baseband Waveform

LTE 10 ms Sample Rate 30.72 MSPS
Number of Carriers 1 x 20 BW (MHz)
E-UTRA Test Model 1.1 Carrier Configuration 1

Crest Factor Reduction
☑ Enable CFR Target PAPR 7.5 dB
Actual PAPR: (Load)

☑ Digital Scale -4 dBFS Digital Freq Offset 20 MHz

Load Waveform Save Waveform

Figure 238. Waveform Characteristics

Carrier Setup

The modulation is based on the wireless standard chosen from the dropdown list box (**LTE** or **Custom**). The suggested modes for adjacent channel leakage ratio (ACLR) conformance testing according to 3GPP Technical Specification (TS) 36.141 are E-UTRA test model (E-TM) 1.1 (maximum power) and E-TM 1.2 (boosting). However, more dynamic waveforms such as E-TM 2.0 can also be selected (although not intended for ACLR compliance tests). The maximum fully occupied bandwidth that can be selected is 20 MHz. However, a number of smaller waveforms may be spliced together, resulting in an aggregated configuration. For example, a 4 × 5 MHz LTE waveform can be generated and transmitted in a 1011 configuration to give 15 MHz of signal bandwidth in 20 MHz of spectrum. A baseband frequency offset may also be specified to position the waveforms at a non-zero (non dc) offset from the RF local oscillator (LO). Digital back off is applied by means of the digital scale input. The digital predistortion (DPD) requires digital expansion headroom to operate; therefore, it is recommended to apply at least −3 dBFS digital back off to the signal. Note that the DPD algorithm can handle up to 40 MHz of fully occupied bandwidth while still exceeding the ACLR specification for some power amplifiers (PAs). However, this result is contingent upon the bandwidth performance of the PA because memory effects increase with increasing signal bandwidth. The user must verify PA performance against the report published by the PA vendor. For wider bandwidths, choose an appropriate Tx DPD profile and a PA that can support video bandwidths exceeding what is required for the signal. PA gain flatness may also play a role in the achievable ACLR performance.

Custom Waveforms

Click within the dropdown menu that currently displays **LTE** in Figure 238 for access to this option. Only LTE frequency division duplex (FDD) downlink (DL) waveforms are included in the DPD GUI library. For other technologies and special configurations, use the **Custom** option and load the waveform using a complex I/Q tabbed text file (no headers). Enter the sample rate and ensure that the waveform time length is as expected. Choose the scaling and the crest factor reduction (CFR) as desired, and load the waveform while leaving all other carrier setup controls as don't care. If the user does not want the DPD GUI to replicate the baseband waveform, set the **Number of Carriers** to 1× mode. Choosing any other value creates copies of the waveform around dc (LO).

CFR Setup

A CFR algorithm is provided along with the GUI for evaluation purposes. When a target peak to average power ratio (PAPR) is specified, the CFR algorithm clips and shapes the baseband waveform to prevent high PAPR on the Tx output. CFR is a typical prerequisite in most DPD setups to prevent the PA from going into deep saturation and to achieve higher power added efficiency (PAE). To validate the CFR, view the complementary cumulative distribution function (CCDF) of the Tx signal on a spectrum analyzer. When setting the PA bias voltages (in the reference setup, the SKY66297-11 is biased at 5.1 V), account for the crest factor of the test waveform. A peak to average ratio (PAR) of 7.5 dB to 8.5 dB is typical for a 20 MHz LTE FDD waveform post CFR. Note that the CFR operations on the test waveforms are precomputed in the GUI software. A CFR block is not integrated into the AD9375 transceiver. Tweaking the PAR value while evaluating the AD9375 DPD for error vector magnitude (EVM) tests is recommended because the optimal point between choosing a high enough PAR (to achieve better EVM performance) while maintaining desired levels of ACLR margin and power output at the PA is user and use case dependent. Also note that, a 10 ms waveform must be used for full frame LTE EVM tests. While gated EVM measurements may be possible and are valid, these measurements may not give the user the same insight into PA and DPD behavior as a full frame test. Synchronization between the REF_CLK source and the spectrum analyzer is recommended to minimize phase inaccuracies that affect EVM. Refer to the EVM Tests section for additional details.

DPD SETUP

When a waveform is loaded, power on the power amplifier (PA) and enable the digital predistortion (DPD). When the PA is powered on, the long-term evolution (LTE) waveform received on the ORx input is visible on the graph as a red trace.

To enable the DPD, take the following steps:

1. Click **Initialize PA Cals** to reset the DPD actuator and run the DPD initialization calibration. Initialization is required for proper time alignment of DPD samples (external delay measurement). To reiterate, DPD does not function unless this initialization step is completed. When run, the AD9375 **Embedded DPD Interface**, **PA Calibrations** section, changes as shown in Figure 240.
2. Check the **Activate DPD** box for the DPD to begin the adaptation process (see Figure 236).

The pink trace in Figure 239 shows the power spectral density (PSD) of the transmitter output with no DPD adaptation. The yellow trace is the PA ouput PSD that is received by the ORx channel. There should be a noticeable improvement in the PA output adjacent channel level rejection (ACLR) from the no DPD case (pink trace) and the DPD on case (yellow trace). If this not the case, or an error shows in the **DPD Status** section, proceed to the Error Messages and Debug Information section.

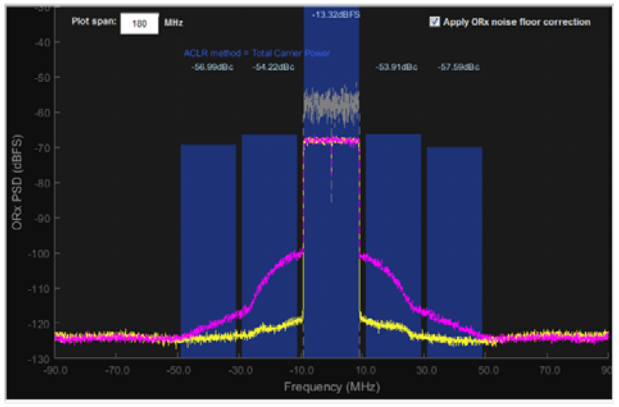


Figure 239. DPD GUI FFT Plot

Click **Reset PA Cal** to reset the DPD at any time. This operation performs a full reset of the DPD actuator and reinitializes the DPD, ensuring proper time alignment for the DPD samples. Once the DPD resets, check **DPD Adaptation** to reenabled it.

ORx Noise Floor Correction

The ORx noise floor correction is a one time GUI calibration where the noise floor on the ORx path is measured and then correction is applied to reduce the effects of the ORx noise floor on the received signal (see Figure 239). Note that this correction runs in the GUI and should not be confused with device calibration. Enabling this correction decreases the noise floor of the displayed ORx signal and generally improves the accuracy of the adjacent channel leakage ratio (ACLR) displayed by the GUI. This setting has no effect on the actual Tx output (observed on, say, a spectrum analyzer) apart from a momentary loss of transmission while the actual calibration measurement is performed.

PA1/PA2 On/Off Control

Refer to the two GPIO, GPIO (PA 1) and GPIO (PA 2), buttons shown at the top of Figure 237. These buttons control the GPIO pins, which in turn can control PA 1 and PA 2, respectively. Use the dropdown menus attached to the GPIO (PA 1) and GPIO (PA 2) buttons to assign the GPIO_3P3 pins. Red indicates that the pin is currently disabled (0), and green indicates that the pin is toggled on (1). The text within the dropdown menu displays the status that the pin toggles to when clicked. Note that this is an optional feature, and the exact voltage reference or PA enabled signal must be derived from the 3.3 V GPIO pin.

Tx Channel Control

The **Tx Channel** dropdown menu controls the channel currently displayed within the ACLR graph and also controls the individual settings (see Figure 236). Enabling or disabling the DPD and/or the closed-loop gain control (CLGC) can be done separately for both channels. Due to the multiplexed nature of the ORx, the **ORx Gain Index** is also common for both channels.

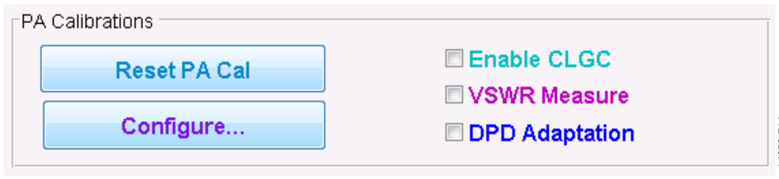


Figure 240. DPD Setup Tab

Tx/ORx Control

The **Tx / ORx Control** section controls some of the parameters for the Tx channel that were selected using the **Tx Channel** dropdown menu (see Figure 236).

ORx Gain Index

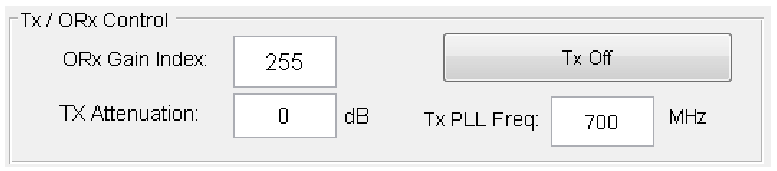
The **ORx Gain Index** box within the **Tx / ORx Control** section can vary from 238 to 255. This value controls the ORx gain (for both TX channels) and starting at a low value during initialization is recommended. Increase the gain index to a value where the **AM-AM** plot does not have too many outliers (AM-AM appears thin and the samples are tight around the linear plot), which implies less noise on the ORx. A good gain index achieves a −12 dBFS value for a single 20 MHz LTE carrier with 7.5 dB peak to average ratio (PAR), which allows a margin of 4.5 dB for the occurrence of statistically rare samples exceeding the 7.5 dB peak value while not saturating the ORx, resulting in linear operation. A clean, linear ORx is key to a successful DPD operation. With the default ORx gain table, the ORx gain changes n dB for an n value change in index; for example, 3 dB for a change from 238 to 241.

Tx Attenuation

The **Tx Attenuation** box within the **Tx / ORx Control** section controls the attenuation for the current channel.

Tx PLL Frequency

The **Tx PLL Freq:** box within the **Tx / ORx Control** section controls the frequency of operation of the current Tx channel. Modifying this value while radio calibrations are running can lead to poor performance with other tracking calibrations. Reprogramming the device (or at least rerunning initialization calibrations at the new frequency of operation) is recommended in such cases. For reprogramming the device or rerunning the initialization calibrations, disconnect the digital predistortion (DPD) GUI and use the TES software or other methods (for example, Python scripts) to perform the necessary operations. Note that the DPD performance degrades with large frequency steps. For such cases, rerun the DPD initialization calibrations within the **PA Calibrations** section and then check off the **Activate DPD** box (see Figure 236).


Tx / ORx Control

ORx Gain Index: 255

TX Attenuation: 0 dB

Tx PLL Freq: 700 MHz

*Figure 241. **Tx/ORx Control** Section*

PA CALIBRATION CONFIGURATION

Calibration configuration is controlled using the window shown in Figure 242. The subsequent sections explain each setting. See the Systems Design Considerations section for details on digital predistortion (DPD) tuning procedure using some of these settings.

General

Adjust the **Delay Offset** controls in decrements or increments of 1/16 of a sample, where the fractional part is the x/16 (see the dropdown menu), and the integer delay values can be adjusted using the integer dropdown menu (see Figure 242). For example, chose –, 1, and 1/16 in the respective dropdown menus for a −1 value with a 1/16 sample delay. Changes in the delay offset values requires rerunning the DPD initialization calibration. Refer to the additionalDelayOffset parameter in Table 193 for a detailed description.

The linear feedback shift register (LFSR) level and iterations control the power level of the wideband signal that is sent during the DPD initialization calibration. **PN-Seq Level** is a fixed value for information only (see Figure 242). Refer to the pathDelayPnSeqLevel parameter in Table 193 for a detailed description.

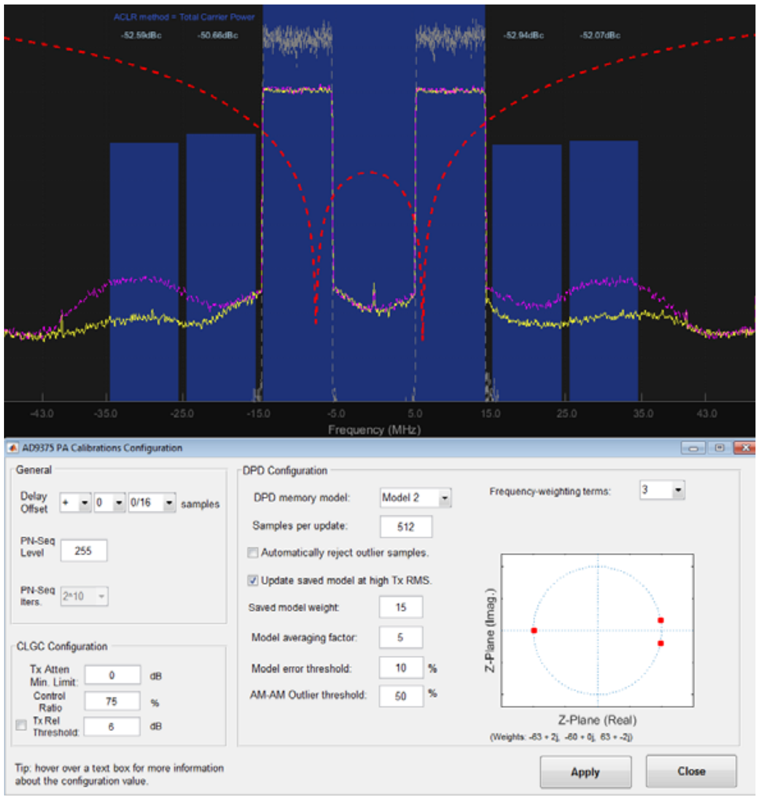
DPD Configuration

The **DPD Configuration** section of the **AD9375 PA Calibrations Configuration** window allows the user to manipulate certain key parameters of the DPD engine (see Figure 242). For more detailed instructions on tuning DPD using these configuration parameters, see the Systems Design Considerations section.

The **DPD memory model** dropdown menu allows the selection of between four different polynomial models to model the PA with, based on various implementations of generalized memory polynomials (see Figure 242). See the DPD Actuator Model Configurations section for more details. Refer to the modelVersion parameter in Table 193 for a detailed description.

The **Samples per update** text box controls the number of input samples required to complete a DPD adaptation (see Figure 242). Refer to the samples parameter in Table 193 for a detailed description.

The **Automatically reject outlier samples** box is deprecated and its use is not supported (see Figure 242). Refer to the robustModeling parameter in Table 193 for more details.



AD9375 PA Calibrations Configuration

General

Delay Offset: + 0 0/16 samples

PN-Seq Level: 255

PN-Seq Iters.: 2^10

CLGC Configuration

Tx Atten Min. Limit: 0 dB

Control Ratio: 75 %

Tx Rel Threshold: 6 dB

DPD Configuration

DPD memory model: Model 2

Samples per update: 512

Automatically reject outlier samples.

[✓] Update saved model at high Tx RMS.

Saved model weight: 15

Model averaging factor: 5

Model error threshold: 10 %

AM-AM Outlier threshold: 50 %

Frequency-weighting terms: 3

(Weights: -63 + 2j, -60 + 0j, 63 + -2j)

Tip: hover over a text box for more information about the configuration value.

Figure 242. ***AD9375*** ***PA Calibrations Configuration*** *Window*

Enable the **Update saved model at high Tx RMS** box to allow the DPD to update a separate high power power amplifier (PA) model that remembers high amplitude PA characteristics and improves adjacent channel leakage ratio (ACLR) in dynamic power conditions (see Figure 242). Refer to the highPowerModelUpdate parameter in Table 193 for a detailed description.

