
ADRV910x System Development User Guide for the RF Agile Transceiver Family**ADRV910X SYSTEM DEVELOPMENT USER GUIDE OVERVIEW**

The ADRV910x is the family designator assigned to the System Development User Guide (ADRV910x for [ADRV9101](#), [ADRV9102](#), [ADRV9103](#) and [ADRV9104](#) family members).

This user guide provides details on functionality across the entire family. Some family members do not include all the features or functions. Refer to the individual product data sheet for each product available features and functions.

TABLE OF CONTENTS

ADRV910X System Development User Guide	
Overview.....	1
Block Diagram.....	4
Products Highlights.....	5
ADRV9104.....	5
ADRV910x Product Family Comparison.....	5
ADRV910x Example Use Cases.....	6
ADRV910x IN A TDD SINGLE-BAND 1T1R NARROWBAND APPLICATION.....	6
Software System Architecture Description.....	8
Software Architecture.....	8
Folder Structure.....	9
Customizing the System Architecture and File Structure.....	10
Software Integration.....	12
Hardware Abstraction Layer.....	12
API Error Handling and Debug.....	14
Developing the Application.....	14
System Initialization and Shutdown.....	17
TES Configuration and Initialization.....	17
API Initialization Sequence.....	18
Initialization Example Code.....	19
Shutdown Sequence.....	20
Serial Peripheral Interface (SPI).....	21
SPI Configuration.....	21
SPI Bus Signals.....	21
SPI Data Transfer Protocol.....	22
ARM Binary Write SPI Mode.....	24
Timing Diagrams.....	24
SPI Test.....	26
Synchronous Serial Interface (SSI).....	27
General Description.....	27
Electrical Specification.....	27
CMOS Synchronous Serial Interface (CSSI).....	29
LVDS Synchronous Serial Interface (LSSI).....	35
Enhanced Rx SSI Mode.....	38
Power Saving for LSSI	38
SSI Timing Parameters.....	38
API Programming	40
CSSI/LSSI Testability and Debug.....	41
Microprocessor and System Control.....	44
System Control.....	44
Timing Parameters Control.....	44
Timing Parameters With Power Saving Modes.....	50
Clock Generation.....	51
Synthesizer and LO Operation.....	52
Overview.....	52
Clock Synthesizer.....	52
RF Synthesizer.....	52
External VCO RFPLL.....	53
External LO.....	58
Transmitter Signal Chain.....	59
Overview.....	59
Digital Front End (DFE).....	59
Analog Front End (AFE).....	63
Transmit Signal Chain API Programming.....	63
Receiver and Observation Receiver Signal Chain.....	64
Overview.....	64
Analog Front End (AFE) Components.....	66
Digital Front End (DFE).....	67
Narrowband Demodulator Subsystem.....	71
Receiver Signal Chain API Programming.....	75
Transmitter/Receiver Signal Chain Calibrations..	76
Initial Calibrations.....	76
Tracking Calibrations.....	82
Receiver Gain Control.....	84
Important Terminology.....	84
Receiver DataPath.....	84
Gain Control Modes.....	86
Gain Control Detectors.....	93
AGC Clock and Gain Block Timing.....	95
Analog Gain Control API Programming.....	97
Digital Gain Control and Interface Gain.....	101
Power Saving And Monitor Mode.....	104
Power Saving Modes	104
Power Down/Up Channel in Calibrated State	105
Dynamic Interframe Power Saving.....	105
Monitor Mode.....	106
Digital Predistortion (DPD).....	107
Background.....	107
ADRV910x DPD Function.....	107
ADRV910x DPD Supported Waveforms.....	109
Closed Loop Gain Control (CLGC).....	109
DPD/CLGC Configuration.....	109
DPD/CLGC Updates Scheduling.....	117
DPD Tuning and Testing.....	117
RF Port Interface.....	122
Introduction.....	122
Transmit Port: TX±.....	122
Receive Ports: RX1±, RX2±.....	122
External Local Oscillator Ports: LO1± AND LO2±.....	122
Device Clock Port : DEV_CLK_IN±.....	122
RF Receiver/Transmitter Ports Impedance Data.....	122
Tx Interface.....	125
Rx Interface.....	127
EXT LO Inputs.....	127

TABLE OF CONTENTS

Impedance Matching Network Examples.....	129	ADRV910x System-On-a-Chip (SoC)	
Connection for Device Clock (DEV_CLK_IN).132		Architecture.....	150
Power Supply Overview.....	134	Resources and Examples.....	153
Power Management Considerations.....	134	PS2 System Clock.....	155
Power Supply Sequence.....	134	PS2 Timers.....	156
Power Supply Connections.....	134	Data Path Direct Memory Access (DMA).....	158
Power Supply Architecture.....	135	PS2 Peripherals.....	176
Power Supply Design Budget	136	Boot Mode Configurations.....	184
Improving Noise Performance.....	137	Hardware Accelerators.....	187
Printed Circuit Board Layout		PS1 Control Via PS2.....	220
Recommendations.....	138	Annex.....	223
Selecting the PCB Material and Stackup.....	138	ADRV910X Evaluation System Start Guide.....	233
Fanout and Trace Space Guidelines.....	138	Initial Setup.....	233
Component Placement and Routing		Hardware Kit.....	233
Priorities.....	139	Hardware Operation.....	236
RF and Data Port Transmission Lane Layout	144	Transceiver Evaluation Software (TES).....	237
Isolation Techniques Used on the		Evaluation System Troubleshooting.....	238
ADRV910x Evaluation Board.....	148	PS2 Evaluation and Development Quick	
Processor Subsystem 2 (PS2).....	150	Start Guide.....	239

BLOCK DIAGRAM

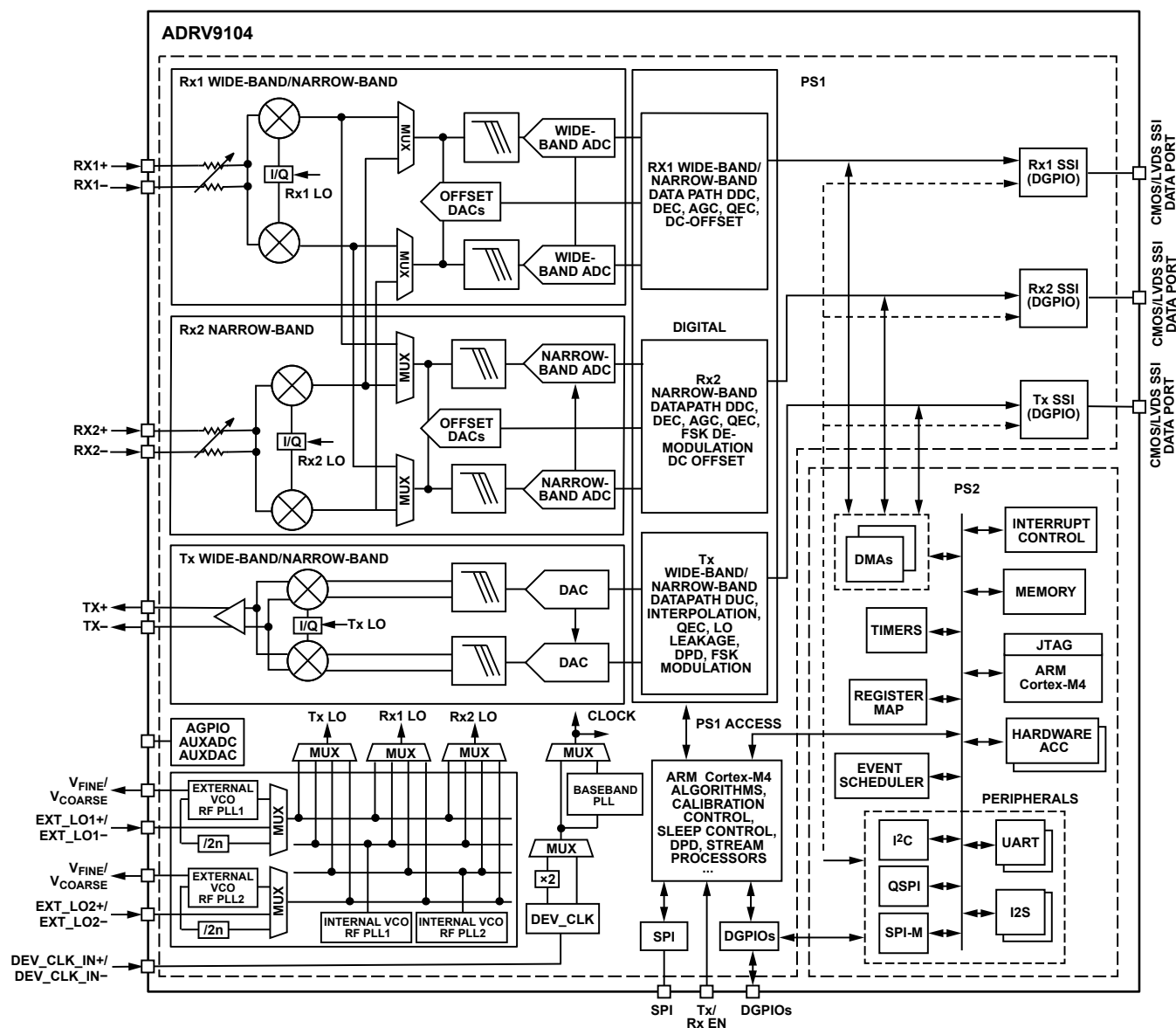


Figure 1. ADRV9104 Block Diagram

PRODUCTS HIGHLIGHTS

ADRV9104

The ADRV9104 is a highly integrated RF transceiver with an integrated application processor that has single-channel transmitter, dual-channel receivers, integrated synthesizers, and digital signal processing functions. It is a high performance, highly linear, high dynamic range transceiver with low power consumption specifically optimized for narrow-band applications, such as low power demanding portable and battery powered equipment.

ADRV9104 operates from 70 MHz to 6000 MHz and covers the VHF, UHF, industrial, scientific, and medical (ISM) bands, and cellular frequency bands in narrow-band and wideband operations. The ADRV9104 is capable of both TDD and FDD operations.

The transceiver consists of three direct conversion signal paths with state-of-the-art noise figure and linearity. Dedicated narrow-band receiver path can support up to 2MHz of RF signal BW. Dedicated wideband receiver path can support up to 40MHz of RF signal BW. Each complete receiver and transmitter subsystem includes dc offset correction (RX), LO leakage correction (TX), quadrature error correction (QEC), and programmable digital filters, which eliminate the need for these functions in the digital baseband processor. In addition, several auxiliary functions, such as auxiliary analog-to-digital converters (ADCs), auxiliary digital-to-analog converters (DACs), and general-purpose inputs/outputs (GPIOs), are integrated to provide additional monitoring and control capability.

The transmitter has internal modulator functions that can support FSK and QPSK/CQPSK modulation for typical Land Mobile Radio (LMR) standards, such as Analog FM, Digital Mobile Radio (DMR), P25, TETRA I. The modulator integrates the symbol mapping, interpolation and pulse shaping functions which allow the baseband processor to send symbols only to ADRV9014.

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

All voltage-controlled oscillator (VCO) and loop filter components are integrated for internal RFLO and Clock PLL to minimize the external component count. Integrated PLL is capable to interface with external Voltage Controlled Oscillator (VCO) to improve overall phase noise performance. The local oscillators (LOs) have flexible configuration options which include fast lock modes.

The transceiver includes low power sleep and monitor modes to save power and extend the battery life of portable devices while monitoring communications, it has the capability to monitor the DMR and Analog FM signal simultaneously with the narrow-band receiver channel when monitor mode is enabled. For some profiles, ADRV9104 can operate from DEV_CLK only with the internal Clock PLL powered down.

The fully integrated, low power digital predistortion (DPD) is optimized for narrow-band /wideband TDD signals and enables linearization of high efficiency power amplifiers.

The ADRV9104 hosts a second processor, Processor Subsystem 2 (PS2), designed for users application specific programming. It features an Cortex M4 ARM processors with up to 928kB of useable memory, operating at frequencies up to 200 MHz, and includes hardware accelerators and multiple peripheral interfaces. This application processor can be used to implement modem like features on the transceiver customized to individual use cases.

The ADRV9104 core can be powered directly from 1.0 V and 1.8 V regulators and is controlled via a standard 4-wire serial port.

High data rate and low data rate interfaces are supported using configurable CMOS or low voltage differential signaling (LVDS) serial synchronous interface (SSI) option.

ADRV910X PRODUCT FAMILY COMPARISON

The configurations of the ADVR910x family are given in [Table 1](#). For descriptions of each feature, please refer to the corresponding section.

Table 1. ADRV910x Family Product Comparison

	ADRV9101	ADRV9102	ADRV9103	ADRV9104
Rx NB	12kHz to 2MHz BW	12kHz to 2MHz BW	12kHz to 2MHz BW	12kHz to 2MHz BW
Rx WB	x	12kHz to 40MHz BW	x	12kHz to 40MHz BW
Tx	12kHz to 2MHz BW	12kHz to 40MHz BW	12kHz to 2MHz BW	12kHz to 40MHz BW
PS2	x	x	✓	✓

ADRV910X EXAMPLE USE CASES

The intention of this section is to provide the reader with the overall idea how ADRV910x integrated transceiver can operate as an RF front end in different applications. The provided list is not exhaustive, and there are other applications in which the ADRV910x can serve.

Each example is accompanied with a table that explains the main limitations and highlights what the customer should look for when implementing the ADRV910x in the end application.

ADRV910X IN A TDD SINGLE-BAND 1T1R NARROWBAND APPLICATION

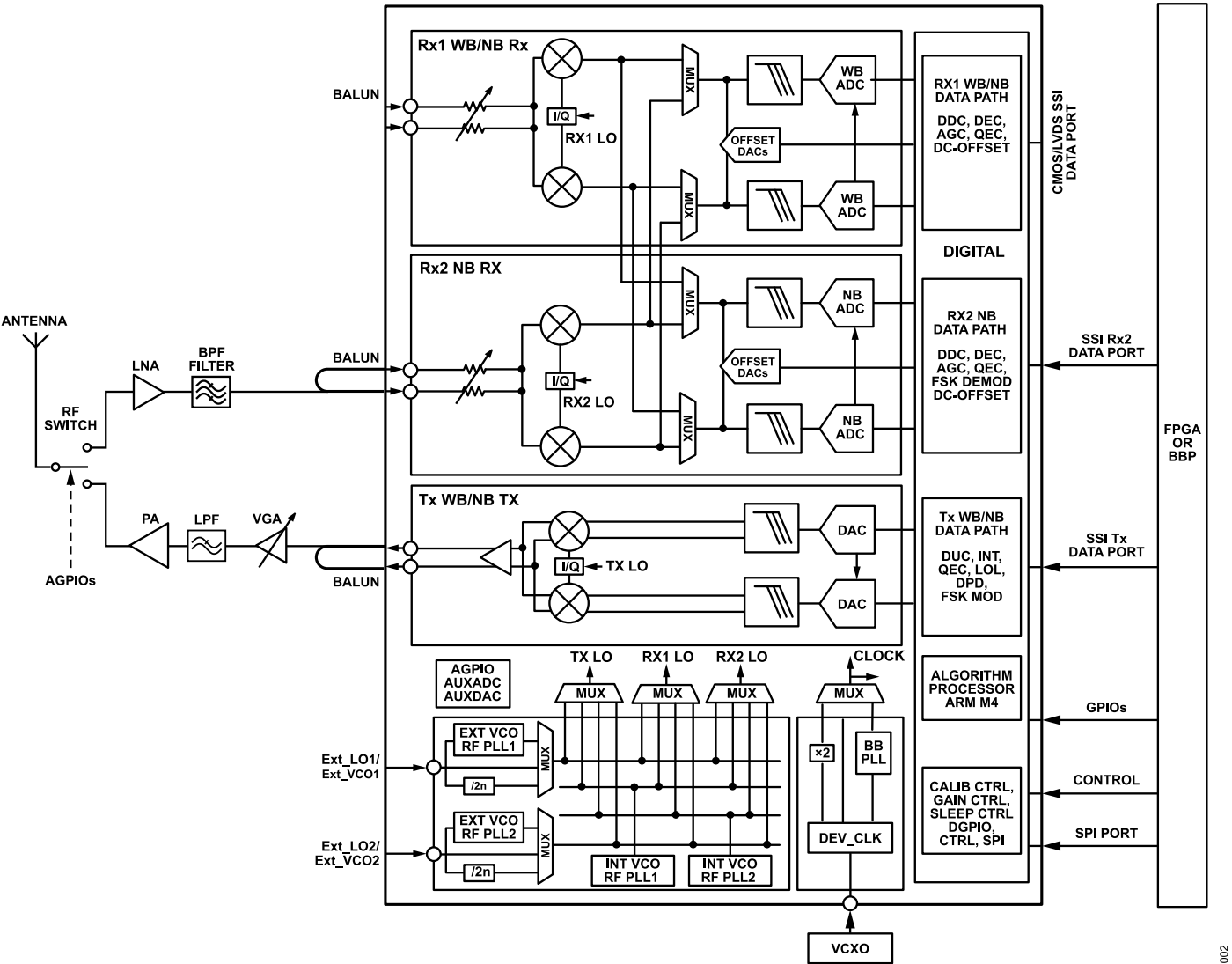


Figure 2. ADRV910x in Single-Band TDD 1T1R Narrowband Application

With a minimum number of external components, the ADRV910x transceiver can be used to build complete RF to bits signal chains that can serve as the RF front end in a TDD Single-Band 1T1R Narrowband Application. The ADRV910x internal AGC can be used to autonomously monitor and set appropriate gain levels for Rx signal chains. For non-time critical TDD type applications, control of the ADRV910x TRx can be done through API commands that use the SPI interface.

Table 2.

Functionality	Constraints and Limitations
LO Generation	ADRV910x can use its internal LO to generate RF LO1 for both uplink and downlink. It is also possible to use external LO/VCO inputs in this mode of operation.
RF Front End	For LO generation, the ADRV910x uses internal or external VCO that generates a square wave type signal. A square wave LO would produce harmonics. For example: depending on RF matching used on the RF ports user 2nd LO harmonic can be as high as -50 dBc and 3rd harmonic

ADRV910X EXAMPLE USE CASES

Table 2. (Continued)

Functionality	Constrains and Limitations
	can be as high as -9 dBc. Therefore, the RF filtering on the Rx and Tx path must ensure that signals at the LO harmonic frequencies (up to 9th in some cases) are not affecting overall system performance.
Calibrations	During Rx initialization sequence user needs to ensure that there are no signals present at the Rx input (external LNA should be disabled) and appropriate termination should be present at LNA output to avoid reflections of Rx calibration tones. During Tx initialization sequence, the user needs to ensure that Power Amplifier is powered down to avoid unwanted emission of Tx calibration tones at the antenna. ADRV910x needs to access Rx datapath during Tx time slots for Tx tracking calibration to operate. If user use Tx observation path with DPD functionality performed by baseband processor, then access to the Rx datapath during Tx slots must be time-shared between DPD operation and Tx calibrations.
AGPIOs	Analog GPIOs (operating at 1.8 V level) can be used as read or write digital levels of in the end user system. AGPIOs can be used to control states of external components such as the RF switch or read back digital logic levels from external components.
DGPIOs	Digital GPIOs can be used to perform real-time monitoring of states of internal ADRV910x blocks. Digital GPIOs operating as inputs can allow user to control Rx gain, Tx attenuation, AGC operation and other elements of ADRV910x TRx.
AuxADC	AuxADC can be used to monitor analog voltage (for example, temperature sensor). Maximum AuxADC input voltage must not exceed 0.9 V.
AuxDAC	AuxDAC can be used to control the VCXO responsible for generating the ADRV910x device clock and control any circuitry that requires analog control voltage up to 1.8 V.
DEV_CLK_OUT	ADRV910x provides divided down version of DEV_CLK reference clock input signal on the DEV_CLK_OUT output. This output is intended to provide reference clock signal to the digital components in the overall system. This output can be configured to be active after power-up and before ADRV910x configuration stage.
Multichip Sync	ADRV910x doesn't support LO phase multichip sync.

SOFTWARE SYSTEM ARCHITECTURE DESCRIPTION

This section provides information about the device driver Application Programming Interface (API) software developed by ADI for the ADRV910x transceivers, as well as outlines the overall architecture, folder structure, and methods for using API software on the customer platform.

Note:

- ▶ This document does not explain the API library functions. Detailed information on the API functions is in the Doxygen document included with the software development kit (SDK) (ADRV910X_API.chm) located at /pkg/production/.
- ▶ The ADRV910X_API.chm is in compressed HTML format. For security reasons, open the .chm files only from a local drive. If there is an attempt to open from a network drive, the file may look empty.
- ▶ The ADRV910x is baseline device for the product family; therefore, all API and evaluation systems use this product number to delineate the product.
- ▶ [Figure 3](#) shows a simple flowchart to the users new to this product family and guides them through the evaluation, testing, and development of the Analog Devices platform and their system.

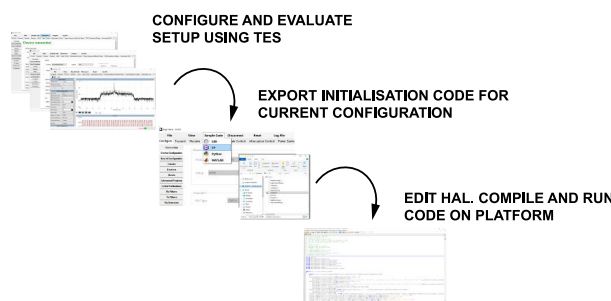


Figure 3. Development Flow Chart

SOFTWARE ARCHITECTURE

[Figure 4](#) shows the software architecture for the ADRV910x evaluation platform from Analog Devices. This chapter takes a high-level overview at the changes that need to be made to this architecture once the baseline setup for the evaluation board has been explained.

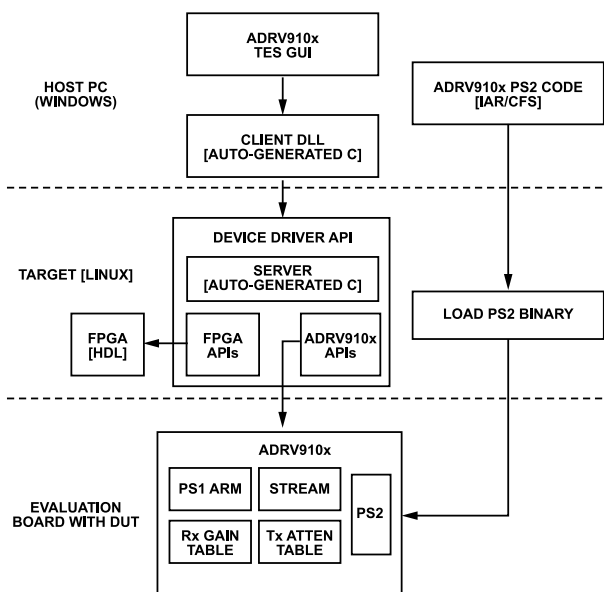


Figure 4. ADRV910x API Software Architecture (ADI Evaluation Platform)

Layering the system, as shown in [Figure 4](#) allows for great flexibility regarding system platform. Any customer beginning their development using the transceiver evaluation software (TES) package can swap out elements in the Target layer of this system architecture with only minor edits to the code base.

SOFTWARE SYSTEM ARCHITECTURE DESCRIPTION

FOLDER STRUCTURE

The API Source files are provided by Analog Devices in the folder structure shown in [Figure 5](#). Each subfolder is explained in the following sections. Analog Devices understands that the developer may desire to use a different folder structure. Whereas Analog Devices provides ADRV910x API source code releases in the folder structure shown in [Figure 5](#), the developer may organize the ADRV910x API into a custom folder organization, if required. This operation, however, does not permit the developer the right to modify the content of the ADRV910x API source code, with the exception of the customer HAL placeholder files, which will be detailed later in this chapter and the Software Integration chapter.

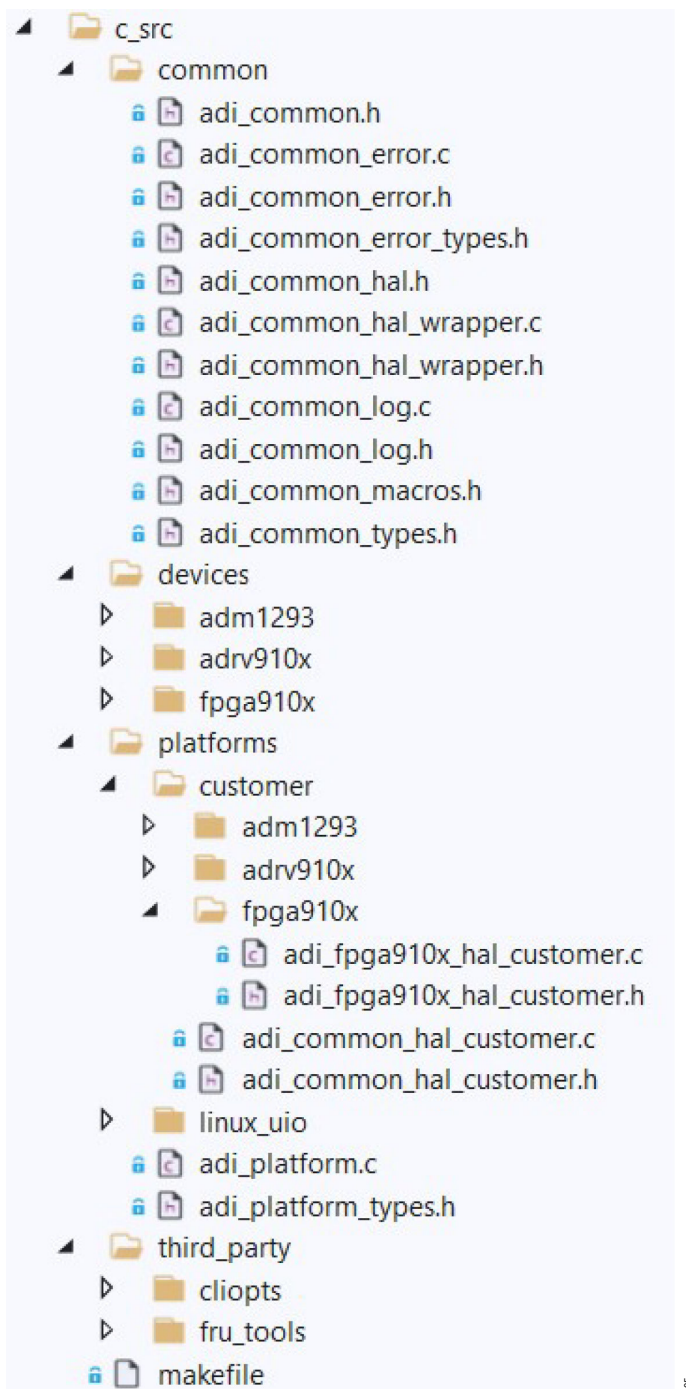


Figure 5. API Folder Structure

SOFTWARE SYSTEM ARCHITECTURE DESCRIPTION

/c_src/common:

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

/c_src/devices:

The device folder includes the main API code for the ADRV910x transceiver as well as auxiliary devices APIs used for the demo of ADRV910x. The /ADRV910x 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 from modifying the files contained in the /ADRV910x or other devices in this section. Software support is not provided when these files have been modified. Analog Devices maintains this code. The only exception is that user may modify #define macros in adi_ADRV910x_user.h, such as modifying polling timeouts and interval settings for various functions.

/c_src/platforms:

The /platforms folder allows a developer to insert custom platform hardware driver code for system integration with the ADRV910X API. The HAL interface is described later in this document. The adi_platform.c/.h files contain function pointers and the required prototypes necessary for the ADRV910x API to work correctly. Do not modify the function prototypes in adi_platform.c. The developer is responsible to implement the code for each function to ensure the correct hardware drivers are called for the platform hardware. In the example code provided by ADI in the customer\ folder, there are placeholder functions left empty to fill their platform-specific code.

/c_src/third_party:

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

CUSTOMIZING THE SYSTEM ARCHITECTURE AND FILE STRUCTURE

The system architecture and file set are uniquely suited for the evaluation platform and change when users progress to developing their own unique solution. This paragraph is designed to give customers at this stage of development a good starting point to build from, as well as build up to more in-depth details which will be provided in [Software Integration](#) chapter. [Figure 6](#) shows the steps to begin the development on a custom platform. The goal here is to migrate from the evaluation platform to a custom, bespoke platform. To do so, take the evaluation platform APIs and HDL out of the TARGET compilation and replace them with the APIs specific to the bespoke platform. To accommodate this migration, use the TES in a slightly different way. Use the TES package to generate C code rather than connecting directly to the TARGET platform through a client-server setup. Make minor edits to the generated files and then deploy the files to the target platform.

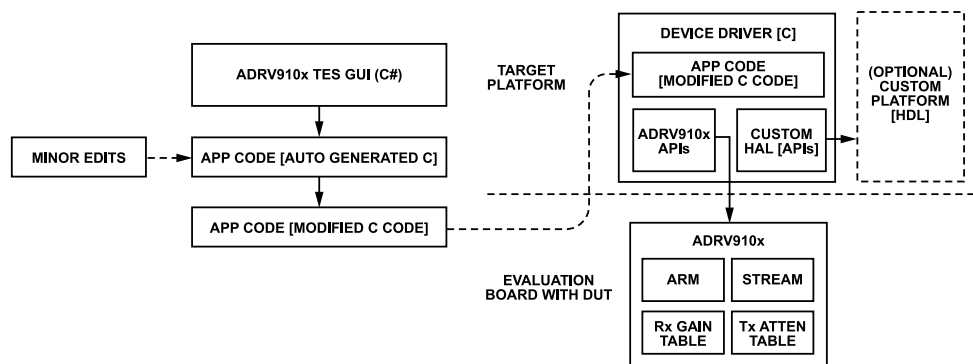


Figure 6. Modified ADRV910x API Software Architecture

These “Minor Edits” listed in [Figure 6](#) consist mostly of removing function calls targeted at hardware components that the customer has no intention of using. For example, by default the generated code seeks to initialize the Evaluation Platform by making calls to “adi_fpga910x_” functions. Given the migration away from this platform, it is necessary to remove those function calls. In a similar fashion, insert any initialization functions needed for the user platform into the generated C code, where they are necessary.

The adi_platform.c file is where the Hardware Abstraction Layer (HAL) is chosen. In the case of the default HAL provided by Analog Devices, it is also implemented in the adi_platform.c file, however no edits should be made to the default HAL stored under linux_uio\ users have been given placeholder files in the customer\ folder. Here all necessary functions have been declared and left empty for the users to fill with their platform-specific code.

SOFTWARE SYSTEM ARCHITECTURE DESCRIPTION

More details are provided in the customer\ folder in the [Software Integration](#) chapter, which goes into more specifics on the HAL. At this point in development, it is recommended to read the files under the customer\ folder before proceeding to edit them.

SOFTWARE INTEGRATION

The ADRV910x API package was developed using the ZCU102 Evaluation platform. This section describes how to use the provided ADRV910x API in a custom hardware/software environment. This is readily accomplished because the API was developed abiding by C99 constructs while maintaining Linux system call transparency. The TES package can be used to produce C99 code to replicate a customer's application while maintaining agnostic processor and operating system integration with the ADRV910x API code.

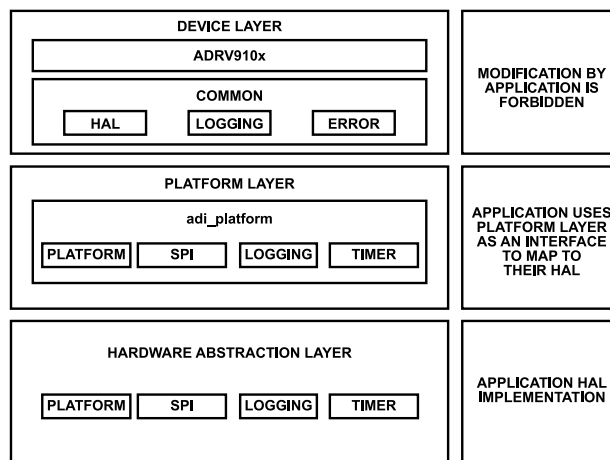


Figure 7. Evaluation System Software Stack

HARDWARE ABSTRACTION LAYER

Users who develop code to target custom hardware platforms use different drivers for the peripherals such as the SPI and GPIO compared to the drivers chosen for the ADI evaluation platform. The hardware abstraction layer (HAL) interface is a set of function pointers that the ADRV910x API uses when it needs to access the target platform hardware. Each device driver (ADRV910x, FPGA910x, ADM1293) has its own HAL, defined in their respective `adi_<device>_hal.h` files. This allows to select only the components which are desired. The user is responsible for implementing the interface defined in each HAL to use the corresponding device driver with their specific platform. The `adi_platform.c` file maps function pointers to the platform-specific HAL implementations. The implementation of this interface is platform dependent and needs to be implemented by the user. The function pointers associated with the user HAL layer must be set in `adi_platform.c`.

Below is a code snippet from the beginning of `adi_platform.c`:

```
#ifndef CUSTOMER_PLATFORM

#include "adi_linux_uio_logging.h"
#include "adi_linux_uio_timer.h"
#include "adi_adrv910x_hal_linux_uio.h"
#include "adi_adm1293_hal_linux_uio.h"
#include "adi_fpga910x_hal_linux_uio.h"

/* Logging interface */
int32_t(*adi_common_hal_LogWrite)(void *devHalCfg, uint32_t logLevel, const char *comment, va_list args) = linux_uio_LogWrite;

/* Timer interface */
int32_t(*adi_common_hal_Wait_us)(void *devHalCfg, uint32_t time_us) = linux_uio_TimerWait_us;
```

Figure 8. Code Snippet from `adi_platform.c`

The file continues from this point to set the functions which handle the register reading and writing, the SPI reading and writing, assigning the RESET pin, etc. Taking note of the second line of the file, a customer need only define a `CUSTOMER_PLATFORM` macro for the compiler pre-processor to use their own platform code. Doing so changes the operation of the `adi_platform.c` file as follows:

```
#else

#include "adi_common_hal_customer.h"
#include "adi_adrv910x_hal_customer.h"
#include "adi_fpga910x_hal_customer.h"

/* Logging interface */
int32_t(*adi_common_hal_LogWrite)(void *devHalCfg, uint32_t logLevel, const char *comment, va_list args) = customer_LogWrite;

/* Timer interface */
int32_t(*adi_common_hal_Wait_us)(void *devHalCfg, uint32_t time_us) = customer_TimerWait_us;
```

Figure 9. Custom Platform Code

SOFTWARE INTEGRATION

Once done, the `adi_platform.c` code will automatically switch to using the placeholder customer code under the `customer/` folder.

Below is provided a code snippet from `adi_adrv910x_hal_customer.c`, located under `customer\adrv910x\`. It shows each function required by the HAL to support a customer-specific platform. Read the example HAL implementation provided under the `linux_uio\` folder to understand the purpose of each function, as well as the acceptable return values. Function names may be modified, if required, provided the pointer assignments in `adi_platform.c` are updated accordingly. However, the function parameters may not be modified as this causes compilation errors.

```
#include "adi_adrv910x_hal_customer.h"

int32_t customer_adrv910x_hal_open(void *devHalCfg)
{
    /* Customer code goes here */
    return 0;
}

int32_t customer_adrv910x_hal_close(void *devHalCfg)
{
    /* Customer code goes here */
    return 0;
}

int32_t customer_adrv910x_hal_spi_write(void *devHalCfg, const uint8_t txData[], uint32_t numTxBytes)
{
    /* Customer code goes here */
    return 0;
}

int32_t customer_adrv910x_hal_spi_read(void *devHalCfg, const uint8_t txData[], uint8_t rxData[], uint32_t numTxRxBytes)
{
    /* Customer code goes here */
    return 0;
}

int32_t customer_adrv910x_hal_resetbPin_set(void *devHalCfg, uint8_t pinLevel)
{
    /* Customer code goes here */
    return 0;
}
```

Figure 10. Code Snippet from `adi_adrv910x_hal_customer.c`

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

/c_src/platforms/customer/

In Figure 11 is shown an expansion of the `customer/` folder. This folder contains all the necessary files for building a custom platform with support for the hardware present on the ADRV910x Eval Board.

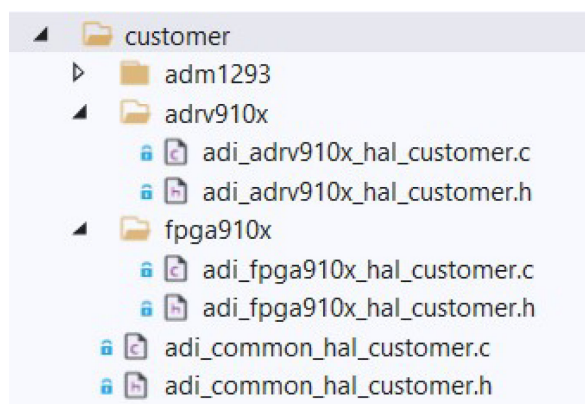


Figure 11. Customer Folder Expanded

Most of these files can safely be left at their defaults, most function calls defined in this section of the code base are optional. For example, the `adm1293\` folder houses the code to interact with the ADM1293 device (used for power monitoring) through I2C. However, this device is only used by TES to provide power measurements on ADRV910x evaluation board. It has no essential function in the operation of the Eval Board. The `fpga910x\` folder houses the function designed to access and manage memory on the user's platform, as well as handle the synchronous serial interface (SSI) type configuration, direct memory access (DMA), and random access memory (RAM) readings. These

SOFTWARE INTEGRATION

functions are necessary if a user's custom application requires a platform connected to the SSI ports with access to RAM, DMA, and register memory. However, not every application needs such a complex platform. Simpler applications, which do not need access to these elements of functionality, can safely ignore this folder.

The most essential folder is the ADRV910x\ folder, which houses the code necessary for instantiating and deconstructing the customer HAL, the SPI functionality and the RESET pin control. These are the minimum functions required to interact with the ADRV910x device provided you can power it.

API ERROR HANDLING AND DEBUG

Logging Functions

The API provides a simple logging feature function that may be enabled for debugging purposes. Available logging levels are given by `adi_common_LogLevel_e` as shown in [Table 3](#).

Table 3. Logging Level

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

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

Error Handling

Each ADRV910x API function returns an `int32_t` value representing a recovery action, with 0 being no action or success. Recovery actions are divided into:

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

DEVELOPING THE APPLICATION

The user application must allocate the `init` (`deviceProfile_t`) and `device` (`adi_adrv910x_Device_t`) structures. Consider allocating memory from the heap for the `adi_adrv910x_Device_t` and `deviceProfile_t` as the structures have members with a large memory size. As part of the SDK, there is a `memory_profile.py` script designed to profile the storage of application variables. Users concerned about memory usage or limitations may find this script beneficial. There is a section of the `README.md` under the `production\` folder dedicated to "Running the memory profiler".

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

The `deviceProfile_t` structure contains the user profile initialization settings to configure the ADRV910x device. This 'init' structure is passed to the ADRV910x API 'init' functions during the initialization phase. This structure contains the device profile settings, system clock settings, data interface settings, and ADRV910x specific SPI subordinate controller settings. The application layer passes a pointer to an instance of the `deviceProfile_t` structure for a particular ADRV910x device to handle most of the device core initialization. After initialization is complete, the `deviceProfile_t` structure may be deallocated if desired.

SOFTWARE INTEGRATION

The `adi_adrv910x_Device_t` data structure contains information for a particular ADRV910x device, including `devHalInfo`, error and caching structures. To support multiple ADRV910x devices, the Application would need to instantiate multiple `adi_adrv910x_Device_t` structures to describe each physical ADRV910x device. Multiple ADRV910x devices can have their own `deviceProfile_t` or can share a common `deviceProfile_t` if they are to be configured identically.

devHalInfo

`devHalInfo` is a structure to define and pass any platform hardware settings to the platform HAL layer functions. The common device structure `adi_common_Device_t` contains `devHalInfo`. The `devHalInfo` is passed to the platform specific HAL function as a `void *devHalCfg`. ADRV910x API functions shall not read or write the `devHalInfo` but pass it as parameter to all HAL function calls.

The application developer must define `devHalInfo` per system HAL implementation requirements. The Application developer may implement any structure to pass any hardware configuration information that the hardware requires between the application layer and platform layer. For example, `devHalInfo` contains SPI chip select information to be used for the physical ADRV910x device.

Note that the API functions are shared across all instances of physical ADRV910x devices. The `devHalInfo` structure defined by the developer identifies which physical ADRV910x device is targeted (SPI chip select) when a particular ADRV910x API function is called. The developer may need to store other hardware information unique to a particular ADRV910x device in this structure such as timer instances or log file information.

Note for ADRV910x API, only one thread may control and configure a specific device instance at any given time.

devStateInfo

The `devStateInfo` member is of type `adi_adrv910x_Info_t` and is a runtime state container for the ADRV910x API. The application layer must allocate memory for this structure, but only the ADRV910x API writes to the structure. The application layer should allocate the `devStateInfo` structure with all zeroes. The API uses the `devStateInfo` structure to keep up with the current state of the API (is it initialized, ARM loaded, etc.), as well as a debug store for any run-time data, such as error codes, error sources, and so forth. It is not intended for the application layer to access the `devStateInfo` member directly because API functions are provided to access information of the last error.

Private Vs. Public API Functions

The API is made up of multiple `.c` and `.h` files. Since the API is written in C, there are no language modifiers to identify a function as private or public as commonly used in object-oriented languages. Per the ADI coding standard, public API functions are denoted by the function name prepended with “`adi_adrv910x_`” (for example, `adi_adrv910x_Rx_Gain_Set()`). Private helper functions lack the `adi_` prefix and are not intended to be called by the user application.

Most functions in the ADRV910x API are prefixed with “`adi_adrv910x_`” and are for public use. However, many of these functions are never called directly from the application. Utility functions that abstract some common operations, specifically initialization of the ADRV910x, are provided in `adi_adrv910x_utilities.c`. For this reason, much of the initialization and other helper functions have been separated from the top-level `adi_adrv910x.c/adi_adrv910x.h` files to help the developer focus on the functions most commonly used by the Application.

Include Files

For each major function block, there are generally three files: `adi_[feature].c`, `adi_[feature].h` and `adi_[feature]_types.h`. The ADRV910x API places typedef definitions in files with ‘`_types`’ suffix such as `adi_adrv910x_types.h`. These `_types.h` files are included within their corresponding `.h` files and do not need to be manually included in the application layer code.

Note that the `adi_adrv910x_user.h` contain the `#defines` for API timeouts and SPI read intervals which may be set as needed by the user platform. The ADRV910x user files are the only API files that the developer may change.

Restrictions

Analog Devices maintains the code in the `/c_src/devices/*` folders. Application developers must not modify this code. Direct SPI read/write operation is forbidden when configuring an ADRV910x or any other ADI devices used for the evaluation of ADRV910x. Developers should only use the high-level API functions defined in the public `*.h` files. Developers should not directly use any SPI read/write functions in the application for ADRV910x configuration or control. Analog Devices does not support any application containing SPI writes that are reverse engineered from the original ADRV910x API.

SOFTWARE INTEGRATION

Delays, Waits, and Sleeps

A subset of ADRV910x APIs require delays to allow the hardware to complete internal configurations. These ADRV910x APIs request the system to perform a wait or sleep by calling the HAL interface function `adi_common_hal_Wait_us`. If the target platform's HAL interface implementation chooses to implement a thread-sleep, it is not permitted for the application to call another API targeting the same ADRV910x device. The application is required to wait/sleep the API to complete before continuing with the configuration of the device.

Wait/sleep periods are defined in `adi_adrv910x_user.h`. The timeout period values are the recommended period required to complete the operation. Modifying these values is not recommended and may impact performance. During this time-out period, the status of ADRV910x is polled. The frequency of the polling the status during this timeout period may be modified by the user by adjusting the value of the polling interval.

SYSTEM INITIALIZATION AND SHUTDOWN

There is a graphical user interface (GUI)-based Transceiver Evaluation Software (TES) to initialize and interact with the ADRV910x device for evaluation. Through this TES, provide high level system configuration parameters such as narrow band or wide band profiles, signal bandwidth, sample rate, enable PS2, HWA and initial gain control settings to initialize the device. The TES uses these parameters to set up an initialization C structure and then makes multiple API calls in a proper order to initialize the device. During the normal operation of the device, the TES allows further user interaction with the device, such as adjusting the transmit/receive gain on the fly. When the operation is completed, safely shut down the device through TES.

Figure 12 shows the high-level flow of the device operation sequence and the user interaction through TES.

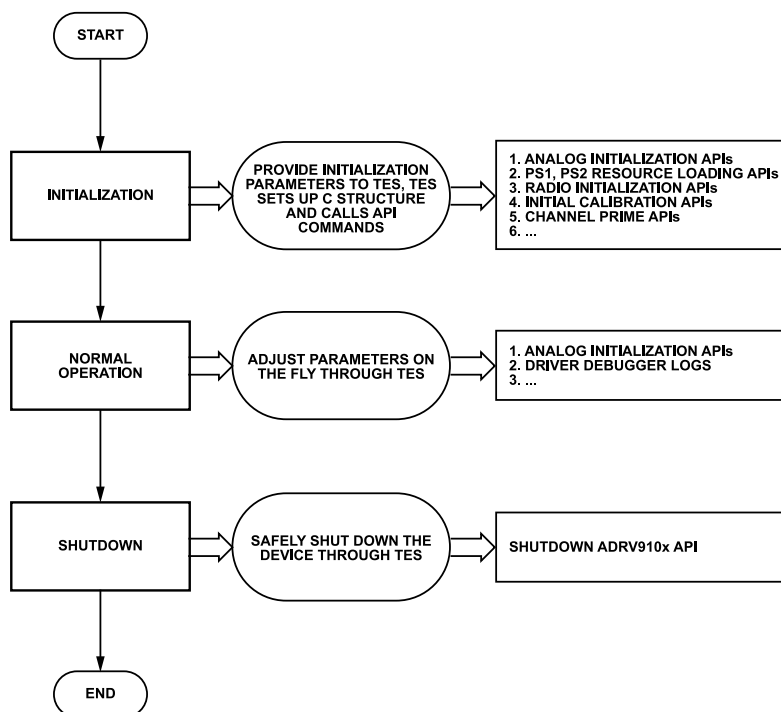


Figure 12. High Level Flowchart of the Device Operation and User Interaction Through TES

This section provides information about the initialization and shutdown process for the ADRV910x device using the APIs developed by ADI. Figure 12 lists the high-level operations used for initialization and shutdown through one or a set of APIs. The further sections discuss the major steps associated with each high-level operation. Note with the SDK, user can initialize the ADRV910x device through own software program independent of the TES. However, it should follow the same API calling procedure described in this document.

Note: This section does not explain every related API function. For detailed information on the API functions, refer to the ADRV910x Device API Doxygen document. In addition, this section does not describe API integration and the hardware abstraction Interface. For more details, see the [Software Integration](#) section.

TES CONFIGURATION AND INITIALIZATION

The TES provides the profile configuration tabs, for Example “Use Case”, “Carriers”, “PS2”, “Init Cals” and so on, which set the ADRV910x use case and initialization profile including channel setup, device clock frequency, channel bandwidth, data port sample rates, LO Mapping, carrier frequency, PS2, etc.

Based on the profile settings, the TES Profile Generator formulates an initialization structure `deviceProfile_t` to contain all the required settings to configure the device. This structure contains the settings for the system configuration, system clock, transmit/receive data structure, and programmable FIR filter, PS2 profiles, etc. For more details, refer to the Doxygen document.

When all tabs are configured, press the **Program Chip** button in TES. This starts the initialization programming. TES sends a series of API commands executed by a dedicated Linux application on the platform. This initialization structure is passed to the ADRV910x API initialization functions during the initialization phase. When programming is completed, the system is fully calibrated. With a few additional API calls, the device is ready to operate.

SYSTEM INITIALIZATION AND SHUTDOWN

The TES also can generate MATLAB, Python (.py), C# or C99 code, which includes all high-level API initialization calls that run in ADRV910x evaluation system. Store these automatically generated codes in a location of choice for future use.

API INITIALIZATION SEQUENCE

As previously mentioned, the initialization sequence consists of a serial of API calls intermixed with user-defined function calls specific to the hardware platform. The API functions perform all the necessary tasks for device configuration, calibration, and control. [Figure 13](#) shows the state machine of the device from power-up to RF enabled. Note: Moving back to the 'STANDBY' state is not currently supported without a device reset.

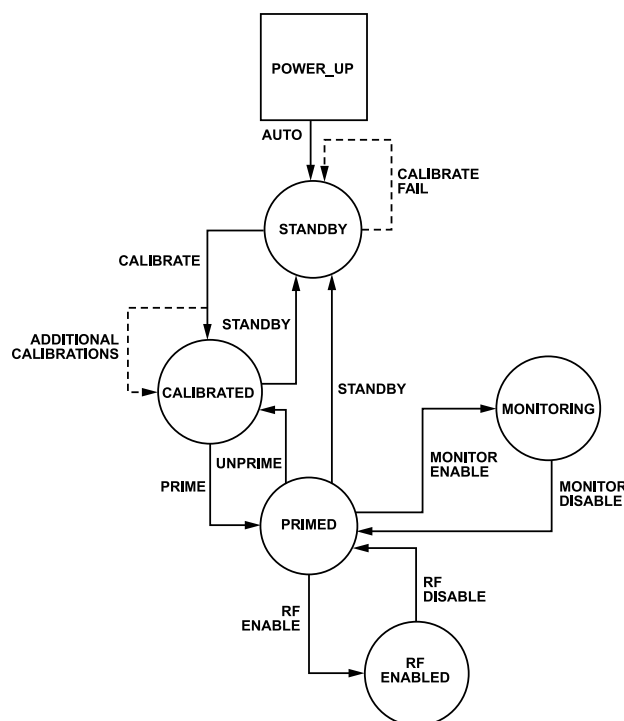


Figure 13. Device State Machine

As shown in [Figure 13](#), after power-up, the device automatically enters the standby state. Then, the device initialization and calibration begin. If it is successful, the device moves to the calibrated state, or else, it remains in the standby state. Note that when TES completes programming, the device is in the calibrated state. After the device is calibrated, move it to the primed state, which indicates the device is ready for operation.

Then, through SPI or PIN mode, move the device to the RF-enabled state by enabling the transmit/receive channels so transmission and reception can start. In TES, after programming, playing the receiver or transmitter moves the device from the calibrated state to the primed state and then to the RF-enabled state.

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

Analog Initialization

Analog initialization API `adi_adrv910x_InitAnalog()` is the very first API call to configure the device after all dependent data structures are initialized. It mainly sets the controller bias, enables the reference clock, and validates the profile settings.

Resource Loading

After analog initialization, a set of APIs is used to load required resources such as stream image, PS1, PS2 ARM images, programmable FIR (PFIR) coefficients, and so on. It also enables the internal microprocessor and initializes the digital clocks.

[Figure 14](#) shows the major APIs and their functionalities.

SYSTEM INITIALIZATION AND SHUTDOWN

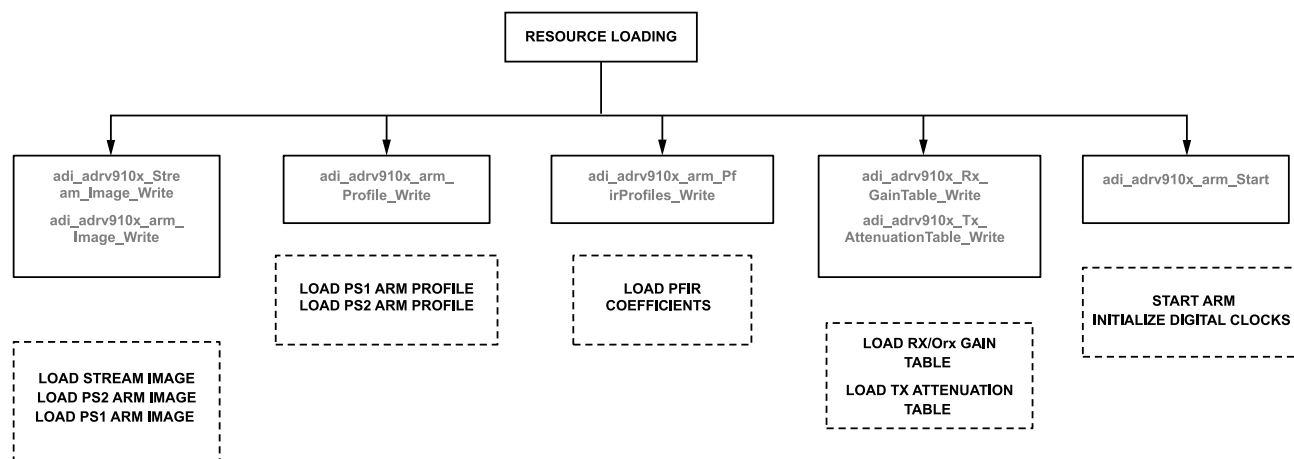


Figure 14. Load Resources and Digital Initialization

There is several SPI write modes for ARM image write, the [Multibyte Data Transfer](#) is recommend if the device boot up is critical, refer to [Figure 17](#) for more details.

Radio Initialization

After resource loading and starting the ARM, the next step is radio initialization through a set of API calls, which is used to load any radio configuration data not passed by profile before performing initial calibrations, such as GPIO configuration, internal LO PLL loop filter configuration, carrier frequencies, etc.

Initial Calibrations

The next step in initialization is to perform initial calibrations through API call `adi_adrv910x_cals_InitCals_Run()` based on calibration mask. To understand calibration mask, see the [Transmitter/Receiver Signal Chain Calibrations](#) section. When initial calibrations are correctly performed, the channel state transitions from the standby to calibrated state.

Prime and RF Enabled

The last step in initialization is to move the device from the calibrated to primed state through API call `adi_adrv910x_Radio_Channels_Prime()`. The Primed state indicates the system is ready for operation when the transmit and receive channels are enabled.

Note that after the channel is primed, to start the normal transmit or reception activities, further transition it from the primed to RF-enabled state with a set of API calls. There are two modes to enable channels: the PIN and SPI modes. Call API `adi_adrv910x_Radio_ChannelEnable_Mode_Set()` to set the PIN mode or SPI mode,

PIN Mode

Toggle the relative enable pins to transition the channel to the RF_Enabled state, for example, `RX1_EN/TX_EN` pins for RX1/TX WB channel, or `RX2_EN/TX_EN` for RX2/TX NB channel.

SPI Mode

Call `adi_adrv910x_Radio_Channel_EnableRf()` to transition the channel to the RF-enabled state.

INITIALIZATION EXAMPLE CODE

After configuring the desired profile parameters in TES, Users can use TES "Sample Code" function to generate the ADRV910x Initialization example code in which contains the profile e data structures, the sequential sets of API calls for all the initialization sequence that described in [API Initialization Sequence](#).

The [Figure 15](#) shows the TES "Sample Code" function to generate the Initialization example code.

Then follow the steps described in [Customizing the System Architecture and File Structure](#), users just need some minor edits on the generated "Example code" to deploy the full ADRV910x initialization sequence on the target platform.

SYSTEM INITIALIZATION AND SHUTDOWN

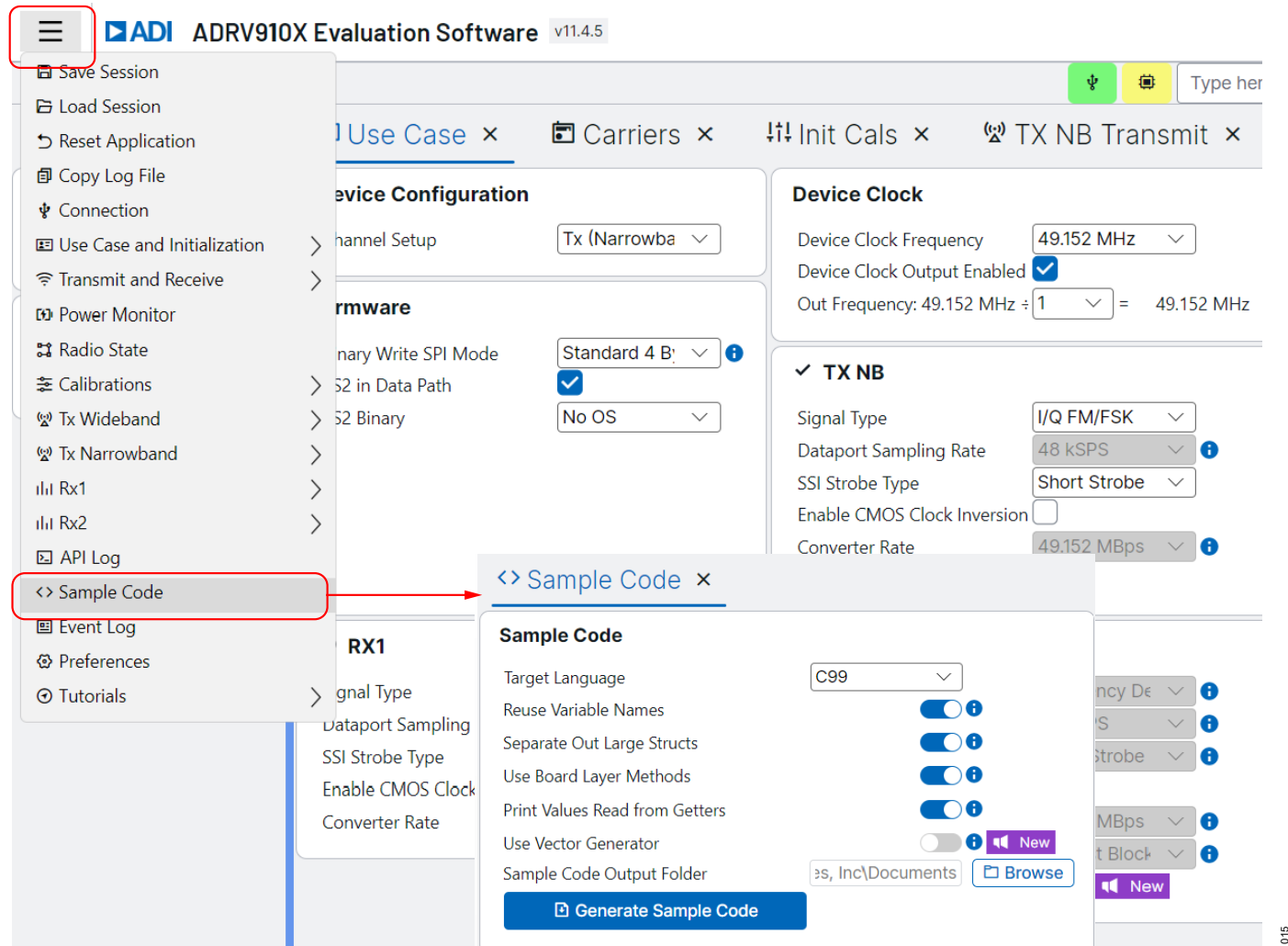


Figure 15. Use TES “Sample Code” to Generate Initialization Example Code

SHUTDOWN SEQUENCE

After completing all the operations, call API `adi_adrv910x_Shutdown()` through the TES to safely shut down the ADRV910x device. It performs hardware reset to reset the ADRV910x device into a safe state for shutdown or re-initialization.

Note:

More PS2 related system initialization functions will be added in future update.

SERIAL PERIPHERAL INTERFACE (SPI)

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

SPI CONFIGURATION

Configure the SPI settings for the device with different SPI controller configurations by configuring the member values of the `adi_adrv910x_SpiSettings_t` data structure. The `adi_adrv910x_SpiSettings_t` data structure contains:

```
typedef struct adi_adrv910x_SpiSettings
{
    uint8_t msbFirst;
    uint8_t enSpiStreaming;
    uint8_t autoIncAddrUp;
    uint8_t fourWireMode;
    adi_adrv910x_CmosPadDrvStr_e cmosPadDrvStrength;
} adi_adrv910x_SpiSettings_t;
```

Figure 16. SPI Settings Data Structure

The parameters for this structure are listed in [Table 4](#).

Table 4. SPI Settings Data Structure

Structure Member	Value	Function	Default
MSBFirst	0x00	Least significant bit first.	0x01
	0x01	Most significant bit first.	
enSpiStreaming	0x00	Disable SW feature. Section Multibyte Data Transfer (SPI Streaming) describes this mode of operation.	0x00
	0x01	Enable SW feature to improve SPI throughput. Section Multibyte Data Transfer (SPI Streaming) describes this mode of operation.	
autoIncAddrUp	0x00	Auto-decrement. Functionality intended to be used with SPI Streaming. Sets address auto-decrement -> next addr = addr -1	01
	0x01	Auto-increment. Functionality intended to be used with SPI Streaming. Sets address auto-increment -> next addr = addr +1	
fourWireMode	0x00	SPI hardware implementation. Use 3-wire SPI (SDIO pin is bidirectional). shows example of SPI 3-wire mode of operation.NOTE: ADI's FPGA platform always uses 4-wire mode.	0x01
	0x01	SPI hardware implementation. Use 4-wire SPI. and show examples of SPI 4-wire mode of operation.NOTE: Default mode for ADI's FPGA platform is 4-wire mode.	
cmosPadDrvStrength	0x00	5 pF load @ 75 MHz	0x01
	0x01	100 pF load @ 20 MHz	

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

For more details, refer to `ADRV910x_API` doxygen file provided in ADRV910x SDK package.

SPI BUS SIGNALS

The SPI bus consists of the following signals:

- ▶ SCLK
- ▶ CSB
- ▶ SDIO and SDO

SCLK

SCLK is the serial interface reference clock driven by the baseband processor (uses the `SPI_CLK` pin). It is only active while CSB is low. The maximum SCLK frequency is 46 MHz.

SERIAL PERIPHERAL INTERFACE (SPI)

CSB

CSB is the active-low chip select that functions as the bus enable signal driven from the baseband processor to the device (uses the SPI_EN pin). CSB is driven low before the first SCLK rising edge and is normally driven high again after the last SCLK falling edge. The device ignores the clock and data signals while CSB is high. CSB also frames communication to and from the device and returns the SPI interface 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 returned to the ready state. Any complete data byte transfers prior to CSB deserting are valid, but all subsequent transfers in a continuous SPI transaction are aborted.

SDIO and SDO

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

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

The SDO and SDIO pins transition to a high-impedance state when the CSB input is high. The device does not provide any weak pull-ups or pull-downs 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 time, an external weak pull-up/down (10 kΩ value) should be added on the PCB.

SPI DATA TRANSFER PROTOCOL

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

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

Phase 1 Instruction Format

The 16-bit control field contains the following information:

Table 5. SPI Control Field

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

R/Wb—Bit 15 of the instruction word determines whether a read or write data transfer occurs after the instruction byte write. Logic high indicates a read operation; logic zero indicates a write operation.

D14:D0—Bits A<14:0> specify the starting byte address for the data transfer during Phase 2 of the I/O operation.

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

Single-Byte Data Transfer

When enSpiStreaming = 0, 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 next LSB to MSB. The next bit signifies if the operation is read (set) or write (clear). If the operation is a write, the baseband processor transmits the

SERIAL PERIPHERAL INTERFACE (SPI)

next 8 bits LSB to MSB. If the operation is a read, the device transmits the next 8 bits LSB to MSB. Once the final bit is transferred, the data lines return to their idle state and the CSB line must be driven high to end the communication session.

In MSB mode, the first bit transmitted is the R/W bit that determines if the operation is a read (set) or write (clear). The MSB of the address is the next bit transmitted from the baseband processor, followed by the remaining 14 bits in order from next MSB to LSB. If the operation is a write, the baseband processor transmits the next 8 bits MSB to LSB. If the operation is a read, the device transmits the next 8 bits MSB to LSB. Once 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 multi-byte 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 `autoIncAddrUp = 0`, the address decrements from the starting address for each subsequent data transfer. If address 0x000 is reached, the next address that is accessed is the last register location defined in the register map. The register address 0x000 is used to setup SPI interface as well as functionality to soft reset the device. Uncontrolled data written to the register address 0x000 can cause SPI misconfigurations or can reset the device. It is strongly recommended that any data transfer using Multi-Byte Data feature to be controlled so that 0x000 is only written once at startup.

For multi-byte 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 read (set) or write (clear). If the operation is a write, the baseband processor transmits the next 8 bits LSB to MSB. After the MSB is received, the address increments or decrements based on the `autoIncAddrUp` parameter. The baseband processor, then continues to transfer data in 8-bit words, LSB to MSB, until the operation is terminated by CSB being driven high. If the operation is a read, the device transmits the next 8 bits LSB to MSB. It then changes the address and continues to transfer data in 8-bit words, LSB to MSB, until the operation is terminated by CSB being driven high.

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

Using multi-byte data transfer mode provides limited benefit because most ADRV910x registers in the device are not consecutive. But it's helpful to accelerate the Stream/ARM binary loading time. It is up to the user to determine if multi-byte data transfer (streaming) mode will be used the binary loading.

Example: LSB-First Multibyte Transfer, Auto-Incrementing Address

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

- ▶ Make sure that `fourWireMode = 1` – the device is configured to work with 4-wire interface.
- ▶ Make sure that `MSBFirst = 0` - SPI operates in LSB first mode.
- ▶ Make sure that `autoIncAddrUp = 1` - the address pointer automatically increments.
- ▶ Make sure that `enSpiStreaming = 1` - a multi-byte 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 0x02A as the starting address.
- ▶ Use the next 32 clock cycles to send the data to be written to the registers, LSB to MSB for each 8-bit word.
- ▶ Make sure the CSB line is driven high after the last bit has been sent to 0x02D to end the data transfer.

Example: MSB-First Multibyte Transfer, Autodecrementing Address

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

- ▶ Make sure that `fourWireMode = 1` – the device is configured to work with 4-wire interface.
- ▶ Make sure that `MSBFirst = 1` - SPI operates in MSB first mode.
- ▶ Make sure that `autoIncAddrUp = 0` - the address pointer automatically decrements.
- ▶ Make sure that `enSpiStreaming = 1` - a multi-byte data transfer is allowed.

SERIAL PERIPHERAL INTERFACE (SPI)

- Force the CSB line low and keep it low until the last byte is transferred.
- Send the instruction word 0_000 0000 0010 1010 (the first 0 indicates a write operation) to select 0x02A as the starting address.
- Use the next 32 clock cycles to send the data to be written to the registers, MSB to LSB for each 8-bit word.
- Make sure the CSB line is driven high after the last bit has been sent to 0x027 to end the data transfer.

ARM BINARY WRITE SPI MODE

The ADRV910x PS1 and PS2 ARM binary files need to be written to the ADRV910x memory before the device is initialized. There is a configurable “spiWriteMode” available in API “adi_adrv910x_arm_Image_Write”. The “STANDARD_BYTES_4” is corresponding to mode that previously discussed in [Single-Byte Data Transfer](#) section, and the “STREAMING_BYTES_1” is corresponding to the mode in [Multibyte Data Transfer](#) section. Refer the API doxgen file for information about the other support modes.

The SPI streaming mode has obviously higher efficient for the binary files writing, [Table 6](#) shows the theoretical binary loading time example for SPI standard mode and streaming mode, where assume the binary file size is 1M bytes, SPI clock rate is 50MHz. Note, the actual measurement time will be longer depending on the SPI driver implementation on user’s platform, the shorter SPI operation transition time, the closer to the theoretical time.

Table 6. Theoretical Binary Loading Time for Standard and Streaming SPI Write Mode

spiWriteMode	Binary Loading Time	Payload Efficiency	Notes
STANDARD_BYTES_4	503.3 ms	33.3%	1 byte payload for one CSB active duration
STREAMING_BYTES_1	169.1 ms	99.2%	N byte payload for one CSB active duration, N is dictated by the platform SPI buffer size, N = 256 in this example

Therefore, streaming mode is recommended if there is a stringent ADRV910x bootup time requirement.

[Figure 17](#) shows the configurable options available in the ADRV910x transceiver evaluation software (TES) that users can used to configure the SPI mode properly based on the custom platform driver support.

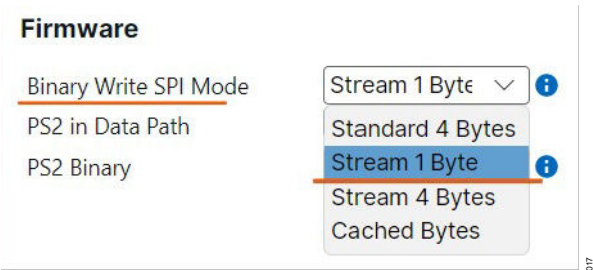


Figure 17. ARM Binary Write SPI Modes

TIMING DIAGRAMS

The diagrams in [Figure 18](#) and [Figure 19](#) illustrate the SPI bus waveforms for a single-register write operation and a single-register read operation, respectively. In the first figure, the value 0x55 is written to register 0x00A. In the second value, register 0x00A is read, and the value returned by the device is 0x55. If the same operations were performed with a 3-wire bus, the SDO line in [Figure 18](#) would be eliminated, and the SDIO and SDO lines in [Figure 11](#) would be 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.

Users should be advised, register 0x00A is not user accessible, the SPI write and read operation in [Figure 18](#) and [Figure 19](#) is only for the SPI timing diagram demonstration purpose.

SERIAL PERIPHERAL INTERFACE (SPI)

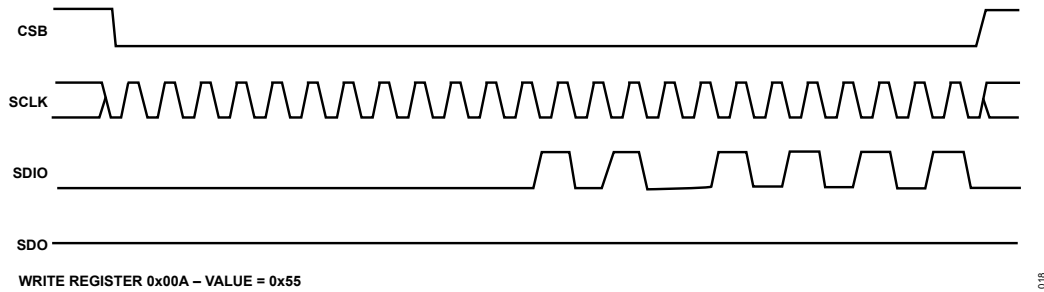


Figure 18. Nominal Timing Diagram, SPI Write Operation

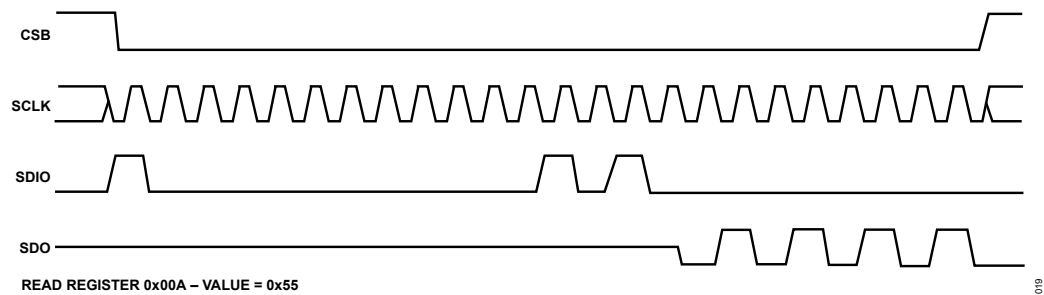


Figure 19. Nominal Timing Diagram, SPI Read Operation

Table 7 lists the timing specifications for the SPI bus. The relationship between these parameters is shown in Figure 20. This diagram shows a 3-wire SPI bus timing diagram with the device returning a value of 0xD4 from a test register and timing parameters marked. Note that this is a single read operation, so the bus-ready parameter after the data is driven from the device (t_{HZS}) is not shown in the diagram.

Table 7. SPI Timing Constraint Values

Parameter	Min	Typical	Max	Description
t_{CP}	28 ns			SCLK period, 3 wire mode
	22ns			SCLK period, 4 wire mode
t_{MP}	10 ns			SCLK pulse width
t_{SC}	0 ns			CSB setup time to first SCLK rising edge
t_{HC}	0 ns			Last SCLK falling edge to CSB hold
t_S	0.5 ns			SDIO data input setup time to SCLK
t_H	2 ns			SDIO data input hold time to SCLK
t_{CO}	4.5 ns		7 ns	SCLK falling edge to output data delay (3-wire mode)
	4.5 ns		7 ns	SCLK falling edge to output data delay (4-wire mode)
t_{HZM}	t_H		$t_{CO(max)}$	Bus turnaround time after Baseband Processor drives the last address bit
t_{HZS}	0 ns		$t_{CO(max)}$	Bus turnaround time after device drives the last data bit

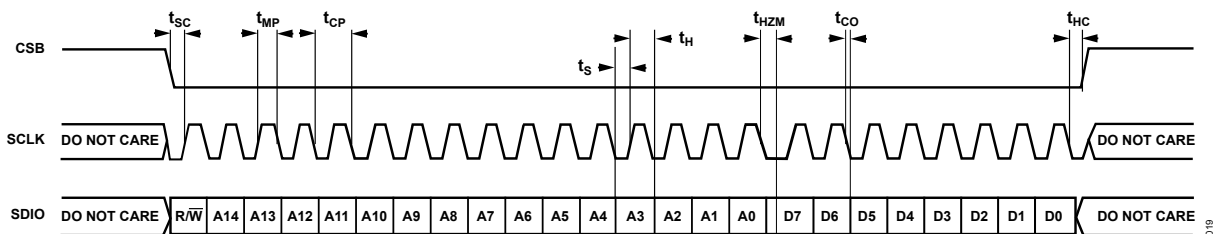


Figure 20. 3-Wire SPI Timing with Parameter Labels, SPI Read

SERIAL PERIPHERAL INTERFACE (SPI)

SPI TEST

The ADRV910x has a scratch SPI register 0x009 for the SPI read/write validation. The following procedures quickly check the SPI function before the relative baseband processor drivers are ready:

- ▶ Power on the ADRV910x device properly.
- ▶ Toggle the reset pin to reset the ADRV910x.
- ▶ Write register 0x0 with value 0x3C to set the ADRV910x SPI to the 4-wire mode, or with value 0x24 to set the ADRV910x SPI to the 3-wire mode.
- ▶ Write any value to scratch register 0x009, then read register 0x009 to validate if the read value is the same as the write value.

Use the oscilloscope to probe the SPI bus signal and check if the SPI controller follows the timing diagrams in [Figure 18](#) and [Figure 19](#) when the above SPI validation cannot pass.

Also use the API `adi_adrv910x_spi_Verify ()` to validate the SPI after the `adi_adrv910x_spi_Configure ()` is set if the BBIC has the available drivers.

SYNCHRONOUS SERIAL INTERFACE (SSI)

GENERAL DESCRIPTION

This document defines the synchronous serial interface (SSI) which transfers data between the ADRV910x and the baseband processor. ADRV910x SSI consists of up to two receive channels and one transmit channel. The channels are independent and can be configured either as CMOS signals (CSSI) for applications that have narrow RF signal bandwidths and low data rate, or as LVDS signals (LSSI) for applications that require high speed, low noise, and longer distance data transfer.

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

- ▶ One lane data mode, I/Q data, or other format data are serialized onto one single lane.
- ▶ Four lanes data mode, which is valid only when the ADRV910x transmits or receives I/Q samples, and the I/Q samples are 16 bits wide. In four-lane data mode, each sample is split into 8 bit blocks of data and sent over one data lane.

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

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

ADRV910x SSI has various and flexible work modes to support many kinds of system scenarios. Users can choose their appropriate work modes according to the interface sample/symbol rate and bit width. [Table 8](#) lists the ADRV910x SSI work modes and the maximum support I/Q sample rate.

Table 8. ADRV910x SSI Work Modes

SSI Modes	Data Lanes Per Channel	Serialization factor Per data lane	Max data lane rate (MHz)	Max Clock rate (MHz)	Max Sample Rate for I/Q (MHz)	Data Type
CSSI 1-lane	1	64/32	80	80	1.25/2.5	SDR
CSSI 1-lane	1	64/32	160	80	2.5/5	DDR
CSSI 1-lane ¹	1	16/8/2	80-SDR/160-DDR	80	Not Applicable	SDR/DDR
CSSI 4-lane	4	8	80	80	10	SDR
CSSI 4-lane	4	8	160	80	20	DDR
LSSI 1-lane	1	32	983.04	491.52	30.72	DDR
LSSI 2-lane	2	16	983.04	491.52	61.44	DDR
LSSI 2-lane ²	2	12	737.28	368.64	61.44	DDR

¹ ADRV910X data port transmit/receive data symbols, refer [CSSI Data Symbols Transmit and Receive](#).

² For User's LVDS data lane rate limitation applications, RX samples are rounded from 16 bits to 12 bits. Tx Sample are extended from 12bits to 16bits.

The following sections will explain the details of the signals that make up the SSI and their properties when configured for each mode.

ELECTRICAL SPECIFICATION

ADRV910X SSI can operate in standard single ended CMOS compatible mode or Low-voltage Differential Signal (LVDS) compatible mode, CMOS SSI and LVDS SSI share the IO pads of ADRV910X. [Figure 21](#) shows the TX/RX channels with their corresponding inputs and outputs in the CMOS and LVDS modes.

SYNCHRONOUS SERIAL INTERFACE (SSI)

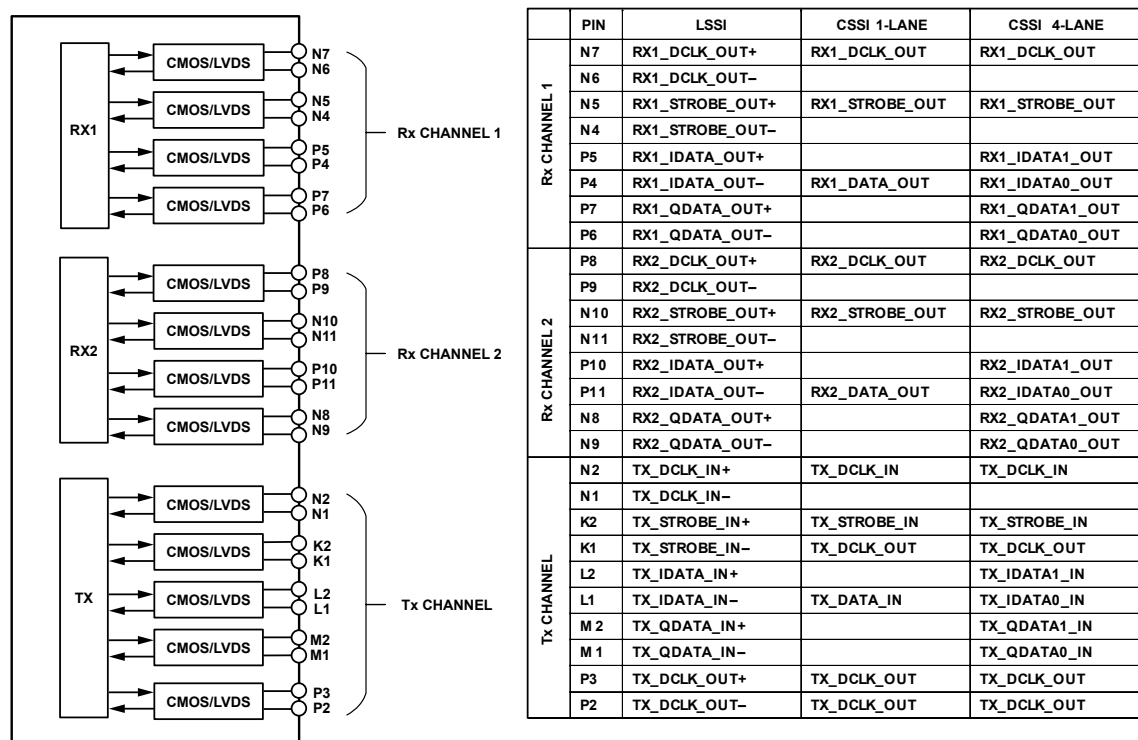


Figure 21. ADRV910x SSI I/Os Mapping

121

Table 9 shows the CMOS SSI electrical specifications. For optimal performance, keep the load capacitance driven by the CMOS outputs to a minimum. The CMOS output drive strength can be increased to compensate for load capacitance larger than 10 pF to increase the edge rate of the output signal during the transition period. The maximum load capacitance is 30 pF at an 80 MHz clock data rate.

In LVDS mode, a 100 Ω differential termination resistor is required for each LVDS pair, which must be located as close to the LVDS receiver. ADRV910X LVDS in circuit has optional internal 100 Ω termination resistor which is enabled for LSSI by default, but ADRV910X LVDS output circuit does not have internal termination resistors, users should develop appropriate LVDS termination resistors in LVDS receiver. The default LVDS out circuit produces 350mV peak at 1.2V common mode level, output swing level can be increased to 450mV for longer trace. LVDS SSI electrical specification is shown in Table 10.

It is recommended to closely match the trace lengths of SSI Clock, Strobe, Data signals into each Transmit or Receive channel. The ADRV910X SSI has configurable delay cells on the LVDS/CMOS input and output circuits which allow the user to make small adjustments to the phase relationship between strobe/data and clock. The adjustable phase delay cell increments in approximately 90 ps per step in LVDS mode, and 170 ps per step in CMOS mode, up to a maximum of seven steps.

Table 9. CSSI Electrical Specification

Symbol	Parameter	Min	Typ	Max	Unit
VDIG_1P8	Interface power supply voltage	1.71	1.8	1.89	V
V _{IH}	Input voltage high	VDIG_1P8 × 0.65		VDIG_1P8 + 0.18	V
V _{IL}	Input voltage low	0		VDIG_1P8 × 0.35	V
V _{OH}	Output voltage high	VDIG_1P8 - 0.45		VDIG_1P8	V
V _{OL}	Output voltage low	0		0.45	V
f _{CLK}	Clock frequency			80	MHz
C _L @ 80MHz	Load capacitance supported for an 80MHz clock waveform		10	30	pF

Table 10. LSSI Electrical Specification

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
VDIG_1P8	Interface power supply voltage		1.71	1.8	1.89	V
V _I	Input voltage range		825		1675	mV

SYNCHRONOUS SERIAL INTERFACE (SSI)

Table 10. LSSI Electrical Specification (Continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
	Input Common Mode Voltage		925	1200	1575	mV
V_{IDTH}	Input differential threshold		-100		+100	mV
R_{IN}	Receiver differential input impedance			100		Ω
V_{OH}	Output voltage high	$R_{LOAD} = 100\Omega \pm 1\%$			1390	mV
V_{OL}	Output voltage low	$R_{LOAD} = 100\Omega \pm 1\%$	1000			mV
$ V_{OD} $	Output differential voltage	$R_{LOAD} = 100\Omega \pm 1\%$		360		mV
V_{OS}	Output offset voltage	$R_{LOAD} = 100\Omega \pm 1\%$	1150	1200	1250	mV
R_O	Output impedance, single ended		80	100	120	Ω
$I_{SA} I_{SB}$	Output current	Driver shorted to ground			17	mA
I_{SAB}	Output current	Drivers shorted together			4.1	mA
	Clock signal duty cycle	500MHz	45	50	55	%
t_R, t_F	Output Rise/Fall Time	300mVp swing		0.371		ns

CMOS SYNCHRONOUS SERIAL INTERFACE (CSSI)

One-lane Mode CSSI

Receive CSSI

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

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

The I and Q samples are serialized out starting with configurable I or Q first and MSB or LSB first, Figure 22 shows the Receive CSSI (Rx1 and Rx2) for a 16-bit I/Q data sample with I sample and MSB first configuration.

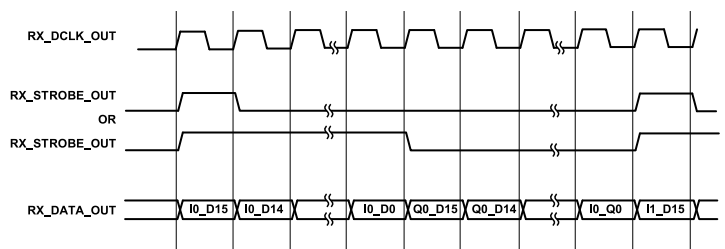


Figure 22. Receive CSSI Timing for 16-bit I/Q data sample (I & MSB first)

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

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

The Transmit CSSI

The one-lane mode transmit CSSI of the Tx channel is a 4-wire digital interface consisting of:

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

SYNCHRONOUS SERIAL INTERFACE (SSI)

The I and Q samples can be deserialized, starting with configurable I or Q first and MSB or LSB first. Figure 23 shows the Transmit CSSI interface for a 16-bit I/Q data sample with I sample and MSB first configuration.

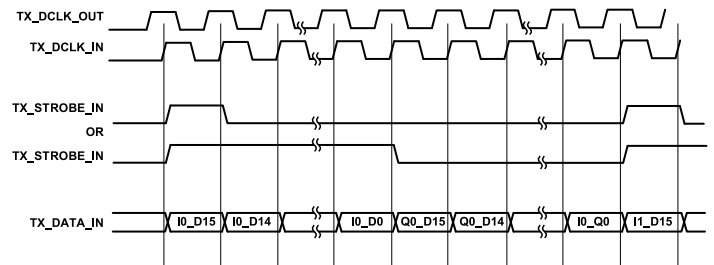


Figure 23. Transmit CSSI Timing for 16-bit I/Q data sample (I & MSB first)

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

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

CSSI Data Symbols Transmit and Receive

The previous sections described data transfer with I/Q format with 16bit width. When the ADRV910x internal modulation/demodulation is enabled, the data transfer between ADRV910X and baseband processor could be 2bits or 16bits I only data (denoted as symbol to differentiate with I/Q complex samples). In a symbol format mode, raw data are transferred through this interface using different data size. The CSSI interface supports 3 additional data formats:

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

Two bits of data can be transferred over a CSSI with an 8 bits data format with 6 dummy bits. The clock and strobe behaviour are similar to the I/Q format described in previous sections.

Figure 24 shows the Receive CSSI (Rx) for 2-bit data symbols.

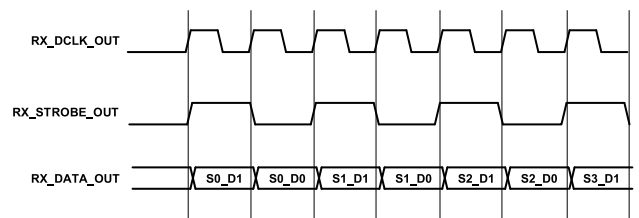


Figure 24. Receive CSSI timing for 2-bit symbols (MSB first)

Figure 25 shows the Transmit CSSI (Tx) for 2-bit data symbols.

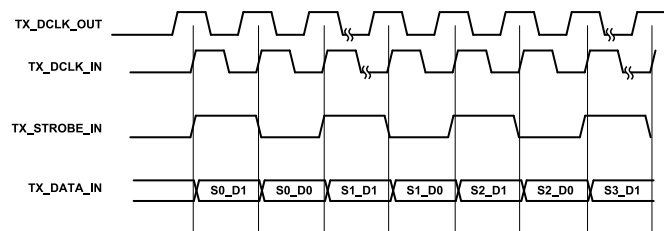


Figure 25. Transmit CSSI timing for 2-bit symbols (MSB first)

Figure 26 shows the Receive CSSI (Rx) for 8-bit data symbols.

SYNCHRONOUS SERIAL INTERFACE (SSI)

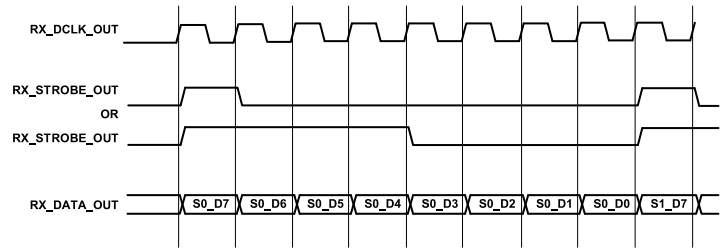


Figure 26. Receive CSSI timing for 8-bit symbols (MSB first)

Figure 27 shows the Transmit CSSI (Tx) for 8-bit data symbols.

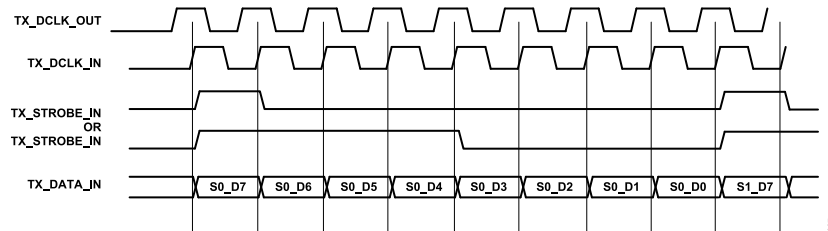


Figure 27. Transmit CSSI timing for 8-bit symbols (MSB first)

Figure 28 shows the Receive CSSI (Rx) for 16-bit data symbols.

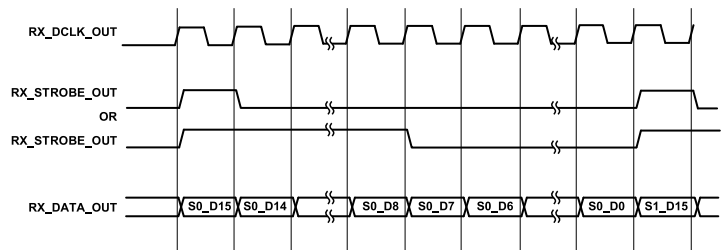


Figure 28. Receive CSSI timing for 16-bit symbols (MSB first)

Figure 29 shows the Transmit CSSI (Tx) for 16-bit data symbols.

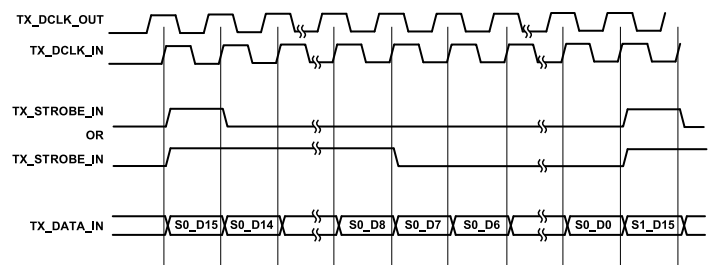


Figure 29. Transmit CSSI timing for 16-bit symbols (MSB first)

Receive CSSI With 2x, 4x and 8x Data Clock Rates

ADRV910X receive CSSI supports the 2, 4 or 8 times of the data clock rate for some applications. With these modes, the SSI clock rate could be 2x, 4x or 8x faster than the sample rate of data path output.

Figure 30, Figure 31 and Figure 32 illustrate the Receive CSSI interface (Rx1 and Rx2) for 16-bit I/Q data sample with 2x, 4x and 8x clock rates. The strobe pulse validates the start of the 32bits I and Q samples, the remaining data bits (repeated samples or dummy data) are ignored.

SYNCHRONOUS SERIAL INTERFACE (SSI)

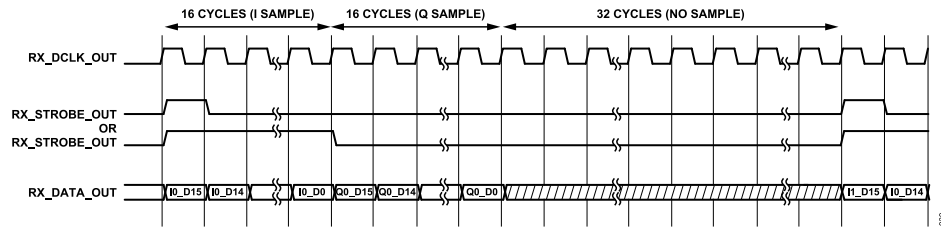


Figure 30. Receive CSSI timing with 2x data clock rate for 16-bit I/Q data sample (MSB first)

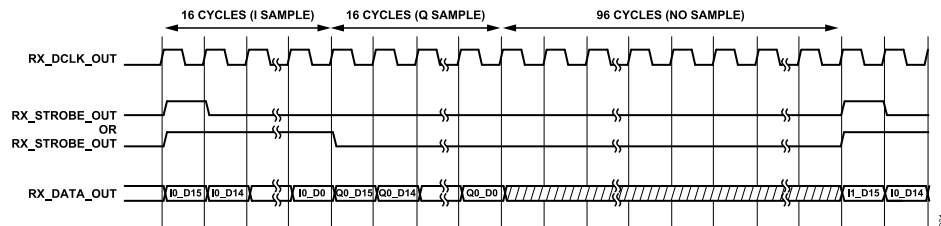


Figure 31. Receive CSSI timing with 4x data clock rate for 16-bit I/Q data sample (MSB first)

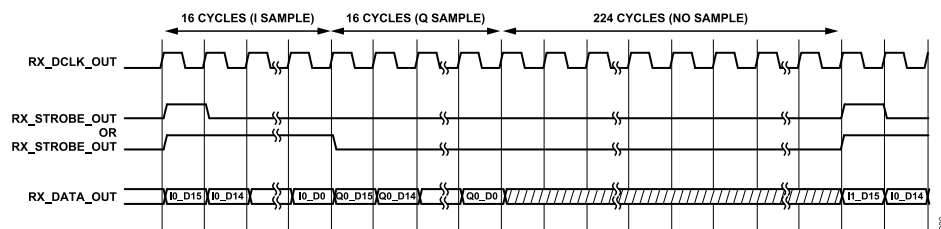


Figure 32. Receive CSSI timing with 8x data clock rate for 16-bit I/Q data sample (MSB first)

Figure 33, Figure 34 and Figure 35 illustrate the Receive CSSI (Rx1 and Rx2) in frequency deviation mode with 16-bit data symbol with 2x, 4x and 8x clock rates. The strobe pulse validates the start of the 16bits data symbol, the remaining data bits are ignored.

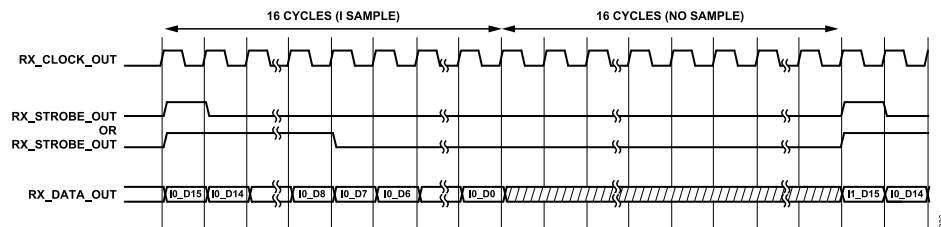


Figure 33. Receive CSSI timing with 2x data clock rate for 16-bit data symbol (MSB first)

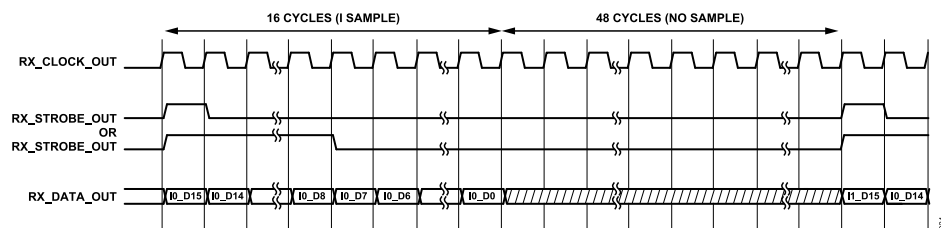


Figure 34. CSSI Receive timing with 4x data clock rate for 16-bit data symbol (MSB first)

SYNCHRONOUS SERIAL INTERFACE (SSI)

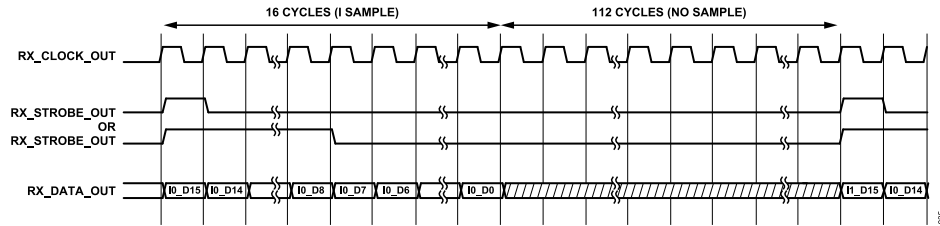


Figure 35. Receive CSSI timing with 8x data clock rate for 16-bit data symbol (MSB first)

Four-lane Mode CSSI

The 4-lane mode receive CSSI of each channel (Rx1 and Rx2) are a 6-wire digital interface consisting of:

- ▶ RX_DCLK_OUT: output clock synchronous data and strobe output signals.
- ▶ RX_STROBE_OUT: output signal indicating the first bit of the serial data sample.
- ▶ RX_IDATA0_OUT: output serial data stream of I sample low byte.
- ▶ RX_IDATA1_OUT: output serial data stream of I sample high byte.
- ▶ RX_QDATA2_OUT: output serial data stream of Q sample low byte.
- ▶ RX_QDATA3_OUT: output serial data stream of Q sample high byte.

Figure 36 shows the Receive CSSI (Rx1 and Rx2) for a 4-lane format with MSB first configuration.

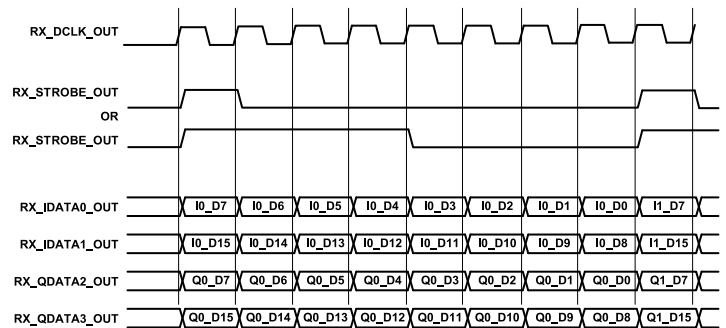


Figure 36. Four-lane Mode Receive CSSI Timing for 16-bit I/Q Data Sample (MSB First)

The four-lane mode CSSI transmit interface of Tx channel is a 7-wire digital interface consisting of:

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

Figure 37 shows the Transmit CSSI for a 4-lane format with MSB first configuration.

SYNCHRONOUS SERIAL INTERFACE (SSI)

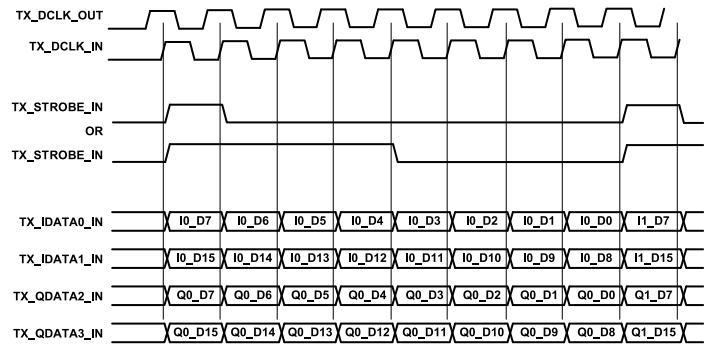


Figure 37. Four-lane Mode Transmit CSSI Timing for 16-bit I/Q Data Sample (MSB First)

Transmit and Receive CSSI Using DDR Clock

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

Figure 38 shows the Rx CMOS SSI with DDR clock in relation with strobe/data. Each edge of the clock (positive and negative) corresponds to a data sample. The RX DDR Clock can be generated in phase with the data/strobe or delayed quarter cycle of the clock period, the optional delayed clock eases the timing interface of the baseband processor to meet setup/hold on baseband processor.

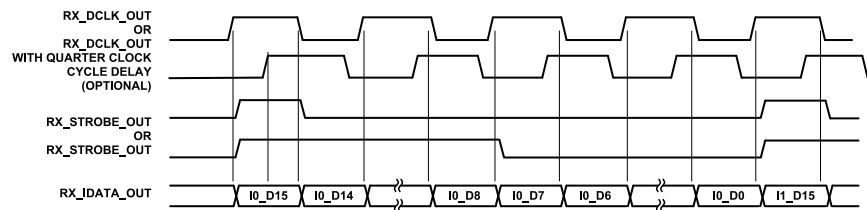


Figure 38. Rx CSSI DDR Clock Relation with Strobe/Data

Figure 39 shows the Tx CMOS SSI with DDR clock in relation with strobe/data, with respect to ADRV910X. Each edge of the clock (positive and negative) will sample the corresponds to a strobe/data sample based on the interface setup/hold timing.

When baseband processor drives out the Tx SSI clock, strobe and data to ADRV910X, the output DDR clock can either be in-phase with the strobe/data, or delayed quarter cycle of the clock period. Regardless of the choice, the timing relation between Tx DDR clock and strobe/data must meet the ADRV910X setup and hold timing specification.

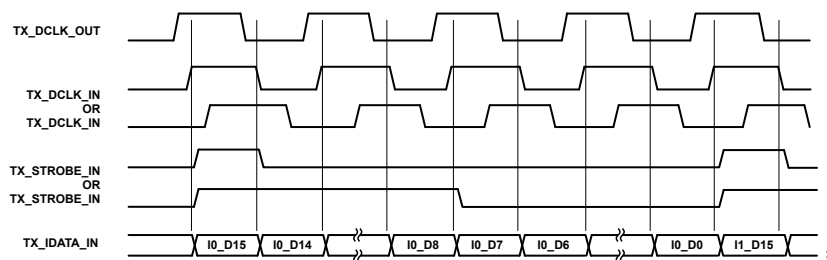


Figure 39. Tx CSSI DDR Clock Relation with Strobe/Data

Figure 40 and Figure 41 illustrate the timing diagram example for four-lane mode Receive, Transmit CSSI with DDR clock, 16-bit I/Q sample.

SYNCHRONOUS SERIAL INTERFACE (SSI)

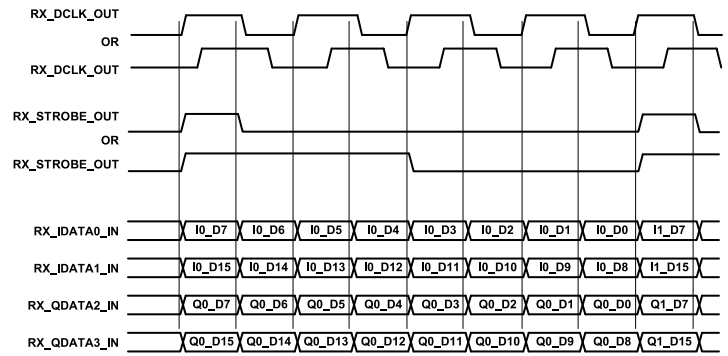


Figure 40. Four-lane Mode Receive CSSI DDR Timing for 16-bit I/Q Data Sample

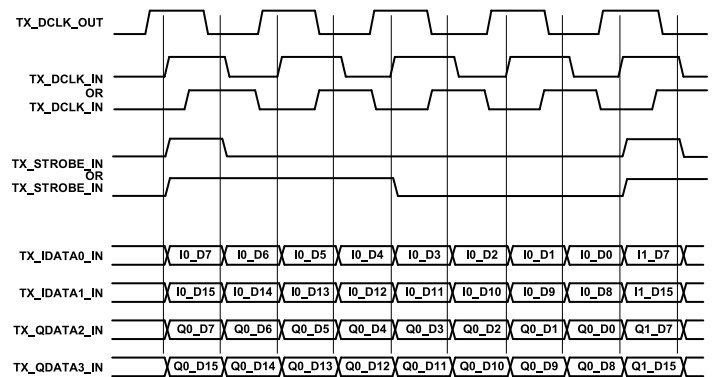


Figure 41. Four-lane Mode Transmit CSSI DDR Timing for 16-bit I/Q Data Sample

LVDS SYNCHRONOUS SERIAL INTERFACE (LSSI)

The Receive LSSI

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

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

Receive LSSI Interface With Separate Lanes for I and Q

Figure 42 shows the receive LSSI (Rx1 and Rx2) for a 16-bit I/Q data sample with MSB first configuration. Figure 43 shows the Receive LSSI for a 12-bit I/Q data sample.