The **Saved model weight** text box controls how much weight or influence the high power startup power amplifier (PA) model is given when computing the final applied predistortion function (see Figure 242). Refer to the modelPriorWeight parameter in Table 193 for a detailed description.

The **update saved model at high Tx RMS** parameter allows enabling or disabling of the DPD prior model update.

The **Model averaging factor** text box controls how much weight the previous DPD correlations can exert on the current adaptation. Refer to the damping parameter in Table 193 for a detailed description.

Increase the value in the **Model error threshold** text box if additional model errors must be allowed in a DPD adaptation before error code 0x09 occurs. This value is not configurable via the application programming interface (API).

Adjust the value in the **AM-AM Outlier threshold** text box to control the occurrence of the AM_AM_OUTLIERS error (0x0A). Generally, this threshold is only violated when the PA is deep into compression and heavily saturated. Refer to the outlierThreshold parameter in Table 193 for a detailed description.

The DPD algorithm works by minimizing the Tx to ORx sample error. However, due to PA or external loop spectral asymmetries, the adjacent channel leakage ratio (ACLR) performance may be worse on certain sides of the desired signal. The AD9375 DPD includes a 4-tap finite impulse response (FIR) filter that can help shape the error such that the DPD focuses more on the error on the poorer performing side of the spectrum. This **frequency-weighting** is achieved by adding zeros and creating an FIR filter that is overlaid on the ACLR spectrum of the DPD GUI as a red curve. By default, one zero is placed at (**64 + 0j**) on the complex Z plane to help minimize the Δ-Σ shaped noise at the ORx band edges. Additional zeros can be placed to create different filter shapes depending on how the user wants to direct the attention of the DPD. Deeper notches instruct the DPD that it must focus less on the errors in the notched out part of the spectrum, while the response shapes outside of the desired signal bands make the DPD focus more on those parts of the spectrum.

For example, see Figure 242 and note the multicarrier 3 × 5 MHz 101 configuration. The locations of the zeros indicate that the DPD must focus more on the ACLR side of the spectrum rather than the error in the carriers themselves. The improvement for tougher cases can be as good as 2 dB to 3 dB in ACLR performance. Note, however, that some error vector magnitude (EVM) degradation may occur due to the lack of focus of the DPD on the carrier themselves. As always, tuning the DPD is a systems choice and must be done with due consideration to the permissible limits of performance tolerance for all systems metrics, 3GPP mandated or otherwise. Refer to the numWeights and weights[3] parameters in Table 193 for a detailed description.

ERROR MESSAGES AND DEBUG INFORMATION

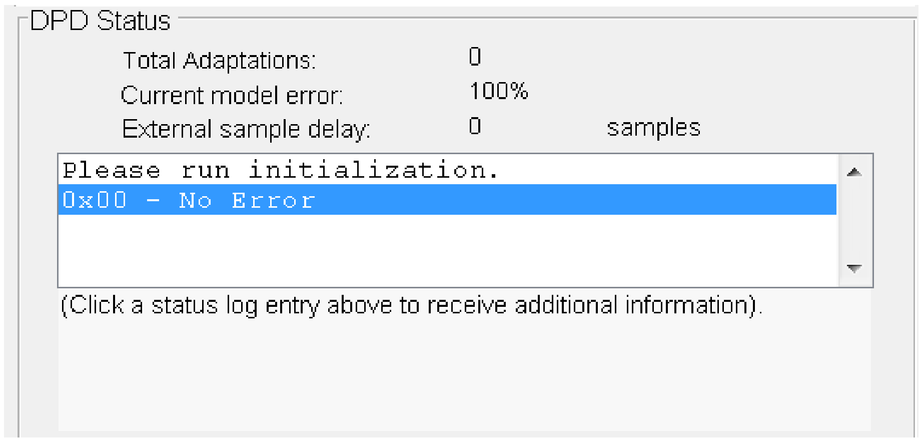
The DPD GUI has the capability to display the operating status of the DPD adaptation. If an error occurs during an adaptation, the error status is displayed in the **DPD Status** window. Click the error status to open the debug message. The **DPD Status** window displays the total number of DPD adaptations that have occurred along with the current model error.

Total Adaptations (see Figure 243) shows the number of successful DPD adaptations that have occurred since the last DPD initialization calibration (**Reset PA Cal**). If an error occurs during a DPD iteration, the total number of adaptations does not count up. For an erroneous adaptation, an error status appears and the DPD adaptation is not applied to the actuator.

It is important to note that the last known good model is always stored in the DPD actuator and applied to the Tx samples until the DPD hardware is reset using a DPD initialization calibration calibration, even if DPD is disabled (that is, the tracking calibration mask does not contain a DPD calibration).

Current model error (see Figure 243) is a measure of how close the current PA model is to the actual PA. This error is calculated before the DPD applies the newly learned coefficients to predict the PA output from the current input samples for a short instance of time to calculate this error. A threshold for this model error can be set in the **DPD Configuration** window. If the error exceeds this threshold, DPD adaptation does not occur, and the DPD returns an ERROR 9 (MODEL_ERROR_TOO_HIGH).

For more information on the DPD, closed-loop gain control (CLGC), and voltage standing wave ratio (VSWR) status codes, respectively, and troubleshooting options, see Table 194, Table 196, and Table 198.



Total Adaptations: 0
Current model error: 100%
External sample delay: 0 samples

```
Please run initialization.
0x00 - No Error
```

(Click a status log entry above to receive additional information).

*Figure 243. **DPD Status** Window and Status Codes*

DPD API

This section describes all the application programming interface (API) data structures and functions that are associated with the digital predistortion (DPD) feature.

ARM SETUP COMMANDS

A few ARM setup commands are introduced in the following subsection that are ancillary to DPD, closed-loop gain control (CLGC), and voltage standing wave ratio (VSWR) operation.

The following API functions enable the use of the DPD, CLGC, and VSWR calibrations. These functions must be called in order to successfully execute these calibrations.

MYKONOS_runInitCals (…)

```
mykonosErr_t
MYKONOS_runInitCals(mykonosDevice_t
*device, uint32_t calMask)
```

Description

This function performs the ARM initialization calibrations (init cals) that are prerequisites for the calibrations present in the enableMask enabled by MYKONOS_enableTrackingCals(…). This command must be called when the device is in the radioOff state.

Parameter

- *device: This is a structure pointer for the device data structure.
- calMask: Initialization calibration mask for initializing the different calibrations.

Table 190. Initial Calibrations Bit Mask

calMask	Bit description
0	Tx baseband filter
1	ADC runner
2	transimpedance amplifier (TIA) 3 dB corner
3	DC offset
4	Tx attenuation delay
5	Rx gain delay
6	Flash calibration
7	Path delay
8	Tx local oscillator (LO) leakage internal
9	Tx LO leakage external
10	Tx quadrature error correction (QEC) initialization
11	Loopback Rx LO delay
12	Loopback Rx QEC initialization
13	Rx LO delay
14	Rx QEC initialization
15	DPD initialization
16	CLGC initialization
17	VSWR initialization

It is important to note the following:

- Running the external initialization calibrations (external LO leakage (LOL), DPD, CLGC, and VSWR initialization calibrations) in a single execution of the MYKONOS_ runInitCals(…) results in running the initialization calibrations in the following order: VSWR, LOL, DPD, and CLGC.
- It is recommended that the user always perform DPD, CLGC, and VSWR initialization calibrations at the Tx attenuation value that corresponds to the final rated power amplifier (PA) operating power conditions. Following this recommendation improves the path delay estimation of the calibrations in general and reduces variability from run to run because the signal noise ratio (SNR) at ORx is better when there is more of the pseudonoise (PN) sequence to correlate with. A small deviation in the estimated path delay can cause a DPD performance degradation of up to 3 dB.
- When using an older ARM version or when running initialization calibrations differently than is suggested in the first bullet, it is important to remember that VSWR initialization calibration sets up the path observed by the ORx switch (forward or reflected), and therefore, must be executed before the other external initialization calibrations, such as the LOL, DPD, and CLGC calibrations, which are calibrations that require knowledge of the external loopback path delay. For example, if each of the external initialization calibrations are run separately, the following sequence is recommended: VSWR, LOL, DPD, and CLGC initialization calibrations. It is important to not override the VSWR switch control by writing to the GPIO pin that is used by the VSWR calibration.

MYKONOS_enableTrackingCals(…)

```
mykonosErr_t
MYKONOS_enableTrackingCals(mykonosDevice_t
*device, uint32_t enableMask)
```

Description

This function sets which ARM tracking calibrations are enabled during the radioOn state. The command must be called during the radioOff state. Any of the closed-loop gain control (CLGC), and voltage standing wave ratio (VSWR), and digital predistortion (DPD) tracking calibrations may be enabled using this function by selecting the appropriate value for the tracking calibration mask.

Parameters

- *device: This is a structure pointer for the device data structure.
- enableMask: This is the tracking calibration mask for enabling the different tracking calibrations. Options are shown in Table 191.

Table 191. Tracking Calibration Bit Mask

enableMask	Bit Description
0	TRACK_RX1_QEC
1	TRACK_RX2_QEC
2	TRACK_ORX1_QEC
3	TRACK_ORX2_QEC
4	TRACK_TX1_LOL
5	TRACK_TX2_LOL
6	TRACK_TX1_QEC
7	TRACK_TX2_QEC
8	TRACK_TX1_DPD
9	TRACK_TX2_DPD
10	TRACK_TX1_CLGC
11	TRACK_TX2_CLGC
12	TRACK_TX1_VSWR
13	TRACK_TX2_VSWR

- To ensure that the tracking calibrations runs correctly, enforce the following sequence:

1. Run initial calibrations with the xxx_INIT calibration bits enabled in the calmask during the initialization process. The application programming interface (API) is MYKONOS_runInitCals(…).
2. Set tracking calibration mask to include the TRACK_TX1_DPD, TRACK_TX2_DPD, TRACK_TX1_CLGC, TRACK_TX2_CLGC, TRACK_TX1_VSWR, and/or TRACK_TX2_VSWR mask bits, depending on the desired calibrations for each channel.
3. Note that the TRACK_ORX1_QEC and the TRACK_ORX2_QEC mask bits must be set in order to have successful DPD, CLGC, and VSWR tracking.
4. The API is MYKONOS_enableTrackingCals(…).

The host must set the ORx path source to OBS_INTERNALCALS by calling MYKONOS_setObsRxPathSource(…). Setting the ORx path source to OBS_INTERNALCALS enables scheduling of regular tracking radio calibrations, such as Tx QEC and Tx local oscillator leakage (LOL) tracking along with DPD tracking in either frequency division duplex (FDD) or time division duplex (TDD), ARM command or pin mode. See the Observation Receiver (ORx) section for ORx channel setup details. Performing the previously mentioned steps enables the ARM to access ORx data and initiates the DPD, CLGC, and VSWR tracking and measurement processes.

MYKONOS_setAllTrackCalState(…)

```
mykonosErr_t
MYKONOS_setAllTrackCalState(mykonosDevice_
t *device, uint32_t trackCals)
```

Description

This function sets which ARM tracking calibrations are allowed to track during the radioOn state (also known as the suspend/resume or pause/resume feature). The command can be called in either the radioOff or radioOn state, with the primary intent being that this function be used by the user in the radioOn state to quickly suspend or resume calibrations. Note that the state of the calibration (paused: no updates scheduled by ARM, or resumed: normal expected behavior) is sticky between the radioOff and the radioOn states when in a tracking state (and even if the respective initialization calibrations are performed). It is the responsibility of the host to control the state of the calibrations. By controlling the trackCals bit mask, calibrations can be suspended (bit value of 0) or active/resumed (bit value of 1).

Parameters

- *device: This is a structure pointer for the device data structure whose calibrations are to be suspended or resumed.
- trackCals: The value specified by this argument is some subset of the currently enabled tracking calibration mask bits that controls which calibrations are to suspended or resumed. An error code is returned if the user attempts to resume a calibration that is not part of the active tracking calibration bit mask defined by enableMask in MYKONOS_enableTrackingCals(…). For the allowed bit mask values, refer to Table 191.

MYKONOS_getAllTrackCalState(…)

```
mykonosErr_t
MYKONOS_getAllTrackCalState(mykonosDevice_
t *device, uint32_t *trackCals)
```

Description

This function reads back the ARM tracking calibrations that are allowed to track during the radioOn state (also known as the suspend/resume or pause/resume feature). The command can be called in either the radioOff or the radioOn state, with the primary intent being that this function be used by the user in the radioOn state. Note that the state of the calibration (paused: no updates scheduled by ARM, or active/resumed: normal expected behavior) is sticky between the radioOff and the radioOn states when in a tracking state (and even if the respective initialization calibrations are performed). It is the responsibility of the host to control the state of the calibrations.

Parameters

- *device: This is a structure pointer for the device data structure whose calibrations are to suspended or resumed.
- *trackCals: The value returned in this pointer shows currently enabled tracking calibration mask bits that represent which calibrations have been suspended (bit value of 0) or active/resumed (bit value of 1). The active tracking calibration bit mask defined by enableMask in MYKONOS_enableTrackingCals(…) can be used to determine which bit mask values shown in Table 191 are available.

DPD API DATA STRUCTURES

The data structures associated with the digital predistortion (DPD) functionality follow.

int8_cpx

The int8_cpx structure is used within the mykonosDpdConfig_t structure to hold an int8_t complex number that is used in the weights member described in Table 193.

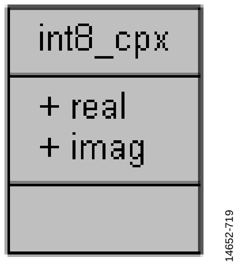


Figure 244. int8_cpx Structure

```
typedef struct
{
  int8_t real;
  int8_t imag;
} int8_cpx;
```

Table 192. int8_cpx Structure Member Description

Structure Member	Valid Values	Description
real	−128…+127	The real part of the complex number used in the weights member of the mykonosDpdConfig_t structure.
imag	−128…+127	The imaginary part of the complex number used in the weights member of the mykonosDpdConfig_t structure.

mykonosDpdConfig_t

mykonosDpdConfig_t is the main data structure that stores all of the parameters relevant to the DPD configuration. This information is loaded into the ARM memory using the MYKONOS_configDpd() function before running the DPD initialization or tracking calibrations. The unified modeling language (UML) representation of the structure is shown in Figure 245.