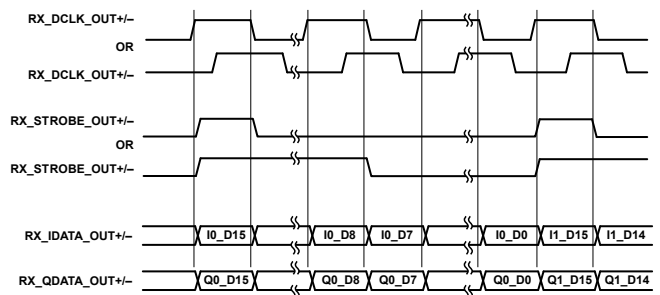


Figure 42. Receiver LSSI Timing for 16-bit I/Q data sample over 2 lanes (MSB First)

SYNCHRONOUS SERIAL INTERFACE (SSI)

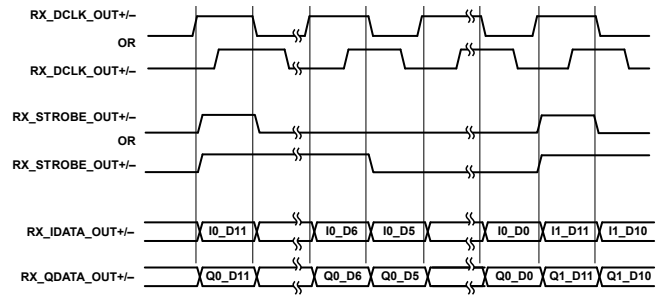


Figure 43. Receive LSSI Timing for 12-bit I/Q data sample over 2 lanes (MSB First)

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

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

In 12bit I/Q mode, 16bit samples from Rx data path will be cut to 12 bits for LSSI, a configurable option for user to choose the 12bit is from LSB or MSB of the 16bit sample data.

Receive LSSI With One Lane for I and Q

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

Figure 44 shows the one lane Receive LSSI (Rx1 and Rx2) for a 16-bit I/Q data sample with I sample and MSB first configuration.

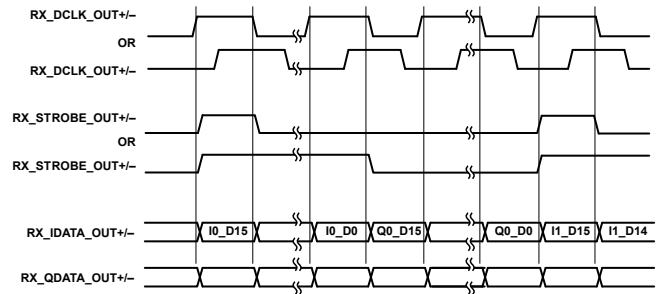


Figure 44. Receiver LSSI Timing for 16-bit I/Q data sample over one lane (I & MSB First)

Receive LSSI With 2×, 4×, and 8× Data Clock Rates

The ADRV910X receive LSSI supports the 2 times, 4 times, or 8 times of the data clock rate for some applications, which is similar with the Receiver CSSI mode (refer to the timing diagrams in [Receive CSSI With 2x, 4x and 8x Data Clock Rates](#)).

The Transmit LSSI

The Transmit LSSI is an 8-wire digital interface consisting of:

- TX_DCLK_IN (±): differential input clock synchronized to the data and strobe inputs.
- TX_STROBE_IN (±): differential input signal indicating the first bit of the serial data sample.
- TX_IDATA_IN (±): differential input serial I data stream.
- TX_QDATA_IN (±): differential input serial Q data stream.
- TX_DCLK_OUT (±): optional differential output reference clock provided to the baseband processor to generate all the above signals, the baseband processor can also use RX_DCLK_OUT as the reference clock when its clock rate is equal with Transmit SSI clock rate.

SYNCHRONOUS SERIAL INTERFACE (SSI)

Transmit LSSI With Separate Lanes for I and Q

Figure 45 shows the transmit LSSI for a 16-bit I/Q data sample with MSB first configuration.

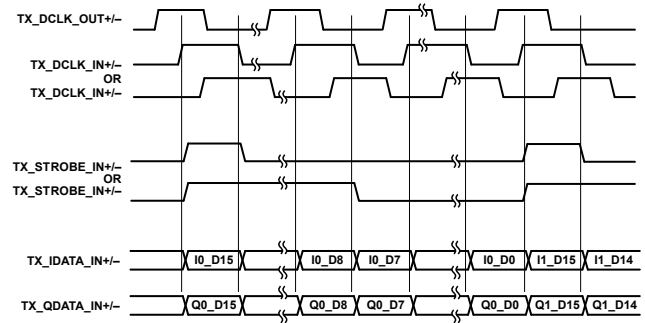


Figure 45. Transmit LSSI Timing for 16-bit I/Q Data Sample on Separate Lanes

Figure 46 shows the transmit LSSI interface for a 12-bit I/Q data sample with MSB first configuration.

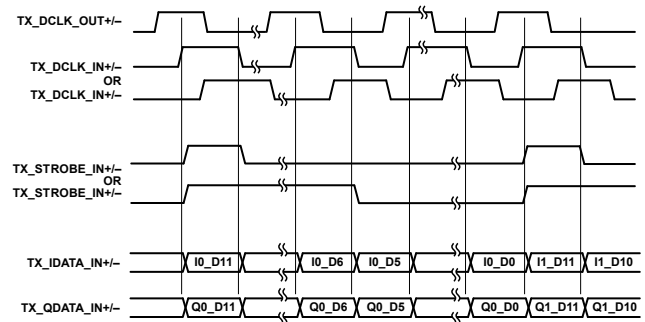


Figure 46. Transmit LSSI Timing for 12-bit I/Q Data Sample on Separate Lanes

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

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

In 12bit I/Q mode, 12 bits samples from LSSI will be extended to 16bit by padding 4 bits zero in LSB for the following Tx data path process.

Transmit LSSI Interface With One Lane for I and Q

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

Figure 47 shows the one lane LSSI interface for a 16-bit I/Q data sample with I sample and MSB first configuration.

SYNCHRONOUS SERIAL INTERFACE (SSI)

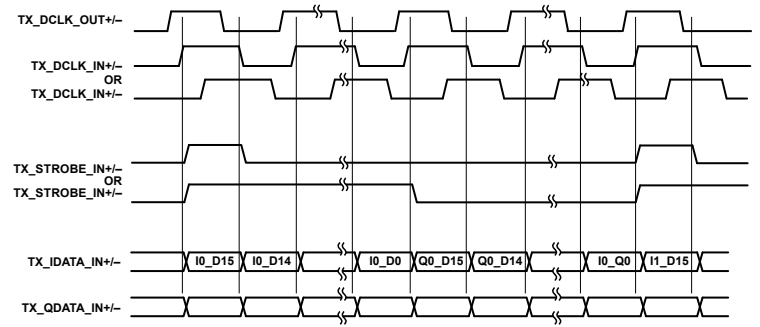


Figure 47. Transmit LSSI Timing for 16-bit I/Q Data Sample Sharing One Lane

ENHANCED RX SSI MODE

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

In the 22bit data samples case, the 32bit interface data bus has the following fields:

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

which produce the following Interface data with 32bit data format for the CMOS and LVDS SSI:

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

In the 15bit data samples case, the 16bit interface data bus has the following fields:

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

which produce the following interface data with 16 bit data format for the CMOS and LVDS SSI:

- ▶ LVDS 16bit: 2 lanes (I & Q) of 16 bit each
 - ▶ LSSI_DATA_I/Q [15:0] = {RxDataPath I/Q rounded[14:0], b0, Gain_Change, Gain [7:0]}
- ▶ CMOS 32bit: 1 Lane (I & Q)
 - ▶ CSSI_DATA [31:0] = {RxDataPathI rounded[14:0], Gain_Change, RxDataPathQ rounded[14:0], Gain_Change }

Some of the other basic configuration modes, such as MSB/LSB first option, I or Q first option (for CMOS 1 lane), Long/Short strobe option are similar to previous SSI LVDS/CMOS 16bit operations.

POWER SAVING FOR LSSI

In TDD mode, the LVDS SSI pads can be powered down/up dynamically based on the TX_EN and RX_EN level to save power.

Note: The function hasn't been supported yet.

SSI TIMING PARAMETERS

Rx SSI and Tx SSI timing diagram are shown in [Figure 48](#) and [Figure 49](#). The preliminary timing specification for CMOS SSI is described in [Table 11](#) and the preliminary timing specification for LVDS SSI is described in [Table 12](#).

SYNCHRONOUS SERIAL INTERFACE (SSI)

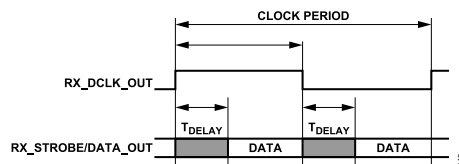


Figure 48. RX SSI Timing Diagram

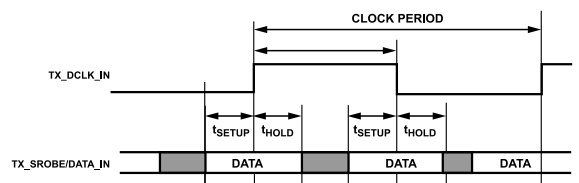


Figure 49. TX SSI Timing Diagram

Table 11. CMOS SSI timing Specification

CMOS SSI	Timing	Description
Cmos Rx $T_{\text{delay Max}}$	4.5ns	Data to Clock delay
Cmos Rx $T_{\text{delay Max}}$	5.5ns	Strobe to Clock delay
Cmos Tx $T_{\text{setup Min}}$	2ns	Strobe setup to Clock
Cmos Tx $T_{\text{setup Min}}$	1.5	Data setup to Clock
Cmos Tx $T_{\text{hold Min}}$	0ns	Strobe/data hold after Clock

Table 12. LVDS SSI timing Specification

LVDS SSI	Timing	Description
Rx $T_{\text{delay (Max)}}$	1 ns	Strobe/data to Clockdelay
Tx $T_{\text{setup (Min)}}$	0 ns	Strobe/data setup to Clock
Tx $T_{\text{hold (Min)}}$	0 ns	Strobe/data hold after Clock

SYNCHRONOUS SERIAL INTERFACE (SSI)

API PROGRAMMING

The ADRV910x SSI configuration is performed in the chip initialization stage and based on the "ssiConfig_t" data structure. In the data structure, the previously mentioned SSI modes are defined for each Tx/Rx channel. [Table 13](#) lists the SSI configuration parameters and some default values. Additionally, find the detailed data structure and enumerator description in the API Doxygen help file.

Table 13. SSI Configuration Data Structure

Parameter	Type	Description	Note
ssiType	enum	Set SSI type	
ssiDataFormatSel	enum	Set SSI data format	
numLaneSel	enum	Set SSI number of lanes	
strobeType	enum	Set SSI strobe type	
lsbFirst	uint8_t	Set LSB first	Default '0', MSB first
qFirst	uint8_t	Set Q data first	Default '0', I data first
txRefClockPin	enum	Set TX SSI reference clock output (TX_DCLK_OUT) options	
lvdsBitInversion	uint8_t	LVDS SSI I/Q/Strobe bit inversions	Default 0
lvdsUseLsbIn12bitMode	uint8_t	Set LVDS 12-bit mode	Default '0', LVDS SSI uses 16-bit mode
rxSsiDataSrc	enum	RX/RXNB SSI data source	
txDataSrc	enum	TX data path data source	
lvdsRxClkInversionEn	bool	Set LVDS RX SSI clock inversion enable	Default 'false'
cmosTxDdrNegStrobeEn	uint8_t	Set CMOS DDR negative strobe enable	Default 'false'
cmosDdrPosClkEn	bool	Set CMOS DDR positive clock enable	Default 'false'
cmosClkInversionEn	bool	Set CMOS DDR clock inversion enable	Default 'false'
DdrEn	bool	Set DDR mode enable	
rxMaskStrobeEn	bool	Set Rx Strobe Mask. Mask the Rx SSI Strobe when interface rate is multiple times of sample rate. See the Receive CSSI With 2x, 4x and 8x Data Clock Rates section.	Default 'false'
padding	uint8_t	reserve	

[Figure 38](#) shows the Rx CMOS SSI with the DDR clock in relation to the strobe/data. To ensure the BBIC gets the best setup/hold timing margin for RX CMOS DDR SSI, with [Table 13](#) RX CMOS DDR relative default SSI configurations (cmosDdrPosClkEn=false, cmosClkInversionEn=false), follow the Rx CMOS SSI output clock/strobe/data phase timing diagram shown in [Figure 50](#). ADI recommends this default RX CMOS SSI DDR configuration.

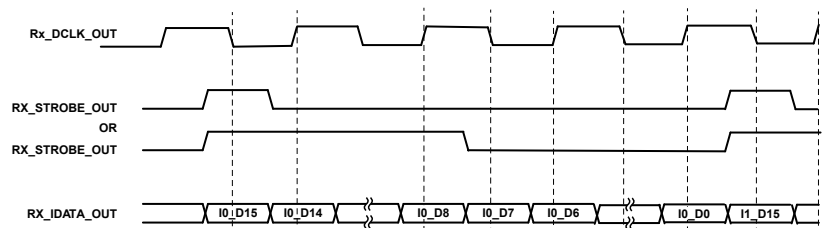


Figure 50. RX CMOS DDR SSI Default (Recommend) Output (cmosDdrPosClkEn=False, cmosClkInversionEn=False)

Also manually edit the CMOS DDR relative configurations based on the BBIC requirements. [Figure 51](#), [Figure 52](#), and [Figure 53](#) show the corresponding Rx output Clock/Strobe/Data timing diagram with different CMOS DDR configurations.

SYNCHRONOUS SERIAL INTERFACE (SSI)

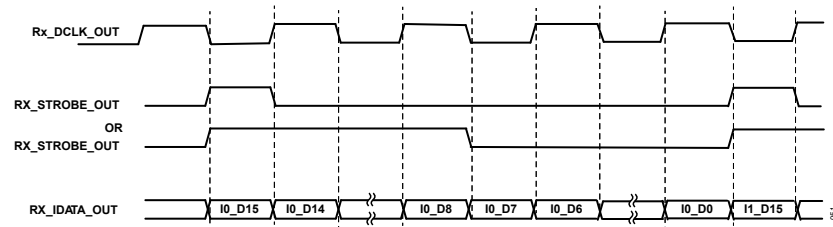


Figure 51. RX CMOS DDR SSI Output (cmosDdrPosClkEn=True, cmosClkInversionEn=False)

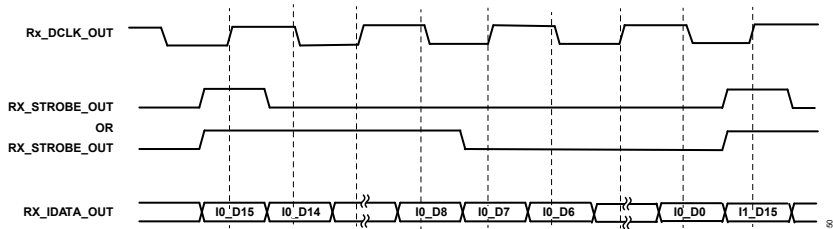


Figure 52. RX CMOS DDR SSI Output (cmosDdrPosClkEn=False, cmosClkInversionEn=True)

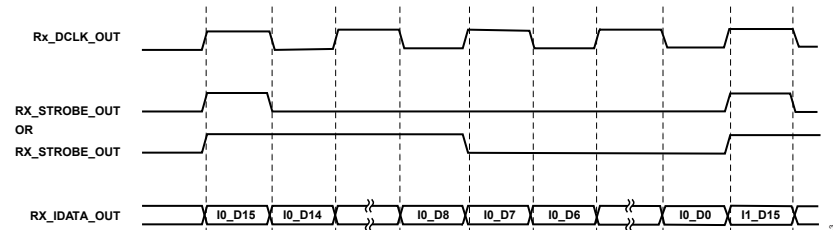


Figure 53. RX CMOS DDR SSI Output (cmosDdrPosClkEn=True, cmosClkInversionEn=True)

A set of API commands can set and inspect the SSI test/debug functions. [Table 14](#) summarizes these.

Table 14. SSI Test/Debug API List

SSI Function Name	Description
adi_adrv910x_Ssi_Rx_TestMode_Configure	Configures the SSI test mode for the specified Rx channel.
adi_adrv910x_Ssi_Tx_TestMode_Configure	Configures the SSI test mode for the specified Tx channel.
adi_adrv910x_Ssi_Tx_TestMode_Status_Inspect	Inspects the SSI test mode status for the specified Tx channel.
adi_adrv910x_Ssi_Loopback_Set	Enables the Rx to Tx SSI loopback.
adi_adrv910x_Ssi_Delay_Configure	Programs the SSI delay configuration.
adi_adrv910x_Ssi_Delay_Inspect	Gets the SSI delay configuration from the adrv910x device.
adi_adrv910x_Ssi_Tx_Disable_LSSI_Pads	Disable of Tx SSI pads
adi_adrv910x_Ssi_Tx_Enable_LSSI_Pads	Enable of Tx SSI pads

CSSI/LSSI TESTABILITY AND DEBUG

The ADRV910x SSI has built-in test pattern generator and checker to quickly test and debug the SSI between the ADRV910x and baseband processor. [Figure 54](#) shows the ADRV910x SSI testability and debug diagram with a baseband processor.

SYNCHRONOUS SERIAL INTERFACE (SSI)

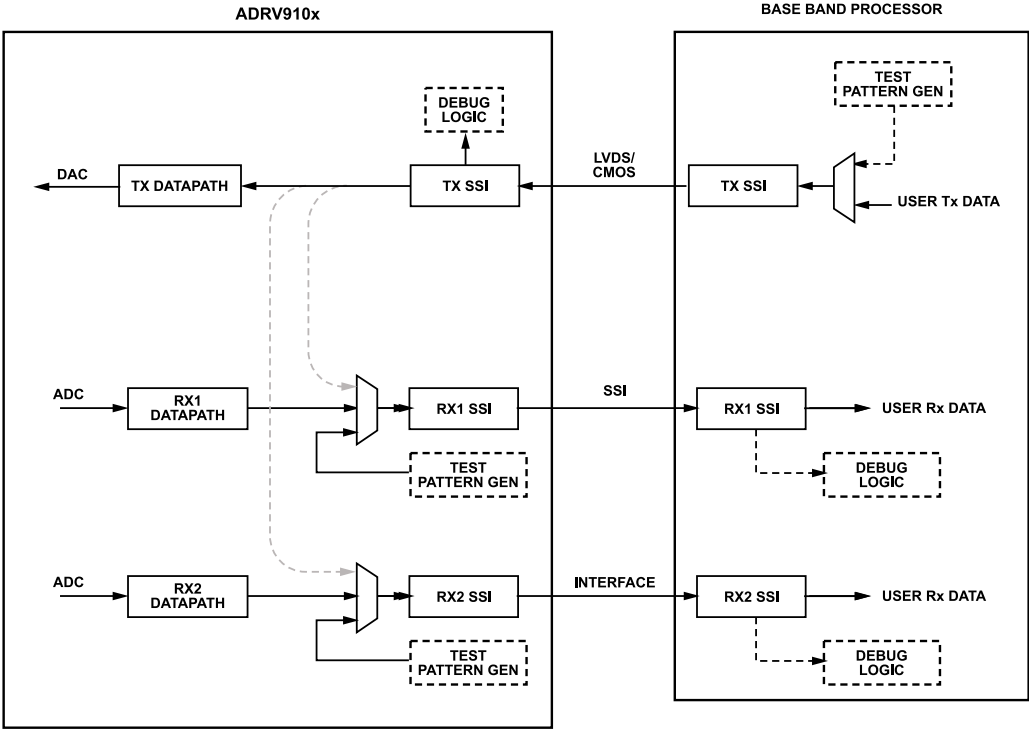


Figure 54. ADRV910x SSI Testability and Debug Diagram

The ADRV910x receive SSI can replace the receive channel data with a fixed pattern, ramp, or pseudorandom bit/binary sequence (PRBS) (LSSI only) pattern and send it to the baseband processor. Enable the receive debug function by calling `adi_adrv910x_Ssi_Rx_TestMode_Configure()`. Check the specified test pattern at the SSI output to test if the receive SSI from the ADRV910x to BBIC functions correctly.

The data structure `adi_adrv910x_RxSsiTestModeCfg_t` enables and configures the RX SSI test pattern.

```
typedef struct adi_adrv910x_RxSsiTestModeCfg
{
    adi_adrv910x_SsiTestModeData_e testData; /*!< Type of data to transmit over SSI */
    uint32_t fixedDataPatternToTransmit; /*!< Value of Fixed pattern to transmit over interface. For vari
    ous SSI data format:
    CMOS: Pattern is truncated to bit3 - bit0 value is transmitted on RxSSI I and Q each nibble (where
    applicable)
    LVDS: Pattern is truncated to bit15 - bit0 value transmitted on RxSSI I and Q (where applicable) */
} adi_adrv910x_RxSsiTestModeCfg_t;
```

The enum `adi_adrv910x_SsiTestModeData_e` enables and chooses the specified test pattern and lists in [Table 15](#).

Table 15. Definition of `adi_adrv910x_SsiTestModeData_e`

ENUM	Description
ADI_ADRV910X_SSI_TESTMODE_DATA_NORMAL	No test mode enabled
ADI_ADRV910X_SSI_TESTMODE_DATA_FIXED_PATTERN	Fixed pattern mode
ADI_ADRV910X_SSI_TESTMODE_DATA_RAMP_NIBBLE	Nibble ramp mode (CSSI only), I Q data is same and each 4-bit of the samples keep ramping (for example, sample0-0x0000, sampe1-0x1111, sample2-0x2222, ...).
ADI_ADRV910X_SSI_TESTMODE_DATA_RAMP_16_BIT	16-bit ramp mode, I Q data is same, 16-bit of the samples keep ramping (for example, sample0-0x0000, sample1-0x0001, sample2-0x0002,...).
ADI_ADRV910X_SSI_TESTMODE_DATA_PRBS15	PRBS15 mode (LSSI only)
ADI_ADRV910X_SSI_TESTMODE_DATA_PRBS7	PRBS7 mode (LSSI only)

SYNCHRONOUS SERIAL INTERFACE (SSI)

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

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

The ADRV910x transmit SSI has a ramp and PRBS (LSSI only) pattern checker. Configure the ADRV910x TX SSI test mode and transmit ramp or PRBS pattern through SSI to ADRV910x to verify if the SSI is functioning correctly. Also transmit a fixed pattern and configure the ADRV910x with the specified fixed pattern to verify the SSI functionality. Call API **adi_adrv910x_Ssi_Tx_TestMode_Configure** to enable and configure the test mode, and transmit the corresponding test patterns to the ADRV910x through Tx SSI, then call **adi_adrv910x_Ssi_Tx_TestMode_Status_Inspect** to get the ADRV910x TX SSI test mode status.

Similarly, data structure **adi_adrv910x_TxSsiTestModeCfg** enables and configures the ADRV910x TX SSI test pattern checker. BBIC transmits relative test patterns and the format follows the description in [Table 15](#). For the fixed pattern mode transmit, the BBIC puts bits 31:16 of fixedDataPatternToCheck on the TX SSI I data and bits 15:0 on Q data.

```
typedef struct adi_adrv910x_TxSsiTestModeCfg
{
    adi_adrv910x_SsiTestModeData_e testData; /*!< Type of data to receive over SSI and check */
    uint32_t fixedDataPatternToCheck; /*!< Value of Fixed pattern to check against pattern received over interface */
} adi_adrv910x_TxSsiTestModeCfg_t;
```

The ADRV910x transmit SSI data output can be looped back to receive the SSI data input using API **adi_adrv910x_Ssi_Loopback_Set**. When the transmit and receive SSI run at the same clock rate, use the pattern generator and checker to verify the functionality of the whole system SSI. Note that both the ADRV910x TX and RX radio state must be in the “RF_ENABLED” state to ensure the TX/RX SSI is enabled when setting the SSI loopback test function.

As mentioned previously, the SSI clock, strobe, and data have programmable delays. Configure these delays using **adi_adrv910x_Ssi_Delay_Configure**. The unit for the SSI delay is step. This aids to meet the timing specifications described in the [SSI Timing Parameters](#) section.

MICROPROCESSOR AND SYSTEM CONTROL

The ADRV910X supports quick configuration from idle states of operation and quick transition between receive and transmit states. Internal blocks called “Stream Processor” handles these transitions. The stream processor within the ADRV910X device performs a series of configuration tasks upon an external request. Upon a request, the stream processor performs a series of actions defined in the image loaded into the ADRV910X during the initialization process.

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

- ▶ TX WB Channel Enable/ TX WB Channel Disable
- ▶ TX NB Channel Enable/ TX NB Channel Disable
- ▶ RX1 Enable/RX1 Disable
- ▶ RX2 Enable/RX2 Disable

Paths are enabled and disabled typically using pins for TDD operation. However, these paths can also be controlled by PS2 or over the SPI bus using an API command. The stream is not limited to path-enabling events and can react to other events such as a DGPIO input signal.

The ADRV910X is flexible in its configuration, and, therefore, the stream is flexible. Similarly, as the initialization structures change with the profile, the stream processor image must change with the configuration. For example, the stream that enables Rx1 differs depending on the choice of either a narrowband or wideband profile. So, it is necessary to use a stream image for each configuration of the device. In this way, when the user saves configuration files (.c) using the ADRV910X TES, a stream image is also saved automatically. This stream file is then used when using these configuration files.

SYSTEM CONTROL

Control of the datapaths within the ADRV910X is done through the API or enable pin controls. For API control, this is reliant on the SPI communication bus and thus, for critical time alignment of powering on/off chains, pin control is recommended for TDD applications.

Independently control each datapath with the following enable signals:

Table 16. Datapath Enable signals

Enable Signal	Data Path
RX1_EN	RX1 Datapath
RX2_EN	RX2 Datapath
TX_EN	TX NB/WB Datapath
TX_EN_OPT	Reserved Signal

For ADRV910X to receive and react to control signals the channel radio state should be at “PRIMED” state. “PRIMED” state indicates that the system is ready for operation when the Tx and Rx channels are enabled by the user. After the channel is primed, the channel needs to be further transitioned from “PRIMED” state to “RF ENABLED” state to start transmitting or reception activities. This can be accomplished by “PIN Mode” or “SPI Mode”.

SPI Mode

- ▶ Call `adi_ADRV910X_Radio_ChannelEnableMode_Set()` to set the SPI mode
- ▶ Call `adi_ADRV910X_Radio_Channel_EnableRf()` to transition the channel to “RF ENABLED” state.

Once Pin or SPI/API mode is executed, the ADRV910X will enable the requested channels. The channels will remain active until further instruction thru Pin or SPI/API command.

PIN Mode

- ▶ Call `adi_ADRV910X_Radio_ChannelEnableMode_Set()` to set the PIN mode
- ▶ Toggle corresponding ENABLE pin to transition the channel to “RF ENABLED” state.

TIMING PARAMETERS CONTROL

The ADRV910X has integrated stream processors to handle various external and internal events. These events must be serviced in real time. These stream processors coupled with programmable delayed enable modules relieve the system firmware (running on integrated microprocessor) from managing all the critical events with a quick and parallel response to external and internal events. This configuration allows the 3-channels (TX, RX1, and RX2) to operate independently from each other by using their own dedicated stream processor.

MICROPROCESSOR AND SYSTEM CONTROL

The ADRV910X can support different applications, a set of programmable timing parameters for both transmitter and receiver meet the particular timing requirements in various TDD applications. It is crucial to understand the ADRV910X timing parameters to ensure all TDD events take place at an accurate time order. In addition, configuring timing parameters in an optimal way, by taking advantage of the multiple power saving modes of the ADRV910X, improves the overall system power consumption performance significantly.

Transmit Timing Definition

Transmit timing parameters define the events in order from the start of transmission at the ADRV910X data port to the end when the transmit burst is sent through the antenna to the air.

Figure 55 shows that a transmit burst consists of a series of valid transmit data with the option of guard data at the beginning and end of the valid data. Based on the transmitter enable signal and a set of configured transmit timing parameters, the ADRV910X further controls the transmitter interface, transmitter internal analog components, as well as the antenna switch (if it is controlled by the ADRV910X instead of the user) to make sure the transmit burst is on air at a deterministic time as desired.

TX timing parameters in Figure 55 can be categorized into three types: ADRV910X Parameter (ADRV910X provides to user), User Parameter (user provides to ADRV910X) and Helper Parameters (determined by user which are not needed to provide to ADRV910X but could be utilized by the user to derive other required timing parameters).

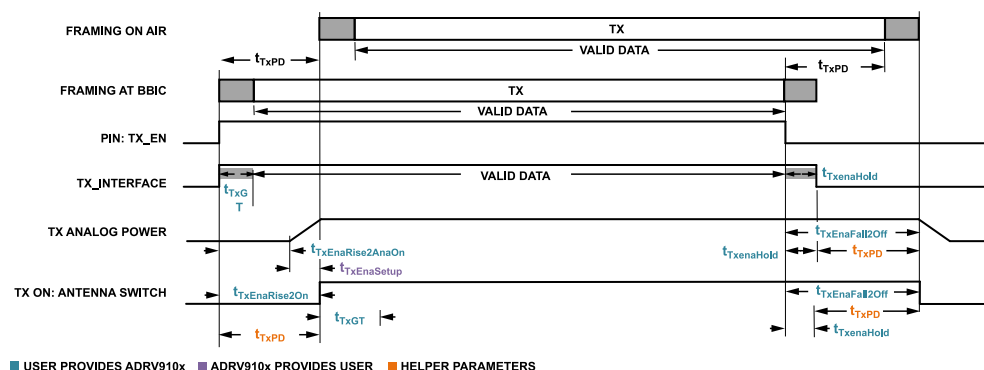


Figure 55. Transmitter Timing Parameters

The details of each timing parameters are described as follows:

- ▶ **Enable Setup Delay ($t_{TxEnaSetup}$)** is the time taken for the ADRV910X to power up the TX Analog Front End (AFE), Enable Setup Delay might include the PLL tuning time if PLL retuning is needed at frame boundary at the scenario that TX and RX share the same LO but at different frequency.
- ▶ **Enable Rise-to-Analog-On Delay ($t_{TxEnaRise2AnaOn}$)** is the delay between the TX enable (TX_EN) rising edge and ADRV910x TX analog power up beginning. This a user-defined parameter aids in saving power in setups with long propagation delays. If the transmitter propagation delay is long, delay the analog power up to save power. Set to 0 if the propagation delay is small. Refer Figure 56 and Figure 57 for the two typical use cases.
- ▶ **Guard Delay (t_{TxGT})** is the guard time at the beginning of the frame, the guard time is part of the frame on air that may or may not contain useful data depends on the standards. **Note:** Guard Delay is reserved timing parameter for future use and should be set to 0. Users might pad the necessary guard data at beginning of the TX frame that sent to ADRV910x.
- ▶ **Enable Hold Delay ($t_{TxEnaHold}$)** is the delay between the falling edge of TX_EN and the transmitter interface being disabled. TX_EN falling edge ideally comes as the last valid data is sent over the interface, therefore, set $t_{TxEnaHold}$ to 0.
- ▶ **Enable Fall-to-Off Delay ($t_{TxEnaFall2Off}$)** is the delay between the TX_EN falling edge and ADRV910X TX AFE powering down and the antenna switching if the antenna switching feature is enabled. Set this delay equal to the internal path delay (discussed next) for all transmit data to be propagated to the front end and to be transmitted out through the antenna.
- ▶ **Enable Rise-to-On Delay ($t_{TxEnaRise2On}$)** is the delay between the TX_EN rising edge and antenna switch changing to the TX channel if the antenna switch is controlled by ADRV910x. Typically, this delay is comparable in length with the propagation delay for transmitter channels. However, if the ADRV910X is not controlling the antenna switch, this parameter is not needed except to determine other parameters.
- ▶ **Propagation Delay (t_{TxPD})** is the delay of data from the ADRV910x TX digital interface input to antenna RF output. Given that this delay encompasses external components, it is impossible for the ADRV910X to determine what this delay is. The user must measure it, it does not have to be provided to the ADRV910X, but it can be used to derive other parameters required by the ADRV910X.

MICROPROCESSOR AND SYSTEM CONTROL

Additionally, **Internal Path Delay** is the delay between the ADRV910x SSI port and RF port for either the transmitter or receiver signal chains, ADRV910X internal calibration algorithm calibrates the internal path delay during the chip initialization stage, the calibration results don't include the SSI port processing delay which might be 1~2 sample clocks. As part of the design of a custom setup, use the Internal Path Delay as an approximation, and further measure the entire Propagation Delay of the setup to ensure the accurate transmitter/receiver timing on air in TDD operations.

Strategies for Transmit Timing Parameters

Figure 56 and Figure 57 show two typical use cases when determining the transmit timing parameters, where t_{TxGT} and $t_{TxEnaHold}$ are omitted since their values are 0.

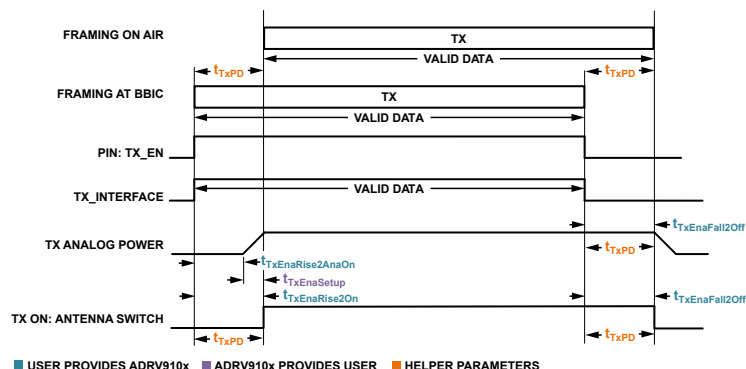
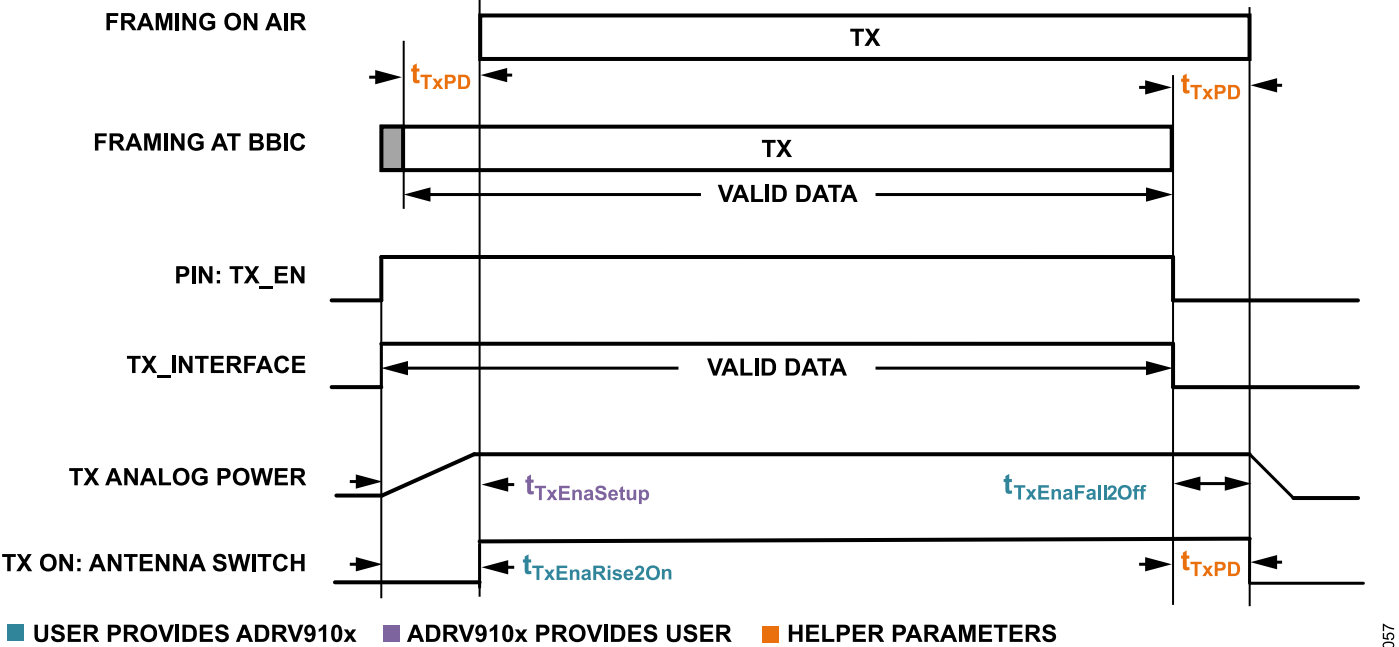


Figure 56. Transmitter Timing Parameters ($t_{TxPD} > t_{TxEnaSetup}$)

Figure 56 shows the use case that TX propagation delay is larger than the enable setup delay ($t_{TxPD} > t_{TxEnaSetup}$), the TX interface and digital data path are enabled when the TX_EN rising edge is asserted, TX analog front end is delayed powering up (by $t_{TxEnaRise2AnaOn}$) while the baseband data is propagating through the digital data path, this saves the power consumption by reducing unnecessary AFE on time. In this case, set the **enableRiseToAnalogOnDelay** to some value less than or equal to: **propagationDelay** – **enableSetupDelay** to get the most power saving. Set **enableRiseToOnDelay** equal to the **propagationDelay** or **enableRiseToAnalogOnDelay** + **enableSetupDelay**, the TX antenna switch timing will be aligned with the TX frame on air timing.

When the frame ends, the TX_EN falling edge should ideally occur right at the last valid data that is sent to TX interface and will be sending to the air. TX interface will be off at TX_EN falling edge, but the TX datapath and analog front end will be kept on for **enableFallToOffDelay** time to allow the last valid data to be propagated to air. To ensure all the data sent over the interface makes it onto the air, set **enableFallToOffDelay** greater than or equal to **propagationDelay**.

MICROPROCESSOR AND SYSTEM CONTROL



057

Figure 57. Transmitter Timing Parameters ($t_{TxPD} < t_{TxEnaSetup}$)

MICROPROCESSOR AND SYSTEM CONTROL

Figure 57 shows the use case that TX propagation delay is smaller than enable setup delay ($t_{TXPD} < t_{TXEnaSetup}$). In such a case, set **enableRiseToAnalogOnDelay** to 0, so that TX analog power up begins immediately after the TX_EN rising edge. To ensure the first TX valid data can be sent on air, some dummy (or zeros) data needs to be sent to TX interface after the TX_EN is asserted, the dummy data length shall be at least $(t_{TXEnaSetup} - t_{TXPD})$. For a more deterministic delay between the TX enable rising edge and antenna switch time, set **enableRiseToOnDelay** to a value greater than or equal to **enableSetupDelay**.

Similarly, when the TX frame ends, TX_EN falling edge aligns with the last valid data that is sent to TX interface, set **enableFallToOffDelay** greater than or equal to **propagationDelay** to ensure all the valid data to be sent over onto air.

Note: the “Valid Data” in Figure 56 and Figure 57 is the data that sent to TX interface and will be sent to antenna output, it may or may not include the guard data that standards require, it's fully controlled by the users.

Receive Timing Definition

Receive timing parameters define the events from the start of reception at the air to the end when the receive burst is sent through the ADRV910X data port to the BBIC.

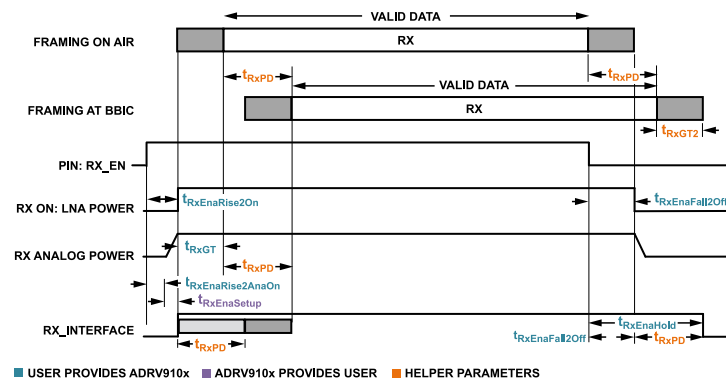


Figure 58. Receive Timing Parameters

Figure 58 shows that a receive burst has a series of valid data with the option guard data at the beginning and end of the valid data. Similar to transmit, based on the configured timing parameters, receive full or partial guard data to comply with the relative standards requirement. Control the RX_EN pin to signal the ADRV910X the start and end of a receive burst at the air. Based on the RX_ENABLE signal and a set of configured receive timing parameters, the ADRV910X further controls receive analog components, receive interface, and the external LNA (if it is controlled by the ADRV910X instead of the user) to make sure the received burst is sent to BBIC at the deterministic time as desired.

Receive timing parameters can be categorized into three types: ADRV910X parameter (ADRV910X provides to the user), user parameter (user provides to the ADRV910X), and helper parameters (determined by the user, which do not have to provide to the ADRV910X but can be used to derive other required timing parameters). The details of each timing parameters are described as follows:

- ▶ **Enable Setup Delay ($t_{RXEnaSetup}$)** is the time taken for the ADRV910X to power up the RX analog front end (AFE), Enable Setup Delay might include the PLL tuning time if PLL retuning is needed at frame boundary at the scenario that TX and RX share the same LO but at different frequency.
- ▶ **Enable Rise-to-Analog-On Delay ($t_{RXEnaRise2AnaOn}$)** is the delay between the RX enable (RX_EN) rising edge and ADRV910x RX analog power up beginning. Set **enableRiseToAnalogOnDelay** always to 0, so that analog power-up begins immediately after the RX_EN rising edge is asserted.
- ▶ **Enable Rise-to-On Delay ($t_{RXEnaRise2On}$)** is the delay between the RX_EN rising edge and RX interface on and LNA power up if the LNA power up is controlled by ADRV910x.
- ▶ **Guard Delay (t_{RXGT})** is the guard time at the beginning of the frame, Guard Delay is reserved timing parameter for future use and it should always be set to 0.
- ▶ **Enable Fall-to-Off Delay ($t_{RXEnaFall2Off}$)** is the delay between the RX_EN falling edge and ADRV910X RX AFE powering down and the LNA power down if the LNA power down feature is enabled. Ideally, RX_EN falling edge arrives when the last valid data is on the air. Set **enable fall to off delay** to 0, ADRV910x RX AFE and external LNA will be started powering down when RX_EN falling edge is asserted.
- ▶ **Enable Hold Delay ($t_{RXEnaHold}$)** is the delay between the RX_EN falling edge and the RX interface being disabled, this delay keeps RX data path and interface on after RX AFE is powered down to ensure the valid RX data is propagated to base band.

MICROPROCESSOR AND SYSTEM CONTROL

- **Propagation Delay (t_{TxPD})** is the delay of data from the antenna RF input ADRV910x RX interface output. Given that this delay encompasses external components, it is impossible for the ADRV910X to determine what this delay is. The user must measure it, it does not have to be provided to the ADRV910X, but it can be used to derive other parameters required by the ADRV910X.

Strategies for Receive Timing Parameters

Figure 59 shows the typical RX timing parameters use case where t_{RxGT} and $t_{RxFall2Off}$ are omitted since their values are always 0. In regular TDD operation, based on the frame timing on air, the RX_EN should be raised at least **enableRiseToOnDelay** time before the start of the actual frame time, this provides the time for to power up the ADRV901x RX analog front end and external LNA. Set **enableRiseToAnalogOnDelay** to 0, so that RX AFE power-up begins immediately after RX_EN rising edge. For the deterministic delay between the RX_EN rising edge and RX interface on and LNA power up time, set **enableRiseToOnDelay** to a value greater than or equal to: **enableRiseToAnalogOnDelay + enableSetupDelay**. The RX interface is turned on when **enableRiseToOnDelay** is expired, but because the RX progradation delay, the initial data coming off the interface is not the valid data received on the air.

When the RX frame ends, RX_EN falling edge ideally aligns with last valid data on the air, the external LNA and ADRV910x RX AFE starts powering down at the RX_EN falling edge, but the RX data path and interface keeps on for at least **EnaHoldDelay** time to propagate all the received valid data and send over RX interface to baseband processor, set **EnaHoldDelay** to a value greater than or equal to **propagationDelay**.

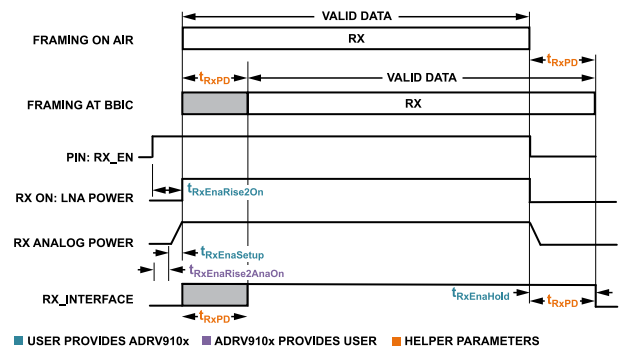


Figure 59. Receive Timing Parameters Use Case

Guard Time Between Edges of TX_EN and RX_EN

In TDD operations, the guard time between edges of TX_EN and RX_EN to make sure the TX and RX analog front end won't be powered up simultaneously. There is no hardware or software restriction to prevent TX_EN/RX_EN toggling at any time, always set the guard time greater than the minimum requirement to make sure the ADRV910X TX/RX works properly.

Guard Time Between TX_EN Falling Edge and RX_EN Rising Edge

There is $t_{TxEnaFall2Off}$ delay to start powering down TX AFE at the TX_EN falling edge, it takes $t_{RxEnaRise2On}$ to power up the RX AFE after the RX_EN rising edge. If $t_{TxEnaFall2Off}$ is equal or bigger than $t_{RxEnaRise2On}$, the RX_EN rising edge can only happen after TX_EN falling edge, the minimum guard time from TX_EN falling edge to RX_EN edge is $(t_{TxEnaFall2Off} - t_{RxEnaRise2On})$. If $t_{TxEnaFall2Off}$ is smaller than $t_{RxEnaRise2On}$, the RX_EN rising edge can be at any time after the TX_EN falling edge, or even the RX_EN rising edge is before the TX_EN falling edge with the maximum allowable ahead time $(t_{RxEnaRise2On} - t_{TxEnaFall2Off})$. Figure 60 shows the two cases.

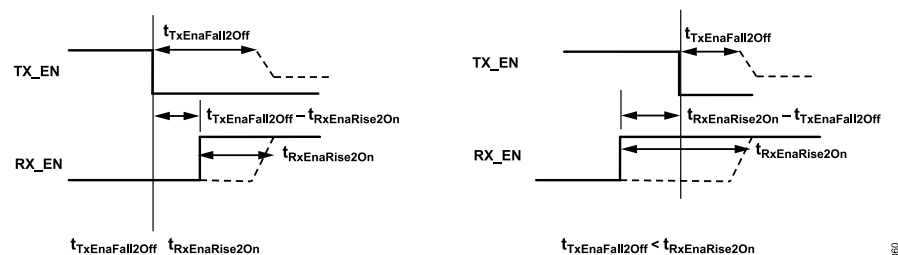


Figure 60. Guard Time Between TX_EN Falling Edge and RX_EN Rising Edge

MICROPROCESSOR AND SYSTEM CONTROL

Guard Time Between RX_EN Falling Edge and TX_EN Rising Edge

Typically, RX AFE power down is started at RX_EN falling edge ($t_{RxEnaFall2Off}=0$), it takes $t_{TxEnaRise2On}$ to power up the TX AFE after the TX_EN rising edge, therefore, the TX_EN rising edge can be at any time after the RX_EN falling edge, or the TX_EN rising edge is before the RX_EN falling edge with the maximum ahead time $t_{TxEnaRise2On}$. Figure 61 shows the guard time from TX_EN rising edge and RX_EN falling edge.

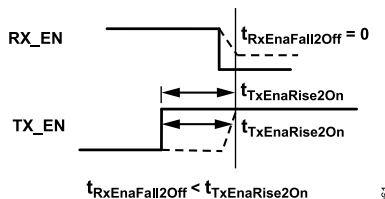


Figure 61. Guard Time Between TX_EN Rising Edge and RX_EN Falling Edge

Guard Time Between TX_EN Falling Edge and TX_EN Rising Edge

The guard time between TX_EN falling edge and TX_EN rising edge makes sure the AFE is powered off in the previous frame before powering up again in the new frame. There is $t_{TxEnaFall2Off}$ delay to start powering down TX AFE at the TX_EN falling edge and it takes $t_{TxEnaRise2On}$ to start powering up the TX AFE after the TX_EN rising edge, therefore the minimum guard time between TX_EN falling edge and TX_EN rising edge should be $(t_{TxEnaFall2Off} - t_{TxEnaRise2On})$ as shown in Figure 62.

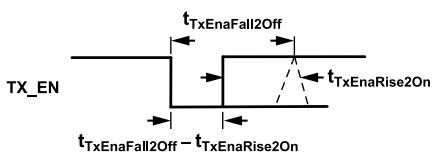


Figure 62. Guard Time Between TX_EN Falling Rising Edge and TX_EN Rising Edge

Guard Time Between RX_EN Falling Edge and RX_EN Rising Edge

The guard time between the RX_EN falling edge and RX_EN rising edge makes sure the interface is turned off at the end of the previous frame before it turns on again for the next frame. The RX interface keeps on for $t_{RxEnaHold}$ time after RX_EN falling edge, there is $t_{RxEnaRise2On}$ delay turning on the Rx interface after RX_EN rising edge, so the minimum guard time between RX_EN falling edge and RX_EN rising edge should be $(t_{RxEnaHold} - t_{RxEnaRise2On})$ as shown in Figure 63.

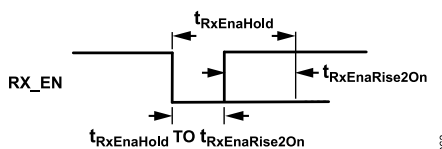


Figure 63. Guard Time Between RX_EN Falling Rising Edge and RX_EN Rising Edge

Hold Time for TX_EN and RX_EN

Hold time for TX_EN and RX_EN is the minimum enable duration time for the enable signal rising edge to enable signal falling edge.

It takes $t_{EnaRise2On}$ time to power up the TX/RX channel, theoretically, the minimum TX/RX enable duration time should be bigger than the $t_{EnaRise2On}$ time. In TDD practice, the shortest TX/RX enable hold time should be $\max [t_{EnaRise2On}, 10 \text{ us}]$.

TIMING PARAMETERS WITH POWER SAVING MODES

TBD.

CLOCK GENERATION

In the ADRV910X, the internal CLKGEN generates all clocks for converters, data path, PS1 and PS2 ARM subsystem, etc.

The DEV_CLK_IN is the base clock of the ADRV910x CLKGEN, the optional clock doubler can double the DEV_CLK_IN frequency, either DEV_CLK_IN or the clock doubler output can be used for the following reference clocks generation. The simplified diagram of CLKGEN is shown in Figure 64. Four different clocks are generated from the DEV_CLK_IN.

- ▶ PLL Ref clock: Used to generate the clock PLL reference, as well as the RF PLL reference.
- ▶ ADC/DAC Ref clock: used as Converters clock and digital system clock when Clock PLL is disabled.
- ▶ Internal Ref clock: digital startup clock, used as a free running clock.
- ▶ User Ref clock: one option source for DEV_CLK_OUT

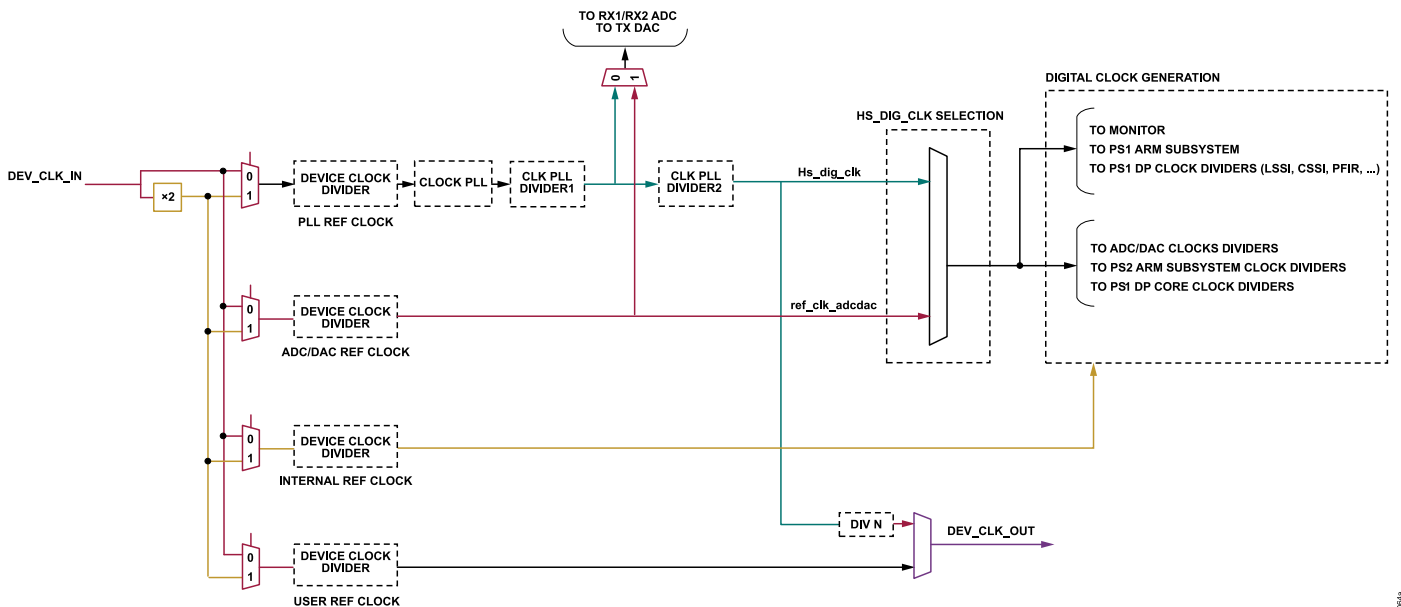


Figure 64. Clock Generation for Converters and Baseband

The ADRV910x Clock PLL with integrated on chip VCO runs at frequency 6.8GHz~9.6GHz, the consecutive dividers to generate ADC/DAC clock and the high-speed digital system clock (Hs_dig_clk).

ADRV910x provides the option to use DEV_CLK_IN (with optional doubler) to directly drive the TX/RX2 narrow band profile ADC/DAC and digital clock generation¹. This is useful for the narrow band profiles to get the best active power consumption performance.

Depends on the profiles setting, the digital clock generation takes the clock either from clock PLL (Hs_dig_clk) or DEV_CLK_IN (ref_clk_adc-dac) to generate the clocks for ADC/DAC, data path, SSI, PS1/PS2 ARM subsystem, etc.

The internal ref clock is used as the startup clock of the chip, the internal reference clock is free running at chip power up.

ADRV910x provides an output clock (DEV_CLK_OUT) for users' system, it could be a divided down version of the DEV_CLK_IN reference clock or Hs_dig_clk from Clock PLL.

The ADRV910x firmware will configure the CLKGEN accordingly based on the profile settings. Users only need to enable DEV_CLK_OUT with a proper divider ratio or disable the DEV_CLK_OUT. Note, the DEV_CLK_OUT has default divider ratio associated with the MODEA pin level when device is powered up, refer Table 49 for more information.

¹ DEV_CLK_IN range will be constrained by RX2 ADC sample rate and dataport input/output sample rate

SYNTHESIZER AND LO OPERATION

OVERVIEW

The ADRV910X family devices employ three fully integrated Phase Locked Loop (PLL) synthesizers: two RF PLL Synthesizers for internal LO, one Clock PLL synthesizer for TX/RX converters and baseband clocking. Each PLL synthesizer is based on a fractional-N architecture and consists of a common block made up of a reference clock divider, phase frequency detector, charge pump, loop filter, feedback divider, and digital control block and voltage-controlled oscillator (VCO). The internal RFPLL VCO has a tuning range of 6GHz to 12GHz, the VCO outputs are steered through a combination of frequency dividers to produce the desired internal LO frequency. The Clock PLL VCO has a tuning range of 6.8GHz to 9.6GHz followed by series of frequency dividers to generate the high-speed digital clock. No external components are required for the synthesizers to cover the entire frequency range of the device.

The ADRV910X family devices also integrate two RFPLLs working with external loop filter and external VCOs to provide the LOs for TX/RX channel, the RFPLLs for external VCO are notated as EXT (VCO) RFPLL to differentiate with the internal LO RFPLL.

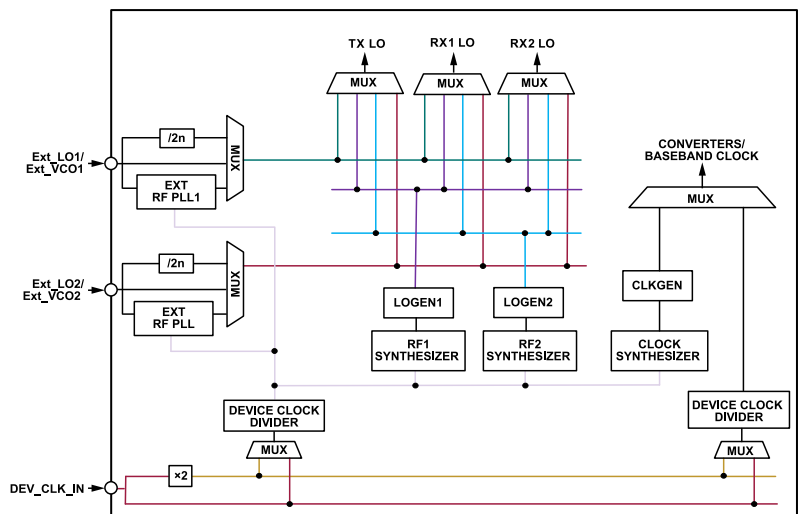


Figure 65. Synthesizer Interconnection and Clock/LO Distribution Block Diagram

The Synthesizer interconnection and clock/LO distribution block diagram is shown in Figure 65. The DEV_CLK_IN provides the reference clock for all the PLLs, the LO mapping network allows the device to provide the flexibility in LO assignment for the internal LOs, external LOs, and external VCOs.

CLOCK SYNTHESIZER

The clock synthesizer is used to generate all the baseband clocking signals necessary to run the device. The reference frequency for the clock PLL is scaled from the device clock or doubler of the device clock by the PLL reference clock divider. Reconfiguration of the clock synthesizer is typically not necessary after initialization. The most direct approach to configuration is to follow the recommended programming sequence and utilize provided API functions to set the clock synthesizer to the desired mode of operation. The Clock Synthesizer VCO runs frequency 6.8GHz~9.6GHz, the dividers at clock synthesizer output generate the clock signals for ADC/DAC and high-speed digital clock.

Clock Synthesizer can be disabled for some narrow band applications, refer [Clock Generation](#) section for more details about the option to use device clock or Clock Synthesizer to generate the converters and baseband clocking.

RF SYNTHESIZER

The device contains two RF PLLs for internal LOs. Each RF PLL uses the PLL block common to all synthesizers in the device and employ a two core VCO block. As with the Clock synthesizer in the device, the reference for RF PLL 1 and 2 are sourced from the device clock or doubler of the device clock. The RF PLLs are fractional-N architectures with a programmable modulus. The default modulus of 8,388,593 is programmed to provide an exact frequency on at least a 5KHz raster using certain reference clocks which are integer multiples of 38.40MHz. The internal RF LO frequency is derived by dividing down the VCO output in the LOGEN block. The tuneable range of the RF LO is 70-6000 MHz.

A switching network is implemented in the device to provide flexibility in LO assignment for the two RF LO sources. Figure 65 shows the switching network.

SYNTHESIZER AND LO OPERATION

LO Phase Noise

Figure 66 shows the typical PLL phase noise contributors. For low offset frequencies (within the PLL loop filter bandwidth), reference clock dominates the phase noise, and for high offset frequencies, VCO noise dominates the phase noise. User can optimize the phase noise by:

- Provide better phase noise reference clock source, especially for narrow band type application that requires good phase noise within the PLL bandwidth. Refer Table 50 DEV_CLK_IN phase noise requirements
- Provide higher reference clock frequency (PFD), the maximum PFD is up to 100MHz
- Adjust loop filter bandwidth to trade-off between close-in band and far-out band noise

When changing the loop filter bandwidth, typically consideration is the wider the bandwidth, the better close-in phase noise, but the worse the far-out phase noise. User should trade-off between the two to find the optimal setting for the specific application.

Figure 67 shows the 520MHz Internal LO phase with 38.4Mhz and 100Mhz reference clock frequency at different PLL loop filter bandwidth that demonstrates how the PLL BW and PFD frequency affect the ADRV910x internal LO phase noise. Users could refer this example to optimize internal the LO phase noise based on the application details.

The internal RF synthesizer PLL loop filter bandwidth can be configured by API `adi_adrv910x_Radio_PllLoopFilter_Set`.

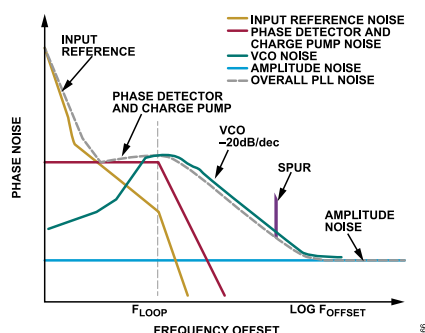


Figure 66. PLL Phase Noise Contributors

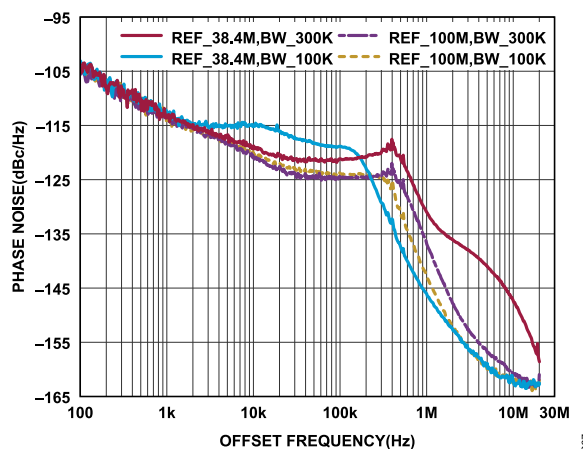


Figure 67. 520Mhz Internal LO phase noise with different reference clock frequency and PLL BW

EXTERNAL VCO RFPLL

The ADRV910X family devices integrate two RFPLLs working with external loop filter and external VCOs to provide LOs for TX/RX channel, this would allow the ADRV910x RF PLL to tune the custom external discrete VCO circuits and provide more flexibility for users to design external VCOs based on the system requirements.

The principal techniques of ADRV910x external VCO RFPLL (EXT RFPLL) are similar with the internal RF synthesizer but extend them to discrete VCOs using varactors. To reduce the PLL tuning voltage range requirement for ADRV910x, the EXT RFPLL utilizes the coarse tune and fine tune loops for the external VCOs, which means the external VCO design needs at least two discrete varactors, one for the coarse tune

SYNTHESIZER AND LO OPERATION

covering the PVT (Process, Voltage, and Temperature) as well as the temperature tracking, which will be driven by EXT RFPLL coarse output with necessary external interface circuits, another varactor is fine tune and is driven by EXT RFPLL charge pump directly.

The coarse frequency tuning loop initially tunes the VCO as close as possible to the target frequency, this is called Calibration phase or coarse tune phase; then the PLL Charge Pump (CP) loop continues to drive the fine varactor to the target frequency until PLL is locked, this is called PLL settling phase or fine tune phase. The fine tune voltage (V_{fine}) is maintained around $0.5 \pm 0.3V$ while PLL is locked, an analog temperature compensation loop monitors the V_{fine} and adjust the coarse tune output to compensate for any deviation in V_{fine} beyond the $0.5 \pm 0.3V$ window, this is the Tracking phase. The ADRV910x EXT RFPLL frequency turning and tracking loop is fully controlled by digital state machine, no user's interaction is needed after setting the desired frequency and necessary PLL parameters.

The ADRV910x EXT RFPLL two loop tuning interface for external VCO is shown in Figure 68.

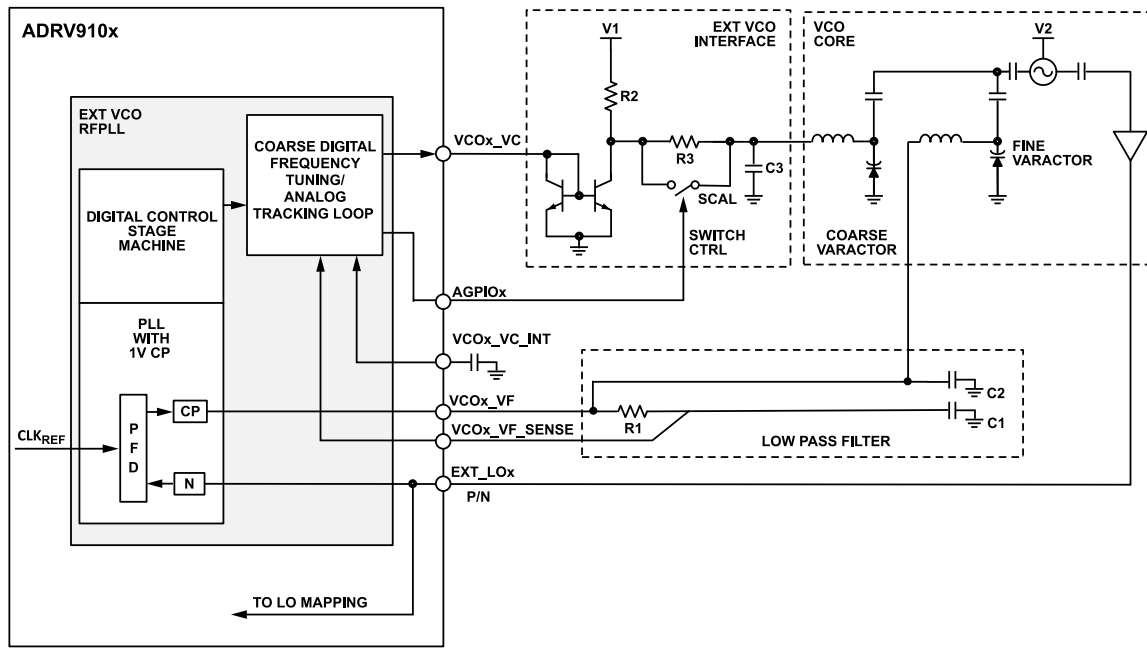


Figure 68. ADRV910x EXT RFPLL Two Loop Tuning Interface for External VCO

The EXT RFPLL coarse tune output is driven by a 12-bit DAC with maximum 500uA current, the current output on VCOx_VC pin is converted to the required tuning voltage on the coarse varactor by the current mirror and external power supply (V1).

The voltage on coarse varactor is denoted as below equation.

$$V_{coarse} = V1 - R2 \cdot I_{Coarse_DAC}$$

where I_{Coarse_DAC} is the Coarse tune DAC output current, V1 is supply voltage, R2 is the resistor between the supply and current mirror output.

The noise from the coarse loop and the current mirror is filtered by a very low frequency RC filter (filter_{R3, C3}) which is bypassed during coarse tuning stage by turning on the switch (Scal) via the control signal from AGPIO_4 (for EXT RFPLL2) or AGPIO_5 (for EXT RFPLL1) Pins. Note, the control signal on AGPIOs is level "high" (1.8V) to be used to turn on the switch.

The charge pump output on VCOx_VF pin goes through the external loop filter and drives the VCO fine varactor, the charge pump current is from 0.1mA to 6.4mA at 0.1mA per step. Users should design the external loop filter based on the charge pump and relative external VCO parameters.

The fine tune voltage is always sensed through VCOx_VF_SENSE pin for the analog tracking loop to maintain the V_{fine} within the target window.

The VCO (buffer) output goes to EXT_LOx port and goes to PLL feedback loop and LO mapping path. The differential mode EXT_LO port is suggested for external VCO signal to ADRV910x.

To update an existing single varactor VCO design for ADRV910x EXT RFPLL, the same varactor in the old design could be used for the coarse tune, an additional fine tune varactor with 10x smaller gain than coarse varactor could be added, the total frequency range covered by the two

SYNTHESIZER AND LO OPERATION

varactors will approximately equal to that of a single varactor, and since the gain of the fine varactor is much smaller, its impact on the VCO design should be minimal.

External VCO Interface Circuits Design Consideration

Users need to overall consider the VCO frequency range, PLL Lock time and phase noise performance to properly design the external VCO interface circuits between ADRV910x coarse tune output and external VCO coarse varactor showed in [Figure 68](#).

As previously described, the voltage on coarse tune (V_{coarse}) is determined by equation $V1 - R2 \cdot I_{\text{Coarse_DAC}}$. $V1$ could be the maximum voltage when Coarse DAC current is 0, and based on the minimum coarse voltage need, the $R2$ value could be derived with maximum Coarse DAC current.

During the “coarse tune phase”, the switch “Scal” is closed to bypass the resistor ($R3$) of the filter $R3, C3$ for the fast coarse tune, for the each tuning step, V_{coarse} takes time to settle (charge the capacitor ($C3$)) which is determined by the RC time constant ($\tau_{R2, C3}$), the larger $\tau_{R2, C3}$ the longer “Coarse tune phase” and PLL lock time. The “extVcoPllSettle” setting is related on the RC time constant ($\tau_{R2, C3}$). Switch “Scal” is opened in the “fine tune phase” and “tracking phase”, so the filter $R3, C3$ could filter the noise from the coarse loop and current mirror circuits to provide the better PLL close in phase noise, the loop filter bandwidth might be only several Hertz or even narrower to get the better close in PLL phase noise.

Once the $V1$ and $R2$ value is decided by the coarse tune voltage range requirement, users could further decide the $C3$ value based on PLL lock time requirement and $R3$ value by PLL close in phase noise requirement. For a given $R2$ setting, we need smaller $C3$ to get shorter coarse tune time, but because the $R3$ can't be unlimited big, we need bigger $C3$ to get narrower filter $R3, C3$ bandwidth for the better close in phase noise. There has to be some trade-off for the PLL lock time and PLL close in phase noise to decide the proper $C3$ and $R3$ value.

There is another option for the external VCO interface circuits if both the PLL lock time and close in phase noise have stringent requirements, [Figure 69](#) shows the second option of external VCO interface circuits block diagram that locates between ADRV910x and external VCO. An operational amplifier is added between the current mirror output and filter $R3, C3$ input. The operational amplifier with small capacitive input and resistive output breaks the $R2/C3$ time constant limitation during coarse tune phase, so the smaller “extVcoPllSettle” value can be set to get shorter PLL lock time. At same time, a bigger value capacitor for $C3$ is possible to get a narrower low pass filter for the best PLL close in phase noise.

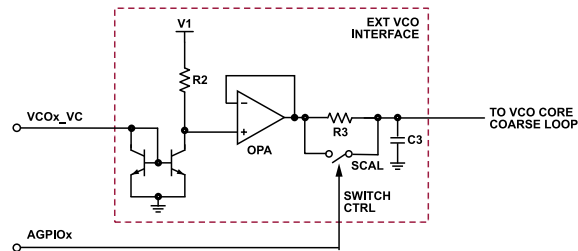


Figure 69. ADRV910x EXT VCO Interface Circuit Option 2

Users should base on their system requirements to decide which external VCO interface circuits in [Figure 68](#) and [Figure 69](#) is best for them.

EXT VCO PLL Lock Time

The external VCO PLL lock time primarily consists of the coarse tune time and the fine tune time. The coarse tune is mainly determined by the “extVcoPllSettle” setting which depends on the external VCO interface circuits. The fine tune time depends on the PLL charge pump current setting, external loop filter bandwidth, etc. If there is PLL lock time requirements for external VCO, users could optimize the external VCO interface design and/or external VCO PLL settings based on the majority factor of PLL Lock time.

[Figure 70](#) and [Figure 71](#) show two external VCO PLL lock time measurement result examples on different external VCO hardware and the corresponding EXT VCO RFPLL settings.

SYNTHESIZER AND LO OPERATION

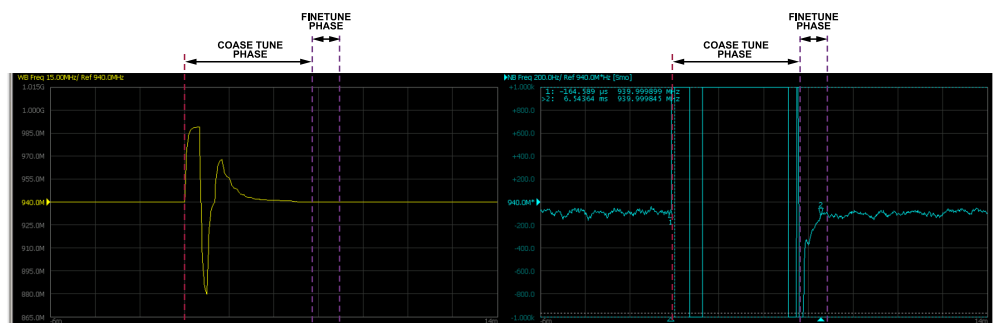


Figure 70. Ext VCO PLL Lock Time Example 1

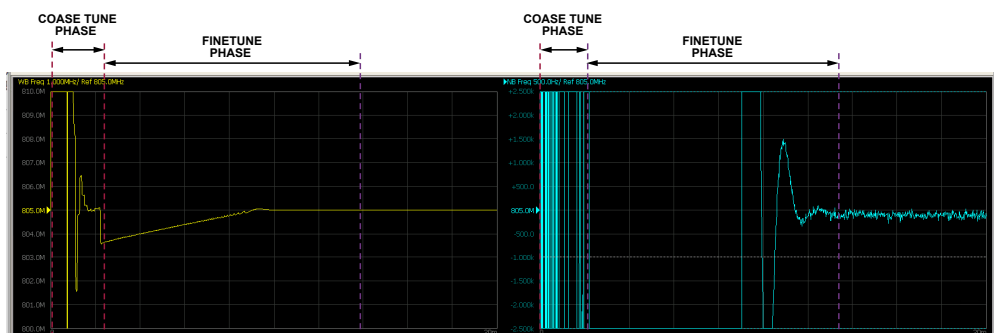


Figure 71. Ext VCO PLL Lock Time Example 2

The PLL lock time in Figure 70 example is mainly dominated by coarse tune phase, if needs, users should try to optimize the VCO interface that has been discussed in [External VCO Interface Circuits Design Consideration](#), so the external VCO PLL setting parameter “extVcoPllSettle” can be set with smaller value to short the coarse tune phase time.

The PLL lock time in Figure 71 example is mainly limited by fine tune phase, users could increase the PLL loop filter bandwidth to short the fine tune phase by increasing the charge pump current or changing the external loop filter components value, but the wider PLL loop filter bandwidth might sacrifice the PLL output phase noise. To have shorter PLL Lock time and maintain the PLL phase noise no change, users could enable the “extVcoPllSpeedUpMode”. In this mode, after coarse tune phase is completed, the external VCO RFPLL uses a larger charge pump current (“extVcoPllSpeedUpChargePumpCurrent”) to expedite the fine tune phase until a timer (“extVcoPllSpeedUpCounter”) is expiry, then switch to the original charge pump current (“extVcoPllChargePumpCurrent”) to lock the PLL and maintain the same PLL loop gain, the process is fully controlled by EXT RFPLL digital stage machine, no users interaction is needed. Figure 72 shows the external VCO PLL Lock time example with “extVcoPllSpeedUpMode” enabled, where the coarse tune phase is no change, but fine tune phase is split to fine tune speed up phase and final phase, and the overall fine tune phase is much faster. There is an obvious frequency disturbance at “fine tune final phase” which is caused by the charge pump current switch from “extVcoPllSpeedUpChargePumpCurrent” to “extVcoChargePumpCurrent”.

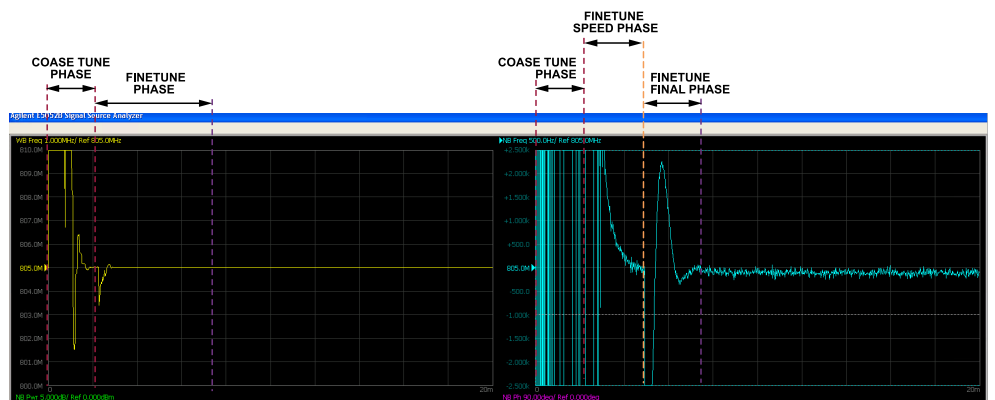


Figure 72. Ext VCO PLL Lock Time Example 3 (Speed Up Mode)

SYNTHESIZER AND LO OPERATION

EXT VCO PLL Modeling Consideration for External VCO Design

For modeling purposes, the ADRV910x external VCO PLL can be viewed as two parallel control loops acting on the VCO. The primary loop is the fine-tuning PLL loop, which dominates the dynamic behavior and determines the transient response, settling behavior, and in-band phase noise. Fine tune loop has the customer-controlled parameters include external VCO characteristics (K_{vco} , tuning range), loop filter, PLL bandwidth, charge pump current, and divider ratio. users can model this loop using any standard PLL tools. In parallel, there is a coarse-tuning loop used for VCO calibration and long-term drift correction (e.g., temperature). This loop is intentionally designed with a very low bandwidth (on the order of Hz, determined by the LPF on coarse path), so it does not affect PLL stability, transient response, or settling time. Because of its low bandwidth, the coarse loop is typically not included explicitly in most PLL modeling tools.

From a noise perspective, the overall PLL response can be treated as the combination (summation) of the fine and coarse loop paths. The noise contribution from the coarse-tuning path can be modeled as an additional low-frequency noise source injected into the VCO control node and shaped by the combined loop transfer function. In practice, this noise is strongly filtered by the low path filter on the coarse tune path and remains negligible outside very low offset frequencies.

As a result:

- ▶ Inside the PLL bandwidth, phase noise is dominated by the PLL (PFD/CP, loop filter, and reference).
- ▶ Outside the PLL bandwidth, phase noise is dominated by the external VCO.
- ▶ The coarse-tuning loop does not materially impact transient behavior or phase noise at offsets of interest for most applications.

This approach should give users sufficient confidence to model and predict the combined performance of their external VCO with the ADRV910x PLL by using standard PLL modeling tools.

EXT VCO RFPLL Configuration

ADRV910x provides configurable parameters for EXT RFLLL to optimize the phase noise and PLL lock time with the custom external VCO, the configurable parameters are defined in "adi_adrv910x_ExtVcoConfig_t" and are passed to ADRV910x by API "adi_adrv910x_Radio_Carrier_Configure". Table 17 shows the external VCO RFPLL configurable parameters details.

Table 17. EXT VCO RFPLL configurable parameters

Parameter	Type	Description	Note
forceExtVcoPllRefClkDivider	bool	If set to true, forces override of reference clock divider using extVcoPllRefClkDivider	"false" by default, ADRV910x firmware will set the EXT RFPLL reference clock divider automatically to meet the $F_{vco}/F_{pll_ref_clk} \geq 20$, $F_{pll_ref_clk} = F_{dev_clk} / 2^{PLLRefClkDivider}$ Note: Pll_ref_clk is the PFD clock frequency.
extVcoPllRefClkDivider	uint8_t	EXT RFPLL reference clock divider setting	Only used if forceExtVcoPllRefClkDivider = "true". $F_{pll_ref_clk} = F_{dev_clk} / 2^{PLLRefClkDivider}$
extVcoPllChargePumpCurrent	uint8_t	Charge pump current	Valid range: min 0 = 0.1mA, max 63 = 6.4mA, step 1 = 0.1mA
extVcoPllSettle	uint8_t	time to wait for EXT RFPLL coarse tune to settle for each tune step	Min: $0 = 2^0 * 256$ cycles, max: $10 = 2^4 * 1024$ cycles. Where the "cycle" is the PLL reference clock. The settling time should typically be larger than $3 * T_{R2, C3}$
extVcoPllSpeedUpMode	Bool	Set to true to enable speed up mode	
extVcoPllSpeedUpCounter	uint16_t	time duration for fine tune speed up phase	Time duration for the speed up phase is calculated using the equation $extVcoPllSpeedUpCounter * 512 * T_{PLLRefClk}$. $T_{PLLRefClk}$ is the PFD clock period.
extVcoPllSpeedUpChargePumpCurrent	uint8_t	Charge pump current for fine tune speed up phase	Valid range: min 0 = 0.1mA, max 63 = 6.4mA, step 1 = 0.1mA. Should be bigger than extVcoPllChargePumpCurrent.
extVcoPllChargePumpBleedCurrent	uint8_t	Charge pump bleed current to linearize the PLL	Larger bleed current might help lower the PLL integer boundary spurs. Valid range: min 0 = 0uA (bleed disabled), step 1 = 5uA. Bleed current must not exceed one-tenth of extVcoPllChargePumpCurrent.

SYNTHESIZER AND LO OPERATION

EXT VCO PLL and External VCO Testability and Debug

The ADRV910x provides debug support for external PLL operation with an external VCO, enabling PLL status monitoring and VCO characterization.

External PLL Status

Use "adi_adrv910x_Radio_ExtPll_Status_Get" to read the external PLL status, "pllLocked" indicates whether the external PLL is locked. "coarseDacCode" is the Coarse tuning DAC code for the external VCO (typical range: 0x000–0xFFFF). Values of 0x000 or 0xFFFF typically indicate coarse-loop non-convergence, suggesting the programmed PLL frequency is outside the external VCO tuning range.

Open-Loop Test Mode

Use API "adi_adrv910x_ExtPllOpenLoopTestMode_Configure" to place the external PLL in open-loop mode for debug and characterization. "coarseLoopSwitchOn" enables/disables the "Scal" in [Figure 68](#) to evaluate low pass filter on coarse tune path impact on VCO phase noise. "cpTest" sets the charge pump output state when the loop is open (recommended setting: GND). "forceUpdateCoarseDACCode" and "coarseDACCode" manually control the coarse DAC to verify VCO tuning range and measure open-loop phase noise at specific frequencies.

...

EXTERNAL LO

The device is provisioned with 2 external LO ports, these ports are available as a pair of balls and are configured to be input for external LO signals or external VCO signals (Refer [External VCO RFPLL](#) chapter).

External LO port can receive a signal between 140MHz and 12GHz through a matched differential impedance of 100Ω and delivers a programmable signal between 70MHz and 6GHz as the LO for transmitters and receivers in the device. The external LO/VCO frequency must be a multiple of 2 times (that is, 2x, 4x, 8x, and so on) of the desired RF signal channel frequency. But for 500MHz to 1000MHz carrier frequency range, the ADRV910X provides special mode of operation for external LO port to work with x1 multiplier in range from 500MHz to 1000MHz.

Singled-ended external LO port is also supported, the matched singled-ended impedance is 50Ω. On-chip duty cycle correction circuit can correct limited range of external LO duty cycle error if it is not 50%. Single ended external LO has limited frequency range, refer [Table 44](#) for more information.

Note, above external LO differential or single ended description is referring the external LO RF input ports, EXT_LO1+/- and EXT_LO2+/- pins respectively. The differential external LO port connection to ADRV910x device is suggested, especially for x1 multiplier external LO mode, the external LO port connection must be differential.

In ADRV910x external LO buffer circuits, there is a configurable "s2dMode" to select the external LO input port phase for internal LO generation. API "adi_adrv910x_cals_InitCalsLoGenParams_Configure" is called to set the s2dMode during the device initialization stage. For TX channel, the "LOGEN_DIFFERENTIAL" mode is helpful to lower the TX spurious (mainly the reference spurs), but this mode needs more stringent LO phase matching on external LO positive port and negative port, especially for lower carrier frequency. Users might choose "LOGEN_DIFFERENTIAL" or "LOGEN_SINGLE_ENDED_POSITIVE_INPUT" for TX channel based on the TX spurious requirement. For RX channel, the "LOGEN_SINGLE_ENDED_POSITIVE_INPUT" is recommended.

TRANSMITTER SIGNAL CHAIN

OVERVIEW

The ADRV910x integrates one direct-conversion (Zero-IF) transmitter and is capable of transmitting narrowband (NB) and wideband (WB) signals. The channel bandwidth can support profiles from several KHz, up to 40MHz, which enables a wide range of applications, such as DMR, P25, and TETRA, etc., as examples of NB standards, and LTE as an example of WB standards.

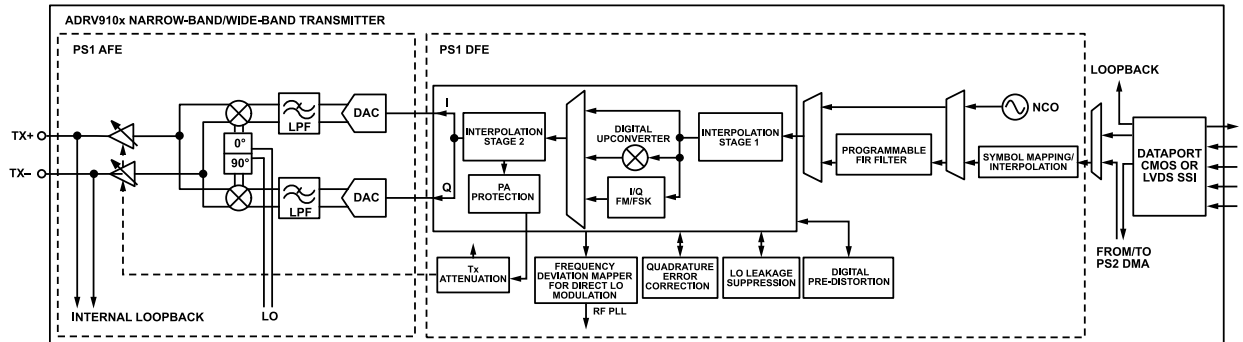


Figure 73. TX Signal Chain and Function Block Diagram

Figure 73 shows the transmitter signal chain and high level function block diagram. The transmitter consists of SSI Data port, Digital Front End (DFE) and Analog Front End (AFE).

The data port supports CMOS or LVDS SSI mode with different lanes configurations and interface rates, it takes the serial data from users' baseband processor via SSI, converts the data to I/Q samples or I only symbols and sends the data to the following TX datapath. For more details, see [Synchronous Serial Interface \(SSI\)](#) session. The data port output can be optionally looped back to RX SSI data port for testing, or sent to the PS2 DMA for further signal processing.

The DFE is composed of independent I and Q signal path with series of interpolation filters and the relative TX signal conditioning functions, such as transmitter attenuation control, power amplifier protection, digital predistortion (DPD), transmitter quadrature error correction (QEC), and transmitter LO leakage (LOL) handling, etc. The optional symbol mapping, programable FIR filters before the datapath interpolation filters is available for some narrowband type signals which allows the users send the I only symbols to ADRV910x and implement the modulation inside the ADRV910x chip.

The Analog Front End is composed of digital-to-analog converters (DAC), analog low pass filters (LPF), up-conversion mixers and the analog attenuator. The DACs are driven by I and Q data samples respectively. The DAC outputs are filtered by the analog LPF and upconverted to the desired RF frequency through the mixer, while the analog attenuator at mixer output is employed to adjust the transmitter output signal power.

The following subsections discuss the major transmitter functionalities.

DIGITAL FRONT END (DFE)

Symbol Mapping With Interpolator

The symbol mapping supports FSK modulation and maps 2 bits symbol of the I interface to one of four configurable values on data path, the symbol mapping also supports QPSK and CQPSK modulation to map the 2 bits to values on both I and Q of the data path. The symbol mapping output can be upsampled with the configurable ratio.

Programmable FIR Filter (PFIR)

There are two PFIR filters (PFIR_I and PFIR_Q) on TX data path, as shown in [Figure 74](#), the PFIRs can be configured to work at several different modes,

- Both PFIR_I and PFIR_Q are bypassed, where the modulated I/Q samples are sent to ADRV910x TX, no extra filtering is needed.
- Both PFIR_I and PFIR_Q are enabled, PFIR_I is for I data path and PFIR_Q is for Q data path. The PFIR works as pulse shaping filter for QPSK/CQPSK modulation symbol mapping output, or other custom filter functions that might be migrated from BBIC to ease the BBIC workload.
- Only PFIR_I is enabled for I data path. The PFIR works as pulse shaping filter for FSK modulation symbol mapping output.
- PFIR_I and PFIR_Q operate sequentially for I data path. an example scenario where the PFIR_I could work as pulse shaping filter for C4FM modulation symbol mapping output, and PFIR_Q is set as invert sinc filter for pulse shaping output.

TRANSMITTER SIGNAL CHAIN

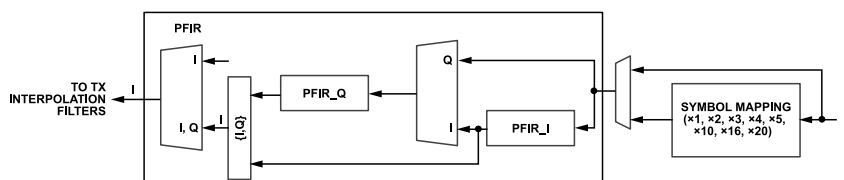


Figure 74. TX Symbol Mapping and Programmable FIR filters

The PFIR has up to 128 taps, each filter tap is 24 bits in width with the signed bit included. Users can load customized PFIR coefficients or bypass the PFIR based on the different applications.

Interpolation Filters

There are a series of interpolation filters (organized into two different decimation stages) in TX signal chain to upsample the input data sample rate to the configured DAC sample rate and filter the imaging artifacts due to the upsampling. The TX signal chain might be set to narrowband mode or wideband mode, the filters combination to accommodate the different profiles will be determined by the Profile Generator (a built-in tool in ADRV910X Transceiver Evaluation Software) before the ADRV910x is initialized.

Attenuation Control

TX Attenuation Table

The transmitter attenuation block controls the TX output power. A TX attenuation table with 960 entries is loaded into the ADRV910x memory during initialization, each entry equals a 0.05 dB attenuation step. Only the first 840 entries in attenuation table are used that gives the up to 42 dB attenuation range. The TX attenuation for each entry is distributed into two portions, analog attenuation portion and digital attenuation portion respectively. In the analog attenuation, there is a digitally controlled step attenuator (DSA) with 64 unit steps on a linear scale. The gaps between the analog gain steps are interpolated by a 12-bit digital multiplier to 0.05 dB steps. The maximum analog attenuation is 36 dB, and the maximum digital attenuation is 6 dB.

Table 18. Sample Rows for Tx Attenuation Table

Tx Attenuation Index	Total Tx Attenuation (dB)	Analog Attenuation Control Word [5:0]	Analog Attenuation (dB)	Digital Attenuation (dB)	Digital Attenuation (linear)	Digital Attenuation Control Word [11:0]
0	0	0	0.00	0.00	1.00	4095
1	0.05	0	0.00	0.05	0.9943	4072
2	0.10	0	0.00	0.10	0.9886	4049
3	0.15	1	0.14	0.01	1.00	4090
4	0.20	1	0.14	0.06	0.9928	4066
...	...	...	...	...	...	...

Table 18 shows the sample rows of the TX attenuation table. The first column is the TX attenuation table index, second column shows the total TX attenuation in dB for the corresponding index, attenuation step size for the adjacent index is 0.05dB. The third column is the control word to calculate the analog attenuation (dB) shown in the fourth column. The equation for this calculation is:

$$\text{Analog Gain (dB)} = 20 \cdot \log_{10}(1 - \text{Analog Attenuation Control Word}/64)$$

The 5th and 6th columns show the required digital attenuation in dB and linear domain respectively to achieve the total attenuation in the second column. The last column stands for the digital attenuation control word to calculate the linear digital attenuation in the sixth column based on the following algorithm:

If *Digital Attenuation Control Word* = 4095

$$\text{Digital Attenuation} = 4096/2^{12}$$

else

$$\text{Digital Attenuation} = \text{Digital Attenuation Control Word}/2^{12}$$

Note, only the third and the seventh columns are actually stored in memory. Other columns shown in Table 18 are only for explanation.

TRANSMITTER SIGNAL CHAIN

The write TX attenuation table is done during the chip initialization stage by API “adi_adrv910x_Tx_AttenuationTable_Write”, and API “adi_adrv910x_Tx_AttenuationTable_Read” can be used to read out the TX attenuation table for verification.

TX Attenuation Mode

There are three modes to control the transmitter attenuation block: bypass, SPI and GPIO. The mode is configured by The API command “adi_adrv910x_Tx_AttenuationMode_Set”.

In the “bypass” mode, the TX attenuation functionality is not used, which means 0 dB total TX attenuation.

In the “SPI” mode, the TX attenuation value is set by API “adi_adrv910x_Tx_Attenuation_Set” directly with the specified attenuation value.

In the “GPIO” mode, two digital GPIO pins are used to increment or decrement the attenuation value. An API command “adi_adrv910x_Tx_Attenuation_PinControl_Configure” configures the digital GPIO pins and sets TX attenuation size for each GPIO pulse.

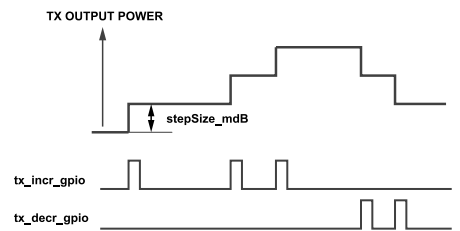


Figure 75. TX Attenuation GPIO Increment/Decrement Mode

TX Power Amplifier Protection

TBD.

TX QEC

In the analog circuitry of a direct-conversion transmitter, there are three major non-idealities: gain variation between the I and Q datapath, phase imbalance (non-90 degrees between LO driving I and Q mixers), and differences in the LPF on I and Q path such as group-delay variations. Without properly correcting them, the output spectrum of the transmitter can significantly degrade due to the undesired images.

Transmitter QEC is designed to estimate and correct the non-idealities through initial and tracking calibrations. The tables are generated to record the initial calibration results, which can be further refined through tracking calibrations on-the-fly. During signal transmission, the mismatch corrections are applied in the transmitter datapath so that the non-idealities are compensated. For more details, see the [Transmitter/Receiver Signal Chain Calibrations](#) section.

TX LOL

Any coupling of the LO to the TX RF output will show up as undesired LO leakage on TX output port which might cause significant TX performance degradation.

TX LOL is designed in the transmit signal chain to correct the LO leakage issue, it estimates the LO leakage through initial and tracking calibrations, and then applies the corrections in the baseband to cancel the undesired tone. For more details, see the [Transmitter/Receiver Signal Chain Calibrations](#) section.

Digital Predistortion (DPD)

DPD is a feature available on the ADRV910x TDD operation mode to achieve higher power amplifier efficiency, while still meeting the error vector magnitude (EVM) and adjacent channel leakage ratio (ACLR) requirements in the transmitter signal chain in compliance with standard requirements. DPD works on the principle of predistorting the transmitter data to cancel distortion caused by power amplifier compression. It uses the tracking calibration to capture the transmitted data samples that looped back through observation receiver to estimate the distortion parameters. By applying the correction in real time, the transmitted signal is predistorted to compensate for the power amplifier nonlinearity. For more details, see the [Digital Predistortion \(DPD\)](#) section.

TRANSMITTER SIGNAL CHAIN

FM/FSK Modulation

The TX signal chain supports frequency modulation (FM) and frequency shift keying (FSK) for standards that use constant-envelope frequency modulation schemes, such as digital mobile radio (DMR), Analog FM, P25 Phase 1, and Phase 2, etc.

ADRV910x TX support two different FM/FSK modulation methods, IQ FM/FSK modulation and Direct FM/FSK modulation.

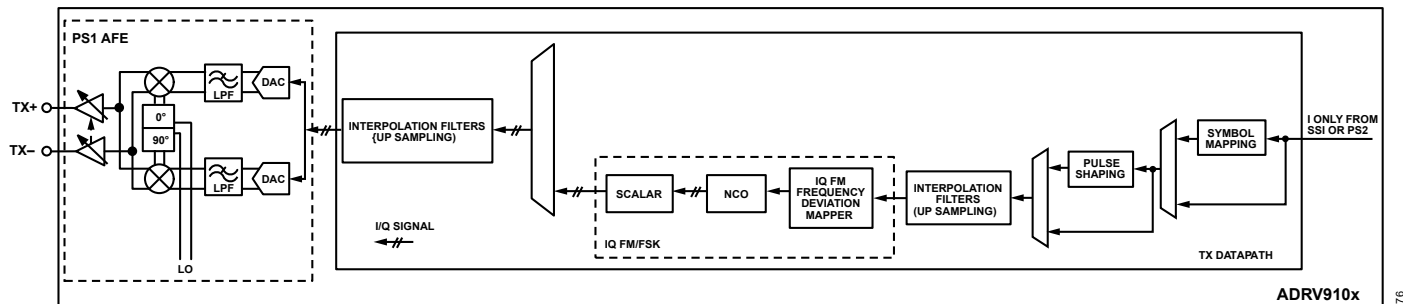


Figure 76. IQ FM/FSK Modulation

Figure 76 shows the function diagram of the IQ FM/FSK modulation inside the ADRV910x TX signal chain. The I only FM/FSK symbols pass through the symbol mapping, pulse shaping, upsampling, the processed data are mapped to frequency tuning words for NCO, where frequency modulation is applied. The modulated I/Q samples are scaled and further up-sampled and sent to DAC, then unconverted to RF carrier frequency using the mixer driven by LO, where the LO could be internal LO, external LO or external VCO.

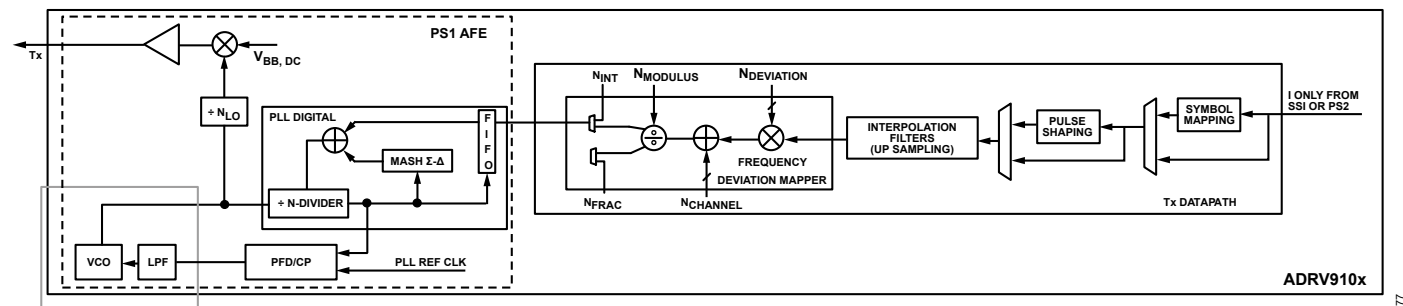


Figure 77. Direct FM/FSK Modulation

Figure 77 shows the function diagram of the direct FM/FSK modulation inside the ADRV910x TX signal chain. The I-only symbols follow similar mapping, shaping, and upsampling, but are then mapped to the equivalent frequency deviation using a frequency deviation mapper. These values pass through a FIFO to align with the PLL reference clock and directly modulate the PLL N-divider. This produces frequency deviations at the LO generator (LOGEN) output, which drives the TX upconverter, resulting in directly modulated RF output signals. The direct FM/FSK modulation can be supported by internal LO (VCO) or external VCO.

When using the external VCO mode, the LPF (PLL loop filter) and VCO in Figure 77 are external to the ADRV910x device, a second external modulation path directly to the VCO is required to meet the FM modulation requirements if the LPF bandwidth is too narrow to support the required FM modulation bandwidth. In this configuration, the ADRV910x internal direct modulation path (via the PLL N-divider) is combined with the external VCO modulation path, forming a two-point modulation scheme, which is commonly used to achieve accurate FM/FSK modulation performance.

For the ADRV910x, IQ FM/FSK and Direct FM/FSK modulation provide equivalent RF performance in principle. However, in external VCO implementations, Direct FM/FSK requires a second modulation path to the VCO (two-point modulation), resulting in higher BOM cost and more complex control.

In IQ FM/FSK modulation mode, ADRV910x TX output requires an external 5th or 7th order low pass filter to suppress the LO harmonics which might degrade the PA output ACPR when PA operates in the high nonlinear or saturated region.

TRANSMITTER SIGNAL CHAIN

TX Frequency Offset Correction

By reusing the NCO and Digital Up Convert (DUC) in [Figure 73](#), the I/Q signals can be added a small frequency offset as a method to correct the small deviations in the transmit LO frequency.

The API “`adi_adrv910x_Tx_FrequencyCorrection_Set`” (TBD) can be used to configure the desired frequency offset in Hz and change the frequency immediately or update it at the start of the next available frame.

ANALOG FRONT END (AFE)

Digital-to-Analog Converter (DAC)

The ADRV910x integrates a 14-bit DAC, which can be operated at a standard rate of 184.32 MHz, 276.48MHz, or 552.96 MHz, for some specified narrow band profiles, the DAC can run at the `DEV_CLK` rate, for example 38.4MHz or 49.152MHz. At the arbitrary sample rate mode, DAC can operate at other different rates as well, the Profile Generator will base on the input sample rate and determine the proper DAC sample rate.

The DAC is auto tuned during the initial calibration stage to remove mismatches in the DAC units, which improves the linearity of the DAC.

Low Pass Filter (LPF)

The analog LPF locates at the DAC output and is used to filter the DAC image signals. The LPF is a second-order Butterworth low pass filter with configurable 3 dB corner frequency. The Profile Generator configures the 3 dB corner frequency based on the profile parameters. The LPF corner frequency, passband flatness and group delay are tuned during the initial calibration stage.

Mixer

The mixer upconverts the baseband signal to RF signal. It is an IQ modulator, which receives a quadrature baseband and LO signal. Due to the image rejection property of IQ modulators, it produces an output only on one side of the LO, i.e., the image is rejected. The LO leakage and quadrature errors of the mixer are calibrated at initialization, and continually tracked by the transmitter LOL and QEC, as discussed.

RF Attenuator

Following the mixer is a digitally controlled step attenuator with 64 linear attenuation steps. This results in a total attenuation range of 36 dB. As discussed in the [Attenuation Control](#), the analog attenuation steps are not linear-in-dB, the extra digital attenuation compensates the analog attenuation step error and provides about 0.05 dB overall attenuation step for each attenuation table entry.

TRANSMIT SIGNAL CHAIN API PROGRAMMING

There are a set of transmitter signal chain APIs for interaction with the ADRV910x device TX. Some of them are discussed in the previous sections, for the full list of available API functions and up-to-date detailed descriptions, refer to the API Doxygen document.

RECEIVER AND OBSERVATION RECEIVER SIGNAL CHAIN

OVERVIEW

The ADRV910x offers dual receive channels. With a minimum number of external components, each receive channel builds a complete RF-to-bits signal chain, which serves as RF front end for a wide range of applications. It supports both the time division duplex (TDD) and frequency division duplex (FDD) modes.

Receiver 1 (RX1) is wide band receive channel can support up to 40MHz RF signal bandwidth, Receiver 2 (RX2) is dedicated narrow band receive channel can support up to 2MHz RF signal bandwidth with optimized low power consumption.

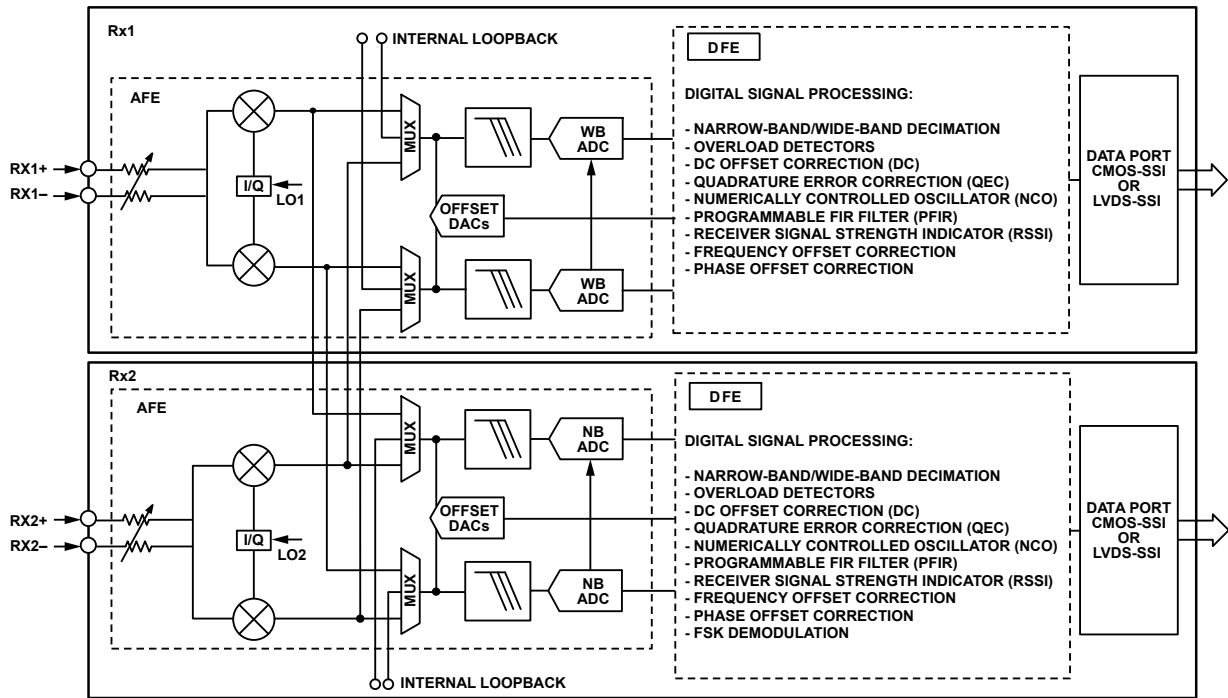


Figure 78. Top Level block diagram of ADRV910x Dual Receiver

Figure 78 shows the top-level architectures of ADRV910x receivers, the RX1 and RX2 path have similar Analog Front End (AFE) subsystem, Digital Front End (DFE) subsystem and the CMOS/LVDS SSI data port.

The AFE subsystem contains five major components: programmable front-end attenuator, matched I and Q mixer, analog mux, low-pass filter (LPF), and analog-to-digital converter (ADC). The attenuators control the signal gain to avoid overloading the datapath when a strong signal interferes. It is followed by the mixers to down convert the received signals for digitization. The analog mux can choose the internal loopback or RX mixer output to the following low pass filter. The output current of the mixers is further converted to voltage and filtered by the LPFs before passing to the ADCs. A pair of wideband ADCs for RX1 and narrowband ADCs for RX2 help to achieve the high linearity and high dynamic range performance of the two receivers.

The DFE subsystem contains a series of digital signal processing components, including decimation filters, digital overload detectors, DC offset correction, quadrature error correction (QEC), digital down conversion (DDC) with numerically controlled oscillator (NCO), a programmable 128-tap FIR filter (PFIR), receiver signal strength indicator (RSSI), frequency offset correction, phase offset correction, etc. The decimation filters decimate the high ADC sample rate to the desired output sample rate and filter the out of channel interferer signals. The overload detectors are used for gain control in the datapath. The DC offset correction, QEC, PFIR, frequency offset correction, phase correction condition the digital signals at different stages of the datapath for the optimal receive performance. The RSSI provides signal power measurement at the end of datapath where indicate the on-channel signal level, it detects the presence of a signal in a desired frequency band, in addition, the RSSI measurement results are used to control the bit-width of the output signal. At the end of the signal chain, through CMOS-SSI or LVDS-SSI data port, the output signal is delivered to the based band processor for further processing. To accommodate the various wideband and narrowband signal processing requirements, the RX2 has extra narrow band demodulation submodules for FSK type signal processing.

Both RX1 and RX2 can support the Zero Intermediate Frequency (ZIF) and Intermediate Frequency (IF) receive architecture. In ZIF mode, the LO frequency is same with the desired receive carrier frequency, the mixer in AFE directly down converts the RF signal to zero frequency. In IF

RECEIVER AND OBSERVATION RECEIVER SIGNAL CHAIN

mode, the LO frequency is offset from the receive carrier frequency, the IF signal at mixer and LPF output is digitalized by ADC, the digital down converter in DFE with the NCO downconverts the IF signal to zero frequency. The IF architecture helps to mitigate the DC offset and the ADC flicker noise affection on the desired signal, especially for the narrow band type signals. Users could enable or disable the IF (offset LO) option for receive path based on the applications.

The RX1 and RX2 has identical front-end attenuator and mixer, the analog mux before the LPF allows signals at RX1/RX2 RF port to be routed to RX2/RX1 baseband circuits (LPF and ADC) and configure the receive path as observation receive path when it's needed in TDD operation mode. In addition, the TX output can be internally looped back to RX path via the analog mux for the TX initial calibration and tracking algorithms.

Figure 79 shows a typical narrow band (NB) TDD application block diagram where the RX2± port is the primary receive port, RX1± port is used as the observation receive port. During the receive slot, the RX2 path works as primary receive. During the transmit slot, the observation channel takes the loopback signal from the transmit path, it might be from external loopback (ELB) path via RX1 RF port, or from the ADRV910X internal loopback (ILB) path, the RX2 AFE analog mux takes the ELB or ILB signal to the RX2 LPF/ADC and DFE for the further signal processing. The RX1 LPF, ADC and DFE is always off for the power saving.

Similarly, Figure 80 shows a typical wide band (WB) TDD application block diagram where the RX1± port is the primary receive port, RX2± port is used as the observation receive port.

There are external loop back path and internal loop back path in Figure 79 and Figure 80. The internal loopback path can be used for TX initial calibrations, the tracking algorithms utilize external loopback (I.E. DPD) or internal loopback path (I.E. TX QEC/LOL) are limited for TDD operation only.

Although, RX1 port can be configured as the RX2 observation receive input port, RX2 port can be configured as the RX1 observation receive input port. ADRV910x doesn't support to use both RX1 and RX2 port as primary port but with different frequency band for one Receive channel.

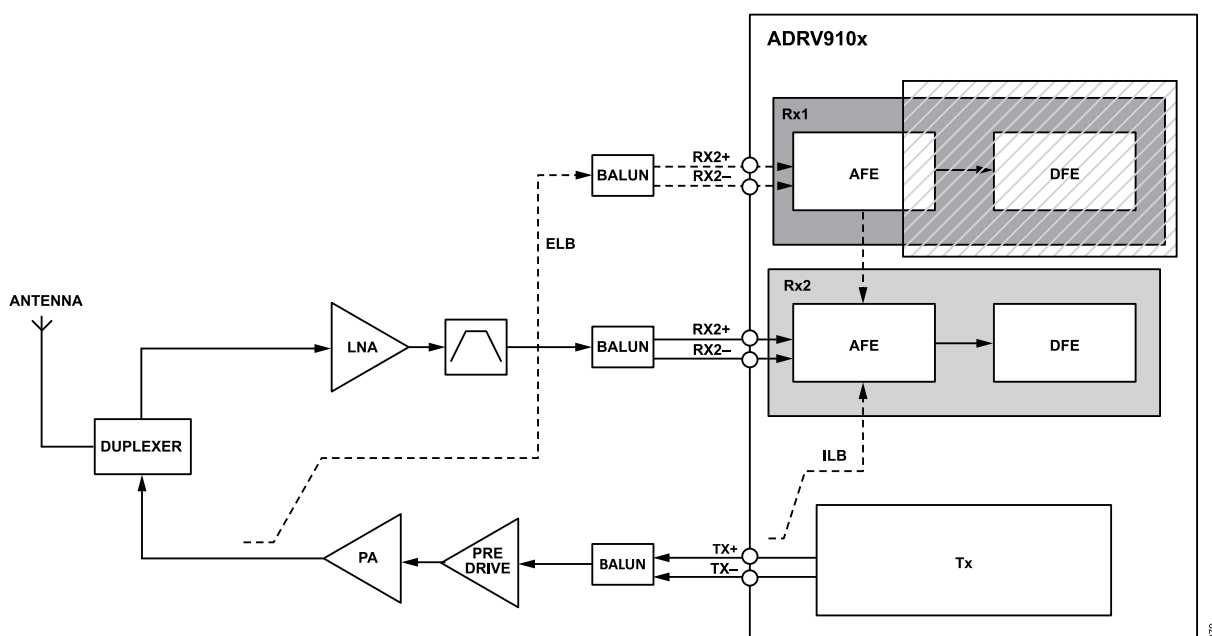


Figure 79. ADRV910x TDD NB RX/ORX/Loopback Diagram

RECEIVER AND OBSERVATION RECEIVER SIGNAL CHAIN

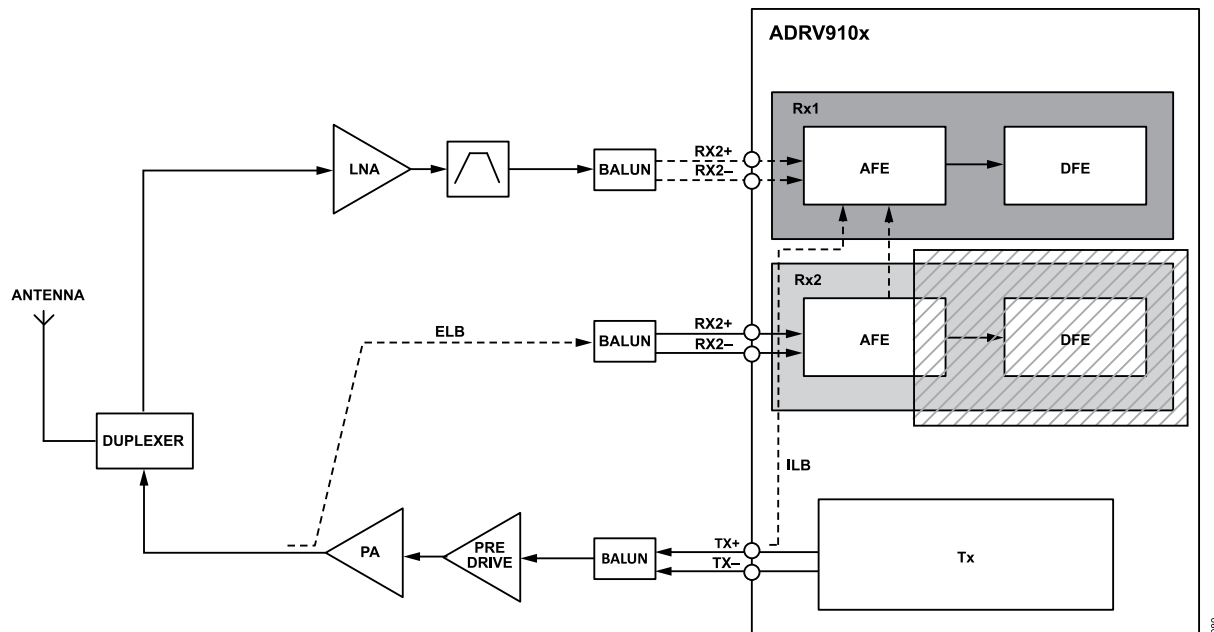


Figure 80. ADRV910x TDD WB RX/ORX/Loopback Diagram

ANALOG FRONT END (AFE) COMPONENTS

Front-end Attenuator

The front-end attenuator is a PI resistive network that provides a constant 100 Ω differential input impedance with the passive mixer. The attenuator has 256 gain settings providing a receiver attenuation range from 0 to $20 \times \log(1/256) = -48$ dB, but the attenuator step size is getting too big for the last few control words (252~255), therefore the range of the attenuation is limited from 0 to -34 dB in ADRV910x device. Calculate the gain of the attenuator by the following equation:

$$\text{ATTEN}_{\text{dB}} =$$

$$20 \times \log_{10} \left[\frac{256 - \text{fe_gain_cw}[7:0]}{256} \right] \quad (1)$$

Where the $\text{fe_gain_cw}[7:0]$ is an 8-bit control word defined in the receiver gain table.

Mixer

The passive down converting mixer follows the RF attenuator. It down converts the RF signal to IF or baseband. The passive mixer uses a non-overlapping $\frac{1}{4}$ duty cycle local oscillator generated by the four phases 50% duty cycle LO. The duty cycle distortion (DCD) circuit controls the non-overlapping time. The DCD is implemented by delaying the rising edge of the 50% duty cycle LO.

Low Pass Filter (LPF)

In the receiver data chain, the LPF sits between the mixer and the ADC converting the baseband signal current to voltage, it functions as receiver baseband filter with programmable cut-off frequencies for ADC anti-aliasing based on the system requirements. The LPF is configured in the transimpedance amplifier (TIA) mode with the single pole in the transfer function, which is denoted as first order LPF. In addition, the LPF provides a static gain of about 20 dB, which is independent of the gain control functionality through the receiver data chain.

RX1 and RX2 have similar LPF function blocks, but their LPF has different tuning bandwidth, the RX1 LPF supports a baseband 3 dB bandwidth of about 6.7MHz to 40MHz, RX2 LPF has narrower tuneable bandwidth, the minimum 3 dB bandwidth is about 1.7MHz and it is the default setting for all RX2 profiles.

The LPF is calibrated during device initialization to ensure a consistent frequency corner across all devices. The 3dB bandwidth is set within the device data structure and is profile dependent. The user could optionally tune the 1 dB/3 dB corner frequency of the LPF based on the application.

RECEIVER AND OBSERVATION RECEIVER SIGNAL CHAIN

Analog to Digital Converter (ADC)

RX1 and RX2 uses different ADCs to support the different maximum RF channel bandwidth, denote as WB ADC and NB ADC respectively. Both WB ADC (RX1) and NB ADC (RX2) are based on the continuous time delta sigma (CTDS) architecture, they have similar noise spectrum density (NSD) and linearity performance but work at different sample rate range to support wider RF bandwidth for RX1 and lower power consumption for RX2. RX1 ADC supports up to 1.1Ghz sample rate, RX2 NB ADC supports sample rate from 33MHz ~ 80MHz which allows to use DEV_CLK directly as the ADC sample clock for the narrow bandwidth profiles.

The RX2 ADC supports two optimization modes:

- Lowest Noise Figure (Option 4): Optimized for minimum receiver noise figure, providing the best sensitivity.
- Highest Blocking Performance (Option 5): Provides approximately +1 dB improved dynamic range at the cost of about +1 dB degradation in noise figure, enhancing blocking rejection performance.

Lowest Noise Figure (Option 4) is the default setting for RX2 channel, Highest Blocking Performance option is limited to profiles with RF channel bandwidths below 25 kHz.

DIGITAL FRONT END (DFE)

Figure 81 shows the RX signal chain and Digital Front End (DFE) block diagram, the DFE follows the Analog Front End output and is composed by series of digital signal processing components. RX1 and RX2 has similar digital function blocks, but RX1 and RX2 deploy different decimation filters designs to accommodate the different ADC sample rate and datapath output rate, there are some function blocks (purple blocks in Figure 81) exist in RX2 only which are designated for FSK demodulation and narrow band type signal processing, more details is introduced in [Narrowband Demodulator Subsystem](#) session.

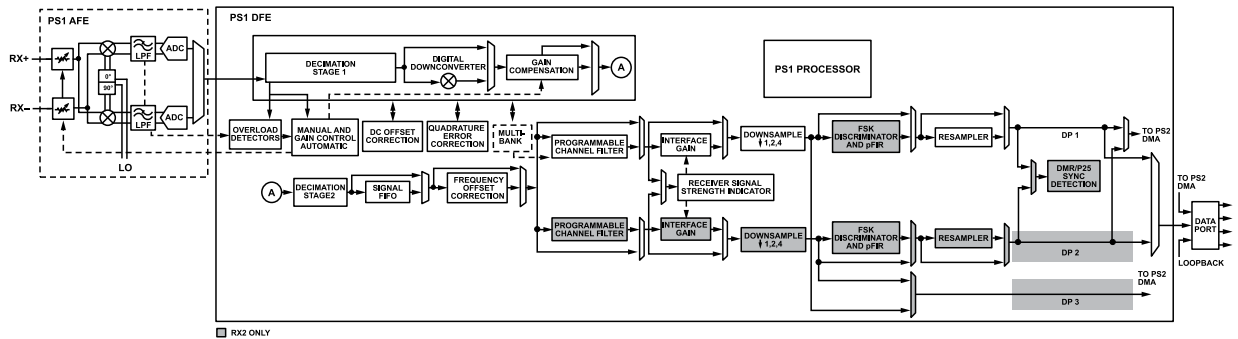


Figure 81. RX Signal Chain and DFE Block Diagram

The following function blocks description are available for both RX1 and RX2 DFE.

Decimation (DEC)

In RX1 data chain, a series of decimation filters (organized into two different decimation stages) convert the RX1 ADC sample rate to a desired sample rate in both the WB and NB modes. Figure 82 shows how to achieve the standard sampling rates for different standards through a flexible combination of decimators in the RX1 signal chain. For simplicity, any other non-DEC blocks are skipped in the diagram.

RECEIVER AND OBSERVATION RECEIVER SIGNAL CHAIN

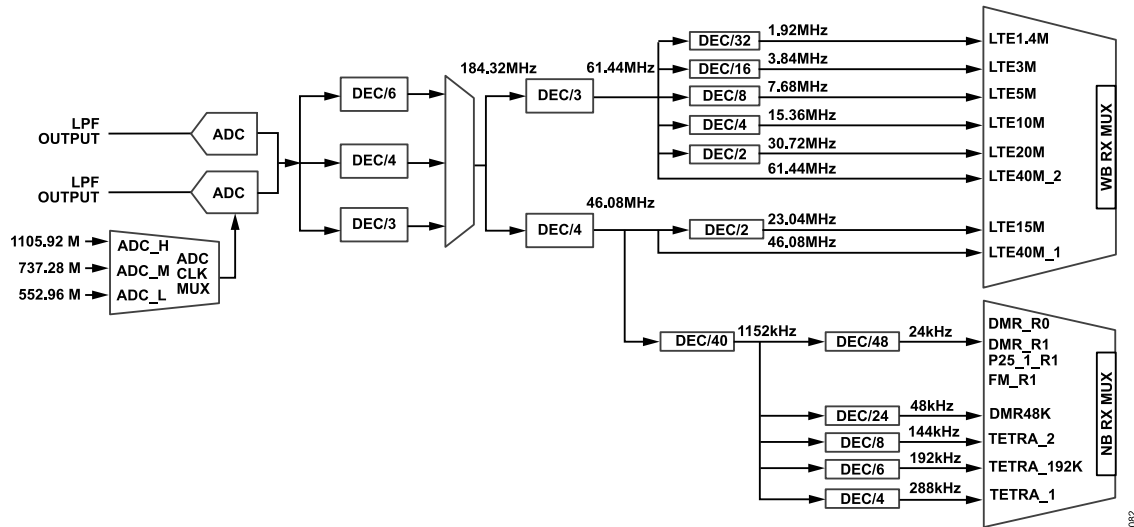


Figure 82. RX1 Decimation Schemes to Support Various Standard Sample Rates

Figure 82 shows that RX1 in the NB and WB mode, three different ADC output sample rates are first decimated to a common rate of 184.32 MHz in standard use cases. Then, it is further converted to two different rates. One is 61.44 MHz for WB mode only and the other is 46.08 MHz for both the NB and WB modes. All LTE standard modes are considered WB and the desired sample rate is further generated from both 61.44 MHz and 46.08 MHz through a decimation rate of 2 to 32. The DMR, FM, P25, and Tetra are NB modes, and the desired sample rate is further generated from 46.08 MHz with a decimation rate of 160 to 1920.

For each decimator shown in Figure 82, there is a combination of lower rate decimators. For example, DEC/40 is implemented as a cascade of DEC/10 and 2 DEC/2 decimators. In addition, the different decimation rates are achieved by strategically enabling and disabling some lower rate decimators. For example, in WB mode, with an initial sample rate of 61.44 MHz, if all lower rate decimators are used, it can achieve a decimation rate of 32. If two of the DEC/2 are disabled, a decimation rate of 8 can be achieved. All the decimation filters are carefully designed to satisfy the system performance requirements.

With the Arbitrary Sample Rate (ASR), the user could get an almost continuous range of sample rates from 24 kHz to 61.44 MHz, except for some “dead zones” due to internal clocking constraints. Achieve this by adjusting the internal CLK PLL frequency as well as a flexible arrangement of decimators.

Similarly, Figure 83 shows the series of decimation filters in RX2 data chain to accommodate the different output sample rate. With the ASR option, users could get sample rates support from 24KHz to 4MHz with some “dead zones” due to internal clocking constraints. RX2 ADC supports 33MHz~80MHz sample rate, the ADC can be driven directly by device clock (refer Clock Generation), the device clock frequency must be integer multiple of the sample rate when using this option.

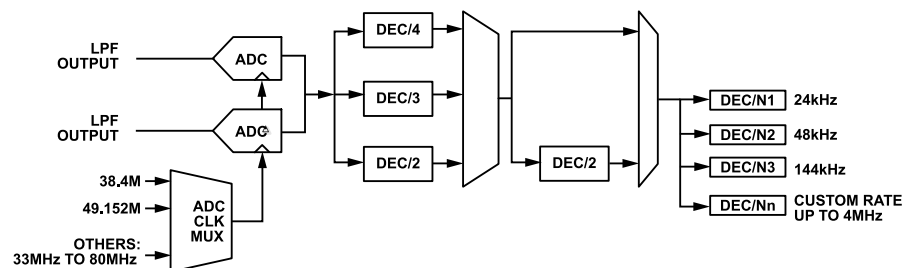


Figure 83. RX2 Decimation Schemes to Support Different Sample Rate

Users could use the ADRV910x Transceiver Evaluation Software (TES) to validate the desired device clock, RF channel bandwidth and the dataport sample rate combination, the profile generator in TES will generate the proper filter configurations if the profile is valid. Users don't need the direct access on the filters configuration.

RECEIVER AND OBSERVATION RECEIVER SIGNAL CHAIN

DC Offset Correction

The source of the DC offset is mainly from the receiver LO leakage caused by the finite isolation between the LO and RF ports of a mixer, which is typical for silicon-based ICs. It generates a high DC component at the center of the desired signal band, especially for ZIF operation. Through the datapath, the induced DC offset is amplified and reduces the ADC dynamic range significantly. In addition to receiver LO leakage, the device mismatch in LPF and ADC also contributes to the DC offset problem. Without properly correcting the DC offset, it can degrade the system performance.

There is two-step approach estimates and corrects the DC offset in ADRV901x. The first step comprises a DC estimation step in the digital domain and a correction procedure in the analog domain, which is called RFDC. The second step is an all-digital DC offset estimation and correction technique that estimates and corrects for any residual DC offset after the first step, which is called BBDC.

In the IF work mode, the carrier is offset LO frequency, the blocker around LO frequency is converted to DC signal at ADC input which could be much bigger than the DC from silicon LO leakage, the bigger blocker DC could overload the RFDC correction algorithm therefore degrade the RX performance. The ADRV901x provides an optional LO Interference Detection (LOID) algorithm (configured by API `adi_adrv910x_Rx_Loid_Configure`), it detects the possible blocker presence around LO frequency, the LOID algorithm can freeze the RFDC tracking for the DC offset correction updates if a blocker is detected and above the configurable threshold, and if a blocker is smaller than the threshold or not detected, the LOID can re-enable the RFDC tracking to restore the DC correction.

Quadrature Error Correction (QEC)

In an ideal analog mixer, the in-phase (I) and quadrature-phase (Q) sinusoidal signals are orthogonal. In addition, the I and Q paths of LPF and ADC should have identical frequency responses. However, in reality, IQ imbalance always exists in the mixer, LPF, and ADC, resulting in quadrature errors, consequently the unexpected image signals. In ZIF direct conversion reception, IQ-imbalance leads to a distortion of the IQ-signals themselves within the respective desired baseband channel. In IF mode, the blocker's image signal might fall in the desired signal band and distort receive performance. The properly quadrature error correction is necessary to achieve the best receiver performance.

In general, quadrature error can be classified as frequency independent error (FIE) and frequency dependent error (FDE). FIE is mainly caused by the mixer I/Q sinusoid mismatch in both gain and phase, while FDE is mainly caused by the inconsistent filter responses.

In ADRV910x receiver, the narrowband and wideband QEC algorithms are developed accordingly to handle quadrature errors in these two modes effectively. Narrow band QEC employs a time-domain adaptive algorithm to estimate both the gain and phase mismatch. Then, estimations are applied to correct the distorted input signal in real-time before passing to digital down converter (DDC). Wideband QEC designs a correction filter to cancel the effect of the mismatch filter by modeling the quadrature error generation as a mismatch filtering process. The correction filter parameters are obtained through the initial calibration by injecting RF tones into the mixer at selected frequencies and then on-the-fly adjustment by processing the receiver data in real-time.

Digital Down Converter (DDC)

DDC is only used when IF reception is employed. By using a programmable NCO configurable from 45 kHz to 21 MHz (Note: RX1 and RX2 has different configurable range), it further converts the IF signal to the baseband.

RX Gain Control

Receivers have automatic and manual gain control modes with the optional gain compensation function, the gain control details are described in [Receiver Gain Control](#) session.

Frequency Offset Correction

In a communication system, the transmitter transmits a desired signal at RF over the air. As the clock reference at the transmitter or the receiver is independent of each other, this can result in the RF carrier frequency offset between the transmitter and the receiver. This frequency difference is called the carrier frequency offset (CFO).

In the receiver data chain, there is a frequency offset correction block to further correct small carrier frequency offsets in both the RX1 and RX2 channel through an API command. The BBIC must estimate and provide the correction value. The correction can occur immediately or relative to the receiver frame boundary.

The frequency offset correction is implemented as a digital down converter (DDC), which consists of an NCO and a complex multiplier in the datapath. As the correction value, ensure the NCO frequency word is in the range of minimum ($\pm 20k$, 20% of sample rate).

RECEIVER AND OBSERVATION RECEIVER SIGNAL CHAIN

The API command `adi_adrv910x_Rx_FrequencyCorrection_Set` is used for the small frequency error correction, provide the frequency correction value at Hz and specify if the correction should take place immediately or at the start of the next available frame.

Programmable Channel Filter (PFIR)

The programmable channel filter is an optional 128-tap programmable FIR filter used in both RX1 and RX2 channel. Each filter tap is 24 bits in width with the signed bit included. The multi banks of customized PFIR coefficients are supported, users might switch the PFIR coefficients on the fly under the control of the baseband processor.

The Receive PFIR is loaded with a customized low-pass filter profile to reduce the adjacent channel interference, which helps in better channel selectivity. For example, as shown in [Figure 84](#), before the CFO is corrected, the Baseband Processor can program a loose filter profile onto the receiver programmable FIR to perform common filtering. However, after the CFO is corrected through the frequency offset correction block, a tight filter profile can be loaded to perform the deep channel selection filtering.

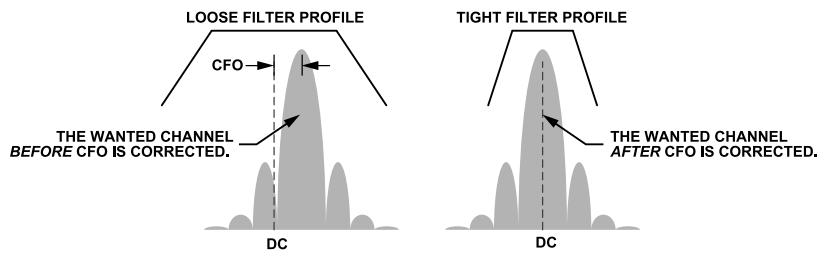


Figure 84. Loose Filter Profile vs. Tight Filter Profile

Receiver Signal Strength Indicator (RSSI)

RSSI measures the received signal power over a period of time, which can be employed to calculate the interface gain to avoid saturating the data port. In addition, in the Monitor Mode, it performs signal detection in WB applications and works together with the FSK Discrimination block to detect signals in NB applications.

An API command `adi_adrv910x_Rx_Rssi_Read()` retrieves the measured signal level. The API function reads back the RSSI status for the given receiver channel and is called any time after the device is fully initialized.

Interface Gain

Due to the bit-width limitation of the data port, an interface gain is applied by shifting the signal properly so as not to clip the output upon saturation. It increases the signal level for small signals to avoid losing sensitivity when a quantizer is used to limit the receiver output data bit-width. The interface gain is automatically adjusted internally inside the device by utilizing the RSSI measurement or through API commands.

Optionally, retrieve the signal level measured by the RSSI through an API command to control the interface gain. For more details, see the [Digital Gain Control and Interface Gain](#) section.

DownSample

There is optional downsample function offers extra decimation by 2 or 4 at the end of RX data path, but this block is only for the down sample to lower the RX data path sample rate to BBIC, there is no decimation filter inside this block, carefully considering the PFIR cut off frequency to avoid the spectrum aliasing when enable the DownSample function.

Resampler

Resampler is an optional block to adjust the sampling phase of the IQ or FD signal, it can be bypassed.

The resampler resamples the incoming received signal by reconstructing intermediate samples between every two input samples according to the resample phase parameter, μ , where $\mu \leq 0.5$.

In the frequency domain, the ideal digital resampler has the transfer function, as shown in the following equation:

$$H_{ideal}(j\omega, \mu) = \begin{cases} T_s \cdot e^{j\omega\mu T_s}, & |\omega/2\pi| \leq B_x \\ \text{don't care}, & |\omega/2\pi| > B_x \end{cases} \quad (2)$$

Where:

RECEIVER AND OBSERVATION RECEIVER SIGNAL CHAIN

B_x is the maximum band width of the input signal.

Filtered by the ideal resampling filter, the input signal $x(kT_s)$ is converted to the output signal $y(kT_s) = x((k + \mu) \cdot T_s)$.

Figure 85 illustrates the functional diagram of the Resampler.

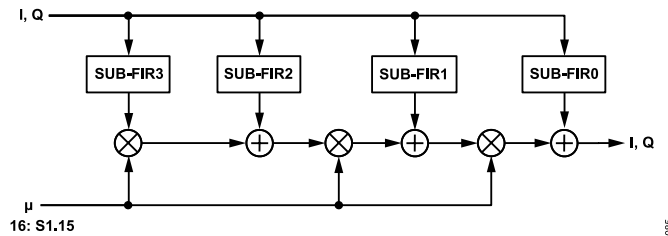


Figure 85. Functional Diagram of the Resampler

Scanning the sampling phase μ from 0 to 0.5, the maximum magnitude difference of the frequency transfer function between the Resampler and the ideal digital resampler is collected and plotted at the upside of Figure 86. The maximum phase error is collected and plotted at the downside of Figure 86.

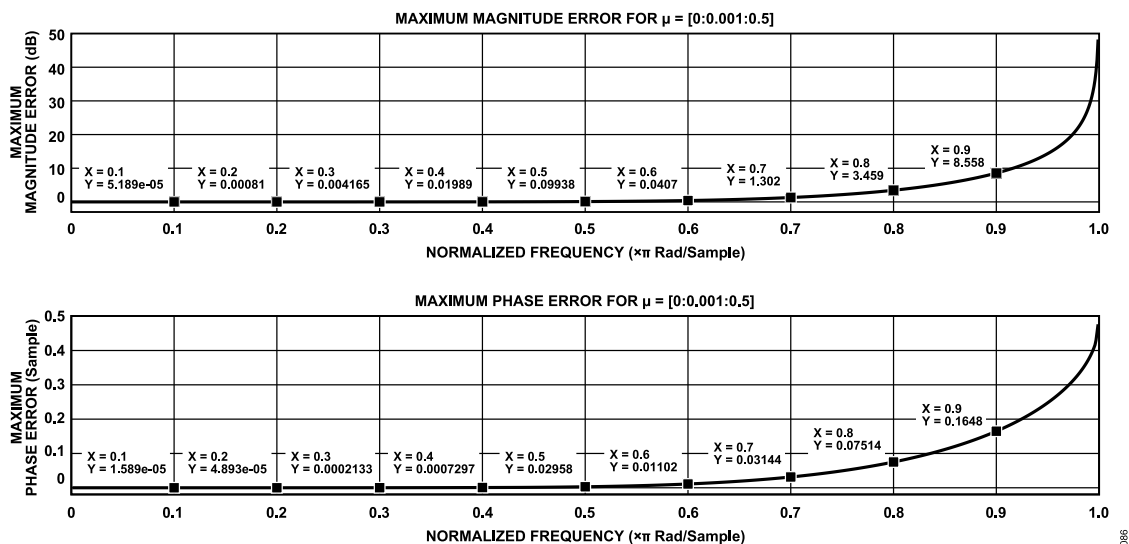


Figure 86. Max magnitude/phase error of resampler

NARROWBAND DEMODULATOR SUBSYSTEM

Narrowband demodulator subsystem, denoted by rx2nbdem is the digital baseband backend partition of RX2 channel. Note that the “narrowband” is commonly for wireless communication systems. If the channel spacing (also known as channel bandwidth) is 1 MHz or less, it is known as a “narrowband system”, otherwise it is called a “wideband system”.

Figure 87 shows the function diagram of rx2nbdem subsystem, incorporating signal buffering, frequency offset correction, programmable channel filtering, interface gain, extra down sampling, frequency discrimination, programmable pulse shaping, resampling and the SYNC detection function, etc.

RECEIVER AND OBSERVATION RECEIVER SIGNAL CHAIN

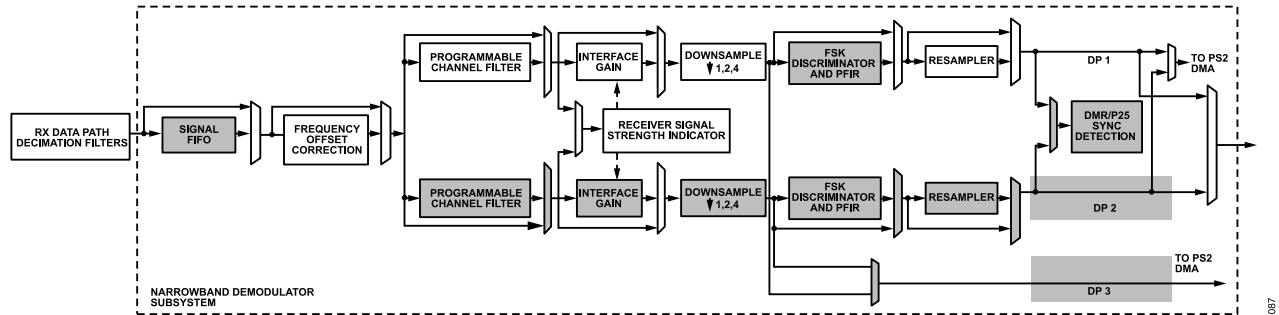


Figure 87. RX2 Narrowband Demodulator Subsystem Block Diagram

The input of rx2nbdem, driven by the RX2 data path decimation filters, is the ZIF digital baseband IQ signal. Programmability exists to bypass each block in the rx2nbdem subsystem. Two identical programmable channel filter and FSK demodulator datapath (dp1 and dp2) provides the capability to detect two different types of signals simultaneously, for example, dp1 is set to detect/receive DMR signal, dp2 is set to detect/receive analog FM signal at same time, and the programmable channel filter in dp1 and dp2 can be loaded with different PFIR coefficients accordingly. Either dp1 or dp2 frequency deviation output can be optionally sent to PS2 DMA for the further signal processing. The additional dp3 is configured to choose either the dp1 or dp2 I/Q samples, and optionally send out the I/Q samples to PS2 DMA for the further signal processing.

The output of rx2nbdem, directly couples to the RX SSI interface. Depending on the programmed functionality, the output can be dp1 or dp2 Frequency Discriminator output (FD and Mage2, refer [Frequency Discriminator](#) section), or I/Q samples from dp1 or dp2.

Signal FIFO

Signal FIFO buffers the input IQ data stream, the Signal FIFO depth is 2048. As a result, it can store for more than 85 ms at the sampling rate of 24Ksps.

Disable or enable the Signal FIFO based on requirements. As the Signal FIFO is disabled, this block is bypassed and cannot be written or read. As the Signal FIFO is enabled, the writing and reading control of the Signal FIFO can be manipulated separately. The FIFO reading clock is configurable and can be 1x, 2x, 4x, or 8x of the FIFO writing clock.

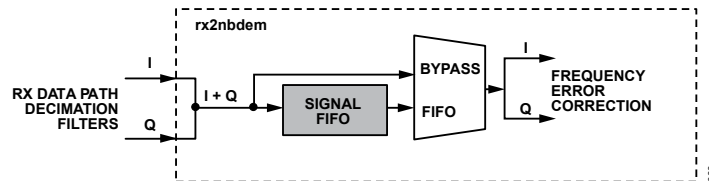


Figure 88. Functional Diagram of Signal FIFO

In the Signal FIFO, as shown in [Figure 88](#), there is an output mux, which has two inputs: "Bypass" is driven by the input IQ stream to the FIFO, and "FIFO" is driven by the output IQ stream from the FIFO. The mux inputs can be switched on demand to drive the following modules in rx2nbdem.

The following example explains how to use the Signal FIFO.

Keep the mux at "Bypass" during the signal capturing phase before the wireless data link is established. The "FIFO" writing control is enabled and reading control is disabled. The Baseband Processor (could be ADRV910x PS2 or external host processor) can keep on detecting the received signal. Meanwhile, the Signal FIFO keeps on buffering the IQ stream. The FIFO writing overflow may or may not happen. If this happens, the FIFO always stores the latest 2048 IQ data.

As the Baseband Processor detects the desired receiver frame from the input data stream and estimates the right starting point of the wanted receiver frame, and further the synchronous parameters, the Baseband Processor can switch the mux from "Bypass" to "FIFO," then enable the reading control of the FIFO, to process the stored data stream and the following data stream seamlessly.

RECEIVER AND OBSERVATION RECEIVER SIGNAL CHAIN

Frequency Offset Correction

The Frequency Offset Correction or Carrier Frequency corrector (CFC) in rx2nbdem removes the carrier frequency offset. More details are in [Digital Front End \(DFE\)/ Frequency Offset Correction](#).

Programmable Channel Filter (PFIR)

As introduced in [Digital Front End \(DFE\)/ Programmable Channel Filter \(PFIR\)](#), the PFIR with customized low-pass filter profile to help the better channel selectivity. In rx2nbdem subsystem, the channel 1 and channel 2 has independent PFIR profiles that allow the different PFIR configurations for the two channels respectively.

Frequency Discriminator

The frequency discriminator in rx2nbdem translates the IQ signal into the frequency deviation (FD) signal, performing the frequency demodulation in the digital domain.

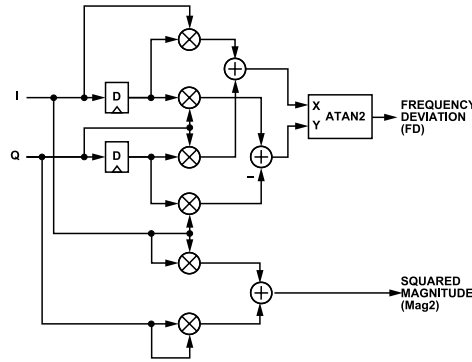


Figure 89. Functional Diagram of Frequency Discriminator

Figure 89 shows that the frequency discriminator outputs the transient frequency deviation (FD) and the transient squared magnitude (Mag2) sample by sample. The output FD and the output Mag2 are defined as the following:

$$FD(n) = \frac{1}{\pi} \text{atan2}[I(n-D)Q(n) - I(n)Q(n-D), I(n)I(n-D) + Q(n)Q(n-D)] \quad (3)$$

$$Mag2(n) = I(n)^2 + Q(n)^2 \quad (4)$$

where atan2 is same as the function in [Octave](#), and D is the programmable delay. Typically,

D is chosen as '1', which means 1 sampling clock delay. (5)

Assuming the input IQ signal is the complex single tone, given by

$$A \cdot e^{\frac{j2\pi f_t n}{f_s}}, \quad n = 0, 1, 2, \dots,$$

Where:

A is the signal magnitude,

f_t is the tone frequency,

f_s is the sampling frequency.

The output FD of the Frequency Discriminator will be $D \cdot 2f_t/f_s$ while the output Mag2 is A^2 .

Programmable Pulse Filter

The optional pulse shaping filtering or low pass filtering is at the output of the frequency discriminator. It's implemented with a 128 taps programmable FIR filter, each tap is 24 bits in width, including the sign-bit, and can load only one set of customized FIR profiles.

RECEIVER AND OBSERVATION RECEIVER SIGNAL CHAIN

SYNC Detection

In LMR waveforms such as DMR, P25, NXDN, and TETRA, the midamble/SYNC field is a deterministic symbol sequence embedded in the burst specifically for synchronization, timing alignment, carrier recovery assistance, and burst identification.

ADRV910x supports the SYNC detection with a "Norm Error" based algorithm. The norm error algorithm determines a sync detection based on the norm error between the received signal and the given sync pattern, the value that the norm calculation must be less than for a sync detection peak to be flagged. The lower this value, the less false detects that will happen, but more misses will happen. The norm error algorithm will also get more misses as the input power decreases, but very rarely false detects, and has a more consistent performance with impairments.

The Correlator in rx2nbдем subsystem performs the correlation operations for SYNC detection, the correlator has two input stream that take the frequency discriminator outputs, the frequency deviation (FD) and the squared magnitude (Mag2) respectively. There are up to 14 correlation processors operate in parallel in Correlator, each correlation processor performs the cross-correlation operations between the FD input stream and the local Sync codes, each correlation processor can be independently loaded one SYNC code and enabled/disabled. For each one FD input, up to 14 cross-correlation output (fd_corr[13:0]) are generated by the 14 correlation processors. Besides the 14 correlation processors, there is a moving average in Correlator, the moving average sums up the continuous FD inputs and generate the fd_mean output. In parallel, 24 samples of input Mag2 are summed up to generate a power sum, denoted as mag_corr which is the corresponding confidence factor for the 14 correlation processor output. Therefore, the Correlator can have 16 outputs at most in parallel, which are fd_corr[13:0], fd_mean, mag_corr. All or the 16 outputs are sent to ADRV910x PS1 processor for the further SYNC detection determination.

ADRV910x SYNC detection support the typical LMR standards, including DMR, NXDN, P25, TETRA I. API "adi_adrv910x_syncDetect_Vector_Configure" and "adi_adrv910x_syncDetect_SyncSearch_Configure" configures SYNC pattern and the detect parameters for the algorithm. Data structure "adi_adrv910x_SyncDetect_SyncSearchCfg_t" defines the parameters for SYNC detect algorithm, as shown in [Table 19](#).

Table 19. adi_adrv910x_SyncDetect_SyncSearchCfg_t Structure Definition

Parameter	Description	Comments
pathDelay	Path delay from RxDmrPd calibration, 0-2047 samples	
powerThreshold_dBm	S9.23, Power threshold in dBm, expected negative value	Power level refers to ADRV910x RX input port. The received power in the interface must be above for the algorithms to consider the possibility of a sync.
detectMultiplier	U1.31, Minimum correlation detection multiplier, ranging from 0 to 1	Norm result multiply factor for the SYNC determination "threshold". Optimal value of 0.35 from test.
preCount	Number of samples power must be above threshold for sync to be detected	Default : 120
postCount	Number of samples power must be above threshold after metric meets power threshold	Default : 120
algoMode	Mode of SYNC Search algorithm, 0-DMR 1-P25 2-NXDN and 3-TETRA1	
dataPath	Data path selection for sync detection	Only support on RX2 DP1

RX2 Narrowband Demodulator Subsystem Configuration in TES

[Figure 90](#) shows rx2nbдем subsystem configuration options in TES.

RX2 Datapath 1 (DP1) and Datapath 2 (DP2) demodulation enable (enable the Frequency Discriminator) can be controlled independently and followed by the optional pulse shaping filter (PFIR), the pulse shaping filter has pre-set coefficients for different standards and supports custom coefficients.

RX2 SSI output data source can be from "RX2 DP1" or RX2 DP2". The RX2 "Signal Type" sets the signal type on SSI output, it could be I/Q data or I only data (FD symbols) depends on whether DP1/DP2's demodulation (frequency discriminator) is enabled. For example, if the "SSI Data Source" is set with "RX2 DP1", [Table 20](#) shows the signals type on SSI output with different settings combination.

"Frequency Deviation" and "Symbol Rate" are radio standard related parameters for "SYNC Detection", for example, 648Hz frequency deviation and 4.8KHz symbol rate for "DMR" radio.

"Sample Delay" is the programmable delay parameter in [Frequency Discriminator](#), set to 1 for frequency modulation (DMR, P25, NXDN), Phase Modulation (TETRA) using a delay of the oversample rate.

RECEIVER AND OBSERVATION RECEIVER SIGNAL CHAIN

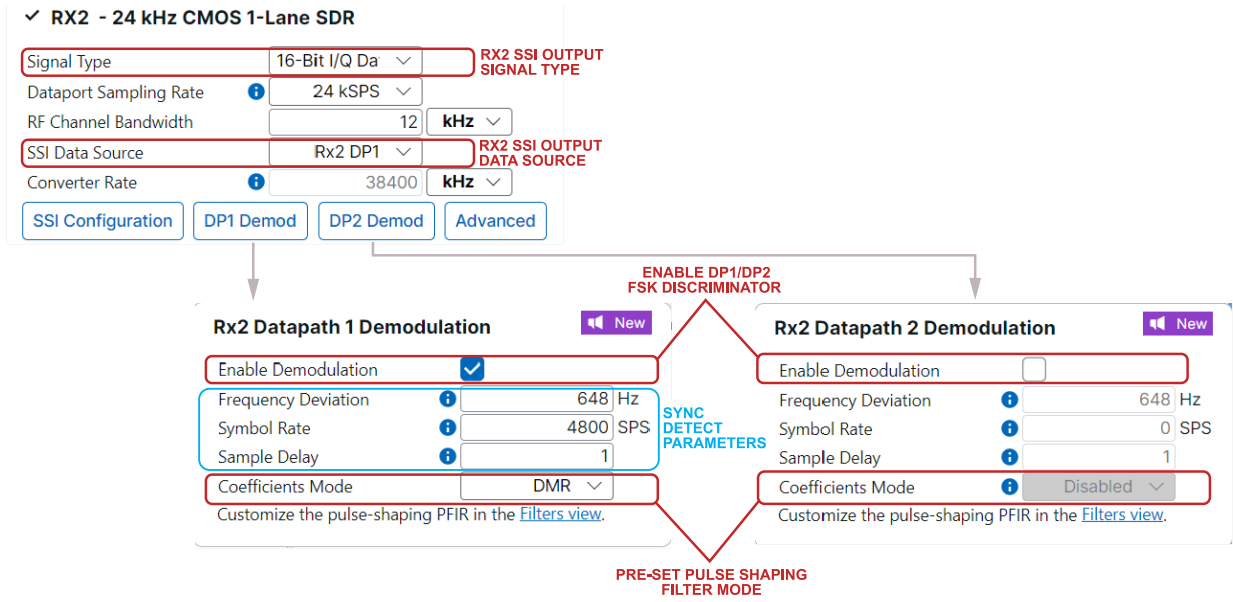


Figure 90. Rx2nbdem Subsystem DP1/DP2 Configuration and Rx2 SSI Settings in TES

Table 20. RX2 SSI Output Signal Type at different Settings Combination

TES Settings		
"Signal Type"	RX2 DP1 "Enable Demodulation"	Signals Output Over SSI
16-bit IQ	Yes	FD (I) and Mag2 (Q) (refer Frequency Discriminator)
16-bit IQ	No	I/Q samples
Frequency Deviation (I only)	Yes	FD (I)
Frequency Deviation (I only)	No	Invalid (only I data of samples are sent over SSI)

RECEIVER SIGNAL CHAIN API PROGRAMMING

There are a set of receiver signal chain APIs for interaction with the ADRV910x device RX. Some of them are discussed in the previous sections. For the full list of available API functions and detailed descriptions, refer to the API Doxygen document.

TRANSMITTER/RECEIVER SIGNAL CHAIN CALIBRATIONS

In the ADRV910x, to achieve optimal performance, an PS1 ARM performs calibrations, which are classified into two categories: initial calibrations performed at initialization before the device is operational and tracking calibrations performed regularly while the device is operational.

Initial calibrations are considered a part of the device initialization, which moves the device from a “STANDBY” state to a “CALIBRATED” state to prepare for transmit/receive operations. Tracking calibrations are performed regularly on-the-fly while the device is operational to track the changes such as attenuation/gain, temperature, and so on.

Most initial calibrations use internally generated tones or wideband signals for calibration, which requires users to satisfy external system requirements. Later sections discuss this topic in more detail. Different from initial calibrations, tracking calibrations use real-time traffic data for calibration. Therefore, tracking calibrations are transparent and do not require any user intervention. The ADRV910x PS1 ARM schedules and performs both initial and tracking calibrations.

Note, the transmitter tracking calibrations will need the internal or external loopback path which is part of RX datapath, therefore, all the TX tracking algorithms are available in TDD mode only.

INITIAL CALIBRATIONS

There are three types of initial calibrations:

- ▶ System initial calibrations
 - ▶ Reserved bit mask for non-channel related initial calibrations, not used.
- ▶ Transmitter initial calibrations
 - ▶ Quadrature error correction (QEC)
 - ▶ Local oscillator (LO) leakage
 - ▶ Loop back path delay (LB PD)
 - ▶ Duty cycle correction (DCC)
 - ▶ Baseband analog filter (BBAF)
 - ▶ Baseband analog filter-group delay (BBAF GD)
 - ▶ Attenuation delay (ATTEN DELAY)
 - ▶ Digital-to-analog converter (DAC)
 - ▶ Path Delay
- ▶ Receiver initial calibrations
 - ▶ WB High Performance ADC resistor/capacitor (HP ADC RC) (reserved, run automatically in analog initialize)
 - ▶ WB High Performance ADC flash offset (HP ADC flash)
 - ▶ WB High Performance ADC DAC (HP ADC DAC) (reserved, not used in ADRV910x)
 - ▶ Duty cycle correction (DCC)
 - ▶ Low power ADC (LP ADC) (reserved, not used in ADRV910x)
 - ▶ Transimpedance amplifier cutoff frequency (TIA Cutoff)
 - ▶ TIA group delay
 - ▶ Quadrature error correction Tone calibration
 - ▶ Frequency independent quadrature error
 - ▶ Internal loop back LO delay (reserved, not used in ADRV910x)
 - ▶ RF DC offset (RF DC)
 - ▶ Gain path delay
 - ▶ DMR path delay

To successfully perform all the initial calibrations, configure the ADRV910x device properly. It is fully controlled by the ADRV910x PS1 ARM. Therefore, no user interaction is required. However, there are requirements for the external system as well. This will be discussed in the later sections.

TRANSMITTER/RECEIVER SIGNAL CHAIN CALIBRATIONS

Initial Calibrations API Programming

The ADRV910x PS1 ARM in the device schedules/performs initial calibrations to optimize performance before device operation. The top-level API function `adi_adrv910x_cals_InitCals_Run()` is used to perform initial calibrations.

The initial calibration is based on the initial calibration configuration defined by `initCals_t` data structure. In this structure, “`sysInitCalMask`” is the initial calibration mask for system calibrations. “`chanInitCalMask[]`” is an array containing a calibration bit mask for channel related initial calibrations, (`chanInitCalMask[0]` is the mask for Rx1/TxWB channels, `chanInitCalMask[1]` is the mask for Rx2/TxNB channels), the TxWB is denoted for TX channel at wideband profile mode, TxNB is for TX channel in narrow band profile mode.

All the initial calibrations are defined in the enumerator “`initCalibrations_e`”. “`calMode`” specifies the mode to run the desired initial calibration algorithms, and “`force`” is a flag, a value of true will force all enabled calibrations to rerun.

The “`initCalMode_e`” enumerator defines the operating modes for initial calibrations. `INIT_CAL_MODE_ALL` runs all the selected initial calibrations, including both receiver and transmitter initial calibrations. `INIT_CAL_MODE_SYSTEM_AND_RX` runs the system and selected receiver initial calibrations (non-loopback paths), and `INIT_CAL_MODE_LOOPBACK_AND_TX` runs the loopback paths (both internal and external loopback paths) and the selected transmitter initial calibrations. `INIT_CAL_MODE_ELB_ONLY` runs all the initial calibrations on external loopback paths only. Do not explicitly use this mode.

[Table 21](#) describes the mask bit assignment for initial calibrations in `InitCalibrations_e`. It also explains the functionality of each initial calibration.

Table 21. Initial Calibration Mask Bit Assignments

Bits	Corresponding Enum	Calibration	Description
D0	INIT_CAL_TX_QEC	Transmitter QEC Initial Calibration	Performs an initial QEC calibration for frequency independent errors for the transmitter path. It estimates the gain and phase mismatch and applies the gain mismatch in the digital domain. Currently, it uses the transmitter path and an ILB path.
D1	INIT_CAL_TX_LO_LEAKAGE	Transmitter LO leakageInitial Calibration	Performs an initial LOL calibration. It estimates the LO leakage and applies the cancellation in the digital domain. Currently, it uses the transmitter path and an ILB path.
D2	INIT_CAL_TX_LB_PD	Transmitter Loop Back Path Delay Calibration	Calibrates the internal transmitter loop back path delay (ILB PD). This information is required to calibrate QEC and LOL.
D3	INIT_CAL_TX_DCC	Transmitter EXT LO/VCO DCC Initial Calibration	Corrects the 50% duty cycle for external LO/VCO when the LO divisor is 1 or 2.
D4	INIT_CAL_TX_BBAF	Transmitter BBAF Initial Calibration	Tunes the low-pass corner frequency and the pass-band flatness of the transmitter baseband analog filter.
D5	INIT_CAL_TX_BBAF_GD	Transmitter BBAF-GD Initial Calibration	Estimates and corrects the filter group delay to remove frequency dependent quadrature error between the I and Q channels in each transmitter.
D6	INIT_CAL_TX_ATTEN_DELAY	Transmitter ATTD Initial Calibration	Estimates the delay between the transmitter analog attenuation and digital attenuation. It is independent of TX attenuation value and frequency region. Perform the calibration only on a single channel.
D7	INIT_CAL_TX_DAC	Transmitter DAC Initial Calibration	Calibrates the DAC for the required profile bandwidth.
D8	INIT_CAL_TX_PATH_DELAY	Transmitter Path Delay Initial Calibration	Estimates the delay between the transmitter input and Tx output.
D9	INIT_CAL_RX_HPADC_RC	Receiver HP ADC RC Initial Calibration	Determines how much the unit R and C vary from the ideal and then tunes the HP ADC's programmable Rs and Cs to obtain the desired values. Without this calibration, the HP ADC's noise performance is negatively impacted. The HP ADC becomes unstable. Only available for RX1 Channel.
D10	INIT_CAL_RX_HPADC_FLASH	Receiver HP ADC Flash Offset Initial Calibration	Optimizes the HP ADC output noise by correcting comparator offsets in the backend flash. Only available for RX1 Channel.
D11	INIT_CAL_RX_HPADC_DAC	Receiver HP ADC DAC Initial Calibration	Corrects for element mismatch HP ADC current. Only available for RX1 Channel and is disabled by default – uncalibrated performance sufficient.
D12	INIT_CAL_RX_DCC	Receiver EXT LO/VCO DCC Initial Calibration	Corrects the 50% duty cycle for external LO/VCO when the LO divisor is 1 or 2.
D13	INIT_CAL_RX_LPADC	Reserved	Not used
D14	INIT_CAL_RX_TIA_CUTOFF	Receiver TIA Cutoff Initial Calibration	Tunes the 3 dB cut-off frequency of the TIA filter.

TRANSMITTER/RECEIVER SIGNAL CHAIN CALIBRATIONS

Table 21. Initial Calibration Mask Bit Assignments (Continued)

Bits	Corresponding Enum	Calibration	Description
D15	INIT_CAL_RX_GROUP_DELAY	Receiver TIA Fine Initial Calibration	Compensates the mismatch in 3 dB cut-off frequency between I and Q path. Corrects quadrature error in the analog domain, which simplifies the correction in the digital domain.
D16	INIT_CAL_RX_QEC_TCAL	Receiver QEC Tone Initial Calibration	Performs an initial QEC calibration in frequency dependent quadrature errors.
D17	INIT_CAL_RX_QEC_FIC	Receiver QEC Frequency Independent Error Initial Calibration	Performs an initial QEC calibration for frequency independent errors for both narrowband and wideband systems.
D18	INIT_CAL_RX_QEC_ILB_LO_DELAY	Receiver ILB LO Delay Initial Calibration	Adjusts the analog delay between the inphase and quadrature LO components at the mixer on the internal loopback path. This is not enabled currently.
D19	INIT_CAL_RX_RF_DC_OFFSET	Receiver RFDC Offset Initial Calibration	Mitigates the RFDC offset added due to LO self-mixing.
D20	INIT_CAL_RX_GAIN_PATH_DELAY	Receiver Gain Path Delay Initial Calibration	Calculates the path delay between the receiver analog and digital attenuation blocks. This delay is then used to offset the onset of receiver analog and digital attenuations relative to each other to compensate for the path delay between these blocks. It is independent of gain index and frequency region. Perform the calibration only on a single channel.
D21	INIT_CAL_RX_DMR_PATH_DELAY	Receiver DMR PathDelay Initial Calibration	Measures the delay from the internal FIFO to sync detection time needed in the DMR and P25 monitor modes.
D22	INIT_CAL_PLL	PLL Initial Calibration	Performs VCO frequency calibration, VCO real-time temperature/aging calibration, and charge pump calibration to make RF PLL ready for operation. This is not enabled currently.
D23	INIT_CAL_AUX_PLL	Reserved	Not Used.

The ADRV910x PS1 ARM proceeds through the calibrations in the required sequential order. The system's initial calibrations are performed first, followed by receiver initial calibrations, and then transmitter initial calibrations. [Table 22](#) and [Table 23](#) show the receiver initial calibration order, and the transmitter initial calibration order, respectively.

Table 22. Rx Initial Calibration Order

Order	Rx Initial Calibrations
1	RX_HPADC_RC
2	RX_HPADC_FLASH
3	RX_HPADC_DAC
4	-
5	RX_TIA_CUTOFF
6	RX_DCC
7	RX_GROUP_DELAY
8	RX_RF_DC_OFFSET
9	RX_QEC_ILB_LO_DELAY
10	RX_QEC_TCAL
11	RX_QEC_FIC
12	RX_GAIN_PATH_DELAY
13	RX_DMR_PATH_DELAY

Table 23. Tx Initial Calibration Order

Order	Tx Initial Calibrations
1	TX_DCC
2	TX_BBAF
3	TX_DAC
4	TX_LB_PD
5	TX_LO_LEAKAGE
6	TX_QEC
7	TX_BBAF_GD

TRANSMITTER/RECEIVER SIGNAL CHAIN CALIBRATIONS

Table 23. Tx Initial Calibration Order (Continued)

Order	Tx Initial Calibrations
8	TX_ATTEN_DELAY
9	TX_PATH_DELAY

The calibration order is mostly determined by the algorithm dependency. It is important to wait for these calibrations to complete successfully before continuing with the other steps of initialization for the device.

System Considerations for Transmitter Initial Calibrations

In this section, high level block diagrams show the device configurations and external system requirements for some transmitter initial calibrations. In all the diagrams, grayed-out lines and blocks are not active in the calibration. Note that as the ADRV910x PS1 ARM performs each of the calibrations, it configures the ADRV910x device per the following diagrams, with respect to enabling/disabling paths, and so on, NO user input is required in this regard. However, it is important that external system conditions should be met.

Among nine transmitter initial calibrations, except for TX_DAC, all other eight calibrations require to insert the tone/wideband signal into the transmitter datapath from an internal signal generator. Therefore, the internal microprocessor disables the data port to avoid interference. Some calibration algorithms, such as TX_LB_PD, TX_QEC, TX_LOL, TX_DCC, and TX_ATTEN_DELAY further require the use of the observation datapath through internal loopback (ILB) to receive the transmitted signal so that a joint analysis is performed by observing the relationship between the transmitted and received signals.

Transmitter Initial Calibrations Utilizing Internal Signal Generation Without Loopback

Figure 91 shows a high-level block diagram of the system configurations for transmitter initial calibrations that require inserting signals into the transmitter datapath without using a loopback path.

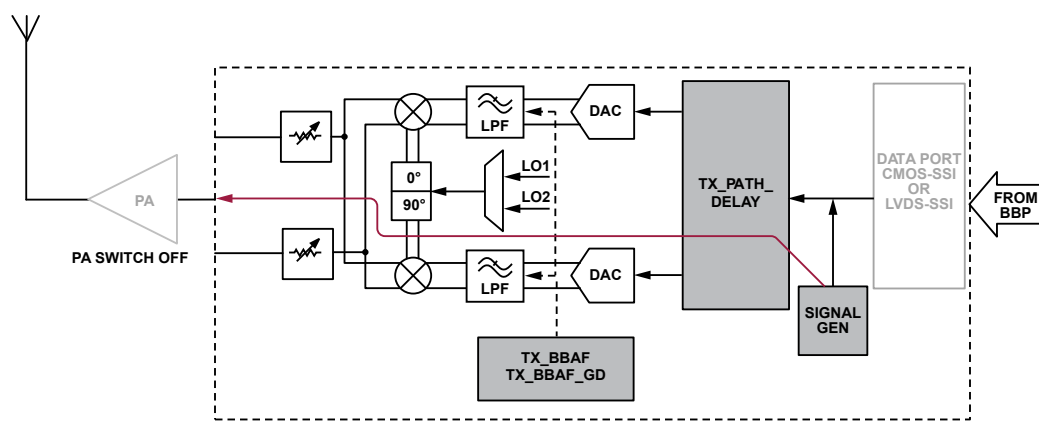


Figure 91. Transmitter Initial Calibration System Configuration with Signal Generation Without Loop Back

Figure 91 shows how an internal signal generator inserts calibration signals (red) into the transmitter datapath. The data port is disabled during initial calibrations to avoid interference. TX_PATH_DELAY is performed in the digital domain, whereas TX_BBAF and TX_BBAF_GD are performed in the analog domain, all use the signal generator to insert tones for calibration. During all these calibrations, test signals are inserted into the transmitter datapath and appear at the transmitter output. So, it is important to switch off the power amplifier to avoid the possible damage due to the big calibration signal from ADRV910x TX output. A 50 Ω termination is also suggested to prevent tone signals bouncing back from the power amplifier input and reaching the ADRV910x transmitter output, thereby confusing internal calibrations.

External system requirements: For transmitter initial calibrations utilizing internal signal generation without loopback, power off the power amplifier in the transmitter path during these calibrations. When the power amplifier is disabled, the load at the transmitter output should be 50 Ω .

Transmitter Initial Calibrations Utilizing Internal Signal Generation and ILB

Figure 92 shows a high-level block diagram of system configurations for transmitter initial calibrations utilizing internal signal generation and internal loop back (ILB) path.

TRANSMITTER/RECEIVER SIGNAL CHAIN CALIBRATIONS

Note, ADRV910x has one transmitter supports both wideband profile and narrowband profile, depends on the transmitter profiles (WB or NB), the ILB will use RX1 (WB) or RX2 (NB).

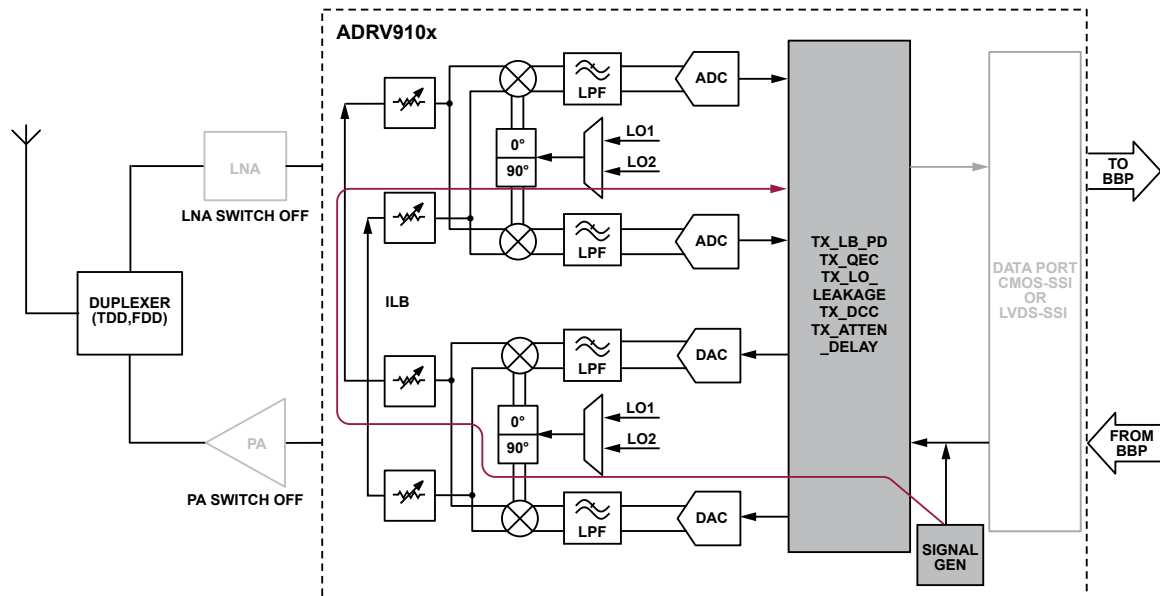


Figure 92. Tx Initial Calibration System Configuration with Signal Generation and Internal Loop Back

TX_LB_PD, TX_QEC, TX_LO_LEAKAGE, TX_DCC, and TX_ATTEN_DELAY use the ILB for calibrations. TX_LO_LEAKAGE and TX_QEC calculate the initial correction parameters. TX_LB_PD provides a measurement of the loop back path delay for the TX_LO_LEAKAGE and TX_QEC algorithms. Both TX_LO_LEAKAGE and TX_QEC calibrations sweep through a series of attenuation values and create a table of initial calibration values. Then, during operation and upon application of a new transmitter attenuation setting, the corresponding QEC and LO_LEAKAGE correction values are applied to the transmitter channel by the ADRV910x PS1 ARM. TX_DCC estimates the duty cycle error in the digital domain but applies the correction in the analog domain. TX_ATTEN_DELAY measures the delay between the transmitter digital attenuation block and the transmitter analog attenuation block, and it uses the ILB to observe and estimate the delay.

Similarly, during all these calibrations, the power amplifier connected to the device output is switched off, and 50 Ω termination is suggested. In addition, it is important to switch off the LNA (or RF switch if no LNA is present externally) external to the receiver datapath to avoid interference from the RF port into the receiver input used for data traffic.

External system requirement: For transmitter initial calibrations using the ILB, power off the power amplifier in the transmitter path during these calibrations. When the power amplifier is disabled, the load at the transmitter output should be 50 Ω . Also switch off the LNA (or RF switch if no LNA is present externally) for the loopback path to avoid receiving signals from the RF port.

Initial Transmitter Calibration Utilizing Internal Signal Generation and ELB

TX BALUN Common-mode rejection ratio (CMRR) affects the LO leakage performance. The TX LO leakage internal loopback initial calibration can't correct the LO leakage caused by the worse Balun CMRR. It is possible to perform TX LO leakage initial calibrations using external loop back (ELB) to further correct the LO leakage. Figure 93 shows the high level block diagram of initial transmit calibrations using internal calibration signal generation and ELB. The ELB could start from the ADRV910x output (ELB type 1) or from the PA output (ELB type 2).

When using the ELB1, the external system requirement is similar with transmitter initial calibration using IIB, power off the PA in the transmitter during these calibrations, also switch off the LNA (or RF switch if no LNA is present externally) for the loopback path to avoid receiving signals from the RF port.

When the ELB2 feedback path is used, enable the power amplifier to make a full external loop between the transmitter outputs and the observation channel inputs. The advantage of this calibration is to obtain a good estimate (gain/phase) of the external loop channel conditions before operation. Note that the calibration signal might be transmitted through the antenna. Although the power level of the calibration signal is set as low as possible, make sure it does not cause any problem when using this option.

Note: Tx LO leakage external loopback initial calibration is not yet supported.

TRANSMITTER/RECEIVER SIGNAL CHAIN CALIBRATIONS

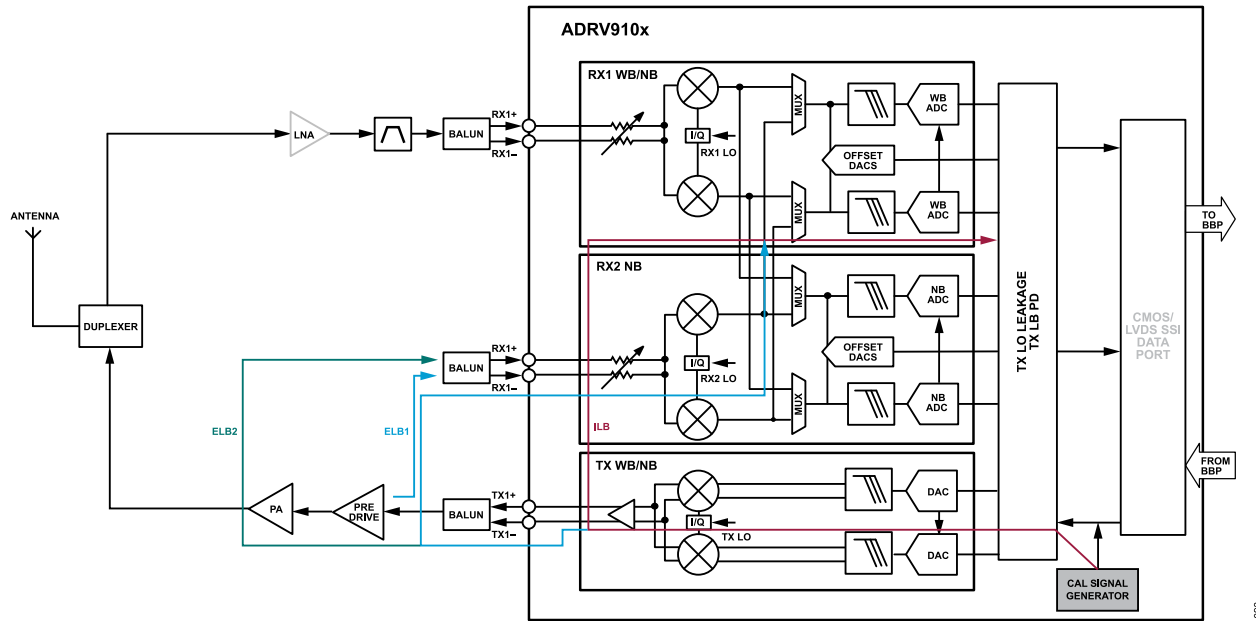


Figure 93. Transmitter Initial Calibration System Configuration Example (TX/RX WB) with Signal Generation and External Loopback

System Considerations for Receiver Initial Calibrations

This section also has high level block diagrams that show the device configurations and external system requirements for receiver initial calibrations. In all the diagrams, greyed-out lines and blocks are not active in the calibration. Note that the ADRV910x PS1 ARM performs each of the calibrations. It configures the ADRV910x device per the diagrams, with respect to enabling/disabling paths, and so on. No user input is required in this regard. However, it is important to meet the external system conditions, such as terminating the receiver input properly for receiver initialization calibrations.

Among 13 different receiver initial calibrations, RX_HPADC_RC, RX_HPADC_FLASH, RX_HPADC_DAC (reserved not enabled), calibrations are performed in the analog domain, and the corrections are applied to HP ADC (RX1 only) while all other receiver initial calibrations are performed in the digital domain. For RX_QEC_FIC, RX_QEC_TCAL, RX_GAIN_PATH_DELAY and RX_DMR_PATH_DELAY, the calibration results are applied in the digital domain for correction. For RX_DCC, RX_RF_DC_OFFSET, RX_TIA_CUTOFF, RX_GROUP_DELAY, and RX_QEC_ILB_LO_DELAY, the calibration results are applied in the analog domain for correction. The Figure 94 shows the high-level block diagram of system configurations for receiver initial calibrations. Note that different calibrations perform at different locations in the receiver datapath, which is simplified in Figure 94.

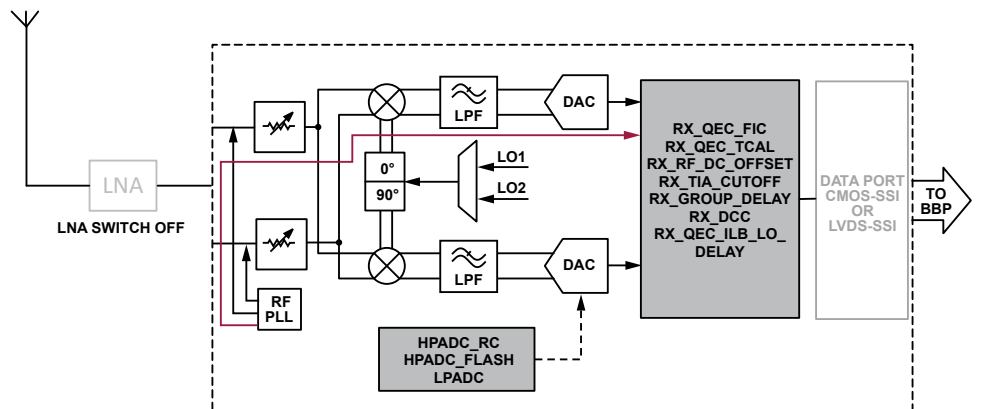


Figure 94. RX Initial Calibration System Configuration

During the receive initial calibration, as shown in Figure 94, the data port is disabled to avoid sending data to the baseband processor. The ADRV910x PS1 ARM controls this without user interaction. Except for the RX_RF_DC_OFFSET calibration, all other digital domain calibration algorithms must be injected with calibration tones generated by the RF PLL and these must be injected internally at the receiver input. For

TRANSMITTER/RECEIVER SIGNAL CHAIN CALIBRATIONS

example, the RX_QEC_TCAL calibration routine sweeps a number of internally generated test tones across the desired frequency band, and then measures quadrature performance and calculates correction coefficients. Therefore, during the receive calibration, any incoming signals from the RF port should be avoided, as these can interfere with the calibration tones. To ensure this, it is important to isolate the receiver input port from the incoming signals by disabling the LNA (or by switching off the external RF switch if no LNA is present externally). A 50 Ω termination is recommended to prevent the tone signals bouncing back from an external LNA output and reaching the receiver input, confusing the internal calibrations.

External system requirements: For optimal performance and lower calibration duration, the device receiver input port should be isolated from incoming signals during receiver initial calibrations. For many receiver calibrations, the calibration tones appear on the receiver pins. Therefore, prevent it from reaching the antenna through the receiver port by terminating it properly. This also prevents the calibration tones from reaching the antenna through the RF coupling. A 50 Ω termination is recommended to prevent tone signals bouncing back from an external LNA output and reaching the receiver input, confusing the internal calibrations.

TRACKING CALIBRATIONS

There are 13 different types of tracking calibrations, which are classified into transmitter and receiver/observation receiver tracking calibrations.

Transmitter Tracking Calibrations

- ▶ Quadrature error correction (QEC) tracking calibration
- ▶ Local oscillator leakage (LOL) tracking calibration
- ▶ Loop back path delay (LB PD) tracking calibration (Reserved, not yet supported)
- ▶ Tx Power Amplifier Correction (Reserved, not yet supported)
- ▶ Digital predistortion and closed loop gain control (DPD-CLGC) tracking calibration

Note, all the Transmitter tracking is available for TDD mode only.

Receiver/Observation Receiver Tracking Calibrations

- ▶ Harmonic distortion (second order) (HD2) tracking calibration (for RX IF mode only)
- ▶ Receiver quadrature error correction wideband poly (RX QEC WBPLOY) tracking calibration
- ▶ Observation receiver quadrature error correction wideband poly (ORx QEC WBPLOY) tracking calibration (Reserved, not yet supported)
- ▶ Baseband DC offset (BBDC) tracking calibration
- ▶ RF DC (RFDC) tracking calibration
- ▶ Quadrature error correction FIC (QEC FIC) tracking calibration
- ▶ Rx Gain Control Detectors tracking calibration
- ▶ RSSI tracking calibration

The ADRV910x PS1 ARM microprocessor fully controls all the tracking calibrations, there is no need for user interactions.

Tracking Calibrations API Programming

The ADRV910x internal microprocessor in the device schedules/performs tracking calibrations to optimize the performance during its operation.

The top-level API function The ADRV910X internal microprocessor in the device schedules/performs tracking calibrations to optimize the performance during its operation.

The top-level API function `adi_ADRV910X_cals_Tracking_Set()` sets the tracking calibrations. The tracking calibrations are performed based on the tracking calibration configurations defined by “`adi_ADRV910x_TrackingCals_t`” data structure which has an array containing calibration bit mask for channel related tracking calibrations, (`chanTrackingCalMask[0]` is the mask for Rx1/TxWB channels, and `chanTrackingCalMask[1]` is the mask for Rx2/TxNB channels). The enumerator “`trackingCalibrations_e`” defines all the tracking calibrations.

[Table 24](#) describes the mask bit assignment for tracking calibrations in the “`trackingCalibrations_e`”. It also explains the functionality of each tracking calibration.

Table 24. Tracking Calibration Mask Bit Assignments

Bits	Corresponding Enum	Calibration	Description
D0	TRACKING_CAL_TX_QEC	Tx QEC Tracking Calibration	Performs tracking QEC calibration for frequency independent errors for the Tx path. It estimates the gain and phase mismatch on-the-fly using

TRANSMITTER/RECEIVER SIGNAL CHAIN CALIBRATIONS

Table 24. Tracking Calibration Mask Bit Assignments (Continued)

Bits	Corresponding Enum	Calibration	Description
			the real-time traffic data and applies the gain mismatch in the digital domain. Similar to the initial calibration, it uses the Txpath and an ILB path.
D1	TRACKING_CAL_TX_LO_LEAKAGE	Tx LOL Tracking Calibration	Performs tracking LOL calibration. It estimates the LOL on-the-fly and applies the cancellation in the digital domain. It uses the Tx path and a ILB.
D2	TRACKING_CAL_TX_LB_PD	Tx Loop Back Path Delay Tracking Calibration	Tracks the Tx loop back path delay (can be for either ILB or ELB) on-the-fly. This information is required for QEC, LOL, and DPD tracking calibrations. Currently, this tracking calibration is not available. QEC, LOL, and DPD tracking calibration use the delay measurement obtained from the Tx loopback path delay initial calibration.
D3	TRACKING_CAL_TX_PAC	TX Power Amplifier Correction Tracking	Reserved, not yet supported
D4	TRACKING_CAL_TX_DPD_CLGC	Tx DPD and CLGC Tracking Calibration	Predistorts the transmit signal in real-time to compensate for the power amplifier nonlinearity to achieve higher power efficiency and also compensate for the gain variation in the power amplifier. For more details, see Digital Predistortion (DPD) section.
D5~D7	Not Used		
D8	TRACKING_CAL_RX_HD2	Rx HD2Tracking Calibration	Corrects the Rx second order harmonic distortion
D9	TRACKING_CAL_RX_QEC_WBPOLY	Rx QEC WBPLOY Tracking Calibration	Corrects the Rx frequency dependent quadrature error for RX1 wideband applications.
D10~D11	Not Used		
D12	TRACKING_CAL_ORX_QEC_WBPOLY	ORx QECWB Tracking Calibration	Corrects ORx frequency dependent quadrature error for RX1 wideband applications.
D13~D18	Not Used		
D19	TRACKING_CAL_RX_BBDC	Rx BBDCTracking Calibration	Mitigates the Rx DC offset at the baseband on-the-fly.
D20	TRACKING_CAL_RX_RFDC	Rx RFDCTracking Calibration	Mitigates the Rx DC offset at the RF (analog) on-the-fly.
D21	TRACKING_CAL_RX_QEC_FIC	Rx QEC FICTracking Calibration	Corrects Rx frequency independent quadrature error for both NB and WB applications.
D22	TRACKING_CAL_RX_GAIN_CONTROL_DETECTORS	Rx Gain ControlDetectorsTrac kingCalibration	Enables/disables gain control detectors on-the-fly. By disabling it, AGC stops responding to changes in the input signal and instead holds its last gain index.
D23	TRACKING_CAL_RX_RSSI	Rx RSSITracking Calibration	Enables/disables Rx signal strength measurement on-the-fly.
D24~D31	Not Used		

External System Requirements for Tracking Calibrations

Different from initial calibrations, tracking calibrations are performed on-the-fly with real-time traffic data. Therefore, it is mostly transparent and fully controlled by the internal microprocessor. The external system requirements are as follows:

- ▶ Make sure the external paths are available when running some of the tracking calibrations on external loopback paths.
- ▶ When using an external loopback path after a power amplifier (ELB), choose a suitable attenuator between the power amplifier output and the observation channel input to prevent the transmitter output data from saturating the observation channel input.
- ▶ When an external DPD is employed in the system, it shares time with the other transmitter tracking calibrations to avoid conflicts.

RECEIVER GAIN CONTROL

The ADRV910x receivers feature automatic and manual gain control modes for flexible gain control in a wide array of applications. It controls the gain at various stages of the receiver datapath to avoid overloading during the onset of a strong interferer. In addition, it ensures that the receiver digital output data is representative of the root mean square (RMS) power of the receiver input signal so that any internal front-end gain changes to avoid overloading are transparent to the baseband processor.

The two gain control modes in the ADRV910x are called Automatic Gain Control (AGC) and Manual Gain Control (MGC). The AGC allows receivers to autonomously adjust the receiver gain depending on the variations of the input signal. It controls the gain of the device based on the information from signal detectors called peak detector and power detector. The receivers can also operate in the MGC mode, where changes in gain are initiated by the baseband processor through API commands or digital GPIO (DGPIO) pins. In the MGC mode, by enabling the signal detectors, the baseband processor can optionally use the information from the signal detectors through the DGPIO pins to properly control the gain.

IMPORTANT TERMINOLOGY

The following is a summary of the important terms used in the following sections.

Manual Gain Control (MGC). A use case where the ADRV910x Receiver gain is manually controlled by baseband processor. is in control of the currently applied gain settings in the receiver chain.

Automatic Gain Control (AGC). The device's own internal AGC, where the gain is automatically controlled by the ADRV910x device itself.

Gain Attack. Indicates the reduction of the receiver gain due to an overloaded signal path.

Gain Recovery. Indicates the increase of the receiver gain due to a reduction in the power of the signal received.

Gain Compensation. The process of compensating for the analog attenuation in the device (prior to the ADC) with a corresponding amount of digital gain before the digital signal is sent to the baseband processor.

High Threshold. Triggers the gain attack event. Some detectors can have multiple high thresholds.

Low Threshold. Triggers the gain recovery event. Some detectors can have multiple low thresholds.

Threshold Overload. When a threshold is exceeded in a signal detector, it is an overload.

Threshold Underload. When a threshold is not exceeded in a signal detector, it is an underload.

Overrange Condition. An overrange condition is when the AGC is required to reduce the gain. This can either be a peak condition, where a programmable number of individual overloads of a high threshold occurred within a defined period of time, or a power condition, where the measured power exceeds a high power threshold.

Underrange Condition. An underrange condition is when the AGC is required to increase the gain. This can either be a peak condition, where a lower threshold is not exceeded a programmable number of times within a defined period of time, or a power condition, where the measured power does not exceed a low power threshold.

RECEIVER DATAPATH

Figure 95 shows the simplified receiver datapath and gain control blocks. The receivers have front-end attenuators before the mixer stage, which are used to attenuate the signal in the RF domain to ensure the signal does not overload the receiver chain. Note: The ADRV910X Receivers provide about 22dB gain so the front end gain attenuator further attenuates the signal from that level. There is also digital gain control capability in the digital domain.

RECEIVER GAIN CONTROL

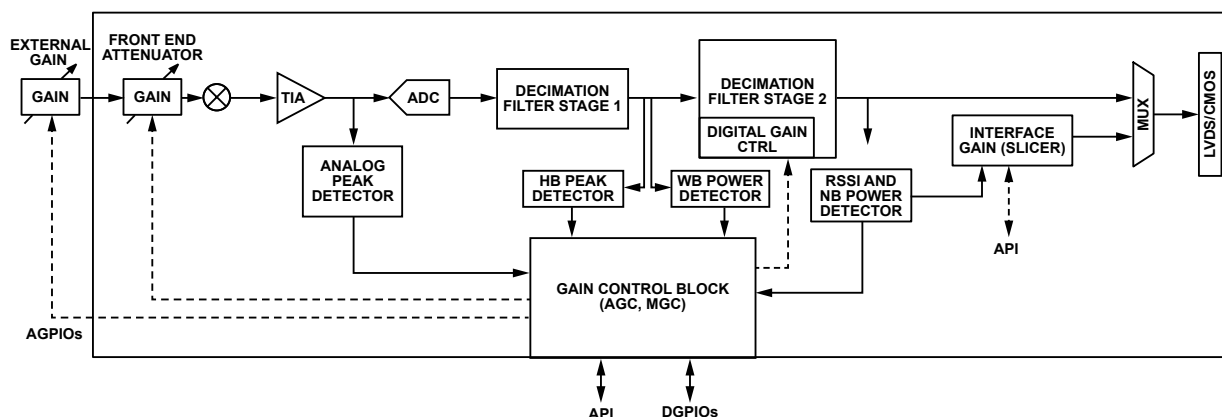


Figure 95. RX Datapath and Gain Control blocks

The receiver chain has multiple observation elements that can monitor the incoming signal. These can be used in either the MGC or AGC mode. Firstly, there is an analog peak detector (APD) before the ADC. Being in the analog baseband, this peak detector sees the signals first, and also has the interfering signals visibility, which can overload the ADC but be filtered as they progress through the digital chain. Note that the APD is located after the TIA filter. The second peak detector is called the HB peak detector because it monitors the data at the output of the half band (HB) filtering block in the receiver chain.

A power measurement detection block is also provided in the receiver chain, which takes the rms power of the received signal over a configurable period. The power measurement detector can be the wideband power detector which locates at the half band filter output of decimation filter stage1, or narrow band detector that locates at the RSSI block.

Besides the front end gain control, the ADRV910x also has the capability to control an external gain element through the analog GPIO (AGPIO) pins. In the digital domain, this device further controls the digital gain in both the wideband (WB) and narrowband (NB) modes. To avoid saturating the output signal due to the limitation of the bit-width of the data port, an optional interface gain (slicer) is applied at the end of the datapath by properly shifting the data. The interface gain can be automatically controlled internally inside the device by using the information provided from the receiver signal strength indicator (RSSI) block or manually controlled through API commands by users.

Figure 95 shows that the gain control block has multiple inputs, which come from the peak detectors, and power detectors. By using the information from those detectors, the gain control block controls the gain of the signal chain using a predefined gain table. Note that the default gain table is loaded into the device during initialization. An API function `adi_adrv910x_Rx_GainTable_Write()` is called to load a custom gain table or reconfigure the gain table. Note that this operation should be done before performing initial calibrations.

In the gain table, each row provides a unique combination of six fields, including front-end attenuator, TIA control, ADC control, external gain control, phase offset, and digital gain/attenuator. Among them, the TIA control that sets the TIA gain, ADC control that sets the ADC gain, and phase offset that compensates for the phase discontinuity during gain change are reserved for future use.

Based on the table row selected, either manually in the MGC mode or automatically in the AGC mode, the gain control block updates the variable gain elements depicted by the dash lines. In the MGC mode, the user controls the variable gain elements through the same gain table using the API commands and DGPIOs.

Table 25 shows the first three and the last three rows in a sample gain table.

Table 25. Sample Rows from the default Rx Gain Table

Gain Table Index	Front-End Attenuator Control Word [7:0]	TIA Control	ADC Control	External Gain Control [1:0]	Phase Offset	Digital Gain/Attenuator Control Word[10:0]
187	251	0	0	0	0	13
188	251	0	0	0	0	23
189	250	0	0	0	0	-4
...	...	...	...	...	...	...
253	28	0	0	0	0	-2
254	14	0	0	0	0	-1
255	0	0	0	0	0	0

RECEIVER GAIN CONTROL

The gain table index is the reference for each unique combination of gain settings in the programmable gain table. The current possible range of the gain table is 187 to 255. The gain index region is user configurable. Call an API function `adi_adrv910x_Rx_MinMaxGainIndex_Set()` right after loading the gain table to load multiple gain table regions and switch between multiple gain table regions during runtime.

The front-end attenuator and digital gain/attenuator are the two fields used in the default gain table. The front-end attenuator is an 8-bit control word. The amount of attenuation applied depends on the value set in this column of the selected gain table index. The following equation provides an approximate relationship between the attenuation in dB and the front-end attenuation control word programmed in the gain table, N:

$$\text{Attenuation (dB)} = 20\log_{10}\left(\frac{256-N}{256}\right) \quad (6)$$

It is shown that index 255 denotes a 0 dB front-end attenuation and the step size between the adjacent gain index is approximately 0.5 dB. Note that when the index is below 195, the actual step size becomes less accurate.

The digital gain/attenuator column is used to apply the gain or attenuation digitally. The 11-bit signed word defines the digital gain applied, which equals the control word times 0.05 in dB. Table 25 shows that for gain index 253, the digital gain is calculated as $-2 \times 0.05 = -0.1$ dB.

There are two types of receiver gain tables. One is for gain correction, in which the digital gain is for correcting the small step size inaccuracy in the front-end attenuator, therefore, the gain step size for each gain index is about 0.5dB. The other one is for gain compensation, which compensates the entire front-end attenuation. The example Table 25 stands for a receiver gain correction table.

The receive channel can also operate as an observation channel (ORx), as discussed in [Receiver and Observation Receiver Signal Chain](#). In such a case, a different gain table is used. It provides 0 dB to 30 dB attenuation in 5 dB step size. The gain index is from 2 to 14. Gain index 14 provides 0 dB attenuation, then the next two indexes (both 13 and 12) provide 5 dB attenuation. The last two indexes (both 3 and 2) provide 30 dB attenuation. The digital gain specified in the gain table is only used for gain correction. In addition, only manual gain control is supported.

Note that the ADRV910x also uses the ORx gain table internally for initial and tracking calibrations, so it should not be modified by user.

GAIN CONTROL MODES

Select the gain control mode through the API function `adi_adrv910x_Rx_GainControl_Mode_Set()` for a specified channel. For more details, refer to the API Doxygen document. `adi_adrv910x_RxGainControlMode_e` is an enum to select the gain mode. Table 26 shows the possible options.

Table 26. Definition of `adi_adrv910x_RxGainControlMode_e`

ENUM	Gain Mode
<code>ADI_ADRV910X_RX_GAIN_CONTROL_MODE_SPI</code>	Manual Gain Control SPI Mode
<code>ADI_ADRV910X_RX_GAIN_CONTROL_MODE_PIN</code>	Manual Gain Control PIN Mode
<code>ADI_ADRV910X_RX_GAIN_CONTROL_MODE_AUTO</code>	Automatic Gain Control Mode

Automatic Gain Control (AGC)

In the AGC mode, a built-in state machine automatically controls the gain based on the user-defined configuration. The AGC is configured to one of two modes:

- ▶ Peak detect mode, where only the peak detectors are used to make gain changes.
- ▶ Peak/Power detect mode, where information from both the power detector and peak detectors are used jointly to make gain changes.

Peak Detect Mode

In this mode, only the peak detectors are used to inform the AGC to make gain changes. This section explains the basic premise of the operation, while the subsequent sections cover the more explicit details of configuring the peak detectors.

The APD and HB detector both have a high threshold and a low threshold, `apdHighTresh`, `apdLowThresh`, `hbHighTresh`, and `hbUnderRangeHighThresh`, respectively. These levels are user programmable, as well as the number of times to exceed a threshold to flag an overrange/underrange condition.

The high thresholds are used as limits on the incoming signal level and are principally set based on the maximum input of the ADC. When an overrange condition occurs, the AGC reduces the gain (gain attack). The low thresholds are used as lower limits on the signal level. When an underrange condition occurs, the AGC increases the gain (gain recovery). The AGC stable state (where it does not adjust gain) occurs when neither an underrange nor overrange condition occurs.

RECEIVER GAIN CONTROL

Each overrange/underrange condition has its own attack and recovery gain step, as shown in Table 27.

Table 27. Peak Detector Gain Steps

Over Range/Under Range	Gain Step
apdHighThresh over range	Reduce gain by apdGainStepAttack
apdLowThresh under range	Increase gain by apdGainStepRecovery
hbHighThresh over range	Reduce gain by hbGainStepAttack
hbUnderRangeHighThresh under range	Increase gain by hbGainStepHighRecovery

An overrange condition occurs when the high thresholds are exceeded a configurable number of times within a configurable period. An underrange condition occurs when the low thresholds are not exceeded a configurable number of times within the same configurable period. These counters make the AGC less sensitive to occasional peaks in the input signal, ensuring that a single peak exceeding a threshold does not necessarily cause the AGC to react. Table 28 outlines the counter parameters for the individual over range/under range conditions.

Table 28. Peak Detector Configurable Counters

Over Range/Under Range	Counter
apdHighThresh over range	apdUpperThreshPeakExceededCnt
apdLowThresh under range	apdLowerThreshPeakExceededCnt
hbHighThresh over range	hbUpperThreshPeakExceededCnt
hbUnderRangeHighThresh under range	hbUnderRangeHighThreshExceededCnt

The AGC uses a gain update counter to time gain changes which happens when the counter expires. The counter value, and therefore the time spacing between possible gain changes, is user programmable through the gainUpdateCounter parameter. The user specifies the period in AGC clock cycles to make gain changes. Typically, this might be set to frame or sub-frame boundary periods.

Once the gain update counter expires, all the peak threshold counters are reset. The gain update period is, therefore, a decision period. The overload thresholds and counters are set based on the number of overloads considered acceptable for the application within the gain update period.

Figure 96 shows an example of the AGC response to a signal versus the APD or HB peak detector threshold levels. APD and HB detector works in the same fashion in this mode. For ease of explanation, only APD is mentioned in the following discussions. The staircase line is representative of the peaks of the signal. Initially, the peaks of the signal are within the apdHighThresh and apdLowThresh. No gain changes are made. An interferer suddenly appears, whose peaks now exceed apdHighThresh. On the next expiry of the gain update counter (assuming a sufficient number of peaks occurred to exceed the “apdUpperThreshPeakExceededCnt”), the AGC decrements the gain index (reduces the gain) by apdGainStepAttack at the next expiry of the gain update counter. This is not sufficient to get the signal peaks within the threshold levels, and hence, the gain is decremented once more, with the peaks now between the two thresholds. The gain is stable in this current gain level until the interferer is removed and the peaks of the desired signal are now below the apdLowThresh, which results in an underrange condition. Hence, the AGC increases gain by the apdGainStepRecovery at the next expiry of the gain update counter, continuing to do so until the peaks of the signal are within the two thresholds once more.

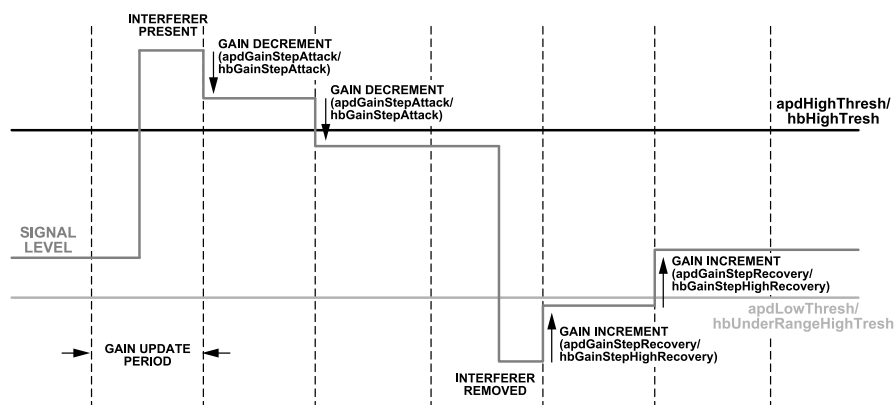


Figure 96. APD/HB Thresholds and Gain Changes Associated with Under-range and Over-range Conditions

RECEIVER GAIN CONTROL

It is possible to enable a **fast attack** mode using `changeGainIfThreshHigh`, whereby the AGC is instructed to reduce gain immediately when an overrange condition occurs instead of waiting until the next expiry of the gain update counter. `changeGainIfThreshHigh` parameter has independent controls for the APD and HB detectors. Values from 0 to 3 are valid, as shown in [Table 29](#).

Table 29. `changeGainIfThreshHigh` Settings

<code>changeGainIfThreshHigh[1:0]</code>	Gain Change following APD Over range	Gain Change following HB Over range
00	After expiry of <code>gainUpdateCounter</code>	After expiry of <code>gainUpdateCounter</code>
01	After expiry of <code>gainUpdateCounter</code>	Immediately
10	Immediately	After expiry of <code>gainUpdateCounter</code>
11	Immediately	Immediately

[Figure 97](#) shows how AGC reacts when the `changeGainIfThreshHigh` is set for the APD or HB detector. In this case, when the interferer appears, the gain is updated as soon as the number of peaks exceeds the peak counter. It does not wait for the next expiry of the gain update counter. Hence, a number of gain changes are made in quick succession, providing a much faster attack than the default operation. The assumption here is that if the ADC is overloaded, then it is best to decrease the gain quickly rather than wait for a suitable moment in the received signal to change the gain. This mode is referred to as the “fast-attack” mode.

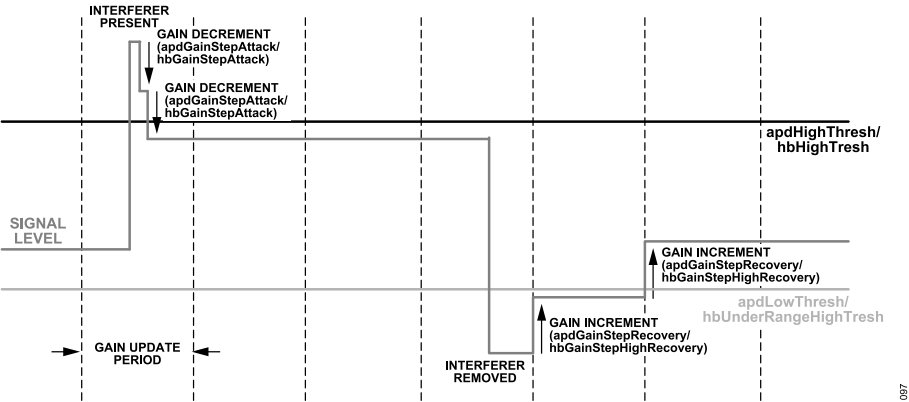


Figure 97. APD/HB Gain Changes with Fast Attack Enabled

Besides the “fast attack” mode, it is also possible to enable a “fast recovery” mode. This functionality is enabled with the `enableFastRecovery-Loop` parameter.

This “fast recovery” mode only works with the HB detector. [Figure 98](#) shows this operation. In this mode, achieve the “fast recovery” using multiple low thresholds and step sizes as well as the update periods. When the signal level falls below `hbUnderRangeLowThresh`, the lowest threshold, the gain is incremented the most by `hbGainStepLowRecovery` following the expiry of a gain update period. Note that in the “fast recovery” mode, the `agcUnderRangeLowInterval` is used instead of the gain update counter to set the gain update period (this also applies to the APD). After sufficient gain increases to bring the signal level above `hbUnderRangeLowThresh`, the gain is incremented by `hbGainStepMidRecovery` after the expiry of `agcUnderRangeMidInterval`, which is a multiple of `agcUnderRangeLowInterval`. Finally, when the signal level is increased above `hbUnderRangeMidThresh`, the gain is incremented by `hbGainStepHighRecovery` following the expiry of `agcUnderRangeHighInterval`, which is a multiple of `agcUnderRangeMidInterval`. The multiple thresholds and interval parameters allow for faster gain recovery.

Typically, `agcUnderRangeHighInterval` is set to gain update counter, as shown in [Figure 98](#). Therefore, when the signal level is below the middle and low thresholds, the recovery happens multiple times within a single gain update counter, making the recovery process much faster. Note that in the “fast recovery” mode, gain recovery might not always happen at the expiry of the gain update counter, which is different from the mode without “fast recovery”. If the gain update counter is set to align with the frame or subframe boundary, it is possible that a fast recovery happens in the middle of a frame or subframe. Therefore, it is recommended not to use the “fast recovery” mode when there is a stringent requirement for keeping a constant gain for an entire frame or subframe.

RECEIVER GAIN CONTROL

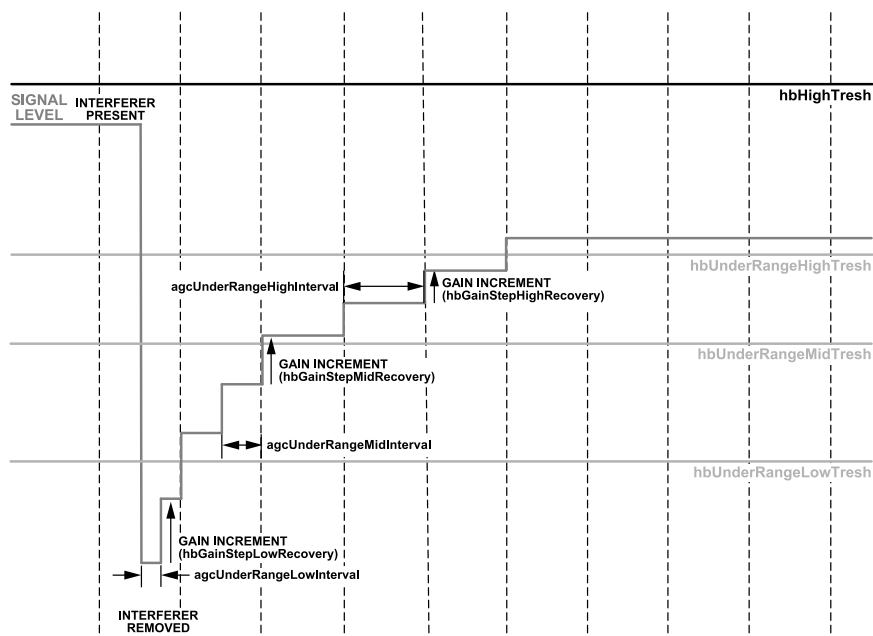


Figure 98. AGC operation with HB Detector in Fast Recovery Mode

It is highly recommended that the `apdHighThresh` and `hbHighThresh` are set to an equivalent dBFS value. Likewise, it is highly recommended that the `apdLowThresh` and the `hbUnderRangeHighThresh` are set to equivalent values. This equivalence is approximate, as these thresholds have unique threshold settings and are not exactly equal. This section discusses the relevant priorities between the detectors and how the AGC reacts when multiple threshold detectors are exceeded. Table 30 shows the priorities between the detectors when multiple overranges occur.

Table 30. Priorities of Attack Gain Steps

<code>apdHighThresh</code> Over range	<code>hbHighThresh</code> Over Range	Gain Change
No	No	No Gain Change
No	Yes	Gain Change by <code>hbGainStepAttack</code>
Yes	No	Gain Change depends
Yes	Yes	Gain Change by <code>apdGainStepAttack</code>

In the case of `apdHighThresh` Overrange is Yes, `hbHighThresh` Overrange is No, there are two possibilities:

1. There is out of band blocker signal which really has strong level to trigger the `apdHighThresh` overrange to high, but the blocker is filtered by digital filters that locate before HB peak detector, therefore `hbHighThresh` overrange is low.
2. The input signal is smaller than `apdHighThresh` and `hbHighThresh`, but due to APD characteristics (Refer Figure 101) issue, APD overestimates signal and falsely trigger the `apdHighThresh` overrange to high since the signal is close to DC.

To overcome the issue in the second scenario, a mitigation mode is designed, which utilizes a secondary digital threshold (smaller than `apdHighThresh` and `hbHighThresh`) to further determine if the `apdHighThresh` overrange flag is false or true. When the mitigation mode is turned on, if the secondary threshold is exceeded, then it confirms the `apdHighThresh` overrange flag is a false condition due to the signal near DC, so the gain does not change. If the secondary threshold is not exceeded, then it confirms `apdHighThresh` overrange high is true and is triggered by out-of-band blocker, so the gain is decremented.

In RX1 AGC implementation, the mitigation mode is always turned on, and the user is not allowed to configure the secondary digital threshold. But for RX2 channel, there is an optional “AGC high threshold mode” can be configured to determine if the mitigation mode is enabled or not. Refer [Rx2 AGC High Threshold Mode](#) section for more detailed information.

For gain recovery, the number of thresholds is dependent on whether fast recovery is enabled or not. Considering the fast recovery scenario, the priority of the thresholds is:

1. `hbUnderRangeLowThresh` Under Range Condition
2. `hbUnderRangeMidThresh` Under Range Condition

RECEIVER GAIN CONTROL

3. hbUnderRangeHighThresh Under Range Condition
4. apdLowThresh Under Range Condition

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

The apdLowThresh has priority in terms of preventing recovery. If apdLowThresh reports an overrange condition (sufficient signal peaks have exceeded its threshold in a gain update counter period), then no further recovery is allowed. apdLowThresh and hbUnderRangeHighThresh are configured to be close to the same value of dBFS, but assuming some small difference between the thresholds, then as soon as apdLowThresh is exceeded, recovery no longer occurs. The reverse is not true. hbUnderRangeHighThresh does not prevent the gain recovery towards the apdLowThresh. Given the strong recommendation that the apdLowThresh and hbUnderRangeHighThresh be set equally, then a condition whereby apdLowThresh is at a lower dBFS level to hbUnderRangeLowThresh or hbUnderRangeMidThresh should not occur.