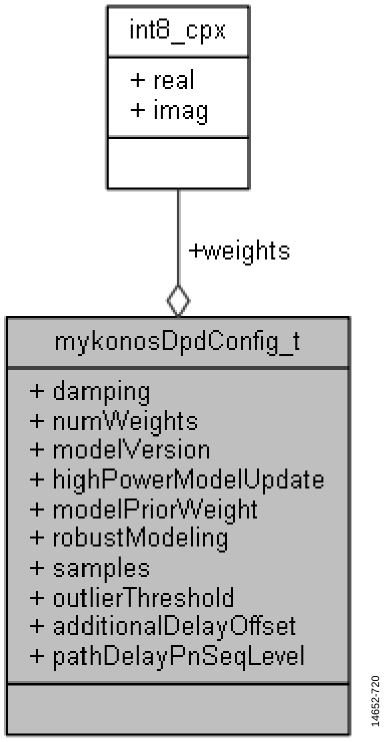


Figure 245. mykonosDpdConfig_t UML Representation

```
typedef struct
{
    uint8_t damping;
    uint8_t numWeights;
    uint8_t modelVersion;
    uint8_t highPowerModelUpdate;
    uint8_t modelPriorWeight;
    uint8_t robustModeling;
    uint16_t samples;
    uint16_t outlierThreshold;
    int16_t  additionalDelayOffset;
    uint16_t pathDelayPnSeqLevel;
    int8_cpx weights[3];
} mykonosDpdConfig_t;
```

Table 193. AD9375 Digital Predistortion (DPD) Configuration Structure Member Description

Structure Member	Valid Values	Description
damping	0…15	1/($2^{DAMPING + 8}$) is a weighting factor multiplied to past DPD correlations when computing a new DPD correlation matrix for every DPD iteration. The default value of this factor is 5, resulting in a 1/8192 value. This parameter may be viewed as a leakage factor to avoid overtraining the model on a new set of samples while retaining some memory or history of the previous correlations (this is because the past correlations are accumulated in a leaky fashion). A 0 value results in infinite damping (that is, past correlation values are accumulated forever with no leakage). It is recommended that this parameter be adjusted in conjunction with the samples parameter. Note that DPD performance can vary quite a bit for dynamic signals when adjusting this parameter.
numWeights	0…3	Number of weights to use in the weights[3] member of this structure for frequency weighting of the DPD error.
modelVersion	0…3	DPD model version: one of four different polynomial models. The choice of the optimum model usually depends on the power amplifier (PA) and operating bandwidth.
highPowerModelUpdate	0, 1	A 1 value for this member results in the update of a separate high power or prior PA model that is used to remember the high amplitude PA characteristics and to improve the adjacent channel leakage ratio (ACLR) in dynamic power conditions. Updates to the high power model are made when the Tx rms power is within 1 dB of the historical peak Tx rms. During periods of lower Tx power, the DPD algorithm includes the stored higher power model for computing the PA predistorter model if the DPD has ever tracked successfully while this parameter was high. During low to high power transitions, DPD already has adaptation coefficients for the Tx higher powered samples improving dynamic performance. Setting this parameter to 0 disables updates to the prior model and uses the initial loaded model instead (if one was provided at startup). Default: 1.
modelPriorWeight	0…32	weight = $2^{modelPriorWeight + 10}$. This parameter controls how much weight or influence the high power/startup PA model is given when computing the final applied predistortion function. A larger value causes adaptation to move less from the high power/startup PA model, limiting low power ACLR improvement but improving high power ACLR during fast low to high power transitions. Default: 20.
robustModeling	0, 1	Note that this feature has been marked for deprecation and use of this feature is discouraged.
samples	64…32768	Number of Tx and ORx I/Q samples to capture. Default is 512. The DPD algorithm computes the model coefficients by sparsely sampling the Tx and ORx data, and this parameter controls the number of capture blocks required per update. Reducing the number of samples per update results in faster processing at the risk of possible reduced performance (depending on input waveform and PA conditions).
outlierThreshold	1…8192	This is the threshold for samples to be discarded if they are outside of the 1:1 line on the **AM-AM** plot. Default: 4096 (50% of 8192). Adjust this input to control the occurrence of the AM_AM_OUTLIERS (0x0A) DPD status message/error from the ARM. Generally, this threshold is exceeded only when the PA is deep into compression and heavily saturated. Too many outliers (higher than percentage of samples set using this parameter) cause the current model adaptation to be discarded. Increase the threshold to force the DPD to adapt to a more nonlinear PA state (performance varies according to PA).
additionalDelayOffset	−64…+64	This parameter adds or reduces delay to the external sample delay assessed during the sample time alignment process. Resolution of additional delay is 1/16 of an ORx sample. Delay Offset = additionalDelayOffset/16. Using 16 for this parameter implies a 1-sample delay. Default: 0. By default, the DPD algorithm in the AD9375 begins the PA model from the peak in the cross correlation between the PA input and output samples (using the sample time alignment delay determined during the DPD initialization calibration). However, for modeling some PAs, it may be beneficial to offset the model slightly from the correlation peak by a fixed amount (usually a fraction of a sample). In many PAs, the peak in the impulse response is not the lead (zero lag) coefficient. Time alignment offset allows the learning model impulse terms before this peak and can improve modeling accuracy and DPD performance. Changes in the delay offset values require rerunning the DPD initialization calibration.

Structure Member	Valid Values	Description
pathDelayPnSeqLevel	1…8192	Amplitude level of broadband pseudonoise (PN) sequence sent out during DPD initialization calibration. Default: 255 (−30 dBFS = 20 $\log_{10}$(value/8192)). Analog Devices recommends using the default value for this parameter unless the PA is highly sensitive to broadband radio frequency (RF) input tones. Note that the user must always perform the DPD, CLGC, and VSWR initialization calibrations at the Tx attenuation value that corresponds to the final rated PA operating power conditions, which improves the estimation of the path delay and reduces variability from run to run.
weights[3]	−128…+127	DPD model error frequency weighting. The DPD algorithm works by minimizing the Tx to ORx sample error. The AD9375 R1 DPD includes a 4-tap FIR filter that can help shape the error such that DPD focuses more on the error at certain frequencies. Each element of the weights member holds a Z-plane zero location that will shape the filter response of the FIR to influence the frequencies at which DPD is most focused. By default, one zero is placed at (64 + 0j) on the complex-Z plane to help minimize the Δ-Σ shaped noise at the ORx band edges. Refer to the PA Calibration Configuration section for more information on this parameter.

mykonosDpdStatus_t

The mykonosDpdStatus_t structure contains four members that provide information on the status of the digital predistortion (DPD) calibration. This structure can be used as an error checker and debug tool for the DPD.

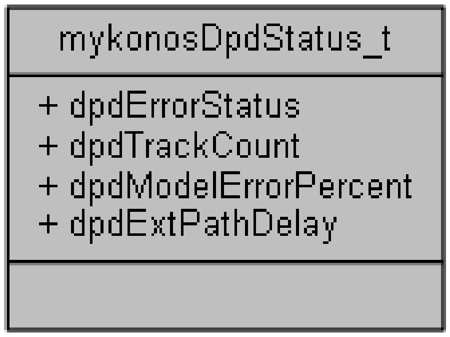


Figure 246. mykonosDpdStatus_t Structure

```
typedef struct
{
    uint32_t dpdErrorStatus;
    uint32_t dpdTrackCount;
    uint32_t dpdModelErrorPercent;
    uint32_t dpdExtPathDelay;
} mykonosDpdStatus_t;
```

Table 194. AD9375 DPD Status Structure Member Description

Structure Member	Value	Description
dpdErrorStatus	0	No error: DPD operation is normal.
	1	ORx disabled: The observation receiver (ORx) for this channel is currently disabled.
	2	Tx disabled: The transmitter for this channel is currently disabled.
	3	Path delay not setup: indicates that the process for sample alignment arrived at an invalid answer. This error is most commonly caused by a Tx or ORx that is not physically connected or a PA that is switched off. Ensure that everything is properly connected, and that the PA is turned on, and then rerun the DPD initialization.
	4	DPD initialization not run: must run DPD initialization calibration for the DPD to start adaptations.
	5	ORx signal too low: The ORx signal is lower than −28 dBFS. Ensure that the PA is on and all SMA cables are connected properly. Consider increasing the ORx gain index or removing the physical attenuation between the PA and ORx input.
	6	ORx signal saturated: The ORx measured too many samples above the configured saturation threshold. Consider reducing the ORx gain index or adding attenuation between the PA and the ORx input. The saturation threshold level is not configurable by application programming interface (API) and is defaulted to 87% of 0 dBFS. When this error occurs, the DPD zeroes out the correlators and starts over to prevent corrupting future adaptations.
	7	Tx signal too low: The Tx signal is off or has too little power for DPD adaptation to proceed. Consider increasing the digital Tx signal power. Threshold is −28 dBFS.
	8	Tx signal saturated: The post DPD Tx signal amplitude exceeds the configured saturation threshold. Decrease the Tx waveform digital peak value (scaling) to allow for sufficient amplitude expansion during predistortion (for example, −3 dBFS). When this error occurs, the DPD zeroes out the correlators, zeroes the prior model used, and starts over to prevent corruption of future adaptations.
	9	Model error high: The DPD model error is higher than 10%, and the calculated model was not applied to the DPD actuator. This condition can occur if the PA is highly nonlinear, or the signal chain is adjusted while DPD adaptation is still being performed. When this error occurs, the DPD zeroes out the correlators and starts over to prevent corruption of future adaptations.
	10	AM-AM outliers: Too many PA input to output samples were out of the configured linearity boundaries. Consider backing off the PA or adjusting the dpdOutlierThreshold in mykonosDpdConfig_t (see Table 192). When this error occurs, the DPD zeroes out the correlators and starts over to prevent corruption of future adaptations.

Structure Member	Value	Description
	11	Invalid Tx profile.
	12	ORx quadrature error correction (QEC) disabled: Set the calmask bits to enable ORx QEC1/QEC2 tracking calibrations because the DPD relies on the assumption that the ORx is a clean representation of the power amplifier (PA) output.
dpdTrackCount	0..0xFFFFFFFF	Number of times the DPD has successfully run since DPD initialization calibration.
dpdModelErrorPercent	0..1000	Percent error of PA model ×10 to include 1 decimal place.
dpdExtPathDelay	0..0xFFFFFFFF	External path delay from Tx output to ORx input, at 1/16 sample resolution of the ORx sample rate. Calculate true value by dividing this member.

DPD FUNCTIONALITY API FUNCTIONS

The application programming interface (API) functions associated with the digital predistortion (DPD) functionality follow. Refer to the Tracking Calibrations section for more information on setting up the calibration from an ARM system standpoint.

MYKONOS_configDpd (…)

```
mykonosErr_t
MYKONOS_configDpd(mykonosDevice_t *device)
```

Description

This function call configures the device with the parameters in the DPD data structure, mykonosDpdConfig_t. This function also performs some sanity checks to the data parameters (range and ARM state checks) in the mykonosDpdConfig_t data structure. This function can be called when the device is in either the radioOn or radioOff state. However, not all parameters in the mykonosDpdConfig_t structure can be changed in radioOn state. This function must be called in the radioOff state when attempting to change the additionalDelayOffset and pathDelayPnSeqLevel parameters. Changes to the latter parameters do not take effect unless modified in a radioOff state, and are ignored by the API and ARM.

MYKONOS_getDpdConfig(…)

```
mykonosErr_t
MYKONOS_getDpdConfig(mykonosDevice_t
*device)
```

Description

This function reads the DPD config structure from the ARM memory and updates the device → tx → dpdConfig data structure. There are no radio state dependencies for this function.

MYKONOS_getDPDStatus(…)

```
mykonosErr_t
MYKONOS_getDPDStatus(mykonosDevice_t
*device, mykonosTxChannels_t txChannel,
mykonosDpdStatus_t *dpdStatus)
```

Description

This function reads back the DPD calibration status from the ARM processor and returns the result to the mykonosDpdStatus_t structure.

Parameters

- *device: This is the device data structure pointer from which the DPD status is read back.
- txChannel: Input argument to set which Tx channel the function reads back the DPD status for. Valid enumeration values are Tx1 and Tx2 only.
- *dpdStatus: This is a pointer to the structure that contains the returned DPD status information.

MYKONOS_saveDpdModel(…)

```
mykonosErr_t
MYKONOS_saveDpdModel(mykonosDevice_t
*device, mykonosTxChannels_t txChannel,
uint8_t *modelDataBuffer, uint32_t
modelNumberBytes)
```

Description

This function reads a copy of the DPD prior model (or high Tx power model) from the ARM memory and saves it to the user memory specified by the modelDataBuffer pointer. The device must be in the radioOff state to call this function.

Parameters

- *device: This is the device data structure pointer from which the DPD model is saved.
- txChannel: specifies the Tx channels for which the DPD models are saved. Valid values are TX1, TX2, and TX1_TX2.
- The user must provide the correct buffer size with the modelNumberBytes argument. 172 bytes per channel are expected.

MYKONOS_restoreDpdModel (…)

```
mykonosErr_t
MYKONOS_restoreDpdModel(mykonosDevice_t
*device, mykonosTxChannels_t txChannel,
uint8_t *modelDataBuffer, uint32_t
modelNumberBytes)
```

Description

This function writes a copy of the user specific digital predistortion (DPD) model to the ARM memory, pointed to by the modelDataBuffer pointer, and instructs the ARM to install that DPD model into the ARM memory as a prior model. The device must be in the radioOff state to call this function. Note that resetting or reinitializing the device overwrites the restored DPD model data.

Parameters

- *device: This is the device data structure pointer on to which the DPD model is restored or loaded.
- txChannel: specifies the Tx channels for which the DPD models are loaded. Valid values are TX1, TX2, and TX1_TX2.
- The user must provide the correct buffer size with the modelNumberBytes argument. 172 bytes per channel are expected. The model data pointed to by the modelDataBuffer pointer must contain modelNumberBytes, resulting in an error otherwise.

MYKONOS_setDpdActState (…)

```
mykonosErr_t
MYKONOS_setDpdActState(mykonosDevice_t
*device, mykonosTxChannels_t txChannel,
uint8_t actState)
```

Description

This function enables the user to bypass the DPD actuator on a given Tx channel. This function can be called in the radioOn state. However, it is recommended that the user suspend (pause) the DPD calibration before using this function to synchronize all internal ARM scheduler tasks. The DPD calibration can be resumed when this function has been successfully executed.

Parameters

- *device: This is the device data structure pointer whose DPD actuator is being bypassed.
- txChannel: specifies the Tx channels for which the DPD actuators must be controlled. Valid values are TX1, TX2, and TX1_TX2.
- By setting the argument actState to 0, the DPD actuator for the specified txChannel is bypassed. A 1 for actState reenables the use of the DPD actuator for the desired channel.

MYKONOS_resetDpd (…)

```
mykonosErr_t
MYKONOS_resetDpd(mykonosDevice_t *device,
mykonosTxChannels_t txChannel, uint8_t
reset)
```

Description

This function enables the user to reset the DPD actuator for a given Tx channel. The user can call this function in the radioOn state. However, it is recommended that the user suspend (pause) the DPD calibration before using this function to synchronize all internal ARM scheduler tasks. The DPD calibration can be resumed when this function successfully executes.

Parameters

- *device: This is the device data structure pointer whose DPD actuator is being bypassed.
- txChannel: specifies the Tx channels for which the DPD actuators must be controlled. Valid values are TX1, TX2, and TX1_TX2.
- reset: A 1 resets the DPD actuator and the prior model. Calling this function with a reset value of 1 is equivalent to performing a DPD initialization calibration but without performing the external path delay measurement in radioOff mode that is typical of the DPD initialization calibration (no PN sequence is transmitted). A 0 is not a valid choice.

Note that a 2 for this argument is reserved for future use.

CLGC TRACKING CALIBRATION

To enable the closed-loop gain control (CLGC), click **Initialize/Reset PA Cals**. Next, check off the **CLGC** box to allow the CLGC to start measuring the gain of the Tx to ORx path and to update statuses. Note that tracking (applying control) does not begin until the selection of the **Track** box shown in Figure 248.

The value in the **Desired Gain** box is the desired loopback gain that is usually negative when the total loop attenuation is higher than the total loop gain. Refer to the tx1DesiredGain/ tx2DesiredGain parameters in Table 195 for a detailed description.

The value displayed in the **Current Gain** box is the current loopback gain. Refer to the currentGain parameter in Table 196 for a detailed description.

The **Track** check box allows the CLGC to control the loop gain by changing the Tx attenuation. When first enabling the CLGC using the DPD GUI (and not via the application programming interface (API)-/delay locked loop (DLL)-based scripts), the current gain is set as the desired gain (that is, the previously configured desired gain value is overwritten). Refer to the allowTx1AttenUpdates and allowTx2AttenUpdates parameters in Table 195 for a detailed description.