Another exception is if the recovery step size for a detector is set to zero. If so, the AGC makes the gain change of the highest priority detector with a non-zero recovery step. [Figure 99](#) provides a flow diagram of the decisions of the AGC when recovering the gain in peak detect mode.

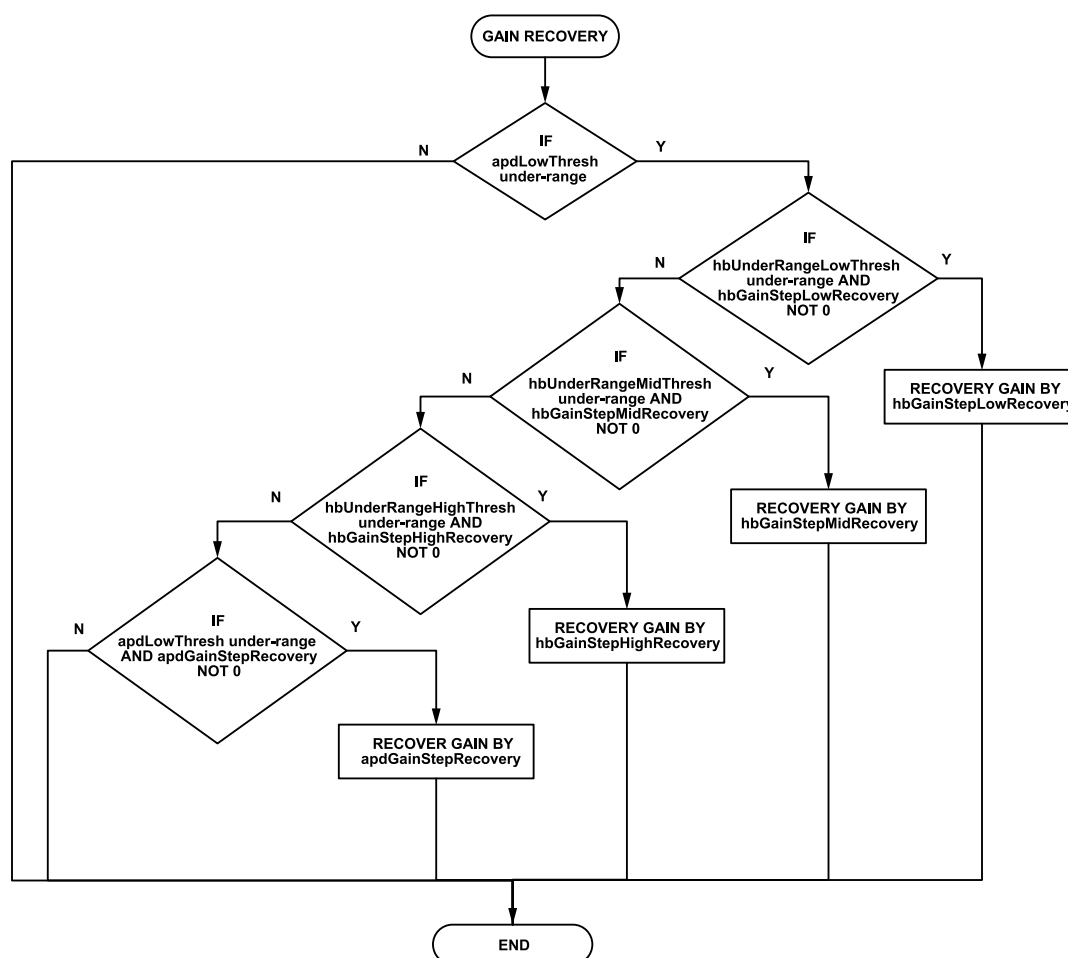


Figure 99. Flow Diagram for AGC Recovery in Peak Detect AGC Mode

059

Peak/Power Detect Mode

In this mode, the peak and power detect work jointly to control the gain of the Rx chain. In the event of an overrange condition, then both the peak and the power detect can instantiate a gain decrement. In the event of an underrange, only the power detect can increment the gain. The power detector changes gain solely at the expiry of the gain update counter. As mentioned earlier, the peak detect can be set in one of two modes (depending on the setting of gainChangeIfThreshHigh) whereby the AGC: 1) waits for the gain update counter to expire before initiating a gain change; or 2) immediately updates the gain as soon as the over range condition occurs (see [Figure 96](#) to [Figure 97](#)). Therefore, in the peak/power detect mode, if the gain attack is instantiated by peak detectors, it is possible to perform fast attack.

RECEIVER GAIN CONTROL

The power detect provides the RMS power measurement of the receiver data at the output of HB filtering block or at the RSSI. In power detect mode, the AGC compares the measured signal level to programmable thresholds which provide a 2nd order control loop, whereby gain can be changed by larger amounts when the signal level is farther from the target level while make smaller gain changes when the signal is closer to the target level. This allows the gain change faster when the level is farther away from the targeted range.

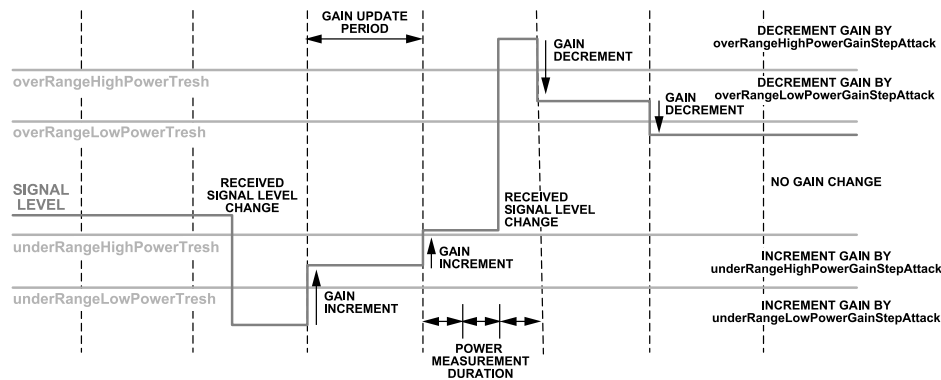


Figure 100. Power Detector Thresholds and Gain Changes for Underrange and Overrange Conditions

Figure 100 shows the operation of the AGC when using the power detector. Considering the power detected in isolation from the peak detectors, the AGC does not modify the gain when the signal level is between overRangeLowPowerThresh and underRangeHighPowerThresh. This is the target range for the power measurement. The associated thresholds are also called inner thresholds.

When the signal level goes below underRangeLowPowerThresh, the AGC waits for the next gain update counter expiry and then increments the gain by underRangeLowPowerGainStepRecovery. When the signal level is greater than underRangeLowPowerThresh but below underRangeHighPowerThresh, the AGC increments the gain by underRangeHighPowerGainStepRecovery. Likewise, when the signal level goes above overRangeHighPowerThresh, the AGC decreases the gain by overRangeHighPowerGainStepAttack, and when the signal level is between overRangeHighPowerThresh and overRangeLowPowerThresh, the AGC decreases the gain by overRangeLowPowerGainStepAttack. underRangeLowPowerThresh and overRangeHighPowerThresh are also called outer thresholds.

It is possible for the AGC to get contrasting requests from the power and peak detectors. An example is an interferer visible to the analog peak detector but significantly attenuated at the power detector. In this case, the APD is requesting a gain decrement, while the power detector is requesting a gain increment. The AGC has the following priority scheme in peak/power detect modes:

1. APD overrange
2. HB overrange
3. APD lower level peak exceeded
4. HB lower level peak exceeded
5. Power measurement

In this example, the gain is decremented because the APD overrange has a higher priority than the power measurement. However, the APD and HB lower level overload act differently in the peak detector and peak/power detect modes. In the peak detect mode, the lower level thresholds for these detectors are used to indicate an underrange condition, which causes the AGC to increase the gain. In the peak/power detect mode, these detectors are not used for gain recovery but used to control gain recovery by setting the API parameter lowThreshPreventGainInc. If this parameter sets and the signal levelling exceeds a lower level threshold, the AGC is prevented from increasing gain regardless of the power measurement.

When a signal has higher than the expected PAR, the power detector can indicate the gain increase while the peak detector's low threshold can still be exceeded. In such a case, the gain increase is prevented to avoid an overloading possibility. In addition, this can prevent an oscillation condition that can otherwise occur to an interferer visible to the APD but filtered before the power detects. In such a case, the peak detect can cause the AGC to decrease the gain. It does this until the interferer is no longer exceeding the defined threshold. At this point, the power detector can request an increase in the gain and does so until the detector's low threshold is exceeded. This might cause an oscillation condition. By using these lower level thresholds of peak detection, the AGC is prevented from increasing the gain as the signal level approaches an overload condition, providing a stable gain level for the receiver chain under such a condition.

RECEIVER GAIN CONTROL

Comparing the Peak Detect and Peak/Power Detect Modes

Among the two detection modes, peak detector offers the quickest response time to overload signals by employing “fast attack” mode. It allows the AGC to respond within hundreds of nanoseconds in overload scenarios. In addition, the peak detector also provides a “fast recovery” option to increase the gain of the desired signal quickly when an interferer disappears. It can also avoid the possible gain index oscillation issue of peak/power detect when the signal has higher than expected peak-to-average ratio (PAR).

With power detection, the gain change can only happen at the expiration of the gain update counter, which is typically set in the order of hundreds of microseconds or milliseconds. However, power detectors are usually more stable and unlikely to cause frequent gain changes. In addition, it can provide tighter control of the signal level by using a set of inner and outer thresholds compared to the peak detector.

It is highly recommended to use peak detection, especially for fast gain control.

Rx2 AGC High Threshold Mode

Ideally the APD thresholds should be frequency independent. However, the APD circuit intrinsically has residue ripples that cause the APD to over-estimate the signal level if the input signal is close to DC, shown in [Figure 101](#), the APD is triggered by a signal that is lower than the threshold when the signal is close to DC.

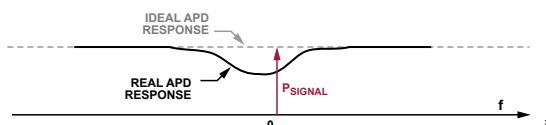


Figure 101. APD Trigger Level vs Input Signal Frequency

For RX2 narrow band channel, the “rxNbAgcHighThresholdMode” allows the very little AGC back off from the ADC saturation point and also overcomes the APD characteristics issue. This mode is very similar with the RX1 AGC mitigation mode, it uses the secondary HB2 peak detector to determine if the apdHighThresh overrange flag is true or false, an additional programmable FIR filter is used to help on the frequency selectivity for the secondary HB2 peak detector, the AGC is taken control by HB2 primary detector with hbHighThresh if the input signals are within the FIR passband, APD will take over the AGC control if the signals are out of the FIR filter passband. The RX2 high threshold mode AGC gain attack priorities are shown in [Table 31](#).

Table 31. Rx2 AGC High Threshold Mode Gain Attack Priorities and Steps

apdHighThresh Overrange	hbHighThresh Overrange	2 nd hbHighThreshOverrange	Gain Change
No	No	No	No Gain Change
No	No	Yes	No Gain Change
Yes	No	Yes	No Gain Change(Mitigation mode prevents the low frequency APD attack)
Yes	No	No	Gain Change by apdGainStepAttack(out of band blocker signal presents)
Yes	Yes	Yes	Gain Change by hbGainStepAttack(inband signal or close in blocker presents)

The target for “rxNbAgcHighThresholdMode” is to allow the highest possible AGC overload threshold (or minimum AGC back off from the ADC saturation point), so the Rx gain would always be the maximum possible value, this mode helps to get the best out of band blocking rejection performance. When “rxNbAgcHighThresholdMode” is enabled, the apdHighThresh and hbHighThresh is not user configurable, ADRV910x firmware will apply about -1 dBFS overload threshold for hbHighThresh and set the apdHighThresh and secondary HB2 peak detector threshold accordingly. All the relative parameters through the “adi_adrv910x_Rx_GainControl_Configure” will be overwritten.

If the “rxNbAgcHighThresholdMode” is not enabled, the AGC overload threshold is configurable by users, but the highest allowable value is about -3dBFS.

Note: AGC high threshold mode is only available RX2 channel, and is only available for profiles has less than 25Khz RF channel BW.

Manual Gain Control (MGC)

The gain control block applies the settings from the selected gain index in the gain table. In the MGC mode, the baseband processor is in control of selecting the gain index. There are two options: API command (ADI_ADRV910X_RX_GAIN_CONTROL_MODE_SPI) and pin control (ADI_ADRV910X_RX_GAIN_CONTROL_MODE_PIN).

RECEIVER GAIN CONTROL

In the API command mode, select a gain index in the gain table through the API function `adi_adrv910x_Rx_Gain_Set()`. The gain index selected for a channel is read back through the API function `adi_adrv910x_Rx_Gain_Get()`.

The pin control MGC mode is useful for the real time control of gain. In this mode, out of 20 digital DGPIO pins, two pins per receiver are used, one increasing and the other decreasing the gain table index. Specify both the maximum and minimum gain index as well as the increment and decrement step size (in the range of 0 to 7 gain table indices). Apply a pulse to the relevant DGPIO pin to trigger an increment or decrement in gain, as shown in [Figure 102](#). Hold this pulse high for at least two AGC clock cycles for a reliable detection of the rising edge to trigger the gain change. The API command `adi_adrv910x_Rx_GainControl_PinMode_Configure()` is used to properly configure this mode.

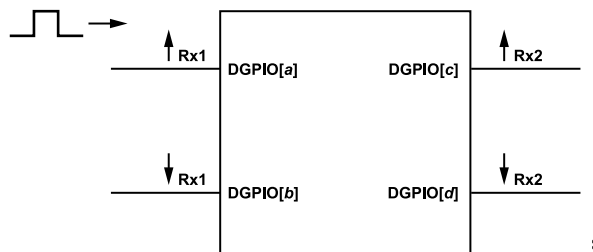


Figure 102. MGC Pin Mode: DGPIO[a-d] Represent Any of DGPIO[0-19]

In the MGC mode, to properly control the gain, optionally retrieve the status of peak detectors and the power detector in the device through a set of DGPIO pins (this can also be done in the AGC mode for observation). To make sure that the status information from the signal detectors is meaningful, it is important to first enable and configure the signal detectors properly, which is done through the API command `adi_adrv910x_Rx_GainControl_Configure()`. The feedback information is configured into two modes; the peak detect and peak/power detect modes. In the peak detect mode, the overrange and underrange conditions of both the APD and HB detectors are provided through the DGPIO pins. In the peak and power detect mode, the overrange and underrange conditions of the power detector and the overrange and underrange conditions of the APD are provided through the DGPIO pins.

GAIN CONTROL DETECTORS

Analog Peak Detector (APD)

The analog peak detector is in the analog domain following the TIA filter and before the ADC input. It functions by comparing the signal level to programmable thresholds. When a threshold is exceeded a programmable number of times in a gain update period, the detector flags an overrange condition.

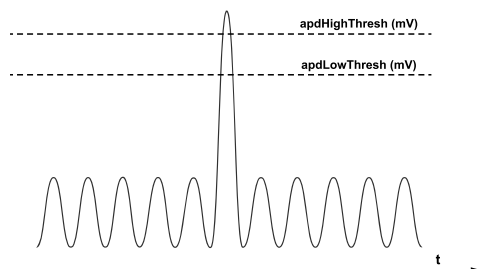


Figure 103. Analog Peak Detector Thresholds

There are two APD thresholds, as shown in [Figure 103](#). These thresholds are contained in the `agcPeak` API structure, `apdHighThresh`, and `apdLowThresh`, respectively. The thresholds are typically considered relative to the full scale voltage of the ADC. To determine the setting of the APD thresholds in terms of the closest possible setting in terms of dBFS of the ADC assuming `apdHighdBFS` and `apdLowdBFS` for `apdHighThresh` and `apdLowThresh`, respectively.

The following equations can be used for RX1 APD threshold configuration code calculation.

$$apdHighThresh = \text{round}\left(\frac{\left(10^{\left(\frac{apdHighdBFS}{20}\right)} \times 850\right) \times 0.91}{15}\right) \quad (7)$$

RECEIVER GAIN CONTROL

$$apdLowThresh = \text{round}\left(\frac{\left(10^{\left(\frac{apdLowdBFS}{20}\right)} \times 850\right) \times 0.91}{15}\right) \quad (8)$$

For RX2 channel, the look-up table is used to map the APD threshold dBFS value to the configuration code.

The APD threshold must be exceeded a programmable number of times within a gain update counter period before an overrange condition occurs, and the APD threshold must not be exceeded a programmable number of times within a gain update counter period before an underrange condition occurs. Both the upper and lower thresholds have a programmable counter in the API structure, as indicated in [Table 28](#).

In the AGC mode, the APD has programmable gain attack and gain recovery step sizes. The APD is held in reset for a configurable amount of time (slowLoopSettlingDelay) following a gain change to ensure the receiver path is settled at the new gain setting.

Half-Band (HB) Peak Detector

The HB peak detector is located in the digital domain at the output of the HB filtering block. It can, therefore, also be referred to as the decimated data overload detector because it works on decimated data. Like the APD detector, it functions by comparing the signal level to programmable thresholds. It monitors the signal level by observing individual samples ($I^2 + Q^2$ or peak I/peak Q if hbOverloadPowerMode is set to '0') over a period of time and compares these samples to the threshold. If a sufficient number of samples exceed the threshold in the period of time, then the threshold is noted as exceeded by the detector. hbOverloadDurationCount controls the duration of the HB measurement, while hbOverloadThreshCount controls the number of samples that should exceed the threshold in that period.

Once the required number of samples exceeds the threshold for the duration required, the detector records that the threshold is exceeded. Same with the APD detector, the HB detector requires a programmable number of times for the threshold to be exceeded in a gain update period before it flags an overrange condition.

[Figure 104](#) shows the two-level approach, which is different from APD. It shows the gain update counter period, with the time broken into subsets of time based on the setting of hbOverloadDurationCount. Each period is considered separately, and hbOverloadThreshCount individual samples must exceed the threshold within hbOverloadDurationCount to declare an overload. These individual samples greater than the threshold are shown in purple. Two examples are shown; one where the number of samples exceeding the threshold is sufficient for the HB peak detector to declare an overload, and a second example where the number of samples exceeding the threshold is not sufficient to declare an overload. The number of overloads is counted, and if the number of overloads of the hbHighThresh exceeds the hbUpperThreshPeakExceededCount in a gain update counter period, an overrange condition is called. Likewise, if the number of overloads of the hbUnderRangeHighThresh does not exceed hbUnderRangeHighThreshExceededCount, an underrange condition is called. Note that if hbOverloadDurationCount is set to equal to the time duration of one sample and hbOverloadThreshCount is set to one, the HB two-level approach becomes similar to the APD algorithm.

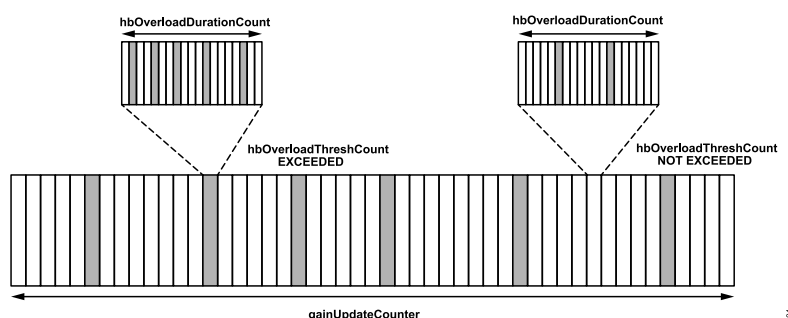


Figure 104. HB Detector's Two level approach for an Overload condition

The HB detector has a number of programmable thresholds. Some of these thresholds are only used in the fast recovery mode of the peak detect AGC configuration, as summarized in

Table 32. HB Overload Thresholds

HB Threshold	Usage
hbHighThresh	Used for gain attack in both peak and peak/power detect AGC modes.
hbUnderRangeHighThresh	Used for gain recovery in peak detect AGC mode. In peak/power detect AGC mode it is used to prevent overloads during gain recovery.

RECEIVER GAIN CONTROL

Table 32. HB Overload Thresholds (Continued)

HB Threshold	Usage
hbUnderRangeMidThresh	Used only when the fast recovery option of the peak detect AGC mode is being utilized.
hbUnderRangeLowThresh	Used only when the fast recovery option of the peak detect AGC mode is being utilized.

For more details of how the AGC uses these thresholds, see the relevant sections in this document. The thresholds are related to an ADC dBFS value using the following equation.

$$hbHighThresh = \text{round}\left(16384 \times 10^{\left(\frac{hbHigh\ dBFS}{20}\right)}\right) - 3 \quad (9)$$

$$hbUnderRangeHighThresh = \text{round}\left(16384 \times 10^{\left(\frac{hbUnderRangeHigh\ dBFS}{20}\right)}\right) - 3 \quad (10)$$

Each threshold has an associated counter such that an overrange condition is not flagged until the threshold is exceeded this amount of times in a gain update period, as indicated in [Table 28](#).

In the AGC mode, the HB peak detector has programmable gain attack and gain recovery step sizes.

Power Detectors

The power measurement block measures the RMS power of the incoming signal at the output of the HB filtering block or at the RSSI block. The number of samples used in the power measurement calculation is configurable using the powerMeasurementDuration API parameter.

$$\text{Power Meas Duration (Rx Sample Clocks)} = 8 \times 2^{\text{powerMeasurementDuration}} \quad (11)$$

where:

receiver Sample Clocks is the number of clocks at the power measurement location.

It is important that this duration does not exceed the gain update counter. The gain update counter resets the power measurement block, and therefore, a valid power measurement must be available before this event. In the case of multiple power measurements in a gain update period, the AGC uses the last fully completed power measurement, and any partial measurements are discarded.

Note that, currently, only inner thresholds of power detector are used. The inner thresholds defined in API are related to an ADC dBFS value using the following equations.

$$\text{underRangeHighPowerThresh} = -(\text{underRangeHighPowerThresh_dBFS}) \quad (12)$$

$$\text{overRangeLowPowerThresh} = -(\text{overRangeLowPowerThresh_dBFS}) \quad (13)$$

The power measurement block has a dynamic range of 60dB. Signals lower than -60dBFS may not be measured accurately. The power measurement could be read through the API function `adi_adrv910x_Rx_DecimatedPower_Get()`.

Detector Overload and QEC

If the device is operating in the AGC mode when an overload is detected, the Rx QEC calibration is frozen. The AGC can adjust the Rx Gain Index to bring the received signal to a level under the overload threshold so that the input signal no longer triggers the overload detector. Then, Rx QEC calibration resumes.

If the device is operating in the MGC mode when an overload is detected, the Rx QEC calibration is frozen as well. However, as the device is operating in the MGC mode, if the input signal remains at a high level above the overload threshold, then Rx QEC remains frozen and does not recover until the input signal is below the overload threshold.

AGC CLOCK AND GAIN BLOCK TIMING

The AGC clock drives the AGC state machine. In the ADRV910x device, the AGC clock might vary from profile to profile.

The AGC state machine contains three states: the gain update counter, followed by slow loop settling (SLS) delay, and 5 AGC clock cycles delay. The total time between the gain updates (gain update period) is a combination of `slowLoopSettlingDelay` and 5 AGC clock cycles. Note that the first `slowLoopSettlingDelay` in darker gray is a part of the gain update counter.

RECEIVER GAIN CONTROL

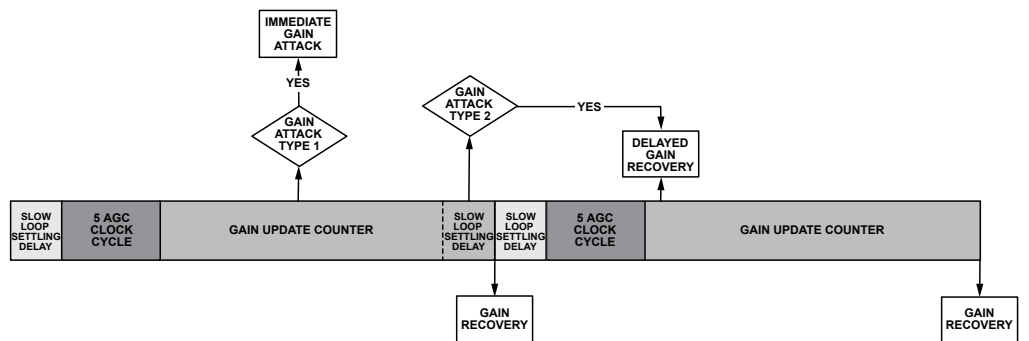


Figure 105. Delayed Gain Attack for Non-delayed Gain Recovery

Figure 105 outlines the operation of the AGC state machine. The diagram outlines possible gain change scenarios rather than a practical example of AGC operation. The possible gain change scenarios are described as follows:

- AGC gain attack within gain update counter, but more than an SLS delay before the gain update counter expiry: Because the SLS is typically several orders of magnitude smaller than the gain update counter, this is the most common gain decrement scenario. This type of AGC gain attack is denoted gain attack type 1, as shown in Figure 105.
- AGC gain attack within gain update counter, but within an SLS delay before the gain update counter expiry: This is a special case, which rarely occurs in applications. This type of AGC gain attack is denoted gain attack type 2, as shown in Figure 105.
- AGC gain recovery at the end of the gain update counter: Note that when the fast recovery is enabled, the gain update counter is substituted with the low underrange interval.

The gain attack may occur within the gain update counter when the fast attack is enabled. A gain recovery event may only occur at the end of the gain update counter (or low underrange interval in the “fast recovery” mode). This is mainly to align the gain recovery (for the desired signal) with the frame or subframe boundary. After a gain attack, a gain change counter with a value equal to the SLS delay is started. No further gain attacks are allowed while this counter is running. This allows to set the minimum time between gain changes. However, the gain change counter also prevents the AGC from moving from the gain update counter state to the slow loop settling delay state as it must wait until the expiry of the SLS delay. Therefore, if a gain attack occurred very close to the end of the gain update counter state, the gain change counter delays the start of the SLS state and shifts the gain recovery event, as shown in Figure 106, whereas in Figure 105, the gain recovery event is always aligned with the end of the gain update counter period.

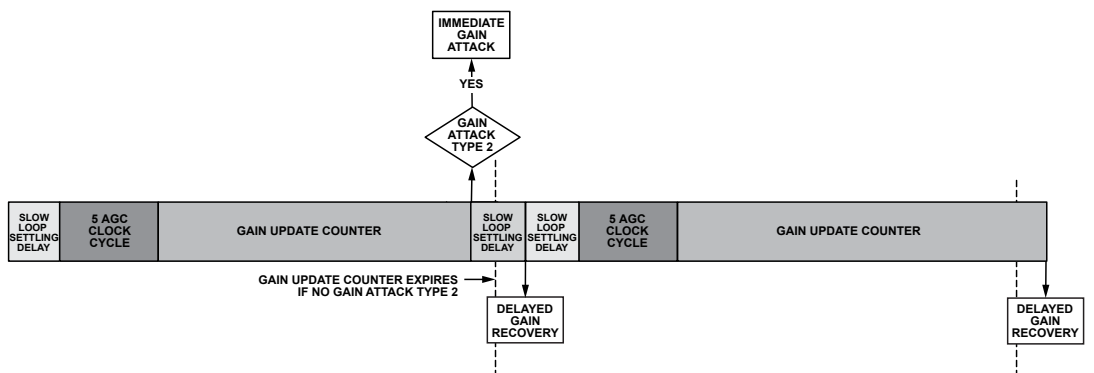


Figure 106. Immediate Gain Attack Causing Delayed Gain Recovery.

To prevent this happening and to maintain a perfectly periodic gain recovery event, the gain attacks are prevented from happening towards the end of the gain update counter state, as shown in Figure 105. If a gain attack happens in this period, it is delayed until the start of the next gain update counter state. This can cause gain attacks to be held off for up to $2 \times \text{SLS} + 5$ delay; therefore, it is recommended to keep the SLS delay as short as possible to minimize the gain attack delay. Note that it is possible to disable this blocking feature, thus allowing gain attacks to occur anywhere within the gain update counter state. However, the periodicity of the gain recovery event is no longer guaranteed as gain attacks towards the end of the gain update counter state causes the gain recovery event to be delayed, as shown in Figure 106.

At the expiry of the gain update counter (or low underrange interval in the “fast recovery” mode), all measurement blocks are reset, and any peak detector counts are reset back to zero. When the receiver is enabled, the counter begins. This might mean that its expiry is at

RECEIVER GAIN CONTROL

an arbitrary phase to the slot boundaries of the signal. The expiry of the counter is aligned to the slot boundaries by setting the parameter `enableSyncPulseForGainCounter`. While this bit is set, the AGC monitors a DGPIIO pin to find a synchronization pulse. This pulse causes the reset of the counter at this point; hence, if the user supplies at the DGPIIO pulse time aligned to these slot boundaries, the expiry of the counter is aligned to the slot boundaries. Note: this function is not yet supported.

For example, considering the 100 μ s gain update period and a 184.32 MHz AGC clock, 18,432 AGC clocks exist in the gain update period.
 $\text{Gain Update Period (AGC Clocks)} = 184.32 \text{ MHz} \times 100 \mu\text{s} = 18,432$

As noted, the full gain update period is the sum of the `gainUpdateCounter`, `slowLoopSettlingDelay`, and a number of AGC clock cycles. If the `slowLoopSettlingDelay` is set to 4, set the gain update counter to 18,423 from the following calculation.

$$\text{Gain Update Period (AGC Clocks)} = \text{gainUpdateCounter} + 2 * \text{slowLoopSettlingDelay} + 5 \quad (14)$$

$$\text{Gain Update Period (AGC Clocks)} = 18,423 + 4 + 5 = 18,432$$

When the receiver is enabled, `attackDelay_us` keeps the AGC inactive for a number of the AGC clock cycles. This means the user can specify one delay for AGC reaction when entering the receive mode and another after the gain change occurs (`slowLoopSettlingDelay`).

Fast Attack Mode Gain Change Speed

Figure 97 shows how AGC reacts in “fast attack” mode which allows a number of continuously gain changes when interferer presents. The time duration between two gain change events is determined by the APD/HB2 peak detector settings and the slow loop Settling delay setting.

As previously discussed, the APD overrange occurs when input signal exceeds the `apdHighThresh` for `apdUpperThreshPeakExceededCnt` times, HB2 overrange occurs when input signal exceeds the `hbHighThresh` for `hbUpperThreshPeakExceededCnt` times. Assuming the interferer signal persists, APD overrange flag could be set after “`apdUpperThreshPeakExceededCnt`” AGC clock cycles, HB2 overrange flag is set after “`hbOverloadDurationCount`” * “`hbUpperThreshPeakExceededCnt`” AGC clock cycles. AGC state machine waits 2*“`slowLoopSettlingDelay`” AGC clock cycles before the AGC gain changes again. Therefore, in fast attack mode, the shortest time duration between two gain change in fast attack mode is:

$$\text{Minimum (apdUpperThreshPeakExceededCnt, hbOverloadDurationCount * hbUpperThreshPeakExceededCnt) + 2*slowLoopSettlingDelay + 3.}$$

Where the equation is denoted by AGC clock cycles.

ANALOG GAIN CONTROL API PROGRAMMING

The Rx gain control mode is configured as the MGC or AGC mode. In both the modes, the API function `adi_adrv910x_Rx_GainControl_Configure()` is used to configure the gain control blocks, such as the peak detectors and power detector for a specific channel and DGPIIO pins to retrieve the signal detectors information. These detectors are used not only in the AGC mode but also in the MGC mode to feed detectors status information. Note that although signal detector information is critical for the MGC mode, it can also be obtained in the AGC mode for observation and debugging.

The next section discusses the composition of the gain control configuration structure `adi_adrv910x_GainControlCfg_t` in detail. Once it is configured, the desired gain control mode can be enabled using `adi_adrv910x_Rx_GainControl_Mode_Set()` API.

If selecting the MGC mode, as discussed, manually control the gain through API commands or DGPIIO pins. In the API command mode, select a gain index in the gain table through the API function `adi_adrv910x_Rx_Gain_Set()`. The API function `adi_adrv910x_Rx_Gain_Get()` can read back the gain index selected for a channel.

Gain Control Data Structures

The data structure `adi_adrv910x_GainControlCfg_t` contains all the gain control parameters and its substructures, `adi_adrv910x_PeakDetector_t`, `adi_adrv910x_PowerDetector_t`, and `adi_adrv910x_ExtLna_t`. Table 33 to Table 35 briefly explain each parameter in which most of them have been referred in the previous section.

Table 33. `adi_adrv910x_GainControlCfg_t` Structure Definition

Parameter	Description	Min Value	Max Value	Default Value
<code>peakWaitTime</code>	Number of gain control clock cycles to wait before enabling peak detectors after a gain change.	0	31	4
<code>maxGainIndex</code>	Maximum gain index allowed. Must be greater than <code>minGainIndex</code> and be a valid gain index.	187	255	255
<code>minGainIndex</code>	Minimum gain index allowed. Must be less than <code>maxGainIndex</code> and be a valid gain index.	187	255	187

RECEIVER GAIN CONTROL

Table 33. adi_adrv910x_GainControlCfg_t Structure Definition (Continued)

Parameter	Description	Min Value	Max Value	Default Value
gainUpdateCounter	Used as a decision period, with the detectors reset on this period. Gain changes in the AGC mode can also be synchronized to this period (the expiry of this counter).	Depends on Overload Detector Settings	4194303 AGC Clock Cycles	11520
attackDelay_us	The duration of AGC is held in reset when the Rx path is enabled.	0	63	10
slowLoopSettlingDelay	Number of AGC clock cycles to wait after a gain change before the AGC changes gain again.	0	127	16
lowThreshPreventGainInc	Only relevant in Peak and Power Detect AGC operation. 1: If AGC is in Peak and Power Detect Mode, then gain increments requested by the power detector are prevented if there are sufficient peaks (APD/HB Low Threshold Exceeded Count) above the apdLowThresh or hbUnderRangeHighThresh. 0: apdLowThresh and hbUnderRangeHighThresh don't cares for gain recovery.	0	1	0
changeGainIfThreshHigh	Applicable in both peak and peak and power detect modes. Refer Table 29 for more information.	0	3	3
agcMode	1: Peak Detect mode 0: Peak Power Detect mode.	0	1	0
resetOnRxon	1: AGC state machine is reset when Rx is disabled. The AGC gainsetting uses the resetOnRxonGainIndex" after resuming the operation. 0: AGC state machine maintains its state when Rx is disabled and the last AGC gain index is used after resuming the operation.	0	1	0
resetOnRxonGainIndex	The AGC index to start with when "resetOnRxon" is set as 1.	187	255	255
enableSyncPulseForGainCounter	1: Allows synchronization of AGC Gain Update Counter to the time-slot boundary. GPIO setup required. 0: AGC Gain Update Counter free runs.	0	1	0
enableFastRecoveryLoop	1: Enables the fast recovery AGC functionality using the HB overload detector. Only applicable in Peak Detect Mode. 0: AGC fast recovery is not enabled.	0	1	0
power	Structure containing all the power detector settings.	N/A	N/A	N/A
peak	Structure containing all the peak detector settings.	N/A	N/A	N/A
extLna	Structure containing all external LNA settings	N/A	N/A	N/A
rxQECFreezeEnable	RXQEC freeze enable/disable	0	1	0
gpioFreezePin	GPIO pin to activate to freeze AGC	0 (not assigned)	TBD	0

Table 34. adi_adrv910x_PowerDetector_t Structure Definition

Parameter	Description	Min Value	Max Value	Default Value
powerEnableMeasurement	1: Power Measurement block enabled. 0: Power Measurement block disabled.	0	1	1
underRangeHighPowerThresh	Threshold (negative sign assumed) which defines the lower boundary on the stable region of the power detect gain control mode.	0	127	10
underRangeLowPowerThresh	Offset (dB, negative sign assumed) from underRangeHighPowerThresh which defines the outer boundary of the power based AGC convergence. Typically, recovery would be set to be larger steps than when the power measurement is less than this threshold.	0	15	4
underRangeHighPowerGainStepRecovery	The number of indices the gain index pointer is increased (gain increase) in the event of the power measurement being less than underRangeHighPowerThresh but greater than underRangeLowPowerThresh.	0	31	2
underRangeLowPowerGainStepRecovery	The number of indices the gain index pointer is increased (gain increase) in the event of the power measurement being less than underRangeLowPowerThresh.	0	31	4
powerMeasurementDuration	Number of IQ samples on which to perform the power measurement. The number of samples corresponding to the 4-bit word is $8 \times 2^{\text{powerMeasurementDuration}[3:0]}$. This value must be less than AGC Gain Update Counter.	0	31	10
powerMeasurementDelay	Measurement delay between successive measurement	0	255 AGC clock cycles	2
rxTddPowerMeasDuration	Following an Rx Enable, the power measurement block is requested to perform a power measurement for a specific period of a frame. This is applicable in TDD modes. This	0	65535 AGC clock cycles	0

RECEIVER GAIN CONTROL

Table 34. *adi_adrv910x_PowerDetector_t Structure Definition (Continued)*

Parameter	Description	Min Value	Max Value	Default Value
	parameter sets the duration of this power measurement. A value of 0 causes the power measurement to run until the next gain update counter expiry.			
rxTddPowerMeasDelay	Following an Rx Enable, the power measurement block can be requested to perform a power measurement for a specific period of a frame. This is applicable in TDD modes. This parameter sets the delay between the Rx Enable and the power measurement starting on Rx.	0	65535 AGC clock cycles	0
overRangeHighPowerThresh	Offset (dB, positive sign assumed) from threshold overRangeLowPowerThresh which defines the outer boundary on the stable region of the power detect gain control mode. Typically attack would be set to be larger steps than when the power measurement is greater than this threshold.	0	15	0
overRangeLowPowerThresh	Threshold (negative sign assumed) which defines the upper boundary on the stable region of the power detect gain control mode.	0	127	7
overRangeHighPowerGainStepAttack	The number of indices that the gain index pointer should be decreased (gain reduction) in the event of the power measurement being greater than overRangeHighPowerThresh.	0	31	4
overRangeLowPowerGainStepAttack	The number of indices that the gain index pointer should be decreased (gain decrease) in the event of the power measurement being less than OverRangeHighPowerThresh but greater than OverRangeLowPowerThresh.	0	31	4
feedback_inner_high_inner_low	A pair of DGPI pins to retrieve the gain change information and power detector inner low threshold not exceeded status	0(not assigned)	19	0
feedback_apd_high_apd_low	A pair of DGPI pins to retrieve the power detector inner high threshold exceeded status and apd high threshold counter exceeded status	0(not assigned)	19	0

Table 35. *adi_adrv910x_PeakDetector_t Structure Definition*

Parameter	Description	Min Value	Max Value	Default Value
agcUnderRangeLowInterval	This sets the time constant (in AGC clock cycles) that the AGC recovers when the signal peaks are less than hbUnderRangeLowThresh. Only applicable when the fast recovery option is enabled in Peak Detect AGC mode.	Depends on HB detector settings	65535	50
agcUnderRangeMidInterval	This sets the time constant (in AGC clock cycles) that the AGC recovers when the signal peaks are less than hbUnderRangeMidThresh. Calculated as (underRangeMidInterval + 1) × underRangeLowInterval. Only applicable when the fast recovery option is enabled in the peak detect AGC mode.	0	63	2
agcUnderRangeHighInterval	This sets the time constant (in AGC clock cycles) that the AGC recovers when the signal peaks are less than hbUnderRangeHighThresh. Calculated as (underRangeHighInterval + 1) × underRangeMidInterval. Only applicable when the fast recovery option is enabled in the peak detect AGC mode.	0	63	4
apdHighThresh	This sets the upper threshold of the analog peak detector. When the input signal exceeds this threshold a programmable number of times (set by its corresponding overload counter) within a gain update period, the overload detector flags. In AGC modes, the gain is reduced when this overload occurs.	apdLowThresh	63	21
apdLowThresh	This sets the lower threshold of the analog peak detector. When the input signal exceeds this threshold a programmable number of times (set by its corresponding overload counter) within a gain update period, the overload detector flags. In the peak AGC mode, the gain is increased when this overload is not occurring. In the power AGC mode, this threshold prevents further gain increases if the lowThreshPreventGainInc bit is set.	0	apdHighThresh	12
apdUpperThreshPeakExceededCount	Sets number of peaks to detect above apdHighThresh to cause an APD High Over Range Event. In AGC modes, this will result in a gain decrement set by apdGainStepAttack.	0	255	6
apdLowerThreshPeakExceededCount	Sets number of peaks to detect above apdLowThresh to cause an APD Low Overload Event. In Peak Detect AGC mode, if an APD Low Overload Event is not occurring then this will result in a gain increment set by apdGainStepRecovery.	0	255	3
apdGainStepAttack	The number of indices the gain index pointer is decreased in the event of an APD high overrange in AGC modes.	0	31	2

RECEIVER GAIN CONTROL

Table 35. adi_adrv910x_PeakDetector_t Structure Definition (Continued)

Parameter	Description	Min Value	Max Value	Default Value
apdGainStepRecovery	The number of indices the gain index pointer is increased in the event of an APD underrange event occurring in the peak detect AGC mode.	0	31	2
enableHbOverload	1: HB Overload Detector enabled 0: HB Overload Detector disabled	0	1	1
hbOverloadDurationCount	The number of clock cycles (at the AGC Clock rate) within which hbOverloadThreshCount must be exceeded for an overload to occur. An HB overload flag is only raised when the number of these overloads exceeds hbUpperThreshPeakExceededCnt or hbLowerThreshPeakExceededCnt within a gain update period	0	7	1
hbOverloadThreshCount	Sets the number of individual samples exceeding hbHighThresh or hbLowThresh necessary within hbOverloadDurationCount for an overload to occur. The HB overload flag is only raised when the number of these overloads exceeds hbUpperThreshPeakExceededCnt or hbLowerThreshPeakExceededCnt within a gain update period.	1	15	1
hbHighThresh	This sets the upper threshold of the HB detector.	0	16383	13044
hbUnderRangeLowThresh	This sets the lower threshold of the HB underrange threshold detectors. Used only when the fast recovery option of the peak detect AGC mode is being used.	0	16383	5826
hbUnderRangeMidThresh	This sets the middle threshold of the HB underrange threshold detectors. Used only when the fast recovery option of the peak detect AGC mode is being used.	0	16383	8230
hbUnderRangeHighThresh	Peak detect mode: Threshold used for gain recovery. Peak detect with fast recovery mode: This sets the highest threshold of the HB underrange threshold detectors. Power detect mode: Threshold used to prevent further gain increases if flowThreshPreventGainInc is set.	0	16383	7335
hbUpperThreshPeakExceededCount	Sets number of individual overloads above hbHighThresh (number of times hbOverloadThreshCount is exceeded in hbOverloadDurationCount) to cause an HB High overrange event. In AGC modes, this results in a gain decrement set by hbGainStepAttack.	0	255	6
hbUnderRangeHighThreshExceededCount	Sets number of individual overloads above hbUnderRangeHighThresh (number of times hbOverloadThreshCount is exceeded in hbOverloadDurationCount) to cause an HB underrange high threshold overload event. In the peak detect AGC mode, not having sufficient peaks to cause the overload is flagged as an underrange event and the gain is recovered by hbGainStepHighRecovery.	0	255	3
hbGainStepHighRecovery	The number of indices the gain index pointer is increased in the event of an HB underrange high threshold underrange event.	0	31	2
hbGainStepLowRecovery	Only applicable in the fast recovery mode of peak detect AGC. This sets the number of indices the gain index pointer is increased in the event of an HB underrange low threshold underrange Event.	0	31	6
hbGainStepMidRecovery	Only applicable in the fast recovery mode of peak detect AGC. This sets the number of indices the gain index pointer is increased in the event of an HB underrange mid threshold underrange Event.	0	31	4
hbGainStepAttack	The number of indices the gain index pointer is decreased in the event of an HB high threshold overrange event in AGC modes.	0	31	2
hbOverloadPowerMode	Sets the measurement mode of the HB detector. HB uses $I^2 + Q^2$ when set to 1. Otherwise the HB uses $\max(I, Q)$ per sample.	0	1	0
hbUnderRangeMidThreshExceededCount	Only applicable in the fast recovery mode of peak detect AGC. Sets number of individual overloads above hbUnderRangeMidThresh (number of times hbOverloadThreshCount is exceeded in hbOverloadDurationCount) to cause an HB underrange mid threshold overload event. In the peak detect AGC mode, not having sufficient peaks to cause the overload is flagged as an underrange event and the gain is recovered by hbGainStepMidRecovery	0	255	3
hbUnderRangeLowThreshExceededCount	Only applicable in the fast recovery mode of peak detect AGC. Sets number of individual overloads above hbUnderRangeLowThresh (number of times hbOverloadThreshCount is exceeded in hbOverloadDurationCount) to cause an HB underrange low threshold overload event. In the peak detect AGC mode, not having sufficient peaks to cause the overload is flagged as an underrange event and the gain is recovered by hbGainStepLowRecovery.	0	255	3
feedback_apd_low_hb_low	A pair of DGPI pins to retrieve the HB low threshold counter exceeded status and APD low threshold counter exceeded status	0(not assigned)	20	0
feedback_apd_high_hb_high	A pair of DGPI pins to retrieve the HB high threshold counter exceeded status and APD high threshold counter exceeded status	0(not assigned)	20	0

RECEIVER GAIN CONTROL

There is a set of receiver gain control APIs to interact with the ADRV910x device. The previous sections mention some of them. [Table 36](#) summarizes the list of API functions with a brief description for each one. Refer to the Doxygen document for more up-to-date information and detailed descriptions.

Table 36. List of Rx Gain Control

Rx Gain API Function Name	Description
adi_adrv910x_Rx_GainControl_Mode_Set	Configures the Rx gain control mode for a specific channel
adi_adrv910x_Rx_GainControl_Mode_Get	Retrieves the currently configured Rx gain control mode
adi_adrv910x_Rx_Gain_Get	Reads the Rx Gain Index for the requested Rx channel
adi_adrv910x_Rx_Gain_Set	Sets the current MGC Gain Index for the requested Rx channel
adi_adrv910x_Rx_GainTable_Write	Programs the gain table settings for Rx channels
adi_adrv910x_Rx_GainTable_Read	Reads the gain table entries for Rx channels requested
adi_adrv910x_Rx_DecimatedPower_Get	Gets the decimated power for the specified channel
adi_adrv910x_Rx_GainControl_Configure	Sets up the device Rx Gain Control for a specified channel
adi_adrv910x_Rx_GainControl_Inspect	Inspects the device Rx Gain Control for a specified channel
adi_adrv910x_Rx_GainControl_MinMaxGainIndex_Set	Sets the min/max gain indexes for gain control operation for the specified channel
adi_adrv910x_Rx_GainControl_MinMaxGainIndex_Get	Gets the min/max gain indexes for gain control for the specified channel
adi_adrv910x_Rx_GainControl_PinMode_Configure	Configures gain control for MGC pin mode
adi_adrv910x_Rx_GainControl_PinMode_Inspect	Inspects gain control configurations for MGC pin mode

DIGITAL GAIN CONTROL AND INTERFACE GAIN

The digital gain control has two major purposes: gain correction to correct the small step size inaccuracy in the analog front-end attenuation, and gain compensation to compensate for the entire analog front-end attenuation. In the gain compensation mode, for example, if 5 dB analog attenuation is applied at the front end of the device, then 5 dB of digital gain is applied. This ensures the digital data is representative of the RMS power of the signal at the receiver input port (plus the nominal receiver analog gain) so that any internal front-end attenuation changes in the device to prevent ADC overloading are transparent to the baseband processor. In this way, the device's AGC is used to react quickly to incoming blockers without the baseband processor needing to track the current gain index for the level of the received signal at the input to the device for signal strength measurements. The gain compensation mode can be used in either AGC or MGC mode. The digital gain can compensate the internal analog attenuator and an external gain component (such as a DSA or LNA). After the digital gain compensation, the signal power should only depend on the input signal power.

The receiver gain table controls the digital gain block, as mentioned earlier. Note that different digital gain is applied when configured in the gain correction or gain compensation mode. The receiver gain table has a unique front-end attenuator setting with a corresponding amount of digital gain, programmed at each index of the table, as shown in [Table 25](#).

Around the end of the receiver datapath, the receiver interface gain is further applied using a "Slicer" block for two major purposes:

- ▶ Avoid digital saturation due to the bit-width limitation of the data port in the gain compensation mode,
- ▶ Ensure the RX output noise is limited only by the receiver thermal noise and is unaffected by datapath quantization noise.

When the gain compensation mode is used, any analog attenuation at ADRV910x front end is compensated by a corresponding digital gain, such that the total gain of receiver is no change. At the large signal input scenario, the applied analog attenuation makes sure the large signal won't saturate the RX ADC, but the compensated digital signal might clip the receiver 16-bit data sample output. For example, assuming there is a 5 dBm signal at the ADRV910x RX input port, and the full scale input power of receiver is about -12.5 dBm which represents the 0 dBFS signal at digital domain, the signal level at receiver output is about 17.5 dBFS after gain compensation, this causes the clipping at receiver output with 16-bit output width. Therefore, a negative 18dB interface gain should be applied to avoid the signal clipping before sending the RX signal output to baseband processor.

On the other hand, for the narrow band profiles with the 16-bit data sample output, the receiver quantization noise is spread over a smaller bandwidth (12KHz in the 24Ksps output sample rate example) and can thus be the dominant of the total receiver noise, this will degrade the RX output SNR, especially for the small signal level scenario, the SNR degradation will affect the RX sensitivity level. By adding the interface gain in the datapath, the thermal noise can be made to dominate the total Rx noise, so the RX output noise won't be limited by the 16-bit data sample output. Basically, applying the interface gain in the narrow band scenario is equivalent to extend the RX output bit width, therefore to get the better noise floor at receiver output.

RECEIVER GAIN CONTROL

Additionally, ADRV910x receiver has the 22 bit output mode which sends all the receive datapath bits to baseband processor without bit truncating, the quantization noise of this mode is much lower than the receiver thermal noise and no interface gain is needed. For more details, see the [Enhanced Rx SSI Mode](#) section.

[Figure 95](#) RX Datapath and Gain Control blocks includes the digital gain control portion of the receiver chain. In NB and WB applications, the the digital gain control is actually performed at different stages of the receiver data chain to achieve optimal performance, which is simplified in [Figure 95](#) “Decimation filter stage 2”. The interface gain (slicer) should depend on the desired signal power alone and must be done only when all the interfering signals are filtered out, therefore close to the end of the datapath. The slicer operation can either be controlled automatically by the device internally or externally through API commands.

When controlled internally (automatic interface gain), the slicer gain is controlled via an PS1 ARM controlled process based on the RSSI measurement results, the configurable maximum signal level target and the signal PAR. The RSSI has configurable measurement duration and moving average duration. The interface gain is decreased when the average RSSI measurement plus current slicer gain is bigger than the addition of maximum signal level target and signal PAR, and interface gain will be increased when the average RSSI measurement plus current slicer gain and 6dB is smaller than the addition of maximum signal level target and signal PAR. There is also a “fast attack” option to bypass the moving average and use the current RSSI for the quickly gain decreasing. The fast attach option can quickly react the signal level change to avoid the saturation but not sacrifice normal tracking during slicer recovery times.

However, it takes some time to react to the signal level change in a normal operation mode. Therefore, the fast attack algorithm is designed to solve this issue. Provide the maximum signal level target and the signal peak to average power ratio (PAPR) to help the automatic interface gain control algorithm to produce the proper interface gain values.

Digital Gain Control Modes

Mode 1: Gain Correction With Internal Interface Gain Control

In this mode, the digital gain block is used for gain correction. It applies a small amount of digital gain/attenuation to provide consistent gain steps in a gain table. The premise is that because the analog attenuator does not have consistent steps in dB across its entire range, the digital gain block is used to even out the steps for consistency (the default table uses the digital gain block to provide consistent 0.5 dB steps).

With internal interface gain control, the device automatically applies the interface gain determined by RSSI, which measures the input signal power right before the slicer. Note that in the gain correction mode, interface gain less than 0 is not needed as the receiver output level should not exceed 0 dBFS through either AGC or MGC. When in NB applications, the interface gain range can be from 0 dB to 18 dB in 6 dB step size. In WB applications, as discussed earlier, the interface gain is always 0.

After applying the interface gain, the signal is provided to the data port. The baseband processor can retrieve the interface gain through API commands to scale the power of the received signal to determine the power at the input to the device (or at the input to an external gain element if included as a part of the digital gain compensation).

Mode 2: Gain Correction With External Interface Gain Control

This mode is similar to mode 1, except the interface gain is controlled manually by BBIC via API. Similarly, when in NB applications, the interface gain range is from 0 dB to 18 dB in 6 dB step size, while in WB applications, the interface gain is fixed at 0 dB.

Mode 3: Gain Compensation With Internal Interface Gain Control

In Gain compensation mode, the device is loaded with the gain tables that apply compensation for the analog front-end attenuation. Thus, as the analog front-end attenuation is increased, equal amount of digital gain is applied. With internal interface gain control, the interface gain is automatically applied based on the RSSI measurement result. If the power level is too high, the slicer shifts the signal properly before sending it to the data port to avoid saturation.

Let us look at a slicer example that considers three different input signal power levels. The power level 1 fits a data length of 16 bit-width. Power level 2 is 0 dB to 6 dB higher than power level 1, which increases the bit-width by 1. Power level 3 is 6 dB to 12 dB higher than power level 1, which further increases bit width by 1. [Figure 107](#) outlines this effect, with gray boxes indicating the valid (used) bits in each case.

RECEIVER GAIN CONTROL

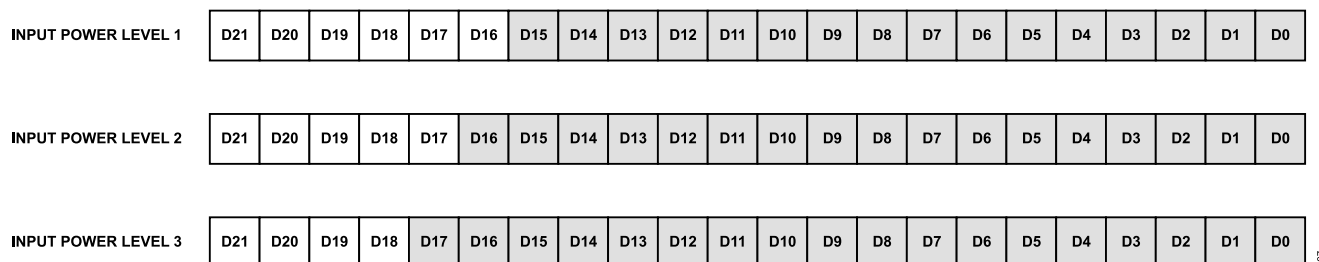


Figure 107. Bit Width of Input Signal with Increasing Power Levels

The slicer is used to truncate internal datapath bit width such that it can fit into the 16 bit resolution of the data port. As the output is a shifted version of the input, the slicer can only handle gains in ± 6 dB steps.

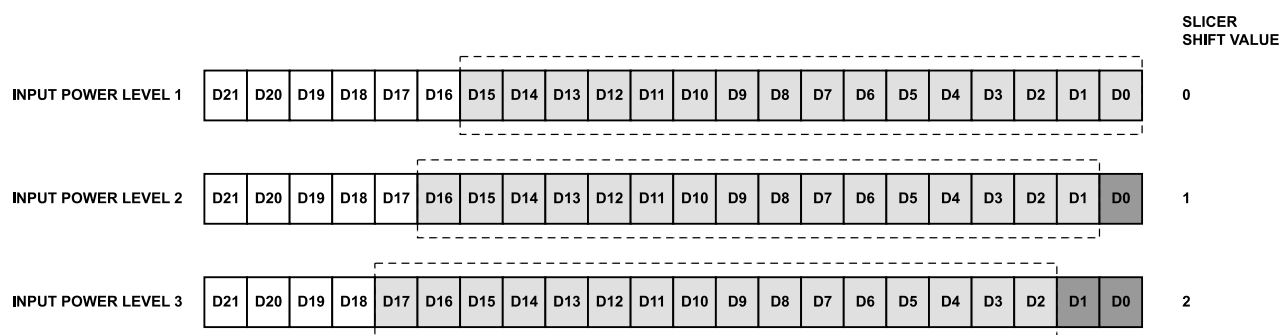


Figure 108. Slicer Bit Selection with Different Input Power Levels

Figure 108 explains the slicer operation. For power level 1, the slicer shift value is calculated as 0, so the 16-bit output data is taken from D15 – D0. As the power level increases, the bit-width of the signal increases. For power level 2, now the bit-width is 17. The slicer shift value becomes 1, so the 16-bit output data is taken from D16 – D1. This is equivalent to applying 6 dB of attenuation by a slicer, which ensures that the bit-width of the signal is 16 once more, i.e., the 16 MSBs are selected (sliced) with the LSB dropped. When the power level further increases, as power level 2, the signal bit-width becomes 18-bit. The slicer shift value becomes 2, so the 16-bit output data is taken from D17 – D2, which is equivalent to applying 12 dB of attenuation by slicer or slice the 16 MSBs dropping the 2 LSBs.

As previously has discussed, the automatic interface gain algorithm adjusts the interface gain such that the measured signal power is less than the addition of maximum signal level target and signal PAR. In gain compensation mode, the interface gain is from -36dB to 18dB for NB applications and is from -36dB to 0 dB for WB application.

Similarly, the baseband processor retrieves the interface gain through API commands to scale the power of the received signal to determine the power at the input to the device (or at the input to an external gain element if included as a part of the digital gain compensation).

Mode 4: Gain Compensation With External Interface Gain Control

This mode is similar to mode 3, except the interface gain is controlled by selecting a proper value via API by baseband processor. The baseband processor measures the input signal power or uses the power measurement done by RSSI in the device to determine the interface gain. Then, through API commands, the slicer operates in the same way as mentioned in mode 3.

Interface Gain API Programming

The API function `adi_adrv910x_Rx_InterfaceGain_Configure` configures the interface gain, the data structure `adi_adrv910x_RxInterfaceGainCtrl_t` defines the interface gain control mode, interface gain update timing, RSSI measurement settings, etc. Refer the API doxygen file for more information. This API should be set in the “Calibrated” state.

For the external interface gain mode, baseband processor can use API `adi_adrv910x_Rx_InterfaceGain_Set` to change the interface gain on the fly while the RX channels are operational.

POWER SAVING AND MONITOR MODE

The ADRV910x is a high-performance integrated transceiver SoC designed with low-power operation in mind. It provides flexibility for dynamically powering up or down TX/RX channels, with multiple power-down levels optimized for TDD operation. For extended idle periods, the device also supports deep-sleep modes to further reduce overall power consumption when signal processing is inactive.

The ADRV910x defines Channel Power Saving modes and System Power Saving modes, offering a range of operating states from low to high power savings, with corresponding short to long recovery times. These modes allow users to balance power consumption and wake-up latency according to system requirements.

The devices support three primary power saving schemes,

- Power up/down the unused channel in the calibrated state
- Dynamic inter-frame power saving runs automatically during normal TDD TX/RX operations. Channel and/or system power-saving modes are dynamically applied as required between active frames.
- Monitor Mode. Allows the host processor to move into a low power state after configuring the ADRV910x into Monitor Mode. ADRV910x then autonomously controls the dedicated hardware and timers to periodically sleep and detect.

Users should select the appropriate power-down modes and power-saving schemes based on the target application scenario.

The following sections describe each scheme in detail.

POWER SAVING MODES

Power-down modes dynamically power down and power up different levels of the ADRV910x components. [Table 37](#) describes 2 levels of Channel Power Saving modes, 2 levels of System Power Saving modes controlled externally by baseband processor and 3 levels of System Power Saving modes controlled by ADRV910x PS2, higher level power saving modes power down more components but with longer recovery time.

Table 37. Power-Down Modes and Related Power-Down Components

Power Down Modes			Channel Power Saving (CPS)		System Power Saving (No PS2) (SPS)		System Power Saving (With PS2) (SPS)		
			0 (default)	1 ¹	0	1	0	1	2
Components	TX	Analog Front End, Digital Datapath (Clock Gating only)	x	x	x	x	x	x	x
		TX RF PLL (Internal LO PLL or External VCO PLL)		x	x	x	x	x	x
		Internal LO VCO (LDO), Optional to output a AGPIO signal to power down external VCO		x	x				
	RX	Analog Front End, Digital Datapath (Clock Gating only)	x	x	x	x	x	x	x
		RX RF PLL (Internal LO PLL or External VCO PLL)		x	x				
		RX Internal LO VCO (LDO), Optional to output a AGPIO signal to power down external VCO		x	x	x	x (PS2 runs on DEV_CLK or Power down)	x (PS2 runs on DEV_CLK or Power down)	
	System	CLK PLL and CLK PLL VCO LDO			x	x	x	x	
		PS1 ARM(+memories)				x		x	x
	Approximate power up time (us)*		TBD	TBD	TBD	TBD	TBD	TBD	TBD

¹ not applicable for external LO Mode

POWER SAVING AND MONITOR MODE

POWER DOWN/UP CHANNEL IN CALIBRATED STATE

TBD.

DYNAMIC INTERFRAME POWER SAVING

Dynamic interframe power saving runs automatically during all regular TDD TX/RX operations. A higher level power-down mode can be configured to get more power saving if the application has a longer TX/RX transition time.

Channel Power Saving (CPS)

Channel power Saving saves power-on channel granularity for dynamic TDD interframe operations. The power saving triggered by the TX_EN/RX_EN rising or falling edges. Figure 109 shows TX/RX EN pins power up/down channels. The rising edge of TX_EN/RX_EN powers up the associated TX or RX channel components according to the selected power-saving mode (Mode 0 or Mode 1), while the falling edge powers down the channel components.

Timing Parameters Control section discusses the detailed control timing requirements for TDD operation. Users must consider both the channel wake-up time and the required on-air frame timing when configuring TDD operation. Proper setup of the power-saving modes and TX_EN/RX_EN timing is required to ensure correct transmitter and receiver activation relative to the frame timing requirements.

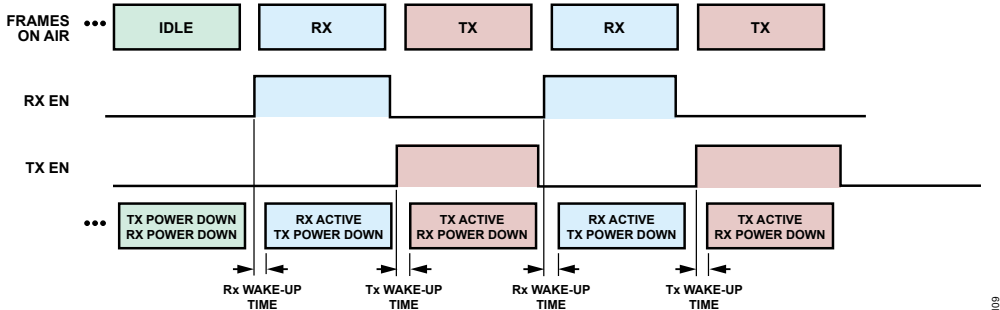


Figure 109. TX/RX EN Pins Trigger Power Saving

System Power Saving (SPS)

System Power Saving (SPS) provides additional power reduction when both TX and RX channels are idle, at the cost of longer wake-up and transition times compared to Channel Power Saving (CPS).

For ADRV910x transceiver only product variants, the SPS is controlled through a DGPIO from baseband processor. For ADRV910x PS2 product variants, the SPS is managed directly by the PS2 subsystem.

Figure 110 shows an example how CPS and SPS combine for the best power savings. Typically, CPS is used during short TX/RX inter-frame periods, while SPS is applied during longer system idle intervals when both TX and RX are inactive. SPS may also be used during extended idle periods within TX-only or RX-only operating scenarios.

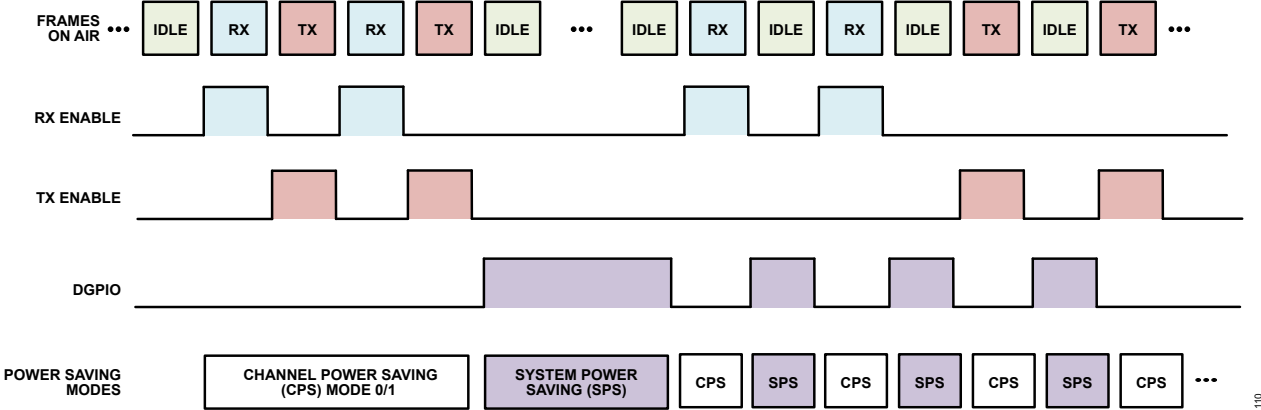


Figure 110. Combined CPS and SPS for Power Saving

POWER SAVING AND MONITOR MODE

MONITOR MODE

TBD.

DIGITAL PREDISTORTION (DPD)

BACKGROUND

It is well known that one of the main criteria of a power amplifier (PA) operation is its ability to maintain linearity, i.e. the gain is constant regardless of the input amplitude. However, in practice, a PA can only maintain linearity up to a certain input level beyond which the gain starts to lower and the PA enters into a nonlinear or compression region as shown in Figure 111. For most low-power linear amplifiers, they operate in the linear region as shown in the “LINEAR REGION” circle. Unfortunately, a PA that operates mostly in the linear region has lower efficiency. PA efficiency is defined as the ratio of output RF power to the DC supply power. Therefore, it is desirable to operate PA at high efficiency to save DC power and reduce heat dissipation.

To achieve higher PA efficiency, the highest input signal peak is usually set at around 1dB (P1dB) compression region as shown in the “1dB COMPRESSION REGION” circle in Figure 111. However, compression of the peak signals produces harmonics and hence intermodulations. Some of the intermodulations fall back right into or adjacent to the carrier spectrum, therefore not only distorting the transmit signal but also widening the spectrum of the transmit signal, so called spectral regrowth. If left untreated, the error vector magnitude (EVM) performance of the transmit signal would be degraded and the spectral regrowth would interfere adjacent channels, resulting in worse than required adjacent channel power ratio (ACPR) performance. Digital Pre-Distortion (DPD) is designed to mitigate this problem.

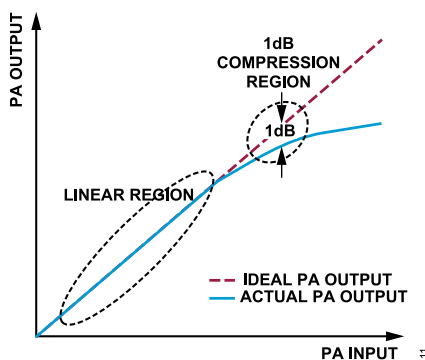


Figure 111. Ideal Power Amplifier Output vs. Actual Power Amplifier Output

ADRV910X DPD FUNCTION

The ADRV910x device provides a fully integrated DPD function that supports both narrow-band (NB) and wide-band (WB) applications in TDD only mode. It is a hardware/software combined solution which performs linearization of the PA by pre-distorting the digital transmit signal with the inverse of the PA's nonlinear characteristics. After amplifying by the PA, the pre-distortion compensates PA's nonlinearity so the amplified RF transmit signal becomes linear. Therefore, the integrated DPD solution allows PA to operate at very high efficiency while achieving a satisfactory EVM and ACPR performance.

Figure 112 depicts a high-level block diagram of the DPD algorithm. As shown in this figure, before the PA, a “Predistorter” block is added in the transmit datapath which distorts the transmit signal $d(t)$ with the inverse of the PA's nonlinear characteristics, as shown by the first Input/Output figure curve. Spectral regrowth is introduced after the pre-distortion. However, after the “pre-distorted” transmit signal $x(t)$ being amplified by the PA, the PA nonlinear characteristics, as shown by the second Input/Output figure curve cancels out the pre-distortion. Therefore, the output of the PA $y(t)$ becomes linear, as shown by the third Input/Output figure curve. In addition, the spectral regrowth after pre-distortion is also corrected. The “DPD Coefficients Computation” block is used to compute the pre-distortion parameters by utilizing “Predistorter” output signal $x(t)$ as well as the power amplifier output signal $y(t)$ through a feedback path. It models the behaviour of the PA in the reverse direction, i.e. from output to input, therefore, it characterizes the inverse of the PA nonlinearity and then feeds the parameters to the “Predistorter”.

DIGITAL PREDISTORTION (DPD)

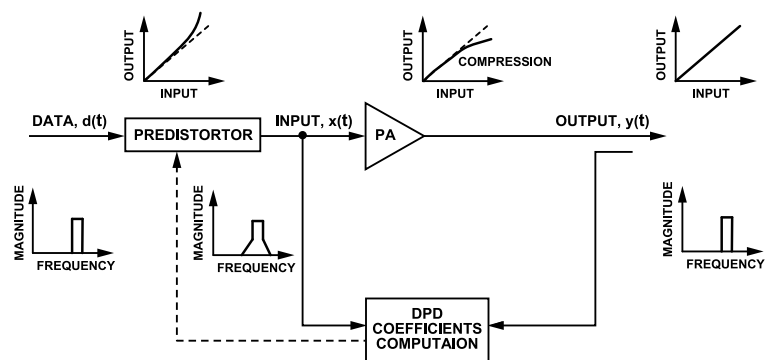


Figure 112. High Level Block Diagram of DPD Algorithm

In the ADRV910x device, DPD is considered as one of the transmitter tracking calibrations. It is a real-time signal processing with iterative updates to account for hardware variations such as temperature and power level changes. Similar to some other transmitter tracking calibrations, it requires a loopback path from the transmitter to the observation receiver (ORx) to perform the calibration. In this case, an external loop back path (ELB) is required, in which, the transmitter output signal after PA is looped back to the ORx as shown in Figure 113. The user must make sure the external loopback path is established before enabling the integrated DPD.

Note, the ADRV910x doesn't have dedicated observation receiver input RF ports, the unused RX1/RX2 ports can be repurposed as the ORx input ports, refer Figure 79, Figure 80 for the external loopback path connection details.

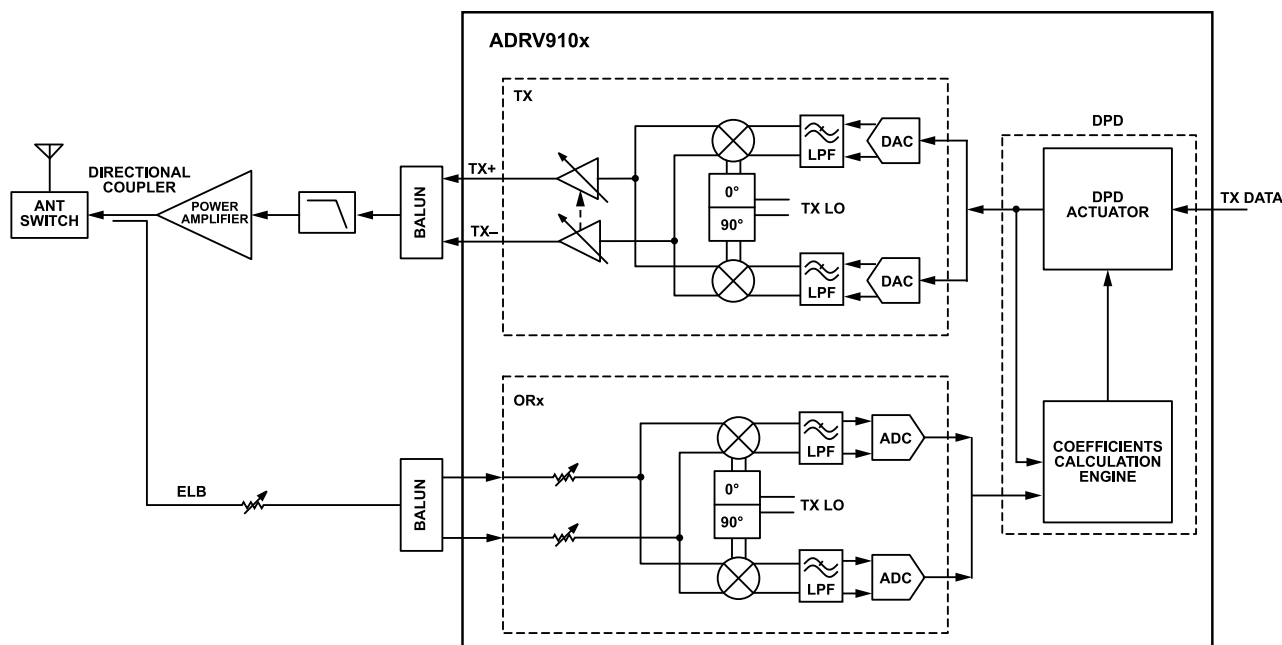


Figure 113. High level Block Diagram of ADRV910X DPD Implementation

Similar to what is shown in Figure 113, ADRV910X DPD includes two major components, a “DPD Actuator” and a “Coefficients Calculation Engine”. The “Coefficients Calculation Engine” computes the DPD coefficients periodically and then updates the “DPD Actuator” for real-time pre-distortion of the transmit signal. The pre-distortion coefficients are associated with polynomial terms defined by the PA model. To meet the real-time processing requirement, polynomial terms that are associated with a common time-delay input data are pre-computed and stored into Look-up Tables (LUT) in the “DPD Actuator”. In the device, two LUTs are used for all waveforms, one is currently being active for performing pre-distortion while the other one is being updated at the background to track the changes and replace the current LUT when ready, resulting in seamless transmit operation. “DPD Actuator” also includes a functionality to perform the calculation of the amplitude of the input signal, which is used to search the LUT. The outputs of the LUT are then multiplied with different time delayed input data according to the configured DPD model and combined to form the final pre-distorted transmit data.

DIGITAL PREDISTORTION (DPD)

ADRV910X DPD SUPPORTED WAVEFORMS

The integrated DPD supports narrowband waveforms such as TETRA. Besides that, the integrated DPD supports some wideband long-term evolution (LTE) and LTE-like waveforms with their associated operational parameters. Other wideband waveforms can be supported if the PA behavior fits the designed hardware amplifier model, as well as the sampling rates and transceiver bandwidth.

To achieve higher PA efficiency and DPD algorithm stability, a waveform with a large PAR is expected to perform crest factor reduction (CFR) in baseband processors before DPD operation. Ensure that CFR is applied to a multicarrier signal before transmission as it keeps the average power higher while maintaining the same peak back off in the digital transmit data. Also, CFR suppresses large peaks to below a preset threshold, thereby eliminating occasional large peaks that can make DPD unstable if the peak is beyond the compression threshold limit of the power amplifier. For example, an LTE signal has a PAR of about 11 dB and the PAR can be reduced by CFR to about 7 dB by trading off the EVM.

Note that ADRV910x device does NOT have the CFR functions, the baseband processor is responsible to perform CFR with an appropriate trade-off between the PAR reduction and EVM degradation before sending the transmit data to the ADRV910x.

CLOSED LOOP GAIN CONTROL (CLGC)

The sole purpose of CLGC is to maintain a fixed gain between the PA output and transmit digital input amplitude. The primary source of gain variation comes from the changes in the gain of the PA, which can vary due to transistor temperature change in response to power output in the short term, RF carrier frequency change within the bandwidth of the amplifier, and slow hardware degradation in the long term. Therefore, it is important to maintain a fixed gain in the transmit path to maintain a precise calibrated transmit power at the antenna port.

CLGC helps to achieve a fixed gain target by adjusting the transmit attenuation automatically by monitoring the gain variation of the PA. It can work with or without the DPD algorithm. When DPD is not activated, CLGC is an optional feature. It's strongly recommended to active the CLGC when DPD is activated. DPD compensates for the instantaneous compression of the peak signal by expanding the input signal so that the peak signals are linearized. When linearization is achieved, the gain in the compression region is adjusted (increased) to match the gain of the lower linear region so that the overall gain is independent of the input or output average power level. This has the advantage of having CLGC focusing on compensating only the PA gain variation mainly due to temperature change. Furthermore, by setting a proper gain target, CLGC can also help to monitor and limit the transmit power level to keep the amplifier output power from rising beyond the linearization capable power limit of the PA. Therefore, it is crucial to enable CLGC while DPD is active.

DPD/CLGC CONFIGURATION

There are two sets of DPD/CLGC parameters that should be properly configured to ensure optimal performance, one is "pre-initial" parameters, as they should be configured before performing initial calibration when the device is at the "STANDBY" state, another is "post-initial" parameters, as they should be configured after performing initial calibration when the device is at the "CALIBRATED" state. The following subsections explain these DPD/CLGC parameters in detail.

DPD/CLGC Pre-Initial Parameters Configuration

The Pre-Initial DPD parameters is set by API *adj_adrv910x_dpd_initial_configure*. Users should have a general understanding of the DPD model used in the device to properly set the pre initial parameters of DPD, the DPD model is described by the following equations:

$$x(n) = \sum_{t=0}^{T-1} \psi_t(|d(n-l_t)|) d(n-k_t) \quad (15)$$

$$\psi_t(|d(n-l_t)|) = \sum_{i=0}^7 b_{t,l_t,i} a_{t,l_t,i} |d(n-l_t)|^i \quad (16)$$

Where:

T is the total number of taps in the DPD model.

$\psi_t(|d(n-l_t)|)$ is the function implemented by the LUT for tap "t".

l_t and k_t are part of the hardware model representing the amplitude and data delay, respectively.

Optionally include/exclude each individual power term in $\psi_t(|d(n-l_t)|)$ by controlling the corresponding $b_{t,l_t,i}$, setting it to either 0 for excluding or 1 for including to better model the PA.

DIGITAL PREDISTORTION (DPD)

$a_{t,l_t,i}$ are coefficients estimated by the “Coefficients Calculation Engine” and used to generate the LUTs by the “DPD Actuator”. For $b_{t,l_t,i}$ and $a_{t,l_t,i}$, the subscript t represents the index for the tap, l_t represents the amplitude delay, and i represents the order of the power term. Only 0th to 7th order power term is supported in function $\psi_t(|d(n - l_t)|)$.

The pre initial DPD configuration parameters are shown in Table 38.

Table 38. DPD Pre Initial configuration Parameters

Parameter	Type	Description	Default	Note
enable	bool	Sets “TRUE” to place the “DPD Actuator” in the data path to prepare for DPD operation.	FALSE	
amplifierType	enum	Selects the type of amplifier ADI_ADRV910X_DPDAMPLIFIER_NONE = 0, ADI_ADRV910X_DPDAMPLIFIER_DEFAULT = 1, ADI_ADRV910X_DPDAMPLIFIER_GAN = 2	1	“1” is the only allowed PA type currently for both the LDMOS and GaN type of PA
lutSize	enum	Determines the LUT size ADI_ADRV910X_DPDLUTSIZE_256 = 0, ADI_ADRV910X_DPDLUTSIZE_512 = 1	1	
model	enum	Selects the DPD model. ADI_ADRV910X_DPDMODEL_0 = 0, ADI_ADRV910X_DPDMODEL_1 = 1, ADI_ADRV910X_DPDMODEL_3 = 3, ADI_ADRV910X_DPDMODEL_4 = 4	4	“4” is the only allowed DPD model currently. User should always choose “4”.
changeModelTapOrders	bool	Sets “TRUE” to use the model tap orders defined by “modelOrdersForEachTap”. Set “FALSE” to ignore “modelOrdersForEachTap” and use the default order.	FALSE	The default model tap order for DPD Model 4 is: [0] = 0x001F, [1] = 0x007F, [2] = 0x001F, [3] = 0x001E
modelOrdersForEachTap	array	A bit map for each of the taps in a model to indicate which power terms are included in the model. Tap 0 and 2 should have the same order.	[0] = 0x001F, [1] = 0x007F, [2] = 0x001F, [3] = 0x001E	The default bit map represents the default tap order for Model 4.
preLutScale	uint8_t (U2.1)	Describes the prescaling factor for the LUT.	2.0	Min=1.0, Max=3.75
clgcEnable	Bool	Enables CLGC functionality.	False	

The following sections describe each parameter.

enable, clgcEnable

The “enable” parameter places the “DPD Actuator” on the datapath to prepare for DPD operation. The “clgcEnable” parameter enables the CLGC functionality.

Note: Setting it to “TRUE” is necessary but not sufficient to start the DPD/CLGC operation. The DPD/CLGC starts when the corresponding tracking calibration bit is also set.

For the DPD/CLGC to work, the profile must indicate an external loopback connection, the external loopback connection should be set to “After PA” for DPD.

amplifierType

Set the PA type to ADI_ADRV910X_DPDAMPLIFIER_DEFAULT if DPD is enabled. The default PA type refers to both the LDMOS and GaN types of PA.

lutSize

The supported LUT sizes are 256 and 512. The size determines the number of entries in the DPD LUT. A more significant number of entries provides better LUT granularity.

model

Users should always set the DPD model to ADI_ADRV910X_DPDMODEL_4 only, the other models are reserved for backward compatibility, do **NOT** use them at this time. Model 4 consists of four taps (T=4), which can be described by the following equation:

DIGITAL PREDISTORTION (DPD)

$$x(n) = \sum_{t=0}^3 \psi_t(|d(n-l_t)|)d(n-k_t) \quad (17)$$

The delay for each tap is described in Table 39.

Table 39. DPD Model 4 Delays

Tap	Delay of Magnitude (<i>l</i>)	Delay of Data (<i>k</i>)
0	0	0
1	1	1
2	2	2
3	2	1

Therefore, the DPD model 4 equation can be rewritten as:

$$x(n) = \psi_0(|d(n)|)d(n) + \psi_1(|d(n-1)|)d(n-1) + \psi_2(|d(n-2)|)d(n-2) + \psi_3(|d(n-2)|)d(n-1) \quad (18)$$

Where:

$\psi_0(|d(n)|)$, $\psi_1(|d(n-1)|)$, $\psi_2(|d(n-2)|)$ and $\psi_3(|d(n-2)|)$ represents 4 Taps, generated by the LUT.

Figure 114 shows the DPD model 4 tap configuration used to generate the final predistorted data $x(t)$.

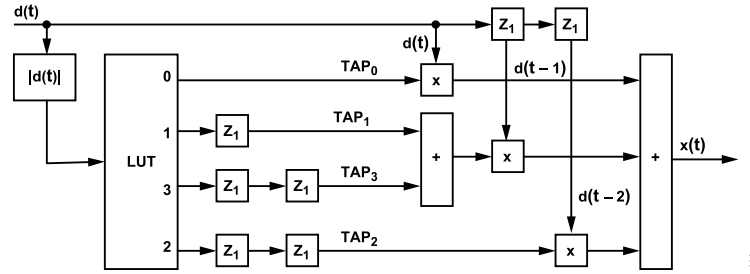


Figure 114. ADRV910x DPD Model 4 LUT configuration

Figure 114 shows that $d(t)$ is the raw complex transmit signal before predistortion. Its amplitude is the basis used by the DPD actuator to predistort the $d(t)$ through its LUT. The LUT consists of four taps calculated with precomputed DPD coefficients α , as shown here:

$$TAP_0 = a_{0,0,0} + a_{0,0,1}|d(t)| + a_{0,0,2}|d(t)|^2 + a_{0,0,3}|d(t)|^3 + a_{0,0,4}|d(t)|^4 \quad (19)$$

$$TAP_1 = a_{1,1,0} + a_{1,1,1}|d(t-1)| + a_{1,1,2}|d(t-1)|^2 + a_{1,1,3}|d(t-1)|^3 + a_{1,1,4}|d(t-1)|^4 + a_{1,1,5}|d(t-1)|^5 + a_{1,1,6}|d(t-1)|^6 \quad (20)$$

$$TAP_2 = a_{2,2,0} + a_{2,2,1}|d(t-2)| + a_{2,2,2}|d(t-2)|^2 + a_{2,2,3}|d(t-2)|^3 + a_{2,2,4}|d(t-2)|^4 \quad (21)$$

$$TAP_3 = a_{3,2,1}|d(t-2)| + a_{3,2,2}|d(t-2)|^2 + a_{3,2,3}|d(t-2)|^3 + a_{3,2,4}|d(t-2)|^4 \quad (22)$$

Note: The TAPx equations represent the default power term setting for each tap in the model 4, from which, derive $b_{t,l_t,i}$ as the following depending on if a power term is included or excluded:

Tap 0: $b_{0,0,0} = 1, b_{0,0,1} = 1, b_{0,0,2} = 1, b_{0,0,3} = 1, b_{0,0,4} = 1, b_{0,0,5} = 0, b_{0,0,6} = 0, b_{0,0,7} = 0$

Tap 1: $b_{1,1,0} = 1, b_{1,1,1} = 1, b_{1,1,2} = 1, b_{1,1,3} = 1, b_{1,1,4} = 1, b_{1,1,5} = 1, b_{1,1,6} = 1, b_{1,1,7} = 0$

Tap 2: $b_{2,2,0} = 1, b_{2,2,1} = 1, b_{2,2,2} = 1, b_{2,2,3} = 1, b_{2,2,4} = 1, b_{2,2,5} = 0, b_{2,2,6} = 0, b_{2,2,7} = 0$

Tap 3: $b_{3,2,0} = 0, b_{3,2,1} = 1, b_{3,2,2} = 1, b_{3,2,3} = 1, b_{3,2,4} = 1, b_{3,2,5} = 0, b_{3,2,6} = 0, b_{3,2,7} = 0$

If using an array B for 4 taps, and for each tap using a byte to represent the above setting (the least significant bit represents the 0 power term), it is clear that the default setting is equivalent to $B[0] = 0x1F$, $B[1] = 0x7F$, $B[2] = 0x1F$ and $B[3] = 0x1E$.

DIGITAL PREDISTORTION (DPD)

Combine the connections from the four outputs to produce the final output, $x(t)$, as the following:

$$x(t) = TAP_0[|d(t)|] \times d(t) + \{TAP_1[|d(t-1)|] + TAP_3[|d(t-2)|]\} \times d(t-1) + TAP_2[|d(t-2)|] \times d(t-2) \quad (23)$$

changeModelTapOrders

This flag selects the default model tap orders or chooses customized model tap orders. If this flag is "TRUE," use "modelOrdersForEachTap" to set the model tap orders for the TX channel. If it is "FALSE," then "modelOrdersForEachTap" is ignored and it uses the default tap orders as discussed ($B[0] = 0x1F$, $B[1] = 0x7F$, $B[2] = 0x1F$ and $B[3] = 0x1E$).

modelOrdersForEachTap

This is an array of bitmaps

$b_{t,l_t,i}$

($i = 0$ to 7 for each tap t ($t=0$ to 3), formulated in the same way as discussed for the default setting. It provides option to customize the order so that a power term can be included or excluded in the polynomial to model the PA. Table 40 shows recommendations to set this field. The [DPD Test on Evaluation Board](#) section discusses the method to select the best model tap orders as a part of DPD tuning recommendations.

Table 40. Suggested Model Orders for Narrowband Waveforms

Taps	Model Orders for Each Tap
Tap 1	$B[1] = 0x1F, 0x3F, 0x7F, 0xFF$
Tap 0 and Tap 2	$B[0] = B[2] = 0x03, 0x07, 0x0F, 0x1F$, (Tap 0 and Tap 2 should be the same.)
Tap 3	$B[3] = 0x0, 0x02, 0x06, 0x0E, 0x1E, 0x3E$

Figure 115 shows the configurable "Customize Model Tap Orders" option in ADRV910x TES, "changeModelTapOrders" is set to "True" when "Customize Model Tap Orders" is checked, then the "modelOrdersForEachTap" can be customized. For simplicity, the GUI uses X to represent $|d(n - l_t)|$.

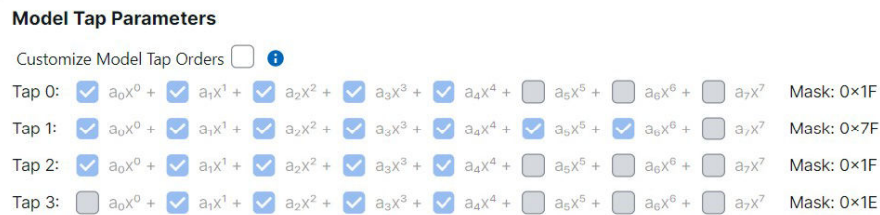


Figure 115. Configuring Customize Model Tap Order Through TES

preLutScale

This value, given as a fixed point U2.2 number, sets the scaling factor before searching the LUT. Set the scaling factor as 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75, 3, 3.25, 3.5, and 3.75. If the signal input to the compander is too small, then use only part of the LUT, scale the input signal magnitude to cover close to the full range of the LUT for better DPD performance. Therefore, users might try different scaling factors to increase the signal level when the input signal is small, which might improve the DPD performance. The scaling factor can be determined according to the dBFS of the input data peak. As an example, if the signal peak power is less than -4 dBFS, apply the scaling factor 3.5.

DPD/CLGC Post-Initial Parameters Configuration

The Post-Initial DPD parameters are defined in "adi_adrv910x_DpdCfg_t" data structure. Table 41 briefly summarizes all the DPD/CLGC post initial calibration parameters described in the data structure.

Table 41. DPD/CLGC Post Initial Parameters

Parameter	Type	Description	Min	Max	Default	Note
numberOfSamples	uint32_t	Specifies the number of samples to use for each iteration of DPD computation.	3	4096	4096	The maximum value is preferred

DIGITAL PREDISTORTION (DPD)**Table 41. DPD/CLGC Post Initial Parameters (Continued)**

Parameter	Type	Description	Min	Max	Default	Note
additionalPowerScale	uint32_t	Provides an estimate of the standard deviation of the modem input data magnitude to scale the data for internal DPD computation.	0	2^{32-1}	4	
rxTxNormalizationLowerThreshold	uint32_t (U2.30)	Signal power for the lower threshold for the normalization of the magnitude and phase of the RX and TX data	0	1.0	0.0031622776602 (-25 dBFS)	
rxTxNormalizationUpperThreshold	uint32_t (U2.30)	Signal power for the upper threshold for the normalization of the magnitude and phase of the RX and TX data	0	1.0	0.031622776602 (-15 dBFS)	
detectionPowerThreshold	uint32_t (U1.31)	Power threshold used for invalid capture detection.				
detectionPeakThreshold	uint32_t (U1.31)	Peak threshold used for invalid capture detection.				
countsLessThanPowerThreshold	uint16_t	If the number of samples below the detectionPowerThreshold exceeds this number, the capture is discarded.				To disable the detection, set it to 4096
countsGreaterThanPeakThreshold	uint16_t	If the number of samples above the detectionPeakThreshold is less than this number, the capture is discarded.				To disable the detection, set it to 0
immediateLutSwitching	bool	Determines whether the LUT switches immediately or at the end of Tx data frame.			TRUE	
useSpecialFrame	bool	DPD only runs on a user-indicated special frame.			FALSE	Currently not supported
resetLuts	bool	Reset LUTs so that no predistortion is applied.			FALSE	Reset LUTs at the start of DPD operation.
timeFilterCoefficient	uint32_t (U1.31)	Coefficient of a time filter to remove spectral spikes from LUT switching.	0	1.0	0	
dpdSamplingRate_Hz	uint32_t	Sampling rate in Hz for the DPD actuator and capture				Read only. No effect on DPD configuration.
clgcLoopOpen	uint8_t	If true, the loop is open and the TX attenuators are not updated			0	
clgcGainTarget_HundredthdB	int32_t	Set as the gain target				
clgcFilterAlpha	uint32_t	Gain filter coefficient.	0	1	0.75	When it is 0, the filter is disabled.
clgcLastGain_HundredthdB	int32_t	Unfiltered gain measured when loop is open.				Only valid for user to retrieve
clgcFilteredGain_HundredthdB	int32_t	Filtered gain measured when loop is open.				Only valid for user to retrieve
captureDelay_us	uint32_t	Amount of time that DPD capture is delayed (beyond normal) relative to the start of the frame.	0		0	
enableRepeatedEstimationInTDD	bool	Reserved				

The following sections describe each parameter.

numberOfSamples

It specifies the number of samples used per DPD/CLGC data capture, with a limit of 4096. The DPD/CLGC performance can be improved if more samples are used. Currently, the linearization channel (LCH) is not supported for TETRA 1. So, set the numberOfSamples to 4096. When LCH is supported, change the numberOfSamples to a different value.

DIGITAL PREDISTORTION (DPD)

additionalPowerScale

This parameter scales the higher power terms when calculating the auto-correlation matrix using transmit data $d(n)$. It keeps the nominal magnitude of each of the power terms about the same to avoid ill condition of the correlation matrix. The scaling factor α can be defined as $\alpha \approx 2 \times std(d(n))$, where “std” stands for standard deviation. Measure α and then pass the information through this parameter.

rxTxNormalizationLowerThreshold/rxTxNormalizationUpperThreshold

These are required parameters to normalize the magnitude and phase of the receive and transmit data. These thresholds are used to pick a linear region to normalize the data. The chosen region must be below the compression point but above the noise. In the AM-AM and AM-PM plot shown in [Figure 120](#), a possible choice of the linear region is highlighted in red. In general, set the rxTxNormalizationUpperThreshold to 0.5 of the peak signal amplitude, and set the rxTxNormalizationLowerThreshold to 0.3 of the peak signal amplitude. Once set, fix the threshold values and these should not vary from capture to capture. Therefore, by knowing the peak transmit signal in dBFS, set the rxTxNormalizationUpperThreshold to “peak Tx dBFS – 6 dB”, and set the rxTxNormalizationLowerThreshold to “peak Tx dBFS – 10.5 dB”. The peak transmit signal is usually set at P1 dB by adjusting the transmitter attenuation setting.

Enter these thresholds in dBFS through TES. If using API, use the linear numbers, which can be calculated as $10^{(\text{threshold_dBFS}/10)}$.

detectionPowerThreshold / countsLessThanPowerThreshold

Data capture for the DPD/CLGC operation is invalid if a specified number of samples (countsLessThanPowerThreshold) are below the defined power threshold (detectionPowerThreshold), the data capture will be discarded. Set countsLessThanPowerThreshold to 4096 will disable the invalid data capture detection.

DetectionPeakThreshold / CountsGreaterThanPeakThreshold

Data capture for the DPD/CLGC operation is valid if a specified number of samples (countsGreaterThanPeakThreshold) is greater than the defined peak threshold (detectionPeakThreshold). Set countsGreaterThanPeakThreshold to 0 will disable the valid data capture detection.

immediateLutSwitching

When a new DPD solution is formed, load the new solution into a spare LUT, which can then be switched with the active LUT. There are two options for LUT switching. When immediateLutSwitching is set to “TRUE”, the new LUT immediately, when ready, swaps out the active LUT. When immediateLutSwitching is set to “FALSE”, after the updated LUT is available, LUT swapping is triggered after the next data frame is completed. Note: This only applies to TDD operations. FDD systems should always use immediate LUT switching.

useSpecialFrame

To achieve optimal performance, DPD must capture the peaks of the signal. Some standards allow the transmission of special data used for DPD estimation. In other cases, the baseband processor can know in advance that a frame contains good data for DPD estimation. In these cases, the user can control which frames are used for DPD. The flag indicates the control of the frames that the DPD can capture. This is currently not supported.

resetLuts

To start the DPD operation from a known state, reset the LUTs. By setting resetLuts to 1, it sets most polynomial terms to 0 to remove the predistortion at the beginning of the DPD operation.

timeFilterCoefficient

This parameter defines the coefficient of a single-pole filter, which mitigates the spectral spikes caused by LUT switching, especially in the FDD and narrowband use cases. Avoid spectral spikes in TDD by not performing an immediate LUT switch. The range of “timeFilterCoefficient” is between 0 and 1. If setting it to 0, it is equivalent to disabling this functionality. Experiment with this parameter and pick the optimal value to reduce the spikes. Note that enabling this feature might hurt the convergence time. The bigger this parameter is, the slower the convergence might be. Therefore, consider this method as an available tool when all other methods are exhausted and employ it with caution.

dpdSamplingRate_Hz

This parameter shows the sampling rate in Hz for the DPD actuator and capture. It is read only and does not have any effect on DPD configurations.

DIGITAL PREDISTORTION (DPD)

clgcLoopOpen

This parameter opens or closes the gain control loop. When it is set to 1, the transmit attenuators are not updated. This measures the gain to determine the target gain. Once the target gain is set, set it to 0 to close the loop and start the CLGC operation.

clgcGainTarget_HundredthdB

Configure this parameter to set the gain target for CLGC with an accuracy of a hundredth of a dB.

clgcFilterAlpha

This parameter stands for the coefficient of a single-pole filter to smooth the gain measurement. The minimum value is 0, which is equivalent to disable this filter by using the instantaneous gain measurement result. The maximum value is 1, and the default value is 0.75. Set the bigger value to achieve smoother gain measurement results.

clgcLastGain_HundredthdB

This parameter represents the unfiltered gain measurement results when the loop is open. Use API to retrieve this value, which is in an accuracy of a hundredth of a dB. Note: This parameter is only valid for retrieving.

clgcFilteredGain_HundredthdB

This parameter represents the filtered gain measurement results when the loop opens (based on the user-configured filter coefficient). Use API to retrieve this value, which is in an accuracy of a hundredth of a dB. Note: This parameter is only valid for retrieving.

captureDelay_us

This parameter delays the DPD/CLGC capture from the beginning of the transmitter frame by an amount of time (in μ s) specified. By default, it is set to 0. However, it can help avoid using transmitter data at the beginning of a transmitter frame, such as preamble or control signal, which has a characteristic not suitable for DPD operation. Note: This parameter applies to both DPD and CLGC.

Configure the DPD/CLGC post initial calibration parameters through TES, as shown in Figure 221. Note: All data capture related configurations such as "Number of Samples," "Capture Delay (us)," "Rx/Tx Normalization", and "Activation" are also used by the CLGC algorithm.

External Loopback Connection

ExtLoopBackType

As shown in [Figure 113](#), the external loopback connection after PA is needed for DPD, in addition to the hardware connection, the external loopback type "After PA" should be set in TES before initializing the ADRV910x devices if DPD is enabled.

peakLoopBackPower

The external loopback signal peak power at ADRV910x observation RX (ORx) input port should be set at about -18dBm with a tolerance of $\pm$ 5dBm, adjust the loopback signal peak power with external attenuator before sending it to ORx to make sure the peak power meets the DPD requirements.

externalLoopbackPathDelay

DPD/CLGC requires the alignment of the transmit signal $x(t)$ with the external loopback capture signal $y(t)$. The externalLoopbackPathDelay parameter compensates for additional delays on the external loopback path from the ADRV910x transmit output to the ORx input. Measure this delay and provide it to the ADRV910x before initial calibration. Then DPD engine uses the measured delay to compensate for the delay between $x(t)$ and $y(t)$. This parameter is critical, especially for wideband applications, due to the high sample rate. In narrowband applications, it is less critical. So, set it to 0 unless there is a larger delay in the external loopback path.

The external loopback path delay can be measured by running the "adi_adrv910x_cals_ExternalPathDelay_Calibrate" after ADRV910x is initialized and calibrated (Radio State is at "Calibrated" state), User should make sure the external loopback hardware connection is available and the PA is powered on when running the external loopback delay calibration.

DIGITAL PREDISTORTION (DPD)

DPD/CLGC Configuration in TES

ADRV910X TES provides the configuration fields to set the DPD/CLGC parameters, as shown in [Figure 116](#), users could open the DPD/CLGC configuration page by clicking the **drop-down menu** in the TES upper left, then clicking on **“Use Case and Initialization”**, then **“Digital Pre-Distortion”** and choosing **“TX Wideband DPD”** or **“Tx Narrowband DPD”** based on the relative profile. The detailed DPD/CLGC configuration fields are shown in [Figure 117](#) (TX WB DPD as example).

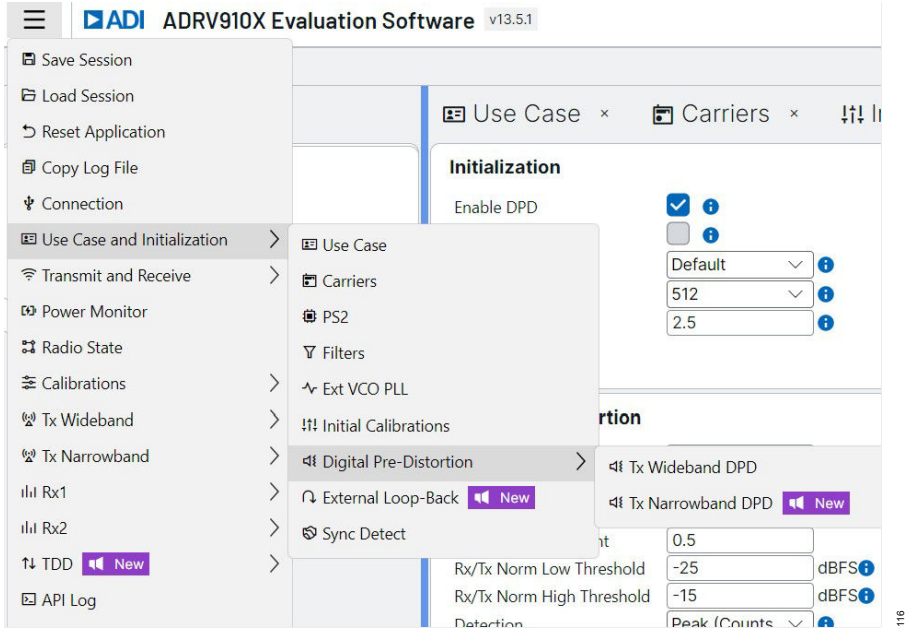


Figure 116. Open DPD/CLGC Configuration Page in TES

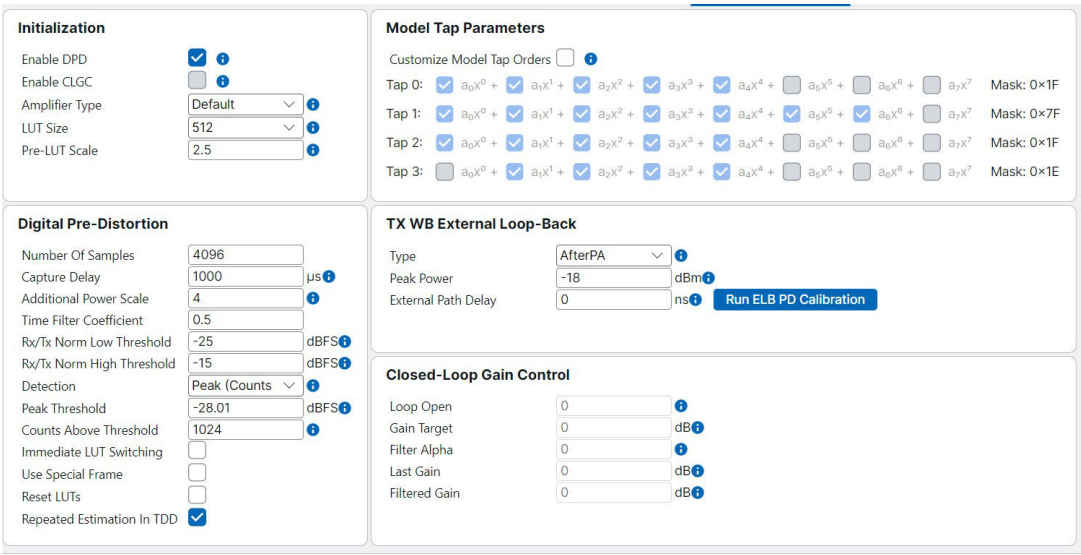


Figure 117. Configuring DPD/CLGC Parameters Through TES

DIGITAL PREDISTORTION (DPD)

DPD/CLGC UPDATES SCHEDULING

DPD/CLGC is one of the tracking calibrations that will run periodically at the start of a frame in TDD profiles. DPD will be scheduled to run approximately every 200 ms for the first four iterations and every 2 sec thereafter.