The **Tx RMS** and **ORx RMS** are a measure of the rms value of the digital power measured by the CLGC at the Tx or ORx measurement points. Refer to the txRms and orxRms parameters in Table 196 for a detailed description.

A description of the **Status** codes can be found in Table 196.

CLGC CONFIGURATION

See Figure 242 to see the following CLGC digital predistortion (DPD) GUI settings.

The **Tx Atten Min Limit** is the absolute minimum attenuation allowed during CLGC tracking. This box allows some level of PA protection from the CLGC gain going too high. Refer to the tx1AttenLimit and tx2AttenLimit parameters in Table 195 for a detailed description.

The **Control Ratio** field controls the rate at which the CLGC tracks the gain changes. Refer to the tx1ControlRatio and tx2ControlRatio parameters in Table 195 for a detailed description.

The **Tx Rel Threshold** feature lets the CLGC flag rapid changes from the target desired gain. Check the box next to this feature to enable the relative threshold. Refer to the tx1RelThreshold and tx2RelThreshold parameters in Table 195 for a detailed description.

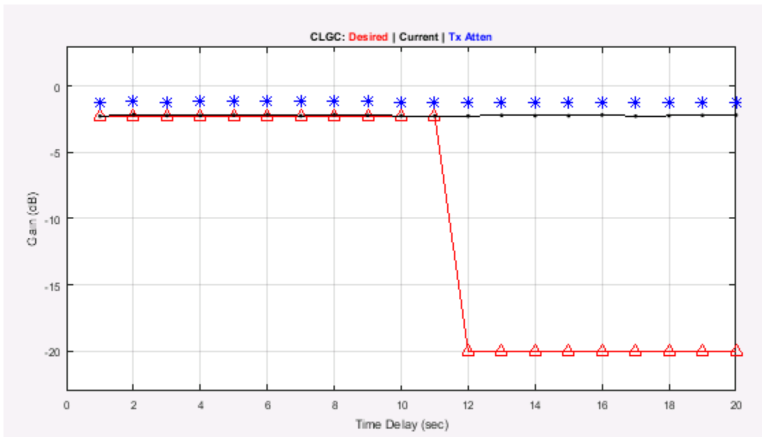
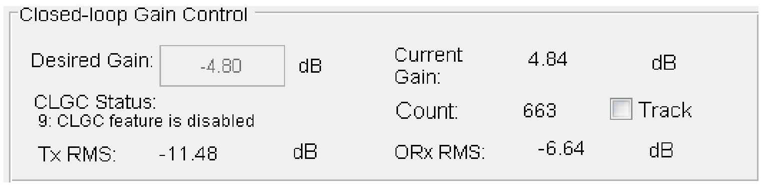


Figure 247. Closed-Loop Gain Control (CLGC) Tracking



Closed-loop Gain Control

Desired Gain: -4.80 dB

Current Gain: 4.84 dB

CLGC Status:
9: CLGC feature is disabled

Count: 663

☐ Track

Tx RMS: -11.48 dB

ORx RMS: -6.64 dB

Figure 248. CLGC Control and Status Display

CLGC API

This section describes all of the application programming interface (API) data structures and functions that are associated with the closed-loop gain control (CLGC) feature.

CLGC API DATA STRUCTURES

The data structures associated with the CLGC functionality are listed in the following subsections.

mykonosClgcConfig_t

The mykonosClgcConfig_t structure is the main data structure that stores all parameters for the CLGC configuration. The information in this structure is loaded into the ARM using the MYKONOS_configClgc() function. The unified modeling language (UML) representation of the structure is shown in Figure 249.

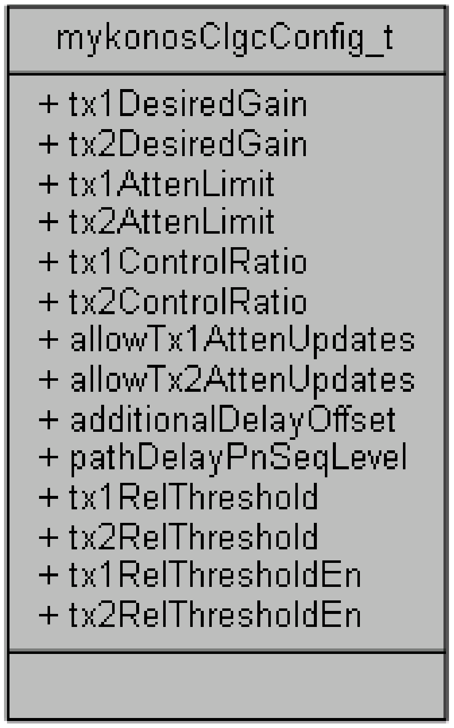


Figure 249. mykonosClgcConfig_t UML Structure

```
typedef struct
{
    int16_t tx1DesiredGain;
    int16_t tx2DesiredGain;
    uint16_t tx1AttenLimit;
    uint16_t tx2AttenLimit;
    uint16_t tx1ControlRatio;
    uint16_t tx2ControlRatio;
    uint8_t allowTx1AttenUpdates;
    uint8_t allowTx2AttenUpdates;
    int16_t additionalDelayOffset;
    uint16_t pathDelayPnSeqLevel;
    uint16_t tx1RelThreshold;
    uint16_t tx2RelThreshold;
    uint8_t tx1RelThresholdEn;
    uint8_t tx2RelThresholdEn;
} mykonosClgcConfig_t;
```

Table 195. AD9375 CLGC Config Structure Member Description

Structure Member	Valid Values	Description
tx1DesiredGain/tx2DesiredGain	−10000…+10000	Value given by 100 × (Desired Gain in dB). Represents total gain and attenuation from AD9375 Tx1/Tx2 output to ORx1/Orx2 input in (dB × 100). Default: 0.
tx1AttenLimit/tx2AttenLimit	0…40	No typical value suggested; user must define value and can be set in increments of 1 dB from 0 dB to 40 dB. Parameter depends on power amplifier (PA) and external loopback attenuation. Value is set to protect PA by making sure that Tx1/Tx2 attenuation is not reduced less than the limit during CLGC tracking. Default value is 0.

Structure Member	Valid Values	Description
tx1ControlRatio/tx2ControlRatio	1…100	This parameter controls the control logic for the CLGC tracking on Tx1/Tx2. Higher values indicate a more aggressive change in the direction of the gain change. This parameter can be seen as a fraction of the gain change that is proportional to the difference between current and desired gains. Default value: 75 which corresponds to a control ratio of 75%.
allowTx1AttenUpdates/allowTx2AttenUpdates	0, 1	A 0 implies that the CLGC calibration runs but that the Tx1/Tx2 attenuation values do not get updated. User can still read back power measurements. A 1 implies that the CLGC calibration runs, and Tx1/Tx2 attenuation values automatically update.
additionalDelayOffset	−64…+64	This parameter adds or reduces delay to the external sample delay assessed during the sample time alignment process performed with the CLGC initialization calibration. Resolution of additional delay is 1/16 of a sample. Delay offset = additionalDelayOffset/16. Using 16 for this parameter implies a 1-sample delay. Default: 0. Changes in the delay offset values requires rerunning the CLGC initialization calibration. Because the effect of tuning this value on calibration performance may not be immediately apparent, it is recommended that a value similar to that in the mykonosDpdConfig_t structure be used here as well.
pathDelayPnSeqLevel	0…8192	Amplitude level of broadband pseudonoise (PN) sequence sent out during CLGC initialization calibration. Default: 255 (−30 dBFS = $20\log_{10}$(Value/8192). Analog Devices recommends using the default value for this parameter unless the PA is highly sensitive to broadband RF input tones. The user must always perform DPD, CLGC, and VSWR initialization calibrations at the Tx attenuation value that corresponds to the final rated power amplifier (PA) operating power conditions, which improves the estimation of the path delay and reduces variability from run to run.
tx1RelThreshold/tx2RelThreshold	0…10000	Value given by 100 × (Relative Gain Threshold in dB). Enforce this threshold by setting tx1RelThresholdEn/tx2RelThresholdEn to 1 in mykonosClgcConfig_t. When the CLGC is tracking at a given gain level in steady state, this threshold represents the maximum dB change allowed in currentGain from one CLGC iteration to the next. ERR_16 is thrown if the relative threshold is violated. Note that the steady state condition implies that the currentGain and desiredGain values are fairly close because the relative threshold test condition checks the currentGain against a bound of desiredGain ± tx1RelThreshold/tx2RelThreshold limit. Default value is 600 (6 dB).
tx1RelThresholdEn/tx2RelThresholdEn	0, 1	A 1 enforces the use of tx1RelThreshold/Tx2RelThreshold limit check set in mykonosClgcConfig_t. A 0 bypasses this check. Default value is 0.

mykonosClgcStatus_t

This structure contains six members that provide information on the closed-loop gain control (CLGC) calibration status. This structure can be used as an error checker and debug tool for the CLGC.

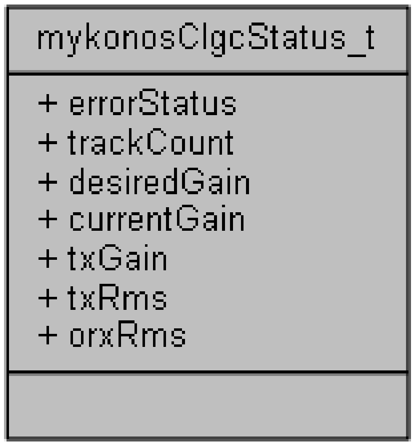


Figure 250. mykonosClgcStatus_t Structure

```
typedef struct
{
    uint32_t errorStatus;
    uint32_t trackCount;
    int32_t desiredGain;
    int32_t currentGain;
    uint32_t txGain;
    int32_t txRms;
    int32_t orxRms;
} mykonosClgcStatus_t;
```

Table 196. AD9375 CLGC Status Structure Member Description

Structure Member	Value	Description
errorStatus	0	No error.
	1	Tx is disabled.
	2	ORx is disabled.
	3	Loopback switch is closed.
	4	Data measurement aborted during capture: calibration interrupted by a higher priority task or calibration. This error is a warning message to note that something interrupted sample collection.
	5	No initialization calibration was run.
	6	Path delay not setup.
	7	CLGC is running but does not apply control. Not enough samples were collected for control signal generation. This error can also occur when the ORx signal is less than the −39 dBFS threshold. Recovering from this error requires manual intervention.
	8	Control value is out of range.
	9	CLGC feature is disabled. This error is displayed when allowTx1AttenUpdates/ allowTx2AttenUpdates are disabled.
	10	Tx attenuation is capped. CLGC control reaches cap limit (Tx1AttenLimit/ Tx2AttenLimit in mykonosClgcConfig_t, see Table 195.).
	11	Gain measurement error during tracking.
	12	No GPIO configured in single ORx configuration.
	13	Tx is not observable with any of the ORx channels.
	14	ORx tracking must be enabled; ORx_QEC tracking calibration must be enabled before running CLGC.
	15	Power amplifier (PA) protection activated: PA Protection hardware feature will preempt any CLGC control. Check GP_INT status.
	16	Relative threshold violated; currentGain of the CLGC increased more than the relative threshold set by tx1RelThreshold/tx2RelThreshold in mykonosClgcConfig_t.
trackCount	0..0xFFFFFFFF	Number of times CLGC has successfully run since CLGC initialization calibration.
desiredGain	−10000..+10000	Desired loop gain (can be negative if total attenuation is greater than amplification) from Tx output to ORx input. Desired Gain_{dB} = desiredGain/100
currentGain	−10000..+10000	Current measured gain in 1/100ths dB scale. Current Gain_{dB} = currentGain/100

Structure Member	Value	Description
txGain	0..4000	Current Tx attenuation setting for a given channel in 0.05 dB resolution, as determined by the CLGC algorithm. However, this value is written to by the CLGC algorithm and may differ slightly from Tx attenuation returned by MYKONOS_getTx1/2Attenuation() due to these actions not being synchronous. Tx Attenuation$_{dB}$ = txGain/200
txRms	−2147483648..+2147483647	RMS Tx digital sample power measured at the output of the DPD actuator. Measurement resolution is 0.01 dB. P$_{rms}$ dBFS = txRms/100.
orxRms	−2147483648..+2147483647	RMS ORx digital sample power measured within the DPD block on the ORx side. Measuremen resolution is 0.01 dB. P$_{rms}$ dBFS = orxRms/100.

CLGC FUNCTIONALITY API FUNCTIONS

The application programming interface (API) functions associated with the closed-loop gain control (CLGC) functionality follow.

MYKONOS_configClgc(…)

```
mykonosErr_t
MYKONOS_configClgc(mykonosDevice_t
*device)
```

Description

This function call configures the device with the parameters in the CLGC data structure, mykonosDpdConfig_t. This function also performs some sanity checks to the data parameters (range and ARM state checks) in the mykonosClgcConfig_t data structure. This function can be called when the device is in either the radioOn or the radioOff state. However, not all parameters in the mykonosClgcConfig_t structure can be changed in the radioOn state. Call this function in the radioOff state when attempting to change the additionalDelayOffset and pathDelayPnSeqLevel parameters. Changes to the latter parameters do not take effect unless modified in a radioOff state and are ignored by the API and ARM.

MYKONOS_getClgcConfig(…)

```
mykonosErr_t
MYKONOS_getClgcConfig(mykonosDevice_t
*device)
```

Description

This function reads the CLGC configuration structure from the ARM memory and updates the device → tx → clgcConfig data structure. There are no radio state dependencies for this function.

MYKONOS_getClgcStatus(…)

```
mykonosErr_t
MYKONOS_getClgcStatus(mykonosDevice_t
*device, mykonosTxChannels_t txChannel,
mykonosClgcStatus_t *clgcStatus)
```

Description

This function reads back the CLGC calibration status from the ARM processor and returns the result to the mykonosClgcStatus_t structure.

Parameters

- *device: This is the device data structure pointer from which the CLGC status is read back.
- txChannel: input argument to set which Tx channel the function reads back the CLGC status for. Valid enumeration values are TX1 or TX2 only.
- *clgcStatus: This is a pointer to the structure that contains the returned CLGC status information.

MYKONOS_setClgcGain(…)

```
mykonosErr_t
MYKONOS_setClgcGain(mykonosDevice_t
*device, mykonosTxChannels_t txChannel,
int16_t gain)
```

Description

This function configures the tx1DesiredGain/tx2DesiredGain parameters in the CLGC data structure, mykonosClgcConfig_t, in the radioOn state.

Parameters

- *device: This is the device data structure pointer for which this function takes effect.
- txChannel: input argument to set which Tx channel is to be modified. Valid enumeration values are TX1, TX2, and TX1_TX2.
- gain: This is the new value written into tx1DesiredGain/tx2DesiredGain.

Note that because the update of most mykonosClgcConfig_t parameters is allowed in the radioOn state, use this function if only the tx1DesiredGain/tx2DesiredGain needs updating. Issuing this command (instead of configClgc() in the radioOn state) can also be slightly faster due to the reduced number of API to ARM interactions.

VSWR TRACKING CALIBRATION

To enable the voltage standing wave ratio (VSWR) measurement, click **Initialize/Reset PA Cals**. Next, check off the **VSWR** box to enable the VSWR tracking calibration. The VSWR monitoring begins, and the **VSWR Monitoring** window shown in Figure 253 starts displaying new values. To calculate the VSWR number, use the rms/complex gains by computing the ratio of the reflected gain to the forward gain to obtain Γ, as shown in Equation 58 found in the Voltage Standing Wave Ratio Measurement Overview section.

VSWR MONITORING

Gain

The VSWR monitoring **Forward Gain** section includes the following:

- **RMS**: forward rms gain measured from Tx to ORx path.
- **REAL**: real part of the forward path complex gain.
- **IMAGINARY**: imaginary part of the forward path complex gain.