DPD TUNING AND TESTING

Tuning the Model Tap Order

DPD can be considered an adaptive filter modeled according to the behaviour of the PA. The ADRV910x default model (model 4) consists of four taps. Each tap consists of a series of polynomial terms to fit the nonlinear behaviour due to compression at higher output power. The order of polynomial terms is determined by intermodulation falling closer to the carrier spectrum. In DPD, the orders of intermodulation to consider are usually the third, fifth, and seventh orders, with decreasing magnitude, respectively. An n th-order intermodulation expands the signal bandwidth 'n' times. By inspecting the bandwidth expansion factor on a spectrum analyzer, estimate how many orders of intermodulation to include in the polynomial terms, in order to suppress the spectral regrowth down to the required ACPR. It is important not to include higher order power terms than needed, which might cause the DPD to be unstable.

ADRV910x DPD model 4 consists of four taps, as shown in the example tap arrangement diagram in Figure 118. The four taps can be classified into three categories.

The main tap, TAP1 is the main tap of the DPD adaptive filter that suppresses most of the spectral regrowth due to intermodulation, it has the greatest number of polynomial terms.

The side taps, TAP0 and TAP2 are the two side taps on each side of the main taps. They are memory terms that compensate for frequency-dependent distortion in the frequency domain and time misalignment between the transmit and receive captured data. The side taps have the same number of polynomial terms, and each side tap has about half of the number of polynomial terms of the main tap.

The cross-term tap, TAP3 is the cross-term that further suppresses the residual spectral regrowth left over by the other three taps. The number of polynomial terms is usually equal to or less than that of each side tap.

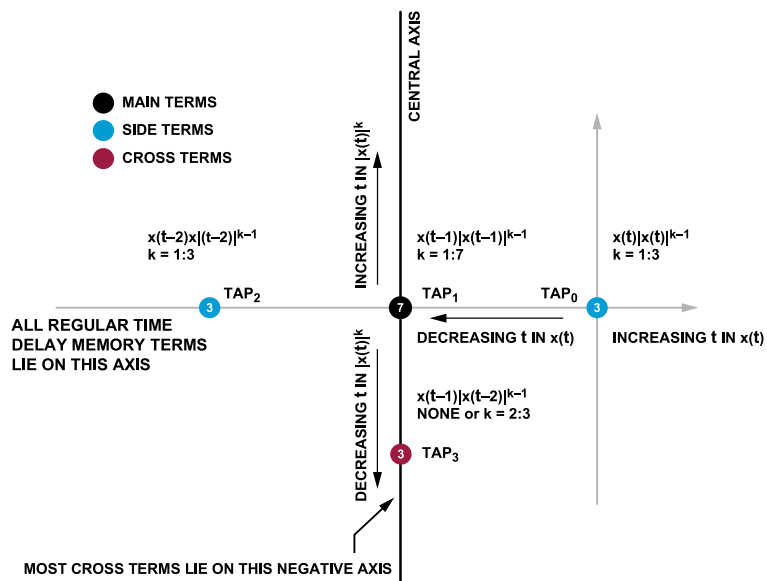


Figure 118. Example Polynomial Constellation Configuration for DPD Model 4

k represents the order in Figure 118. To handle the seventh harmonics, the main tap must include the power terms up to $k = 7$. To find the best model tap order for a specific PA design, use the following recommended procedures:

1. Set the amplifier output power to have a compression ratio of 1 dB or slightly less, as shown in Figure 111, i.e., the maximum peak of the output signal is 1 dB below ideal linearity.
2. Determine the initial highest polynomial order of the main tap, TAP1, by measuring the spectral regrowth bandwidth to carrier bandwidth ratio. Include all lower-order polynomial terms. For example, if the bandwidth ratio is 5, set the initial highest polynomial to $k = 5$.

DIGITAL PREDISTORTION (DPD)

3. Set the other taps to zeros, i.e., turn off the other taps.
4. Use only TAP1 to execute DPD with order 5, then 6, and 7 (i.e., up to two orders above the initial estimate). Measure the ACPR for each case. Select the one that yields a better ACPR. If the difference is small, select a lower-order one, say 5.
5. While keeping the main tap determined above, set the side taps, TAP0 and TAP2, to about half the order of the main tap. In the above example, the main tap has an order of 5. Select the initial side tap order as 2. Execute DPD with the main tap of order 5 and the side tap order of 2, then 3, and 4. Select the side tap order that yields the lowest ACPR, say 3.
6. While keeping the main tap and side tap orders determined above, set the initial cross-term tap order, TAP 3, to be 2. Execute DPD with the cross-term tap order of 2 and 3. Select none, 2, and 3, which yields the best ACPR. If the difference is small, select the lower order one, including "none" taps. Some power amplifiers do not need a cross-term.
7. Iterate the above procedure as necessary with different combinations until confident with the selections. Keep all tap order selections to be minimal, that satisfies the ACPR requirement with a 5 dB margin, which helps to keep DPD more stable. For example, if the ACPR requirement is -60 dB, set the ACPR target as -65 dB.

DPD Test on Evaluation Board

Figure 119 describes an example setup to test the integrated DPD with the ADRV910X evaluation board in narrowband applications (I.E, TETRA I), where the TX/RX2 profile is configured, external loopback uses RX1 port as observation Rx input port, refer Figure 79 for the ORX connection. Figure 119 shows that an LPF is required at the TX output port to filter out the transmitter harmonics before feeding the signal to the PA driver (If the PA driver has an internal LPF, then the external LPF is not needed). As the device uses a square-wave mixer, it produces strong odd-order harmonics. Without filtering those harmonics, the DPD performance can be impacted. The step attenuators external to the ADRV910X evaluation board are optional. Note: It is important to set up the external loopback path before operating the integrated DPD. The optional external LO/VCO can be used if the internal LO phase noise is not good enough.

For wideband applications (TX/RX1 profile), the setup is similar, but the external loopback path might use RX2 port as observation input port, refer Figure 80 for the ORX connection.

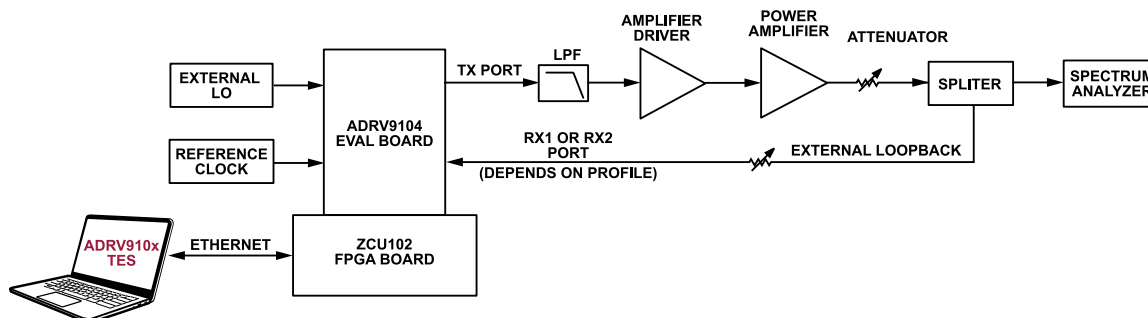


Figure 119. An Example Setup for Testing the Integrated DPD in Narrowband Applications

To test the DPD on ADRV9104 evaluation board, connect the hardware as Figure 119 shown. Once the setup is ready, further configure the TES and available external components properly, which includes the following major steps:

1. Select the desired profile, i.e. TX/RX2 narrow band profile or TX/RX1 wide band profile. Configure the relative profile parameters.
2. Configure the DPD parameters as shown in Figure 116, Figure 117
3. Measure and Set the peak power of the loopback signal (ideally, adjust it to -18 dBm $\pm$ 5 dB. Do this by tuning the external step attenuator)
4. Measure the external loopback delay and set the value to ADRV910x before running DPD, refer Measuring the External Path Delay.
5. Program the device, load, and play transmit waveform file.
6. Properly tune the transmitter attenuation and/or the step attenuator to make sure that the ACPR performance at the device transmitter output is satisfactory before passing to the PA. In addition, make sure that the transmit peak signal is around the P1 dB compression region.
7. Compare the ACPR performance through a spectrum analyzer with and without using the integrated DPD.

Measuring the External Path Delay

DPD is an adaptive algorithm that requires taking the error of the transmitter and observation receiver. Before taking the error of the transmitter and the receiver, the two signals need to be properly aligned—especially if a high sample rate profile is used (for example, LTE10). The alignment is critical because the intervals between samples are small. Therefore, users need to measure the external path delay and set the value to ADRV910x before running the DPD.

DIGITAL PREDISTORTION (DPD)

After programming the chip (channel radio state is "CALIBRATED"), power up the power amplifier and its pre-driver, call API `adi_adrv910x_cals_ExternalPathDelay_Calibrate` to measure external path delay, then call API `adi_adrv910x_cals_ExternalPathDelay_Get` to get the measurement result and set the value by API `adi_adrv910x_cals_ExternalPathDelay_Set`.

The external path delay won't change for a given profile and hardware, therefore the delay measurement ideally only needs to be done once, and the value can be set during the chip initialization stage.

DPD Monitoring

ADRV910x DPD Performance

Figure 120 and Figure 121 shows the AM-AM and AM-PM performance of the raw transmit signal input versus the PA output before and after DPD. The test waveform is TETRA1. From Figure 121, it can be seen clearly that the PA non-linearity is successfully corrected by the integrated DPD.

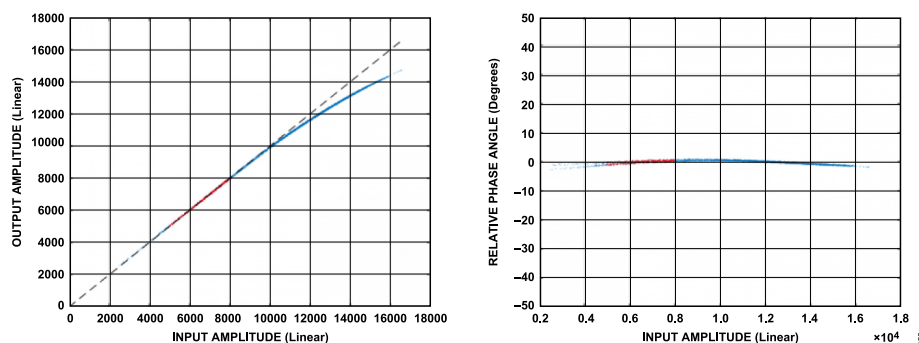


Figure 120. Raw Transmit Signal Input versus Non-Linearized PA Output

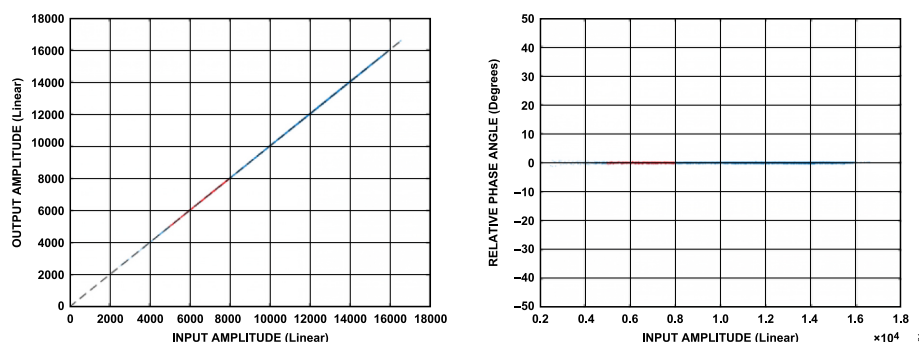


Figure 121. Raw Transmit Signal Input versus Linearized PA Output

Figure 122 shows the DPD test results at narrow band (TX/RX2) mode with TETRA I waveform, where the blue plot is the PA output ACPR before DPD and the yellow plot is the PA ACPR after DPD.

Figure 123 and Figure 124 are the PA output ACPR results before and after DPD at wide band (TX/RX1) mode with a LTE20 waveform.

Note: DPD test is with a SKY65160-11 power amplifier which has ~33.8dBm P1dB (actual test number). ADRV910x is set at internal LO with 38.4MHz reference clock.

Note: TETRA I test waveform PAR is ~3dB, LTE20 test waveform PAR is ~9.3dB.

DIGITAL PREDISTORTION (DPD)

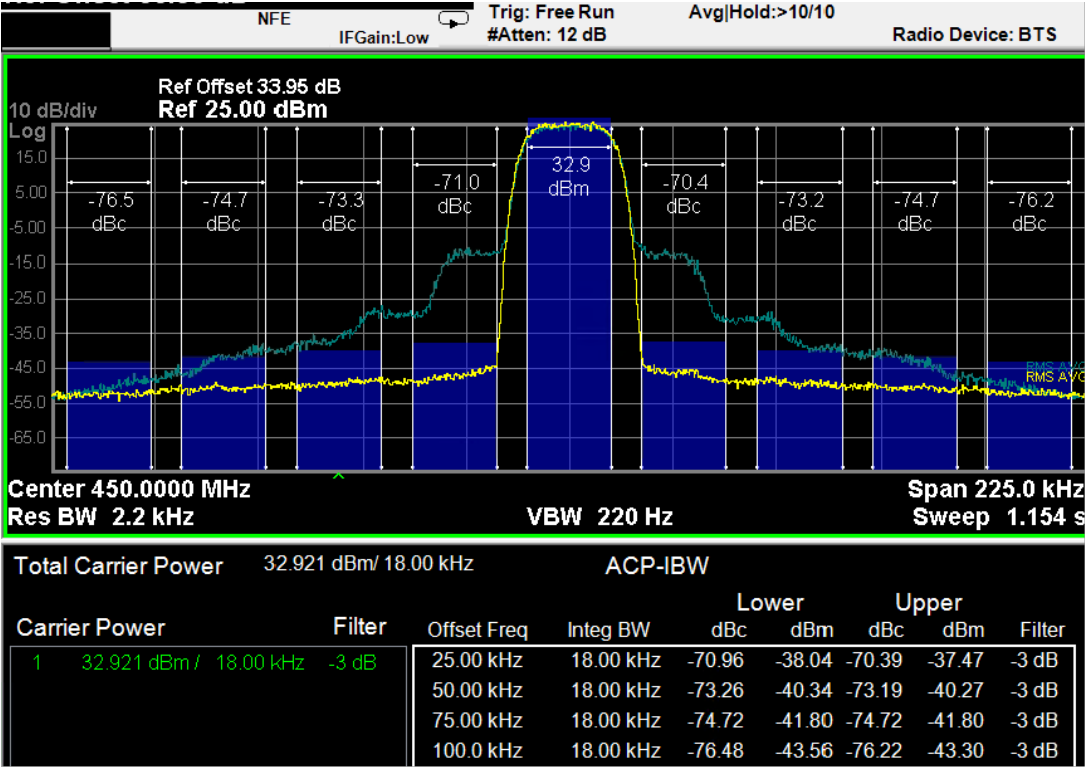


Figure 122. Narrow band (Tetra I) DPD test result : PA Output ACPR before and after DPD

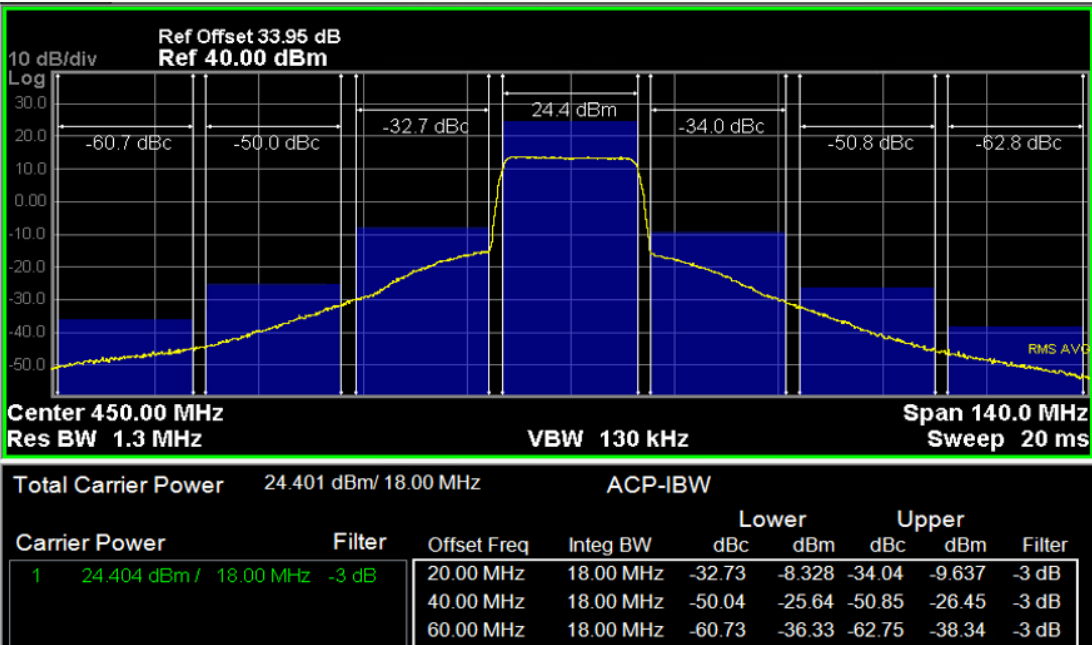


Figure 123. Wide band (LTE20) DPD test result : PA Output ACPR before DPD

DIGITAL PREDISTORTION (DPD)

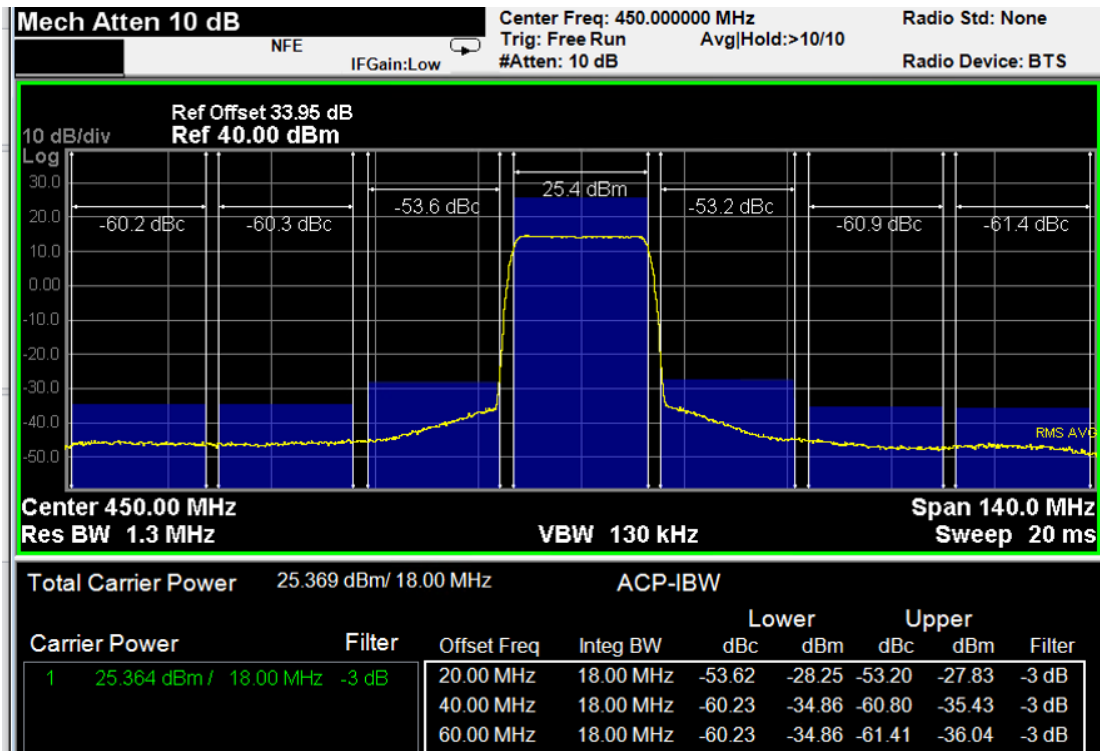


Figure 124. Wide band (LTE20) DPD test result : PA Output ACPR after DPD

RF PORT INTERFACE

INTRODUCTION

The RF ports of the ADRV910X, consisting of the transmit (TX $\pm$) and receive ports (RX1 $\pm$, RX2 $\pm$), support an operational frequency range from 70 MHz to 6 GHz. This wide frequency range fulfills the requirements of many application spaces. However, for optimized performance within a narrowband with minimal amplitude roll-off and optimized linearity and noise performance, the RF ports are impedance-matched.

This section provides some examples of impedance matching networks to select frequency bands. Locating a balun/transformer to cover the entire frequency range of the ADRV910X with minimal phase and amplitude imbalance is a challenging task given the limited selection of commercially available baluns. So, the example RF matching networks are chosen based on the available baluns/transformers and RF trace implementations on the evaluation PCB.

TRANSMIT PORT: TX $\pm$

The ADRV910X uses a direct conversion transmitter architecture. The differential output impedance of transmitter outputs is matched to 50 Ω as shown in [Figure 125](#). Additionally, bias the transmitter output to a low noise 1.8 V supply.

RECEIVE PORTS: RX1 $\pm$, RX2 $\pm$

The ADRV910X has two RF inputs to allow tracking algorithms to be run in conjunction with receiving data. The mixer architecture is very linear and inherently wideband, which facilitates wideband impedance matching. The differential input impedance of the receiver inputs is 100 Ω , as shown in [Figure 126](#). When selecting a balun/transformer for the receive paths, use a 2:1 impedance transformation to accommodate the 50 Ω single-ended impedance to 100 Ω differential impedance, as required by the ADRV910X receiver inputs. The receiver input pins are self-biased internally to 650mV and, therefore, require AC-coupling/DC-blocking capacitors at their inputs.

EXTERNAL LOCAL OSCILLATOR PORTS: LO1 $\pm$ AND LO2 $\pm$

Apply two external LO/VCO inputs (LO1 and LO2) to the ADRV910X and use each external LO/VCO signal for any of the two receivers or the transmitter instead of an internally generated LO signal. The AC-coupling interface is needed for both the positive and negative sides of the external LO input pins, which are internally biased. Similar to the receiver RF interface, a balun with 2:1 impedance transformation is necessary to accommodate the 50 Ω single-ended impedance to 100 Ω differential impedance, as required by the ADRV910X external LO inputs.

DEVICE CLOCK PORT : DEV_CLK_IN $\pm$

There are two low-frequency (up to 80 MHz) clock interface modes and an LVDS-type clock interface mode that can support the clock signal running as fast as 100 MHz. For the high-frequency clock interface, off-chip 100 Ω resistive termination is required, along with AC-coupling caps. The subsequent section [Connection for Device Clock \(DEV_CLK_IN\)](#) provides more information.

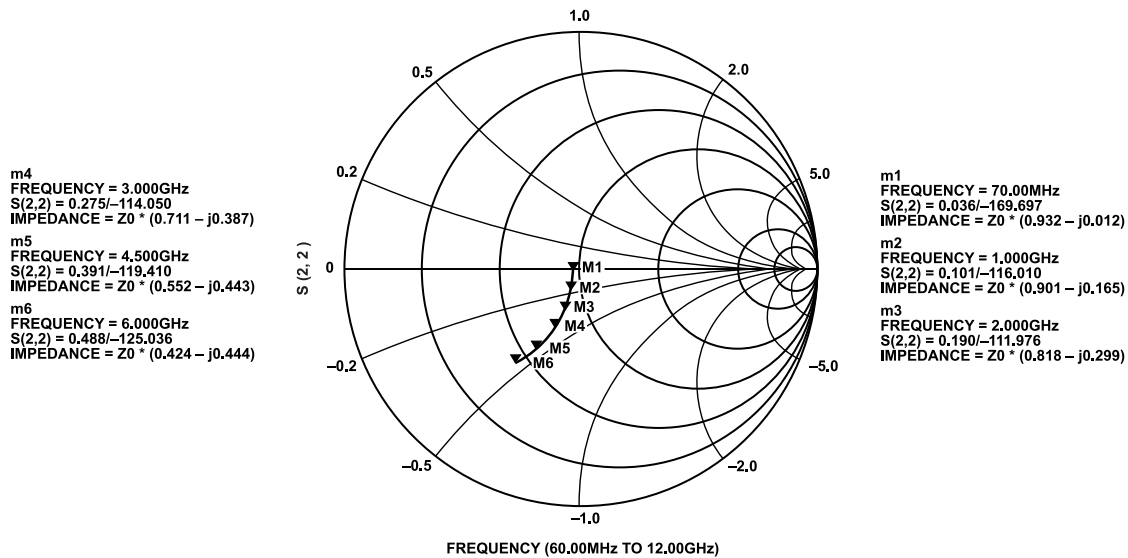
RF RECEIVER/TRANSMITTER PORTS IMPEDANCE DATA

This section provides the port impedance data for all the transmitter and receivers in the ADRV910x integrated transceiver. Note the following:

- ▶ The ADRV910x ball pads are the reference plane for this data.
- ▶ Single-ended mode port impedance data is not available. However, a rough assessment is possible by taking the differential mode port impedance data, and dividing the real and imaginary components by two.

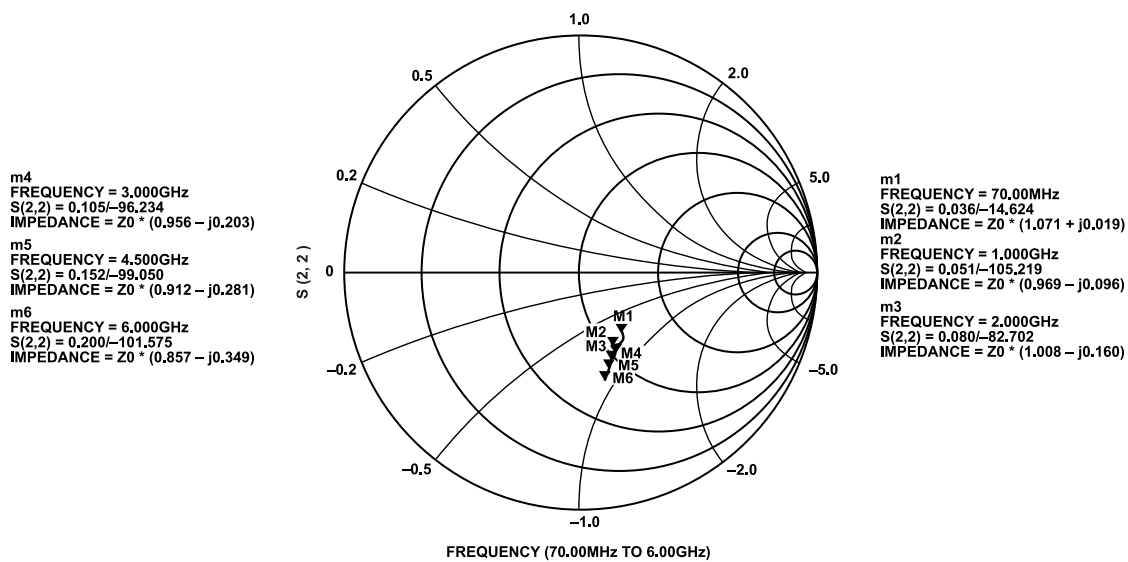
Contact Analog Devices Applications Engineering for the impedance data in the Touchstone format.

RF PORT INTERFACE



125

Figure 125. ADRV910x Transmitter Port Series Equivalent Differential Impedance



126

Figure 126. ADRV910x Receiver Port Series Equivalent Differential Impedance

Advanced Design System (ADS) Setup Using DAC and SEDZ File

ADI supplies the port impedance as an *.s1p Series Equivalent Differential Z(impedance) file. This format allows simple interface to ADS by using the Data Access Component (DAC). In the below diagram Term1 is the single ended input or output and Term2 represents the differential input or output RF port on ADRV910X. The Pi on the single ended side and the differential Pi configuration on the differential side allows maximum flexibility in designing matching circuits, and is suggested for all design layouts as it can step the impedance up or down as needed with appropriate component population.

RF PORT INTERFACE

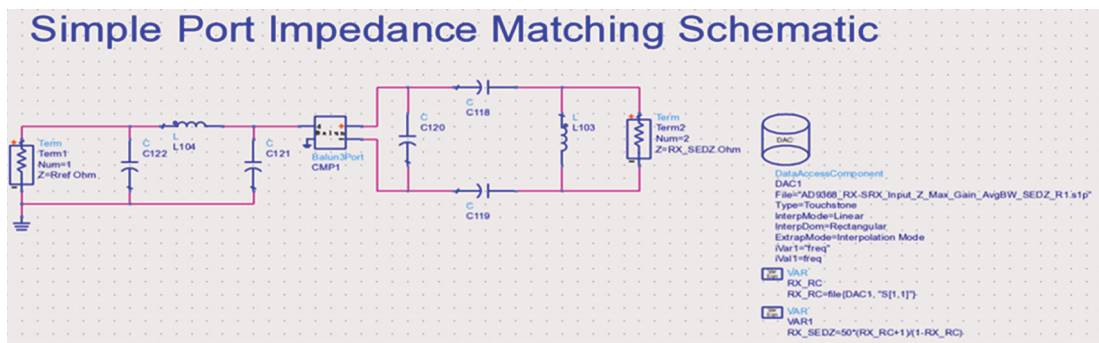


Figure 127. Simulation Setup in ADS with SEDZ S1P Files and DAC Component

Operation is as follows:

1. The DAC block reads the rf port*.s1p file. This is the device RF port reflection coefficient.
2. The two equations convert the RF port reflection coefficient to a complex impedance. The result is the RX_SEDZ variable.
3. The RF port calculated complex impedance (RX_SEDZ) is utilized to define the Term2 impedance. a. Term2 is used in differential mode and Term1 is used in single-ended mode.

Setting up the simulation this way allows one to measure the S11, S22, and S21 of the 3-port system without complex math operations within the display page. Note for highest accuracy, EM modelling result of the PCB artwork and S-parameters of the matching components and balun should be used in the simulations.

The first differential shunt reactive component such as L103 in Figure 127 is inserted to tune out parallel parasitic reactance of input impedance of the device. If ac-coupling cap is necessary, C118 and C119 can be used utilized for this purpose.

For a wide-band match application, because of well controlled input/output impedance characteristics of ADRV910X for entire range of its RX/TX operational frequency band, minimal matching network can be implemented to control undesirable impedance deviation typically associated with the high side of frequency range a balun operates. Additionally, by selecting a balun with the same differential side termination impedance as the impedance of RX inputs and of TX output, a broad-band match can be accomplished by avoiding unnecessary impedance transformation network which is inherently band-limiting.

One can also consider adding additional differential series capacitive component on the balanced side of balun to facilitate ac-coupling and Pi match on both sides of the balun as shown below.

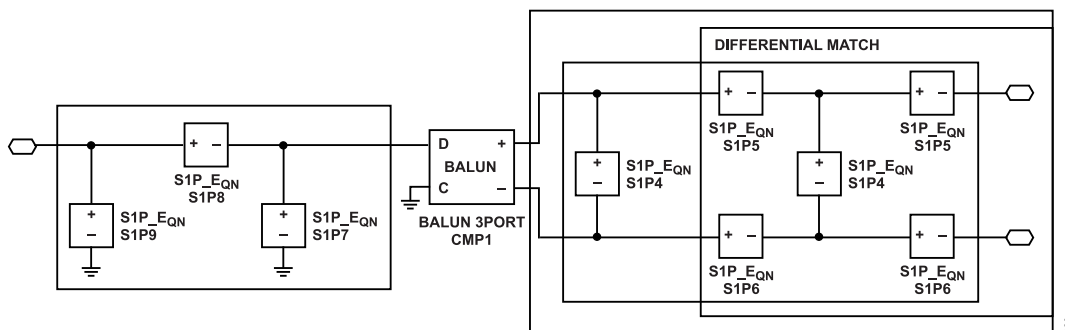


Figure 128. RF Matching Network with Additional Series AC-Coupling Capacitors

For a narrowband impedance match application to filter out signals outside of frequency band of interest, one can use a Pi match technique for desired bandwidth of impedance match with a selected balun's terminal impedance. Pi match can be considered as two L match networks back to back and would allow independent control of Q and impedance ratio obtainable from a matching network. Narrowband matching network tuned for frequency bands of RX and TX can further improve out of band rejection of a transceiver for frequency duplexed systems.

RF PORT INTERFACE

TX INTERFACE

This section considers the dc biasing of the ADRV910x transmitter (Tx) output and how to interface the Tx port. ADRV910x transmitter operates over a range of frequencies. At full output power, each differential output side draws approximately 100mA of DC bias current. The Tx output is DC biased to a 1.8V supply voltage using either RF chokes (wire-wound inductors) or a transformer centre tap connection.

Careful design of the DC bias network is required to ensure optimal RF performance levels. When designing the DC bias network, select components with low DC resistance (R_{DCR}) to minimize the voltage drop across the series parasitic resistance element with either of the suggested dc bias schemes suggested in Figure 129. 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, PO 1dB PO MAX, and so forth) to degrade. The choke inductance (LC) should be selected high enough relative to the load impedance such that it does not degrade the output power.

The recommended DC bias network is shown in Figure 130. This network has fewer parasitic and fewer total components.

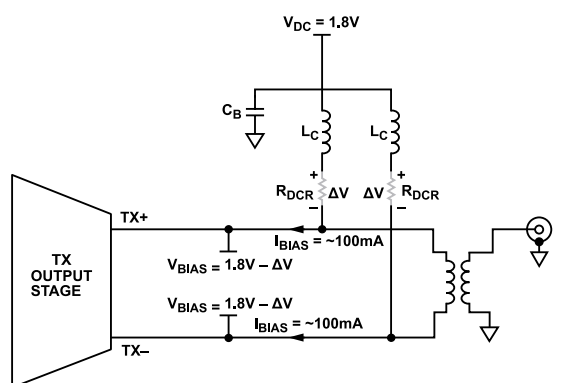


Figure 129. RF DC Bias Configurations Depicting Parasitic Losses Due to Wire Wound Chokes

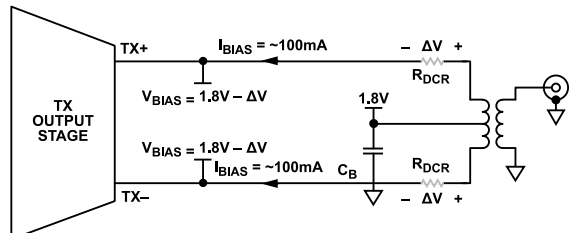


Figure 130. RF DC Bias Configurations Depicting Parasitic Losses Due to Center Tapped Transformers

Figure 131 to Figure 134 identify four basic differential transmitter output configurations. Impedance matching networks (balun single-ended port) are most likely to be required to achieve optimum device performance from ADRV910x. Also, the transmitter outputs must be AC-coupled in most applications due to the DC bias voltage applied to the differential output lines of the transmitter.

The recommended RF transmitter interface is shown in Figure 131 featuring a center tapped balun. This configuration offers the lowest component count of the options presented.

Brief descriptions of the transmitter port interface schemes are provided as follows.

- ▶ Centre tapped transformer passes the bias voltage directly to the transmitter outputs
- ▶ RF chokes are used to bias the differential transmitter output lines. Additional coupling capacitors (CC) are added in the creation of a transmission line balun
- ▶ RF chokes are used to bias the differential transmitter output lines and connect into a transformer
- ▶ RF chokes are used to bias the differential output lines that are ac-coupled into the input of a driver amplifier.

RF PORT INTERFACE

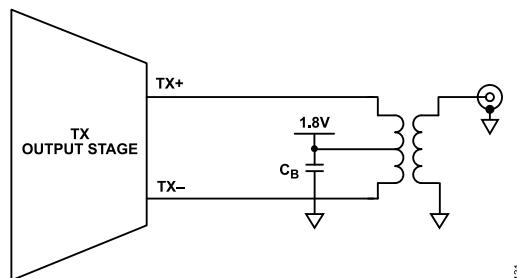


Figure 131. RF Transmitter Interface Configuration A

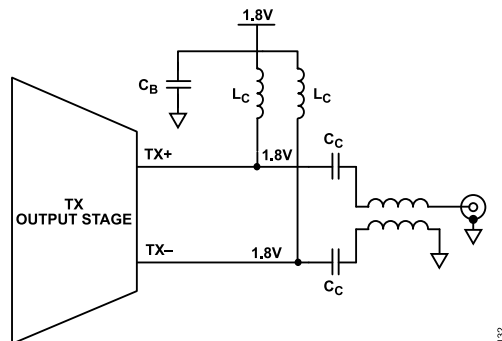


Figure 132. RF Transmitter Interface Configuration B

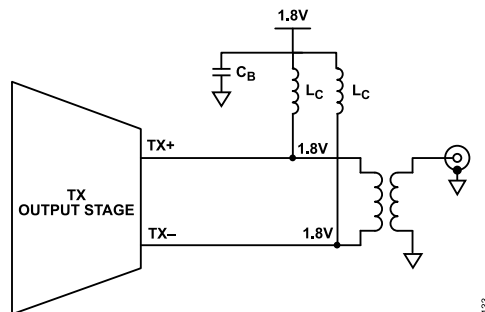


Figure 133. RF Transmitter Interface Configuration C

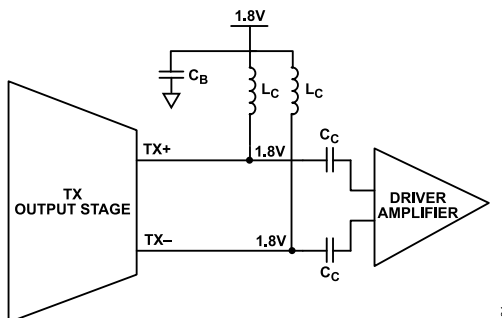


Figure 134. RF Transmitter Interface Configuration D

If a Tx balun is selected that requires a set of external dc bias chokes, careful planning is required. It is necessary to find the optimum compromise between the choke physical size, choke dc resistance (R_{DCR}) and the balun low frequency insertion loss. In commercially available dc bias chokes, resistance decreases as size increases. However, as choke inductance increases, resistance increases. Therefore, it is undesirable to use physically small chokes with high inductance as they exhibit the greatest resistance. For example, the voltage drop of a 500 nH, 0603 choke at 100 mA is roughly 50 mV.

RF PORT INTERFACE

Table 42. Sample Wire-Wound DC Bias Choke Resistance vs. Size vs. Inductance

Inductance (nH)	Resistance (Size: 0603)	Resistance (Size: 1206)
100	0.10	0.08
200	0.15	0.10
300	0.16	0.12
400	0.28	0.14
500	0.45	0.15
600	0.52	0.20

When selecting a DC bias choke inductor, shunting impedance of the choke inductor would need to be high for TX frequency band to minimize its loading to outputs. Therefore, the self-resonant frequency of the selected choke inductor must be higher than intended TX frequency.

RX INTERFACE

ADRV910x has two independent receive input channels (Rx1 and Rx2). The differential input signals would be applied to an integrated mixer. The mixer input pins are internally biased to 0.65 Volt and would need to be AC coupled depending on the common mode voltage level of the external circuit.

Important considerations for the receiver RF port interface are as follows:

1. Device to be interfaced: filter, balun, T/R switch, external LNA, etc. Does this device represent a short to ground at DC?
2. Rx1 and Rx2 maximum safe input power is 18 dBm (peak).
3. Rx1 and Rx2 optimum DC bias voltage is 0.65 V bias to ground.
4. Board Design: reference planes, transmission lines, impedance matching, etc.

Figure 135 shows possible differential receiver port interface circuits. The options in Figure 135 and Figure 136 are valid for both receiver inputs operating in differential mode, though only the Rx1 signal names are indicated. Impedance matching may be necessary to obtain datasheet performance levels.

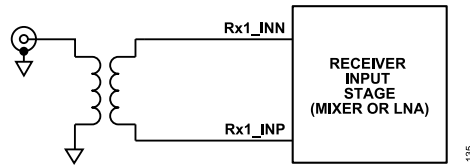


Figure 135. Differential Receiver Interface Using a Transformer

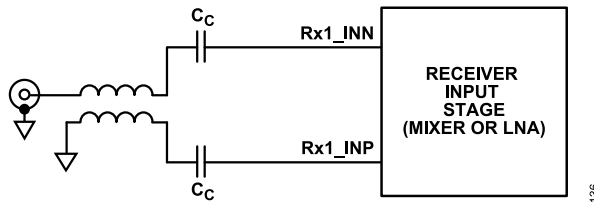


Figure 136. Differential Receiver Interface Using a Transmission Line Balun

Given wide RF bandwidth applications, SMD balun devices function well offering acceptable differential balance and insertion loss in a relatively small (0603, 0805) package.

EXT LO INPUTS

Unlike the internal synthesizers that always operate from 6 GHz to 12 GHz regardless of the RF tune frequency, when external LO pins are used the frequency applied must be a multiple of 2 (that is, 2x, 4x, 8x, and so on) of the desired RF signal channel frequency, 1x external LO is supported for 500Mhz-1GHz carrier frequency. The LO input signal is internally divided by a series of dividers to generate the required LO quadrature relationship at desired RF frequency of upconversion or downconversion for internal transmitter and receivers. Table 43 describes specification for EXT LO input pins when input pins are driven by a differential signal by use of balun.

RF PORT INTERFACE

Alternatively single-ended external LO signal source can be used for positive side of EXT LO pin with negative side of input pin terminated with a capacitor to provide ac ground. The frequency of external LO source should be set to 4x of desired RX or TX frequency with EXT LO divider configured to division of 4 for the best in-phase and quadrature generation of LO necessary for internal receivers and transmitters. [Table 44](#) describes specification for single-ended EXT LO input source.

In general, higher power level of external LO signal applied gives better phase noise to some extent. The minimum input power level that satisfies RX/TX phase noise requirements with some margin should be used. Refer to [Table 43](#) and [Table 44](#) for power level recommendation.

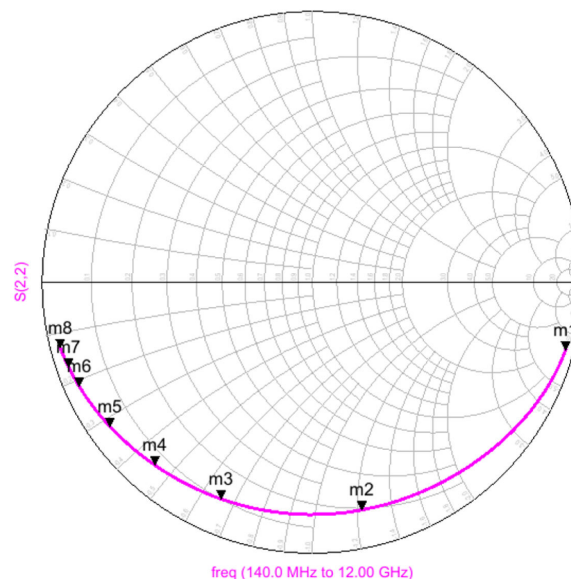
Table 43. Specifications for ADRV910X RF EXT LO Differential Input Pins

Parameter	Note	Min	Typical	Max	Unit
External LO Frequency	FEXTLO	140		12000	MHz
RF Channel Frequency	FCHANNEL	70		6000	MHz
External LO Power	100 Ω matching; signal amplitude depends on FEXTLO frequency; typical = 0dBm for FEXTLO $\leq$ 2GHz; from 2 GHz, add 0.5dB/GHz; or example, typical = +3dBm for FEXTLO = 8GHz	-6	0		dBm
Input Impedance	Nominal, small signal input. Note below.		100		Ω
Differential Phase Error	Combined differential phase error, differential amplitude error, duty cycle error, and even order harmonic content			20	Degrees
Differential Amplitude Error	Combined differential phase error, differential amplitude error, duty cycle error, and even order harmonic content			1	dB
Duty Cycle Error	Combined differential phase error, differential amplitude error, duty cycle error, and even order harmonic content			2.5	%
Even Order Harmonic Content	Combined differential phase error, differential amplitude error, duty cycle error, and even order harmonic content			-30	dBc
EXT LO Source Modulus	Must match internal modulus on ADRV910x	8386560			

Table 44. Specifications for ADRV910X RF EXT LO Single-Ended Input Pin

Parameter	Note	Min	Typical	Max	Unit
External LO frequency	FEXTLO	140		2000	MHz
RF Channel frequency	FCHANNEL	70		1000	MHz
External LO power	50 Ω matching	0	+3	+6	dBm
Input Impedance	nominal, small signal input. Note below.		50		Ω
EXT LO Source Modulus	Must match internal modulus on ADRV910X	8386560			

m5	freq=4.500 GHz S(2,2)=0.919 / -144.653 impedance = Z0 * (0.046 - j0.318)
m6	freq=7.000 GHz S(2,2)=0.944 / -156.022 impedance = Z0 * (0.030 - j0.212)
m7	freq=9.000 GHz S(2,2)=0.955 / -160.969 impedance = Z0 * (0.024 - j0.168)
m8	freq=12.00 GHz S(2,2)=0.964 / -165.535 impedance = Z0 * (0.019 - j0.127)



m1	freq=140.0 MHz S(2,2)=0.970 / -14.915 impedance = Z0 * (0.881 - j7.538)
m2	freq=1.000 GHz S(2,2)=0.860 / -77.761 impedance = Z0 * (0.190 - j1.222)
m3	freq=2.000 GHz S(2,2)=0.868 / -112.938 impedance = Z0 * (0.102 - j0.658)
m4	freq=3.000 GHz S(2,2)=0.891 / -130.910 impedance = Z0 * (0.069 - j0.455)

Figure 137. External LO Series Equivalent Differential Input Impedance

RF PORT INTERFACE

Care should be taken when selecting an on-board balun for this application. Combination of amplitude and phase balance performance of the balun can affect quadrature error performance. Additionally, duty cycle and differential second order harmonic distortion impact the ability of to correct quadrature error. The recommended minimum requirement for Ext LO input pins is a combination of no more than 20 degree differential phase error, 1dB differential amplitude error, 2.5% duty cycle error, and less than -30dBc even order harmonics (primarily 2nd order).

The ADRV910X provides special mode of operation for external LO in range from 500MHz to 1000MHz. In that region it is possible to inject external LO that will produce RF Channel frequency with x1 multiplier. For example:

For FEXTLO = 500 MHz the FCHANNEL = 500 MHz

For FEXTLO = 1000 MHz, the FCHANNEL = 1000 MHz

IMPEDANCE MATCHING NETWORK EXAMPLES

Impedance matching networks are required to achieve performance levels noted on the datasheet. This section provides example topologies and components used on the ADRV910x evaluation Board. The impedance matching networks provided in this section have not been evaluated in terms of Mean Time to Failure (MTTF) in high volume production. Please consult with component vendors for long-term reliability concerns. Additionally, please consult with balun vendors to determine appropriate conditions for DC biasing.

The schematics in [Figure 138](#), [Figure 139](#) and [Figure 140](#) show two or three circuit elements in parallel marked DNI (Do Not Include). This was done on the evaluation board schematic to accommodate different component configurations for different frequency ranges. Only one set of SMD component pads are placed on the board to provide a physical location that can be used for the selected parallel circuit element. For example, R216, L216, and C216 components only have one set of SMD pads for one SMD component. The schematic shows that in a generic port impedance matching network, the series elements may be either a resistor, inductor or a capacitor whereas the shunt elements may be either an inductor or a capacitor. Only one component of each parallel combination is placed in a practical application. Note that in some matching circuits, some shunt elements may not be required. All components for a given physical location remain DNI in those particular applications.

RX1± and RX2± Impedance Matching Network

The ADRV910X evaluation board utilizes both the top and bottom layers of the PCB evaluation platform to accommodate two balun footprints. The 0805 footprint accommodates the high frequency narrowband baluns while the backside accommodates the larger DB1627 case style transformer.

The PCB traces of the evaluation board were included in the simulation when designing the impedance match. [Table 45](#) and [Table 46](#) provide impedance matching networks specific to the ADRV910X evaluation board to support the carrier frequency range from 70MHz to 3GHz. The component values apply to RX1± and RX2± respectively..

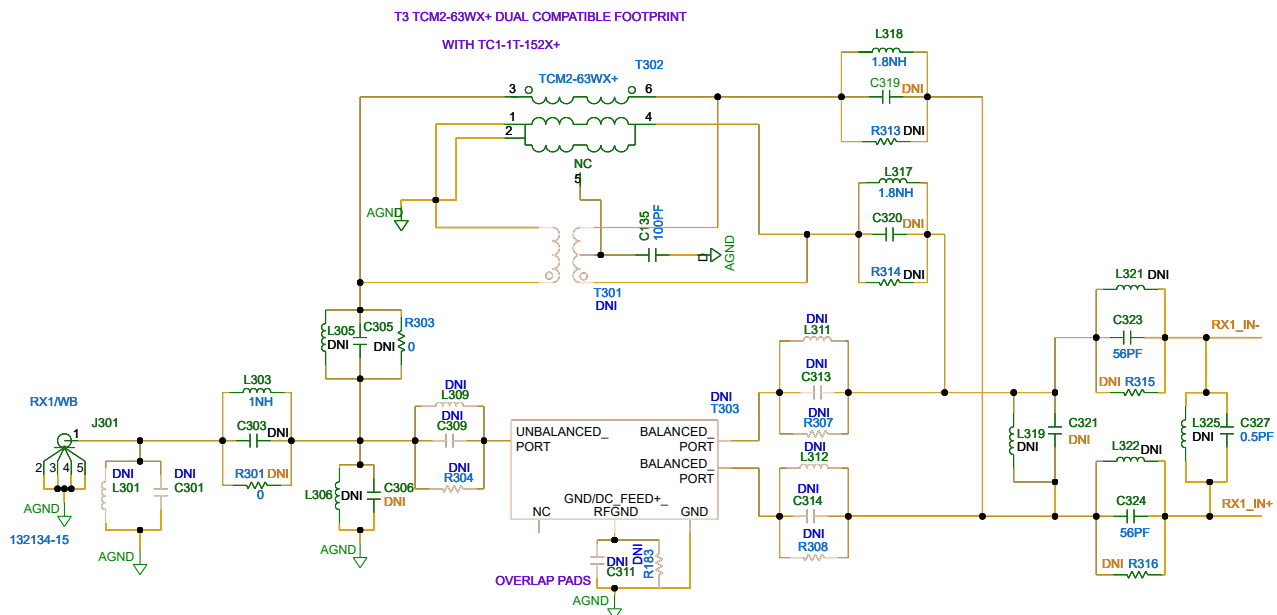


Figure 138. RX1 WB Impedance Matching Network (70MHz~3GHz)

RF PORT INTERFACE

Table 45. RX1 WB Impedance Matching Network

Location	Value	Manufacturer	Part Number
L303	1.0 nH	Murata	LQP03TN1N0B02D
L317/L318	1.8nH	Murata	LQP03HQ1N8W02
C323/C324	56pF	Murata	GRM0335C1H560GA01D
C327	0.5pF	Murata	GJM0335C1ER50WB01D

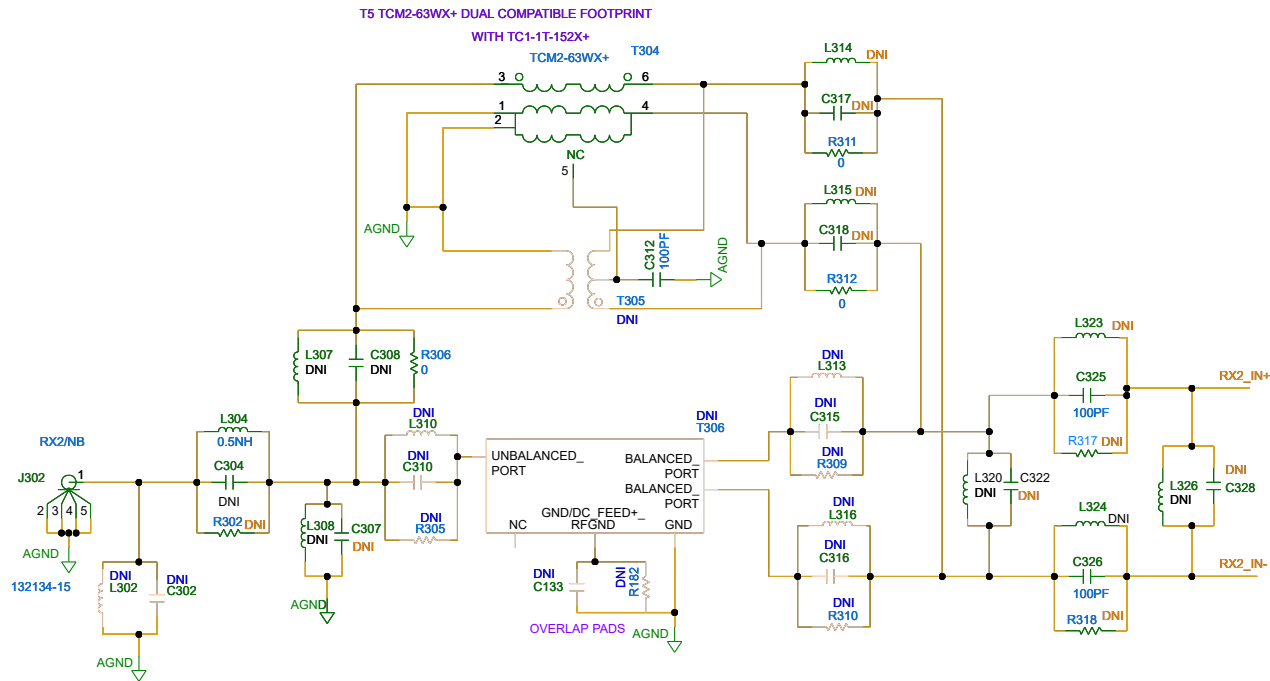


Figure 139. RX2 NB Impedance Matching Network (70MHz~3GHz)

Table 46. RX2 NB Impedance Matching Network

Location	Value	Manufacturer	Part Number
L304	0.5 nH	Murata	LQP03TG0N5B02D
C325/C326	100pF	Murata	GRM0335C1E101FA01D

TX± Impedance Matching Network

For the TX path, the ADRV910x evaluation board utilizes both the top and bottom layers of the PCB evaluation platform to accommodate two balun footprints. The 0805 footprint accommodates the high frequency narrowband baluns while the backside accommodates the larger AT224-1A case style transformer.

The ADRV910X evaluation board provides two options in providing the DC common mode bias for the TX outputs. For transformers that provide a DC feed pin, this can be used to bias the TX output. For transformers that do not provide a DC feed pin, the TX output is biased to 1.8V through pull up inductors. Only one bias option should be chosen, and provisions should be made to disable the unused path.

The PCB traces of the evaluation board were included in the simulation when designing the impedance match. Figure 140 and Table 47 provides impedance matching networks specific to the ADRV910X evaluation board. The component values apply to TX±. Placement of C102 should be as close to DC feed pin of balun T403 as its purpose is to eliminate TX spectrum spurs and dampen the transients. Ground terminal of C102 should be tied to a ground plane and the cap should be oriented in the same direction of ground plane surrounding TX input trace so that the return current forms as small a loop as possible with the ground plane.

RF PORT INTERFACE

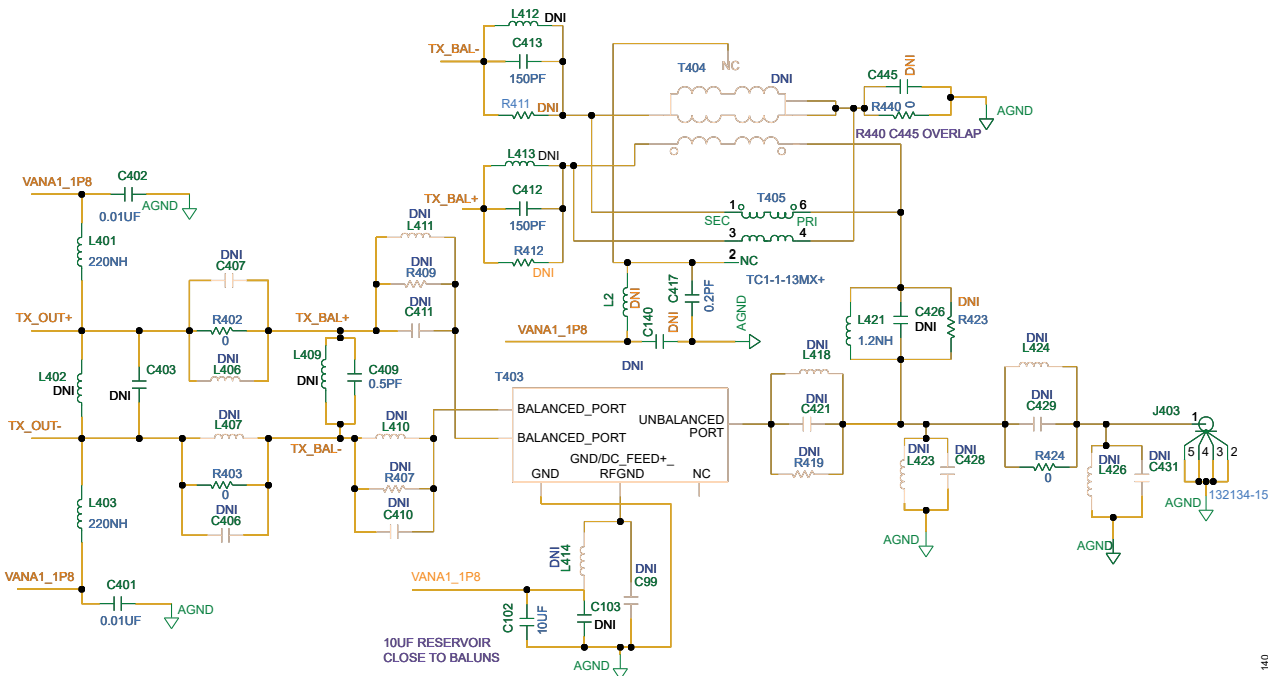


Figure 140. TX Impedance Matching Network (70MHz~3GHz)

Table 47. TX Impedance Matching Network

Location	Value	Manufacturer	Part Number
L401/L403	220nH	Murata	LQW18ANR22J00D
C409	0.5pF	Murata	GJM0335C1ER50WB01D
L421	1.2nH	Murata	LQP03TN1N2B02D

External LO Port Impedance Matching Network

External LO1 and LO2 port can be used for injecting LO signal or external VCO signal with very high spectral purity for internal receivers and transmitter. RF matching network for these ports would be implemented on single-ended and differential sides of balun to reduce insertion loss due to reflections at the desired LO frequency. Method of obtaining matching network is similar to RX and TX port matching. Depending on the selected divide ratio of ADRV910X external LO input frequency divider setting, a band of frequency in which external LO matching network need to operate should be correctly derived by the division ratio chosen.

Specially, the external LO ports might not need a matched network for sub 1GHz LO frequency as long as the LO source is close to the ADRV910x device (short trace lengths), users could decide if the matched network is necessary or not based on their application and design details.

For external LO path, the ADRV910X evaluation board utilizes both the top and bottom layers of the PCB evaluation platform to accommodate two balun footprints to cover the full LO frequency range. Figure 141 and Table 48 provide the impedance matching networks specific to the ADRV910X evaluation board.

RF PORT INTERFACE

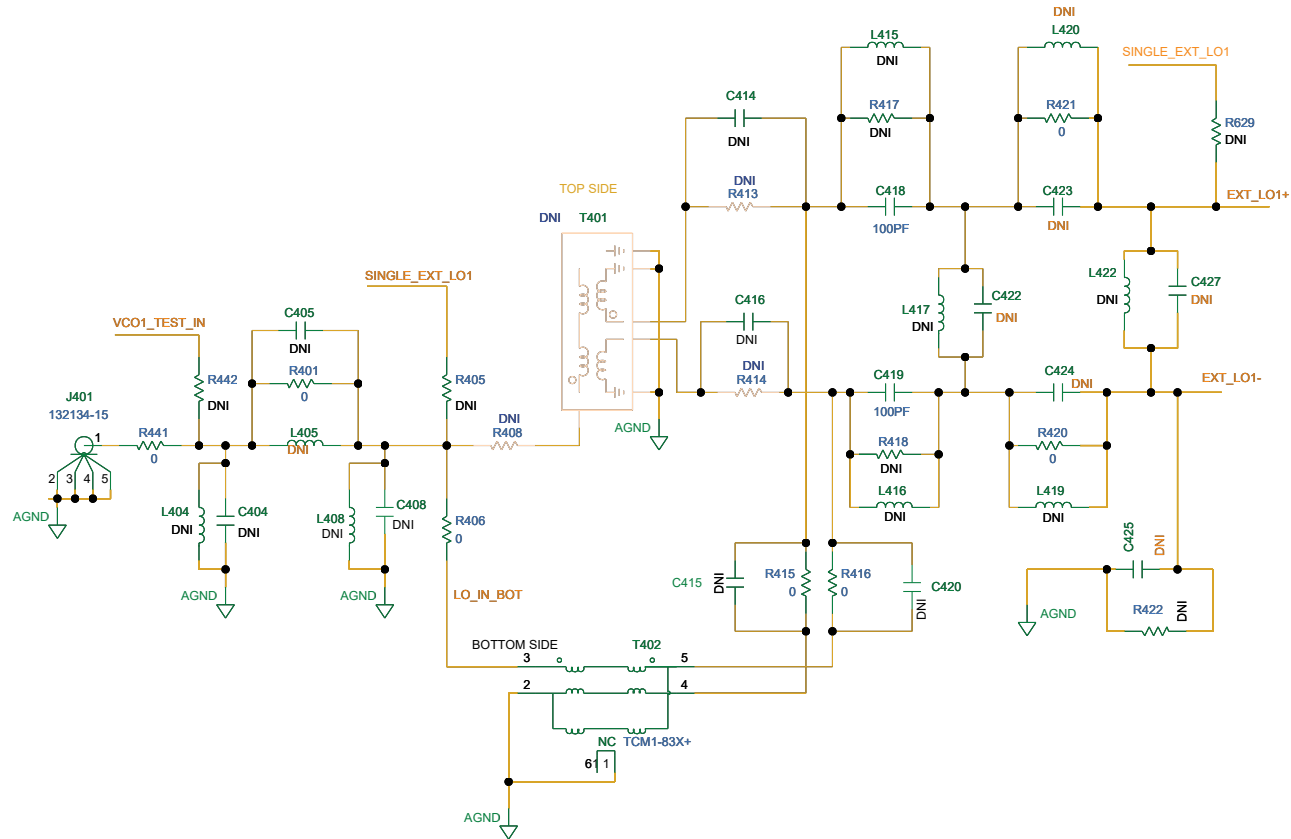


Figure 141. External LO Impedance Matching Network

Table 48. External LO Impedance Matching Network

Location	Value	Manufacturer	Part Number
C418/C419	100pF	Murata	GRM0335C1H101JA01D
T402		MINI CIRCUITS	TCM1-83X+

CONNECTION FOR DEVICE CLOCK (DEV_CLK_IN)

ADRV910x can accommodate 3 different types of external clock signals applied at device clock input pins. A differential low voltage differential signaling (LVDS) clock signal or a single-ended clipped sinewave clock signal from a TCXO can be applied to the device input pins. Furthermore, a crystal can be connected to device clock input pins to configure it as a crystal oscillator/driver by applying DC voltage into MODEA pin as shown in Table 49.

Table 49. Device Clock Input Interface Modes Description

Voltage Applied at MODEA Pin	Device Clock Input Electrical Interface	DEV_CLK_OUT Divider Value Applied to DEV_CLK_IN Signal	Note
0 V (grounded)	LVDS	/16	Up to 100 MHz clock
0.45 V	CMOS or XTAL	/2	CMOS(10MHz to 80MHz)/XTAL(20 MHz to 80 MHz) with Nominal Gm multiplier = x8
0.9 V	CMOS or XTAL	/2	CMOS(10MHz to 80MHz)/XTAL(20 MHz to 80 MHz) with Nominal Gm multiplier = x6
1.35 V	CMOS or XTAL	/2	CMOS(10MHz to 80MHz)/XTAL(20 MHz to 80 MHz) with Nominal Gm multiplier = x2
1.8 V	CMOS or XTAL	/2	CMOS(10MHz to 80MHz)/XTAL(20 MHz to 80 MHz) with Nominal Gm multiplier = x4

By applying 1.8 V to MODEA pin A for CMOS interface mode, a clipped sinewave clock signal from a TCXO can be applied to pin named DEV_CLK_IN+ via a AC coupling capacitor and pin DEV_CLK_IN- should be left unconnected as shown in Figure 143.

RF PORT INTERFACE

A Xtal should be connected to both DEV_CLK_IN+ and DEV_CLK_IN- pins with a DC voltage between 0.45 and 1.8 V applied to MODEA pin as shown in [Figure 144](#).

When LVDS mode input clock interface is selected with MODEA pin grounded, an external clock is used as the reference clock for the RFPLL and the Clocking PLL on the device and thus must be a very clean clock source. Connect the external clock inputs to the DEV_CLK_IN+ and DEV_CLK_IN- balls via AC coupling capacitors and should be terminated with 100 Ω as shown in [Figure 142](#). The inputs are biased on the device to a 200 mV voltage level. The operational frequency range of the DEV_CLK signal is between 10 MHz and 100 MHz. Ensure that the external clock peak-to-peak amplitude does not exceed 800mV (Note that either positive and negative side of differential input pins should not exceed 400mV peak-to-peak.). For best synthesizer performance, a high slew rate signal is best with fast rise and fall times.

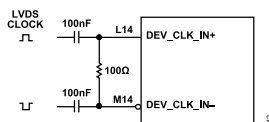


Figure 142. LVDS Interface Mode

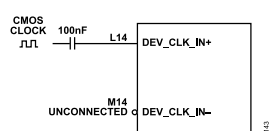


Figure 143. CMOS Interface Mode

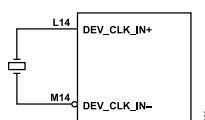


Figure 144. Crystal (XTAL) Interface Mode

Device clock input board traces connected to device clock inputs balls should be implemented with stripline transmission lines using inner copper layers in PCB stackup. The frequency of device clock input signal can go as high as 100MHz and stripline transmission line approach will provide better signal integrity of clock signal especially at higher frequency as well as superior shielding of RF emission of device clock signal.

The DEV_CLK_IN signal is available on the DEV_CLK_OUT pin. [Table 49](#) describes default division applied to DEV_CLK_IN signal after power up. Users can change this divider later on using API command. It should be noted that DEV_CLK_OUT pin is a CMOS type pin with 80MHz of its maximum frequency of operation. It is intended to be used to provide clock to BBIC or on-board microcontroller or audio CODEC type devices. It is not intended to be used by another RF sensitive IC.

DEV_CLK_IN Phase Noise Requirements

As discussed in [LO Phase Noise](#) section, the DEV_CLK_IN dominates the close in LO phase noise. To prevent performance degradation, the DEV_CLK reference must be a very clean signal. Best performance from the synthesizer would result if the applied reference were ideal, however that is unrealistic. As an example, [Table 50](#) provides recommendation for DEV_CLK when ADRV910X is intended to operate with LMR type standards. Ideally DEV_CLK phase noise requirement should be derived from customer specific application and its requirements set for adjacent channel rejection.

In general, using a higher phase noise source can degrade performance delivered by ADRV910X transceiver.

Table 50. DEV_CLK_IN Phase Noise Requirements for LMR Type Applications

Frequency Offset From Carrier	PLL Loop Bandwidth Optimized for LMR Type Applications, 38.4 MHz (dBc/Hz)
100 Hz	-106
1000 Hz	-151
10 kHz	-151
100 kHz	-151
10 MHz	-151

POWER SUPPLY OVERVIEW

POWER MANAGEMENT CONSIDERATIONS

The ADRV910X family of devices requires four power supply domains, all of which must be connected for the ADRV910X to function:

1. 1.0 V digital: Connect this supply to the device through the two VDIG_1P0 pins. This supply feeds all digital blocks. Ideally, this supply should be isolated from the VDDA_1P0 supplies, but in cases where there are restrictions on the PCB footprints, the two supplies can be merged, potentially at the expense of some performance. It has a tolerance of $\pm 5\%$ at the input pins on the ADRV910X and can draw high currents, meaning that it is critical to account for resistive losses when designing the power delivery network, and to ensure that the traces are balanced so that each pin sees the same impedance from the power supply.
2. 1.0 V analog: This domain groups all other 1.0 V pins that are not VDIG_1P0. It supplies the LOs, PLLs, the Rx mixers, the Tx upconverter, the TIAs, DAC and ADCs, and the clock VCO and PLL for clock generation. This supply should be low noise as it supplies blocks that are sensitive to noise, and should ideally be isolated from the VDIG_1P0 supply for optimal performance. It has a tolerance of $\pm 2.5\%$ and can draw potentially draw high currents, so it is important that resistive losses are accounted for when designing the power delivery network.
3. 1.8 V digital: Connect this supply to the device through the VDIG_1P8 pin. This domain serves as the voltage reference for the digital interfaces (SPI, SSI, and GPIOs), and as such should be supplied from the same output as the one used for the BBIC. It has a tolerance of $\pm 5\%$.
4. 1.8 V analog: This domain groups all other 1.8 V pins that are not VDIG_1P8. This domain is used for DEV_CLK, external VCOs (and the associated PLLs), DAC and ADCs, AuxDACs and AuxADC, and the AGPIO reference voltage. It has a tolerance of $\pm 5\%$.

It is essential that each supply pin is locally decoupled using a 1 μ F 0201 capacitor placed directly under the ball.

Note: During operation, supply currents can vary significantly, especially if operating in the TDD mode. The supply must have adequate capacity to provide the necessary current (as indicated in the user guide) to maintain the performance criteria over all the process and temperature variations. Analog Devices recommends adding at least 15% margin to all supply maximums to ensure proper operation under all conditions. It is also critical to ensure that the supply remains stable when supplying the chip in TDD modes.

POWER SUPPLY SEQUENCE

The ADRV910X requires a specific power-up sequence to avoid undesired power-up currents. The optimal power-on sequence requires the 1.0 V domains (VDIG_1P0 and VDDA_1P0) to power up first, followed by the 1.8 V domains (VDIG_1P8 and VDDA_1P8). Once the power supply has stabilized, toggle the RESET signal before proceeding with the initialization and configuration. The power-down sequence recommendation is similar to power-up. Disable the 1.8 V supplies before disabling the 1.0 V supplies. If such a sequence is not possible, then disable all the sources of the supplies simultaneously to ensure there are no back-feeding circuits to power down.

POWER SUPPLY CONNECTIONS

Careful low-noise power management design is a key aspect to ensure good performance. [Table 51](#) lists the pin number, pin type and name, expected voltage on that pin, and a brief description of routing technique together with the block it powers on the chip. Power supply to the ADRV910x is delivered after the star configuration, where a separate trace from a common power plane is used to power each power-supply pin. Refer to the [Printed Circuit Board Layout Recommendations](#) section and an evaluation board CAD layout file for details.

Table 51. Power Supply Connections: Balls and Descriptions

Ball Name	Ball Number	Description
VSSA	A1, A2, A3, A5, A6, A9, A10, A11, A13, A14, B4, B6, B7, B8, B9, B11, B13, C2, C3, C4, C13, D1, D2, D4, D6, D9, D11, D13, D14, E4, E7, E8, E9, E11, F2, F3, F12, F13, G3, G12, H1, H2, H3, H4, H5, H6, H7, H8, H9, H11, H12, H13, H14, K10, K11, K12, K13, K14, M12, N13, N14	Analog ground.
VSSA/TEST $\pm$	H10, J10	Reserved for internal testing. Connect to analog ground.
VSSD	L6, L7, P1, P14	Digital ground.
VANA1_1P0	A4	1.0 V analog supply for Rx1 TIA, Tx Baseband Filter. This pin is sensitive to supply noise.
VANA2_1P0	B12	1.0 V analog supply for Rx2 TIA. This pin is sensitive to supply noise.
VTXLO_1P0	B3	1.0 V analog supply for Tx LO distribution and Tx upconverter. This pin is sensitive to supply noise.
VRX1LO_1P0	B5	1.0 V analog supply for Rx1 LO distribution, Rx1 mixer. This pin is sensitive to supply noise.
VRX2LO_1P0	B10	1.0 V analog supply for Rx2 LO distribution, Rx2 mixer. This pin is sensitive to supply noise.

POWER SUPPLY OVERVIEW

Table 51. Power Supply Connections: Balls and Descriptions (Continued)

Ball Name	Ball Number	Description
VRFLO1_1P0	D10	1.0 V analog supply for RF LO 1 generation. This pin is sensitive to supply noise.
VRFLO2_1P0	D5	1.0 V analog supply for RF LO 2 generation. This pin is sensitive to supply noise.
VRFSYN1_1P0	D8	1.0 V analog supply for RF PLL1. This pin is sensitive to supply noise.
VRFSYN2_1P0	D7	1.0 V analog supply for RF PLL2. This pin is sensitive to supply noise.
VEXTLO1_1P0	F14	1.0 V analog supply for external LO1. This pin is sensitive to supply noise.
VEXTLO2_1P0	F1	1.0 V analog supply for external LO2. This pin is sensitive to supply noise.
VCONV1_1P0	G7	1.0 V analog supply for Tx DAC, Rx1 ADC. This pin is sensitive to supply noise.
VCONV2_1P0	G9	1.0 V analog supply for Rx2 ADC. This pin is sensitive to supply noise.
VAGPIO_1P0	G8	1.0 V analog supply for auxiliary DACs, auxiliary ADCs, and AGPIO signals.
VDEVCLK_1P0	L13	1.0 V analog supply for DEV_CLK circuitry. This pin is sensitive to supply noise.
VCLKVCO_1P0	J12	1.0 V analog supply for clock PLL VCO, clock generation.
VCLKVCO_CAP	J11	0.8 V internal supply node for clock PLL. Bypass VCLKVCO_CAP with a 4.7 μ F capacitor.
VCLSYN_1P0	J14	1.0 V analog supply for clock PLL.
VDIG_1P0	K6, K7	1.0 V digital core supply. Connect pins K6 and K7 together. Use a wide trace to connect the VDIG_1P0 pins to a supply domain separate from other 1.0 V supplies, and provide reservoir capacitance close to the chip.
VANA1_1P8	B2	1.8 V analog supply for Rx1 mixer, Rx1 TIA, Tx mixer, Tx baseband filter. This pin is sensitive to supply noise.
VANA2_1P8	A12	1.8 V analog supply for Rx2 mixer, Rx2 TIA, and internal references. This pin is sensitive to supply noise.
VRFVCO1_1P8	G11	1.8 V analog supply for RF VCO1, RF PLL1 (for external VCO1). This pin is sensitive to supply noise.
VRFVCO2_1P8	G4	1.8 V analog supply for RF VCO2, RF PLL2 (for external VCO2). This pin is sensitive to supply noise.
VRFVCO1_CAP	F11	1.6 V internal supply node for RF PLL1 circuitry. Bypass VRFVCO1_CAP with a 4.7 μ F capacitor.
VRFVCO2_CAP	F4	1.6 V internal supply node for RF PLL2 circuitry. Bypass VRFVCO2_CAP with a 4.7 μ F capacitor.
VCONV1_1P8	F7	1.8 V analog supply for Tx DAC, Rx1 ADC. This pin is sensitive to supply noise.
VCONV2_1P8	F9	1.8 V analog supply for Rx2 ADC. This pin is sensitive to supply noise.
VAGPIO_1P8	F8	1.8 V analog supply for DEV_CLK, MCS, Auxiliary DACs, Auxiliary ADCs, and AGPIO signals.
VDIG_1P8	M6	1.8 V digital supply input for data port interface (CMOS SSI and LVDS SSI modes), SPI signals, control input and output signals, and DGPIO interface.

POWER SUPPLY ARCHITECTURE

The power supply solution provided on the ADRV910x Customer Evaluation PCB is designed to be easily adapted to suit a wide array of systems. The default configuration is designed to fit moderately space constrained applications, but space for additional components has been included in the layout to allow prototyping of more demanding applications at the expense of space.

POWER SUPPLY OVERVIEW

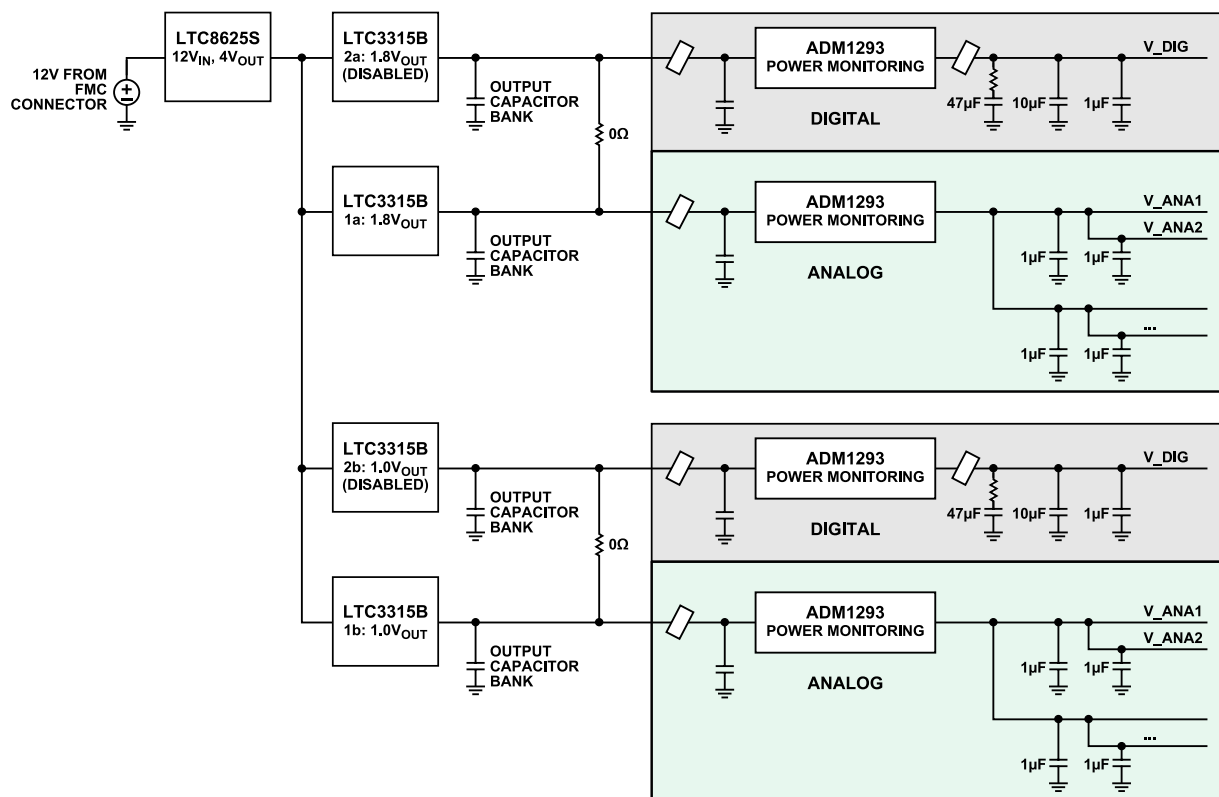


Figure 145. Block Diagram of the Power Supply Included on the ADRV910x Evaluation Board

The 12 V supply from the FMC connector is first stepped down to a lower voltage using the LT8625S, which is more representative of most systems that the ADRV910x is likely to be used in. In this case, the output voltage of the LT8625S is 4 V.

The ADRV910x itself is powered from a dual output buck converter, the LTC3315B. This can supply up to 2 A to the 1.0 V supplies, and a further 2 A to the 1.8 V supplies. If an application requires more than 2 A from either output, a second LTC3315B is included on the PCB. Enabling the second LTC3315B allows the splitting of the analog supply pins for the digital supply pins, improving noise isolation. The LTC3315B works with input voltages between 2.25 V and 5.5 V.

The outputs of the LTC3315B are loaded with capacitor banks to improve stability, which are followed by ferrite bead and capacitor filters. Following these filters is the ADM1293 power supply monitoring circuit. This consists of one ADM1293 per supply domain, which measures the voltage and uses a series resistor (in this case, 20 mOhm with a custom footprint) to measure current draw in each specific domain with a 2.5% accuracy. The inclusion of the ADM1293 monitoring circuit is purely for the purpose of evaluation, as it allows the user to measure the power consumption in any configuration of the ADRV910x. However, the series resistance in this circuit has the additional benefit of helping to de-Q the filters on each domain, which improves supply stability.

POWER SUPPLY DESIGN BUDGET

ADRV910x power consumption varies significantly depending on the selected profile and operating mode. The recommended approach for custom profile power budgeting is to measure the actual current consumption on the evaluation board under representative operating conditions.

Table 52 and Table 53 provide reference values for maximum supply current in different operating modes. It is recommended to include at least 20% additional margin in the power supply design budget.

Table 52. Narrow band (TX/RX2) Maximum Current Estimation

Power Rail	Max Current (mA)	Comment
VDD 1.0	300	With PS2 enable
VDDA 1.0	270	
VDD 1.8	5	CMOS SSI

POWER SUPPLY OVERVIEW

Table 52. Narrow band (TX/RX2) Maximum Current Estimation (Continued)

Power Rail	Max Current (mA)	Comment
VDDA 1.8	300	Internal LO

Table 53. Wide band (TX/RX1) and NB RX2 Maximum Current Estimation

Power Rail	Max Current (mA)	Comment
VDD 1.0	500	With PS2 enable
VDDA 1.0	400	
VDD 1.8	40	LVDS SSI
VDDA 1.8	500	Internal LO

IMPROVING NOISE PERFORMANCE

In cases where supply noise is suspected to degrade the performance of the ADRV910X, changes can be made to the supply and its power distribution network to improve its noise filtering capability. These changes include:

- ▶ Using a supply source with better noise performance (e.g. a low-dropout linear regulator, or a Silent Switcher™ switching regulator).
- ▶ Ensuring that the layout of the power distribution network does not cause coupling between noisy and sensitive rails.
- ▶ Using separate regulator outputs for the digital domains and the analog domains.
- ▶ Separating any shorted supply pins, and installing their associated ferrite bead and capacitor filters to further isolate them from each other.

Note: when installing additional filtering, it is important that there is sufficient damping included to prevent resonant peaks in the frequency response of those filters from destabilizing the supply.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

The ADRV910x is a highly integrated RF agile transceiver SOC with significant signal conditioning integrated onto one chip. Due to the integration complexity of the ADRV910x and its high pin count, a carefully printed circuit board (PCB) layout is important to optimize performance. This section provides a checklist of issues to look for and guidelines on how to optimize the PCB to mitigate performance issues. The goal of this document is to help achieve the best possible performance from the ADRV910x while reducing board layout effort. It is assumed that the reader is an experienced analog/RF engineer who understands RF PCB layouts and has an understanding of RF transmission lines as well as low-noise analog design techniques. The ADRV910x evaluation card is used as the reference for this information, but all guidelines are best practices that can be applied to other reference designs. This document provides guidelines for system designers and discusses the following issues relative to layout and power management.

SELECTING THE PCB MATERIAL AND STACKUP

Figure 146 shows the PCB stackup used for the ADRV910x customer evaluation boards. These boards employ 10 layers to achieve proper routing and isolation to best demonstrate all device functionality. The dielectric material used is I-SPEED with a thickness of 7 mil on outer layers. The board design uses the I-SPEED laminate for its low loss tangent at high frequencies. The ground planes under the I-SPEED laminate (layers 2 and 9) are the reference planes for the transmission lines routed on the outer surfaces. These layers are solid copper planes under the RF traces with no discontinuities. Layers 2 and 9 are crucial to maintaining the RF signal integrity.

RF traces on the outer layers must be a controlled impedance to get the best performance. These outer layers use 0.5 ounce copper. The 0.5- and 1-ounce copper thicknesses are used for all the inner layers in this board. All ground planes on this board are full copper floods with no splits except vias, through-hole components, and isolation structures.

Layers 5 and 6 are mainly used to route power-supply domains. The data port interface lines are routed on layers 1, 8, and 10. Those layers have impedance control set to 100 Ω differential for the differential LVDS pairs. The remaining digital signals are routed on inner layers 3 and 8.

There are no buried-in or blind vias used in this PCB design. All vias used in the PCB design are through-hole type. For vias carrying high frequency or RF sensitive signals, apply the back drilling technique.

Layer	Cu Thick. (mils)	Cu Foil wt (oz)	DK	Lam. Thick. (mils)	Description
1	2.00	0.5 oz	3.56	7.00	Core I-Speed 7.00mils 2x1086 0.5 oz / 0.5 oz VLP2 18.25Gx24.25
2	0.60	0.5 oz	3.40	5.07	Prepreg I-Speed 1035(74.5)/1035(74.5) 18.25Gx24.25
3	0.60	0.5 oz	3.26	5.00	Core I-Speed 5.00mils 2x1067 0.5 oz / 1 oz VLP2 18.25Gx24.25
4	1.20	1 oz	3.40	4.80	Prepreg I-Speed 1035(74.5)/1035(74.5) 18.25Gx24.25
5	1.20	1 oz	3.45	5.00	Core I-Speed 5.00mils 2x1067 1 oz / 1 oz VLP2 18.25Gx24.25
6	1.20	1 oz	3.40	4.80	Prepreg I-Speed 1035(74.5)/1035(74.5) 18.25Gx24.25
7	1.20	1 oz	3.26	5.00	Core I-Speed 5.00mils 2x1067 0.5 oz / 1 oz VLP2 18.25Gx24.25
8	0.60	0.5 oz	3.40	5.08	Prepreg I-Speed 1035(74.5)/1035(74.5) 18.25Gx24.25
9	0.60	0.5 oz	3.56	7.00	Core I-Speed 7.00mils 2x1086 0.5 oz / 0.5 oz VLP2 18.25Gx24.25
10	2.00	0.5 oz			

Figure 146. ADRV910x Customer Evaluation Card PCB Stackup

FANOUT AND TRACE SPACE GUIDELINES

The ADRV910x device family uses a 196-pin ball grid array (BGA) 10 mm × 10 mm package. The pitch between the pins is 0.65 mm. This small pitch makes it impractical to route all signals on a single layer. There are RF pins on the outer edges of the ADRV910x package. This helps in routing the critical signals without a fanout via. Each digital signal is routed from the BGA pad using a 3 mil trace. The trace is connected to the BGA using the via-in-the-pad structure. The signals are buried in the inner layers of the board for routing to other parts of the system.

Take extra care to ensure that the DEV_CLK signal is shielded from any potential source of noise. The recommended approach is to use differential signalling for the DEV_CLK clock. Route the data port interface signals as 100 Ω differential pairs when used in the LVDS-SSI mode.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

Figure 147 shows the fan-out scheme of the ADRV910x evaluation card. There are no traces routed between the BGA pads on the top layer except the EXT_LO1-/EXT_LO2- signals. As mentioned, the ADRV910x evaluation card uses the via-in-the-pad technique. Use this routing approach for the ADRV910x if there are no issues with manufacturing capabilities.

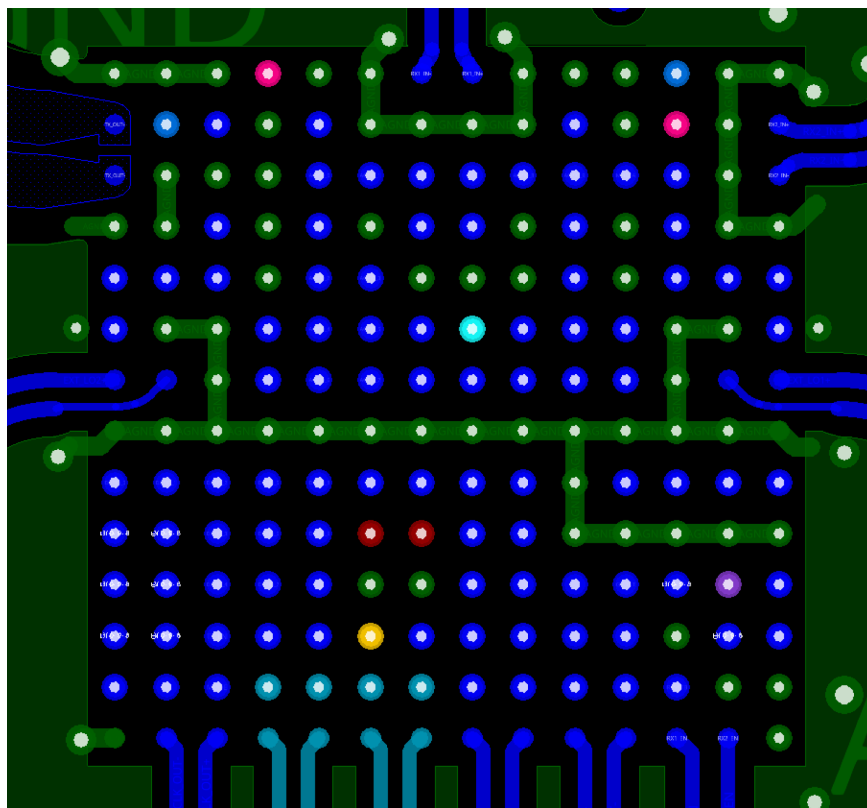


Figure 147. Trace Fan-Out Scheme on ADRV910x Evaluation Card (PCB Layer TOP Enabled)

COMPONENT PLACEMENT AND ROUTING PRIORITIES

The ADRV910x transceiver requires a few external components to function, and the ones needed require careful placement and routing to optimize performance. This section provides a priority order and checklist to properly place and route critical signals and components, as well as those whose location and isolation are not as critical.

Board layout design involves compromise. The recommendations in this section are intended for wide RF bandwidth applications. For narrow RF bandwidth applications, the board line impedance parameters within this document may not be optimal.

The following list provides general suggestions for board design:

- ▶ Match the customer board design as close as possible to the ADRV910X board design.
- ▶ Be attentive to power distribution and power ground return methodology.
- ▶ Do not run high-speed digital lines close to the DC power distribution routes or RF line routes

Signals With Highest Routing Priority

RF lines, external VCO PLL, DEV_CLK clock are the most critical signals. Route these with the highest priority. Figure 148 shows the general directions to route each signal so that are properly isolated from noisy signals.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

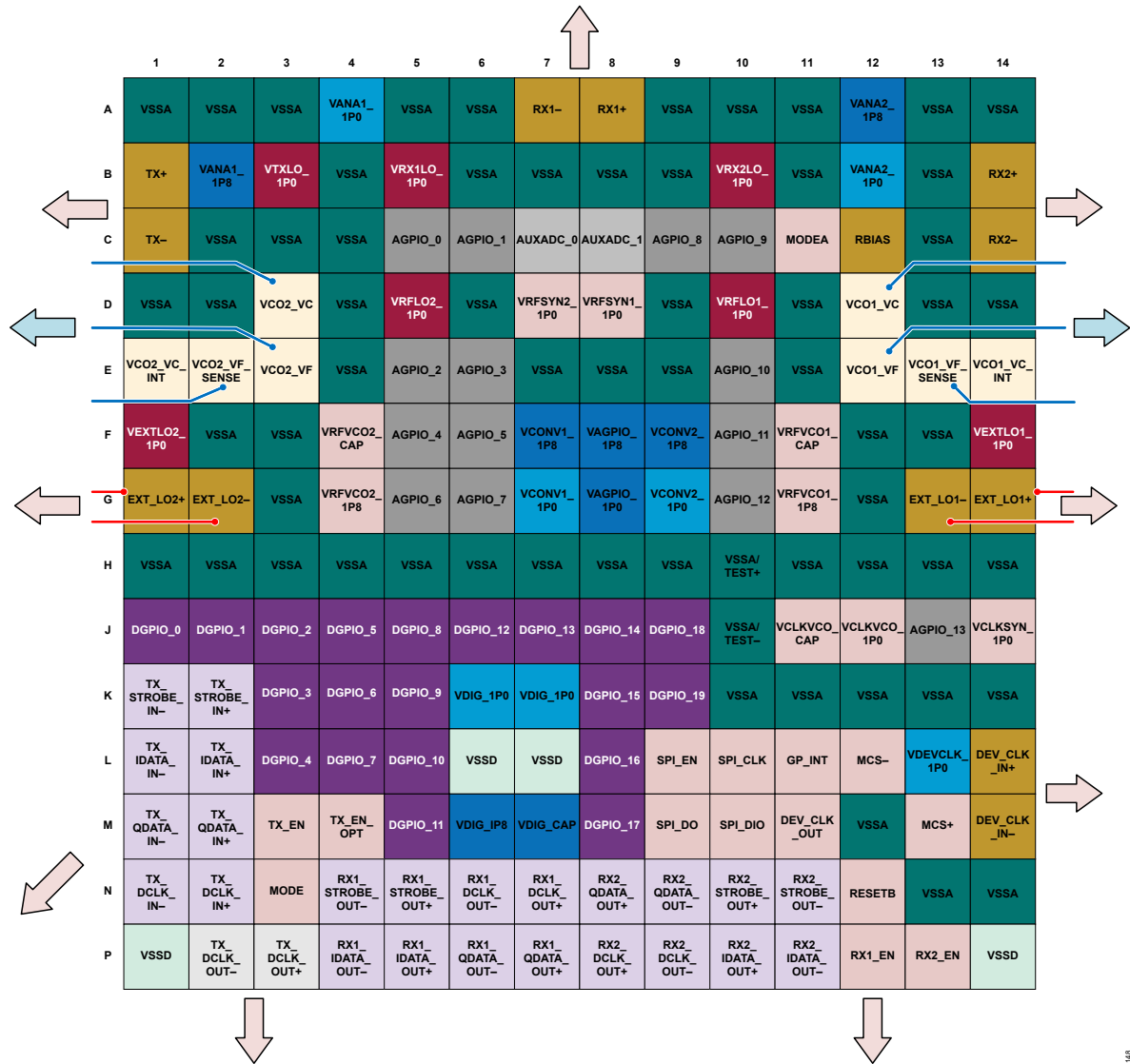


Figure 148. RF IO, EXT RFPLL, DEV_CLK and Data Port signal Routing Guidelines

- ▶ RF baluns are typically used to interface single-ended signals to the differential receiver and transmitter ports. These baluns and their associated matching circuits affect the overall RF performance. Make every effort to optimize the component selection and placement to avoid performance degradation. The section describes the proper matching circuit placement and routing in more detail. Please see the respective paragraph for more information.
- ▶ Use microstrip or coplanar waveguides (CPWG) for transmission lines. These structures do not require via structures that cause additional impedance discontinuities that vary across frequency.
- ▶ 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 transmitter. This is a compromise impedance with respect to frequency and a good starting point for design. Optimize the Z_{DIFF} to fit a narrower frequency range. It is desirable to design the lines for reasonable coupling (-10 dB to -15 dB) to promote adequate electromagnetic interference (EMI) suppression performance.
- ▶ In most cases, the required board artwork stackup is going to be different than the ADRV910x evaluation board stackup. Optimization of RF transmission lines specific to the desired board environment is essential to the design and layout process.
- ▶ The ADRV910x evaluation board uses microstrip lines for receiver and transmitter RF traces. Some data port signals are routed using a combination of microstrip lines at the bottom of the PCB and stripline traces on the internal layers due to board complexity. In general, RF traces should avoid using vias if a direct line route is possible.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

- ▶ Differential lines from the balun to the RX and TX pins must be as short as possible. The length of the single-ended transmission line should also be short to minimize the effects of parasitic coupling.
- ▶ The system designer can optimize the RF performance with the proper selection of balun, matching components, and AC-coupling capacitors.
- ▶ The external LO traces and the DEV_CLK_IN traces can require matching components as well to ensure optimal performance. The matching network design is explained in greater detail in the respective section of this document.
- ▶ RF signal path isolation is critical to achieving the level of isolation specified in the ADRV910x data sheet. The [Isolation Techniques Used on the ADRV910x Evaluation Board](#) section provides more details on proper isolation.
- ▶ For RF transmitter output, install a 10 μ F capacitor near the balun power-supply pin connected to the VANA1_1P8 supplies. If baluns with no DC supply connection are used, supply power to the transmitter outputs using RF chokes. Connect chokes between the VANA1_1P8 and TX output. In both cases, the 10 μ F capacitor acts as a reservoir for transmitter supply current. The [Tx Interface](#) section describes the transmitter output power supply configuration in more detail.
- ▶ Connect the external clock inputs to the DEV_CLK_IN + (L14) and DEV_CLK_IN - (M14) pins using AC-coupling capacitors. Use a 100 Ω termination at the input to the device. [Figure 149](#) illustrates the recommended placement for termination resistor near the DEV_CLK_IN pins. Traces are shielded by surrounding ground with vias staggered along the edge of the differential trace pair. This arrangement creates a shielded channel that protects the reference clock from any interference from other signals.

Note: this is valid only for differential DEV_CLK_IN mode.

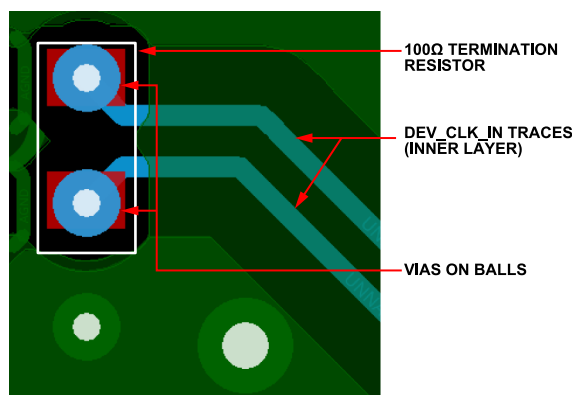


Figure 149. DEV_CLK_IN signal Routing Recommendations

- ▶ The EXT_LO1+ (G14), EXT_LO1- (G13), EXT_LO2+ (G2), EXT_LO2- (G1) pins are internally DC-biased. If an external LO(VCO) is used, connect it through AC-coupling capacitors.
- ▶ EXT RFPLL coarse tune (VCOx_VC, D12/D3) and fine tune (VCOx_VF, E12/E3) signals are noise sensitive, routes these signals away from the power supply pins/vias. Place the coarse tune integration capacitor as close as possible to the VCOx_VC_INT pins (E14/E1).
- ▶ The data port interface is routed at the beginning of the PCB design and with the same priority as RF signals. This is especially important if data port runs in the LVDS configuration. Pay attention to provide appropriate isolation between the data port differential pairs.

Signals With Secondary Routing Priority

Power-supply quality has a direct impact on the overall system performance. To achieve optimal performance, follow the recommendations for power-supply routing. The following recommendations outline how to route different power domains, which supplies to connect to the same supply but separated by a ferrite bead.

A general recommendation for power-supply routing is to follow the star methodology, in which each power domain is delivered by a separate trace from the source supply. Make sure that each power trace is surrounded by ground. [Figure 150](#) shows an example of such traces routed on the evaluation card on Layer 5. Each trace is separated from any other signal by ground plane fill and vias. This approach is essential to provide necessary isolation between power domains.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

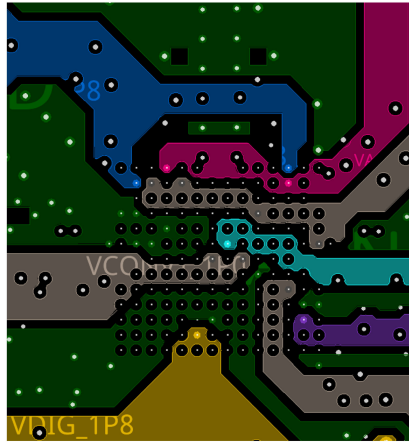


Figure 150. Layout Example of Power-Supply Connections Routed with Ground Shielding (Layer 5)

Figure 151 shows an example of how to place the ferrite beads, reservoir capacitors, and decoupling capacitors. The recommendation is to connect a ferrite bead between a power plane and ADRV910x at a distance from the ADRV910x. The ferrite bead supplies a trace with a reservoir capacitor connected to it. That trace is then shielded with ground and provides power to the ADRV910x power pin. Place a 1 μ F capacitor near the power-supply pin with the ground side of the bypass capacitor placed so that ground currents flow away from other power pins and their bypass capacitors.

Note, the ferrite beads on ADRV910x evaluation board are default bypassed, users could choose to install some or all of them when there is noise/performance issue needs to be debugged. Refer [Improving Noise Performance](#) session for more information.

Place ceramic 4.7 μ F bypass capacitors at the VCLKVCO_CAP, VRFVCO1_CAP, VRFVCO2_CAP, VDIG_CAP pins which are internal LDO outputs. Place these capacitors as close as possible to the device with the ground side of the bypass capacitor placed so that ground currents flow away from other power pins and their bypass capacitors, if at all possible.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

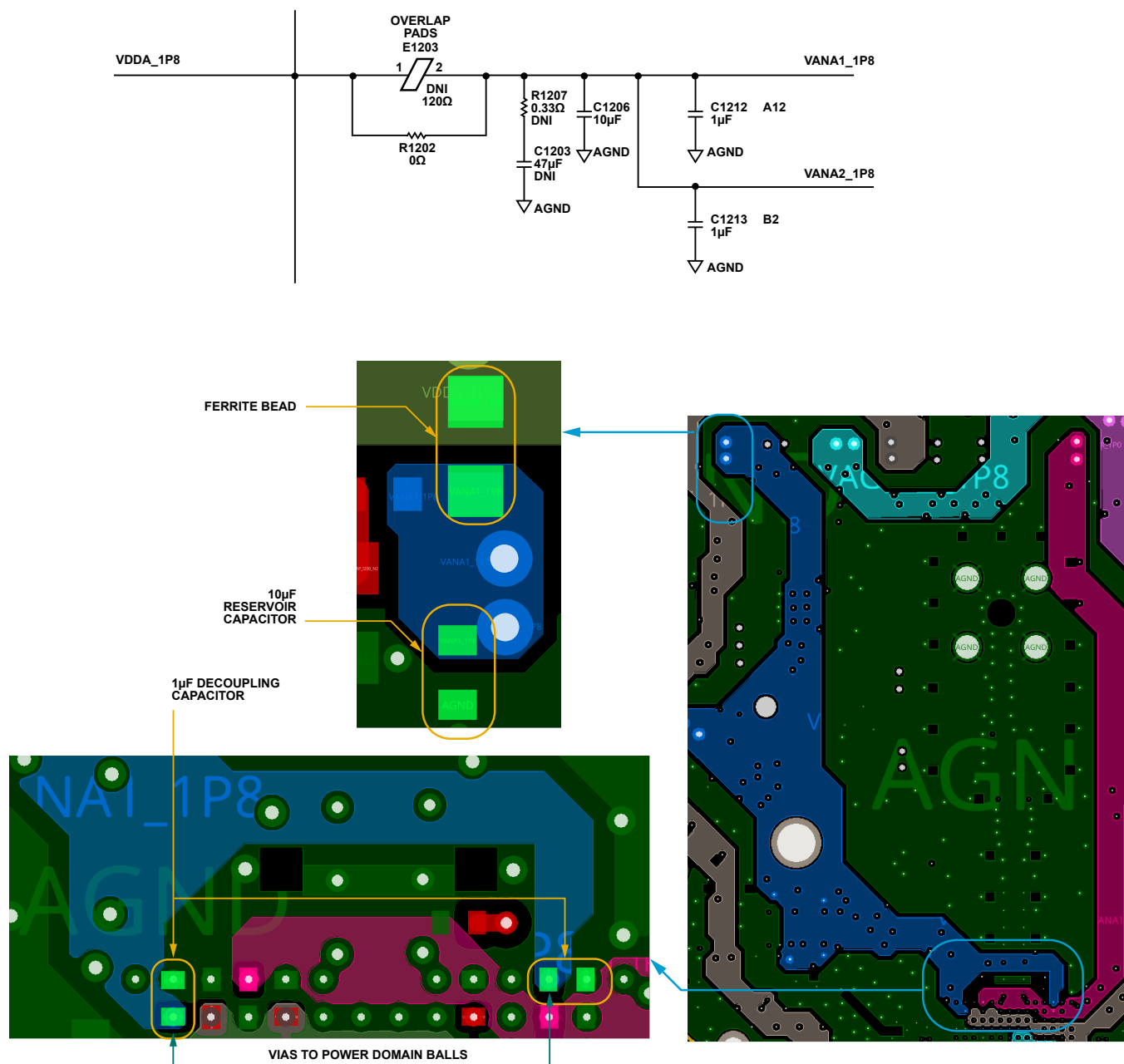


Figure 151. Placement Example of Ferrite Beads, Reservoir, and Bypass Capacitors (Layers: 5-Power, and BOTTOM)

Signals With Lowest Routing Priority

The following guidelines govern the signals with the lowest signal routing priority. Route these after all critical signal routes are completed so they do not interfere with the critical component placement and routing.