The VSWR monitoring **Reverse Gain** section includes the following:

- **RMS**: reflected rms gain measured from antenna to ORx.
- **REAL**: real part of the reflected path complex gain.
- **IMAGINARY**: imaginary part of the reflected path complex gain.

ORx Switch GPIO Setting

The **ORx Switch GPIO Setting** section controls the switch that selects between the forward path and the reflected path. These two paths (forward and reflected) are time multiplexed into the ORx and controlled by the VSWR calibration via a designated GPIO_3.3 V pin.

Note that to change the GPIO pin or GPIO polarity for the switch, the VSWR measurement must be disabled by unchecking the **VSWR** check box.

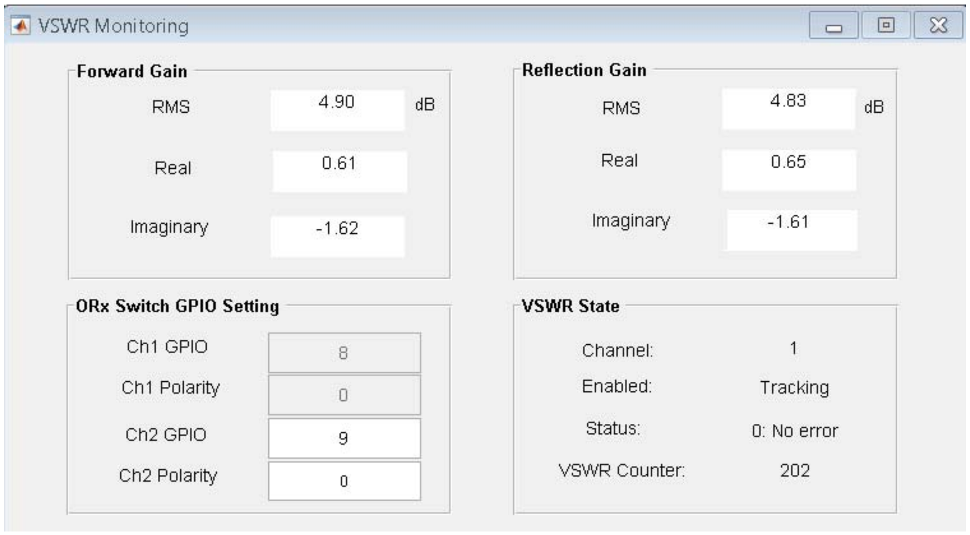
The **ORx Switch GPIO Setting** section includes the following:

- **Ch1 GPIO**: Use this box to select the 3.3 V GPIO pin used to control the VSWR switch on Tx1.
- **Ch1 Polarity**: Use this box to select the polarity of the 3.3 V GPIO pin for the forward path on Tx1 (1 = high level, and 0 = low level). The opposite polarity is used for the reflection path.
- **Ch2 GPIO**: Use this box to select the 3.3 V GPIO pin used to control the VSWR switch on Tx2.
- **Ch2 Polarity**: Use this box to select the polarity of the 3.3 V GPIO pin for the forward path on Tx2 (1 = high level, and 0 = low level). The opposite polarity is used for the reflection path.

VSWR State

The VSWR monitoring **VSWR State** section includes the following:

- **Channel**: The Tx channel for which the VSWR status is being displayed.
- **Enabled**: displays whether the VSWR calibration is currently enabled or disabled.
- **Status**: displays the VSWR error code.
- **VSWR Counter**: displays the number of times the VSWR has been successfully scheduled since VSWR initialization calibration.



Forward Gain
RMS: 4.90 dB
Real: 0.61
Imaginary: -1.62

Reflection Gain
RMS: 4.83 dB
Real: 0.65
Imaginary: -1.61

ORx Switch GPIO Setting
Ch1 GPIO: 8
Ch1 Polarity: 0
Ch2 GPIO: 9
Ch2 Polarity: 0

VSWR State
Channel: 1
Enabled: Tracking
Status: 0: No error
VSWR Counter: 202

Figure 251. VSWR Monitoring Window

VSWR API

This section describes all of the application programming interface (API) data structures and functions that are associated with the voltage standing wave ratio (VSWR) feature.

VSWR API DATA STRUCTURES

The data structures associated with the VSWR functionality are listed in the following subsections.

mykonosVswrConfig_t

The mykonosVswrConfig_t structure is the main data structure that stores all parameters for the VSWR measurement configuration. The information in this structure is loaded into the ARM using the MYKONOS_configVswr() function. The UML representation of the structure is shown in Figure 252.

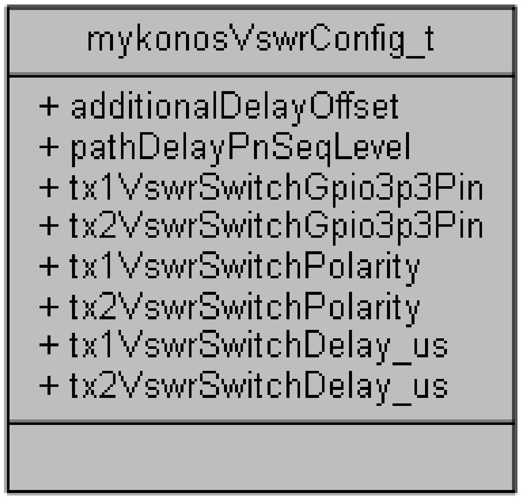


Figure 252. mykonosVswrConfig_t UML Structure

```
typedef struct
{
    int16_t additionalDelayOffset;
    uint16_t pathDelayPnSeqLevel;

    uint8_t tx1VswrSwitchGpio3p3Pin;
    uint8_t tx2VswrSwitchGpio3p3Pin;
    uint8_t tx1VswrSwitchPolarity;
    uint8_t tx2VswrSwitchPolarity;
    uint8_t tx1VswrSwitchDelay_us;
    uint8_t tx2VswrSwitchDelay_us;
} mykonosVswrConfig_t;
```

Table 197. AD9375 VSWR Configuration Structure Member Description

Structure Member	Valid Values	Description
additionalDelayOffset	−64…+64	This parameter adds or reduces delay to the external sample delay assessed during the sample time alignment process. Resolution of additional delay is 1/16 of an ORx sample. Delay offset = additionalDelayOffset/16. Using 16 for this parameter implies a 1-sample delay. Default: 0. Changes in the delay offset values requires rerunning the VSWR initialization calibration. Because the effect of tuning, this value on calibration performance may not be immediately apparent, it is recommended that a value similar to that in the mykonosDpdConfig_t structure is used here, as well.
pathDelayPnSeqLevel	0…8192	Amplitude level of broadband pseudonoise (PN) sequence sent out during VSWR initialization calibration. Default: 255 (−30 dBFS = 20 log10(Value/8192). Analog Devices recommends using the default value for this parameter unless the power amplifier (PA) is highly sensitive to broadband RF input tones. The user must always perform the DPD, CLGC, and VSWR initialization calibrations at the Tx attenuation value that corresponds to the final rated PA operating power conditions, which improves the estimation of the path delay and reduces variability from run to run.

Structure Member	Valid Values	Description
tx1VswrSwitchGpio3p3Pin/tx2VswrSwitchGpio3p3Pin	0…11	The GPIO_3P3_x pin used to control the VSWR switch for Tx1/Tx2 (output from AD9375).
tx1VswrSwitchPolarity/tx2VswrSwitchPolarity	0, 1	The GPIO_3P3_x pin polarity for the forward path of Tx1/Tx2, opposite used for reflection path (1 = high level, and 0 = low level).
tx1VswrSwitchDelay_ms/tx2VswrSwitchDelay_us	0…255	Delay for Tx1/Tx2 VSWR calibration to expect reflection data at ORx (until data capture starts) after flipping the VSWR switch. These have microsecond resolution.

mykonosVswrStatus_t

The mykonosVswrStatus_t structure contains members that provide information on the status of the voltage standing wave ratio (VSWR) measurement. This structure can be used as a status/error checker and debug tool for the VSWR.

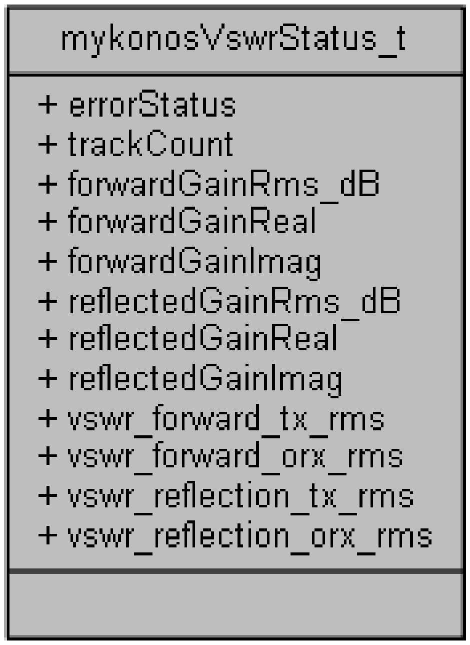


Figure 253. mykonosVswrStatus_t Structure

```
typedef struct
{
    uint32_t errorStatus;
    uint32_t trackCount;
    int32_t forwardGainRms_dB;
    int32_t forwardGainReal;
    int32_t forwardGainImag;
    int32_t reflectedGainRms_dB;
    int32_t reflectedGainReal;
    int32_t reflectedGainImag;
    int32_t vswr_forward_tx_rms;
    int32_t vswr_forward_orx_rms;
    int32_t vswr_reflection_tx_rms;
    int32_t vswr_reflection_orx_rms;
} mykonosVswrStatus_t;
```

Table 198. AD9375 VSWR Status Structure Member Description

Structure Member	Value	Description
errorStatus	0	No error: normal VSWR operation.
	1	Tx datapath not enabled: the transmitter for this channel is currently disabled.
	2	ORx datapath is not enabled.
	3	Loopback switch is closed.
	4	VSWR initialization calibration not run: rerun VSWR initialization calibration.
	5	Path delay error: check external path and rerun VSWR initialization calibration.
	6	Data measurement was aborted: warning message to show that something interrupted data collection.
	7	VSWR disabled: check tracking calibration mask.
	8	Entered calibration but VSWR measurement could not be completed.
	9	No GPIO pin configured for single ORx configuration. Used in shared ORx mode.
	10	Tx is not observable with any of the ORx channels. Used in shared ORx mode.
trackCount	0..0xFFFFFFFF	Number of times VSWR has successfully run since VSWR initialization calibration.
forwardGainRms_dB	0..1000	Forward rms gain measured from Tx to ORx path (1 = 0.01 dB gain).
forwardGainReal	0..1000	Real part of the forward path complex gain (1 = 0.01 linear gain).
forwardGainImag	0..1000	Imaginary part of the forward path complex gain (1 = 0.01 linear gain).
reflectedGainRms_dB	0..1000	Measured reflected path gain in rms (1 = 0.01 dB gain).
reflectedGainReal	0..1000	Real part of the reflected path complex gain (1 = 0.01 linear gain).
reflectedGainImag	0..1000	Imaginary part of the reflected path complex gain (1 = 0.01 linear gain).

Structure Member	Value	Description
vswr_forward_tx_rms	−2147483648..+2147483647	RMS Tx digital sample power measured at the output of the DPD actuator in the forward measurement mode. Measurement resolution is 0.01 dB. Expect a 21 dB offset from the JESD204B dBFS value as well as from the txRms data reported by the CLGC. P_{rms} dBFS = txRms/100 +21 dB.
vswr_forward_orx_rms	−2147483648..+2147483647	RMS ORx digital sample power measured at the DPD block for t.he ORx data in the forward measurement mode. Measurement resolution is 0.01 dB. Expect a 21 dB offset from the JESD204B dBFS value as well as from the txRms data reported by the CLGC. P_{rms} dBFS = orxRms/100 +21 dB.
vswr_reflection_tx_rms	−2147483648..+2147483647	RMS Tx digital sample power measured at the output of the DPD actuator for the reverse measurement. Measurement resolution is 0.01 dB. Expect a 21 dB offset from the JESD204B dBFS value as well as from the txRms data reported by the CLGC. P_{rms} dBFS = txRms/100 +21 dB.
vswr_reflection_orx_rms	−2147483648..+2147483647	RMS ORx digital sample power measured at the DPD block for the ORx data in the reverse measurement mode. Measurement resolution is 0.01 dB. Expect a 21 dB offset from the JESD204B dBFS value as well as from the orxRms data reported by the CLGC. P_{rms} dBFS = orxRms/100 +21 dB.

VSWR FUNCTIONALITY API FUNCTIONS

The application programming interface (API) functions associated with the voltage standing wave ratio (VSWR) functionality are listed in the following subsections.

MYKONOS_configVswr(…)

```
mykonosErr_t
MYKONOS_configVswr(mykonosDevice_t
*device)
```

Description

This function call configures the device with the parameters in the VSWR data structure, mykonosVswrConfig_t. This function also performs some sanity checks to the data parameters (range and ARM state checks) in the mykonosVswrConfig_t data structure. The device must be in the radioOff state before this function can be called.

MYKONOS_getVswrConfig(…)

```
mykonosErr_t
MYKONOS_getVswrConfig(mykonosDevice_t
*device)
```

Description

This function reads the VSWR config structure from the ARM memory and updates the device → tx → vswrConfig data structure. There are no radio state dependencies for this function.

MYKONOS_getVswrStatus(…)

```
mykonosErr_t
MYKONOS_getVswrStatus(mykonosDevice_t
*device, mykonosTxChannels_t txChannel,
mykonosVswrStatus_t *vswrStatus)
```

Description

This function reads back the VSWR calibration status from the ARM processor and returns the result to the mykonosVswrStatus_t structure.

Parameters

- *device: This is the device data structure pointer from which the VSWR status is read back.
- txChannel: Input argument to set which Tx channel the function reads back the VSWR status for. Valid enumeration values are TX1 or TX2 only.
- *vswrStatus: This is a pointer to the structure that contains the returned VSWR status information.

SYSTEMS DESIGN CONSIDERATIONS

DPD TUNING PROCEDURE

The following tuning procedure is suggested to iteratively find the digital predistortion (DPD) configuration parameters that optimize performance for any given power amplifier (PA). As the DPD affects and is affected by many system considerations, a balance must be found between the different performance metrics such that all metrics are within the limits of tolerance of the application or use case. Refer to the mykonosDpdConfig_t section for more details regarding the parameters being adjusted here.

1. Starting with the default setup and a given test waveform (for example, 1 × 20 MHz E-TM 1.1), apply the DPD and note the adjacent channel leakage ratio (ACLR) and other systems metrics.
2. Choose an appropriate DPD modelVersion for the given PA: the model that minimizes asymmetry and performs close to, if not better than, the values noted in Step 1 is probably the best suited for the given PA.
3. Increase or decrease the damping or model averaging factor such that any fluctuations in the spectrum are down to an acceptable minimum.
4. Increase or decrease the samples per update parameter, such that any fluctuations in the spectrum (time varying ACLR) are down to an acceptable minimum. Note that increasing the value of this parameter implies longer DPD adaptation cycles and vice versa. For some PAs, increasing this value may improve the spectral performance (regrowth) further away from the local oscillator (LO) frequency (dc). Because Step 3 and Step 4 are both complementary and similar in some respects, tuning iteratively between these steps before proceeding is recommended.
5. Note the external sample delay in the **DPD Status** section leading up to this step (see Figure 243). Adjust the additionalDelayOffset value in decrements or increments of 1/16 of a sample starting from the default value of 0. Iteratively, find the best value of delay offset that improves the ACLR value (up to 2 dB to 3 dB can be expected in some cases). For some fractional sample delays, the **AM-AM** plot starts misbehaving. It is recommended that the user not exceed these arbitrary boundary values.
6. Once some acceptable level of performance is achieved, adjust the frequency weights to shape the DPD error spectrum more evenly and correct asymmetries in the ACLR.
7. Enable the closed-loop gain control (CLGC) if desired (refer to the CLGC Tracking Cal section for controlling the CLGC). Iterate Step 1 through Step 6 as needed with each gain level. Note that the compensation of PA gain flatness vs. frequency or temperature is not an objective of the AD9375 DPD/CLGC and must be handled by the system of the customer (baseband processor (BBP) or other control).
8. Adjust the modelPriorWeight such that the right level of DPD adaptation agility or inertia is achieved at different attenuation and/or baseband power levels. Refer to Table 193 and the DPD Model Save/Restore Functionality section for more guidance.

Refer to the debug notes in the Troubleshooting Issues by Making Changes to the DPD Configuration Structure section.

DPD MODEL SAVE/RESTORE FUNCTIONALITY

The save and restore functionality included in the AD9375 is designed to speed up or enhance the DPD adaptation in the field. This function can save the DPD model data during the factory calibration phase of base station equipment deployment, such that the saved model(s) can be reused when the operating conditions in the field are similar to the factory test conditions. Therefore, by training the DPD on a known set of parameters and test conditions considered difficult, and by choosing a higher value of the model prior weight (see Table 193) when restoring the saved model, DPD gets a head start on tackling similar conditions in the field.

An alternate application for this functionality is when frequency hopping within certain bands is desired, where the latest DPD model can be saved before switching frequencies, and a host baseband processor (BBP) can restore this saved model when returning to the older frequency of operation. The user is encouraged to devise a scheme that suits the application (for example, with the help of the parameters detailed in Table 194) to determine whether the current DPD model is a good model or not. An example is using the dpdModelErrorPercent as guidance for checking a predefined acceptable threshold before saving the DPD model.

Note that it is the responsibility of the user to record metadata, such as the frequency of operation, DPD configuration structure, and the Tx channel and PA used, while saving the DPD model data. Restoring a DPD model to a different Tx path or configuration than from when the model was saved does not yield the desired performance. Ideally, with all else being equal, restoring a saved model replicates the exact same DPD performance as when the DPD model was saved. Note that the device needs to be in radioOff mode before saving or restoring.

Note that because this DPD relies on having a good prior model when adapting to PA behavior under different operating conditions, users are highly encouraged to leverage the save and restore feature for preloading a prior model.

Practical System Considerations to When Using Save and Restore

As mentioned in the DPD Model Save/Restore Functionality section, the save and restore feature is a useful technique to jumpstart the digital predistortion (DPD) with a known good model. However, the following important practical system considerations and trade-offs must be considered while evaluating and implementing this feature:

- The quality of the saved DPD model is completely dependent on the power amplifier (PA) type and conditions under which the DPD model was saved. This dependence implies that if the operating conditions between when the factory calibration was performed and when the basestation equipment first boots up in the field are drastically different, the saved DPD model may lead to poor DPD adapatations. For example, if the prior model was saved when the system was operational for fifteen minutes in the factory, and it had sufficient time to self heat and reach thermal equilibrium with the environment while the boot-up conditions are drastically different and may be either hotter or cooler than the factory conditions.
- Temperature range considerations. Because the behavior of a PA can change with operating temperature, it may be necessary to save the prior models in different temperature regions. For example, consider saving three models for room, hot, and cold temperature ranges.
- Massive multiple input, multiple output (MIMO) system design. When designing massive MIMO systems, it can become impractical to save models for each path of a 64T64R system, for instance, during the factory calibration due to time, memory, and/or practical limitations.

In addition, note that, due to the limitations in the use cases previously described, it is recommended to also explore alternative options, such as booting the system with no prior model (for example, right after the DPD initialization calibration) and monitoring the DPD status for indications on when a given model of a channel is suitable for saving to memory. Controlling when the highPowerModelUpdate member in the Mykonos-DpdConfig_t is enabled allows the system to be setup to operate with a prior model. If the output spectrum cannot be retrieved in the system of the user (that is, no ORx is routed), the DPD model error and error status in mykonosDpdStatus_t can give a fair indication of whether a given channel has adapted successfully. Some experimentation is required to determine what works best for the overall system of the user, and trade offs between software complexity and system requirements may be needed.

GUIDELINES FOR DEVELOPING AND TROUBLESHOOTING A DPD SYSTEM

In the course of developing a system that uses any DPD solution, there are a number of tests that must be run to pass conformance testing limits set by regulatory bodies such as 3GPP, the FCC, and ETSI. Therefore, while it is important to verify that the basic DPD requirements of the user are met with the integrated DPD solution of the AD9375 (such as adjacent channel leakage ratio (ACLR) correction with desired occupied bandwidth), the user is also encouraged to study various other system design metrics, such as time division duplexed (TDD) configuration timing and its impact on the ARM calibrations scheduler, RF trace and antenna coupling in a MIMO application, impact on factory calibration and test times when using the save/restore functionality for loading a good prior model, studying the impact of 3GPP LTE E-TM2 or broadcast channel only waveforms on spectrum emission mask (SEM) and ACLR requirements, and so on. Most of these topics are covered under separate headings in the following sections, with this section dedicated to troubleshooting DPD specific problems for a system that uses the AD9375. These subsections are especially useful when the customer has migrated from the evaluation phase to bringing up a system prototype.

Dissecting the DPD, CLGC, and VSWR Status Logs

One of the best possible resources that a user has access to when identifying system issues is to make extensive use of the DPD, closed-loop gain control (CLGC), voltage standing wave ratio (VSWR), and other calibration status structures. It is highly recommended that the user build diagnostics into the RF software suite that allow the baseband processor (BBP) to collect time stamped status logs. This strategy helps the user identify issues in the system by observing the response of calibrations to various stimuli (waveforms, operating conditions, and so on) during system bring-up as well as during normal field operation. Because the DPD family of calibrations are dependent on, or seek to control components that reside outside of the AD9375, the calibrations are sensitive to any system level disturbances. Therefore, it is important to have access to contextual diagnostic status readbacks that allow the analysis of the calibration statuses of the system.

EVM Tests

More often than not, the preliminary digital predistorition (DPD) evaluation of the user centers around adjacent channel leakage ratio (ACLR) tests for various signal conditions. However, because the closed loop of the DPD extends well beyond the AD9375 and encompasses the power amplifier (PA) and any other loopback components, any variation that affects the output spectrum is expected to be corrected by the DPD algorithm This correction necessitates the study of both out of band (ACLR and SEM) and in-band (error vector magnitude (EVM)) metrics. Because the EVM captures any latent phase issues in the PA and other loopback components, it is important to study the residual EVM of the entire signal chain with and without DPD with various signals and in different operating conditions. For example, short duration pulses trigger most PA gain and phase drifts. The E-TM2 waveform has sparsely occupied data resource blocks (RBs) and contains a distinct time domain pulsed signature when compared to the E-TM3.1 waveform, which looks more or less uniform (due to the fully occupied nature of the waveform). Therefore, performing EVM tests along with ACLR tests results in a better system check.

Troubleshooting Issues by Making Changes to the DPD Configuration Structure

When initially tuning for ACLR and SEM specifications with a static signal, it is recommended to set the damping or model averaging factor to 1 and the modelPriorWeight to 0. The significance of these parameters becomes more apparent when dealing with dynamic signal cases where the baseband processor (BBP) may switch between different types of signals and/or be accompanied by ramping power up and down digitally for entire waveforms, or by controlling certain RBs. By making these changes, the objective is to identify if the damped correlations or the prior model has degraded the performance. These tests are especially useful when performing long-term stability tests.

DESIGNING A SYSTEM FOR A TDD APPLICATION

Long term evolution (LTE) supports the use of a time division duplexed (TDD) frame, and the AD9375 DPD can be used when the device is set up in TDD mode. Refer to the TDD Configuration and Setup section for more details. TDD configurations vary from a downlink (DL) to uplink (UL) ratio of 20% (TDD Configuration 0) to 80% (TDD Configuration 5). Because DPD runs as an ARM calibration only when the ORx is configured in INTERNAL_CALS mode, the amount of DL data observable by DPD depends greatly on the TDD configuration used and the time allowed by the user for the AD9375 to perform all of its calibrations. Therefore, when some low DL duty-cycle configurations are used, the possibility exists that certain calibrations do not get sufficient time to execute.

To understand this further, it is important to remember that the ARM schedules all of the calibrations based on a priority table. Each calibration attempts to run and capture the requisite number of sample points as soon as the update interval expires, with higher priority calibrations having the ability to interrupt the lower priority calibrations at specific points within a the execution of a calibration. Therefore, it is shown that certain low priority calibrations fail to update in certain TDD configurations. See Table 199 for the update rate for the AD9375 calibrations.

Table 199. Default Priority and Update Intervals for the AD9375 Calibrations

Priority	Calibration	Update Interval (ms)
1	Tx local oscillator leakage (LOL) external	2000
2	DPD	250
3	Closed-loop gain control (CLGC)	250
4	Voltage standing wave ratio (VSWR)	1000
5	Tx quadrature error correction (QEC)	30000
6	ORx QEC	1000
7	Rx QEC	5

Because the DPD configuration structure allows the capturing of a variable number of samples, it is important to tune the ACLR performance not just in frequency division duplex (FDD) but also in TDD, such that a good trade-off exists between acceptable ACLR performance and other metrics, while also allowing sufficient time for the lower priority calibrations to run. In the event that the lower priority calibrations do not run, performance deviates from that stated in the AD9375 data sheet. When the samples parameter fails to provide the necessary margin for calibrations to run, consider changing the update rate of the DPD and CLGC calibrations (because these are the calibrations that take the longest times for their respective data captures). To inquire about instructions for changing the update rate, go to the following: www.analog.com/en/applications/technology/sdr-radioverse-pavilion-home/rf-transceiver-support.html.

EXTERNAL LOOPBACK FLATNESS REQUIREMENTS

The adjacent channel leakage ratio (ACLR) improvement that the AD9375 digital predistortion (DPD) can provide degrades at frequencies where the loopback channel is not flat. The loopback flatness requirement at a given frequency depends on the ideal DPD ACLR improvement at the given frequency. The ideal ACLR improvement is defined as the ACLR improvement observed for the PA with the DPD using a near perfect loopback channel (over 5 × bandwidth).

The theoretical formula for how much the loopback channel gain can vary without degrading the ACLR from this ideal case by more than 1 dBc follows:

$$20\log_{10}\left(1-10^{\frac{-ACLR\ -\ 9}{20}}\right) < GAIN <$$

$$20\log_{10}\left(1+10^{\frac{-ACLR\ -\ 9}{20}}\right)$$

where:
ACLR is the ideal ACLR improvement at the frequency of interest.
GAIN is an allowed gain at that frequency. Note that GAIN is given in dB relative to the loopback gain at the LTE carrier frequency.

For example, in Figure 255, a single 20 MHz LTE carrier is transmitted and centered on the local oscillator (LO) through a gallium nitride (GaN) power amplifier (PA) and with a near ideal loopback channel. The ACLR improvement with DPD is observed as 28 dBc at ±10 MHz.

Looking at Figure 254, gain flatness at ±10 MHz must be <0.1 dB to not degrade digital predistortion (DPD) performance (at ±10 MHz) by more than 1 dBc from the ideal loopback case.

Outside of the ±10 MHz frequency range, the ACLR improvement is smaller, 22 dBc at ±20 MHz. Then, Figure 254 shows that the required loopback flatness out to this frequency is only < 0.25 dB. At ±30 MHz, the ACLR improvement is only 12 dBc. According to Figure 254, loopback flatness is then relaxed and must only be <0.8 dB. Beyond about ±50 MHz, the ideal ACLR improvement is only 5 dBc. In this case, the loopback channel can have up to 1.6 dB greater gain or as low as −1.9 dB less gain and not affect performance. Note that if no ACLR improvement occurs at a given frequency, the flatness at that frequency can be as poor as ±3 dB and not cause noticeable distortion.

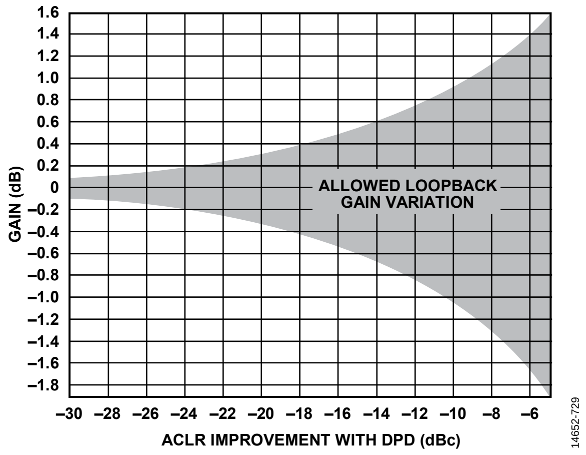
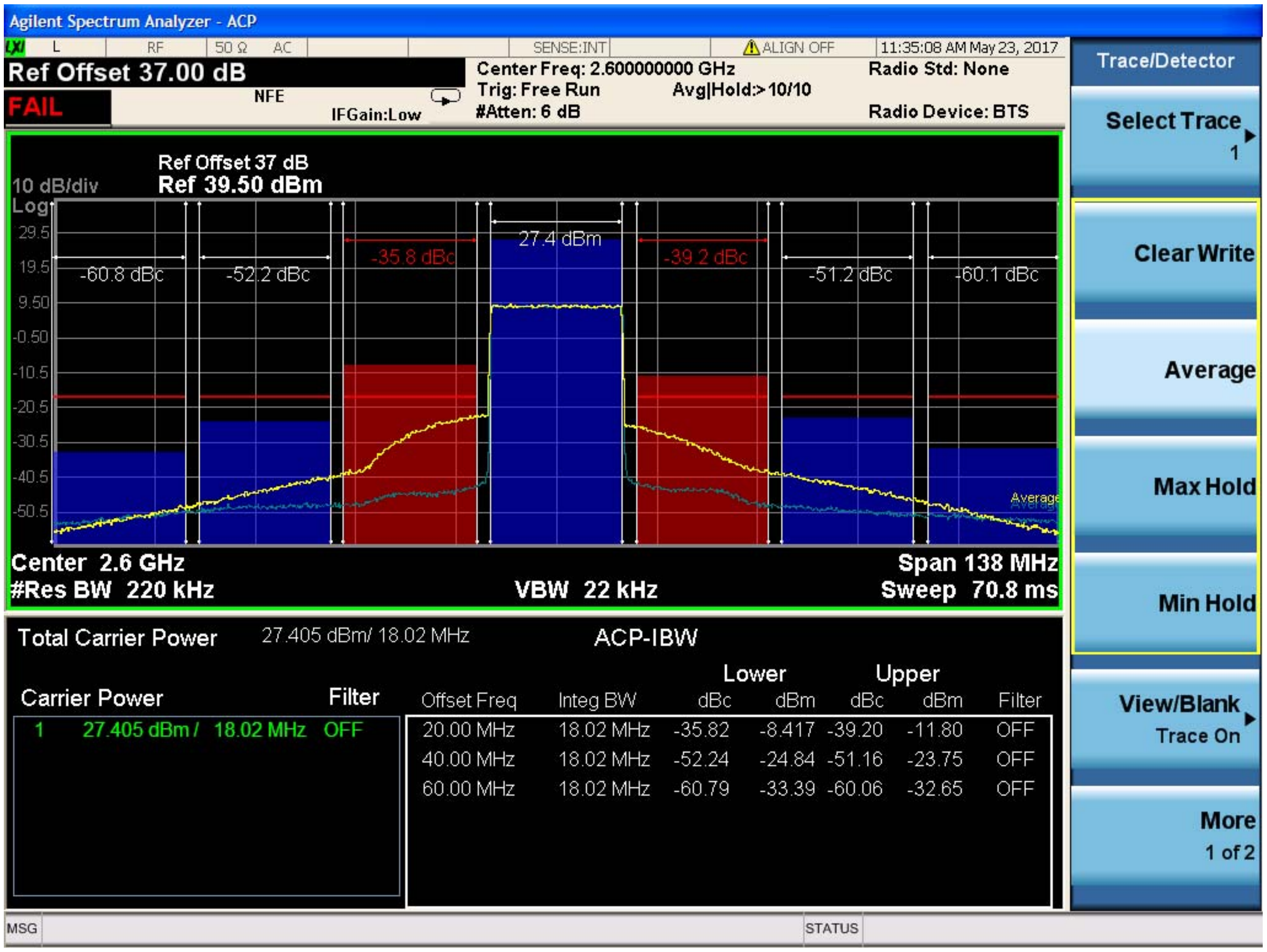


Figure 254. Range Over Which Loopback Gain Can Vary Without Affecting DPD ACLR Improvement by More Than 1 dBc of Degradation

Therefore, the DPD loopback flatness requirements are a function of the desired ACLR improvement. To not degrade optimal predistortion results by more than 1 dB, obey the flatness requirements shown in Figure 254.