- ▶ Connect a 4.99 kΩ resistor to the RBIAS pin (C12). This resistor must have a 1% tolerance or better.
- ▶ The device supports joint test action group (JTAG) boundary scan and PS2 ARM debugging, and the MODE pin is used to access the function. Connect the MODE pin (N3) to ground for normal operation.
- ▶ Connect the RESETB pin (N12) to VDIG_1P8 with a 10 kΩ resistor for normal operation. Reset the device by driving this pin low.
- ▶ When routing digital signals from rows J and below, it is important to route them away from the analog section (rows A through G). The digital-signal routing should not pass above row H. Refer [Figure 148](#) for device pin configuration diagram.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

- ▶ Route the AGPIO_X signals using inner PCB layers. These signals control the analog blocks such as power amplifiers or low-noise amplifiers, external VCO coarse tune switch. Also use the AGPIO_0 through AGPIO_3 as general-purpose analog outputs when muxed with the internal AUXDAC outputs. To prevent noise coupling into these signals, route them away from the digital region (above the row H). Route the AuxADC_X signals using inner PCB layers. These signals sense the analog voltage levels such as temperature sensors.
- ▶ The MODEA signal sets up the operation of the DEV_CLK_IN $\pm$ pins (LVDS differential, CMOS single-ended, XTAL with different bias voltage). Follow the recommendations in the [Connection for Device Clock \(DEV_CLK_IN\)](#) section when controlling this pin.

RF AND DATA PORT TRANSMISSION LANE LAYOUT

RF Line Design Summary

The RF line design is a compromise among many variables. Line impedance, line-to-line coupling, and physical size represent the parameters subject to compromise. Smallest physical size is in direct opposition to the ZCM of the line, which is directly opposed to the line EMI performance. In addition, the interface between the RF line width and the device ball pad diameter on the PCB represents a potential discontinuity. As the RF line width approaches the ball pad diameter, the risk associated with potential interface discontinuity reduces.

Balanced lines for differential mode signalling used between the device and the RF balun should be as short as possible. The length of the single-ended transmission lines for RF signals should also be as short as possible. Keeping signal paths as short as possible reduces susceptibility to undesired signal coupling and the effects of parasitic capacitance, inductance, and loss on the transfer function of the transmission line and impedance matching network system. The routing of these signal paths is the most critical factor in optimizing performance and, therefore, route these prior to any other signals and maintain the highest priority in the PCB layout process.

The circuit in [Figure 152](#) shows the layout topology for the chosen receiver matching network. Note the location and orientation of each component (placement is critical for expected performance). Similarly, the circuit in [Figure 153](#) shows the layout topology used for the transmitter matching network.

All the RF signals must have a solid ground reference under each path to maintain the desired impedance. None of the critical traces should run over a discontinuity in the ground reference.

There are three different Balun options available for on ADRV910x evaluation board to cover the carrier frequency range for 70MHz to 6GHz, but only one Balun and the relative RF matching network are assembled for the specified board.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

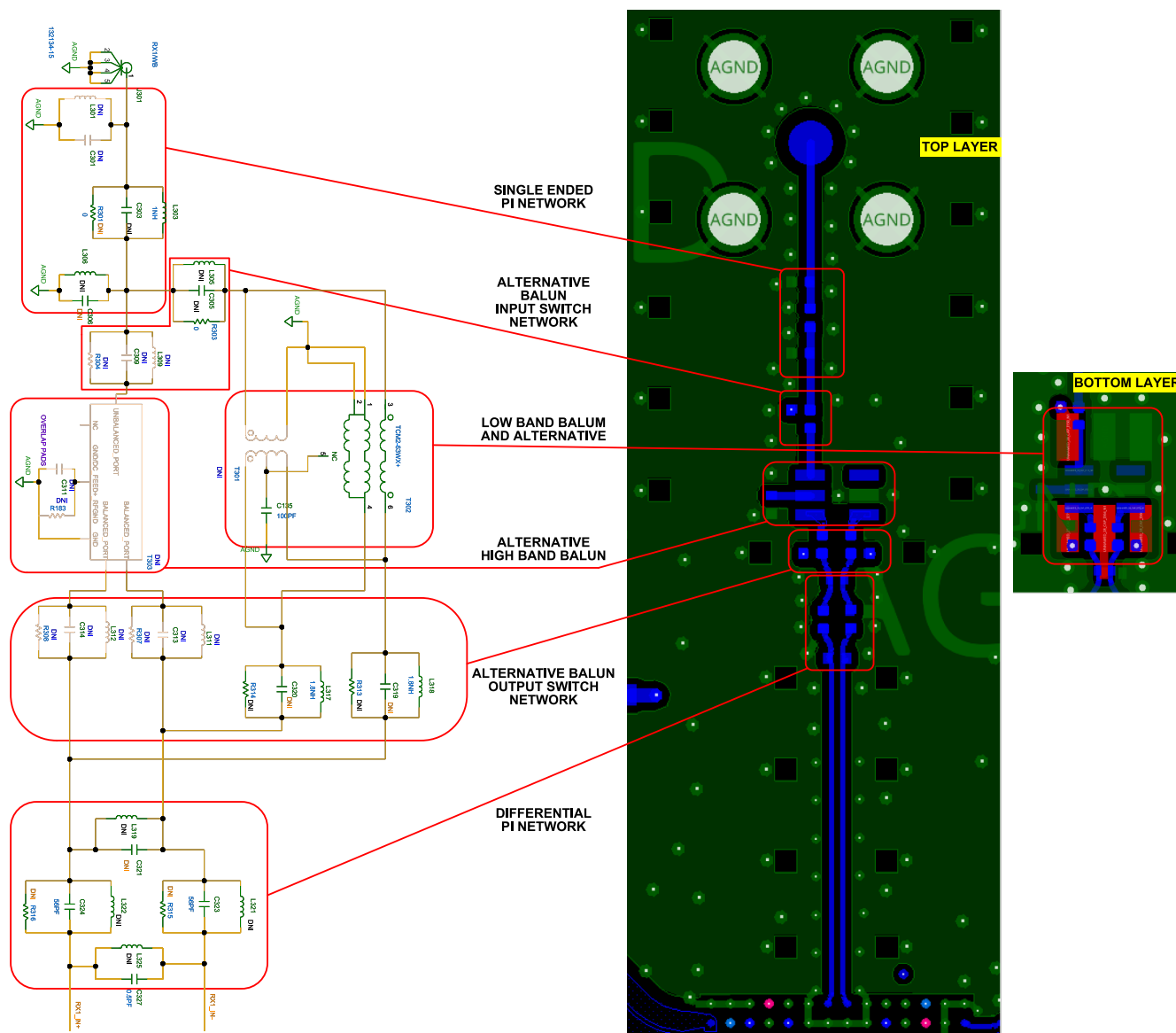


Figure 152. RX1 Matching Network on ADRV910x Evaluation Board

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

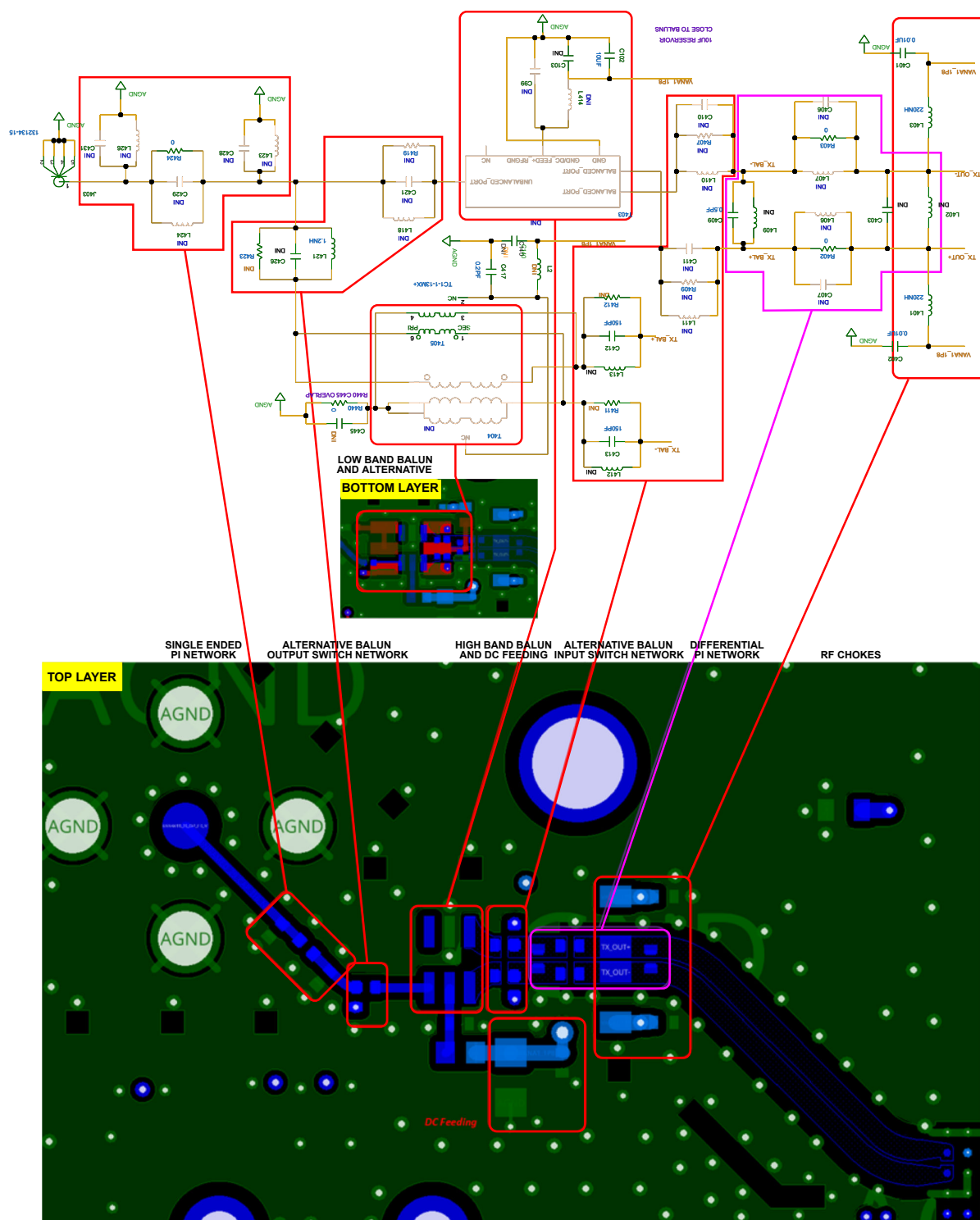


Figure 153. Transmitter Matching Network on ADRV910x Evaluation Board

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

Transmitter Balun and DC Supply Options

As discussed in [Tx Interface](#) section, TX output is DC biased to a 1.8V supply voltage using either RF chokes (wire-wound inductors) or a transformer centre tap connection. Careful design of the DC bias network is required to ensure optimal RF performance levels. When designing the DC bias network, select components with low DC resistance (R_{DCR}) to minimize the voltage drop across the series parasitic resistance element with either of the suggested dc bias schemes suggested in [Figure 129](#) and [Figure 130](#).

The ADRV910X evaluation board provides flexibility to configure the transmitter output to work with either a center-tapped transformer (balun) or a set of two closely matched wire-wound chokes.

Impedance matching networks on the balun single-ended port are usually required to achieve optimum performance. In addition, AC-coupling is often required on the single-ended side if the balun contains a DC path from one of the transmitter's differential outputs to the single-ended port.

The transmitter requires approximately 200 mA supplied through an external connection. The PCB layout of the ADRV910x board allows the use of external chokes to provide 1.8 V power domain to the ADRV910X outputs. This allows users to try different baluns that do not have a DC center tap pin to supply the bias voltage to the transmitter outputs. In such scenario, take care to match both the chokes to avoid potential current spikes. Difference in parameters between both the chokes can cause unwanted emission at transmitter output.

When using a transmitter balun able to conduct DC, place the decoupling cap near the transmitter balun as close as possible to the balun's DC feed pin. A 10 μ F reservoir capacitor and a 0.1 μ F decoupling capacitor are helpful on the DC feed pin to eliminate transmitter spectrum spurs and dampen the transients.

To reduce switching transients when attenuation settings change, the balun DC feed is powered directly by the 1.8 V plane. The geometry of the 1.8 V plane is designed on ADRV910x evaluation board so that each pair of chokes is associated with its transmitter output. The VANA1_1P8 is used to power the TX output.

Note that if the differential input to the balun can create a DC short to ground (through the balun), the series matching components must be capacitors. If a short can form on the single-ended side, the single-end series blocking element must be a capacitor.

SSI Data Port Trace Routing Recommendations

The data port interface transfers I/Q data between the BBIC/FPGA and ADRV910X transmitter and receiver datapaths. There are two possible modes of operation for the SSI data port:

- ▶ CMOS-SSI mode (single-ended), with clock rate for data transfer up to 80 MHz.
- ▶ LVDS-SSI mode (differential), with clock rate for data transfer up to 500 MHz DDR (1000 MHz data rate).

Follow the correct layout practice while routing the SSI signals.

If selecting the CMOS-SSI mode, single-ended signal lines between the ADRV910X and BBIC/FPGA must be as short as possible. Also reduce the trace capacitance to minimize the current needed by the ADRV910X to drive the line. Refer to the ADRV910X data sheet for details on pin drive capabilities.

If selecting the LVDS-SSI mode, route all the LVDS signals as 100 Ω differential pairs. When routing the PCB layout for the LVDS-SSI data lines, the designer must route the signals using stripline or microstrip traces. There are positives and negatives for each.

- ▶ Stripline has less loss and emits less EMI than microstrip lines, but stripline traces require the use of vias that can add complexity to the task of controlling the impedance by adding line inductance.
- ▶ Microstrip is easier to implement if the component placement and density allow for routing on the top layer, simplifying the task of controlling the impedance.

If using the top layer of the PCB is problematic or the advantages of stripline are desirable, then follow these recommendations:

- ▶ Minimize the number of vias.
- ▶ Use blind vias wherever possible to eliminate via stub effects and use micro vias to minimize via inductance.
- ▶ If using standard vias, use maximum via length to minimize the stub size. For example, on an eight-layer board, use Layer 7 for the stripline pair.
- ▶ For each via pair, place a pair of ground vias close to them to minimize the impedance discontinuity.

In the LVDS-SSI mode:

- ▶ For transmitter data port inputs, implement 100 Ω termination inside the ADRV910X.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

- For receiver data port outputs, implement 100 Ω termination at the receiver end.

ISOLATION TECHNIQUES USED ON THE ADRV910X EVALUATION BOARD

Given the density of sensitive and critical signals, there are significant isolation challenges when designing a PCB for the ADRV910x. Analytically determining aggressor-to-victim isolation in a system is very complex and involves considering vector combinations of aggressor signals and coupling mechanisms.

The following are the primary coupling mechanisms between the RF I/O paths on the evaluation board:

- Magnetic field coupling
- Surface propagation
- Cross-domain coupling through ground

There are several strategies to reduce the impact of these coupling mechanisms on the ADRV910x customer evaluation. Open large slots in the ground plane between the RF I/O paths. These discontinuities prevent surface propagation. These structures consist of a combination of slots and square apertures. Both structures are present on every copper layer of the PCB stack. The advantage of using square apertures is that signals can be routed between the openings without disturbing the isolation benefits provided by the array of apertures. A careful designer notices various bends in the routing of differential paths. Develop and tune these routes through iterative electromagnetic simulation to minimize magnetic field coupling between differential paths. Figure 154 shows these techniques.

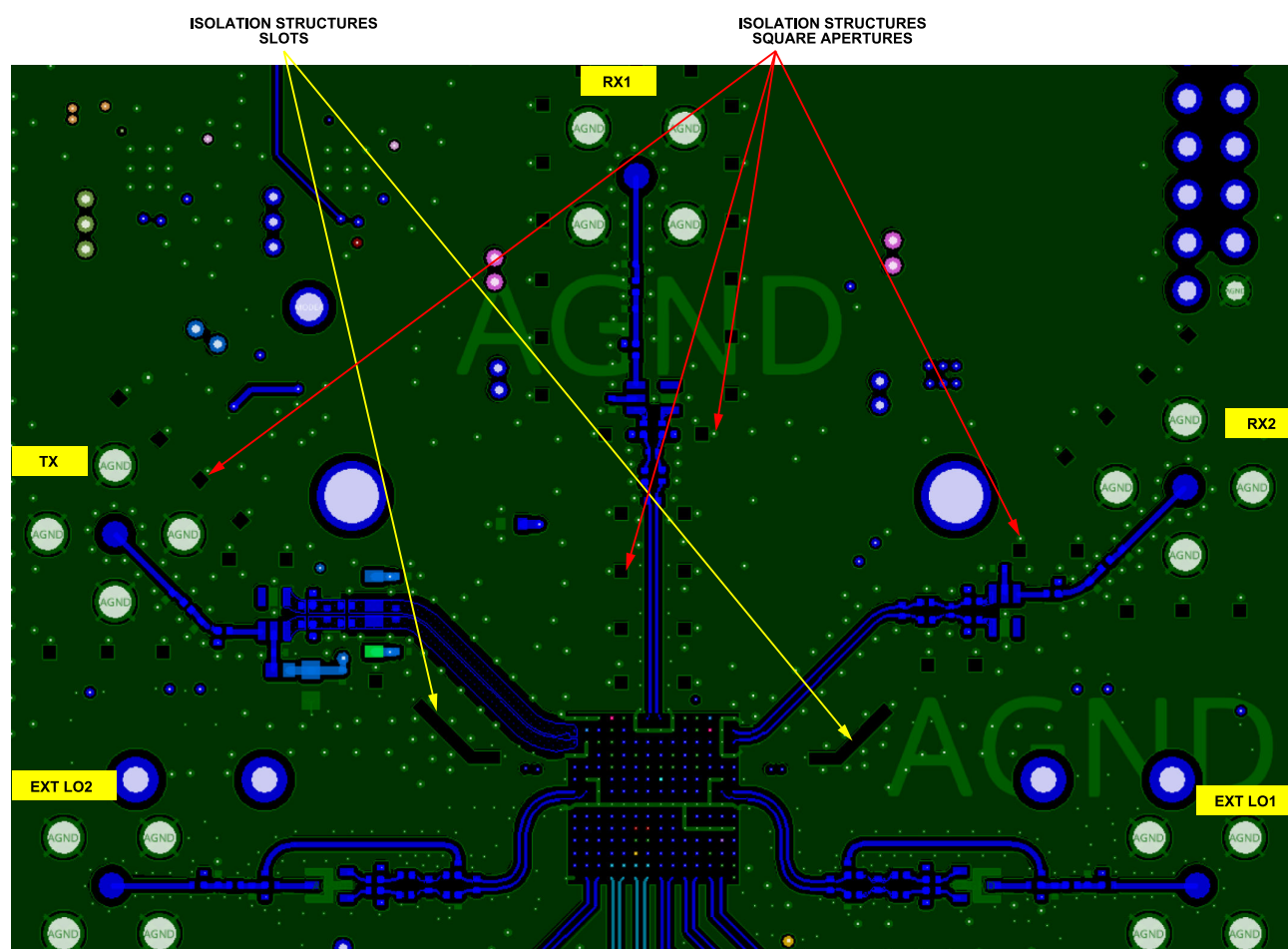


Figure 154. RF IO Isolation Structures

When utilizing the proposed isolating structures, it is important to place ground vias around the slots and apertures. Figure 155 shows the methodology used on the ADRV910x evaluation board. When using slots, place the ground vias at each end of the slots and along each side. When using square apertures, place at least one single ground via next to each square. These vias are through-hole vias connecting the top to the bottom layer and all layers in between. The function of these vias is to steer return current to the ground planes near the apertures.

PRINTED CIRCUIT BOARD LAYOUT RECOMMENDATIONS

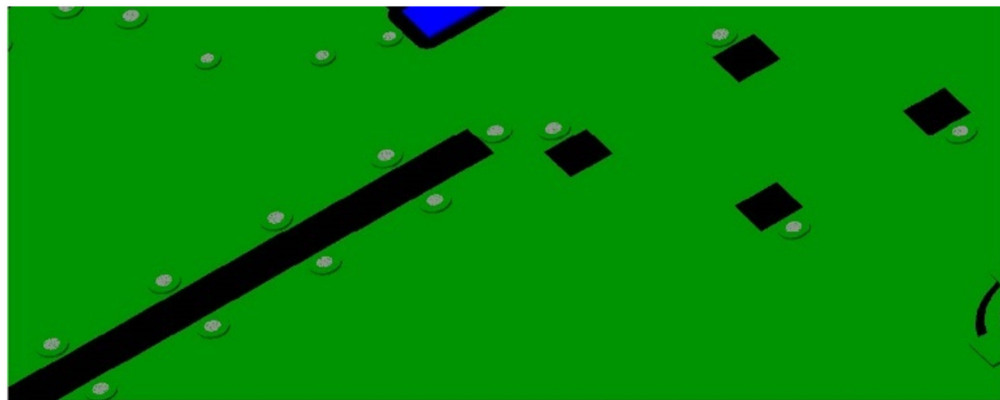


Figure 155. Current Steering Vias Placed Near Isolation Slots and Square Apertures

For accurate slot spacing and square apertures layout, use simulation software when designing a PCB for an ADRV910x based transceiver. As a general rule, the spacing between square apertures should be no more than one-tenth of the shortest wavelength. Calculate the wavelength calculated using below equation.

$$Wave\ Length[m] = \frac{300}{Frequency[MHz] * \sqrt{\epsilon_r}} \quad (24)$$

Where:

ϵ_r is the dielectric constant of the isolator material

For ISOLA I-speed material, $\epsilon_r = 3.56$, and for FR4-408 HR material, $\epsilon_r = 3.77$.

Example: Given a maximum RF signal frequency of 6 GHz, for ISOLA I-speed material, using microstrip structures and $\epsilon_r = 3.56$, the minimum wavelength is approximately 26.4 mm. To fulfil the one-tenth of a wavelength rule, the square aperture spacing should be at a distance of 2.64 mm or closer.

Provide additional shielding by connecting VSSA balls under the device to form a shield around RF I/O ball pairs. This ground provides a termination for stray electric fields. Figure 154 shows how this is done for the RF I/O ports. Use ground vias along single-ended RF I/O traces. Optimal via spacing is one-tenth of a wavelength, but that spacing can vary somewhat due to practical layout considerations

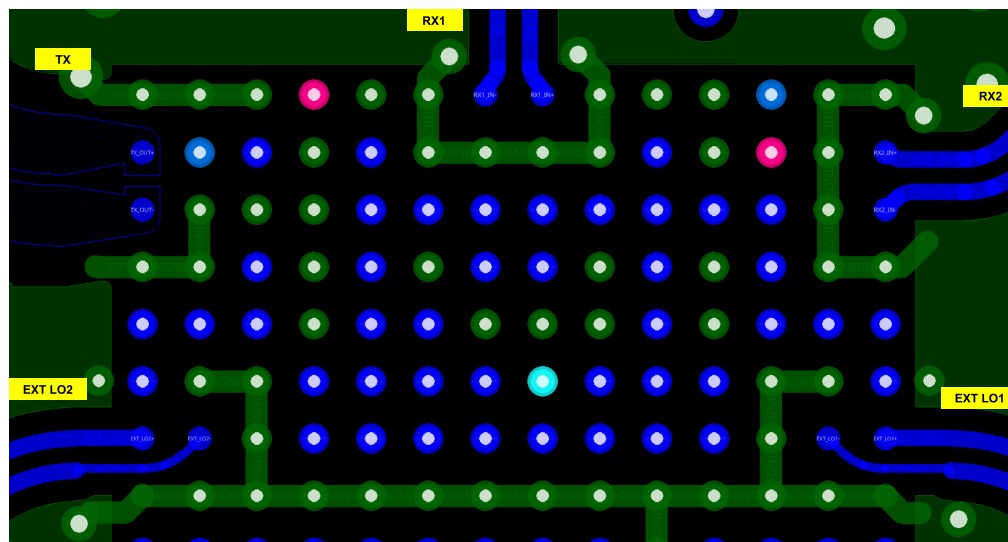


Figure 156. Shielding of RF I/O Launches

Space and align the RF I/O baluns to reduce magnetic coupling from the structures in the balun package. Take care to reduce the crosstalk over shared grounds between baluns. Another precaution is to place and orient the SMA connectors to minimize the connector-to-connector coupling between ports.

PROCESSOR SUBSYSTEM 2 (PS2)

The ADRV910x SoC is a general-purpose software-defined radio (SDR) with extended capabilities for modem functionalities. All ADRV910x variants include a Processor Subsystem 1 (PS1) for the RF agile transceiver functionalities and the ADRV9103 and ADRV9104 variants also include a Processor Subsystem 2 (PS2) open microcontroller for extended user-defined modem capabilities with a variety of scalable hardware accelerators and peripheral interfaces.

This section covers the descriptions of the hardware components in PS2 and the user programming interfaces.

ADRV910X SYSTEM-ON-A-CHIP (SOC) ARCHITECTURE

Overview

The ADRV910x System on Chip (SoC) facilitates the rapid development of flexible wireless connectivity solutions by supporting the implementation of system control and signal processing functionality. This can include the integration of various protocol stack layers, ranging from the PHY layer to the upper layers. It features two processor subsystems, PS1 and PS2, and includes a monitoring function, as illustrated in Figure 157. The Monitor implements the low power operation mode, either powering down or powering up the PS1 and PS2 when the system becomes idle or active. This capability helps users to achieve the lowest system-level power consumption for their ADRV910x SoC based designs.

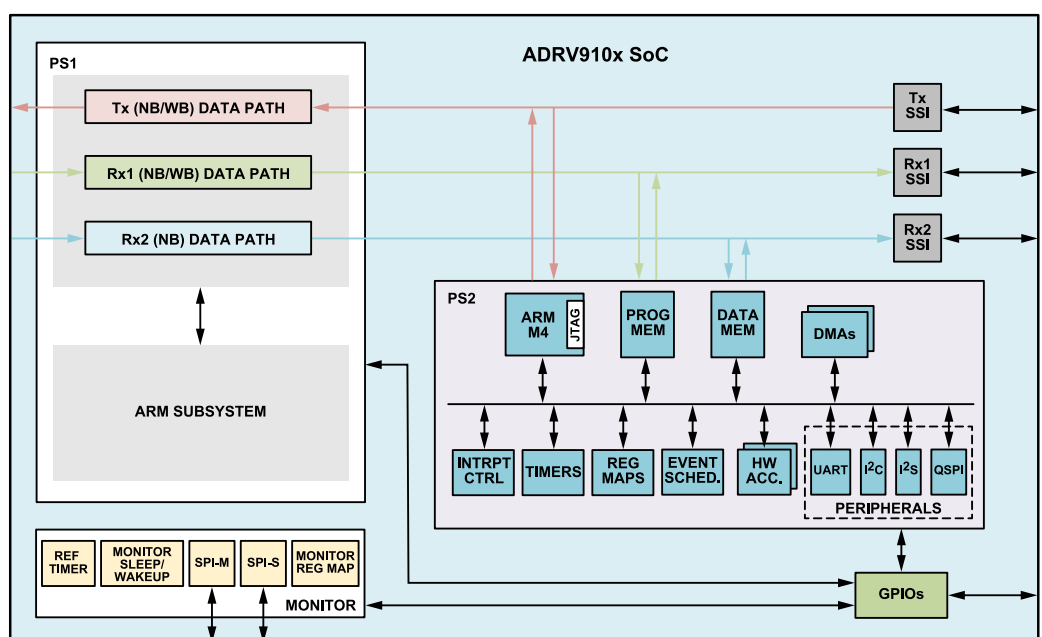


Figure 157. High Level Function Block Diagram of ADRV910x SoC

The PS1 includes all the transceiver functionalities supporting both narrow-band (NB) and wide-band (WB) applications. It consists of two major components:

- ▶ Three Data Paths:
 - ▶ Tx NB/WB data path and algorithms with CMOS and LVDS SSI interface (configurable bandwidth from 12kHz to 2MHz (ADRV9101/3) or 12kHz to 40MHz (ADRV9102/4))
 - ▶ Rx1 NB/WB data path and algorithms with CMOS and LVDS SSI interface (configurable bandwidth from 12kHz to 40MHz)
 - ▶ Rx2 NB data path and algorithms with CMOS and LVDS SSI interface (configurable bandwidth from 12kHz to 2MHz)
- ▶ ARM Subsystem
 - ▶ The ARM subsystem in PS1 implements the transceiver internal algorithms such as initial and tracking calibrations. It is a closed subsystem not accessible by users.

The PS2 provides users with the capability to perform digital signal processing for Tx data, before transmitting it to the PS1 data path, and for Rx data, after receiving it from the PS1 data path. It consists of:

- ▶ ARM processor

PROCESSOR SUBSYSTEM 2 (PS2)

- ▶ ARM Memory
- ▶ DMAs
- ▶ Interrupt controllers
- ▶ Timers
- ▶ Register Maps
- ▶ Event Scheduler
- ▶ Peripherals
- ▶ Hardware Accelerators

As mentioned, the Monitor can be used for either powering down or powering up the PS1 and PS2 when the system becomes idle or active. Note that the Monitor itself is always powered on. It includes:

- ▶ SPI Register Maps
- ▶ SPI Subordinate
- ▶ Sleep/Wakeup Timers
- ▶ Reference Timer
- ▶ SPI Main
- ▶ Monitor Register Maps

As shown in [Figure 157](#), the PS2 has an ARM Cortex-M4 core. Different from the PS1 ARM, which is a closed subsystem, the PS2 is an open platform that performs the user's custom application code. The user can implement a set of digital signal processing algorithms to pre-process the Tx data before sending it to the PS1 data path and post-process the Rx data after receiving it from the PS1 data path. The user can implement multiple layers or even the entire stack of the communication protocol on the PS2 if the memory and MIPS resource requirements are satisfied. Multiple peripherals such as UART, QSPI, I2S and I2C are provided for data communications between PS2 and the external Host IC.

Multiple direct memory access (DMA) blocks are provided to transfer data efficiently:

- ▶ between PS2 memory and SSI or PS1 data path,
- ▶ between PS2 memory and PS2 memory,
- ▶ and between PS2 memory and peripherals.

In addition, a set of hardware accelerators (HWA) are offered in PS2. While not an exhaustive listing, the PS2 HWAs provide mathematical operations such as Arctan and vector multiplication, signal processing functionalities such as FIR and IIR filtering, and even advanced signal processing algorithms such as Turbo, Convolutional and Trellis encoding and decoding.

By utilizing the PS2 functionalities provided in the ADRV910x SoC, users are empowered to achieve a more integrated system design by significantly reducing the signal processing requirements of the baseband integrated circuit, resulting in an optimal system solution for Size, Weight, Power and Cost (SWaP-C). Note not all the PS2 functionalities are supported in the current release.

Examples of Tx and Rx Data Capture and Drive Through PS2

[Figure 158](#) describes three examples of Tx and Rx data capture and drive through PS2:

- ▶ The red line represents the signal flow of Tx data being captured from the Tx SSI interface into PS2 Memory and then after processing in PS2, being driven from PS2 memory to PS1 Tx data path.
- ▶ The green line represents the signal flow of Rx1 data samples being captured into PS2 memory from the PS1 Rx1 data path and then after processing in PS2, being driven from PS2 memory to Rx1 SSI interfaces.
- ▶ The blue line represents the signal flow of Rx2 data samples being captured into PS2 memory from the PS1 Rx2 data path and then after processing in PS2, being driven from PS2 memory to Rx2 SSI interfaces.

PROCESSOR SUBSYSTEM 2 (PS2)

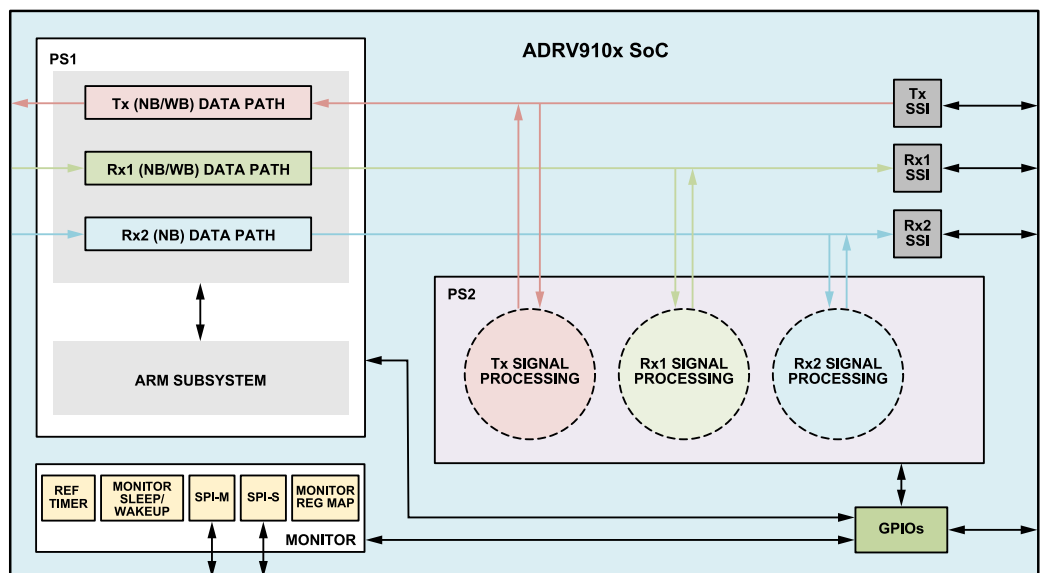


Figure 158. Examples of Tx and Rx Data Capture and Drive through PS2

ADRV910x SoC HOST BOOT Using SPI Subordinate

Figure 159 illustrates the data flow when the Host IC downloads the PS1 and PS2 ARM images to the ADRV910x SoC. In this scenario, the Host IC serves as the SPI main, transmitting both ARM images to the SPI-Subordinate interface of the ADRV910x SoC. The Host IC indicates which ARM image is being transferred, and the ADRV910x SoC directs the data to the appropriate memory banks. Note currently this is the default boot mode.

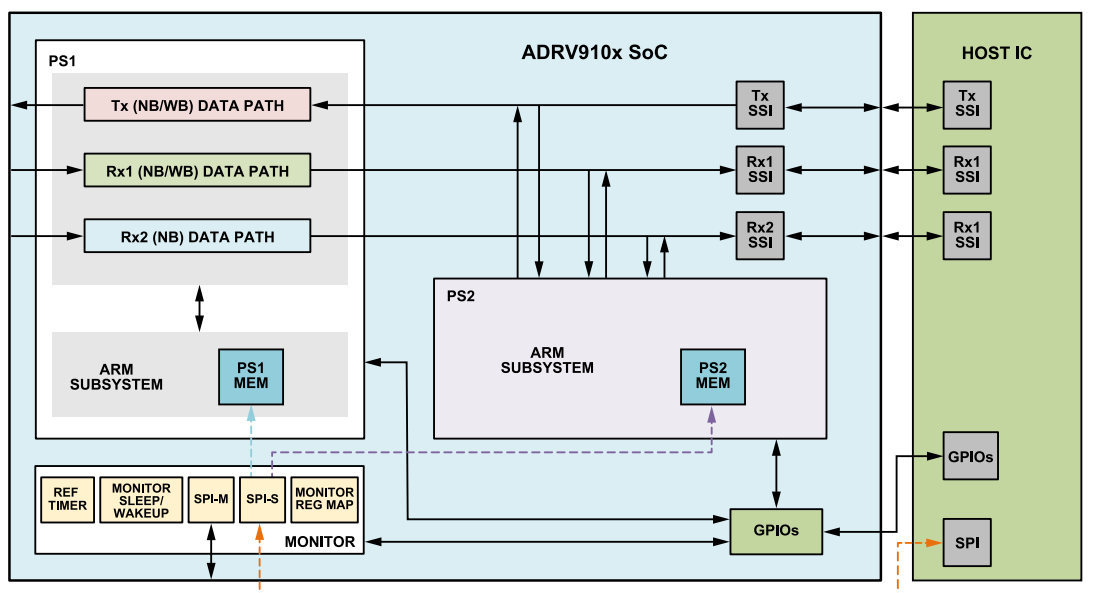


Figure 159. Data Download from Host to PS1 and PS2 ARM through SPI Main

The boot up procedure includes the following major steps:

- ▶ PS1 analog initialization
- ▶ PS1 resource loading including PS1 ARM image, stream image, ARM profile, gain tables, etc.
- ▶ PS2 ARM image loading
- ▶ Start PS1 ARM
- ▶ Perform PS1 initial calibrations.

PROCESSOR SUBSYSTEM 2 (PS2)

- Start PS2 ARM

Table 54 summarizes the PS2 related boot up APIs and PS2 JTAG enabling API.

Table 54. PS2 Related Boot Up APIs

PS2 Boot Up APIs	Description
adi_adrv910x_arm_Image_Write	Write PS1 or PS2 ARM image
adi_adrv910x_arm_Start	Start PS1 or PS2 ARM
adi_adrv910x_arm_ConfigJtag_ps2	Enable PS2 JTAG Interface

To understand the ADRV910x SoC boot up procedures in more details, please refer to the initialization code generated by the ADRV910x Transceiver Evaluation Software (TES).

PS2 Program and Data Memory Map

The ADRV910x silicon PS2 memory is extended to 928 KB of memory that is partitioned into 2 memory sections. All memory is partitioned into 16 KB banks. Memory addresses are assigned as described in Table 55.

Table 55. PS2 System Memory Sections in ADRV910x

RAM Bank	Start address	End Address
Section 1		
Memory BANK0	0x01000000	0x01003FFF
Memory BANK1	0x01004000	0x01007FFF
Memory BANK2	0x01008000	0x0100BFFF
...	...	...
Memory BANK31	0x0107C000	0x0107FFFF
Section 2		
Memory BANK32	0x20000000	0x20003FFF
Memory BANK33	0x20004000	0x20007FFF
Memory BANK34	0x20008000	0x2000BFFF
...	...	...
Memory BANK57	0x20064000	0x20067FFF

Each memory bank can be accessed in parallel. For example, HWA1 could read/write from Bank0 while HWA2 could read/write from Bank1. Similar for the arm core and DMAs. In this way multiple memory banks can be accessed in parallel, improving system efficiency.

Upon reset, ARM fetches “Boot Address” and “Stack Pointer” from address 0x00000000 and jumps to the program entry point specified by the “Boot Address”.

RESOURCES AND EXAMPLES

The scope of this user guide is to describe the functionality of the ADRV910x PS2. To aid with the development of systems that utilize the ADRV910x PS2, several additional resources are given.

ADRV910x Software Development Kit (SDK)

The SDK is the user's primary means of developing their radio system with the ADRV910x SoC. It is subdivided into three sections related to the relevant topic of development:

- Evaluation – Includes Transceiver Evaluation Software (TES) with PS2 Demo.
- Prototype – Includes support for setting up custom PS1 configuration, to allow creating full system prototypes with PS2.
- Production – Includes access to PS2 drivers with detailed descriptions and example code.

Transceiver Evaluation Software PS2 Demo

The Transceiver Evaluation Software (TES) is used for evaluating the functionality of the ADRV910x device and it includes PS2 demos. Some of the PS2 features that TES demos are:

- PS2 enablement and resource loading

PROCESSOR SUBSYSTEM 2 (PS2)

- ▶ Tx SSI to PS2 to Rx SSI loopback
- ▶ UART
- ▶ RF enablement by PS2
- ▶ GPIO / Timer
- ▶ Clock scaling for power saving
- ▶ Hardware accelerator FM signal conditioning demo

Further details of the TES PS2 demo and setup are included in the [PS2 Evaluation Through TES](#) section.

PS2 Driver Description

A detailed description of all the currently supported PS2 drivers is given in doxygen format as part of the SDK. The doxygen file for the PS2 is provided in the SDK at the location: **Unpacked CMSIS Pack*\ADRV910X PS2 FW Example\doc\html\index.html*.

To access a driver description, navigate through the modules and select the driver of interest. For example, to view the *adi_timer_open()* API, navigate to the timer section and select the API. For each API: a description of the function of the API, parameters description, and return value description will be given.

◆ **adi_timer_Open()**

```
adi_driver_Result_e adi_timer_Open ( uint8_t *      tid,
                                     const adi_timer_Params_t * timerParams
                                     )
```

Initializes timer instance.

Configures the timer peripheral with the clock source and prescale values passed and enables timer IRQ.

Parameters

[in] **tid** address to hold the Timer instance id that gets returned

[in] **timerParams** Pointer to timer config parameters passed to driver

Returns

Timer driver error code

Figure 160. Example Detailed Driver Description Given for *adi_timer_open()*

Example Code

Example code for all the PS2 functions is provided as part of the SDK. The example code gives a detailed description and example of using various features. This aims to speed up the evaluation and development process for the PS2 drivers.

The example code is provided at **Unpacked CMSIS Pack*\ADRV910X PS2 FW Example\examples\baremetal(or \freertos)\example_main.c* and examples are given for both 'baremetal' and 'freertos' cases. The example code can be run using CodeFusion Studio or IAR as described in the [PS2 Prototyping and Production Development](#) section.

A code snippet of the example code showing the watchdog timer usage is given below.

PROCESSOR SUBSYSTEM 2 (PS2)

```

void exampleWDTimerUsage()
{
    uint32_t timeout_val =
        (uint32_t)(5000000 * 192) / 32; /* timeout_val = 30000000 which corresponds to 5secs*/
    uint32_t remTime = timeout_val;
    TEST_PRINTF_MSG("Starting %s\r\n", __func__);

    /* Initialize WDTimer by setting Timeout value */
    adi_wdtimer_Open(timeout_val);

    /* If user wishes to handle anything on watchdog timer expiry,
       register a callback function to be called on watch dog time expiry */
    adi_wdtimer_CallbackRegister(watchdogCbKfn);

    /* Enable/Start Watchdog Counter */
    adi_wdtimer_Start();

    for (int i = 0; i < 10; i++)
    {
        /* Wait for around 4secs.(approximately)
           * The watchdog is not yet timeout, the user can restart the watchdog to prevent the interrupt to be generated. */
        do
        {
            remTime--;
        } while (remTime > 6000000u);

        /* Restart Watchdog Counter before it timed out */
        adi_wdtimer_Restart();
        remTime = timeout_val;
    }
    adi_wdtimer_Close();
    TEST_PRINTF_MSG("watchdog timer restarted 10 times\r\n");
}

```

Figure 161. Code snippet for watchdog timer usage from example code

TES PS2 Demo Usecase Code

The source code for the TES PS2 demo is also provided in the CMSIS pack. This includes all the PS2 features in the TES demo including the DMA driver demo.

The demo code is provided at ** Unpacked CMSIS Pack * \ADRV910X PS2 FW Example\examples\ usecases_baremetal(or \freertos)\ _main.c*.

This code is given as an **example only** and is not intended to be run or edited. This demo code helps to describe the functionality behind the TES PS2 demo. TES will automatically load this firmware when initialised, so no user interaction is needed.

A functional description of the TES PS2 demos are given in [TES PS2 Features](#).

PS2 SYSTEM CLOCK

The PS2 Clock is driven by either the ADRV910x device clock or the phase-locked loop (PLL) clocking. It runs at up to 200MHz when using the PLL. The ARM clock can be retrieved from ps2CpuClk in adi_sdk_common.h after the profile has been initialized.

Clock Scaling

A divider is provided to reduce the rate of the PS2 system clock. The default divider is 1 i.e. the system clock will run at the max rate of up to 200MHz. The range of the clock divider is 1:5. When the clock divider is set, the entire PS2 will run at the reduced rate.

All peripherals including the following I2C, UART, I2S, ToMem DMA, FromMem DMA, GP Timers and Watch-Dog Timer will run at the reduced rate. This must be considered in the overall system design.

Clock scaling can primarily be used in two scenarios:

1. **Constant divider:** The divider can be set to a constant value so the system will always be run at a reduced rate. This is useful to reduce the total power consumption as long as the reduce clock rate is sufficient for the application. The peripheral timings should be considered for this device clock.
2. **Power saving mode:** The PS2 can be dynamically put into a low power state by using the clock divider. In this low power mode, the PS2 can still operate at the reduce rate. Before entering the state, all peripherals should be closed. The peripherals can be re-opened after this function call again, if required.

When the PS2 system clock is scaled, the reference clock ([Reference Timer Access](#)) always remains constant and is never powered down or reduced.

Clock Scaling Examples and Resources

For detailed information on the clock scaling drivers, refer to the doxygen files as detailed in the [Resources and Examples](#) chapter.

PROCESSOR SUBSYSTEM 2 (PS2)

Examples for the clock scaling is given in the example code as detailed in the [Resources and Examples](#) chapter. Run this code example to see an example of closing the required peripherals and reducing the clock rate for a power saving example.

PS2 TIMERS

Introduction

ADRV910x PS2 includes four General Purpose (GP) Timers, one Watchdog Timer and one SysTick Timer, all controllable by ARM, as illustrated in [Figure 162](#).

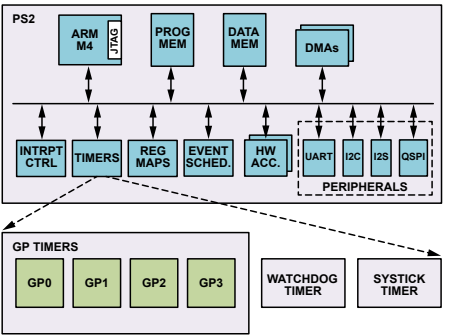


Figure 162. PS2 General Purpose Timers, Watchdog Timer and SysTick Timer

[Table 56](#) compares the major features and parameters of the GP Timers, Watchdog Timer and SysTick Timer. The counter type is different for different type of timers. GP Timers and SysTick Timer can count up while Watchdog Timer can count down. For each timer, a set of prescaler factors are provided to achieve different time resolutions. More details will be discussed in later sections.

Table 56. A Comparison of Major Features of PS2 GP Timers, WD Timer and SysTick Timer

Timer	Counter Resolution	Counter Type	PrescalerFactor	TimerSource Clock
GP Timer 0	32-bit	Up	x1, x4, x16, x256 or x32768	ARM clock
GP Timer 1	32-bit	Up	x1, x4, x16, x256 or x32768	ARM clock
GP Timer 2	32-bit	Up	x1, x4, x16, x256 or x32768	ARM clock
GP Timer 3	32-bit	Up	x1, x4, x16, x256 or x32768	ARM clock
Watchdog Timer	32-bit	Down	x32	ARM clock
SysTick Timer	8-bit	Up	x1024	REF clock

GP Timers

The goal of the GP Timer is to measure time by counting the elapsed number of ARM clock cycles and notify ARM through interrupt that the elapsed time has reached the timeout value. Those timers are based on a 32-bit resolution internal counter operating on the ARM clock.

Feature List

Below is a summary of features for GP Timers:

- ▶ A 32-bit resolution counter
- ▶ Operating mode: **One-Shot** or **Periodic**
- ▶ Count Direction: **Up**
- ▶ Timer time resolution: **prescaler** factor to increase the timeout time of the counter
- ▶ Timer timeout: **interrupt**
- ▶ Readback on-the-fly counter value

Operating Mode

The GP Timer has 2 operating modes:

- ▶ One shot mode: The timer counts the elapsed number of ARM clock cycles until the timeout value is reached, then it generates an interrupt to the ARM and stops.

PROCESSOR SUBSYSTEM 2 (PS2)

- ▶ Periodic mode: The counter is cyclic (loop). It counts the elapsed number of ARM clock cycle until the timeout value is reached then is reset to the start value and restart counting. At every end of loop (i.e: timeout reached), the timer generates an interrupt to the ARM.

Up Counter

The GP Timer has the capability to increment the counter value:

- ▶ The counter starts at 0 and get incremented until the timeout value is reached.

Time Resolution

The time resolution of the timer corresponds to the elapsed time it takes to increment/decrement the counter by 1. It is measured in number of ARM clock cycles and can be configured as a prescaler factor.

The available prescaler factors for GP Timers are:

- ▶ prescaler = x1: counter is incremented at every ARM clock cycle (default)
- ▶ prescaler = x4: counter is incremented at every 4 ARM clock cycles
- ▶ prescaler = x16: counter is incremented at every 16 ARM clock cycles
- ▶ prescaler = x256: counter is incremented at every 256 ARM clock cycles
- ▶ prescaler = x32768: counter is incremented at every 32768 ARM clock cycles

The time out value in us can be calculated by the following equation:

$$Timeout(us) = \frac{Timeout_value * prescaler}{Freq_{ARM_CLK}(MHz)} \quad (25)$$

Example:

- ▶ ARM Clock Freq = 192MHz
- ▶ prescaler = x16
- ▶ Timeout_value = 200

Then the timeout time in microsecond would be: $Timeout = (200 * 16) / 192 = 16.67\mu s$

Timeout Interrupt

A timeout event occurs when the timer reaches the maximum value programmed. This event causes the timer timeout flag to be set in the timer interrupt flag register and an interrupt to be asserted to the ARM. The ARM must clear the timer interrupt flag register.

Watchdog Timer

The goal of the Watchdog Timer is to reset the device when a problem occurs. A timeout interrupt will be sent to the ARM for the user to reset the device or initiate recovery action. It is up to ARM to restart the Watchdog counter to the initial value before it reaches timeout and generates an interrupt to ARM. Similar to the GP Timers, the Watchdog Timer is based on a 32-bit resolution internal counter operating on the ARM clock.

Feature List

Below is a summary of features for the watchdog timer:

- ▶ A 32-bit programmable downcounter
- ▶ Fixed x32 prescaler factor

Watchdog Timeout Interrupt

A timeout event occurs when the counter reaches 0. This event causes the generation of an interrupt to the ARM. Restart the Watchdog Timer before it reaches the timeout. The ARM must periodically reload the Watchdog Timer to the initial value so that timeout is never reached. By restarting the counter, it prevents the Watchdog Timer from sending an interrupt to the ARM to reset the chip.

PROCESSOR SUBSYSTEM 2 (PS2)

SysTick Timer

SysTick Timer is dedicated to real-time operating systems and is used to generate a clock pulse **every 1ms** to the ARM. Different from the GP Timer and Watchdog Timer, this timer is based on an 8-bit resolution internal counter operating on the REF_CLK clock.

Feature List

Below is a summary of features for the systick timer:

- ▶ An 8-bit resolution free running up counter
- ▶ Continuous mode (cannot be disabled)
- ▶ 8-bit programmable timeout value with a fix x1024 prescaler factor (equivalent to 18-bit timeout value)

Up Counter Resolution

This counter is operating in continuous mode: The counter will count to the 1ms timeout value, is reset to 0 and restarts counting. It can neither be stopped nor cleared.

The following formula is used to calculate the value to program to the counter to generate the 1ms system Tick clock provided to ARM:

$$value = \frac{1000 * Freq_{REF_CLK}(MHz)}{1024} \quad (26)$$

Reference Timer Access

The Reference Timer is a free running 32-bit counter that is part of the Monitor block (as shown in [Figure 157](#)) and is in the always ON domain. It has 4 unique events with respective interrupt masks that can be used to trigger the reference timer interrupt, which are routed to PS1/PS2. It also allows us to increment or decrement the current counter value on the fly using an 8-bit delta value, which can be used for synchronization purposes.

The main purpose of the reference timer is that when PS1 and/or PS2 is in power down mode, the reference timer always stays on. This allows to keep a timer always on and recover the timing after leaving power-down modes. This is especially used for keeping track of frame timing in power down modes.

Timer Examples and Resources

For detailed information on the timer drivers, refer to the doxygen files as detailed in the [Resources and Examples](#) chapter.

Examples for each timer is given in the example code as detailed in the [Resources and Examples](#) chapter. Run this code example to see an example of how each of the timers can be used and called including watchdog/GP/systick timer creation and callbacks.

DATA PATH DIRECT MEMORY ACCESS (DMA)

This section covers the various direct memory access (DMA) PS2 uses to interface with data path and SSI Interfaces, as shown in [Figure 163](#).

PROCESSOR SUBSYSTEM 2 (PS2)

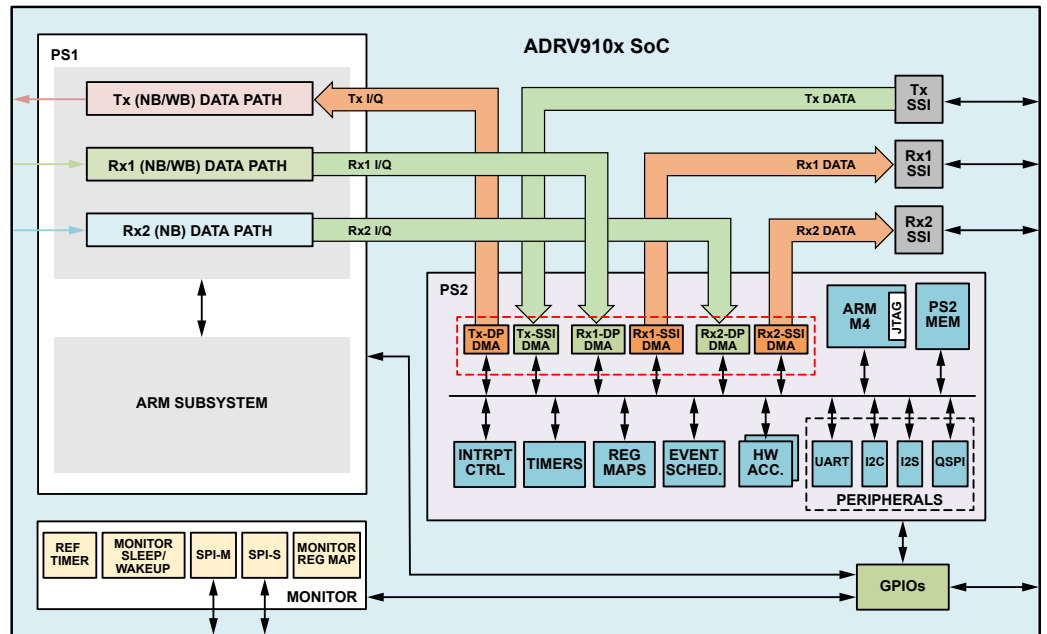


Figure 163. Data Path DMA in ADRV910x PS2

DMA's that transfer data from memory to the SSI or data path interfaces are named "From-Mem DMA". They are highlighted in orange in [Figure 163](#) and include:

- ▶ Tx-DP DMA
- ▶ Rx1-SSI DMA
- ▶ Rx2-SSI DMA

DMA's that transfer data from the SSI or data path interfaces to memory are named "To-Mem DMA". They are highlighted in green in [Figure 163](#) and include:

- ▶ Tx-SSI DMA
- ▶ Rx1-DP DMA
- ▶ Rx2-DP DMA

From-Mem DMA: From Memory to DataPath or RxSSI

The From-Mem DMA function is used to transfer data from memory to the Tx data path or to the RX-SSI interface as illustrated:

PROCESSOR SUBSYSTEM 2 (PS2)

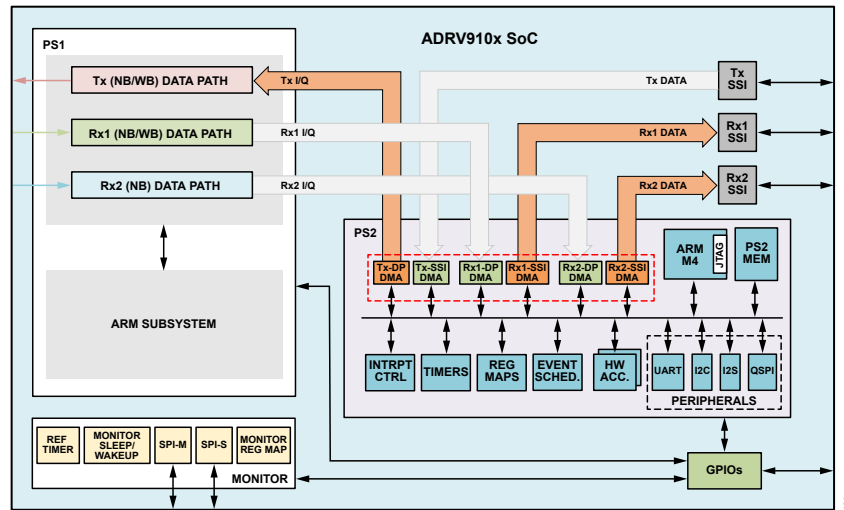


Figure 164. From-Mem DMAs in ADRV910x PS2

In this case, two From-Mem DMAs are used to transfer data from memory to Rx1 and Rx2 SSI, and one From-Mem DMA is used to transfer data from memory to Tx Data Path input of PS1.

From-Mem DMA Using One Buffer Mode

Figure 165 shows the block diagram of From-Mem DMA using One Buffer mode.

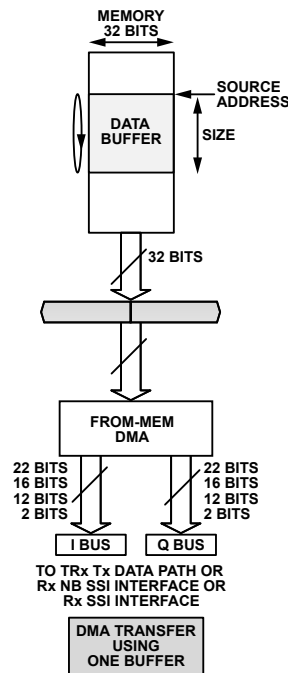


Figure 165. From-Mem DMA Using a Single Buffer Source

When using One Buffer mode, the From-Mem DMA can be configured to fetch data from memory in two transfer modes:

- “Single” data transfer mode: starting from a source address to the end address after N data words. After sending one buffer of data, the From-Mem DMA stops.
- “Circular” data transfer mode: starting from a source address to the end address after N data words, then automatically restarting and jumping back to the start source address for the next transfer.

PROCESSOR SUBSYSTEM 2 (PS2)

When using One Buffer mode to store data, I and Q data can be stored differently as follows:

- ▶ One data buffer with packed I & Q in one word: Uses I address only
- ▶ One data buffer for I and another data buffer for Q: Uses I address and Q address
- ▶ Interleaved I and Q in one data buffer: Uses I address and Q address

Various data formats (22, 16, 12 and 2 bits) are supported on the SSI and data path interfaces. The data formatting is done by the DMA.

From-Mem DMA Using Two Buffer Mode

Figure 166 shows the block diagram of From-Mem DMA using Two Buffer (ping-pong) mode.

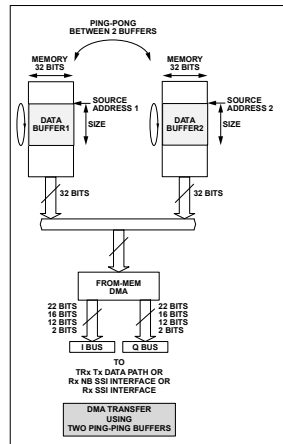


Figure 166. From-Mem DMA Using Two Ping-Pong Buffer Sources

When using Two Buffer mode in a ping-pong scheme, the From-Mem DMA can be configured to fetch N data words from one data buffer, then switches to the second data buffer to fetch the next N data words. In this case, two source addresses of data buffers are used to read the data memory, as shown in Figure 166.

From-Mem DMA Feature List

Data Buffer Source: Memory Data Format

The data in memory is fetched by From-Mem DMA to drive data bus destination (the Tx data path or the Rx1/Rx2 SSI interfaces). The data type in memory can be:

- ▶ Real data samples (I data Only) or
- ▶ Complex data samples (I data, Q data)

These data samples are stored in the system memories in 32 bits words.

The From-Mem DMA reads data samples from memory and drives the data bus destination as the following:

- ▶ Supported data formats are: 2bits, 12bits, 16bits or 22bits for both real data samples and complex data samples.
- ▶ All data samples are LSB aligned on the data bus destination.
- ▶ Programmable default value on the data bus when DMA is disabled.

The data samples can be packed or not in memory. A packed mode is when multiple data samples of I or Q, or both I and Q are present in a 32-bit word in memory.

The following formats are supported:

- ▶ **Non-Packed I/Q mode:** I and Q samples are stored in different memory location
 - ▶ LSB align,

PROCESSOR SUBSYSTEM 2 (PS2)

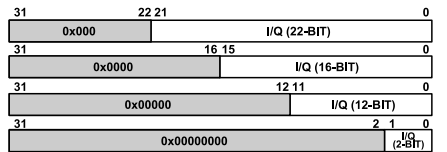


Figure 167. Non-packed I/Q Mode with LSB Aligned

- LSB aligned with sign extended,

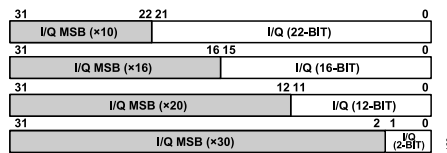


Figure 168. Non-packed I/Q Mode with LSB Aligned and Sign Extended

- MSB aligned.

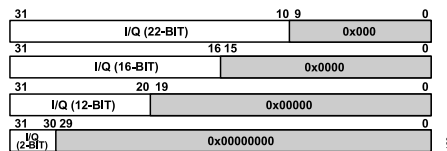


Figure 169. Non-packed I/Q Mode with MSB Aligned

- **Packed I/Q Mode:** I and Q samples can be packed in the same 32 bits word in memory, examples:

Note:

- Not supported for 22 bits data format
- For 16 bits I and Q:

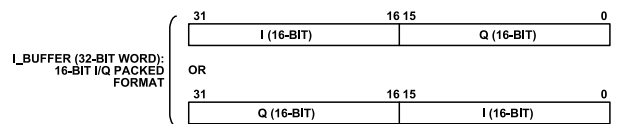


Figure 170. Packed I/Q Mode for 16 bits I and Q

- For 12 bits I and Q:

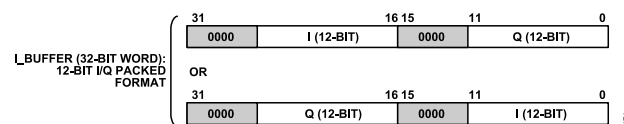


Figure 171. Packed I/Q Mode for 12 bits I and Q

- For 2 bits I and Q:

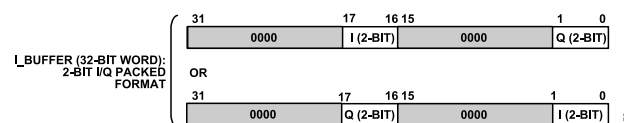


Figure 172. Packed I/Q Mode for 2 bits I and Q

PROCESSOR SUBSYSTEM 2 (PS2)

- In packed I/Q mode, the I & Q order is configurable and only 1 sample of I and Q can be packed into one 32-bit word.
- For the 16-bit packed I-Q data, the default data arranged in memory as follows: {Idata[31:16], QData[15:0]}
 - This default data arrangement can be altered to { Qdata[31:16], Idata[15:0]} by updating “*complexOneWordFlip* = 1”.
- **Packed I and Packed Q Mode:** a number of I samples are packed in 32 bits word and a number of Q samples are packed in one 32 bits word:
 - For 22 bits I and Q: Format NOT Supported
 - For 16 bits I and Q: (2 samples per 32-bit word)

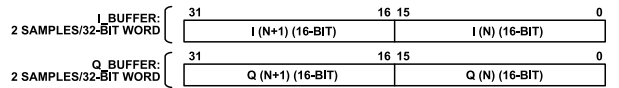


Figure 173. Packed I and Packed Q Mode for 16 bits I and Q

- For 12 bits I and Q: (2 samples per 32-bit word)

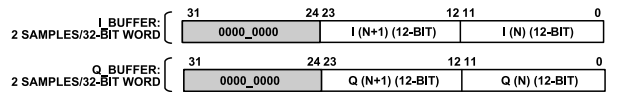


Figure 174. Packed I and Packed Q Mode for 12 bits I and Q

- For 2 bits I and Q: (16 samples per 32-bit word)

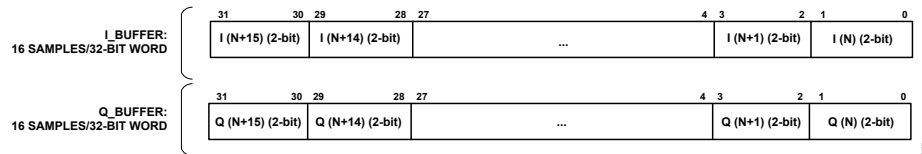


Figure 175. Packed I and Packed Q Mode for 2 bits I and Q

Table 57 summarizes the various supported data sample formats in memory:

Table 57. Supported Data Sample Formats in Memory for From-Mem DMA

Sample Type	Sample Size	Sample Packing	Sample Alignment	Description
Complex: I and Q	I:22bits, Q:22bits	Non-Packed	Word 32bits: MSB_10bits = 0, LSB_22bits = I Word 32bits: MSB_10bits = 0, LSB_22bits = Q or Word 32bits: MSB_22bits = I, LSB_10bits = 0 Word 32bits: MSB_22bits = Q, LSB_10bits = 0	I and Q are in different memory words.
	I:16bits, Q:16bits	Packed	Word 32bits: MSB_16bits = I, LSB_16bits = Q or Word 32bits: MSB_16bits = Q, LSB_16bits = I	I and Q are in same word 32bits.
	I:16bits, Q:16bits	Packed	Word 32bits: MSB_16bits = I(N+1), LSB_16bits = I(N) Word 32bits: MSB_16bits = Q(N+1), LSB_16bits = Q(N)	I and Q are in different memory words
	I:16bits, Q:16bits	Non-Packed	Word 32bits: MSB_16bits = 0, LSB_16bits = I Word 32bits: MSB_16bits = 0, LSB_16bits = Q or Word 32bits: MSB_16bits = I, LSB_16bits = 0 Word 32bits: MSB_16bits = Q, LSB_16bits = 0	I and Q are in separate words.
	I:12bits, Q:12bits	Packed	Word 32bits: MSB_4bits = 0, MSB_12bits = I, LSB_4bits = 0, LSB_12bits = Q or Word 32bits: MSB_4bits = 0, MSB_12bits = Q, LSB_4bits = 0, LSB_12bits = I	I and Q are in same word 32bits.
	I:12bits, Q:12bits	Packed	Word 32bits: MSB_8bits = 0, MSB_12bits = I(N+1), LSB_12bits = I(N)	I and Q are in different memory words

PROCESSOR SUBSYSTEM 2 (PS2)

Table 57. Supported Data Sample Formats in Memory for From-Mem DMA (Continued)

Sample Type	Sample Size	Sample Packing	Sample Alignment	Description
			Word 32bits: MSB_8bits = 0, MSB_12bits = Q(N+1), LSB_12bits = Q(N)	
	I:12bits, Q:12bits	Non-Packed	Word 32bits: MSB_20bits = 0, LSB_12bits = I Word 32bits: MSB_20bits = 0, LSB_12bits = Q or Word 32bits: MSB_12bits = I, LSB_20bits = 0 Word 32bits: MSB_12bits = Q, LSB_20bits = 0	I and Q are in separate words of 32bits.
	I:2bits, Q:2bits	Packed	Word 32bits: MSB_14bits = 0, MSB_2bits = I, LSB_14bits = 0, LSB_2bits = Q or Word 32bits: MSB_14bits = 0, MSB_2bits = Q, LSB_14bits = 0, LSB_2bits = I	I and Q are in same word 32bits.
	I:2bits, Q:2bits	Packed	Word 32bits: 16 x 2-bit consecutive I samples (starting on LSB bits) Word 32bits: 16 x 2-bit consecutive Q samples (starting on LSB bits)	I and Q are in different memory words
	I:2bits, Q:2bits	Non-Packed	Word 32bits: MSB_30bits = 0, LSB_2bits = I Word 32bits: MSB_30bits = 0, LSB_2bits = Q or Word 32bits: MSB_2bits = I, LSB_30bits = 0 Word 32bits: MSB_2bits = Q, LSB_30bits = 0	I and Q are in separate words of 32bits.
Real: I-Only	I:22bits	Non-Packed	Word 32bits: MSB_10bits = 0, LSB_22bits = I or Word 32bits: MSB_22bits = I, LSB_10bits = 0	I samples are in separate words.
	I:16bits	Packed	Word 32bits: MSB_16bits = I(N+1), LSB_16bits = I(N)	First sample is in LSB. I samples are packed in one word. Packed I-only does not support MSB/LSB samples flip First sample is always LSB aligned
	I:16bits	Non-Packed	Word 32bits: MSB_16bits = 0, LSB_16bits = I0 (LSB aligned) Or Word 32bits: MSB_16bits = I0, LSB_16bits = 0 (MSB aligned)	I samples are in separate words. Also support LSB aligned with sign extension
	I:12bits	Packed	Word 32bits: MSB_8bits = 0, MSB_12bits = I(N+1), LSB_12bits = I(N)	I samples are packed in one word. Packed I-only does not support MSB/LSB samples flip. First sample is always LSB aligned
	I:12bits	Non-Packed	Word 32bits: MSB_20bits = 0, LSB_12bits = I0 Or Word 32bits: MSB_12bits = I0, LSB_20bits = 0	I samples are in separate words. Also support LSB aligned with sign extension
	I:2bits	Packed	Word 32bits: 16 x 2-bit consecutive I samples (starting on LSB bits) ► Word 32bits: I15, ..., I1, I0	I samples are 2bits each and are packed in one word. Packed I-only does not support MSB/LSB samples flip. First sample is always LSB aligned
	I:2bits	Non-Packed	Word 32bits: MSB_30bits = 0, LSB_2bits = I0 Or Word 32bits: MSB_2bits = I0, LSB_30bits = 0	I samples are 2bits each and are in separate words. Also support LSB aligned with sign extension

I and Q can be in the same memory buffer or in different memory buffer. Their corresponding addresses can be independently configured, which means they can be the same or different address values.

Data Buffer Addressing

There are different addressing schemes supported by the DMA:

- Separate addresses for I buffer and Q buffer. If I and Q are packed in the same word, I buffer is used to store the word.
- Supports circular addressing: when end of buffer address is reached, the next address goes back to buffer start address.
- Step address increment by byte for I and Q: Useful when I and Q are interleaved in one buffer (Example: address increment by 1, 2 or m bytes).
- Alternate addressing for I and Q: to support ping-pong scheme.

PROCESSOR SUBSYSTEM 2 (PS2)

Data Transfer: Block Transfer of N Data Words

In the From-Mem DMA, a **block transfer** is defined by the transfer of M (I, Q) samples from data buffers located in system memories to the data bus interfaces (either SSI or data path interface).

The DMA Block transfer size is configurable by ARM and supports up to 65536 (I, Q) samples.

$M = \text{Block transfer size register width} = 16\text{bits}$ (Maximum size = $2^{16} = 65536$)

A full DMA transfer is defined by a succession of blocks transfer. The following DMA transfer type are supported:

Single block of data transfer:

In single block transfer, the DMA completes after one block is transferred. No method is provided to stop before finishing the one block transfer.

N blocks of data transfer:

In N block transfer mode, the DMA completes after N blocks are transferred. The transfer can only be aborted at the end of the current block transfer (to prevent from stopping in the middle of the block transfer).

Continuous blocks of data transfer:

In continuous block transfer, the DMA completes at the end of the current block transfer when a stop signal is activated.

DMA Control

The From-Mem DMA can be started and stopped using corresponding functions in the driver.

- ▶ Start DMA using driver
- ▶ Stop DMA using driver

The From-Mem DMA has the capability to generate interrupts to ARM:

- ▶ End of each buffer transfer interrupt
- ▶ End of DMA transfer interrupt
- ▶ DMA error

From-Mem DMA: Configuration and Control Procedures

This section describes the procedure to properly configure and control the From-Mem DMA:

- ▶ Configure DMA (data format, transfer type, memory data buffers, interrupts to PS2 ARM processor and etc.)
- ▶ Open DMA
- ▶ Register DMA callback function if needed
- ▶ Start DMA
- ▶ Stop DMA (depending on profile selected)
- ▶ Unregister DMA callback function if registered
- ▶ Close DMA

Configure From-Mem DMA Mode of Operation

The mode of operation of the From-Mem DMA is defined by the following arguments:

Table 58. From-Mem DMA Mode of Operation Definitions

Argument	Description
Data Type	Data type can be: ▶ Complex (I & Q data): using 2 buffers (I buffer and Q buffer) ▶ Real (I data only): using I only buffer
Data Width	Data width should be compliant with the data bus interfaces it is connecting with. SSI and Datapath interfaces are supporting: ▶ 22 bits data

PROCESSOR SUBSYSTEM 2 (PS2)

Table 58. From-Mem DMA Mode of Operation Definitions (Continued)

Argument	Description
	▶ 16 bits data ▶ 12 bits data ▶ 2 bits data
Transfer Type	Transfer type defines the mode of operation of the DMA: ▶ Single block transfer (One Buffer mode) ▶ N block transfer using single buffer source (One Buffer mode) ▶ Continuous block transfer using single buffer source (One Buffer mode) ▶ N block transfer using alternate buffers sources (Two Buffer ping-pong mode) ▶ Continuous block transfer using alternate buffers sources (Two Buffer Ping-Pong mode)
Packing Mode	Multiple data can be packed in the same 32bits word: ▶ IQ Packed mode: 1 sample of I and Q data are packed in the same 32-bit I buffer ▶ I packed; Q packed: Several samples of I are packed in 32-bit I buffer, several samples of Q are packed in 32-bit Q buffer ▶ Number of data per word: 2*16-bit or 2*12-bit or N*2-bit (N=2,3,...16)

Table 59 describes all the valid modes of operation (profiles) of the DMA.

All DMA profiles marked as “pack mode supported” can use the **packed I and packed Q format** where:

- ▶ Several I samples are packed into one 32-bit word from I buffer,
- and
- ▶ Several Q samples are packed into one 32-bit word from Q buffer.

Table 59. Valid Profiles of From-Mem DMA

Idx	DMA Profile Name	Pack Mode Supported	Data Type	Data Width	Transfer Type	Buffer Number
1	Real – 2 bits sample – SINGLE block transfer	yes	real	2	single	1 buffer (I-Only)
2	Real – 12 bits sample – SINGLE block transfer	yes	real	12	single	1 buffer (I-Only)
3	Real – 16 bits sample – SINGLE block transfer	yes	real	16	single	1 buffer (I-Only)
4	Real – 22 bits sample – SINGLE block transfer	no	real	22	single	1 buffer (I-Only)
5	Complex – 2 bits sample – SINGLE block transfer	yes	complex	2	single	2 buffers (I buffer & Q buffer)
6	Complex – 12 bits sample – SINGLE block transfer	yes	complex	12	single	2 buffers (I buffer & Q buffer)
7	Complex – 16 bits sample – SINGLE block transfer	yes	complex	16	single	2 buffers (I buffer & Q buffer)
8	Complex – 22 bits sample – SINGLE block transfer	no	complex	22	single	2 buffers (I buffer & Q buffer)
9	Complex IQ Packed – 2 bits sample – SINGLE block transfer	IQ packed	complex	2	single	1 buffer (I & Q buffer)
10	Complex IQ Packed – 12 bits sample – SINGLE block transfer	IQ packed	complex	12	single	1 buffer (I & Q buffer)
11	Complex IQ Packed – 16 bits sample – SINGLE block transfer	IQ packed	complex	16	single	1 buffer (I & Q buffer)
12	Real – 2 bits sample – N blocks transfer	yes	real	2	N Count	1 buffer (I-Only)
13	Real – 12 bits sample – N blocks transfer	yes	real	12	N Count	1 buffer (I-Only)
14	Real – 16 bits sample – N blocks transfer	yes	real	16	N Count	1 buffer (I-Only)
15	Real – 22 bits sample – N blocks transfer	no	real	22	N Count	1 buffer (I-Only)
16	Complex – 2 bits sample – N blocks transfer	yes	complex	2	N Count	2 buffers (I buffer & Q buffer)
17	Complex – 12 bits sample – N blocks transfer	yes	complex	12	N Count	2 buffers (I buffer & Q buffer)
18	Complex – 16 bits sample – N blocks transfer	yes	complex	16	N Count	2 buffers (I buffer & Q buffer)
19	Complex – 22 bits sample – N blocks transfer	no	complex	22	N Count	2 buffers (I buffer & Q buffer)

PROCESSOR SUBSYSTEM 2 (PS2)

Table 59. Valid Profiles of From-Mem DMA (Continued)

Idx	DMA Profile Name	Pack Mode Supported	Data Type	Data Width	Transfer Type	Buffer Number
20	Complex IQ Packed – 2 bits sample – N blocks transfer	IQ packed	complex	2	N Count	1 buffer (I & Q buffer)
21	Complex IQ Packed – 12 bits sample – N blocks transfer	IQ packed	complex	12	N Count	1 buffer (I & Q buffer)
22	Complex IQ Packed – 16 bits sample – N blocks transfer	IQ packed	complex	16	N Count	1 buffer (I & Q buffer)
23	Real – 2 bits sample – Continuous block transfer	yes	real	2	Continuous	1 buffer (I-Only)
24	Real – 12 bits sample – Continuous block transfer	yes	real	12	Continuous	1 buffer (I-Only)
25	Real – 16 bits sample – Continuous block transfer	yes	real	16	Continuous	1 buffer (I-Only)
26	Real – 22 bits sample – Continuous block transfer	no	real	22	Continuous	1 buffer (I-Only)
27	Complex – 2 bits sample – Continuous block transfer	yes	complex	2	Continuous	2 buffers (I buffer & Q buffer)
28	Complex – 12 bits sample – Continuous block transfer	yes	complex	12	Continuous	2 buffers (I buffer & Q buffer)
29	Complex – 16 bits sample – Continuous block transfer	yes	complex	16	Continuous	2 buffers (I buffer & Q buffer)
30	Complex – 22 bits sample – Continuous block transfer	no	complex	22	Continuous	2 buffers (I buffer & Q buffer)
31	Complex IQ Packed – 2 bits sample – Continuous block transfer	IQ packed	complex	2	Continuous	1 buffer (I & Q buffer)
32	Complex IQ Packed – 12 bits sample – Continuous block transfer	IQ packed	complex	12	Continuous	1 buffer (I & Q buffer)
33	Complex IQ Packed – 16 bits sample – Continuous block transfer	IQ packed	complex	16	Continuous	1 buffer (I & Q buffer)
34	Real – 2 bits sample – N blocks transfer	yes	real	2	N Count Alternate	1 buffer (I-Only)
35	Real – 12 bits sample – N blocks transfer	yes	real	12	N Count Alternate	1 buffer (I-Only)
36	Real – 16 bits sample – N blocks transfer	yes	real	16	N Count Alternate	1 buffer (I-Only)
37	Real – 22 bits sample – N blocks transfer	no	real	22	N Count Alternate	1 buffer (I-Only)
38	Complex – 2 bits sample – N blocks transfer	yes	complex	2	N Count Alternate	2 buffers (I buffer & Q buffer)
39	Complex – 12 bits sample – N blocks transfer	yes	complex	12	N Count Alternate	2 buffers (I buffer & Q buffer)
40	Complex – 16 bits sample – N blocks transfer	yes	complex	16	N Count Alternate	2 buffers (I buffer & Q buffer)
41	Complex – 22 bits sample – N blocks transfer	no	complex	22	N Count Alternate	2 buffers (I buffer & Q buffer)
42	Complex IQ Packed – 2 bits sample – N blocks transfer	IQ packed	complex	2	N Count Alternate	1 buffer (I & Q buffer)
43	Complex IQ Packed – 12 bits sample – N blocks transfer	IQ packed	complex	12	N Count Alternate	1 buffer (I & Q buffer)
44	Complex IQ Packed – 16 bits sample – N blocks transfer	IQ packed	complex	16	N Count Alternate	1 buffer (I & Q buffer)
45	Real – 2 bits sample – Continuous block transfer	yes	real	2	Continuous Alternate	1 buffer (I-Only)
46	Real – 12 bits sample – Continuous block transfer	yes	real	12	Continuous Alternate	1 buffer (I-Only)
47	Real – 16 bits sample – Continuous block transfer	yes	real	16	Continuous Alternate	1 buffer (I-Only)
48	Real – 22 bits sample – Continuous block transfer	no	real	22	Continuous Alternate	1 buffer (I-Only)
49	Complex – 2 bits sample – Continuous block transfer	yes	complex	2	Continuous Alternate	2 buffers (I buffer & Q buffer)
50	Complex – 12 bits sample – Continuous block transfer	yes	complex	12	Continuous Alternate	2 buffers (I buffer & Q buffer)
51	Complex – 16 bits sample – Continuous block transfer	yes	complex	16	Continuous Alternate	2 buffers (I buffer & Q buffer)
52	Complex – 22 bits sample – Continuous block transfer	no	complex	22	Continuous Alternate	2 buffers (I buffer & Q buffer)
53	Complex IQ Packed – 2 bits sample – Continuous block transfer	IQ packed	complex	2	Continuous Alternate	1 buffer (I & Q buffer)
54	Complex IQ Packed – 12 bits sample – Continuous block transfer	IQ packed	complex	12	Continuous Alternate	1 buffer (I & Q buffer)

PROCESSOR SUBSYSTEM 2 (PS2)

Table 59. Valid Profiles of From-Mem DMA (Continued)

Idx	DMA Profile Name	Pack Mode Supported	Data Type	Data Width	Transfer Type	Buffer Number
55	Complex IQ Packed – 16 bits sample – Continuous block transfer	IQ packed	complex	16	Continuous Alternate	1 buffer (I & Q buffer)

Configure From-Mem DMA Memory Data Buffers

To fetch data (I/Q samples) from memories, the From-Mem DMA must know the location and size of the data buffers. The following information must be provided to the DMA to complete the configuration of the DMA:

Applicable to all profiles:

Table 60. From-Mem DMA Memory Data Buffer Configurations Applicable to All Profiles

Configuration Parameters	Explanations
Main I data buffer Start address	32 bits start address located in PS2 memories (32-bit aligned).
Main Q data buffer Start address	32 bits start address located in PS2 memories (32-bit aligned) unused if real data type used or if 1 complex buffer used (I&Q same word)
Number of Data Words	Size of the block transfer for Idata & Qdata buffer (16 bits) Applicable for both main and alternate buffers
I data buffer address increment	Address increment to fetch next Idata in memory (Step Size) (16 bits)
Q data buffer address increment	Address increment to fetch next Qdata in memory (Step Size) (16 bits)

Applicable to alternate transfer type profile only:

Table 61. From-Mem DMA Memory Data Buffer Configurations Applicable to Alternate Transfer Type Profile Only

Configuration Parameters	Explanations
Alternate I data buffer start address	32 bits start address located in PS2 memories (32-bit aligned).
Alternate Q data buffer start address	32 bits start address located in PS2 memories (32-bit aligned).

Configure From-Mem DMA Interrupts

A DMA interrupt is connected to PS2 ARM processor and can be enabled by ARM to control and monitor the block transfer of the From-Mem DMA.

Three interrupts status bits are used to monitor the processing of the block transfer:

Table 62. From-Mem DMA Interrupts

Interrupt name	Description
DMA end of transfer of all blocks	This interrupt is triggered when the DMA execution is done. ▶ In continuous mode, it happens when ARM ends the DMA. ▶ In Single transfer or N block transfer, it happens when all block transfers are done.
DMA end of current block transfer	Notify that the current block transfer finished. For alternate buffer transfer mode, it can be used by ARM to update the unused buffer with new data for next block transfer execution while the other buffer is being fetched by the DMA.
DMA error	(For debug purpose) Error:It signifies that the DMA could not fetch data from the memory location. Check the memory buffer address accessible to the DMA.

Start/Stop From-Mem DMA

From-Mem DMA can be started by PS2 ARM using the start function provided by the driver.

From-Mem DMA can be stopped by:

- ▶ In “Single” mode: will be ended automatically once the last transfer is done.
- ▶ In “N block transfer” mode: will be ended automatically once the last transfer is done or PS2 ARM sending a stop command to the DMA using stop function provided by the driver.
- ▶ In “Continuous” mode: PS2 ARM can send a stop command to the DMA using stop function provided by the driver.

PROCESSOR SUBSYSTEM 2 (PS2)

To-Mem DMA: From Rx Data Path or Tx SSI to Memory

The same To-Mem DMA is used to transfer data from Rx Datapath or Tx-SSI interface to Memory as shown in Figure 176:

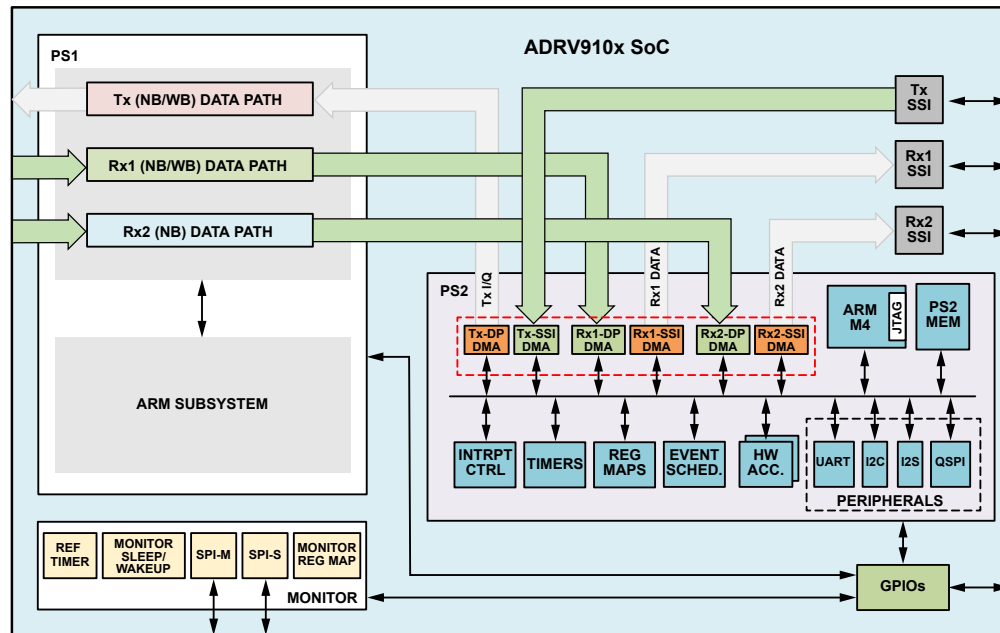


Figure 176. To-Mem DMA in ADRV910x PS2

In this case, two To-Mem DMA are used to transfer data from Rx1 and Rx2 Data Path to memory, and one To-Mem DMA is used to transfer data from Tx SSI Interfaces to memory.

To-Mem DMA Using One Buffer Mode

When using One data buffer, the To-Mem DMA can be configured to store data to memory in two transfer modes:

- In a “Single” data transfer mode: starting from a source address to the end address after transferring N data words to fill in the destination buffer. After filling a single buffer, the DMA stops.
- In a “Circular” data transfer mode: starting from a source address to the end address after N data words, then automatically restarting and jumping back to the start destination address for the next transfer.

As shown in the following diagram:

PROCESSOR SUBSYSTEM 2 (PS2)

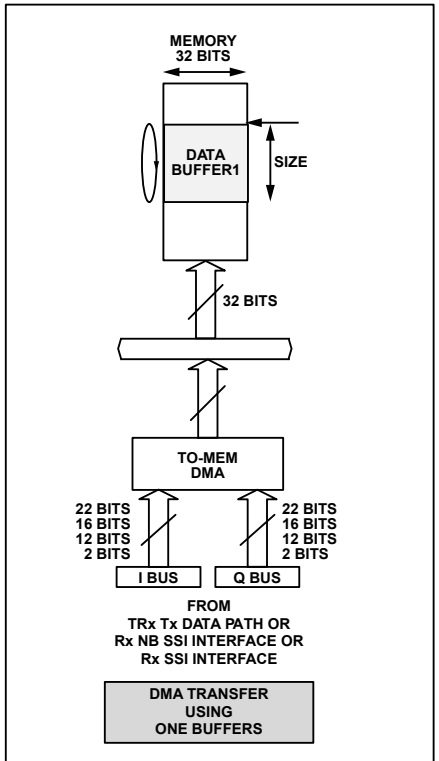


Figure 177. To-Mem DMA Using a Single Buffer Source

When using One buffer mode to store data, I and Q data can be stored differently as follows:

- One data buffer with packed I & Q: Uses I Address only
- One data buffer for I and another data buffer for Q: Uses I address and Q address
- Interleaved I and Q in one data buffer: Uses I address and Q address

Various data formats (22, 16, 12 and 2 bits) are supported on the SSI and data path interfaces. The data formatting is done by the DMA.

To-Mem DMA Using Two Buffer Mode

When using a Two Buffers mode in a Ping-pong scheme, the To-Mem DMA can be configured to fetch N data words from a data source (data path or SSI) to store the in one buffer, then switches to the second data buffer to store the next N data words.

In this case, two destination addresses of data buffers are used to write the data memory, as shown in the following diagram:

PROCESSOR SUBSYSTEM 2 (PS2)

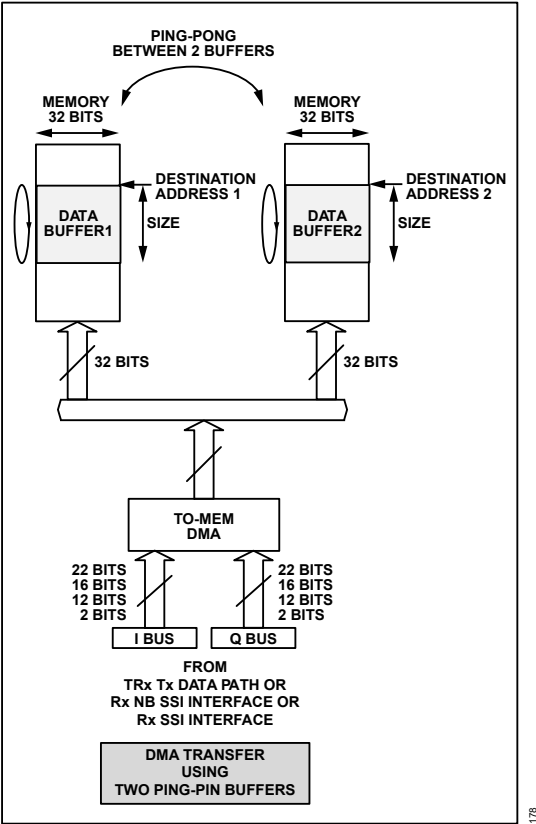


Figure 178. To-Mem DMA Using Two Ping-Pong Buffer Sources

To-Mem DMA Feature List

The feature list of To-Mem DMA is similar to From-Mem DMA. Please refer to [From-Mem DMA Feature List](#) section.

To-MEM DMA Configuration Procedure

This section describes the procedure to properly configure and control the To-Mem DMA:

1. Configure DMA (data format, transfer type, memory data buffers, interrupts to PS2 ARM processor and etc.)
2. Open DMA
3. Register DMA callback function if needed
4. Start DMA
5. Stop DMA (depending on profile selected),
6. Unregister DMA callback function if registered
7. Close DMA

Configure To-Mem DMA Mode of Operation

The mode of operation of the To-Mem DMA is defined by the following arguments.

Table 63. To-Mem DMA Mode of Operation Definitions

Argument	Description
Data Type	Data type can be: ► Complex (I & Q data): using 2 buffers (I buffer and Q buffer) ► Real (I data only): using I only buffer
Data width	Data width should be compliant with the data bus interfaces it is connecting with. SSI and Datapath interfaces are supporting:

PROCESSOR SUBSYSTEM 2 (PS2)

Table 63. To-Mem DMA Mode of Operation Definitions (Continued)

Argument	Description
	▶ 22 bits data ▶ 16 bits data ▶ 12 bits data ▶ 2 bits data
Transfer Type	Transfer type defines the mode of operation of the DMA: ▶ Single block transfer (using 1 buffer) ▶ N block transfer using single buffer source ▶ Continuous block transfer using single buffer source ▶ N block transfer using alternate buffers sources (Ping-Pong mode) ▶ Continuous block transfer using alternate buffers sources (Ping-Pong mode)
Packing Mode	Multiple data can be packed in the same 32bits word: ▶ IQ Packed mode: 1 sample of I and Q data are packed in the same 32-bit I buffer ▶ I packed; Q packed: Several samples of I are packed in 32-bit I buffer, several samples of Q are packed in 32-bit Q buffer ▶ Number of data per word: 2*16-bit or 2*12-bit or N*2-bit (N=2,3,...16)

Table 64 describes all the valid mode of operation of the DMA.

All DMA profiles marked as “pack mode supported” can use the **packed I and packed Q format** where several I samples are packed into 32-bit word from I buffer and several Q samples are packed into 32-bit word from Q buffer.

Table 64. Valid Profiles of To-Mem DMA

Idx	DMA Profile Name	Pack Mode Supported	Data Type	Data Width	Transfer Type	Buffer Number
1	Real – 2 bits sample – SINGLE block transfer	yes	real	2	single	1 buffer (I-Only)
2	Real – 12 bits sample – SINGLE block transfer	yes	real	12	single	1 buffer (I-Only)
3	Real – 16 bits sample – SINGLE block transfer	yes	real	16	single	1 buffer (I-Only)
4	Real – 22 bits sample – SINGLE block transfer	no	real	22	single	1 buffer (I-Only)
5	Complex – 2 bits sample – SINGLE block transfer	yes	complex	2	single	2 buffers (I buffer & Q buffer)
6	Complex – 12 bits sample – SINGLE block transfer	yes	complex	12	single	2 buffers (I buffer & Q buffer)
7	Complex – 16 bits sample – SINGLE block transfer	yes	complex	16	single	2 buffers (I buffer & Q buffer)
8	Complex – 22 bits sample – SINGLE block transfer	no	complex	22	single	2 buffers (I buffer & Q buffer)
9	Complex IQ Packed – 2 bits sample – SINGLE block transfer	IQ packed	complex	2	single	1 buffer (I & Q buffer)
10	Complex IQ Packed – 12 bits sample – SINGLE block transfer	IQ packed	complex	12	single	1 buffer (I & Q buffer)
11	Complex IQ Packed – 16 bits sample – SINGLE block transfer	IQ packed	complex	16	single	1 buffer (I & Q buffer)
12	Real – 2 bits sample – N blocks transfer	yes	real	2	N Count	1 buffer (I-Only)
13	Real – 12 bits sample – N blocks transfer	yes	real	12	N Count	1 buffer (I-Only)
14	Real – 16 bits sample – N blocks transfer	yes	real	16	N Count	1 buffer (I-Only)
15	Real – 22 bits sample – N blocks transfer	no	real	22	N Count	1 buffer (I-Only)
16	Complex – 2 bits sample – N blocks transfer	yes	complex	2	N Count	2 buffers (I buffer & Q buffer)
17	Complex – 12 bits sample – N blocks transfer	yes	complex	12	N Count	2 buffers (I buffer & Q buffer)
18	Complex – 16 bits sample – N blocks transfer	yes	complex	16	N Count	2 buffers (I buffer & Q buffer)
19	Complex – 22 bits sample – N blocks transfer	no	complex	22	N Count	2 buffers (I buffer & Q buffer)
20	Complex IQ Packed – 2 bits sample – N blocks transfer	IQ packed	complex	2	N Count	1 buffer (I & Q buffer)
21	Complex IQ Packed – 12 bits sample – N blocks transfer	IQ packed	complex	12	N Count	1 buffer (I & Q buffer)
22	Complex IQ Packed – 16 bits sample – N blocks transfer	IQ packed	complex	16	N Count	1 buffer (I & Q buffer)

PROCESSOR SUBSYSTEM 2 (PS2)

Table 64. Valid Profiles of To-Mem DMA (Continued)

Idx	DMA Profile Name	Pack Mode Supported	Data Type	Data Width	Transfer Type	Buffer Number
23	Real – 2 bits sample – Continuous block transfer	yes	real	2	Continuous	1 buffer (I-Only)
24	Real – 12 bits sample – Continuous block transfer	yes	real	12	Continuous	1 buffer (I-Only)
25	Real – 16 bits sample – Continuous block transfer	yes	real	16	Continuous	1 buffer (I-Only)
26	Real – 22 bits sample – Continuous block transfer	no	real	22	Continuous	1 buffer (I-Only)
27	Complex – 2 bits sample – Continuous block transfer	yes	complex	2	Continuous	2 buffers (I buffer & Q buffer)
28	Complex – 12 bits sample – Continuous block transfer	yes	complex	12	Continuous	2 buffers (I buffer & Q buffer)
29	Complex – 16 bits sample – Continuous block transfer	yes	complex	16	Continuous	2 buffers (I buffer & Q buffer)
30	Complex – 22 bits sample – Continuous block transfer	no	complex	22	Continuous	2 buffers (I buffer & Q buffer)
31	Complex IQ Packed – 2 bits sample – Continuous block transfer	IQ packed	complex	2	Continuous	1 buffer (I & Q buffer)
32	Complex IQ Packed – 12 bits sample – Continuous block transfer	IQ packed	complex	12	Continuous	1 buffer (I & Q buffer)
33	Complex IQ Packed – 16 bits sample – Continuous block transfer	IQ packed	complex	16	Continuous	1 buffer (I & Q buffer)
34	Real – 2 bits sample – N blocks transfer	yes	real	2	N Count Alternate	1 buffer (I-Only)
35	Real – 12 bits sample – N blocks transfer	yes	real	12	N Count Alternate	1 buffer (I-Only)
36	Real – 16 bits sample – N blocks transfer	yes	real	16	N Count Alternate	1 buffer (I-Only)
37	Real – 22 bits sample – N blocks transfer	no	real	22	N Count Alternate	1 buffer (I-Only)
38	Complex – 2 bits sample – N blocks transfer	yes	complex	2	N Count Alternate	2 buffers (I buffer & Q buffer)
39	Complex – 12 bits sample – N blocks transfer	yes	complex	12	N Count Alternate	2 buffers (I buffer & Q buffer)
40	Complex – 16 bits sample – N blocks transfer	yes	complex	16	N Count Alternate	2 buffers (I buffer & Q buffer)
41	Complex – 22 bits sample – N blocks transfer	no	complex	22	N Count Alternate	2 buffers (I buffer & Q buffer)
42	Complex IQ Packed – 2 bits sample – N blocks transfer	IQ packed	complex	2	N Count Alternate	1 buffer (I & Q buffer)
43	Complex IQ Packed – 12 bits sample – N blocks transfer	IQ packed	complex	12	N Count Alternate	1 buffer (I & Q buffer)
44	Complex IQ Packed – 16 bits sample – N blocks transfer	IQ packed	complex	16	N Count Alternate	1 buffer (I & Q buffer)
45	Real – 2 bits sample – Continuous block transfer	yes	real	2	Continuous Alternate	1 buffer (I-Only)
46	Real – 12 bits sample – Continuous block transfer	yes	real	12	Continuous Alternate	1 buffer (I-Only)
47	Real – 16 bits sample – Continuous block transfer	yes	real	16	Continuous Alternate	1 buffer (I-Only)
48	Real – 22 bits sample – Continuous block transfer	no	real	22	Continuous Alternate	1 buffer (I-Only)
49	Complex – 2 bits sample – Continuous block transfer	yes	complex	2	Continuous Alternate	2 buffers (I buffer & Q buffer)
50	Complex – 12 bits sample – Continuous block transfer	yes	complex	12	Continuous Alternate	2 buffers (I buffer & Q buffer)
51	Complex – 16 bits sample – Continuous block transfer	yes	complex	16	Continuous Alternate	2 buffers (I buffer & Q buffer)
52	Complex – 22 bits sample – Continuous block transfer	no	complex	22	Continuous Alternate	2 buffers (I buffer & Q buffer)
53	Complex IQ Packed – 2 bits sample – Continuous block transfer	IQ packed	complex	2	Continuous Alternate	1 buffer (I & Q buffer)
54	Complex IQ Packed – 12 bits sample – Continuous block transfer	IQ packed	complex	12	Continuous Alternate	1 buffer (I & Q buffer)
55	Complex IQ Packed – 16 bits sample – Continuous block transfer	IQ packed	complex	16	Continuous Alternate	1 buffer (I & Q buffer)

PROCESSOR SUBSYSTEM 2 (PS2)

Configure To-Mem DMA Memory Data Buffers

To write data (I/Q samples) to memories, the To-Mem DMA must know the location and size of the data buffers. The following information must be provided to the DMA to complete the configuration of the DMA:

Applicable to all profiles:

Table 65. To-Mem DMA Memory Data Buffer Configurations Applicable to All Profiles

Configuration Parameters	Explanations
Main I data buffer start address	32 bits start address located in PS2 memories (32-bit aligned)
Main Q data buffer start address	32 bits start address located in PS2 memories (32-bit aligned) unused if real data type used or if 1 complex buffer used (I&Q same word)
Data buffer size	Size of the block transfer for I data & Q data buffer Applicable for both main and alternate buffers
I data buffer address increment	Address increment to fetch next I data in memory
Q data buffer address increment	Address increment to fetch next Q data in memory

Applicable to Alternate transfer type profile only:

Table 66. To-Mem DMA Memory Data Buffer Configurations Applicable to Alternate Transfer Type Profile Only

Configuration Parameters	Explanations
Alternate I data buffer start address	32 bits start address located in PS2 memories (32-bit aligned)
Alternate Q data buffer start address	32 bits start address located in PS2 memories (32-bit aligned)

Configure To-Mem DMA Interrupts

A DMA interrupt is connected to PS2 ARM processor and can be enabled by ARM to control and monitor the block transfer of the To-Mem DMA.

3 interrupts status bits are used to monitor the processing of the block transfer:

Table 67. To-Mem DMA Interrupts

Interrupt name	Description
DMA End of Transfer of all blocks.	This interrupt is triggered when the DMA execution is done. ► In continuous mode, it happens when ARM ends the DMA. ► In single transfer or N block transfer, it happens when all block transfers are done.
DMA Current block transfer done	Notify that the current block transfer finished. For alternate buffer transfer mode, it can be used by ARM to copy the unused buffer to another location for next block transfer execution while the other buffer is being filled by the DMA.
DMA Error status	(For debug purpose) Error: It signifies that the DMA could not store data to the memory location. Check the memory buffer address accessible to the DMA (Read Status Value).

Start/Stop To-Mem DMA

To-Mem DMA can be started by PS2 ARM using the start function provided by the driver.

To-Mem DMA can be stopped by:

- In “Single” mode: will be ended automatically once the last transfer is done.
- In “N block transfer” mode: will be ended automatically once the last transfer is done or PS2 ARM sending a stop command to the DMA using stop function provided by the driver.
- In “Continuous” mode: PS2 ARM can send a stop command to the DMA using stop function provided by the driver.

DMA Examples and Resources

For detailed information on the peripheral drivers, refer to the doxygen files as detailed in the [Resources and Examples](#) chapter.

A demo of the DMAs is given in the Transceiver Evaluation Software tab. When ‘PS2 in Data Path’ is selected (refer [Figure 246](#)), all signal data will run through the DMAs. Run the TES to see the DMAs working in a live demo.

PROCESSOR SUBSYSTEM 2 (PS2)

If 1T profile is selected, with PS2 enabled, under the control of TES, the FPGA sends the Tx data from the FPGA SSI interface to the ADRV910x SSI interface. Then the Tx data is sent to the PS2 memory from Tx SSI by PS2 **"To-Mem DMA"**. The same Tx data is then sent to PS1 Tx1 data path from the PS2 memory by PS2 **"From-Mem DMA"**. User could observe the Tx RF output using a spectrum analyzer.