Ref Offset 37.00 dB
Center Freq: 2.600000000 GHz
Trig: Free Run
#Atten: 6 dB
Avg|Hold:>10/10
Radio Std: None
Radio Device: BTS

Ref Offset 37 dB
Ref 39.50 dBm

Center 2.6 GHz
#Res BW 220 kHz
VBW 22 kHz
Span 138 MHz
Sweep 70.8 ms

Total Carrier Power 27.405 dBm/ 18.02 MHz

ACP-IBW

Carrier Power			Filter
1	27.405 dBm /	18.02 MHz	OFF

		Lower		Upper		
Offset Freq	Integ BW	dBc	dBm	dBc	dBm	Filter
20.00 MHz	18.02 MHz	-35.82	-8.417	-39.20	-11.80	OFF
40.00 MHz	18.02 MHz	-52.24	-24.84	-51.16	-23.75	OFF
60.00 MHz	18.02 MHz	-60.79	-33.39	-60.06	-32.65	OFF

Figure 255. Power Spectral Density of PA Output Showing Before (Magenta) and After DPD (Yellow)

CLGC CONVERGENCE TIME

The closed-loop gain control (CLGC) convergence time is typically < 5 seconds for full stable convergence. Figure 256 shows a power vs. time graph of the convergence time required for a 20 dB change in the CLGC tx1DesiredGain/tx2DesiredGain parameter (see Table 195). Note that the CLGC is continuously tracking when these changes are applied, and that the AD9375 is in radioOff mode when the config parameter is updated.

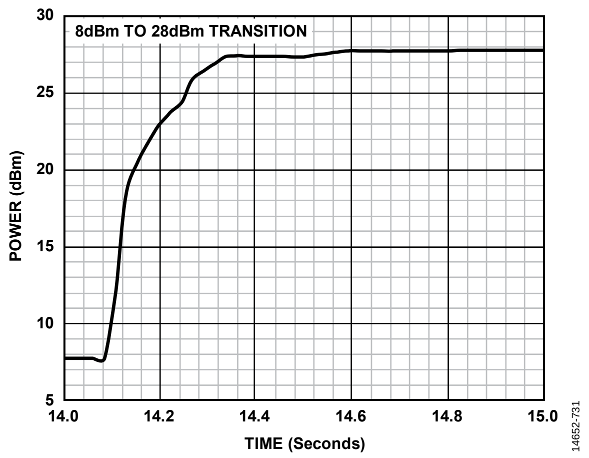


Figure 256. Power vs. Time Showing CLGC Tracking a 20 dB Change in the Desired Gain Parameter

DPD LIMITATIONS

The following subsections outline AD9375 digital predistortion (DPD) limitations and guidelines to observe during system design that allow workarounds of these limitations.

Need for Crest Factor Reduction

The AD9375 DPD seeks to predistort the true radio frequency (RF) behavior of a power amplifier (PA) using a simple behavioral model with a limited number of polynomial terms. Therefore, with the reduced complexity of the DPD solution, it is difficult to model large behavioral variations in the PA at widely varying power levels. One of the easiest ways to improve the reliability and stability of DPD is to employ a crest factor reduction (CFR) algorithm in the baseband processor (BBP) to reduce the peak to average ratio (PAR) of the baseband signal seen by the DPD block. The CFR algorithm helps to prevent cases where the DPD observes a sparse sample set of the nonlinear behavior of the PA (as is the case with high PAR signals) and makes poor guesses or extrapolations of the unobserved behavior. These erroneous estimates can often manifest as spurious signals adjacent to the desired signal on a spectrum analyzer. For example, if a user starts DPD tracking with an E-TM2 signal (PAR = ~12 dB) and no prior information, the DPD sees a very poor excitation of the nonlinear PA behavior and makes poor assumptions. The situation is made even worse when the prior model of the DPD is corrupted with this poor data set. The solution is to train the DPD on a fully occupied E-TM3.1-like signal with an 8 dB PAR or lower to update the prior model with the important nonlinear characteristics of the PA before switching to an E-TM2 or similar broadcast signal. Note that the PA must be operating at its rated power for the model to be a meaningful prior model.

Analog Devices highly recommends the use of a CFR algorithm, however rudimentary, to improve the stability of the DPD algorithm. Even though the user may devise a scheme involving the DPD Model Save/Restore Functionality, the need for a CFR is not entirely obviated, and the CFR must only be used after consulting with an Analog Devices representative at www.analog.com/en/landing-pages/001/sdr-radioverse-pavilion/support.html.

Maximum Occupied Signal Bandwidth

The maximum occupied linearizable signal bandwidth for most PAs is 40 MHz with this DPD. Some PAs may be linearizable at wider RF bandwidths (contiguous or noncontiguous carrier aggregation); however, typical levels of ACLR correction with DPD cannot be guaranteed nor justified in these cases. For PA recommendations that suit specific applications, consult with an Analog Devices representative at www.analog.com/en/landing-pages/001/sdr-radioverse-pavilion/support.html.

Using DPD with GaN PAs

Even though the AD9375 DPD models the RF behavior at baseband of a given PA and doesn't really care about the PA process type nor design architecture, certain types of PAs such as gallium nitride (GaN) PAs can pose significant linearization challenges. Note that some GaN PAs may be linearizable after following the standard tuning procedure; however, there exists a very clear motivation to look at specific system level tests to qualify a PA with DPD. For example, even though the ACLR performance may be within the system design specifications, it is important to also measure error vector magnitude (EVM) for standard signals such as E-TM3.1 and E-TM2. For PA recommendations that suit user specific applications, consult with an Analog Devices representative at www.analog.com/en/landing-pages/001/sdr-radioverse-pavilion/support.html.

OTHER CONSIDERATIONS

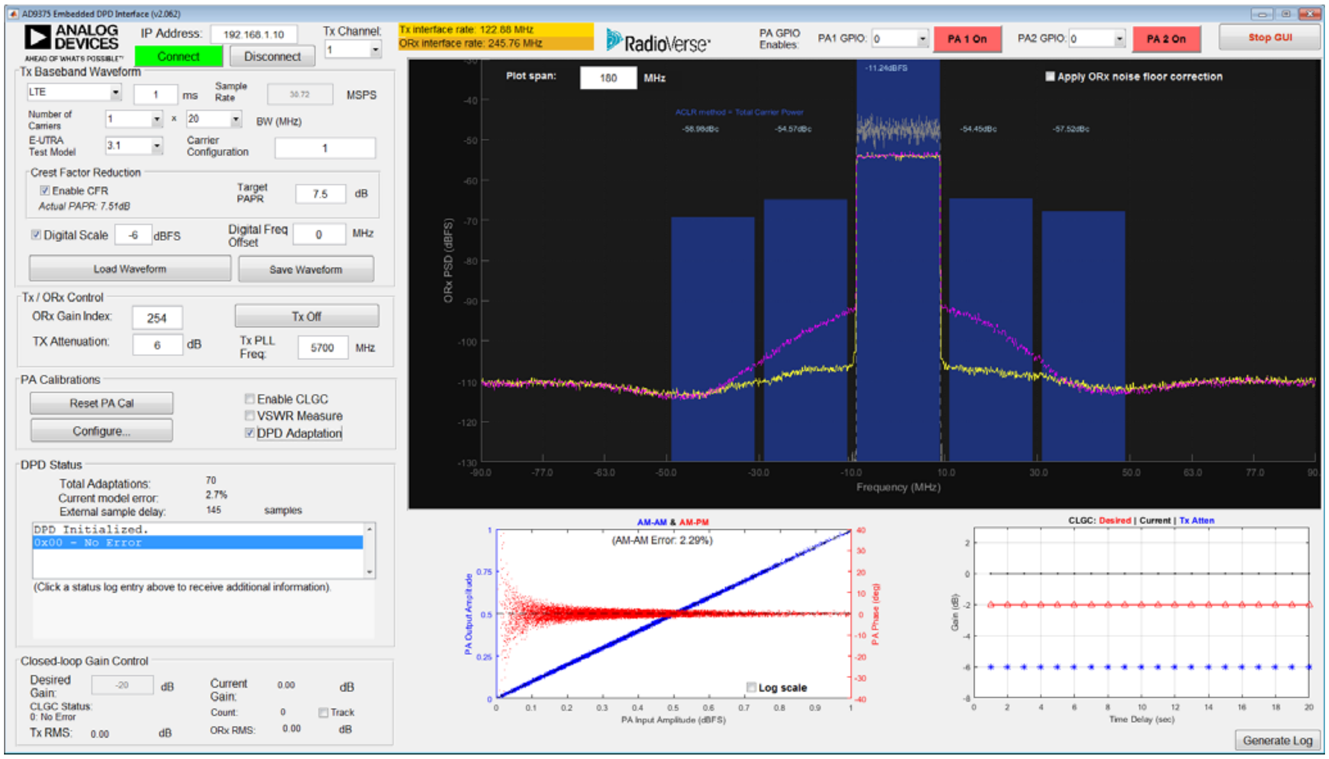
The following are other considerations to keep in mind:

- It is recommended to keep Tx to ORx isolation greater than −55 dBc to avoid any negative effects on DPD performance, which is a useful consideration to keep in mind during printed circuit board (PCB) layout.
- The minimum ORx rms power threshold required by the CLGC is set to −39 dBFS by default; however, this number can be modified, if required, at the expense of accepting some degradation in tracking tolerance at ORx power levels below −39 dBFS. To inquire about setup instructions, go to www.analog.com/en/landing-pages/001/sdr-radioverse-pavilion/support.html.

TYPICAL TEST SETUP AND DPD PERFORMANCE

The SKY66297-11 power amplifier (PA) is optimized for signals of no more than 20 MHz bandwidth. A waveform setup with a total bandwidth greater than 20 MHz pushes the PA out of its optimized range, and performance measures, such as the adjacent channel leakage ratio (ACLR) and error vector magnitude (EVM), may start to degrade. This degradation is due to limitations of the PA and cannot necessarily be attributed to the digital predistortion (DPD) algorithm. To see the full benefit that the AD9375 DPD solution can deliver, the PA must be tested using signals that conform to the limitations of each individual PA. As a first order test to confirm the DPD performance, recreate similar settings as shown in Figure 257 and Figure 258.

Figure 257 shows a typical setup for testing the DPD with the SKY66297-11 PA while keeping the PA within its optimized bandwidth. Note that these are the initial conditions and some configurations inputs can be changed to achieve better performance. Prior to enabling the DPD, the **AM-AM** plot must show compression (weak nonlinearity). If not, reduce the Tx attenuation until the PA **AM-AM** output starts to roll over and compress. Figure 258 shows the DPD performance with the closed-loop gain control (CLGC) enabled. The **ORx Gain Index** can be increased up to a point where most of the points on the **AM-AM** plot are close to the linear curve as possible. The ACLR achieved using this setup after enabling DPD is typically −50 dBc (or better) at a 28 dBm output power (P_{OUT}).



IP Address: 192.168.1.10
Tx Channel: 1
Tx interface rate: 122.88 MHz
ORx interface rate: 245.76 MHz

Tx Baseband Waveform
LTE 1 ms Sample Rate 30.72 MSPS
Number of Carriers 1 × 20 BW (MHz)
E-UTRA Test Model 3.1 Carrier Configuration 1
Crest Factor Reduction
☑ Enable CFR Target PAPR 7.5 dB
Actual PAPR: 7.51dB
☑ Digital Scale -6 dBFS Digital Freq Offset 0 MHz

Tx / ORx Control
ORx Gain Index: 254
TX Attenuation: 6 dB Tx PLL Freq: 5700 MHz

PA Calibrations
☐ Enable CLGC
☐ VSWR Measure
☑ DPD Adaptation

DPD Status
Total Adaptations: 70
Current model error: 2.7%
External sample delay: 145 samples

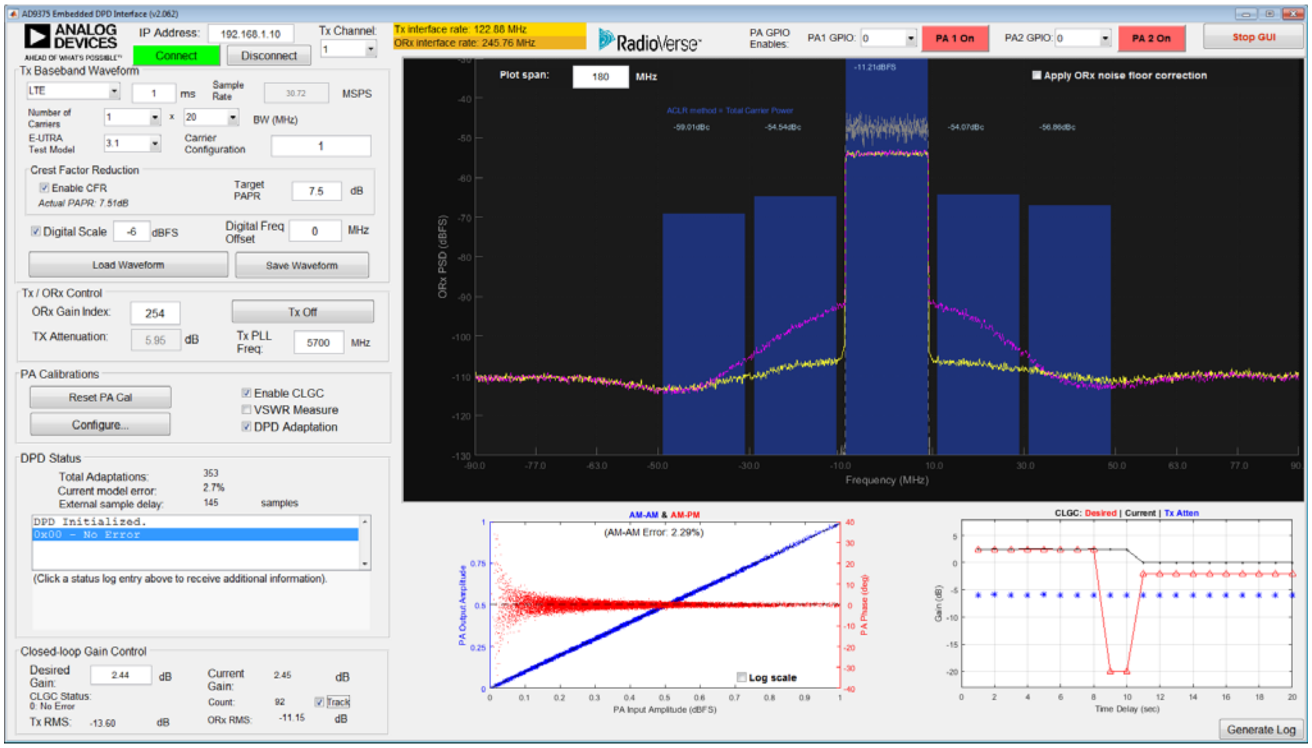
```
DPD Initialized.
0x00 - No Error
```

(Click a status log entry above to receive additional information).

Closed-loop Gain Control
Desired Gain: -20 dB Current Gain: 0.00 dB
CLGC Status: 0: No Error Count: 0 ☐ Track
Tx RMS: 0.00 dB ORx RMS: 0.00 dB

PA GPIO Enables PA1 GPIO: 0 PA2 GPIO: 0
Plot span: 180 MHz
☐ Apply ORx noise floor correction

Figure 257. Typical Test Setup



AD9375 Embedded DPD Interface (v2.062)

IP Address: 192.168.1.10 | Connect | Disconnect | Tx Channel: 1

Tx interface rate: 122.88 MHz
ORx interface rate: 245.76 MHz

PA GPIO Enables: PA1 GPIO: 0 | PA 1 On | PA2 GPIO: 0 | PA 2 On

Plot span: 180 MHz

Tx Baseband Waveform

LTE, 1 ms, Sample Rate 30.72 MSPS
Number of Carriers: 1 × 20 BW (MHz)
E-UTRA Test Model: 3.1; Carrier Configuration: 1

Crest Factor Reduction
☑ Enable CFR; Target PAPR 7.5 dB
Actual PAPR: 7.51dB

☑ Digital Scale -6 dBFS; Digital Freq Offset 0 MHz

Load Waveform | Save Waveform

Tx / ORx Control

ORx Gain Index: 254 | Tx Off
TX Attenuation: 5.95 dB; Tx PLL Freq: 5700 MHz

PA Calibrations

Reset PA Cal | Configure...
☑ Enable CLGC
☐ VSWR Measure
☑ DPD Adaptation

DPD Status

Total Adaptations: 353
Current model error: 2.7%
External sample delay: 145 samples

```
DPD Initialized.
0x00 - No Error
```

(Click a status log entry above to receive additional information).

Closed-loop Gain Control

Desired Gain: 2.44 dB; Current Gain: 2.45 dB
CLGC Status: 0: No Error; Count: 92 ☑ Track
Tx RMS: -13.60 dB; ORx RMS: -11.15 dB

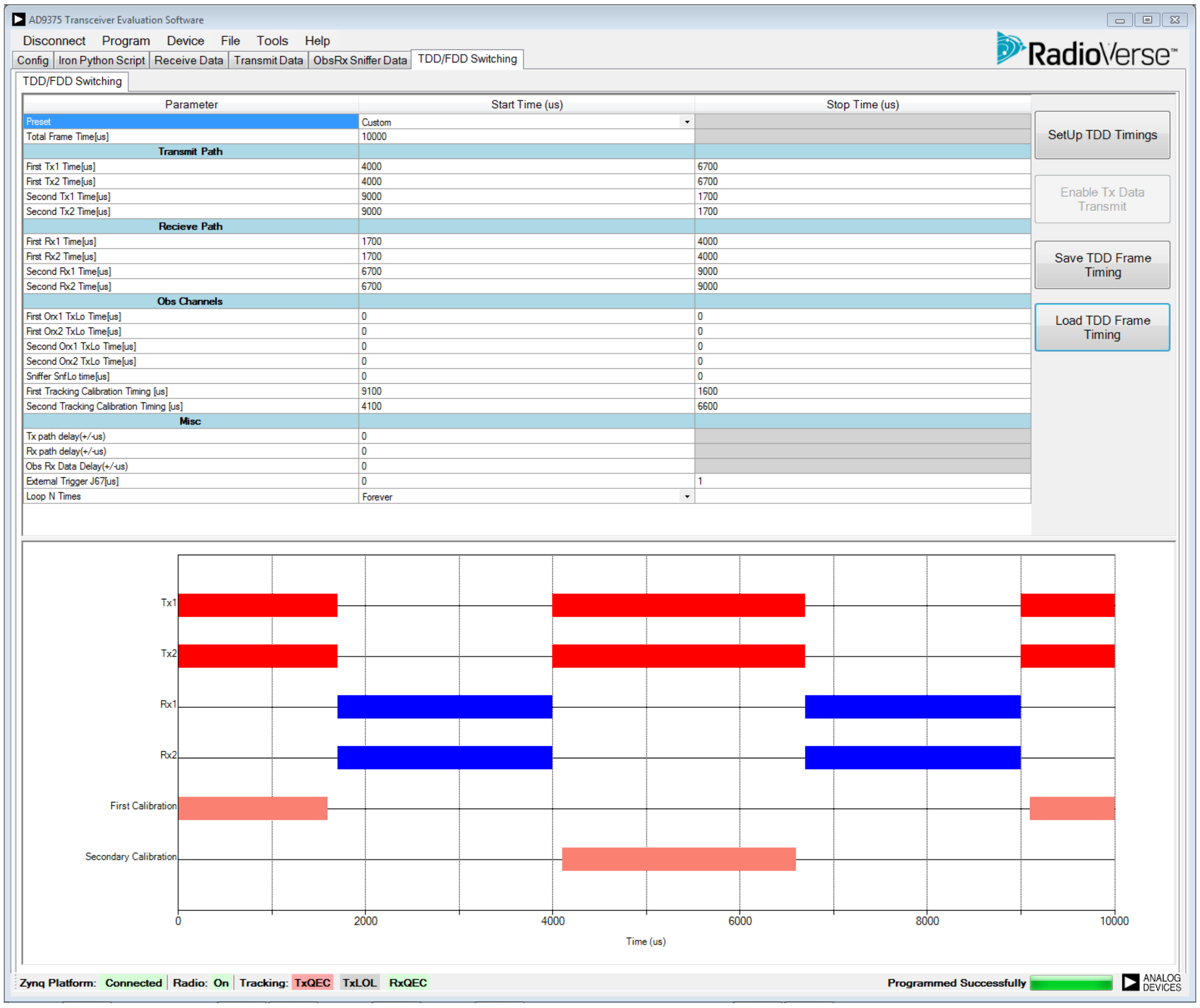
Figure 258. DPD Performance with CLGC Enabled

TDD SETUP INSTRUCTIONS

Apart from the normal user actions necessary for digital predistortion (DPD) operation, take the following steps to perform the DPD with time division duplexed (TDD) waveforms.

1. Program the device to operate in TDD mode using the TES or scripts.
2. Disconnect from the TES and launch the DPD GUI. Because the DPD GUI only has frequency division duplex (FDD) waveforms in its library, use the **Custom** waveform dropdown menu to load the desired TDD waveform. Refer to the Waveform Setup section for more information.
3. Close the DPD GUI and connect using the TES. Navigate to the **TDD/FDD Switching** tab and set up the Tx and Rx frame timing according to the baseband signal loaded in Step 2. Refer to Figure 259 for a Configuration 1 setup.
4. Choose the tracking calibration timing to observe different Tx or downlink (DL) bursts in the TDD waveform (ORx 1 and ORx2 cannot be active at the same time) to enable the DPD to adapt to both the Tx1 and the Tx2 outputs. Allow for some guard period (variable) around the Tx switching time to maximize DPD performance. Alternatively, the user can enable tracking calibrations throughout the frame so that Rx calibrations can also be scheduled during Rx or uplink (UL) bursts. Note that, by default, the DPD, the closed-loop gain control (CLGC), and the voltage standing wave ratio (VSWR) do not track in ORx1 or ORx2 path source modes.
5. Click **SetUp TDD Timings** → **Enable Tx Data Transmit** to start the TDD bursts. After disconnecting from the TES, running the DPD initialization calibration, and enabling the DPD adaptation, observe the DPD status. Note that a spectrum analyzer must be used for observing the spectrum because the ORx data display (ACLR and **AM-AM** plots) used in the GUI cannot display TDD bursts.

Alternatively, the TES **Transmit Data** tab can load a TDD waveform (in this case, skip Step 2). However, the DPD must be separately enabled. If there is some misalignment observed in gated measurements, repeat Step 5 until the frame bursts are properly aligned.

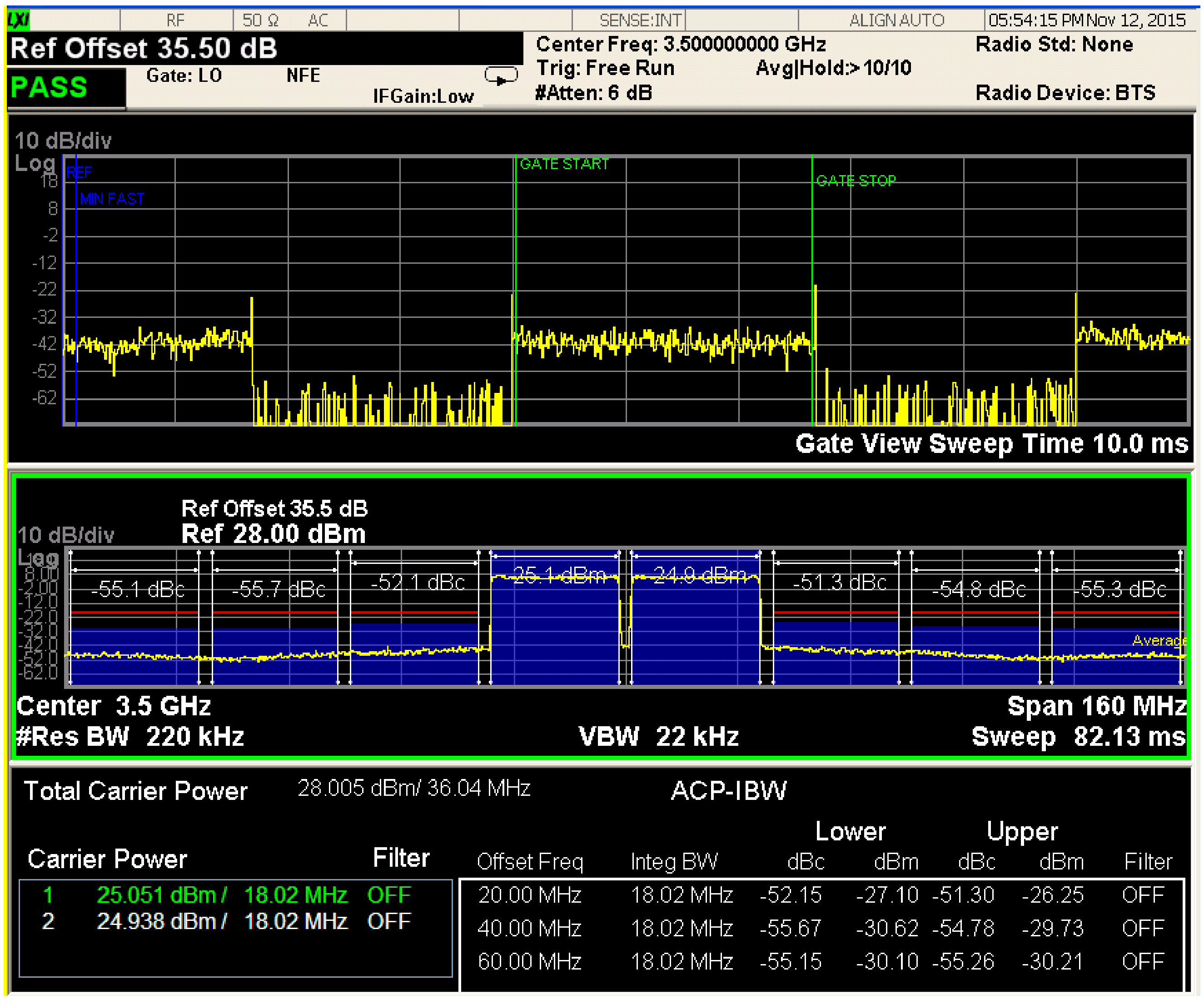


Parameter	Start Time (us)	Stop Time (us)
Preset	Custom	
Total Frame Time[us]	10000	
Transmit Path		
First Tx1 Time[us]	4000	6700
First Tx2 Time[us]	4000	6700
Second Tx1 Time[us]	9000	1700
Second Tx2 Time[us]	9000	1700
Recieve Path		
First Rx1 Time[us]	1700	4000
First Rx2 Time[us]	1700	4000
Second Rx1 Time[us]	6700	9000
Second Rx2 Time[us]	6700	9000
Obs Channels		
First Orx1 TxLo Time[us]	0	0
First Orx2 TxLo Time[us]	0	0
Second Orx1 TxLo Time[us]	0	0
Second Orx2 TxLo Time[us]	0	0
Sniffer SnfLo time[us]	0	0
First Tracking Calibration Timing [us]	9100	1600
Second Tracking Calibration Timing [us]	4100	6600
Misc		
Tx path delay(+/-us)	0	
Rx path delay(+/-us)	0	
Obs Rx Data Delay(+/-us)	0	
External Trigger J67[us]	0	1
Loop N Times	Forever	

Zynq Platform: Connected | Radio: On | Tracking: TxQEC TxLOL RxQEC | Programmed Successfully

14652-734

Figure 259. TES TDD Configuration 1 Waveform Switching Configuration with Tx Tracking Calibrations



05:54:15 PM Nov 12, 2015

Ref Offset 35.50 dB

Gate: LO

IFGain:Low

Center Freq: 3.500000000 GHz
Trig: Free Run
#Atten: 6 dB

Avg|Hold:> 10/10

Radio Std: None

Radio Device: BTS

Gate View Sweep Time 10.0 ms

Ref Offset 35.5 dB
Ref 28.00 dBm

Center 3.5 GHz
#Res BW 220 kHz
VBW 22 kHz
Span 160 MHz
Sweep 82.13 ms

Total Carrier Power 28.005 dBm/ 36.04 MHz

ACP-IBW

Carrier Power			Filter
1	25.051 dBm /	18.02 MHz	OFF
2	24.938 dBm /	18.02 MHz	OFF

		Lower		Upper		
Offset Freq	Integ BW	dBc	dBm	dBc	dBm	Filter
20.00 MHz	18.02 MHz	-52.15	-27.10	-51.30	-26.25	OFF
40.00 MHz	18.02 MHz	-55.67	-30.62	-54.78	-29.73	OFF
60.00 MHz	18.02 MHz	-55.15	-30.10	-55.26	-30.21	OFF

14652-735

Figure 260. TDD Configuration 1 Waveform with Digital Predistortion (DPD), Gated PXA Measurement

ESD Caution
ESD (electrostatic discharge) sensitive device. Charged devices and circuit boards can discharge without detection. Although this product features patented or proprietary protection circuitry, damage may occur on devices subjected to high energy ESD. Therefore, proper ESD precautions should be taken to avoid performance degradation or loss of functionality.

Legal Terms and Conditions
Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners. Information contained within this document is subject to change without notice. Software or hardware provided by Analog Devices may not be disassembled, decompiled or reverse engineered. Analog Devices' standard terms and conditions for products purchased from Analog Devices can be found at: http://www.analog.com/en/content/analog_devices_terms_and_conditions/fca.html.

©2016–2017 Analog Devices, Inc. All rights reserved. Trademarks and registered trademarks are the property of their respective owners.
UG14652-0-5/17(B)

ANALOG DEVICES

www.analog.com