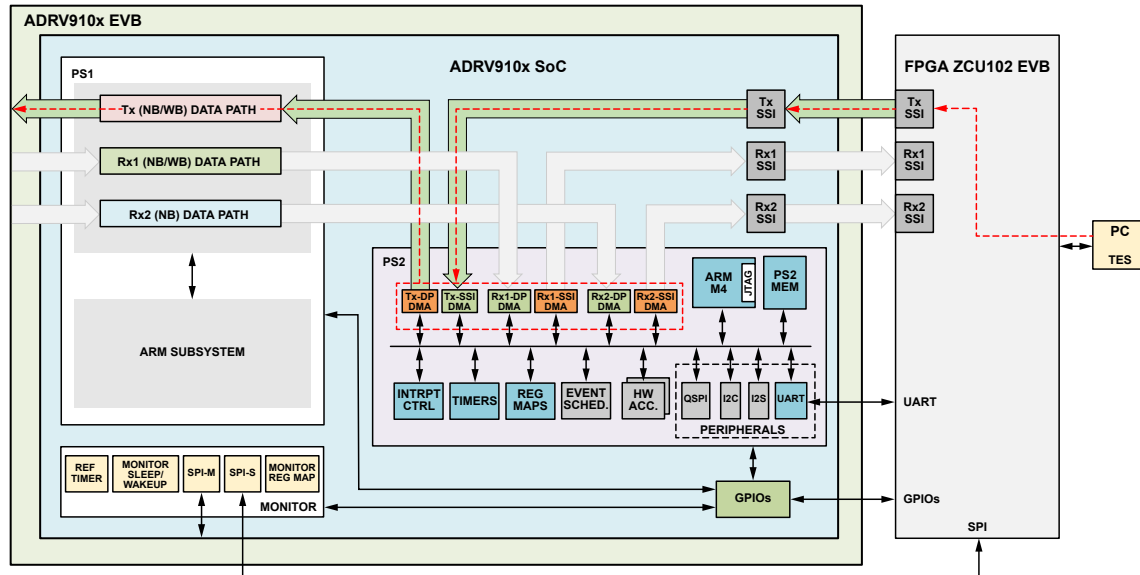


Figure 179. Data Path in 1T Profile with PS2 Enabled

If 1R profile is selected, with PS2 enabled, under the control of TES, the Rx2 input data will first go through the ADRV910x PS1 Rx2 data path, after that, the Rx2 data will be sent to the PS2 memory by PS2 **"To-Mem DMA"**. Then the Rx2 data is sent to Rx2 SSI interface from the PS2 memory by PS2 **"From-Mem DMA"**. FPGA receives the Rx2 data from the Rx2 SSI interface. User could observe the Rx2 data in both time and frequency domain in TES and further capture the Rx2 data for offline analysis.

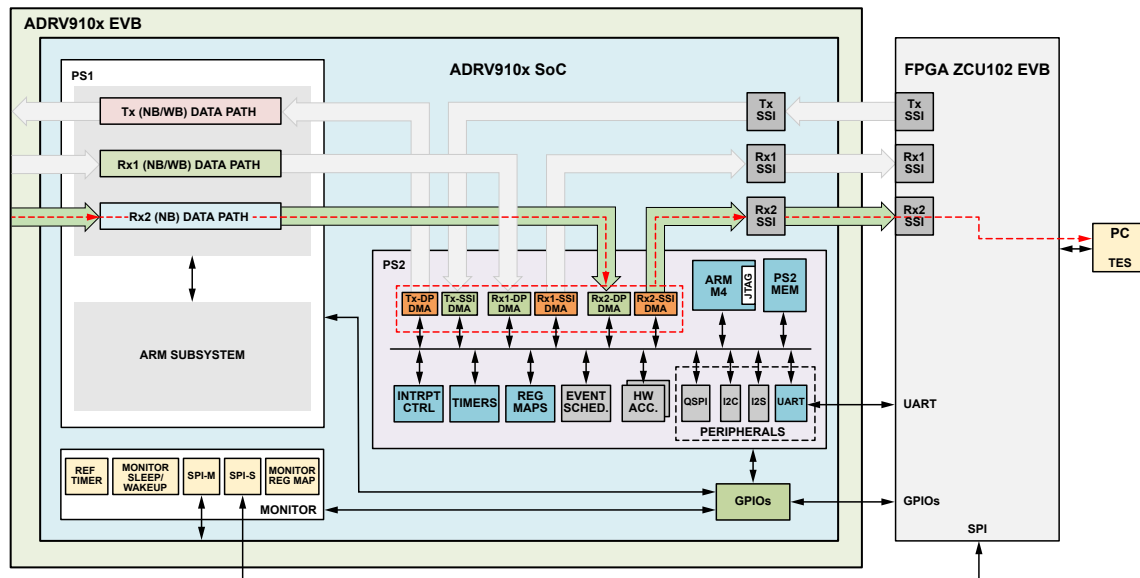


Figure 180. Data Path in 1R Profile with PS2 Enabled

If "SSI loopback Enabled" is selected (SSI loopback), with PS2 enabled, it demonstrates an example use case of looping Tx signal received from Tx SSI to Rx SSI through PS2 memory consisting of the following steps as shown:

- ▶ Tx data is sent to the ADRV910x through Tx SSI interface by FPGA
- ▶ The Tx data is sent to the PS2 memory by PS2 **"To-Mem DMA"**
- ▶ The Tx data is sent to the Rx2 SSI interface by PS2 **"From-Mem DMA"**

PROCESSOR SUBSYSTEM 2 (PS2)

- Tx data is received by the FPGA from the Rx2 SSI interface

User could verify in TES that the Rx2 received data is the same as the Tx transmit data.

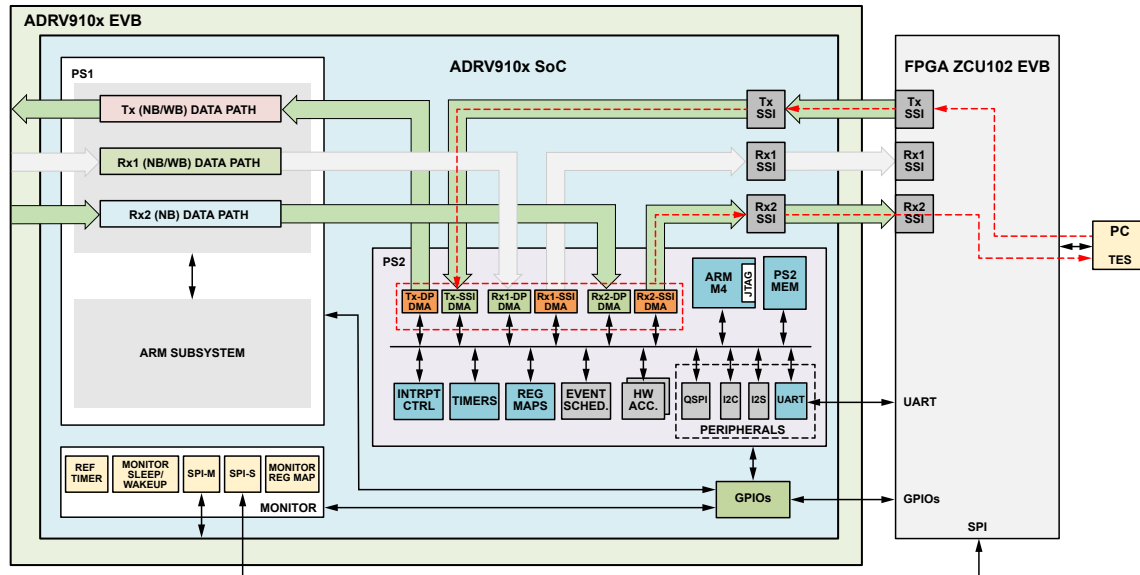


Figure 181. Data Path in SSI Loopback Demo User Case

As mentioned earlier, in the loopback demo use case, user could choose one of the two options for both Tx and Rx2:

- 16-Bits I/Q Data (24ksps data port sample rate)
- 16-Bits I-only Direct FM/FSK Symbols (48ksps data port sample rate)

In addition to that, this use case can be operated with both “No OS” or “free RTOS” modes.

The source code for the TES PS2 demo is given in the ‘usecase file’ as detailed in the [Resources and Examples](#) chapter.

PS2 PERIPHERALS

UART

Universal Asynchronous Receiver-Transmitter (UART) is one of the commonly used device-to-device communication protocols. It uses serial communication between two devices with a configurable baud rate. It is asynchronous in the sense that there is no clock signal to synchronize the output bits from the transmitting device and the input bits going to the receiving device. The “synchronicity” between the transmitting and receiving device is achieved by having the same baud rate on both devices.

As shown in [Figure 182](#), each UART device has a dedicated transmit pin (TX) and a receive pin (RX) for transmitting and receiving signals. The TX and RX pins of one UART are connected to the RX and TX pins of the other UART, respectively. In addition to that, to prevent the UART transmitter from sending data while the receiving counterpart is not ready, a request-to-send (RTS)/ clear-to-send (CTS) hardware flow control mechanism is supported. Similarly, the RTS and CTS pins of one UART are connected to the CTS and RTS pins of the other UART, respectively. These two lines allow the receiving UART and the transmitting UART to alert each other for data transfer. The signals are active low by default. When the receiving UART is ready to accept the data, it pulls its RTS line low, informing the transmitting UART, through its CTS line, to begin sending data. Transmitting UART is halted when its CTS pin state becomes high, indicating that the receiving UART receive buffer is getting full. Further data may cause an overrun error. This ensures a reliable data connection between the transmitting and receiving UART. The transmitting UART could be connected to a controlling data bus that sends data in a parallel format. The transmitting UART converts the data from parallel to serial and sends the data bit by bit to the receiving UART. After receiving the serial data, the receiving UART converts the serial data to parallel and sends it to the data bus. Note either UART1 or UART2 in [Figure 182](#) could be either a transmitting or receiving UART.

PROCESSOR SUBSYSTEM 2 (PS2)

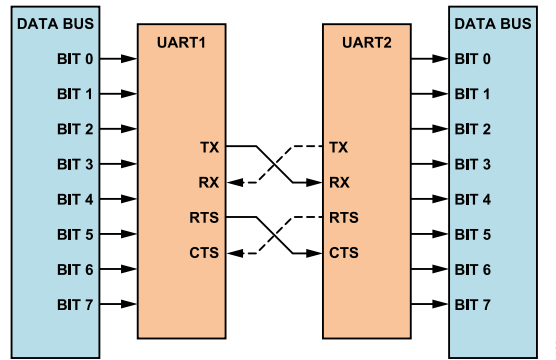


Figure 182. UART Transmitting and Receiving with Data Bus

For UART pin mapping, refer to 'PS2 Peripherals Mapping'.

UART Clock Setting

As mentioned earlier, for UART communications, the baud rate needs to be set the same on both the transmitting and receiving UART. The baud rate defines the rate at which information is transferred to a communication channel. It specifies the number of signal units transferred per second. For each signal unit, it could include multiple bits. Due to the varied number of bits in a signal unit in the UART protocol, for clarity, bit rate which specifies the number of bits transferred per second (bits/s) is used in the following descriptions.

PS2 UART block has 2 clocks: one is the AHB clock which is the same as ARM clock and the other one is the UART clock which is running of a divided version of the AHB clock. Based on the ARM clock and the desired bit rate, the divisor is automatically calculated by the UART drivers to achieve the desired bit rate.

Based on the ADRV910x ARM clock rate, the UART block could achieve a set of different bit rates such as 1100, 3000, 6000, 12000, 245000, 96000, 144000, 192000, 384000, 576000, 1152000, 1280000 and 256000 bits/s.

Bit rate could be converted to baud rate based on the number of bits in one UART packet.

UART Configurations

Transfer Modes

ADRV910x UART can use one of the following five transfer modes:

- ▶ Non-blocking ISR mode for both TX and RX
- ▶ Non-blocking ISR mode for TX only
- ▶ Non-blocking ISR mode for RX only
- ▶ Non-blocking DMA mode for TX and RX
- ▶ Non-blocking DMA mode for TX
- ▶ Non-blocking DMA mode for RX
- ▶ Blocking mode

For "Non-blocking ISR mode (for both TX and RX, for TX only and for RX only)", data transfer happens only when a UART transmit or receive interrupt is generated and received by the ARM. For "Non-blocking DMA mode (for Both TX and RX, for TX only and for RX only)", UART data transfer is controlled by DMA which does not block other ARM operations. For "Blocking mode", ARM will not respond to other requests until UART data transfer is completed.

Bit Rate

Bit rate configures the desired bits per second for UART data transfer.

Packet Format

Figure 183 describes the ADRV910x UART packet format. As shown in Figure 183, a UART packet includes a mandatory 1 bit Start Bit, 5 to 8 bits varied length Data Frame, an optional 1 bit Parity Bit and 1 to 2 bits Stop Bits.

PROCESSOR SUBSYSTEM 2 (PS2)



Figure 183. ADRV910x UART Packet Format

Start Bit

By default, the UART data transmission line is held at a high voltage level when it's not transmitting data. To start the data transfer, the transmitting UART pulls the transmission line from high to low for one bit (one clock cycle). When the receiving UART detects the high to low voltage transition, it begins to read the data bits at the specified bit rate.

Data Frame

Each data frame could contain a varied length of 5 to 8 bits which can be configured by user. The bits are arranged from the least significant bit (right after the Start Bit) to the most significant bit (before the optional Parity Bit).

Parity Bit

Parity Bit is an optional bit user can insert between the most significant data bit and the stop bit. When it is used, depending on the user configuration of “even parity” or “odd parity”, the bit is calculated based on the following rule:

- ▶ For even parity: the total count of logical-1 bits including the data bits and the Parity Bit must be an even value.
- ▶ For odd parity: the total count of logical-1 bits including the data bits and the Parity Bit must be an odd value.

The Parity Bit provides a method for the receiving UART to tell if any data has changed during transmission since bits can be changed by electromagnetic radiation, mismatched baud rates, or long-distance data transfers. After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number based on the rule described above. When the Parity Bit matches the data, the UART transmission was free of errors.

Stop Bits

Stop Bits are used to signal the end of the data packet by driving the data transmission line from a low voltage to a high voltage for 1 to 2 bits duration. Two options are provided to user as below:

- ▶ 1 stop bit regardless of data frame length
- ▶ 1.5 or 2 Stop Bits based on word length:
 - ▶ For 5 bits data length, 1.5 Stop Bits are used
 - ▶ For 6/7/8 bits data length, 2 Stop Bits are used.

Hardware Flow Control

Hardware flow control with RTS and CTS pins are optional, which can be enabled or disabled by user.

User Defined Callback Function

User could define further operations through a callback function based on the following UART states:

- ▶ UART is currently Busy
- ▶ UART Tx has completed
- ▶ UART Rx has completed
- ▶ Framing Error Detected
 - ▶ If the data line is not in the expected state (high) when the “stop” bit is expected, the UART will signal a framing error.
- ▶ Parity Error Detected
 - ▶ Parity errors occur when parity is implemented on the data link and there is a corruption that causes a parity mismatch in the received data.
- ▶ Overrun Error Detected
 - ▶ A UART overrun is an error that occurs when the UART FIFO cannot keep up with data being received.

PROCESSOR SUBSYSTEM 2 (PS2)

The call back function is usually used in non-blocking ISR mode of operation.

UART Internal Loopback

By enabling UART loopback option, it disconnects the ADRV910x UART receiver's input from the UART RX pin, and internally redirects the transmit output to the ADRV910x UART receiver.

UART For IQ Transfer

UART can be used for IQ data transfer for the Rx1/Rx2/Tx channels. Instead of using the dedicated SSI lines, IQ data can be transferred over the UART bus to/from ADRV910x. The datapath DMAs are used to buffer the data between the datapaths and the UART. [Figure 184](#) shows the signal flow in the Tx case. The IQ data is sent from the PC/FPGA over UART into a buffer on PS2. This buffer is read onto the Tx datapath to PS1 for transmission. In the Rx case, the data is received from PS1 on Rx1 or Rx2, and written to a buffer on PS2. The data is then read from the buffer over UART to the FPGA/PC.

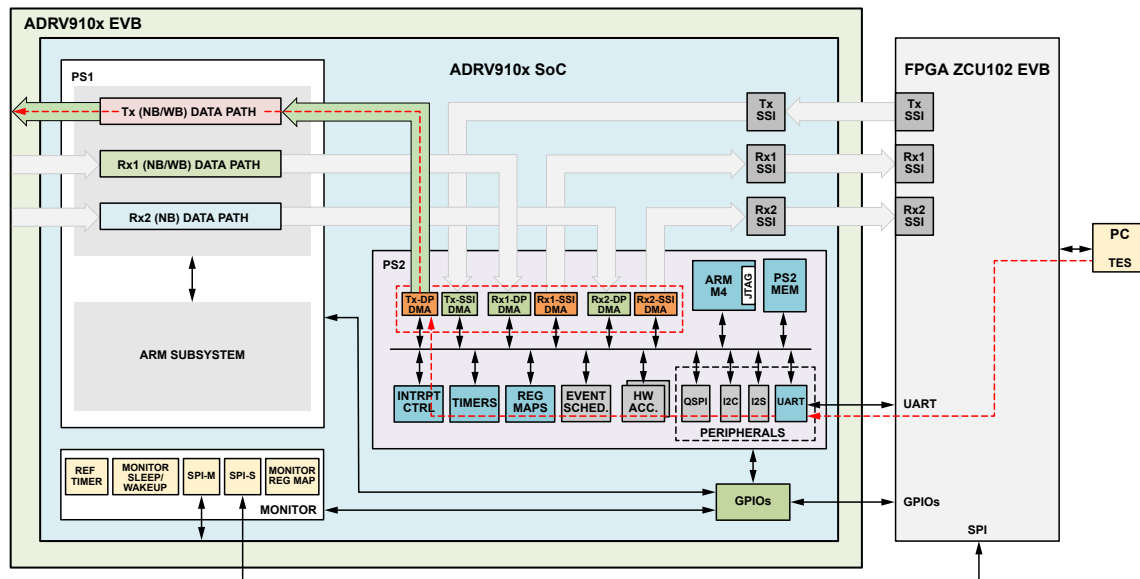


Figure 184. UART For Tx IQ Transfer Signal Flow Block Diagram

Note that the UART is limited to a max data rate of 12.5 Mbps when using system clock of 192 MHz.

An example of this data transfer is given in TES and described in [TES PS2 Features](#).

General Purpose IO (GPIO)

In ADRV910x, all interface pads are considered GPIOs by default:

- ▶ 20 Digital GPIOs (DGPIOs)
- ▶ 14 Analog GPIOs (AGPIOs)
- ▶ 26 SSI GPIOs (8 RX1 + 8 RX2 + 10 TX)
- ▶ 5 Misc GPIOs (RX1_EN, RX2_EN, TX_EN, TX_EN_OPT, GP_INT)

GPIOs Feature List

The following is a list of features for GPIOs:

- ▶ Source (output) & destination (input) control capabilities: bidirectional pads
- ▶ Shared GPIO resources between PS1 and PS2 subsystem
- ▶ Specific function triggers from Host to PS2 subsystem (GPIO interrupts, ...)
- ▶ PS2 Peripheral mapping: GPIOs to map peripherals (UART, I2C, I2S, QSPI-M, SPI-M)

PROCESSOR SUBSYSTEM 2 (PS2)

GPIO Control

GPIO Source/Destination Control

All the interface pads are considered GPIOs by default. In ADRV910x, the total number of GPIOs is 65.

Table 68. GPIOs in ADRV910x

GPIO type	GPIO Number	Description
DGPIO	20	Digital GPIO
AGPIO	14	Analog GPIO
SSI_RX1_GPIO	8	RX1 SSI pads used as GPIO
SSI_RX2_GPIO	8	RX2 SSI pads used as GPIO
SSI_TX_GPIO	10	TX SSI pads used as GPIO
RX_TX_EN_GPIO	4	RX/TX Enable pads used as GPIO
GP_INT_GPIO	1	General Purpose Interrupt pad used as GPIO
Total	65	

Those GPIO resources are shared between PS1 and PS2, so PS2 users must ensure that those resources are available before updating the configuration. In summary:

- ▶ The direction of each GPIO is programmable,
- ▶ The control of each GPIO level in output mode is programmable,
- ▶ The status of each GPIO level in input mode is readable using the functions provided by the driver.

PS2 Digital GPIO Interrupts

Host has the capability to trigger PS2 ARM interrupts through digital GPIO setting. PS2 ARM subsystem has 4 interrupts that can be independently connected to any digital GPIO pads source (DGPIO). The selected DGPIO must be configured as input using the function provided by the driver.

Each interrupt source can be independently configured from PS2 subsystem to map one of the 20 digital GPIO pads to this interrupt. Each interrupt source is maskable.

Those digital GPIO interrupts are sampled on the PS2 ARM clock. The interrupt can be level or pulse triggered but must be held active high **at least** 2 cycles of ARM clock cycles to be detected by ARM. The polarity of those interrupts is active high and cannot be changed.

Note: only digital GPIOs can be mapped on those interrupts. PS2 does not have the capability to map other GPIOs such as analog, SSI or Misc GPIOs.

I2S

The ADRV910x platform includes a flexible serial audio interface that supports standard 2-channel I²S, left/right-justified formats, and scalable TDM framing up to 16 channels. The current implementation extends the traditional I²S protocol with selectable bit-widths, programmable framing, GPIO-based pin-muxing, and DMA-centric data movement.

The I²S block can operate as controller or peripheral, can transmit or receive independently, and is designed to interface with ADCs, DACs, DSPs, and external SoCs using industry-standard audio formats. All audio samples are handled in two's-complement, MSB-first format. There are two interfaces available on ADRV910x: I2S1 and I2S2, where both can be used independently with pin mapping provided in [Table 69](#).

For more information, please refer to [I2S - Wikipedia](#).

ADRV910x I2S features include:

- ▶ Supports controller or peripheral clocking, with BCLK and FSYNC either generated internally or received from external audio devices.
- ▶ Two independent I²S ports, each providing SDATAI, SDATAO, BCLK, and FSYNC pins.
- ▶ Stereo (2-channel) I²S mode, MSB-first, two's-complement, with 16-bit and 24-bit widths.
- ▶ TDM mode supporting 2–16 channels, with selectable slot widths (16/24/32 bits).
- ▶ Selectable data formats: I²S-delay-by-1, left-justified, and right-justified modes.
- ▶ Programmable FSYNC timing, including bypass and asynchronous TDM framing options.

PROCESSOR SUBSYSTEM 2 (PS2)

- ▶ Full controller/peripheral Tx/Rx combinations: Controller-Tx, Controller-Rx, Peripheral-Tx, Peripheral-Rx.
- ▶ Flexible BCLK derivation from AHB clock via BCLK_DIV, with recommended presets for standard 48/96/192 kHz audio.
- ▶ Tri-state unused channels or bit-slots for shared-bus multi-device operation.
- ▶ DMA-optimized architecture, with one request per frame (not per sample). Required for high-channel-count TDM.
- ▶ No ARM interrupt support, so DMA or cycle-matched ARM polling is required.
- ▶ Built-in data-alignment controls for fine timing adjustments.
- ▶ High-speed operation up to ~24.576 MHz BCLK, with recommended pad drive-strength enhancements.

QSPI-M

The ADRV910x includes an enhanced Quad SPI (QSPI) peripheral with support for high-speed serial flash access and multi-I/O data transport. The QSPI pins may be mapped to several GPIOs as defined in [Table 69](#). Only one QSPI mapping set may be active at a time.

The QSPI block implements an enhanced Serial Peripheral Interface capable of high-bandwidth communication using 1-bit (standard SPI), 2-bit (Dual-I/O), or 4-bit (Quad-I/O) data transfers. It supports both legacy FIFO-based SPI transfers and a high-performance memory-mapped mode that enables direct execution of code (XIP) from external serial flash memory.

QSPI includes programmable serial timing (CPOL/CPHA), variable word sizes (8/16/32-bit), multi-peripheral support, flow-control signalling, fast read modes, DMA operation for transmit/receive, Dual/Quad I/O modes, and automated read-header generation in memory-mapped mode. Up to 128 MB of SPI flash may be mapped directly into processor address space.

QSPI operates similarly to SPI but with 4 data wires, please refer to: [Introduction to SPI](#) for more regular SPI information.

ADRV910x QSPI includes the following features:

- ▶ Supports controller/main operation for control of SPI-compatible peripheral devices.
- ▶ Single-, Dual-, and Quad-I/O operation for 1-bit, 2-bit, or 4-bit wide transfers.
- ▶ Memory-mapped operating mode, enabling direct execution from off-chip SPI flash (XIP).
- ▶ Programmable CPOL/CPHA settings supporting all four SPI modes.
- ▶ Selectable 8-, 16-, or 32-bit transfer sizes.
- ▶ Programmable baud-rate divider for generating the serial clock (SCK).
- ▶ Flow-control support via RDY pin for interaction with slower peripheral devices.
- ▶ Multiple chip-select outputs, with hardware or software-controlled CS timing.
- ▶ Transmit and receive FIFOs (depth depends on selected word size).
- ▶ DMA support for both transmit and receive channels.
- ▶ Fast-read, Dual-Output, Dual-I/O, Quad-Output, and Quad-I/O read modes.
- ▶ Automated read-header generation in memory-mapped mode.
- ▶ Merge option to reduce repeated read-headers during sequential memory access.
- ▶ Wrap support (flash-dependent) to optimize sequential fetch patterns.
- ▶ Interrupts for Tx/Rx FIFO levels, transfer start/finish, under/overflow, collisions and errors.
- ▶ Open-drain output option for use in multi-controller environments.
- ▶ Loopback test mode for self-diagnostic verification.
- ▶ Prefetch buffer to reduce bus stall time during memory-mapped access.

I²C

ADRV910x includes a I²C bus which can be used as a main and/or subordinate device. The I²C pins can be mapped to several different GPIOs as per '[PS2 Peripherals Mapping](#)'. However, only one mapping can be used at a time.

For general information on I²C, please refer to: [I²C Primer](#). ADRV910x I²C includes the following features:

- ▶ Main and/or subordinate modes (controller/peripheral).
- ▶ 2-byte transmit and receive FIFO for the main and subordinate.
- ▶ Support for repeated starts.*
- ▶ Continuous read mode for the main or up to 512 bytes fixed read.

PROCESSOR SUBSYSTEM 2 (PS2)

- ▶ Clock stretching supported for the subordinate and the main.*
- ▶ Support for 4 seven-bit device addresses in the subordinate.
- ▶ Support for internal and external loopback.
- ▶ Support for DMA.*
- ▶ Support one ARM IRQ for main and one ARM IRQ for subordinate.
- ▶ Support for configurable pin mapping.
- ▶ Support a pre I²C clock divider from ARM clock to generate I²C's root clock.
- ▶ Queuing up to 10 transfers (Tx/Rx).

*Not yet supported

SPI-M

ADRV910x has the ability to be used as controller/main for SPI communication. The SPI pins are able to be mapped to several different GPIOs as per [Table 69](#) as seen above. Only one mapping set of SPI interface can be used at any one time.

For more general information on SPI, please refer to: [Introduction to SPI](#). ADRV910x SPI includes the following features:

- ▶ Controller/main for control of SPI peripheral/subordinate devices.
- ▶ Full-duplex transfers with programmable CPOL/CPHA clock formats.
- ▶ Support for MSB-first and LSB-first transfers.
- ▶ Dedicated transmit and receive FIFOs (8 bytes each).
- ▶ Configurable interrupt generation after 1–8 bytes (Tx/Rx).
- ▶ Continuous transfer mode with optional multi-frame support (CNT + FRAMECONT).
- ▶ Read-command mode (half-duplex), including TXBYTES, OVERLAP, and CMDEN controls.
- ▶ 3-pin bidirectional data mode support (shared MOSI/MISO in read-command mode).
- ▶ Loopback test mode.
- ▶ Wired-OR (open-drain) output option for multi-controller/multi-peripheral busses.
- ▶ Multiple chip-select outputs (up to four CS lines) with software CS override.
- ▶ Flow-control support.
- ▶ Programmable baud-rate divider for SCLK generation.
- ▶ Rx overflow overwrite option and Tx underflow zero-padding.
- ▶ Tx/Rx FIFO flush controls.
- ▶ DMA support for both transmit and receive, requiring 16-bit DMA transfers.
- ▶ SPI interrupts for Tx, Rx, underflow, overflow, transfer-done, ready-edge, and CS-edge.
- ▶ Transfer byte counter and real-time transfer count register for debugging.
- ▶ Support for 3 or 4-wire SPI physical interfaces.

PS2 Peripherals Mapping

The following table shows the PS2 peripherals mapping on all the interfaces pads (DGPIOs, AGPIOs, SSI Lanes, RX/TX enables, GP interrupt).

By default, all the interface pads are configured as GPIO. In that mode, they can be controlled as input or output using the functions provided in the C code driver.

In this table:

- ▶ each line represents 1 GPIO pad,
- ▶ each column represents 1 function that can be mapped to this GPIO pad.

PROCESSOR SUBSYSTEM 2 (PS2)

Table 69. PS2 Peripheral Mapping

DGPIO	Pin Name	PS2 peripheral							JTAG
		UART1	UART2	I2C	SPI-M	QSPI-M	I2S1	I2S2	
0	DGPIO_0	-	-	-	CS0	MSDIO0	-	-	-
1	DGPIO_1	-	-	-	CS1	MSDIO1	-	-	-
2	DGPIO_2	-	-	-	CS2	MSDIO2	-	-	-
3	DGPIO_3	-	-	-	CS3	MSDIO3	-	-	TDO
4	DGPIO_4	-	-	-	CS0	MCS1	-	-	TRST
5	DGPIO_5	-	-	-	CS1	-	-	-	TDI
6	DGPIO_6	-	-	SCL	CS2	-	-	-	TMS
7	DGPIO_7	-	-	SDA	CS3	-	-	-	TCLK
8	DGPIO_8	-	-	-	CS0	-	-	-	-
9	DGPIO_9	-	-	-	SCLK	-	-	-	-
10	DGPIO_10	-	-	-	MISO	-	-	-	-
11	DGPIO_11	-	-	-	MOSI	-	-	-	-
12	DGPIO_12	-	-	-	CS0	MCS0	-	SDATAO	-
13	DGPIO_13	-	-	-	SCLK	MCLK	-	BCLK	-
14	DGPIO_14	TX	-	-	MISO	MSDIO3	-	FSYNC	-
15	DGPIO_15	RX	-	-	MOSI	MCS1	-	SDATAI	-
16	DGPIO_16	RTS	-	-	CS1	MCS1	SDATAO	-	-
17	DGPIO_17	CTS	-	-	CS2	-	BCLK	-	-
18	DGPIO_18	TX	-	SCL	CS3	-	FSYNC	-	-
19	DGPIO_19	RX	-	SDA	-	-	SDATAI	-	-
-	AGPIO_0	-	-	-	CS0	-	-	-	-
-	AGPIO_1	-	-	-	CS1	-	-	-	-
-	AGPIO_2	-	-	-	CS2	-	-	-	-
-	AGPIO_3	-	-	-	CS3	-	-	-	-
-	AGPIO_4	-	-	-	CS0	-	-	-	-
-	AGPIO_5	-	-	-	CS1	-	-	-	-
-	AGPIO_6	-	-	-	CS2	-	-	-	-
-	AGPIO_7	-	-	-	CS3	-	-	-	-
-	AGPIO_8	-	-	-	CS0	-	-	-	-
-	AGPIO_9	-	-	-	SCLK	-	-	-	-
-	AGPIO_10	-	-	-	MISO	-	-	-	-
-	AGPIO_11	-	-	-	MOSI	-	-	-	-
-	AGPIO_12	-	-	-	-	-	-	-	-
-	AGPIO_13	-	-	-	-	-	-	-	-
20	SSI_TX_CLK_N	-	TX	-	-	MCS1	-	-	-
21	SSI_TX_CLK_P	-	RX	-	-	MCS0	-	-	-
22	SSI_TX_IDATA_N	-	RTS	-	-	MCLK	-	SDATAO	-
23	SSI_TX_IDATA_P	-	CTS	-	-	MSDIO0	-	BCLK	-
24	SSI_TX_QDATA_N	-	-	SCL	-	MSDIO1	-	FSYNC	-
25	SSI_TX_QDATA_P	-	-	SDA	-	MSDIO2	-	SDATAI	-
26	SSI_TX_STROBE_P	TX	-	-	-	MSDIO3	SDATAO	-	-
27	SSI_TX_STROBE_N	RX	-	-	-	-	BCLK	-	-
28	SSI_TX_REFCLK_N	RTS	-	-	-	-	FSYNC	-	-
29	SSI_TX_REFCLK_P	CTS	-	-	-	-	SDATAI	-	-
30	SSI_RX2_IDATA_N	-	TX	-	-	-	-	-	-
31	SSI_RX2_IDATA_P	-	RX	-	-	-	-	-	-

PROCESSOR SUBSYSTEM 2 (PS2)

Table 69. PS2 Peripheral Mapping (Continued)

DGPIO	Pin Name	PS2 peripheral							JTAG
		UART1	UART2	I2C	SPI-M	QSPI-M	I2S1	I2S2	
32	SSI_RX2_QDATA_N	-	RTS	-	-	-	-	-	-
33	SSI_RX2_QDATA_P	-	CTS	-	-	-	-	-	-
34	SSI_RX2_STROBE_P	TX	-	-	-	-	-	SDATAO	-
35	SSI_RX2_STROBE_N	RX	-	-	-	-	-	BCLK	-
36	SSI_RX2_CLK_P	CTS	-	-	-	-	-	FSYNC	-
37	SSI_RX2_CLK_N	RTS	-	-	-	-	-	SDATAI	-
38	SSI_RX1_IDATA_N	-	TX	SCL	CS3	-	-	-	-
39	SSI_RX1_IDATA_P	-	RX	SDA	CS2	-	-	-	-
40	SSI_RX1_QDATA_N	-	RTS	-	SCLK	-	SDATAO	-	-
41	SSI_RX1_QDATA_P	-	CTS	-	MISO	-	BCLK	-	-
42	SSI_RX1_STROBE_P	-	-	-	MOSI	-	FSYNC	-	-
43	SSI_RX1_STROBE_N	-	-	-	CS0	-	SDATAI	-	-
44	SSI_RX1_CLK_P	-	-	-	CS1	MCLK	-	-	-
45	SSI_RX1_CLK_N	-	-	-	-	MSDIO0	-	-	-
46	TX_EN1	TX	-	SCL	-	MSDIO1	-	-	-
47	TX_EN2	RX	-	SDA	-	MSDIO2	-	-	-
48	RX_EN1	RTS	-	-	-	MSDIO3	-	-	-
49	RX_EN2	CTS	-	-	-	MCS0	-	-	-
50	GP_INT	-	-	-	-	MCS1	-	-	-

The GPIO pads can be repurposed for the following functions:

- ▶ SSI interfaces are used to transmit or receive I/Q samples to/from the host. There are 3 interfaces: RX1, RX2 using 8 GPIO pads and TX using 10 GPIO pads.
- ▶ RX/TX Enable pads are used as inputs to trigger RX or TX operation in PS1: 1 GPIO for each channel (RX1 NB/WB, RX2 NB, TX WB, TX NB)
- ▶ GP Interrupt pad: General Purpose Interrupt sent by ADRV910x to the Host.
- ▶ PS2 peripherals: UART1 / UART2 / I2C / SPI-M / QSPI-M / I2S1 / I2S2
- ▶ Debug: JTAG connect

Peripherals Examples and Resources

For detailed information on the peripheral drivers, refer to the doxygen files as detailed in the [Resources and Examples](#) chapter.

Examples for each peripheral is given in the example code as detailed in the [Resources and Examples](#) chapter. Run this code example to see an example of how each of the peripherals can be used and called including UART Tx and Rx cases, I²C and GPIO configuration.

A demo of all peripherals is given in the Transceiver Evaluation Software under the 'PS2' tab. Run the TES to see the peripherals working in a live demo including GPIO and UART loopback. The source code for the TES PS2 demo is given in the 'usecase project' as detailed in the [Resources and Examples](#) chapter.

BOOT MODE CONFIGURATIONS

Overview

The ADRV910x system is designed with volatile internal memory for both PS1 and PS2. This means that all firmware, configuration data, and operational parameters stored in memory are lost whenever the device is reset or powered down. Hence, the image must be loaded to ADRV910x memory at every boot-up.

To address this requirement, ADRV910x supports two primary boot modes:

- ▶ **Host-Driven Boot:** In this mode, an external baseband processor (BBIC) or FPGA acts as the controller, loading images directly into the device memory via SPI interface.

PROCESSOR SUBSYSTEM 2 (PS2)

- **Self-Boot:** In this mode, ADRV910x can operate autonomously by retrieving firmware and configuration data from an external QSPI Flash device. The SoC itself becomes the controller, enabling fully independent operation without a BBIC or FPGA after reset. Alternatively in this mode, it can also act as a peripheral and host-boot via QSPI or UART.

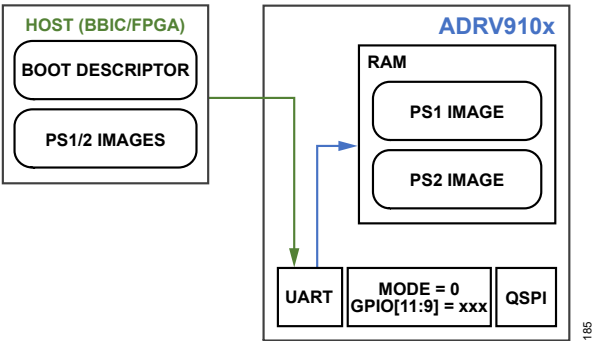


Figure 185. Host-Boot Functional Mode Operation.

ADRV910x Boot Modes

As mentioned earlier, ADRV910x supports two primary boot modes, selectable via the **MODE** pad and controlled by the host in combination with specific digital GPIO assignments.

► **MODE = 0: Normal Functional Mode (RAM Boot Mode)**

This is the default operating mode of ADRV910x. The device functions in Host Mode using the SPI peripheral interface for configuration and control.

► **MODE = 1 & DGPI0[11] = 0: ROM Boot Modes**

When **MODE** is asserted and **DGPI0[11]** is low, the boot source is determined by **DGPI0[10:9]**:

- **00:** QSPI Controller – Self-boot from external Flash memory.
- **01:** QSPI Peripheral – Host-boot via QSPI interface.
- **10:** UART – Host-boot via UART interface.

It is imperative that these **DGPI0s** are held at the above values **indefinitely** when **MODE** pad is high, or until device reset, from where **MODE** pad can be deasserted and **DGPI0[11:9]** return to their normal functions and do not need to remain at a certain value.

Peripheral Assignment Limitations

When self-boot mode is enabled by pulling **MODE** pad high, certain peripheral interfaces now have hard-coded **DGPI0** assignments and cannot be changed. Refer to [Table 70](#) below. Due to QSPI interface pins allocation, JTAG interface is unavailable when self-boot mode is enabled. This is due to a conflict that occurs in pin mapping, refer to [Table 69](#) for all the pin mapping.

Table 70. Static **DGPI0** Peripheral Assignments

	DGPI0	Peripheral Interface Signal
QSPI (controller or peripheral)	DGPI0[0]	MSDIO0
	DGPI0[1]	MSDIO1
	DGPI0[2]	MSDIO2
	DGPI0[3]	MSDIO3
	DGPI0[12]	CS0
	DGPI0[13]	MCLK
UART1	DGPI0[14]	TX
	DGPI0[15]	RX
	DGPI0[16]	RTS
	DGPI0[17]	CTS

PROCESSOR SUBSYSTEM 2 (PS2)

QSPI Controller Boot Mode

In QSPI Controller Boot Mode, ADRV910x operates as a self-contained system, autonomously loading its firmware and configuration data from an external QSPI Flash memory device. This mode is selected by asserting the **MODE** pad and configuring **DGPIO[11:9] = 000** at reset.

QSPI Controller Operation:

- ▶ Upon reset, the internal ROM code is executed.
- ▶ The QSPI interface is configured in controller mode, and the device accesses the external Flash IC.
- ▶ A boot descriptor is read from Flash, which specifies the addresses and sizes of the code and data to be loaded.
- ▶ The ROM copies the PS2 application image and any required data into internal SRAM.
- ▶ Integrity checks (CRC) are performed before execution is transferred to the loaded application.
- ▶ No host processor is required after reset; the device boots and operates independently.

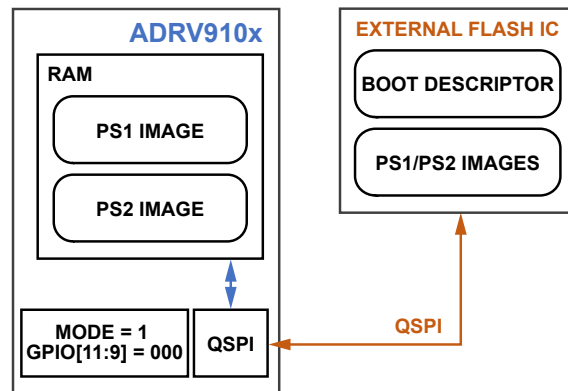


Figure 186. QSPI Controller Boot Mode Operation

QSPI Peripheral Boot Mode

In QSPI Peripheral Boot Mode, ADRV910x acts as a QSPI peripheral device, allowing an external host (such as a baseband processor or FPGA) to directly load firmware and configuration data into the device's internal memory through QSPI interface. This mode is selected by setting the **MODE** pad to 1 and configuring **DGPIO[11:9] = 001** at reset.

QSPI Peripheral Operation:

- ▶ After reset, the ROM detects QSPI Peripheral mode and configures the QSPI interface accordingly.
- ▶ The host processor (BBIC/FPGA) communicates with ADRV910x over the QSPI interface, sending a boot descriptor and the required code/data.
- ▶ The host controls the sequencing and timing of the image transfer.
- ▶ Once the transfer is complete, the host can command the device to jump to the loaded application code.
- ▶ The device remains under host control during the entire boot process.

PROCESSOR SUBSYSTEM 2 (PS2)

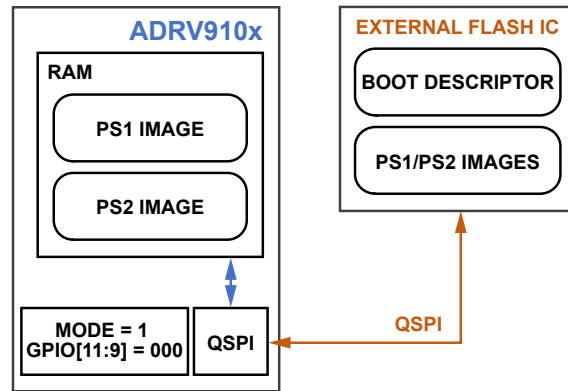


Figure 187. QSPI Peripheral Boot Mode Operation

UART Boot Mode

In UART Boot Mode, ADRV910x receives its firmware and configuration data from an external host (such as a baseband processor or FPGA) over UART interface. This mode is selected by setting the **MODE** pad to 1 and configuring **DGPIO[11:9] = 010** at reset.

UART Operation:

- ▶ Upon reset, the ROM configures the UART interface and waits for commands from the host.
- ▶ The host sends a boot descriptor and image data using a defined command protocol, with framing and acknowledgement characters (BEGIN, END, ACK, NACK).
- ▶ The device loads the received code and data into internal memory.
- ▶ After successful transfer and verification, the device jumps to the loaded application code.
- ▶ The UART baud rate is determined by the system clock, and the protocol supports robust error handling.
- ▶ The device remains under host control during the entire boot process.

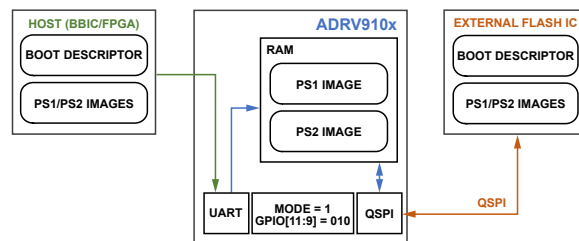


Figure 188. UART Boot Mode Operation

HARDWARE ACCELERATORS

ADRV910x-SoC contains 2 HWAs in PS2 Sub-system. They are connected to PS2 memory through an internal subsystem interconnect bus. Each HWA consists of several processing units (PUs) which implement specific DSP/Communication functions.

Once initialized by a firmware running on PS2-ARM processor, the HWA fetches input user data from PS2 Memory and stores results back to PS2 Memory using its own direct memory read/write access. the HWA-PU executes independently and in parallel with ARM.

The following diagram illustrates the HWA features:

PROCESSOR SUBSYSTEM 2 (PS2)

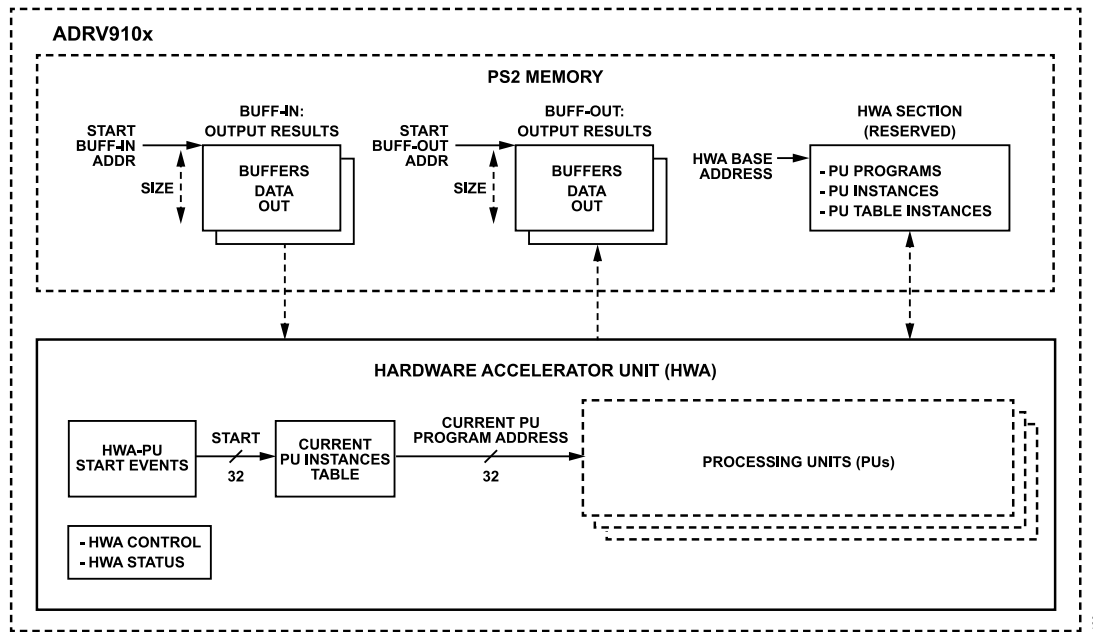


Figure 189. Hardware Accelerator Block Diagram

During a Processor Unit (PU) execution, the HWA fetches PU's instructions from PS2 memory to run specific PU program. The HWA-PU programs are stored in PS2-Memory as HWA binary section loaded during PS2 initialization.

A HWA consists of:

- ▶ A HWA Unit in the PS2 subsystem which can access PS2 memory through its sub-system bus interconnect:
 - ▶ It contains Processing Units with HWA control/Status information.
 - ▶ It can access HWA Memory Section
 - ▶ It can access HWA Input and Output Buffers.
- ▶ A HWA Memory Section containing:
 - ▶ HWA-PU program binary
 - ▶ PU instances
 - ▶ PU Table instances

HWA Unit

There are 2 Hardware Accelerator Units (HWAs) in PS2 sub-system: HWA-1 and HWA-2.

Each HWA contains several processing units (PUs) like: Limiter, CRC, Trellis Encoder, etc. as described in the 'Processing Unit (PU)' section. There are processing units (PUs) that are supported both by HWA-1 and HWA-2 and some are different between them.

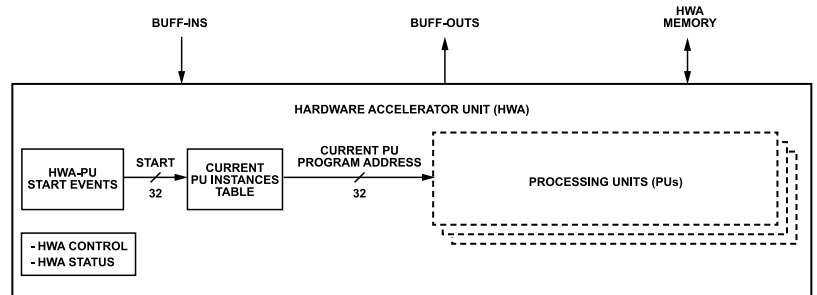


Figure 190. HWA Unit Block Diagram

PROCESSOR SUBSYSTEM 2 (PS2)

To use a processing unit (PU), the user starts by creating an instance of the PU with its related parameters. Then, this instance is registered in the HWA Memory Section. Several instances of the same PU can be created with different parameters.

The HWA can use only 32 PU instances at a time. The PUs run sequentially using HWA Start Driver. These 32 PU instances are pre-selected in a Table of PU Instances. Several Tables can be pre-defined, and only one is currently used by the HWA. A PU Instances Table is created and registered in the HWA Memory Section.

At any time, the HWA unit can execute 1 PU instance out of 32 selected PU instances from the selected Table.

The general procedure to execute PUs is:

- ▶ Select which HWA to use: HWA-1 or HWA-2 or both.
- ▶ Create HWA-PU instances with corresponding parameters.
- ▶ Create HWA-PU Table instances by specifying which PU instances to use per table. (Max: 32 PU instances per table)
- ▶ Select which Table to use for the HWA. This will be the “current” PU-Table. At the end of PU processing, the user can swap tables as needed.
- ▶ Start HWA-PUs by enabling one or several PUs out of the Table. The LOWER indexes of the Table have high priority if multiple PUs are enabled at the same time. (Index 0 has higher priority than index 1)

The HWA control information consists of the PU-Table instance selection and the PU start activation.

For each HWA, an ARM interrupt is generated at the end of each PU processing. HWA Status information can be checked to verify the PU completions.

For each PU, different types of input and output buffers can be used.

HWA Buffers

HWA-PUs use memory buffers from PS2 System Memory to process user inputs and parameters and to store output results. Depending on the type of HWA Processing Unit, different memory buffer types are needed to store user inputs and output results in memory. PS2 memory uses 32 bit words so all buffer types are formed from this size.

Each buffer is specified by the bufferHandle_t, which is usually used for the inputs and outputs of a PU. The buffer parameters are as follows:

Table 71. Buffer Parameters

Parameter	Note
principalBuf	Principal(main) buffer of the input/output data
alternateBuf	For ‘ping-pong’ mode only, the alternative buffer of the input/output data
numSamples	Number of data samples present in the buffer.

Buffer ‘ping-pong’ mode refers to the mode where there are two active buffers (principal/alternate). When the last entry in the principal buffer is read/written, the process will automatically jump to the first index of the alternative buffer. In this way, the buffers are used in a ‘ping-pong’ method where they are constantly switched between. It is then possible to update one buffer while the other is in use.

The input and output buffers for each PU is created within the PU configuration. An example of the buffer for a downsampler PU instance is shown in [Figure 191](#).

```
dsRealConfig_t dsConfig_downBy3 =
{
    .realIn = {
        .principalBuf = real_16_bit_down_by_3_input_data,
        .numSamples = 96
    },
    .realOut = {
        .principalBuf = real_16_bit_down_by_3_hwa_result,
        .numSamples = 32
    },
    .dataFormat = ADI_DRV_HWA_REAL_16BIT,
    .numSamples = 32,
    .ratio = 3,
    .phase = 0
};
```

Figure 191. Downsampler PU Instance Buffer Creation

The PU instance is created with the following configuration:

PROCESSOR SUBSYSTEM 2 (PS2)

- Real Input Buffer
 - Address of buffer (real_16_bit_down_by_3_input_data)
 - Number of samples (96)
- Real Output Buffer
 - Address of buffer (real_16_bit_down_by_3_hwa_result)
 - Number of samples (32)
- Format of both buffers: Packed Real 16 bit

Other parameters (numSamples, ratio, phase) are related to the PU instance and not related to the buffers.

Data Buffer Formats

The following formats for data buffers are supported. These formats can be used for PU input/output buffers as described in 'PU Use and Description'. Refer to the API driver description for details on how to use these buffer formats.

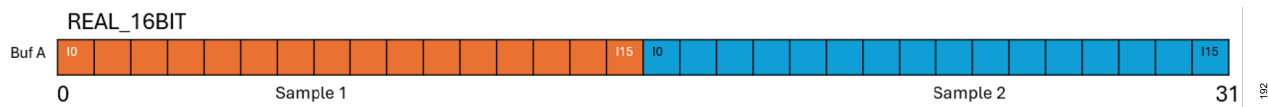


Figure 192. Real Data in 16 bit Packed Format

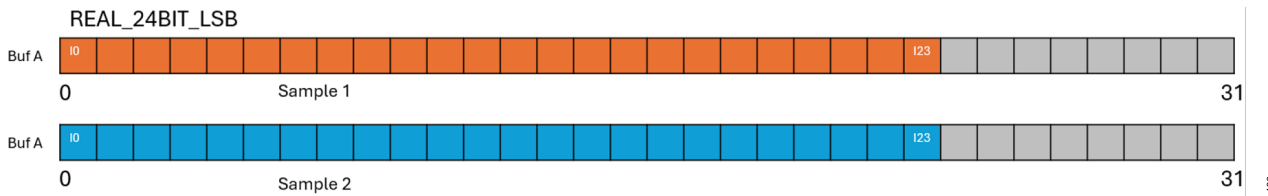


Figure 193. Real Data in 24 bit LSB Aligned Format

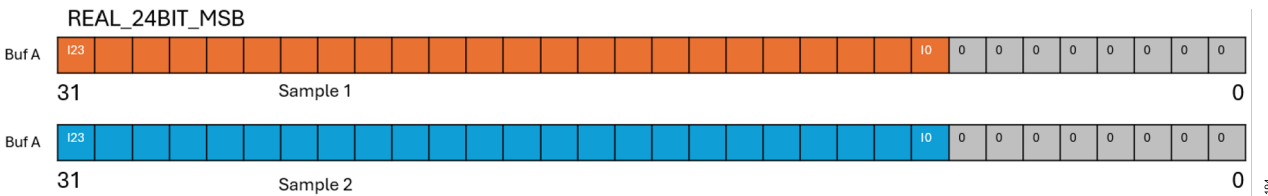


Figure 194. Real Data in 24 bit MSB Aligned Format (with Trailing 0 in LSB)

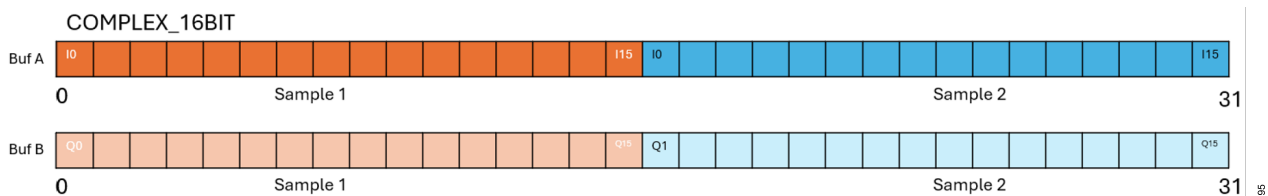


Figure 195. Complex Non Interleaved Data with One Each Buffer for Real and Imag Data Both Buffer of 16 Bitwidth in Packed Format

PROCESSOR SUBSYSTEM 2 (PS2)

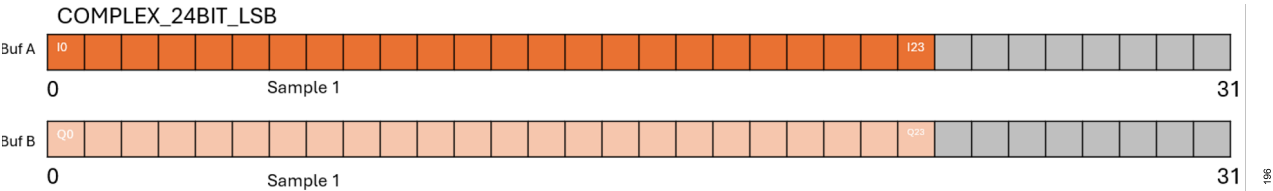


Figure 196. Complex Non Interleaved Data with One Each Buffer for Real and Imag Data Both Buffer of 24 Bitwidth LSB Aligned Format

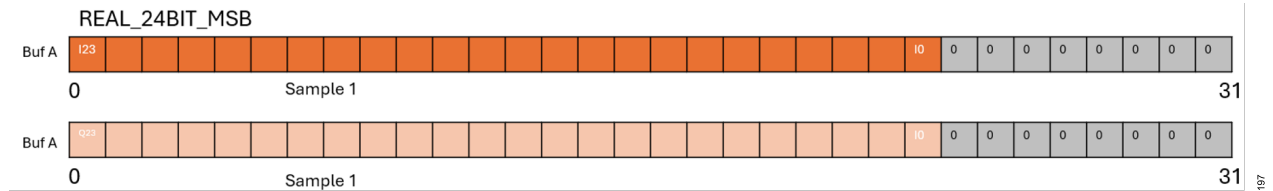


Figure 197. Complex Non Interleaved Data with One Each Buffer for Real and Imag Data Both Buffer of 24 Bitwidth MSB Aligned Format

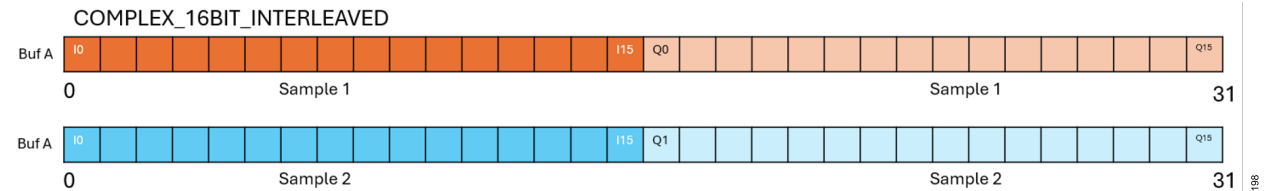


Figure 198. Complex Data in a Single Interleaved Buffer with Real and Imag Data in an Alternate Fashion with Both Real and Imag Data of 16 Bitwidth in a Packed Format

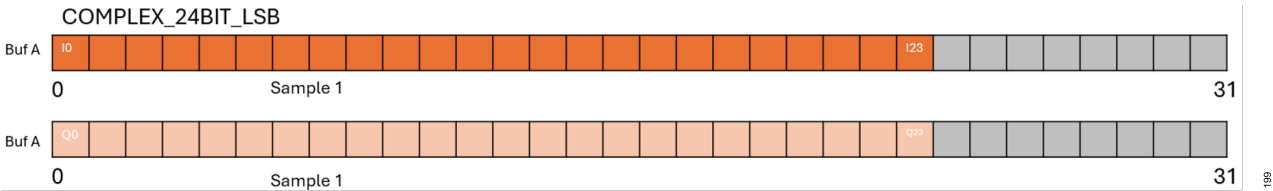


Figure 199. Complex Data in a Single Interleaved Buffer with Real and Imag Data in an Alternate Fashion with Both Real and Imag Data of 24 Bitwidth in LSB Aligned Format

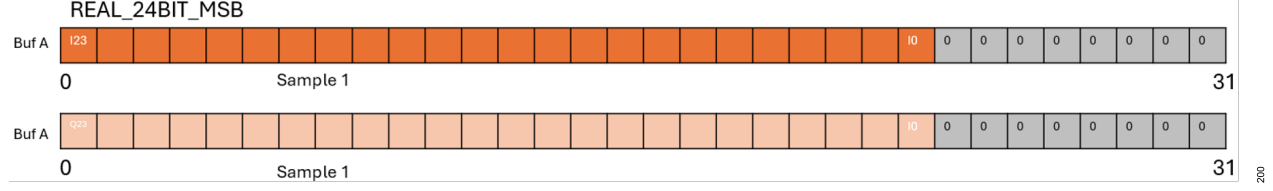


Figure 200. Complex Data in a Single Interleaved Buffer with Real and Imag Data in an Alternate Fashion with Both Real and Imag Data of 24 Bitwidth in MSB Aligned Format

PROCESSOR SUBSYSTEM 2 (PS2)

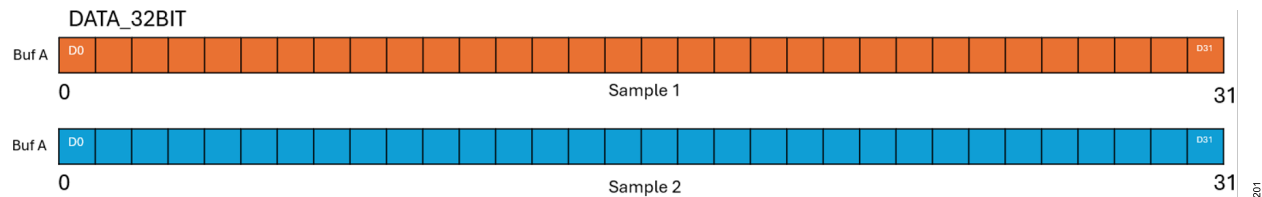


Figure 201. 32 bit Data Format

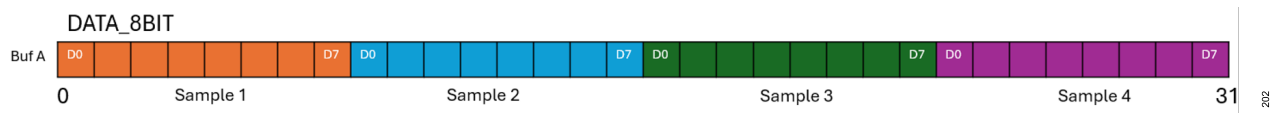


Figure 202. Data in 8 bit Packed Format



Figure 203. Data in 16 bit Unpacked Sign Extended Format

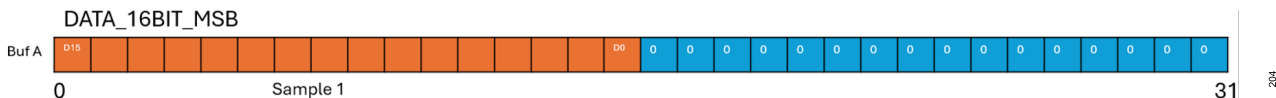


Figure 204. Data in 16 bit Unpacked MSB Aligned Format with All 0 in LSB

HWA Processing Units

Each HWA consists of several processing units (PUs) which implement specific DSP/Communication functions. A description of each of these PUs is given in this section.

A *PU program* is an instance of a processing unit. Any number of PU programs can be created e.g. 5 downsampler PU programs with different parameters can be created. A hardware accelerator can only run one PU program at a time.

Each PU will have one or multiple drivers associated with it used to configure the PU. Each driver will generally have the same structure. For the below, “PU” refers to the processing unit name: downsampler, crc, mult, fir, iir, etc.

PU instance create: A HWA-PU instance is created using the following Driver

```
adi_driver_Result_e adi_PU_InstanceConfig(uint8_t hwaId, PUConfig_t *config, adi_hwaProgTable_Handle_t *progTable, uint32_t progTableIdx);
```

Table 72. PU Instance Create parameters

Parameter	type	Description
hwaId	uint8_t	Refer to the HWA instance: ► ADI_DRV_HWA_INST1 ► ADI_DRV_HWA_INST2
config	PUConfig_t	PU configuration structure
progTable	adi_hwaProgTable_Handle_t	Program table in which the PU instance needs to be updated.
progTableIdx	uint32_t	Table index in which the PU instance needs to be updated in the table.

PU instance update: Once a HWA-PU instance is created, its parameters can be update later during run time, using the same driver.

PROCESSOR SUBSYSTEM 2 (PS2)

A *PU table* is an ordered list of PU programs to be assigned to a hardware accelerator. A PU can hold up to 32 PU programs at a time. Any number of PU tables can be created, however, a HWA can be assigned one PU table at a time. The order of the PU table details which PU program runs in which order i.e. the PU program at index 0 runs and then index 1 and so on. When starting the PU table execution, it is possible to enable or disable any of the PU programs contained in it. An interrupt can be driven on completion of the PU table.

An Example PU Table is given:

Table 73. Example PU Table

Index	PU Program	Enabled
0	Downsampler_1	✓
1	NCO_1	x
2	Upsampler_1	✓
3	Vector_Multiply_1	✓
4	Downsampler_2	✓
5	Limiter_1	x

- ▶ The trigger_start value is set to 0x1D (00011101), where the bit positions indicate which processing units are active. In this case, the processing units at indices 0, 2, 3, and 4 are enabled when the Program Start API is called.
- ▶ When this table is run, the enabled PU programs will run in order, i.e. Downsampler_1, Upsampler_1, Vector_Multiply_1, Downsampler_2.

Processing units are independent of the execution of its program. In that case, the processing unit can have several instances that will not interfere between each other.

For the detailed information about the APIs used for creating, updating and executing the PU table, refer to the doxygen files and Bare metal examples as detailed in the Resources and Examples chapter.

PU Use and Description

In the following section, each of the supported PUs will be described. Each PU can be used in each hardware accelerator (HWA_1/HWA_2), unless noted otherwise.

A description of each of the input and output parameters is given. For detailed information on the drivers, refer to the doxygen file (**Unpacked CMSIS Pack* \ADRV910X PS2 FW Example \doc \html \index.html*.) and example code (**Unpacked CMSIS Pack* \ADRV910X PS2 FW Example \examples \baremetal \example_main.c*) as detailed in the [Resources and Examples](#) chapter.

Each PU can support either a single input/output buffer (such as for real data) and some PUs support two input/output buffers (such as for complex data).

The support buffer types for the single input/output PU buffers are given in 'HWA Buffers'. Not every PU can support every buffer type. The supported buffers for each PU is given in the driver description.

Real data:

Drivers for 'real' data (I-only) should use the buffer types marked as '_REAL_'. An example of this is the downsampler driver with real data.

```

/*
 * @brief Down Sampler Real driver object
 */
typedef struct dsRealConfig
{
    bufferHandle_t realIn;      /*!< Real input buffer handle */
    bufferHandle_t realOut;     /*!< Real output buffer handle */
    bufferFormat_e dataFormat; /*!< Data format for both input and output*/
    uint32_t numSamples;       /*!< Number of samples to be produced by hwa after 1 run of Downsampler*/
    uint32_t ratio;            /*!< Downsampling ratio */
    uint32_t phase;            /*!< Downsampling initial phase. This value can not be >= ratio */
    uint32_t drvContext[HWA_DOWN_SAMPLER_REAL_CONTEXT_SIZE]; /*!< memory reserved for Driver context. Do not initialize or modify this variable*/
}dsRealConfig_t;

```

Figure 205. Real Data Driver Example

Complex data:

Drivers for 'complex' data (I/Q) should use the buffer types marked as '_COMPLEX_'. An example of this is the downsampler driver with complex data.

PROCESSOR SUBSYSTEM 2 (PS2)

```

/*
 * @brief Down Sampler Complex driver object
 */
typedef struct dsComplexConfig
{
    union {
        bufferHandle_t realIn; /*!< Real input buffer handle: Since this is a union, use the pointer under this handle to point real input if complex data is n
        bufferHandle_t cplxIn; /*!< Complex input buffer handle: use the pointer under this handle instead, if complex data is interleaved */
    };
    union {
        bufferHandle_t realOut; /*!< Real output buffer handle: Since this is a union, use the pointer under this handle to point real output if complex data is
        bufferHandle_t cplxOut; /*!< Complex output buffer handle: use the pointer under this handle instead, if complex data is interleaved */
    };
    bufferHandle_t imagIn; /*!< Imaginary input buffer handle: use the pointer under this handle to point the imag input in case of non interleaved complex
    bufferHandle_t imagOut; /*!< Imaginary output buffer handle: use the pointer under this handle to point the imag output in case of non interleaved co
    bufferFormat_e dataFormat; /*!< Data format for both input and output*/
    uint32_t numSamples; /*!< Number of samples to be produced by hwa after 1 run of Downsampler*/
    uint32_t ratio; /*!< Downsampling ratio */
    uint32_t phase; /*!< Downsampling initial phase. This value can not be >= ratio */
    uint32_t drvContext[HWA_DOWN_SAMPLER_COMPLEX_CONTEXT_SIZE]; /*!< memory reserved for Driver context. Do not initialize or modify this variable*/
}dsComplexConfig_t;

```

206

Figure 206. Complex Data Driver Example

Other Data:

Drivers that use data of a different format (not I-only or I/Q), should use the buffer types marked as `_DATA_`. An Example of this is the interleaver modulo driver:

```

/*
 * @brief Interleaver modulo driver object
 */
typedef struct interleaverModuloConfig
{
    bufferHandle_t bufIn; /*!< Input buffer handle */
    bufferHandle_t bufOut; /*!< Output buffer handle */
    bufferFormat_e dataFormat; /*!< Data format for both input and output buffers. Allowed Data formats for data are ADI_DRV_HWA_DATA-8BIT for Interleaver byte
    uint32_t modulo; /*!< Modulo value */
    uint32_t numSamples; /*!< Number of samples to be processed by the Interleaver modulo instance */
    intTypes_e intType; /*!< Interleaver type. ADI_DRV_HWA_INTL_BYTE for byte interleaver and ADI_DRV_HWA_INTL_BIT for bit interleaver */
    uint32_t drvContext[HWA_INTERLEAVER_CONTEXT_SIZE]; /*!< memory reserved for Driver context. Do not initialize or modify this variable*/
}interleaverModuloConfig_t;

```

207

Figure 207. Other Data Driver Example

Program Based PUs

The following functions are program-based processing units. Each function can be instantiated many times with different input & output parameters.

Down Sampler

Down sampling is a digital signal processing technique used to reduce the sample rate of a signal while maintaining its essential information. The Down-Sampler processing unit reads data samples from input Memory Buffers and writes out data samples to output Memory Buffer by skipping some data samples using user specified down sample ratio.

This function performs a rate reduction without filtering. If N is the down ratio. It keeps 1 of N samples in the output.

$$\text{numSampleOut} = \frac{\text{numSampleIn}}{\text{downRatio}} \quad (27)$$

Example: 16 input data samples and Down-Sample Ratio = 4 (initial phase = 0)

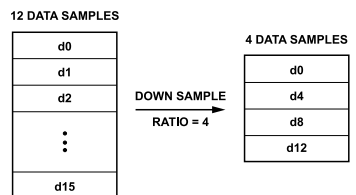


Figure 208. Down Sampler Example

Besides of buffer inputs and outputs, there are only 3 parameters for a Down-Sampler:

Table 74. Down Sampler Parameters

Parameter	Range	Description
numSample	[1: 65535]	Number of output samples

PROCESSOR SUBSYSTEM 2 (PS2)

Table 74. Down Sampler Parameters (Continued)

Parameter	Range	Description
ratio	[2: 65535]	Down ratio: Drop N-1 samples from input buffer
phase	[0 : ratio-1]	Downsampling initial phase. The phase where the first sample is taken from. Subsequent samples will be kept at the same interval. Use this to decide which sample to keep.

Limitations:

- Down sample ratio = 1 is not supported because it will have no effect on the input buffer.

Up Sampler

Up sampling is a method used to increase the sampling rate of the signal. The Up-Sampler processing unit reads data samples from input Memory Buffers and writes out data samples to output Memory Buffer by inserting zero data samples using user specified up sample Ratio.

Performs a rate change without filtering. For an up-sample ratio N, Up sampler inserts N-1 zeros between samples.

$$numSampleOut = numSampleIn * upRatio \quad (28)$$

Example: 4 input Data samples and Up-Sample Ratio = 3

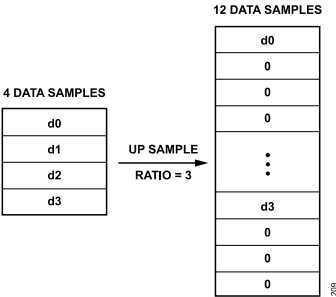


Figure 209. Up Sampler Example

Besides of buffer inputs and outputs, there are only 2 parameters for an Up-Sampler:

Table 75. Up-Sampler parameters

Parameter	Range	Description
numSample	[1: 65535]	Number of input samples
ratio	[2: 65535]	Up ratio: Insert (N-1) zero samples from input buffer

Limitations:

- Up ratio=1 is not supported because it will have no effect on the input buffer.

Multiply

The PU Multiplier has two drivers associated with: Vector multiply and scalar multiply. These drivers can be used independently from each other.

Vector Multiply

This function does an element by element multiply on two vectors to produce an output vector. The input vectors can be real or complex. Refer to the driver description for supported formats.

Performs the fixed-point functional equivalent of:

$$y(k) = x_1(k) x_2(k) \quad \forall k = 1 \text{ to } N \quad (29)$$

PROCESSOR SUBSYSTEM 2 (PS2)

Scalar Multiply

This function does an element by element multiply of one vector by a constant (scalar) to produce an output vector. The input vector can be real only. Refer to the driver description for supported formats.

Performs the fixed-point functional equivalent of:

$$y(k) = x_1(k) M \quad \forall k = 1 \text{ to } N, \text{ where } M \text{ is a constant} \quad (30)$$

Finite Impulse Response (FIR)

A Finite Impulse Response (FIR) filter is a type of digital filter characterized by a finite duration of the impulse response. It processes input data samples using a set of coefficients (filter taps) to produce output samples. FIR processing unit Implements a fixed-point finite impulse response filter. The input data and coefficients can be real or complex. Refer to the driver description for supported formats.

It performs the fixed-point functional equivalent of:

$$y(k) = \sum_{n=0}^{N-1} c_n x(k-n) \quad (31)$$

where c_n are the coefficients.

Note:

- If the impulse response of the filter is $h(n) \forall n = 0, 1, \dots, N-1$ then the coefficients to the FIR processing unit are in the order $h(N-1), h(N-2), \dots, h(1)$.

The following table shows valid FIR input/coefficients/output combinations:

Table 76. FIR input/output combinations

Input	Coefficients	Output
real	real	real
complex	real	complex
complex	complex	complex

Types of FIR processing units supported.

Table 77. Types of FIR processing units supported

Type	API	Description
FIR-REAL COEF-REAL	adi_firrealcoefreal_InstanceConfig()	► Real input/output with real coefficients standard FIR
FIR-REAL COEF-REAL FAST	adi_firrealcoefrealfast_InstanceConfig()	► Real input/output with real coefficients 'fast' FIR ► 'Fast' refers to using two MAC (Multiply-Accumulate) units to speed up the processing time by half. But this comes with 2 compromises: 1. If number of taps are odd, user need to have coef buffer that can have one more coef. Eg: if 31 taps are used coef buffer should have memory allocated for 32 taps). 2. If asymmetric coef are used, then user need to reverse the order of the coef and pass the coef buffer to the driver.
FIR-COMPLEX COEF-REAL	adi_fircplxcoefreal_InstanceConfig()	► Complex input/output with real coefficients standard FIR
FIR-COMPLEX COEF-COMPLEX	adi_fircplxcoefcplx_InstanceConfig()	► Complex input/output with complex coefficients standard FIR

FIR DECIMATOR

It is the process of reducing the sampling rate of a signal.

FIR DECIMATOR involves two steps: FIR filtering and then downsampling by a ratio of Decimating factor D.

PROCESSOR SUBSYSTEM 2 (PS2)

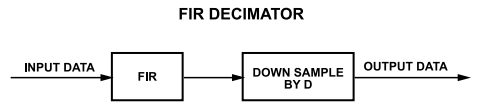


Figure 210. FIR Decimator

FIR-Decimator supports all the 4 types of FIR drivers mentioned previously.

In addition to FIR parameters, FIR Decimator has one extra parameter:

- decRatio: It denotes the decimation factor.

FIR INTERPOLATOR

Interpolation is the process of increasing the sampling rate of a signal. It's the opposite of decimation. In interpolation, new samples are generated in between the existing samples to increase the sample rate. The FIR INTERPOLATOR involves two steps: Up sampling the input data by an interpolator factor L and FIR filtering the data.

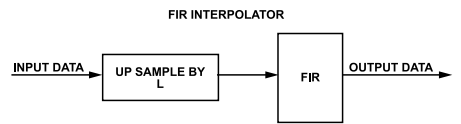


Figure 211. FIR Interpolator

FIR-Interpolator supports all the 4 types of FIR drivers mentioned previously.

In addition to FIR parameters, FIR Interpolator has one extra parameter:

- intRatio: It denotes the interpolator factor.

Infinite Impulse Response (IIR)

Infinite Impulse Response (IIR) filters are fundamental in digital signal processing, particularly used for tasks where specific frequency components need to be enhanced or suppressed. An IIR filter of any order is characterized by its recursive nature, meaning its output depends not only on the current and previous input values but also on its own past output values.

IIR filters with an order greater than 2 require decomposition into 2nd and/or 1st order sections. This involves breaking down a higher-degree polynomial, which represents the recursive IIR filter, into a series of simpler first and second-order polynomials. These simpler polynomials are then cascaded to reconstruct the original higher-order filter. The process is represented by the equation:

$$y(n) = -a_1 y(n-1) - \dots - a_k y(n-k) + b_0 x(n) + b_1 x(n-1) + \dots + b_k x(n-k)$$

The transfer function of this system is given by:

$$\frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \dots + b_k z^{-k}}{1 + a_1 z^{-1} + \dots + a_k z^{-k}} \quad (32)$$

$$= k' \frac{1 + \beta_1 z^{-1} + \dots + \beta_{k-1} z^{-(k-1)}}{1 + a_1 z^{-1} + \dots + a_k z^{-k}} \quad (33)$$

$$= k \frac{b_{10} + \beta_{11} z^{-1} + \dots + \beta_{12} z^{-k}}{1 + a_{11} z^{-1} + \dots + a_{12} z^{-k}} \dots \frac{\beta_{h1} + \beta_{h1} z^{-1} + \dots + \beta_{h2} z^{-k}}{1 + a_{h1} z^{-1} + \dots + a_{h2} z^{-k}} \quad (34)$$

Note:

- Here, each section of the equation represents a 2nd order IIR filter, and the implementation typically uses a 2nd order transpose form.
- The transfer function of each 2nd order section is given below.

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \quad (35)$$

PROCESSOR SUBSYSTEM 2 (PS2)

► The coefficients of the filter are as shown in the figure below. Note that the coefficients are $-a_1$ and $-a_2$.

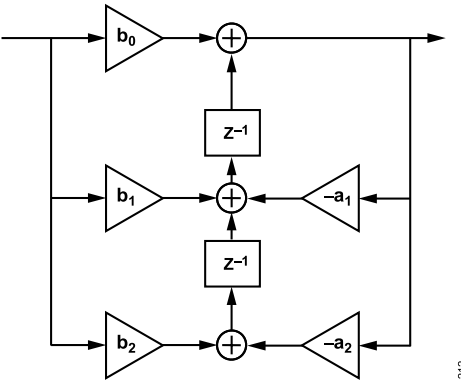


Figure 212. IIR-Real Parameters

Table 78. IIR Parameters

Parameters	Description
realIn/realOut	Input/output buffers
inputFormat/ outputFormat	Input/output buffers formats
numeratorCoef	Array with values of B0, B1 and B2
denominatorCoef	Array with values of A2 and A3. A1 is const : 0x100000 so not required to provide.
gain	Value of K to be assigned
states	Array with values of X1 and X2
numSample	Number of input samples

Bit Transpose

The Bit Transpose algorithm rearranges a bit matrix (an array of bits) by switching its rows and columns along the diagonal, resulting in a new matrix. This process flips a matrix so that rows become columns and vice versa.

Requirements:

- The physical memory width is 32 bits.
- The size of the matrix to be transposed is customizable in terms of both rows and columns, which can be of different sizes.
- The starting point for transposition is at the top right corner of the memory array, with the least significant bit (LSB) justified.
- Each row's first column bit starts at the LSB position in physical memory.
- Unused bits in each row remain unpopulated.

Example: Input: 6x10 matrix → (Bit Transpose) Output : 10x6 matrix

PROCESSOR SUBSYSTEM 2 (PS2)

MEMORY ARRAY ORGANIZED IN ROWS

[31:10]	9	8	7	6	5	4	3	2	1	0	
XXXXX...X	b09	b08	b07	b06	b05	b04	b03	b02	b01	b00	0
XXXXX...X	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	1
XXXXX...X	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	2
XXXXX...X	b39	b38	b37	b36	b35	b34	b33	b32	b31	b30	3
XXXXX...X	b49	b48	b47	b46	b45	b44	b43	b42	b41	b40	4
XXXXX...X	b59	b58	b57	b56	b55	b54	b53	b52	b51	b50	5

MEMORY ARRAY ORGANIZED IN COLUMNS

[31:6]	5	4	3	2	1	0	
XXXXXX...X	b50	b40	b30	b20	b10	b00	0
XXXXXX...X	b51	b41	b31	b21	b11	b01	1
XXXXXX...X	b52	b42	b32	b22	b12	b02	2
XXXXXX...X	b53	b43	b33	b23	b13	b03	3
XXXXXX...X	b54	b44	b34	b24	b14	b04	4
XXXXXX...X	b55	b45	b35	b25	b15	b05	5
XXXXXX...X	b56	b46	b36	b26	b16	b06	6
XXXXXX...X	b57	b47	b37	b27	b17	b07	7
XXXXXX...X	b58	b48	b38	b28	b18	b08	8
XXXXXX...X	b59	b49	b39	b29	b19	b09	9

213

Figure 213. Bit Transpose Example

Table 79. Bit Transpose Parameters

Parameter	Range	Details
rowNum	[1 to 32]	Number of rows of the input matrix
colNum	[1 to 32]	Number of columns of the input matrix
msb16	{0,1}	0 indicates LSB alignment. 1 indicates MSB alignment

Limitations:

- ▶ The current program supports up to 32x32 matrix bit transpose. For larger matrix, the program will run several times to handle different quadrants of the matrix.
- ▶ Any unused bits in the output buffer needs to be masked to see the correct result.

Byte Transpose

The Byte Transpose algorithm transposes an array of memory of bytes from rows to columns.

Requirements:

- ▶ Physical memory is 4-bytes wide (32-bits)
- ▶ Memory size transposed is programmable up to 32x32 bytes (rows/columns)
 - ▶ row and column sizes fully programmable
 - ▶ row and column sizes do not need to match.
- ▶ Bytes are densely packed in memory with new row column byte immediately following previous row last column byte (no byte skipping to start a new row)
- ▶ Unused ending bytes in physical memory are mapped as 'don't care'.
- ▶ Byte transpose is done on a 4-byte boundary.

For example, in a 4x7 and 6x7 matrix transposition:

- ▶ Each byte in a row is flipped to a columnar position.
- ▶ The 4-byte memory width means that bytes wrap around when a row is filled.
- ▶ The transposition respects the 4-byte boundary, ensuring data alignment.

PROCESSOR SUBSYSTEM 2 (PS2)

4 × 7 MEMORY ARRAY ORGANIZED IN ROWS
BYTES[3:0]

3	2	1	0	
b03	b02	b01	b00	0
b10	b06	b05	b04	1
b14	b13	b12	b11	2
b21	b20	b16	b15	3
b25	b24	b23	b22	4
b32	b31	b30	b26	5
b36	b35	b34	b33	6

4 × 7 MEMORY ARRAY ORGANIZED IN COLUMNS
BYTES[3:0]

3	2	1	0	
b30	b20	b10	b00	0
b31	b21	b11	b01	1
b32	b22	b12	b02	2
b33	b23	b13	b03	3
b34	b24	b14	b04	4
b35	b25	b15	b05	5
b36	b26	b16	b06	6

Figure 214. Byte Transpose Example

6 × 7 MEMORY ARRAY ORGANIZED IN ROWS
BYTES[3:0]

3	2	1	0	
b03	b02	b01	b00	0
b10	b06	b05	b04	1
b14	b13	b12	b11	2
b21	b20	b16	b15	3
b25	b24	b23	b22	4
b32	b31	b30	b26	5
b36	b35	b34	b33	6
b43	b42	b41	b40	7
b50	b46	b45	b44	8
b54	b53	b52	b51	9
xx	xx	b56	b55	10

6 × 7 MEMORY ARRAY ORGANIZED IN COLUMNS
BYTES[3:0]

3	2	1	0	
b30	b20	b10	b00	0
b11	b01	b50	b40	1
b51	b41	b31	b21	2
b32	b22	b12	b02	3
b13	b03	b52	b42	4
b53	b43	b33	b23	5
b34	b24	b14	b04	6
b15	b05	b54	b44	7
b55	b45	b35	b25	8
b36	b26	b16	b06	9
xx	xx	b56	b46	10

Figure 215. 6x7 Byte Transpose Example

Table 80. Byte Transpose Parameters

Parameter	Range	Details
rowNum	[1 to 256]	Number of rows of the input matrix
colNum	[1 to 256]	Number of columns of the input matrix

Limitations:

- The current program supports up to 16x16 matrix byte transpose. For larger matrix, the program will run several times to handle different quadrants of the matrix.

Arctan

This HWA processing unit computes the arctangent (inverse tangent) of complex numbers. Unlike the standard arctangent, which can only determine angles in two quadrants, the 4-quadrant arctangent can determine angles in all four quadrants of the Cartesian plane. This is essential when dealing with complex numbers, as they have both real and imaginary components.

Output Angles

1. Full angle Mode:

- Output Bit Width is 16-bit or 24-bit
- To convert to radians, multiply the output by π
- The output will be quantized and saturated based on output Bit Width
- The output ranges between -1.0 and $(1.0 - 2^{-(bw-1)})$ inclusive as S1.15 or S1.23

2. Octant Mode:

- Output Bit Width is 3-bits.
- Output is a vector of indices $[0, \dots, 7]$ corresponding to the octant starting at angle $(\text{index}) \times (\pi/4)$ and ending before angle $(\text{index} + 1) \times (\pi/4)$

PROCESSOR SUBSYSTEM 2 (PS2)

Besides the buffer inputs and outputs, there are only 2 parameters for a Arctan PU:

Table 81. Arctan Parameters

Parameter	Range	Description
dataBufIn	[-1.0, 1.0)	Input vector samples of either 16-bit or 22-bit resolution. ► For 16-bit resolution (Data format: ADI_DRV_HWA_BUFF_16BITS) ► For 22-bit resolution (data Format: ADI_DRV_HWA_BUFF_24BITS)
dataBufOut	[-1.0, 1.0)	► Full Angle Mode: Output vector samples of either 16-bit or 22-bit resolution. ► Octant Mode: Output vector samples of 3-bit resolution. (Data is packed 4 values per word32, Data Format: ADI_DRV_HWA_BUFF_8BITS Packed)
numSample	[1; 16383]	Number of output samples
iteration	[3; 16]	Number of Iterations for Angle mode
octant_mode	true/false	Boolean to select the Output mode: ► Angle: false ► Octant: true

Scrambler/Descrambler

They are designed to manipulate data using specific algorithms for various purposes like reducing predictability or interference. These components use a pseudo-random sequence generated by a Linear Feedback Shift Register (LFSR).

Scramblers (1 bit) are used to alter the original data pattern into a seemingly random pattern, making it less susceptible to interference and ensuring a more uniform distribution of 1's and 0's. The scrambling operation is typically done at the bit level (data Width = 1). The data would be packed into 32-bit words as it would be in the system and an array of 32-bit words is passed.

Two types of scramblers can be configured in the processing unit:

- Type 0: Scrambler with XORs inside the shift register. The polynomial of generator would be $p(x) = g_N x^N + g_{N-1} x^{N-1} + \dots + g_0 x^0$.
- Type 1: Scrambler with XORs outside the shift register. This generator produces the same repetitive sequence but with the polynomial $p(x) = g_0 x^N + g_1 x^{N-1} + \dots + g_N x^0$.

Note:

- When both Type 0 and Type 1 configurations begin with an identical initial value in their respective shift registers, the resulting sequences they generate will be similar but shifted in phase. It's feasible to determine an appropriate initial value for both types of generators to produce a specific sequence.
- The operation labelled "multiply" in the below figure is either an XOR for one bit mode or for eight-bit mode it can be a 1's complement or a 2's complement (when the bit out of the LFSR is a one).

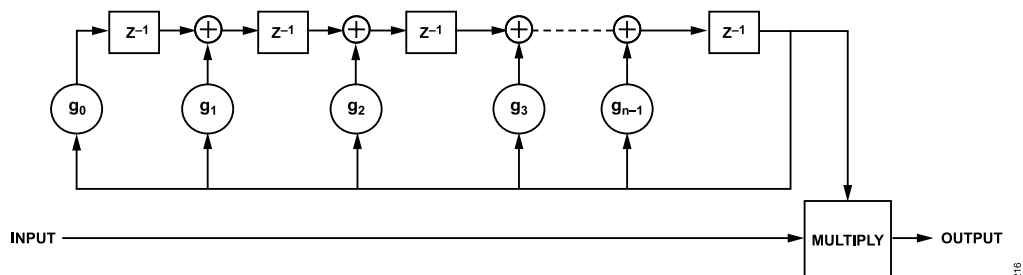


Figure 216. Scrambler with XORs inside the shift register: Type 0

PROCESSOR SUBSYSTEM 2 (PS2)

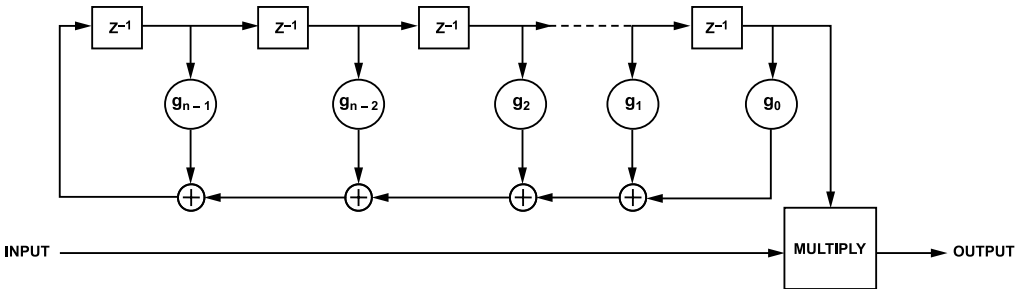


Figure 217. Scrambler with XORs outside the shift register: Type 1

Descramblers (1 byte) are used to revert data back to its original form. The descrambler on the receive side often works with soft-decision data (Data Width = 8) since the operation is typically before the decoder. In this case the function takes the input as an array of bytes, each byte representing soft-decision value for a bit.

Table 82. Descrambler (byte) parameters.

Name	Description
numByte	Total input message bytes
signSel	0: unsigned (math will be 1's compliment) 1: signed (math will be 2's compliment)

Limiter

For each sample, the Limiter checks if the sample is greater than an Upper-Limit or if it is lower than a Lower-Limit.

Generated output sample:

- If a sample is greater than the specified Upper-Limit, the output is replaced by the Upper-Limit value.
- If a sample is lower than the specified Lower-Limit, the output is replaced by the Lower-Limit value.
- Otherwise, keep the same sample as the input.

This function limits a signal to the specified minimum and maximum sample values.

Numerically Controlled Oscillator (NCO)

The Numerically Controlled Oscillator (NCO) generates a number of specified cosine and/or sine values for a given initial phase, frequency tuning word, phase offset, and scaler.

The Numerically Controlled Oscillator (NCO) uses 4 configuration parameters as inputs:

- Initial phase
- FTW: Frequency Tuning Word
- FTW Phase Offset
- Scaler: Scaling factor at the output of the NCO

The NCO result can be stored in several formats as follow:

1. Cosine only stored in 1 buffer (packed only)
2. Sine and Cosine packed stored in the same buffer (packed dual)
3. Sine and Cosine stored in separated buffers (packed only)

RSSI

The Receiver Signal Strength Indicator (RSSI) can be calculated for any buffer of samples. The RSSI is the sum of squares of input data.

PROCESSOR SUBSYSTEM 2 (PS2)

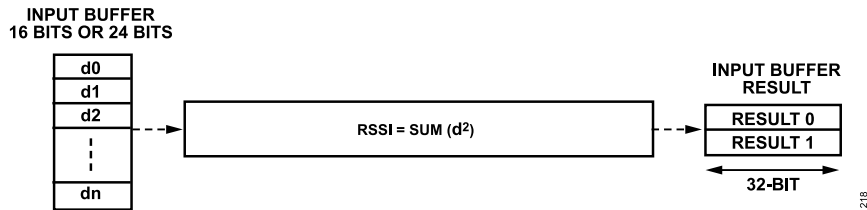


Figure 218. RSSI Functionality

Note:

- This processing unit only calculates the sum of squares of the input data $y = \sum_{n=0}^{N-1} x(n)x^*(n)$.
- **To calculate the average power:** The output from RSSI should be explicitly divided by N (Number of samples). The RSSI PU only outputs the sum of squares of the input data.

Log

The ‘Log’ function which exists in PS2 is not a typical PU like the other PUs describe in this section. The ‘Log’ block does not operate on buffers and can’t be added to the program table. However, the ‘Log’ function is implemented in a hardware block. The input takes one value at a time and returns either log10 or log2 or both output values.

Table 83. Log HWA parameters

Parameters	Details
Input	range [0.0, 2.0). Format U1.31
Output	► log value of input argument in S6.10 ► returns the minimum value (0x8000) which corresponds to -32.0 for a zero input. ► If run for dual log2 and log10 outputs, the log2 result will be in bits[31:16] and the log10 result will be in bits[15:0], otherwise the result will be in bits[15:0].

CRC Encoder/Decoder

The Linear Feedback Shift Register (LFSR) based CRC encoder/decoder accelerator takes a message $M(x)$, a polynomial $G(x)$ of degree n and calculates the remainder $R(x)=(M(x).2^n, \text{Mod}(G(x)))$.

Operations are done using polynomial arithmetic modulo 2. The generated n-bit CRC code can optionally be passed through a mask as specified by the standard before the codeword, $[M(x).2^n: R(x)]$, is sent by the transmit.

The same accelerator can be used by the receiver (Decoder) to compute the remainder $R'(x)$ and compares it with $R(x)$ for any errors.

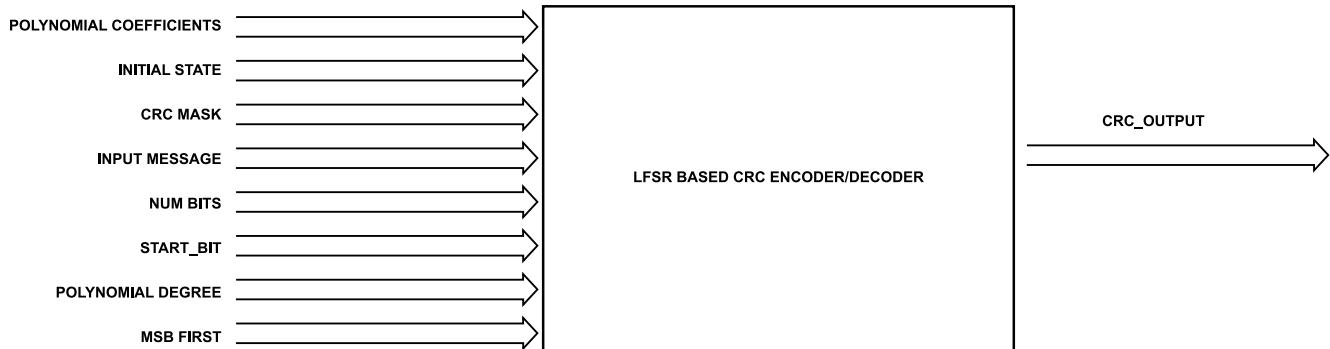


Figure 219. CRC

Refer to the driver description for formatting of the input/output parameters.

PROCESSOR SUBSYSTEM 2 (PS2)

Vector Addition/Subtraction:

Vector Add processing units adds 2 input vectors, real or complex of data format 16-bits (S1.15) and 24-bits (S1.23) to generate an output vector.

Vector Sub processing units subtracts 2 input vectors, real or complex of data format 16-bits (S1.15) and 24-bits (S1.23) to generate an output vector.

SUM

Calculates the sum of all the elements of a vector.

$$y = \sum_{n=0}^{N-1} x(n) \tag{36}$$

Integrator

The Integrate function provides the sum of current input sample with previous computed sum. The input vector can be real or complex. Supported data formats are 16 bits (S1.15) and 24 bits (S1.23).

Performs the fixed-point functional equivalent of:

$$y(n) = y(n - 1) + x(n) \tag{37}$$

Reframer

A Frame of bits consists of a set of N bits stored in Memory of 32bits per word. A Frame can be in one word or uses several words. It can also be MSB or LSB Valid data in memory.

This REFRAMER processing unit reads a FRAME from an input Buffer, and shifts it left or right, then writes it back to an output Buffer.

Reframer examples:

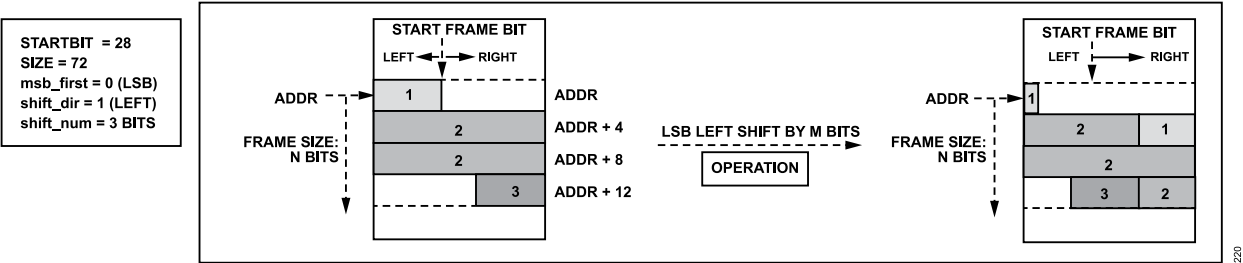


Figure 220. Reframer LSB Left Shift

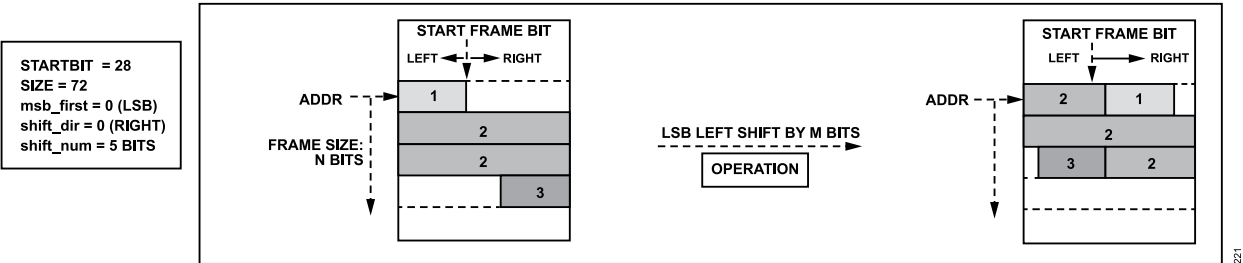


Figure 221. Reframer LSB Right Shift

Reframer Parameters

Table 84. reframer Parameters

Parameters	Description
startBit	Starting bit position of the frame
Size	Number of bits in the Frame

PROCESSOR SUBSYSTEM 2 (PS2)

Table 84. reframer Parameters (Continued)

Parameters	Description
Msb_first	0 (LSB), 1 (MSB)
Shift_dir	► For LSB -> 0 (Left), 1 (Right) ► For MSB -> 0 (Right), 1 (Left)
Shift_num	Number of bits to shift

Congruent PN Generator

The congruent PN (phase-noise) generator is a HWA specific to P.25 applications. This processing unit operates using a 12-bit input word (U_0) and produces six output vectors denoted as m_1 through m_6 . These vectors collectively generate a 114-bit PN sequence.

Functionality:

- Input Word (u_0): The accelerator takes a 12-bit input word, U_0 . It is assumed to be LSB justified with upper 20 bits set to 0. (U0.32)
- Output Vectors (m_1 to m_6): It generates six output vectors, each represented as U0.32 and LSB justified.
 - m_1 , m_2 , and m_3 are 23 bits each, with the upper 9 bits set to 0.
 - m_4 , m_5 , and m_6 are 15 bits each, with the upper 17 bits set to 0.
- PN Sequence Generation Equations:
 - The PN sequence is generated using the following equations:
 - $P_0 = 16 \cdot U_0$
 - $P_n = (173 \cdot P_{(n-1)} + 13849) \bmod(65536) \quad 1 \leq n \leq 114$
 - Bit Representation:
 - Let $P_n(15)$ represent bit 15 of the n^{th} congruent PN sequence.
 - The output vectors are constructed as follows:
 - m_1 : Comprises the first 23 bits: $[P_1(15), P_2(15), P_3(15), \dots, P_{23}(15)]$
 - m_2 : Contains the next 23 bits: $[P_{24}(15), P_{25}(15), P_{26}(15), \dots, P_{46}(15)]$
 - m_3 : Contains the next 23 bits: $[P_{47}(15), P_{48}(15), P_{49}(15), \dots, P_{69}(15)]$
 - m_4 : Contains the next 15 bits: $[P_{70}(15), P_{71}(15), P_{72}(15), \dots, P_{84}(15)]$
 - m_5 : Contains the next 15 bits: $[P_{85}(15), P_{86}(15), P_{87}(15), \dots, P_{99}(15)]$
 - m_6 : Contains the last 15 bits: $[P_{100}(15), P_{101}(15), P_{102}(15), \dots, P_{114}(15)]$

Correlator

This PU performs a fixed-point correlation of two vectors. Real and complex data are supported with formats S1.15 and S1.23. If complex, the second input can be optionally conjugated.

Correlator performs the fixed-point functional equivalent of:

$$y = \sum_{n=0}^{N-1} x_1(n) x_2(n) \quad (38)$$

Or

$$y = \sum_{n=0}^{N-1} x_1(n) x_2^*(n) \quad (39)$$

Parameters:

- useConj parameter is used to select conjugate of Input Data 2 or Not (in case of complex data).
- Output buffer is also used as Initial/Intermediate State of the Correlator. It needs to be initialized with the state value when created. In case of 2 output buffer mode both the output buffers need to be initialized with corresponding Real and imaginary state values.

PROCESSOR SUBSYSTEM 2 (PS2)

Linear Block Code Generator: LBCG

LBCG processing unit performs a bit matrix multiply in GF(2). It can be used to multiply information bits by a generator matrix to produce linear block code codeword and it can also be used to multiply a parity check matrix by a codeword to produce a parity check vector or syndrome.

All linear block codes can be generated by multiplication with a generator matrix.

$$c = xG$$

where x is a row vector of data to be encoded, G is the generator matrix and c is the code word.

Generator Matrix: The elements of Generator Matrix $G_{i,k}$ are transposed, right and top justified when passing to the LBCG driver. The table below shows how a 4x7 matrix would be filled in memory.

Table 85. 4x7 LBCG Generator Matrix

	d31	d30			d4	d3	d2	d1	d0
word 0	0	0		0	0	G _{0,6}	G _{1,6}	G _{2,6}	G _{3,6}
word 1	0	0		0	0	G _{0,5}	G _{1,5}	G _{2,5}	G _{3,5}
word 2	0	0		0	0	G _{0,4}	G _{1,4}	G _{2,4}	G _{3,4}
word 3	0	0		0	0	G _{0,3}	G _{1,3}	G _{2,3}	G _{3,3}
word 4	0	0		0	0	G _{0,2}	G _{1,2}	G _{2,2}	G _{3,2}
word 5	0	0		0	0	G _{0,1}	G _{1,1}	G _{2,1}	G _{3,1}
word 6	0	0		0	0	G _{0,0}	G _{1,0}	G _{2,0}	G _{3,0}
word 7	0	0		0	0	0	0	0	0
word 8	0	0		0	0	0	0	0	0

If the input code word length is length N_r and output code word length is N_c . The PU performs the following calculations to form the output code word w . Here w_i are the bits of the output code word, v_i are the bits of the input code word and $d_{i,k}$ are the bits of the k^{th} row and i^{th} column of the generator matrix data.

$$w_k = \sum_{i=0}^{N_c-1} v_i * d_{k,i} \quad (40)$$

Example: Encoding input data "1011" with a Hamming (7,4,3) code. Here $N_r=4$, $N_c=7$

input data word: 0000000000000000000000001011

Table 86. Generator matrix

Generator matrix						
d6	d5	d4	d3	d2	d1	d0
0	0	0	1	1	0	1
0	0	0	0	1	1	1
0	0	0	1	1	1	0
0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0

Output Calculation:

Table 87. Output Calculation

						</

PROCESSOR SUBSYSTEM 2 (PS2)

Table 87. Output Calculation (Continued)

		v3	v2	v1	v0		
	input	1	0	1	1		
k	d4	d3	d2	d1	d0	Output	
2	0	1	1	1	0	0	$w_2 = d_{2,3}v_3 + d_{2,2}v_2 + d_{2,1}v_1 + d_{2,0}v_0$
3	0	0	0	0	1	1	$w_3 = d_{3,3}v_3 + d_{3,2}v_2 + d_{3,1}v_1 + d_{3,0}v_0$
4	0	0	0	1	0	1	$w_4 = d_{4,3}v_3 + d_{4,2}v_2 + d_{4,1}v_1 + d_{4,0}v_0$
5	0	0	1	0	0	0	$w_5 = d_{5,3}v_3 + d_{5,2}v_2 + d_{5,1}v_1 + d_{5,0}v_0$
6	0	1	0	0	0	1	$w_6 = d_{6,3}v_3 + d_{6,2}v_2 + d_{6,1}v_1 + d_{6,0}v_0$
7	0	0	0	0	0	0	

The Output data word: 00000000000000000000000001011000

LBCG parameters

Table 88. LBCG Parameter description

Parameters	Description
dataBufIn	LBCG input data buffer handle (32 bit or 64 bit)
matrixBufIn	LBCG input Generator Matrix (Parity Check Matrix) buffer handle (either 32 of 32-bit words or 64 of 64-bit words)
dataBufOut	LBCG output data buffer handle
numSample	Number of samples to generate
matrix_size	Size of the Generator Matrix
Mode	0=32bits mode 1=64bits mode (To be activated if the matrix is higher than 32x32)

PRBS

PRBS processing unit generates pseudo-random bit sequence using a linear feedback shift register (LFSR) with a polynomial. The PRBS PU supports two types of configurations as shown in the figures below.

- Type 0 (Galois): PRBS Generator with XORs inside the shift register. The polynomial of generator would be

$$p(x) = g_N x^N + g_{N-1} x^{N-1} + \dots + g_0 x^0.$$

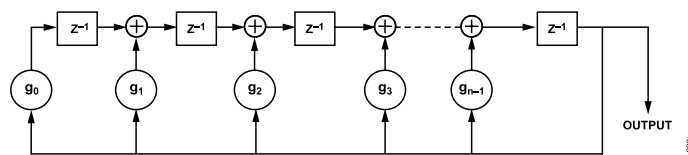


Figure 222. Type 0

- Type 1 (Fibonacci): PRBS Generator with XORs outside the shift register. This generator produces the same repetitive sequence but with the polynomial $p(x) = g_0 x^N + g_1 x^{N-1} + \dots + g_N x^0$.

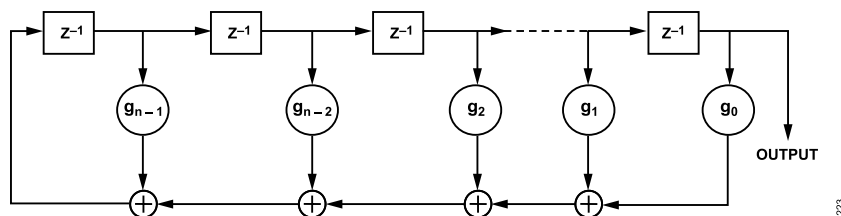


Figure 223. Type 1

PROCESSOR SUBSYSTEM 2 (PS2)

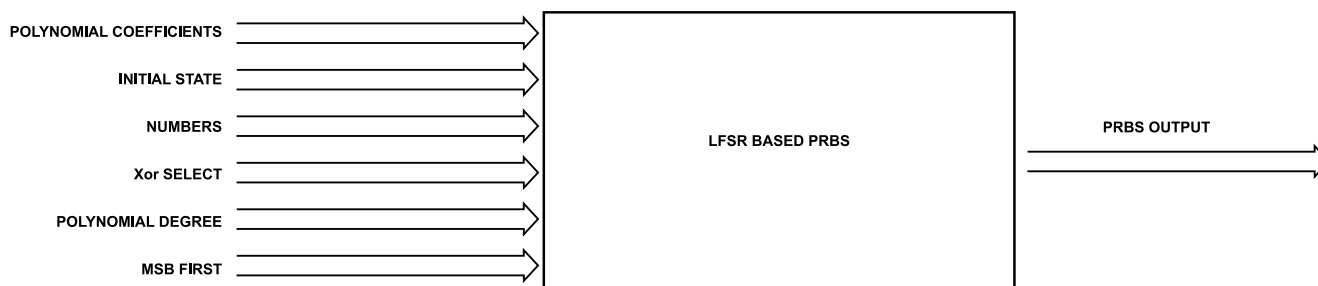


Figure 224. LFSR based PRBS

For details on formatting the input/output parameters, refer to the driver description.

Interleaver/De-Interleaver

Interleaving is an error correction algorithm to reduce burst errors in transmitted data. Two interleave modes are supported:

- Modulo: Total number of bits interleaved (Interleave depth)
- Look Up Table (LUT): Look up table is used to hardcode interleaved index.

Interleaver(bit) processing unit is used to reorder the input data such that contiguous bits in memory are spaced apart. Both modulo mode and LUT mode are supported.

De-Interleaver (byte) processing unit is used to revert interleaved data back to its original form. Both modulo mode and LUT mode are supported.

Note: The bit_mode interleaver/deinterleaver needs to reverse the DMA and Tetra modulo $\text{mod}(a, K)$, $\text{mod}(m, S)$ equation parameters to accommodate the ADRV910x HWA implementation.

- DMR/Tetra Standard:

Source address is incremental, destination address is modulo equation.

- ADRV910x Hardware Implementation:

Source address is modulo equation, destination address is incremental.

The below table shows the modulo equation parameters for modulo bit_mode and byte mode. The interleave parameters are from the DMR and Tetra Standards. The deinterleave parameters have been derived for each of the modes listed in the table.

Table 89. Interleaver/Deinterleaver Modulo Parameters

DMR/Tetra Standard	DMR/Tetra Spec		bit mode		byte mode	
	Interleave Parameters		interleave	deinterleave	interleave	deinterleave
	a	mod K	$\text{mod}(a, K)$	$\text{mod}(a, k)$	$\text{mod}(a, K)$	$\text{mod}(a, K)$
DMR						
DMR BPTC (196,96)	181	196	(13, 196)	(181, 196)	(181, 196)	(13, 196)
DMR BPTC single burst variable length	17	32	(17, 32)	(17, 32)	(17, 32)	(17, 32)
Tetra Phase modulation	m	mod S	$\text{mod}(m, S)$	$\text{mod}(m, S)$	$\text{mod}(m, S)$	$\text{mod}(m, S)$
BSCH	11	120	(11, 120)	(11, 120)	(11, 120)	(11, 120)
SCH/HD, BNCH, STCH (dqpsk)	101	216	(77, 216)	(101, 216)	(101, 216)	(77, 216)
SCH/HU (dqpsk)	13	168	(17, 168)	(13, 168)	(13, 168)	(17, 168)
SCH/F (dqpsk)	103	432	(151, 432)	(103, 432)	(103, 432)	(151, 432)
SCH-P8/HD (d8psk)	19	324	(307, 324)	(19, 324)	(19, 324)	(307, 324)
SCH-P8/HU (d8psk)	17	252	(89, 252)	(17, 252)	(17, 252)	(89, 252)
SCH-P8/F (d8psk)	23	648	(479, 648)	(23, 648)	(23, 648)	(479, 648)

The section below elaborates on finding the deinterleave parameters.

PROCESSOR SUBSYSTEM 2 (PS2)

Finding the deinterleaver step “a” or “m” parameter for mod() equation Note that DMR mod(a, K) is equivalent to Tetra mod(m, S) Need to solve for x in modulo K arithmetic, so: $y = ax \bmod K$ is the modulo equation

- a is the interleaver step (constant)
- K is the interleaver depth (constant)
- x is the incrementing index 0 to K-1 (variable)
- y is the interleave index for each value of x

The deinterleave step is the value of x where $y = 1$

- $\text{mod}(ax, K) = 1$
- Need to solve for x to solve deinterleaver step size

A simple for loop can be used to solve for x where $\text{mod}(ax, K) = 1$ DMR mod(181, 196) interleaver equation is used as the example For $x=0:195$ $\text{tmp_y} = \text{mod}(x*181, 196)$; if ($\text{tmp_y}==1$) $\text{dintlv_a} = x$; end deinterleaver step = dintlv_a for deinterleaver $\text{mod}(ax, K)$

Interleaver(bit)/De-Interleaver(byte) parameters is shown in below table.

Table 90. Interleaver/De-Interleaver parameters

Parameters	Description
dataBufIn	Input data buffer handle ► Input message data in case of Interleaver ► Interleaved input data in case of De-Interleaver
dataBufOut	Interleaver modulo/LUT output data buffer handle (Interleaved data) ► Interleaved output data in case of Interleaver ► Original message data in case of De-Interleaver
modulo	Modulo parameter denotes the interleave depth
Size	Bit size of input/output vector
lutBufIn	LUT table buffer handle

Maximum Likelihood Decoder: ML Decoder

A maximum likelihood decoder processing unit uses log likelihood ratio values to select the most likely codeword. The input LLR is defined as

$$\Lambda(x) = \ln\left(\frac{p(y=0|x)}{p(y=1|x)}\right) \quad (41)$$

Where $p(y=0|x)$ is the probability of the data being a zero based on the received value of x.

For each codeword, the likelihood of that codeword is calculated as the sum of the LLR values for the bits in the codeword that are zero.

$$L = \sum_{C_k=0} \Lambda(x_k) \quad (42)$$

The codeword that has the largest likelihood value is returned as the output codeword.

Note that each LLR input is in S3.5 8-bit format.

ML decoder parameters

Table 91. ML Decoder Parameters

Parameters	Description
dataBufIn	Input message data buffer handle ► For mode = 32-bit, the step size is 1. ► For mode = 64-bit, the step size is 8. ► 4 LLR input can be stored in each 32-bit word. ► Supported format is NOT_PACKED 32-bit LSB format
maskBufIn	ML Decoder mask buffer handle

PROCESSOR SUBSYSTEM 2 (PS2)

Table 91. ML Decoder Parameters (Continued)

Parameters	Description
	▶ The size is 2 words, and the supported format is NOT_PACKED 32-bit LSB format. ▶ Each bit position indicates whether a specific soft data input should be masked.
coefBufIn	table of valid codewords ▶ Each code word is represented in 32-bit NOT_PACKED format. ▶ Size of the table = parameter 'size' of the mldecoder function
dataBufOut	Output data buffer handle
mode_64bits	▶ 0=32 bits mode ▶ 1=64 bits mode
size	Size of the code word.

Hard Syndrome Decoder (HS Decoder)

The input codeword is multiplied by the Parity Check Matrix to generate a syndrome. This syndrome is used as an index into the Syndrome Table. If the syndrome is correctable, the Syndrome Table will return the error vector, which is XORed into the input codeword to result in an error corrected output codeword. If the syndrome is not correctable, then bit 31 will be set in the output codeword to represent a detected but not correctable error.

Syndrome Table:

- ▶ An array of 31-bit error vectors and an invalid bit. Bits 30-0 are the error vector and bit 31 is the invalid bit. The entries in this table are indexed by the syndromes+ 1.
- ▶ The length of the table is equal to 2^P , where P is the number of parity check bits generated by the parity check matrix. It contains a maximum of 512 entries.

Parity Check Matrix:

- ▶ Parity check matrix is formatted for the Linear Block Code Generator and is used to generate the syndromes.
- ▶ The format of the parity check matrix ensures that the parity bits are positioned in the least significant bits (LSBs) of the output. This arrangement is essential because it aligns with the syndrome table's lookup requirements.
- ▶ **Note** that, similar to the parity check matrix (Generation matrix) in LBCG HWA driver, the elements of Parity Check Matrix are transposed, right and top justified when passing to the HS Decoder.

HS Decoder parameters

Table 92. HS Decoder Parameters

Parameters	Description
dataBufIn	Input code word of up to 31 bits are supported. The MSBit should be 0 and used as flag for dataBufOut.
matrixBufIn	Input parity check matrix (Similar to the Generator matrix in LBCG HWA the elements are transposed, right and top justified)
tableBufIn	Input Syndrome table with maximum of 512 entries and minimum of 2 entries.
dataBufOut	Corrected Codeword. If bit 31 is set, the Input Codeword could not be corrected.
size	Number of input message samples
matrixSize	Parity check matrix size.

Convolutional Encoder

This processing unit implements a generic bit-wise convolutional encoder with supporting rates 2,3,4,5, constraint length (K) 2 through 9, recursive codes, and automatic termination.

Functionality

- ▶ **Initial State:** The initial state of the convolutional encoder is configured by the User based on the type of encoding.
 - ▶ Non-Tail-Biting Code: The initial state is set to zero.

PROCESSOR SUBSYSTEM 2 (PS2)

- ▶ Tail-Biting Code: The last K-1 bits of the input block are repacked and loaded as the initial state. This ensures that the encoding can wrap around seamlessly.
- ▶ Continuation of a Previous Block: If the encoder is continuing from a previous encoding process, the end state of the previous block is used as the initial state for the current block.
- ▶ **Encoder Termination:** The encoder supports automatic termination to ensure that encoded blocks end in a known state (typically zero), which simplifies the decoding process.
 - ▶ Non-Recursive Codes: The encoder automatically appends K-1 zero tail bits to the input (implicitly, does not affect input buffer), extending the output by $R \times (K-1)$ bits.
 - ▶ Recursive Codes: The tail bits depend on the feedback path's configuration and the end state of the encoder before termination. These tail bits are dynamically generated to drive the internal state to zero.
- ▶ **Continuous Mode:** In continuous mode, the encoder processes input blocks without necessarily reaching a terminated state at the end of each block.
 - ▶ Uninterrupted Processing: Multiple sub-blocks of an input stream can be encoded in a sequence without resetting the encoder's state between blocks.
 - ▶ State Management: The end state of one block can be used as the initial state for the subsequent block, enabling continuous encoding.
 - ▶ Final Block Termination: The last block in a sequence can optionally be terminated, ensuring the entire sequence ends in a known state for decoding.

Controls

1. Rate (R): Determines the number of outputs per input bit, configurable from 2 to 5. This rate directly influences the coding redundancy and error correction capability.
2. Constraint Length (K): Represents the number of delay elements plus one, adjustable from 2 to 9. The constraint length affects the memory depth of the encoder, thereby impacting the complexity and performance of the error correction process.
3. Forward Polynomial (ff_poly): Configures the polynomial for each output bit, determining the combination of input bits that influence each output. This control is essential for defining the code's structure and error handling characteristics.
4. Feedback Polynomial (fb_poly): Sets the feedback path for recursive encoding, crucial for generating dependencies between consecutive input blocks, enhancing the encoder's ability to correct longer error sequences.
5. Invert Mask: Optionally inverts the output bits. This feature can be used to balance the signal's zero and one density.

Convolutional Encoder Parameters

Table 93. Convolutional Encoder Parameters

Parameters	Description
dataBufIn	Input data buffer handle
initBufIn	initial state of the LFSR input, and final state of the LFSR (on output)The LSbit (bit 0) is the newest input bit (the shift register is shifted left as each input bit is processed).Note that on output the final state will be will be always be the length of the entire LFSR (9 bits) for any K, but depending on K the some upper MSbits may be ignored by the CE engine.
cfgBufIn	Consists of 5 words. ▶ Word1: feed backward polynomial ▶ Word2: Feed Forward polynomial 1 & 2 -> FF_pol1=[8:0], FF_pol2=[24:16] ▶ Word3: FF polynomial 3 & 4 -> FF_pol3=[8:0], FF_pol4=[24:16] ▶ Word4: FF polynomial 5 -> FF_pol5 = [8:0] ▶ Word5: invert Mask Note that the polynomial bit order is LSbit (bit 0) to the lowest order term. For example if a generator poly is $1+D^2+D^3$ its polynomial configuration byte would be 0x0D = 0b1101.The LSbit (bit 0) controls inversion of the 1st output bit and bit R-1 controls the Rth output bit.
dataBufOut	output data buffer handle
numBits	Number of input bits
R	Sets the code Rate = $1/R$, Range: 2-5
K	Code Constraint length (Code uses K-1 delay elements)
terminate	▶ 1: Enables automatic termination

PROCESSOR SUBSYSTEM 2 (PS2)

Table 93. Convolutional Encoder Parameters (Continued)

Parameters	Description
	► 0: Disables automatic termination

Format Conversion

The format conversion driver converts 16bits or 24bits data buffer into different format.

Here is the summary of all the conversion available:

Table 94. Available Format Conversion

Data format	Input buffer format	Output buffer format
16 bits data format	16bits packed	Int32 LSB aligned with sign extension
	16bits packed	Int32 MSB aligned with 0s inserted on LSB
	Int32 LSB aligned with sign extension	16bits packed
	Int32 MSB aligned with 0s inserted on LSB	16bits packed
	16bits LSB notpacked	Int32 LSB aligned with sign extension
	16bits LSB notpacked	Int32 MSB aligned with 0s inserted on LSB
	16bits MSB notpacked	Int32 LSB aligned with sign extension
	Int32 LSB aligned with sign extension	16bits LSB notpacked
	Int32 MSB aligned with 0s inserted on LSB	16bits LSB notpacked
	Int32 LSB aligned with sign extension	16bits MSB notpacked
24 bits data format	24bits LSB notpacked	Int32 LSB aligned with sign extension
	24bits LSB notpacked	Int32 MSB aligned with 0s inserted on LSB
	16bits MSB notpacked	Int32 LSB aligned with sign extension
	Int32 LSB aligned with sign extension	24bits LSB notpacked
	Int32 MSB aligned with 0s inserted on LSB	24bits LSB notpacked
	Int32 LSB aligned with sign extension	16bits MSB notpacked

Limitation:

- Complex interleaved buffer conversion is not supported.

Memory Copy

Copy a buffer from one memory location to another without ARM processor intervention. Copy of one or two buffers at a time is supported.

Symbol Mapping

The symbol mapping is the process of converting digital data (bits) into modulated symbols that can be transmitted over a communication channel. The mapping depends on the modulation scheme used, where symbols represent changes in phase, frequency, or amplitude of the transmitted signal.

The PU uses LUT table to convert the input signal into Symbol.

PROCESSOR SUBSYSTEM 2 (PS2)

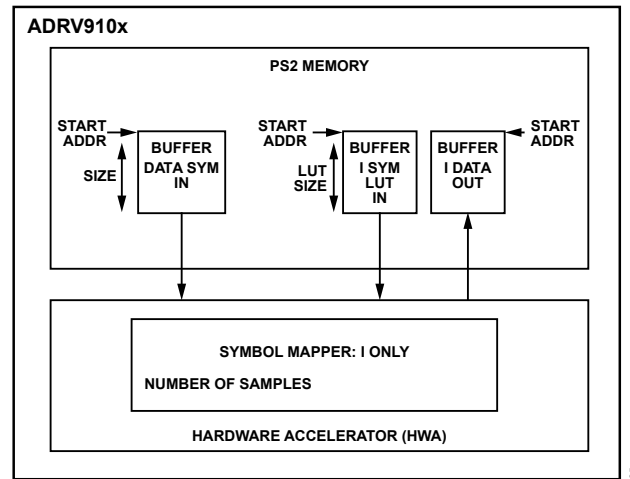


Figure 225. Symbol Mapping of One Input Buffer and One Output Buffer

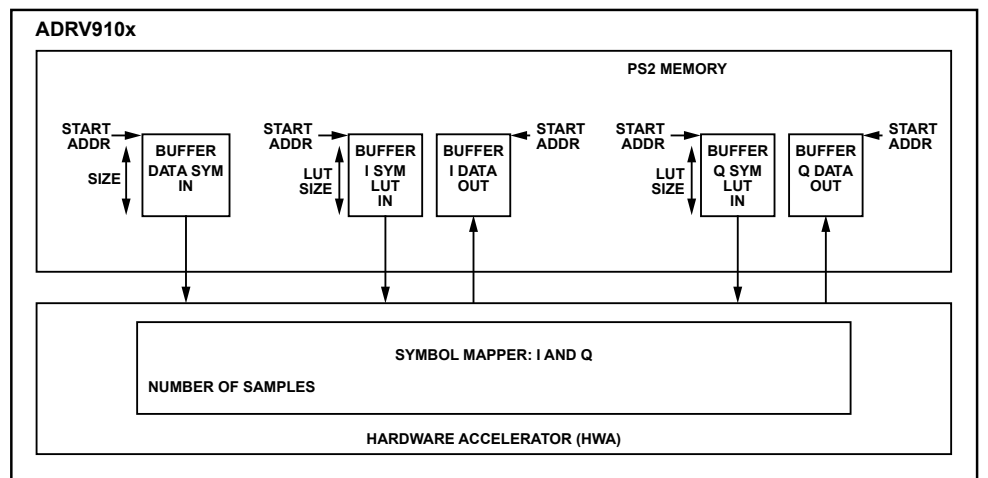


Figure 226. Symbol Mapping of One Input Buffer and One Output Buffer

Limitation:

- Available only in HWA1.
- Input buffer contains data that must be LSB aligned.

Min/Max

The minimum function returns the minimum value of a vector.

The maximum function returns the maximum value of a vector.

Repack (Reverse Byte/Bit)

This PU rearranges the bit order of incoming data streams, converting LSB-first data to MSB-first format or vice versa. Repack can also be used to convert data between different bit widths. Repack can provide a start offset to the data. The buffers will be repacked as per the formats described in [Data Buffer Formats](#).

Limitation:

The expected output buffer cannot exceed 15 words.

PROCESSOR SUBSYSTEM 2 (PS2)

Reed-Solomon Encoder/Detect

The Reed-Solomon encoding process begins by dividing the original data into blocks of symbols. The encoder then calculates and appends a set of redundant parity symbols to each data block. The number of parity symbols added determines the error-correcting capability of the code. This combination of original data symbols and parity symbols forms a codeword, which is then ready for transmission or storage.

Upon receiving the codeword, which may contain errors due to noise or interference, the Reed-Solomon decoder analyzes it to identify any discrepancies. Using the redundant parity symbols, the decoder corrects the identified errors, reconstructing the original data block. The corrected data block is then output, ensuring that the data integrity is maintained despite any errors that occurred during transmission or storage.

This encoding and decoding process ensures robust error correction, making Reed-Solomon codes highly effective in maintaining data accuracy in various applications.

Trellis Encoder (TCM Encoder)

This processing unit (PU) implements the P25/DMR Trellis Coded Modulation Rate $\frac{1}{2}$ and $\frac{3}{4}$ Encoders that encode a bitstream into dibits (pair of bits) ready for input to the interleaver as specified in:

1. P25 Air interface, Clause 7. for Rates $\frac{1}{2}$ and $\frac{3}{4}$ code.
2. DMR Air Interface, Annex B.2.4., for Rate $\frac{3}{4}$ code.

The TCM-E (Trellis Coded Modulation Encoder) processing unit has a parameter to select Rate $\frac{3}{4}$ or Rate $\frac{1}{2}$, and it supports only 1 input/output mode operating on packed bits ordered LSbit to MSbit.

Table 95. Trellis Encoder Input-Output combination for Rate $\frac{3}{4}$ and $\frac{1}{2}$

Rate	Input Block	Output Block
$\frac{3}{4}$	144 packed bits (4.5 U32s) or 48 tribits	196 packed bits (6 $\frac{1}{8}$ U32s) or 98 dibits or 49 constellation points (dibit pairs)
$\frac{1}{2}$	96 packed bits (3 U32s) or 48 dibits	

Look Up Tables (LUTs):

The user is recommended to use the predefined LUTs provided below. A 64-LUT is used for a rate of $\frac{3}{4}$, and a 16-LUT for a rate of $\frac{1}{2}$.

Table 96. Trellis Encoder Look Up Table (LUT)

Rate	LUT data	LUT input passed to the Trellis Encoder PU
$\frac{1}{2}$	Constellation_R12_LUT16[16]={0x8, 0x3, 0x4, 0xf, 0xb, 0x0, 0x7, 0xc, 0x6, 0xd, 0xa, 0x1, 0x5, 0xe, 0x9, 0x2}	Constellation_R12_LUT16_PACKED32[16/8]={C70BF438, 29E51AD6}
$\frac{3}{4}$	Constellation_R34_LUT64[64]={0x8, 0x7, 0xb, 0x4, 0xd, 0x2, 0xe, 0x1, 0xb, 0x4, 0xd, 0x2, 0xe, 0x1, 0x8, 0x7, 0xa, 0x5, 0x9, 0x6, 0xf, 0x0, 0xc, 0x3, 0x9, 0x6, 0xf, 0x0, 0xc, 0x3, 0xa, 0x5, 0xf, 0x0, 0xc, 0x3, 0xa, 0x5, 0x9, 0x6, 0xc, 0x3, 0xa, 0x5, 0x9, 0x6, 0xf, 0x0, 0xd, 0x2, 0xe, 0x1, 0x8, 0x7, 0xb, 0x4, 0xe, 0x1, 0x8, 0x7, 0xb, 0x4, 0xd, 0x2}	Constellation_R34_LUT64_PACKED32={1E2D4B78, 781E2D4B, 3C0F695A, 5A3C0F69, 695A3C0F, 0F695A3C, 4B781E2D, 2D4B781E}

Sequence Based PUs

A sequence is the execution of several programs back to back without the intervention of ARM.

Chase II Decoder

The Chase II decoder is a soft decision decoder that finds the maximum likelihood of a codeword from a reduced list of codewords. The reduced list of codewords is produced finding the N least reliable bits (based on the LLR soft decision input data) and generating 2^N test vectors with all possible combinations of those least reliable bits and finding the nearest valid codewords to the test vectors with a hard decision decoder. That new valid codeword list is sent to the ML decoder with the LLR values.

PROCESSOR SUBSYSTEM 2 (PS2)

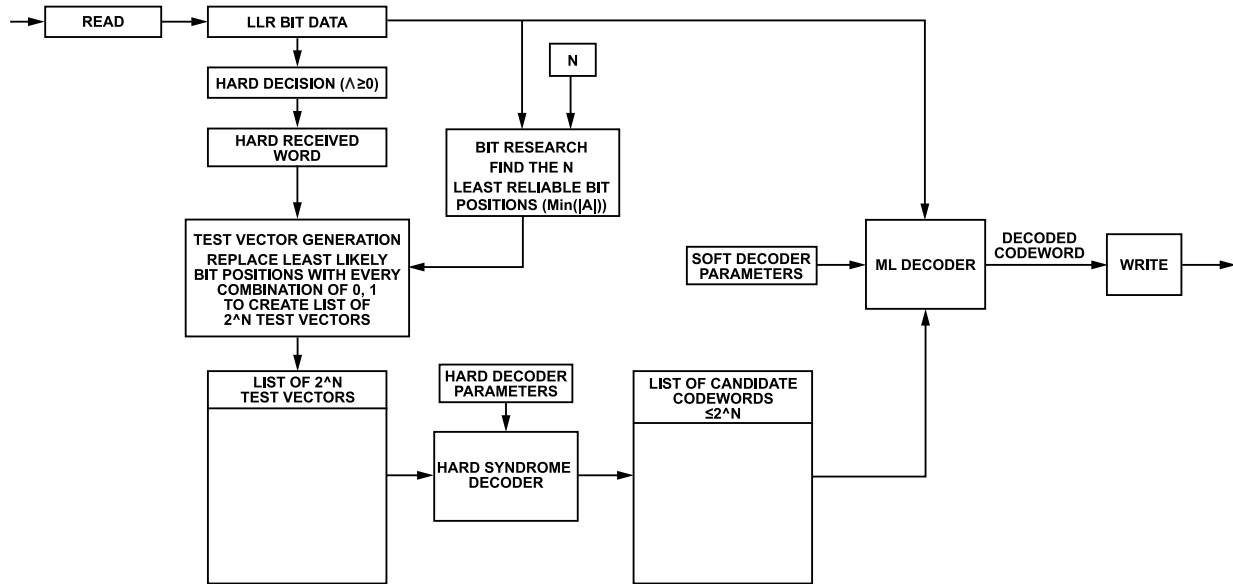


Figure 227. Chase II Decoder Block Diagram

Note:

- To use the Chase II Decoder, Sequence_Create driver need to be initialized and run. Do refer to the example code for more details on how to create it and use.
- The Sequence create driver creates the sequence of HWA processing units that are required for Chase II Decoder.

Chase 2 decoder parameters

Table 97. Chase II Decoder Parameters

Parameters	Description
dataBufIn	Input S3.5 LLR data buffer handle (Supported Format: PACKED 8-bit of length = softBitNum)
hsdMatrixBufIn	Input HS decoder matrix data buffer handle (Same as the Parity Check Matrix in HS Decoder PU)
hsdTableBufIn	Input HS decoder codeword table data buffer handle (Same as Syndrome table in the HS Decoder PU)
dataBufOut	Output data buffer handle: Outputs 1 32-bit word with softBitsNum LSBs set. If MSB of the output is 1, this means the input soft bits did not produce valid code word.
softBitNum	Number of Softbit input (up to 30)
reliableNum	Number of unreliable bits to search over (up to 4)
table_prog	Program table to add Chase II Decoder programs
table_prog_idx	Program table start index to populate Chase II Decoder programs

FFT/IFFT

This driver provides an implementation of the **Radix-2 Fast Fourier Transform (FFT)** algorithm. The FFT is an efficient way to compute the Discrete Fourier Transform (DFT) of a sequence, with a time complexity of $O(N \log N)$, where N is the size of the input sequence. The Radix-2 FFT works by recursively dividing the input sequence into smaller subsequences and applying the **butterfly operation** to combine results using **Twiddle factors**.

Twiddle Factor: $W_N^k = e^{-2\pi i \frac{k}{N}}$, used to combine the results of even and odd indexed parts of the sequence.

Butterfly Operation: Combines the FFT results of the even and odd indexed subsequences into the final FFT result.

For each recursion step, combine the results of the even and odd indexed subsequences using the butterfly operation:

$$X[k] = X_{\text{even}}[k] + W_N^k \cdot X_{\text{odd}}[k] \quad (43)$$

$$X[k + \frac{N}{2}] = X_{\text{even}}[k] - W_N^k \cdot X_{\text{odd}}[k] \quad (44)$$

PROCESSOR SUBSYSTEM 2 (PS2)

The basic FFT butterfly for an N point FFT/IFFT is given below:

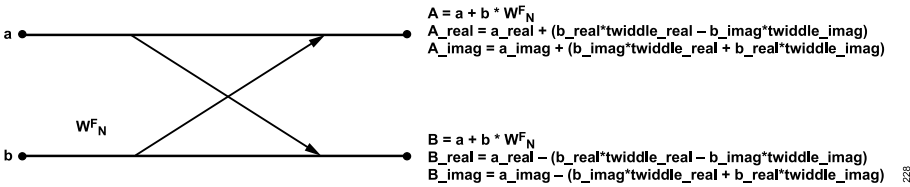


Figure 228. FFT butterfly for an N point FFT/IFFT

Limitation:

- The input sequence must have a length N that is a power of 2 (i.e., $N=2^m$).

Trellis Decoder (TCM Decoder)

This HWA implements the P25/DMR TCM Rate $\frac{1}{2}$ and $\frac{3}{4}$ decoders that input 98 LLRs (soft inputs) and generate 144 or 96 output bits.

Table 98. Trellis Decoder Input-Output combination for Rate $\frac{3}{4}$ and $\frac{1}{2}$

Rate	Input Block	Output block
3/4	98 S1.7 (int_8)	144 packed bits48 tribits 4.5 U32s
1/2		96 packed bits48 dibits 3 U32s

Note:

- To use the Trellis Decoder, Sequence_Create driver need to be initialized and run. Do refer to the example code for more details on how to create it and use.
- The Sequence create driver creates the sequence of HWA processing units that are required for Trellis Decoder.

Turbo Product Encoder (BPTC)

In the Turbo product encoder processing unit (Block Product Turbo Code (BPTC)), output code word is formed by making a rectangular array of information bits and then encoding the rows and columns of the rectangle.

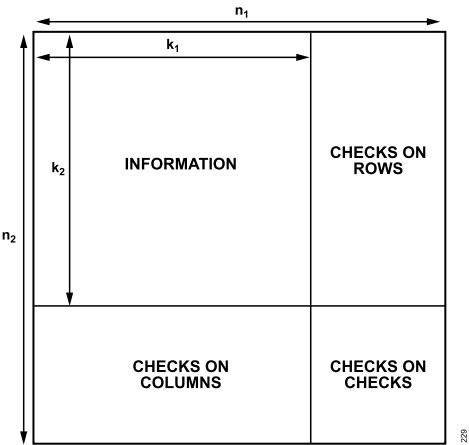


Figure 229. BPTC Format

The rows are encoded using the (n_1, k_1, δ_1) encoder, resulting in parity bits for each row and the columns are encoded using the (n_2, k_2, δ_2) encoder, resulting in parity bits for each column.

- n_1, n_2 : number of bits in the encoded row, number of bits in the encoded column respectively.
- k_1, k_2 : The number of information bits in the row and column respectively.
- δ_1, δ_2 : The minimum Hamming distance (minimum number of bit errors that can be corrected) for the row code and column code respectively.

PROCESSOR SUBSYSTEM 2 (PS2)

Turbo Product Encoder parameters:

Table 99. Turbo Product Encoder Parameters

Parameters	Description
dataBufIn	TURBO PRODUCT ENCODER input data buffer handle (Supported Data type: 32-bit Not PACKED) ► It has messageRowNum of LSBits set per word and has messageColNum words.
rowMatrixBufIn	Input row generation matrix data buffer handle (Supported Data type: 32-bit Not PACKED)
colMatrixBufIn	Input column generation matrix data buffer handle (Supported Data type: 32-bit Not PACKED)
dataBufOut	Output code word buffer handle (Supported Data type: 32-bit Not PACKED) ► It has codewordRowNum of LSBits set per word and has codewordColNum words.
messageRowNum	Number of rows in the input message data (Should be less than codewordRowNum and must not exceed 30)
messageColNum	Number of columns in the input message data (Should be less than codewordColNum and <30)
codewordRowNum	Number of rows in the output codeword generated
codewordColNum	Number of columns in the output code word generated
table_prog	Program table to add Turbo product encoder programs
table_prog_idx	Program table start index to populate Turbo product encoder programs

Turbo Product Decoder (Pyndiah)

The turbo product decoder uses an iterative method to decode Turbo Block Product Codes.

A block product code is formed by making a rectangular array of information bits and then encoding the rows and columns of the rectangle as per [Figure 229](#).

The input to the decoder is an array of log likelihood ratios (LLR), corresponding to each bit in the code. Where the LLR value is defined as:

$$\Lambda(r_i) = \log\left(\frac{p(c_i = 0 | r_i)}{p(c_i = 1 | r_i)}\right) \quad (45)$$

Viterbi Decoder

Viterbi decoding is an algorithm used to decode data that has been encoded with a convolutional code, which is commonly used in digital communication systems to correct errors introduced during transmission. The algorithm works by finding the most likely sequence of hidden states (the path) that results in the observed sequence of received symbols. This is achieved through a process of dynamic programming, which systematically examines all possible paths and selects the one with the highest probability.

Limitation:

- Available only in HWA2.

HWA Cycle Counter

One performance measurement of the processing units of the ADRV910x is the number of clock cycles it takes to complete a PUs operation. ADRV910x drivers provide a method of counting the clock cycles for a PU under a specified configuration. The cycle counter works for both program based HWA and sequence based HWA. The process for counting the number of cycles it takes for a PU to run is as follows:

1. Configure the PU to the desired configuration
2. Start the cycle counter using `adi_hwa1_CycleCounterEnable()`
3. Start the PU – the PU will run to completion as normal
4. In the callback function for the HWA, retrieve the cycle count using `adi_hwa1_CycleCounterGet()`
5. The cycle count can be cleared for the next measurement if needed using `adi_hwa1_CycleCounterClear()`

An example of the cycle counter is given as part of the `example_baremetal` project. A summary of some preliminary results is also given in [HWA Cycle Count Results](#).

HWA Examples and Resources

For detailed information on HWA and PU drivers, refer to the doxygen files as detailed in the [Resources and Examples](#) chapter.

PROCESSOR SUBSYSTEM 2 (PS2)

An example of each PU use is given in in the example code as detailed in the [Resources and Examples](#) chapter. Run this code example to see an example of how each of the PUs can be used and called including setting the parameters, adding to HWA table, starting and stopping the HWA.

Example: Signal Conditioning for an FSK Application

A demo of the HWA example use case with ADRV910x PS1 TX/RX data path is given in the Transceiver Evaluation Software tab. When 'PS2 in Data Path' and "HWA is Enabled" are selected (refer [Figure 246](#)), all signal data will run through the TX/RX Data Path, DMAs, HWAs, etc. Run the TES to see the HWAs example use cases working in a live demo.

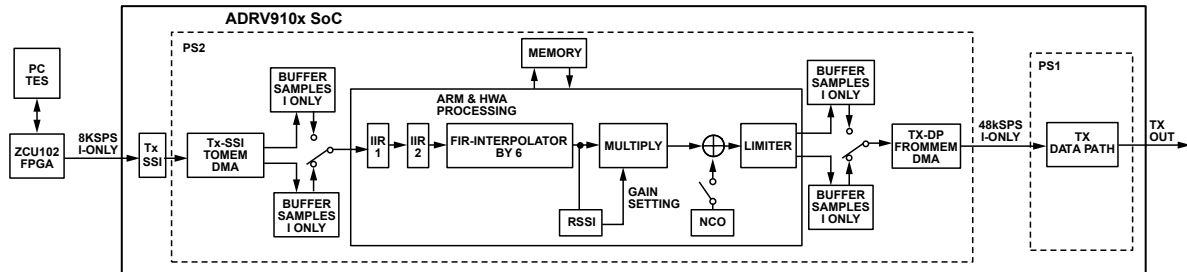


Figure 230. TX FSK Signal Processing Demo Block Diagrams with PS2 HWAs

The Tx FSK Signal Processing demo implements a simple signal conditioning for an FSK application. The incoming signal is filtered by the IIR and FIR PUs to remove unwanted higher frequencies. The filtered signal is then gain corrected. A simple gain correction block is implemented based on the RSSI value of the signal. If the signal RSSI is too high, the data is multiplied by a constant to decrease the amplitude. A limiter is also included to avoid saturation.

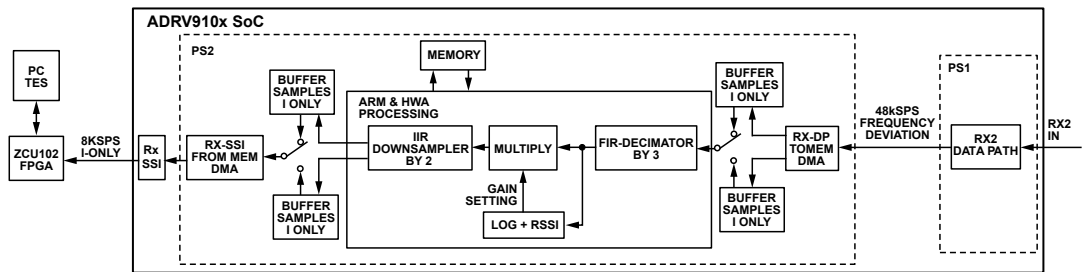


Figure 231. RX Frequency Deviation Signal Processing Demo Block Diagrams with PS2 HWAs

The Rx Frequency Deviation Signal Processing demo implements a simple signal conditioning for an FSK application. The incoming signal is filtered by the FIR PUs to remove unwanted higher frequencies. The filtered signal is then gain corrected. A simple gain correction block is implemented based on the RSSI value of the signal. If the signal RSSI is too high, the data is multiplied by a constant to decrease the amplitude.

The Tx and Rx signal processing demos shows how PUs can be chained and used in conjunction to perform signal processing.

Example: DMR Link Control Message

This example demonstrates digital transmit and receive operation modes utilizing HWA in PS2 to perform message encoding/decoding from data. This example relates to the Digital Mobile Radio (DMR) Air Interface (AI) protocol (ETSI TS 102 361-1). Reference will be made explicitly to this protocol; refer to the standard for further information.

This example encodes data into a package for transmission and decodes a received packet. The message structure created will be in the form of a 'general data burst' (ref 6.2). This burst includes data/info, slot type information and a 'SYNC' message.

The complete code example described is provided in the 'examples_baremetal' project within the ADRV910x SDK in the 'DMR Example' test.

Tx DMR Message

The Tx message takes 9 bytes (72 bits) of payload and generates a burst. The message created is an example of a 'full LC message' (ref 7.1). The message is created using a combination of PUs and software functions.

PROCESSOR SUBSYSTEM 2 (PS2)

Full Link Control (FULL LC) PDU Generation

The FULL LC PDU (ref 9.1.6) contains: header information, LC data and an LC CRC. In this example, the header information and data are stored and read from memory. The steps to create the LC PDU are as follows:

- ▶ The LC CRC is generated using the Reed-Solomon PU from the LC data. The RS parity bits are created in the form RS(12,9) (ref B.3.6). A mask is applied to the LC CRC (ref B.3.12).
- ▶ The full LC message is BPTC(196,96) encoded using the Turbo Product Encoder PU (ref B1.1). Prior to encoding, the LC message data is formatted correctly for use in the Turbo Product Encoder PU. The message is byte reversed (using byte reverse PU), data is reframed to add 3 zero bits at the beginning (using reframe PU), and repacked from packed 32-bit format to unpacked 11-bit format (using repack PU).
- ▶ The now encoded message is modulo interleaved (ref B.1.1) using the interleaver PU. Prior to interleaving, the encoded data is formatted correctly for use in the interleaver PU. The data is reframed to add 1 zero bits at the beginning (using reframe PU), and repacked from unpacked 15-bit format to packed 32-bit format (using repack PU).

Slot PDU Generation

The Slot PDU (ref 9.1.3) contains: color code, data type and a slot type parity. In this example, the color code, data type are stored and read from memory. The steps to create the slot PDU are as follows:

- ▶ The 'slot type parity' is encoded as Golay (20,8) as per B.3.1. The Linear Block Code Generator (LBCG) PU is used to generate the Golay encoder. The generation matrix used by the LBCG is defined in `lbcg_Gen_Matrix` as per B.3.1.
- ▶ Prior to the golay encoder, the data needs to be formatted for use by the LBCG PU. There is no PU in ADRV910x for this format conversion, so a software utility `dmr_lbcg_pack()` is used.

Burst Building and Symbols

The burst output is created by arranging the LC data, Slot PDU and Sync Data. The burst created in the form of 'full LC message' (ref 7.1). Several instances of the reframe PU are used to arrange the blocks of the burst.

The burst is lastly arranged into symbols for transmission. The modulation scheme used is a form of 4FSK (ref 10.2). There is a symbol mapping PU in ADRV910x for this symbol creation but is not available yet, so a software utility `dmr_symbols()` is used.

Rx DMR Message

For this use case, the objective is to process a received frame and decode corresponding transmitted PDUs. DMR burst processing depends on the type of received burst delineated by the sync pattern. For this use case example, the following assumptions are made:

- ▶ Entire burst including padding symbols have been received and buffered
- ▶ Samples are 16-bit data in packed format
- ▶ Start and end of frame is known
- ▶ Received data words are 16-bit frequency deviations words
- ▶ Burst includes a known sync pattern
- ▶ Optimum Sampling Epoch is known
- ▶ Receive burst is mobile sourced — Padding bits are all zero
- ▶ System Colour Code is known—As a result one of possible 16 slot types can be received

There are three main actions taken to decode the DMR burst as follows:

1. Parse the frame. As the frame is buffered and all the parameters of the burst is known, the constituent parts of the burst (i.e. SYNC code, FULL LC PDU, and slot PDU) can be parsed directly (ref 7.1).
2. For each part of the DMR burst, a soft decision decoding is applied using a log likelihood ratio (LLR).
3. For each parts of the burst, it is decoded and formatted to retrieve the encoded data.

The steps taken to decode the full DMR burst are described as follows.

SYNC Code

The SYNC code is retrieved and processed as follows:

- ▶ The frame is parsed to retrieve the SYNC code. The formatting of the burst is known so it can be parsed directly (ref 7.1).

PROCESSOR SUBSYSTEM 2 (PS2)

- ▶ As the expected SYNC code is known, the parsed code can be compared against it. This is used to generate an LLR factor. The LLR factor is used throughout the decoding to get a more accurate LLR. The LLR factor is rounded using a software function **gen_rnd_sat()**.

Slot PDU

The Slot PDU (ref 9.1.3) contains: color code, data type and a slot type parity. In this example, the color code, data type are decoded and written to memory. The slot PDU is retrieved as follows:

- ▶ The frame is parsed to retrieve the slot PDU. The formatting of the burst is known so it can be parsed directly (ref 7.1).
- ▶ The slot PDU is processed by applying the LLR, rounding, and formatting.
- ▶ The Maximum Likelihood (ML) decoder is used to compute the minimum mean square (MMS) decoding of the Slot PDU.
- ▶ The slot PDU is encoded as is encoded as Golay (20,8) as per B.3.1. A golay syndrome detector is implemented using the CRC PU. The MMS, found previously is used in the CRC. This syndrome detection is used to detect an errors in the burst.
- ▶ If there are no errors, retrieve the slot type using a software utility **getSlotType()**. It is expected that the slot type is '1' corresponding to a Full LC header.

FULL LC PDU

The LC PDU is retrieved as follows:

- ▶ It is expected that the frame type will have been identified as a full LC header. The frame is parsed to retrieve the LC PDU. The formatting of the burst is known so it can be parsed directly (ref 7.1).
- ▶ The LC PDU is processed by applying the LLR, rounding, and formatting.
- ▶ The LC PDU is modulo interleaved (ref B.1.1). The interleaver PU is used to de-interleave the LC PDU.
- ▶ The full LC message is BPTC(196,96) encoded using the Turbo Product Encoder PU (ref B1.1). The turbo product decoder (Pyndiah) is used to decode the LC message. Prior to the Pyndiah decoding, the LC message is reformatted using a software utility **preBPTCRepack()**. This repacks the data in the correct format for use by the Pyndiah decoder.
- ▶ After interleaving, the encoded data is reformatted to retrieve the LC message. The data is in (196,96) Block Product Turbo Code format (ref B1.1). The message is extracted by removing the parity bits (keep the first 9 rows and exclude the last four columns). The parity bits can also be extracted.
- ▶ The LC message is reformatted by repacking data into a packed 32bit format using the repack PU, the reframe PU is used to remove 3 zero bits at the start of the frame (ref B1.1), and the bits are reversed to convert to LSB format using the byte reverse PU.
- ▶ A parity check is done on the reformatted LC message using the Reed-Solomon PU. The Reed-Solomon PU decoder produces a parity bit. If the parity is zero, the decode was successful.
- ▶ The full LC message is now retrieved and error checked.

DMR Example Test and Debug

The complete code example described is provided in the 'examples_baremetal' project within the ADRV910x SDK in the 'DMR Example' test. This can be accessed in the way described in [Resources and Examples](#).

In addition to the DMR example burst creation and decoding, two addition features are given: timing and print data output.

The timer for DMR logs how long various parts of the DMR processing takes. This can be used to see the processing time of the DMR example. Set DMR_TSTAMP to 1 to log the timestamps.

The print data output will print various parts of the DMR message during the processing. This can be used to inspect the data formatting as it is being processed. Set DMR_DEBUG to 1 to print the data output.

PS1 CONTROL VIA PS2

As described in the [Software Architecture](#), control of the ADRV910x is hardware agnostic. Using a Hardware Abstraction Layer (HAL), any suitable host can be used to control the ADRV910x. In a similar way, the ADRV910x PS2 can directly control the ADRV910x PS1 on the same device.

ADRV910x APIs can be compiled and run directly on the PS2 ARM controller and interface to the PS1 using the supplied HAL. The HAL between PS2 and PS1 is similar to a HAL using an external host, however, no SPI bus is needed for the control since the commands and signals are sent internally only. The PS2 can directly control PS1 via API to make real time changes like gain control, carrier frequency, state changes etc.

PROCESSOR SUBSYSTEM 2 (PS2)

PS1 can support multiple clients in the same application so real time control of PS1 can be commanded by PS2 and an external host. It is expected that an external host is used to initialize PS1 at boot-up.

Supported APIs

ADRV910x supports a wide range of APIs for control of the transceiver. The APIs used for an external host are the same as used for PS2 control. The full set of APIs are provided as part of the ADRV910x SDK including a doxygen document to view all APIs and descriptions which is located at “\adrv910x-sdk\pkg\production\ADRV910X_API.chm”.

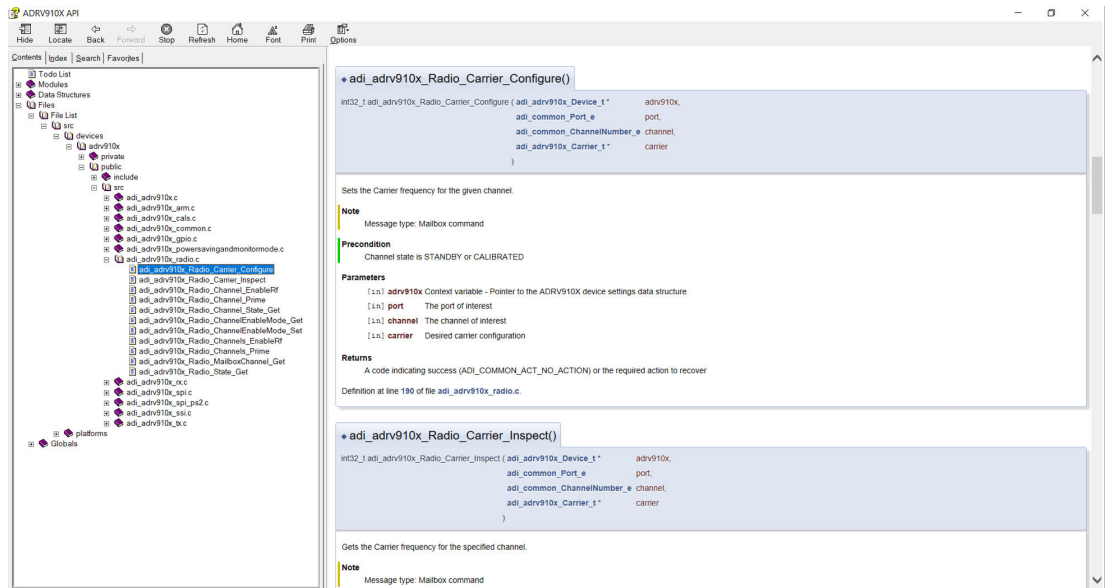


Figure 232. Screenshot of ADRV910x PS1 API Doxygen Document

Although all relevant PS1 APIs are included and can be used by PS2, there are a few exceptions listed in Table 100, PS1 is required to be booted and configured prior to PS1 start, hence some APIs are considered pre-requisites to PS2 start and thus are not supported.

Table 100. PS1 APIs Excluded from PS2 Control

API	Reason
adi_adrv910x_InitAnalog	Pre-requisite to PS2 start. adi_adrv910x_InitDevPS2 should be called initially before other PS1 calls are made by PS2.
adi_adrv910x_ssi_Rx_TestMode_Configure	SSI should be configured by BBIC.
adi_adrv910x_Ssi_Tx_TestMode_Configure	SSI should be configured by BBIC.
adi_adrv910x_Ssi_Tx_TestMode_Status_Inspect	SSI should be configured by BBIC.
adi_adrv910x_Ssi_Loopback_Set	SSI should be configured by BBIC.
adi_adrv910x_Ssi_Delay_Configure	SSI should be configured by BBIC.
adi_adrv910x_Ssi_Delay_Inspect	SSI should be configured by BBIC.
adi_adrv910x_Ssi_Tx_Disable_LSSI_Pads	SSI should be configured by BBIC.
adi_adrv910x_Ssi_Tx_Enable_LSSI_Pads	SSI should be configured by BBIC.
adi_adrv910x_arm_System_Program	Pre-requisite to PS2 start.
adi_adrv910x_arm_Image_Write	Pre-requisite to PS2 start.
adi_adrv910x_arm_Profile_Write	Pre-requisite to PS2 start.
ADI_ADRV910X_ARM_PFIProfiles_Write	Pre-requisite to PS2 start.
adi_adrv910x_arm_Start	Pre-requisite to PS2 start.
adi_adrv910x_arm_AhbSpiBridge_Enable	Pre-requisite to PS2 start.
adi_adrv910x_arm_AhbSpiBridge_Disable	Pre-requisite to PS2 start.
adi_adrv910x_gpio_ManualInput_Configure	Users should use the native PS2 GPIO driver/API

PROCESSOR SUBSYSTEM 2 (PS2)

PS2 SDK Variation

Initialization

A new PS2 specific function, `adi_adrv910x_InitDevPS2`, has been added to support initialization of the 'device' object/structure for PS2. This function also takes an optional parameter to set a timer function. The SDK API requires a timer with microsecond resolution to manage communication with PS1. By default, the SDK will consume hardware timer 3 in PS2 to perform the timing function. The default implementation operates in polled mode without interrupts.

API Hardware Semaphore/Mutex Implementation

Since multiple clients (i.e. PS2, BBIC) can control PS1 in the same application, it is necessary to provide protection against collisions in access of registers and PS1 mailbox. Hardware semaphore/mutex protection is included in PS1 APIs although it is recommended that user created applications avoid the possibility of these kind of collisions occurring.

All PS1 APIs (include APIs not supported by PS2) optionally include hardware semaphore/mutex protection. The implementation uses hardware semaphore seven (7) in the monitor core, leaving the first seven (0-6) semaphores for configurable use. The semaphore/mutex protection may be optionally enabled by defining the macro 'API_MUTEX' in the build environment. When 'API_MUTEX' is defined access to the registers and PS1 mailbox are serialized between PS2 and BBIC. The API semaphore protection is not meant to make the API re-entrant on a single system, but to ensure access from multiple clients (PS2, BBIC) do not interfere with each other.

PS1 Control Resources and Examples

For detailed information on PS1 APIs, refer to the doxygen files located at "`\\adrv910x-sdk\\pkg\\production\\ADRV910X_API.chm`". As described in [Initialization Example Code](#), autogenerated sample code can be retrieved from TES based on the desired configurations. This code can be used as a basis for a PS2 application, or otherwise.

An example of some PS1 APIs with PS2 control is given in the example code as detailed in the [Resources and Examples](#) chapter. Run this code example to see an example of how PS2 can be used to control PS1. The examples given do not contain an exhaustive list of supported APIs. The examples give a subset of commonly used APIs to show the process of including PS1 APIs in a PS2 application. [Figure 233](#) shows the PS1 control menu provided in the example files.

PROCESSOR SUBSYSTEM 2 (PS2)

```

Terminal I/O
Output: Log file: Off

Example tests started...

--- Main Menu ---
[aa] UART
[ab] Timer
[ac] Hardware Accelerators
[ad] PS1 Control
[**] Run All Tests
[!!] Exit Menu

Choice>>
ad

--- PS1 Control Menu ---
[aa] PS1 Test Info
[ab] PS1 Get Radio State
[ac] PS1 Carrier Configure
[ad] PS1 Radio Calibrate
[ae] PS1 Rx Tracking Cals Set
[af] PS1 Rx Gain Set
[ag] PS1 Rx Gain Control Configure
[**] Run All Tests
[!!] Exit Menu

Choice>>
aa

Before testing PS1 APIs, PS1 should be initialised
PS1 should be configured with the following settings:
- Device Clock: 38.4 MHz
- Rx2 (Narrowband) profile selected
- PS2 in Data Path enabled
- Carrier: 450MHz
- External LO , LO Divisor 2, 900.884MHz
- Device Calibrated

Choice>>

Input: Ctrl codes Options...
Buffer size: 0

```

Figure 233. PS1 Control Menu

PS1 Control Example: Sync Detect With Frequency and Time Offset Correction

A sync detect feature is provided in PS1 as described. Also provided is extending the sync detect functionality with PS2. This example may be used as is for sync detection or as an example of PS1 control and interaction by PS2.

Once the sync pattern is detected by PS2, a number of processes are taken by PS2:

- ▶ The sync detect information is provided to PS2 from PS1 via a mailbox command. The sync detect information includes the carrier frequency offset, sync sample offset, outputs of correlator and sync correlator number.
- ▶ The DMA is opened to receive the Rx data as in [Data Path Direct Memory Access \(DMA\)](#). The data from DMA is searched to verify the sync location. PS2 checks if the received sync data matches the expected sync pattern. It uses a correlator to determine the sync values and compares them against a predefined pattern.
- ▶ PS2 computes the frequency offset correction based on the current carrier frequency offset. It updates the NCO control word and sends it to the PS1 using the ADRV910x API.
- ▶ PS2 computes the timing offset correction based on the sync correlator values. It updates the resampler input and sends it to the PS1 using the ADRV910x API.

This example is provided in the TES GUI in the sync detect section. The source code for the PS2 example is provided in the usecase project as described in [Resources and Examples](#).

ANNEX

ADRV910x SoC supports multiple boot modes. The default normal boot mode in ADRV910x SoC is by SPI Peripheral. In this mode, the host processor configures the ADRV910x SoC using SPI-S interface to download the ARM image to PS1 and PS2 memories, refer [Figure 159](#).

PROCESSOR SUBSYSTEM 2 (PS2)

To shorten the ADRV910x SoC boot up time, an optional boot mode in ADRV910x SoC could be through CMOS or LVDS TX SSI interface, the ARM image for PS1 and PS2 could be downloaded by TX SSI interface with much faster speed than SPI-S mode. The host processor needs to frame the data in a START/ADDRESS/DATA/STOP packet format before sending it through TX SSI interface. And the TX SSI interface should be properly configured including the clock selection using device clock or PLL clock.

HWA Cycle Count Results

Below are some example results of the HWA clock cycle counter. The specified configuration and cycle count is given. To see how this measurement was performed, refer to [HWA Cycle Counter](#).

Table 101. Preliminary Example Results of HWA Clock Cycle Count

Number	HWA	Available on	Data Format	Key Parameters	Cycle Count
1	UpSampler	HWA1/HWA2		Input Size, Up Ratio	
			Packed Real 16-Bit	256, 3	10336
			Packed Real 16-Bit	256, 5	15456
			Packed Real 16-Bit	512, 3	20576
			Packed Real 16-Bit	512, 5	30816
			MSB Real 24-Bit	256, 3	10336
			MSB Complex 24-Bit	256, 3	11129
2	DownSampler	HWA1/HWA2		Output Size, Down Ratio	
			Packed Real 16-Bit	171, 3	1964
			Packed Real 16-Bit	103, 5	1216
			Packed Real 16-Bit	342, 3	3845
			Packed Real 16-Bit	205, 5	1964
			MSB Real 24-Bit	171, 3	1216
			MSB Complex 24-Bit	171, 3	2160
3	Limiter	HWA1/HWA2		Input Size	
			Packed Real 16-Bit	200	2138
			Packed Real 16-Bit	16384	163978
			MSB Real 24-Bit	200	2138
			MSB Complex 24-Bit	200	2366
4	NCO	HWA1/HWA2		Output Size	
			Complex 16-Bit Interleaved	1024	16521
			Complex 16-Bit Interleaved	2048	32905
			Complex 16-Bit Non-Interleaved	1024	17545
5	LBCG	HWA1/HWA2	Data 32-Bit	Input Rows x Columns	
				4 x 32	487
				4 x 64	922
				32 x 4	487
				64 x 4	922
6	Bit Transpose	HWA1/HWA2	Data 32-Bit	Inputs Rows x Columns	
				4 x 16	388
				4 x 32	499
				16 x 4	456
				32 x 4	616
7	Byte Transpose	HWA1/HWA2	Data 32-Bit	Input Rows x Columns	
				4 x 16	1091
				4 x 32	2099
				16 x 4	863

PROCESSOR SUBSYSTEM 2 (PS2)**Table 101. Preliminary Example Results of HWA Clock Cycle Count (Continued)**

Number	HWA	Available on	Data Format	Key Parameters	Cycle Count
8	CRC Encoder/CRC Decoder	HWA1/HWA2	Data 32-Bit	32 x 4	1567
				PolynomialDegree, NumBits	
				32, 80	413
				32, 1056	1719
9	Bit Scrambler/Descrambler	HWA1/HWA2	Data 32-Bit	63, 1056	1719
				PolynomialDegree, NumBitBytes	
				16, 128	455
				16, 64	367
10	Byte Scrambler/Descrambler	HWA1/HWA2	Data 32-Bit	7, 128	455
				7, 64	367
				PolynomialDegree, NumBitBytes	
				44, 1024	4120
11	PRBS Generator	HWA1/HWA2	Data 32-Bit	44, 256	1687
				32, 1024	4120
				32, 256	1687
				PolynomialDegree, NumBits	
12	Congruent PN Generator	HWA1/HWA2	Data 32-Bit	63, 1024	1687
				31, 1024	1687
				7, 96	411
				7, 31	309
13	Log2/Log10	Functional Block			
14	ATAN	HWA1	Input: MSB 24-Bit Complex	NumSamples, Iterations	
			Output: MSB 24-Bit Real	400, 16	12119
				200, 16	6119
15	MULT	HWA1/HWA2		Input NumSamples	
			Packed Real 16-Bit	256	3562
			Packed Real 16-Bit	512	6890
			MSB Real 24-Bit	256	3562
			MSB Complex NI 24-Bit	256	5480
16	SUM	HWA1/HWA2		Input NumSamples	
			Packed Real 16-Bit	200	2642
			Packed Real 16-Bit	400	5042
			MSB Real 24-Bit	200	2642
			MSB Complex NI 24-Bit	200	2860
17	SUB	HWA1/HWA2		Input NumSamples	
			Packed Real 16-Bit	264	3681
			Packed Real 16-Bit	528	7113
			MSB Real 24-Bit	264	3683
			MSB Complex NI 24-Bit	264	5789
18	ADD	HWA1/HWA2		Input NumSamples	
			Packed Real 16-Bit	264	3683

PROCESSOR SUBSYSTEM 2 (PS2)

Table 101. Preliminary Example Results of HWA Clock Cycle Count (Continued)

Number	HWA	Available on	Data Format	Key Parameters	Cycle Count
19	RSSI/Power	HWA1/HWA2	Packed Real 16-Bit	528	7115
			MSB Real 24-Bit	264	3683
			MSB Complex NI 24-Bit	264	5789
				Input NumSamples	
			Packed Real 16-Bit	200	2743
20	Integrator	HWA1/HWA2	Packed Real 16-Bit	400	5343
			MSB Real 24-Bit	200	2743
			MSB Complex NI 24-Bit	200	2743
				Input NumSamples	
			Packed Real 16-Bit	200	2642
21	Correlator	HWA1/HWA2	Packed Real 16-Bit	400	5042
			MSB Real 24-Bit	200	2642
			MSB Complex NI 24-Bit	200	2860
				Input NumSamples	
			Packed Real 16-Bit	256	3502
22	FIR	HWA1/HWA2	Packed Real 16-Bit	512	6830
			MSB Real 24-Bit	256	3507
			MSB Complex NI 24-Bit	256	3546
			DataFormat, CoefFormat	NumSamples, NumCoef	
			Packed Real 16-Bit, Packed Real 16-Bit	256, 32	116739
			Packed Real 16-Bit, Packed Real 16-Bit	256, 16	63491
			Packed Real 16-Bit, Packed Real 16-Bit	128, 32	58499
			Packed Real 16-Bit, Packed Real 16-Bit	128, 16	31872
23	FIR Decimate	HWA1/HWA2	NI Complex 16-Bit, Packed Real 16-Bit	256, 31	123142
			NI Complex 16-Bit, NI Complex 16-Bit	256, 31	131079
			DataFormat, CoefFormat	NumSamples, NumCoef, DecRatio	
			Packed Real 16-Bit, Packed Real 16-Bit	256, 32, 2	120201
			Packed Real 16-Bit, Packed Real 16-Bit	256, 16, 2	66953
			Packed Real 16-Bit, Packed Real 16-Bit	128, 32, 2	60233
			Packed Real 16-Bit, Packed Real 16-Bit	128, 16, 2	33609
			Packed Real 16-Bit, Packed Real 16-Bit	256, 32, 4	118729
			Packed Real 16-Bit, Packed Real 16-Bit	256, 16, 4	65481
			Packed Real 16-Bit, Packed Real 16-Bit		

PROCESSOR SUBSYSTEM 2 (PS2)

Table 101. Preliminary Example Results of HWA Clock Cycle Count (Continued)

Number	HWA	Available on	Data Format	Key Parameters	Cycle Count
			Packed Real 16-Bit, Packed Real 16-Bit	128, 32, 4	59497
			Packed Real 16-Bit, Packed Real 16-Bit	128, 16, 4	32873
			NI Complex 16-Bit, Packed Real 16-Bit	256, 31, 4	124937
			NI Complex 16-Bit, NI Complex 16-Bit	510, 31, 255	261433
24	FIR Interpolate	HWA1/HWA2	DataFormat, CoefFormat	NumSamples, NumCoef, InterRatio	
			Packed Real 16-Bit, Packed Real 16-Bit	256, 32, 2	233984
			Packed Real 16-Bit, Packed Real 16-Bit	256, 16, 2	127488
			Packed Real 16-Bit, Packed Real 16-Bit	128, 32, 2	117120
			Packed Real 16-Bit, Packed Real 16-Bit	128, 16, 2	63872
			Packed Real 16-Bit, Packed Real 16-Bit	256, 32, 4	61824
			Packed Real 16-Bit, Packed Real 16-Bit	256, 16, 4	248832
			Packed Real 16-Bit, Packed Real 16-Bit	128, 32, 4	231040
			Packed Real 16-Bit, Packed Real 16-Bit	128, 16, 4	124544
			NI Complex 16-Bit, Packed Real 16-Bit	256, 31, 4	483584
			NI Complex 16-Bit, NI Complex 16-Bit	2, 31, 255	253272
25	IIR	HWA1/HWA2	InputFormat, OutputFormat	NumSamples	
			Packed Real 16-Bit, MSB Real 24-Bit	256	5141
			MSB Real 24-Bit, MSB Real 24-Bit	256	5396
			MSB Real 24-Bit, MSB Real 24-Bit	256	5396
			MSB Real 24-Bit, Packed Real 16-Bit	256	5141
					Total for 8th Order Result: 21074
			Packed Real 16-Bit, MSB Real 24-Bit	128	2706
			MSB Real 24-Bit, MSB Real 24-Bit	128	2836
			MSB Real 24-Bit, MSB Real 24-Bit	128	2836
			MSB Real 24-Bit, Packed Real 16-Bit	128	2706
					Total for 8th Order Result: 11084

PROCESSOR SUBSYSTEM 2 (PS2)

Table 101. Preliminary Example Results of HWA Clock Cycle Count (Continued)

Number	HWA	Available on	Data Format	Key Parameters	Cycle Count
26/27	Interleaver/De-Interleaver	HWA1/HWA2	Data 32-Bit	NumSamples	
				196	2488
				392	4390
				196	2414
				392	4570
				144	2174
				288	3698
				144	1986
				288	3715
28	Re-Framer	HWA1/HWA2	Data 32-Bit	NumBits	
				65535	67856
				64	314
29	Maximum Likelihood Decoder	HWA2	Data 32-Bit	NumCodewords, NumSoftBits	
				512, 16	6408
				511, 13	6396
				512, 40	41189
30	Hard Syndrome Decoder	HWA2	Data 32-Bit	66, 64	5509
				NumSamples, ParityMatrixSize	
				25, 5	2472
				25, 3	1922
31	Turbo Product Encoder	HWA1/HWA2	Data 32-Bit	10, 5	1092
				10, 3	872
				NumRowsIn, NumColsIn, NumRowsOut, NumColsOut	
				9, 11, 13, 15	10194
32	Trellis Decoder	HWA1	Packed Real 16-Bit	25, 25, 30, 30	22887
				14, 11, 19, 15	12309
				Rate = 3/4	145527
33	Chase II Decoder	HWA2	Data 8-Bit	Rate = 1/2	85344
				ParityMatrixSize, SoftBitNum, ReliableNum	
				8, 30, 2	1858
				4, 8, 2	1586
34	Convolution Encoder	HWA2	Data 32-Bit	4, 8, 4	3710
				NumBits	
				512	1913
35	Trellis Encoder	HWA2	Data 32-Bit	66	541
				Rate = 3/4	1515
				Rate = 1/2	1493
36	Pyndiah-Chase II Decoder	HWA2	Data 32-Bit	RowMatrixSize, ColMatrixSize, RowSyndromeTabSize, ColSyndromeTabSize, ReliableNum	
				4, 4, 16, 16, 2	1774453
				5, 1, 32, 2, 4	2157886
				5, 1, 32, 2, 3	2394207
37	Reed-Solomon Encoder/Detect	HWA1/HWA2	Data 8-Bit	NumIn, NumOut/Parity	
				239, 16	93920
				20, 16	8072
				9, 3	1068

PROCESSOR SUBSYSTEM 2 (PS2)**Table 101. Preliminary Example Results of HWA Clock Cycle Count (Continued)**

Number	HWA	Available on	Data Format	Key Parameters	Cycle Count
38	Symbol Mapping	HWA1		NumSamples, BitSize	
			Packed Real 16-Bit	256, 2	8568
			NI Complex 16-Bit	256, 4	9080
			Interleaved Complex 16-Bit	1024, 8	35960
39	Repack	HWA1/HWA2	Data 32-Bit	Input NumSamples, Output NumSamples	
				6, 8	449
				10, 14	663
40	Memory Copy	HWA1/HWA2	Data 32-Bit	NumSamples	
				60	744
				120	1404
41	Format Conversion	HWA1/HWA2	Input: Packed Real 16-Bit, Output: LSB Data 16-Bit	NumSamples	
				100	1733
				200	3333
42	Byte Reverse	HWA1/HWA2	Data 32-Bit	NumSamples	
				6	173
				24	497
43	Fast FIR-RealCoef-RealData	HWA1/HWA2	Data & Coef = Packed Real 16-Bit	NumSamples, NumCoef	
				256, 32	81990
				256, 16	51334
				128, 32	41126
44	Fast Real Decimation	HWA1/HWA2	Data & Coef = Packed Real 16-Bit	128, 16	25798
				Input NumSamples, NumCoef, DecRatio, Output NumSamples	
				256, 32, 2, 128	85449
				256, 16, 2, 128	54793
				128, 32, 2, 64	42857
				128, 16, 2, 64	27529
				256, 32, 4, 64	83977
				256, 16, 4, 64	53321
45	Fast Real Interpolation	HWA1/HWA2	Data & Coef = Packed Real 16-Bit	128, 32, 4, 32	42121
				128, 16, 4, 32	26793
				Input NumSamples, NumCoef, InterRatio, Output NumSamples	
				256, 32, 2, 512	166528
				256, 16, 2, 512	105216
				128, 32, 2, 256	83392
				128, 16, 2, 256	52736
				256, 32, 4, 1024	326912
46	Viterbi	HWA2	Input: Data 8-Bit Output: Data 32-Bit	256, 16, 4, 1024	204288
				128, 32, 4, 512	163584
				128, 16, 4, 512	102272
				R, K, NumSoftInputs, tbMode	PM + TB
				2, 5, 23, INIT_STATE	92251 + 9116
				2, 5, 101, MAX_PM	405621 + 41924
46	Viterbi	HWA2	Input: Data 8-Bit Output: Data 32-Bit	2, 9, 27, INIT_STATE	1138598 + 16560
				2, 9, 102, MAX_PM	4309338 + 73177
				4, 9, 104, MAX_PM	4767826 + 73071

PROCESSOR SUBSYSTEM 2 (PS2)

Table 101. Preliminary Example Results of HWA Clock Cycle Count (Continued)

Number	HWA	Available on	Data Format	Key Parameters	Cycle Count
47	FFT/IFFT	HWA1/HWA2		FFT Type, NumPoints	
			Interleaved Complex 16-Bit	Radix2Div2, 8	3801
			Interleaved Complex 16-Bit	Radix2Div2, 64	38412
			NI Complex 16-Bit	Radix2DivN, 8	3906
			NI Complex 16-Bit	Radix2DivN, 64	41916
			NI Complex 16-Bit	Radix3Div2, 48	37908
			NI Complex 16-Bit	Radix3Div2, 768	815326
			NI Complex 16-Bit	Radix3DivN, 48	37758
			NI Complex 16-Bit	Radix3DivN, 768	805915
48	Min	HWA1/HWA2		NumSamples	
			Packed Real 16-Bit	256	3793
			Packed Real 16-Bit	512	7369
			MSB Real 24-Bit	256	3777
49	Max	HWA1/HWA2		NumSamples	
			Packed Real 16-Bit	256	3785
			Packed Real 16-Bit	512	7377
			MSB Real 24-Bit	256	3761

Q Formatting Standard Description

This section describes the two types of Q Formatting that is mentioned in this document.

1. Ua.b Format(Unsigned Fixed-Point Format)

- ▶ **Description:** In the Ua.b format, "U" stands for unsigned, meaning all numbers are non-negative. The "a" represents the number of integer bits, and "b" represents the number of fractional bits.
- ▶ **Range:** The range of values that can be represented in Ua.b format is from 0 to $2^a - 2^{-b}$.
- ▶ **Example:** For U2.3 format:
 - ▶ Binary Representation: **10.011**
 - ▶ Binary to Decimal Conversion:
 - ▶ Integer Part: $1 \times 2^1 + 0 \times 2^0 = 2$
 - ▶ Decimal Part: $0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = 0.375$
 - ▶ Total: $2 + 0.375 = 2.375$
 - ▶ Decimal to Binary Conversion:
 - ▶ Multiply the decimal number by 2^b (2.375×2^3)
 - ▶ That equals to $2.375 \times 2^3 = 19$.
 - ▶ 19 in Binary format is **10011**.

2. Sa.b Format(Signed Fixed-Point Format)

- ▶ **Description:** In Sa.b format, "S" indicates the format is signed, allowing for both positive and negative numbers. The "a" includes the sign bit along with the integer bits, and "b" is the number of fractional bits.
- ▶ **Range:** The range of values in Sa.b format is from -2^{a-1} to $2^{a-1} - 2^{-b}$.
- ▶ **Example:** For S2.2 format:
 - ▶ Binary Representation: **10.11** (Most Significant Bit is a signed bit)
 - ▶ Binary to Decimal Conversion:
 - ▶ Integer Part: $-1 \times 2^1 + 0 \times 2^0 = -2$

PROCESSOR SUBSYSTEM 2 (PS2)

- ▶ Decimal Part: $1 \times 2^{-1} + 1 \times 2^{-2} = 0.75$
- ▶ Total: $-2 + 0.75 = -1.25$
- ▶ Decimal to binary Conversion:
 - ▶ Multiply the positive decimal number by 2^b ($1.25 \times 2^2 = 5$)
 - ▶ 5 in Binary format is 0101.
 - ▶ 2's Complement of 0101 is **1011**.

TX-SSI Data Format for Code Download

Multiple blocks of 32-bit data words can be downloaded through the Tx-SSI interface to PS1 or PS2 Memory. The Tx-SSI interface uses the I and Q buses to send code over the lanes.

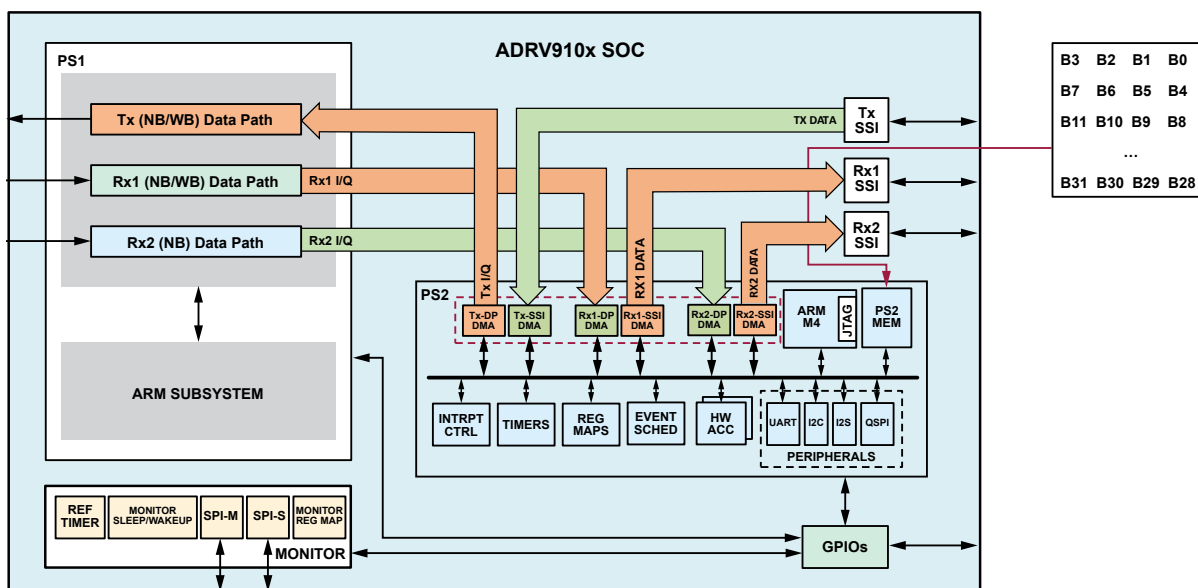


Figure 234. High Level Block Diagram of Code Download Using Tx-SSI Interface

Both CMOS Tx-SSI and LVDS Tx-SSI interfaces support the code download.

Mapping WORD32 = (Byte3, Byte2, Byte1, Byte0) to Tx-SSI:

Tx-CSSI:

2 lanes with I of 16 bits and Q of 16 bits.

▶ Lane 0 (I16): Byte3, Byte2

▶ Lane 1 (Q16): Byte1, Byte0

4 lanes with I of 16 bits and Q of 16 bits.

With Word32= Byte3, Byte2, Byte1, Byte0

Lane 0 (I8-lsb): Byte2

Lane 1 (I8-msb): Byte3

Lane 2 (Q8-lsb): Byte0

Lane 3 (Q8-msb): Byte1

Tx-LSSI:

2 lanes with I of 16 bits and Q of 16bits.

LVDS Lane 0 (I16): Byte3, Byte2

LVDS Lane 1 (Q16): Byte1, Byte0

A block of data starts with a sync pattern: "SYNC_START" which consists of 32-bit unique pattern for I and Q.

PROCESSOR SUBSYSTEM 2 (PS2)

A block of data continues with the destination start address: "START_ADDR" which consist of 32-bit address to ADRV910x.

A block of data ends with a sync pattern: "SYNC_END" which consists of 32-bit unique pattern for I and Q.

Both SYNC patterns are 2 unique word values of 32-bits (I and Q) that are user programmable and don't exist in the code to download.

The following is an example of downloading a block of 32-bit words:

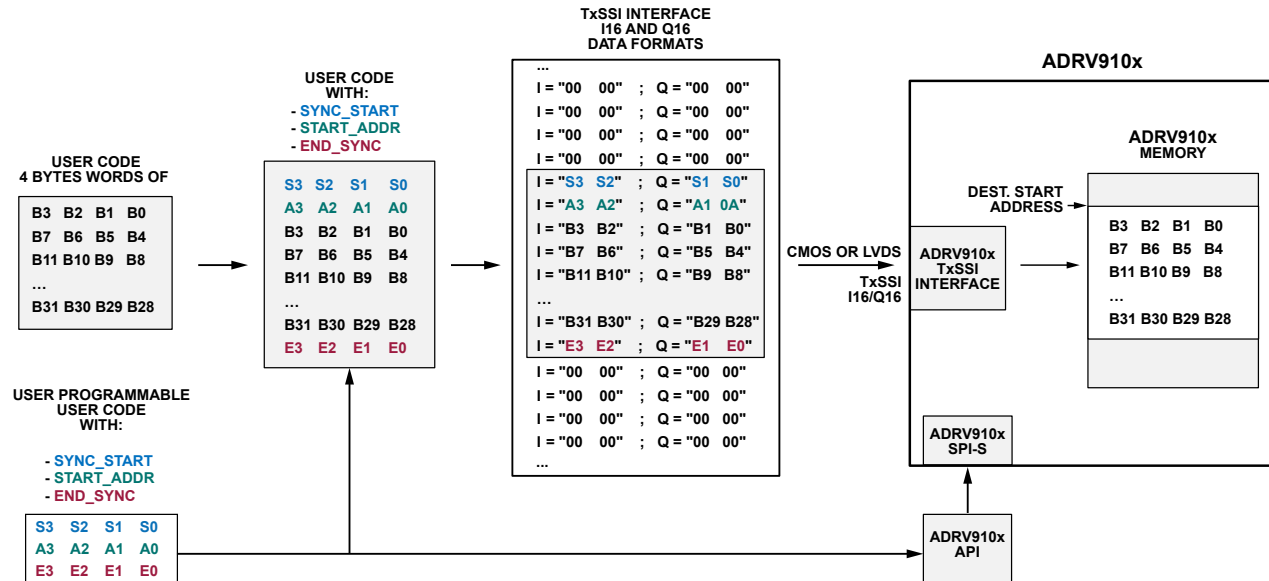


Figure 235. An Example of Downloading a Block of 32-bit Words

The following shows the Tx-SSI Lane mapping with different formats:



Figure 236. Tx-SSI Lane Mapping with Different Formats

An API is used to configure ADRV910x and to pre-program the expected SYNC patterns in ADRV910x before the start of code download.

Note: Code Download via SSI is not yet supported.

ADRV910X EVALUATION SYSTEM START GUIDE

The ADRV910X family demonstration system enables customers to evaluate the device without having to develop custom software or hardware. The system is comprised of a radio daughtercard, a ZCU102 FPGA motherboard, an SD card with operating system, a 12 V power supply for the ZCU102 that connects to a wall outlet, and a C#-based evaluation software application. The evaluation system uses an Ethernet interface to communicate with the PC.

INITIAL SETUP

The ADRV910X Transceiver **E**valuation **S**oftware (**TES**) is the GUI to communicate with the evaluation platform. It can run with or without evaluation hardware connected. When TES runs without the hardware connected, it can be fully configured for a demo mode. If the evaluation hardware is connected, set up the desired operating parameters with TES and then the software can program the evaluation hardware. After the device is configured, the evaluation software can be used to transmit waveforms using custom waveform files as well as observe signals received on one of the receiver input ports. The TES can be used to generate an initialization sequence in the form of C99/C#/Python/Matlab.

HARDWARE KIT

The ADRV910X demonstration system kit contains the following:

- ▶ The Customer Evaluation (CE) board in form of a daughter card with FMC connector
- ▶ One (1) SD card containing image of Linux operating system with the required evaluation software
 - ▶ SD card type is 16 GB size, type 10

Requirements

The hardware and software require the following:

- ▶ The ADRV910X demonstration system kit
- ▶ A ZCU102 evaluation platform (FPGA platforms not included in the ADRV910X demonstration kit) *
- ▶ One (1) 12 V power supply for powering the ZCU102 FPGA Platform.
- ▶ The operating system on the controlling PC must be Windows® 10 (x86 and x64), or Windows 11
- ▶ The PC must have a free Ethernet port with the following constraints:
 - ▶ If the Ethernet port is occupied by another LAN connection, use a USB-to-Ethernet adapter
 - ▶ The PC should be able to access over this dedicated Ethernet connection the following ports:
 - ▶ 22—SSH protocol
 - ▶ 8080—access to the evaluation software on the FPGA Platform
 - ▶ TES—contact your ADI representative to obtain access to this software
- ▶ For PS2 development (not included in the ADRV910X demonstration kit) *:
 - ▶ Codefusion Studio or IAR v9.32.1
 - ▶ JTAG Debugger
 - ▶ **Optional:** JTAG Debugger adapter (Note: ADRV910x Evaluation Board uses a 19-pin FTSH connector for JTAG connection. An adapter may be needed, dependent on the debugger used.)

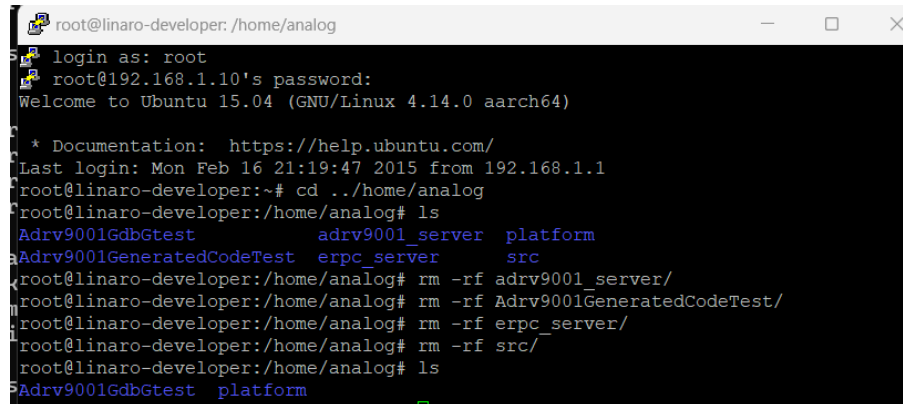
SD Card Imaging

The ADRV910x evaluation system is shipped with an SD card preloaded with the required boot image. If the SD card image becomes corrupted or unusable, follow the instructions on [Engineer Zone support forum for the TES GUI & Software Support](#) and use the [ADRV9001 SD Card image for ZCU102 FPGA](#) to re-write a new SD card image file for ZCU102 FPGA. After the image is restored, follow the below steps to make the SD card compatible with the ADRV910x evaluation system.

1. Install the new SD in J100 slot of the ZCU102 FPGA, and power up the FPGA board
2. Using "PuTTY" start a new SSH session with the following settings
 - ▶ IP address: 192.168.1.10
 - ▶ Port: 22
 - ▶ Username: root
 - ▶ Password: analog

ADRV910X EVALUATION SYSTEM START GUIDE

3. Log into the FPGA and manually delete all other folders except "Adrv9001GdbGtest" and "platform" at directory "/home/analog/", refer [Figure 237](#)
4. Power cycle the FPGA
5. Open the ADRV910x evaluation software, start the "connection"



```

root@linaro-developer: /home/analog
login as: root
root@192.168.1.10's password:
Welcome to Ubuntu 15.04 (GNU/Linux 4.14.0 aarch64)

* Documentation:  https://help.ubuntu.com/
Last login: Mon Feb 16 21:19:47 2015 from 192.168.1.1
root@linaro-developer:~# cd ../home/analog
root@linaro-developer:/home/analog# ls
Adrv9001GdbGtest  adrv9001_server  platform
Adrv9001GeneratedCodeTest  erpc_server      src
root@linaro-developer:/home/analog# rm -rf adrv9001_server/
root@linaro-developer:/home/analog# rm -rf Adrv9001GeneratedCodeTest/
root@linaro-developer:/home/analog# rm -rf erpc_server/
root@linaro-developer:/home/analog# rm -rf src/
root@linaro-developer:/home/analog# ls
Adrv9001GdbGtest  platform
  
```

Figure 237. Manually Remove the Unrelated Folders

Note: Make sure to check if there is any encryption when writing to an SD card that will prevent the FPGA from reading the card – this may be an issue if BitLocker is enabled on the PC used to write to the SD card.

ZCU102 Hardware Setup

ZCU102 FPGA platform setup requires the following steps:

1. All jumpers are in the positions shown in [Figure 238](#).
2. SW6 is in position as shown in [Figure 238](#) (2, 3, 4 = A position)
3. The SD card included with the evaluation kit is placed in the J100 slot of the ZYNQ platform

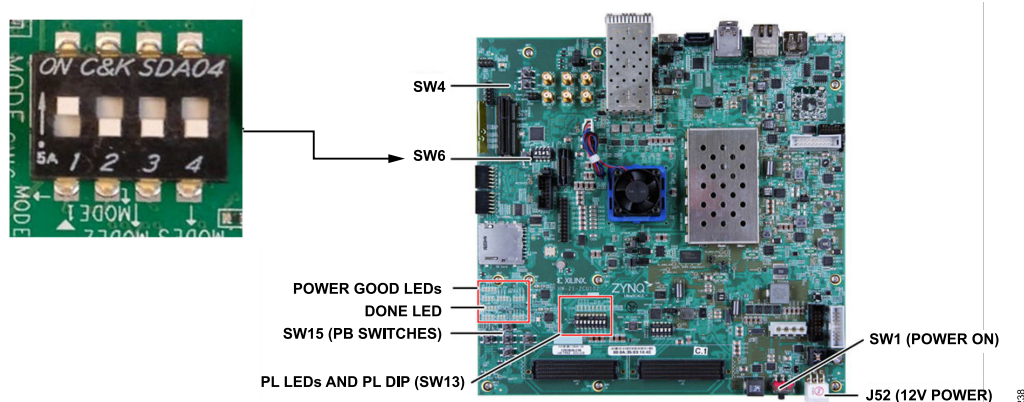


Figure 238. ZCU102 FPGA Evaluation Board with Jumper Settings and Switch Position Configuration

The evaluation hardware setup is shown in [Figure 239](#).

ADRV910X EVALUATION SYSTEM START GUIDE

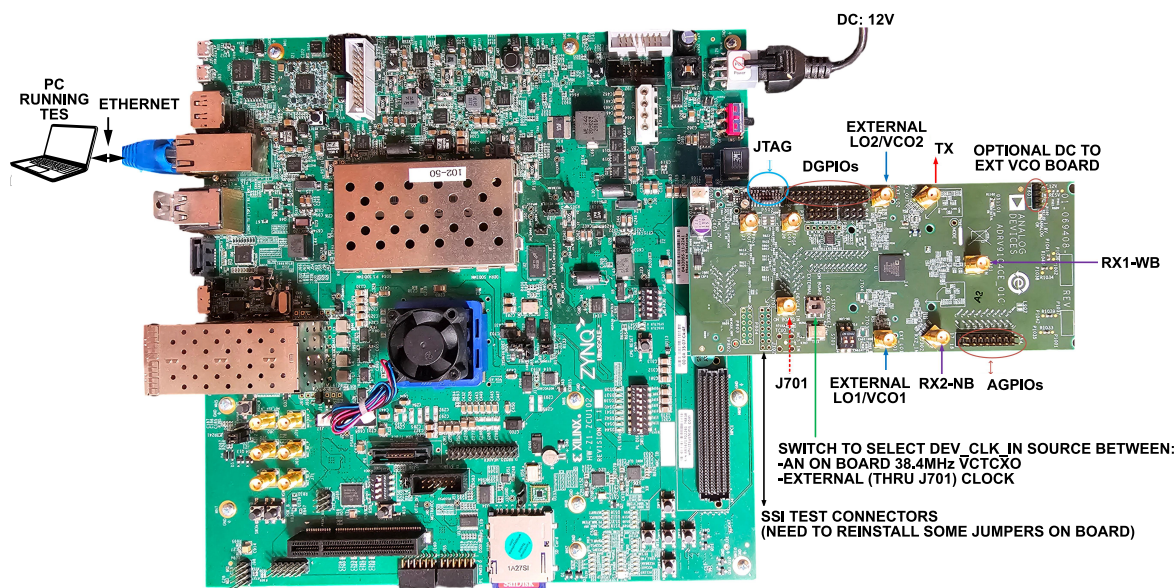


Figure 239. ADRV910x Evaluation Card and ZCU102 Evaluation Platform with Connections Required for Testing

239

To set up the evaluation board for testing, follow steps listed below:

1. Connect the ADRV910x evaluation card and the ZCU102 evaluation platform together as shown in Figure 239. Use the FMC connector (J5). Take care to be sure the connectors are properly aligned.
2. Make sure that all jumpers on the ZCU102 evaluation platform as well as the SW6 position (2, 3, 4 = "A" position) match settings shown in Figure 238.
3. Insert the SD card that came with the ADRV910x evaluation kit into ZCU102 evaluation platform SD card slot (J100).
4. On the ADRV910x evaluation card:
 - a. Provide a device clock (frequency must match the setting selected in the TES), at a +6dBm power level to J701 connector. (This signal drives the reference clock into the LTC6957 clock distribution chip on the board – the LTC6957 fans out the DEV_CLK for the ADRV910x and REF_CLK for the FPGA). There is also an option of using the on board 38.4MHz VCTCXO by switching on S702.
 - b. Provide an EXT LO/VCO clock at a +6dBm power level to either J401 or J402 connector, depends on the TES profile settings.
 - c. It should be noted that quality of clock source used to generate DEV_CLK / EXT LO will directly impact overall system performance. User must ensure that high quality, stable and low phase noise clock source is used here.
 - d. The DEV_CLK and EXT LO sources should be synchronized in case Ext LO is in use.
5. Connect a 12V, 5A power supply to the FPGA Platform at the J52 header.
6. Connect the FPGA Platform to the PC with an Ethernet cable (connect to P12). There is no driver installation required.
 - a. In the case when the Ethernet port is already occupied by another connection, use an USB-to-Ethernet adapter.
 - b. On an Ethernet connection dedicated to the FPGA Platform, the user must manually set the following, refer to Figure 240 for more details.
 1. IPv4 Address to 192.168.1.2
 2. IPv4 Subnet Mask to 255.255.255.0
 - c. The user should make sure that ports listed below are not blocked by firewall software on their PC:
 - 22—SSH protocol
 - 8080—access to the evaluation software on FPGA Platform
 - Note that the ZCU102 FPGA evaluation platform IP address is set by default to: 192.168.1.10.

ADRV910X EVALUATION SYSTEM START GUIDE

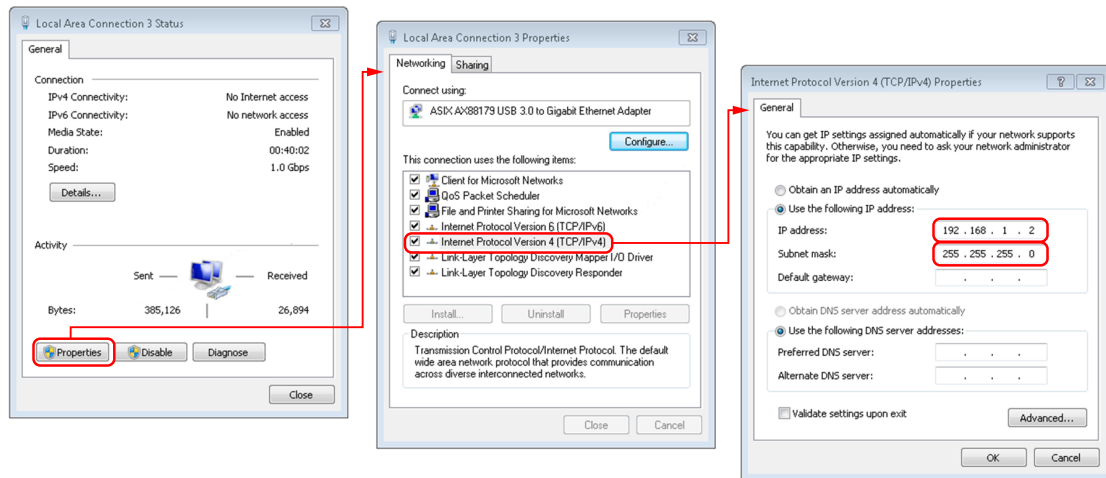


Figure 240. IP Settings for Ethernet Port Dedicated for ZCU102 Evaluation Platform

HARDWARE OPERATION

1. Turn on the evaluation system by switching the ZCU102 evaluation platform power switch (SW1) to the on position.
2. The ZCU102 evaluation platform uses a Linux operating system. It takes approximately 30 seconds before the system is ready for operation and can accept commands from PC software. Boot status can be observed on ZCU102 evaluation platform GPIO LEDs (DS37 – DS40). The correct sequence should follow the description below:
 - a. After SW1 is turned on, all 4 LEDs should start flashing (moving single OFF light).
 - b. These LEDs will continue to flash in the same sequence through FPGA boot up, TES connection and TES program.
 - c. When boot is done 'Init' LED is green and 'Init Done' led is on in the Power and Status LEDs.
3. The **DS1101/DS1102** on ADRV910x EVB should be **ON** which indicates the power supply is good.
4. **DS802** (SYSTEM OK) should be **ON** (for 01d or later hardware revision).
5. The ADRV910X reference clock signal (in range from 10 MHz to 100 MHz, CW tone, 0 dBm typical) should be connected to J701 or switch S702 to "On" to use the 38.4Mhz on board reference clock.
6. For receiver testing on the ADRV910X evaluation card, use a high-performance signal generator with low phase noise to provide an input signal to the selected Rx RF input. Use a shielded RG-58, 50Ω coaxial cable (1m or shorter) to connect the signal generator.
7. To set the input level near the Rx receiver's full scale, the generator level (for a single tone signal) should be set to approximately -12dBm. This level depends on the input frequency and the gain settings through the path.
8. For transmitter testing, connect a spectrum analyser to either Tx output on the ADRV910X evaluation card. Use a shielded RG-58, 50Ω coaxial cable (1m or shorter) to connect the spectrum analyzer.
9. For external LO mode test, the external LO and reference clock signal sources should be synchronized

Note: the on onboard 38.4MHz reference clock might have up to several KHz frequency error, this will cause the RX input signal to be out of the desired RX channel, therefore for RX2 narrowband mode test, we recommend using the external reference clock via J701.

Note: there should be no input signal applied to the Rx input when programming the chip and performing the initial calibration.

Important

The ADRV910X evaluation system uses a Linux operating system. Linux requires time to boot up as well as soft shut down before hardware power off. The user is expected to send the shutdown command manually from TES as shown in [Figure 241](#), before physically switching off power using SW1. If this advice is not followed, the file system on the SD card can get corrupted and the ADRV910X evaluation system might stop operating.

ADRV910X EVALUATION SYSTEM START GUIDE

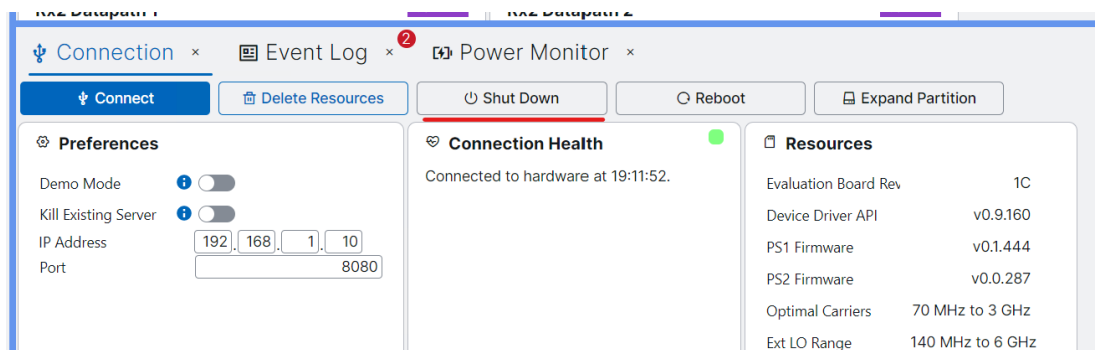


Figure 241. Shutdown ZCU102 System

TRANSCIEVER EVALUATION SOFTWARE (TES)

Installation and Configuration

Contact an Analog Devices representative for access to the ADRV910x software development kit (SDK) download. After the SDK download, unzip the files (unless it is already unzipped). The TES installer locates at folder \adrv910x-sdk\pkg\evaluation.

The TES installer does not demand administrator privileges by default. However, to install the TES to a folder that requires administrative privileges, run the installation process with the requisite administrator privileges. After running an executable file, a standard installation process follows. Figure 242 shows the recommended configuration.

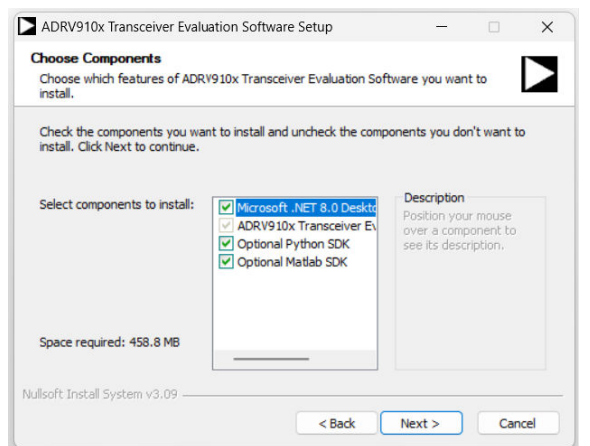


Figure 242. TES Installation Components

Starting the Transceiver Evaluation Software

Start the TES by clicking "Start → ADRV910x Transceiver Evaluation Software". Figure 243 shows the opening page of the TES after activation.

ADRV910X EVALUATION SYSTEM START GUIDE

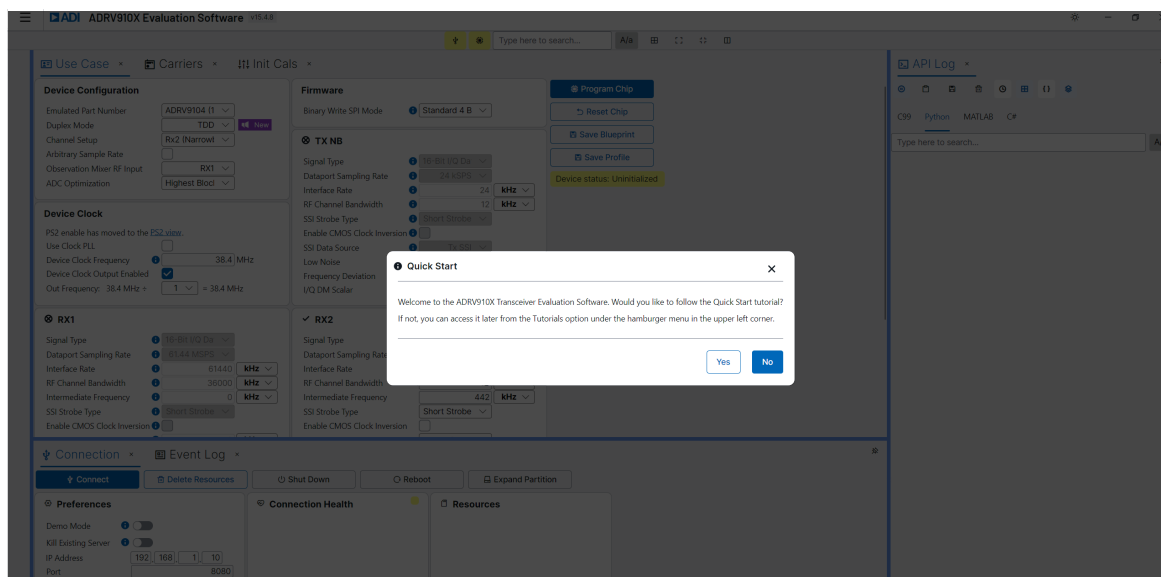


Figure 243. The Start Page of the Transceiver Evaluation Software

The “Quick Start” tutorial is suggested automatically for the first time users, we strongly suggest the users going through the tutorial to better understand how to operate the software.

EVALUATION SYSTEM TROUBLESHOOTING

The following is a quick help guide describing what to do if the system is not operational. This guide assumes the user followed instructions and assembled the setup according to the hardware configuration described in this document.

TES Connection Issues

1. Evaluation hardware platform incorrectly configured (refer to [ZCU102 Hardware Setup](#)).
 - a. Check that the switch on the ZCU102 platform is set properly to boot from the SD card. Find these settings in [ZCU102 Hardware Setup](#)
 - b. Check if FPGA is boot-up properly (refer to [Hardware Operation](#) step 2),
 - c. Check if ADRV910x Evaluation board is in the right FMC connector
2. Ethernet connection (firewall on IP address, port blocked, and others).
 - a. Make sure the firewall settings allow communication to the ZCU102 FPGA platform.
 - b. Check that the IP address for the Ethernet is set correctly. With a direct connection to the PC, the IP address in the TES must be 192.168.1.10 port 8080. Set the LAN connection to 192.168.1.2. With an indirect connection to the PC, set the IP address to the IP address that the router dynamically assigned to the FPGA.
 - c. Check if the Ethernet cable is properly connected between the PC used to run the TES and the ZCU102 FPGA platform. The LEDs on the ZCU102 FPGA platform next to the Ethernet socket must flash when the connection is active.
 - d. If the cable is properly connected, then check if MS Windows OS is able to communicate over the Ethernet port with the ZCU102 FPGA platform. Check if the IP number and open ports for the Ethernet connection used to communicate with the ZCU102 FPGA platform follow the advice described in the section.
 1. Run cmd.exe and then type: ping 192.168.1.10. There must be a reply from the ZCU102 FPGA platform. If there is no reply, re-examine the connection with the ZCU102 FPGA platform.
 2. If the connection with the ZCU102 FPGA platform is established but the TES still reports that hardware is not available, ensure the firewall on the Ethernet connection used to communicate with the ZCU102 FPGA platform does not block ports 22 (SSH) and 8080 (Evaluation Software). Both ports must be open for normal operation.
3. SD card not compatible with FPGA platform, or SD Card file is crashed
 - a. There is a different SD card image required for the different products. Analog Devices provides the SD card imaging software. To find the settings, revisit [SD Card Imaging](#)

ADRV910X EVALUATION SYSTEM START GUIDE

4. Check for physical damage to the EVB,
 - a. The DS1101/DS1102 should be On
5. Manually "Delete Resources" then retry the "Connect", refer [Figure 241](#).
6. If issues still persist, find some common connection issues on the Engineer Zone forum (TBD).

Program Chip Failure

It's common to fail the chip programming if the evaluation system is not set properly before programming the chip, I.E, no device clock or the device clock frequency isn't right, no external LO applied or external LO signal source isn't synchronized with the device clock source, RX input port has signals when programming, etc.

The TES logs the "Exception" event including program failure, unexpected errors when using TES, etc. Users could go to the "Event log" page and find the exception message, then double click the relative message, a new window that contains the exception details and suggestions is popped up for users' reference to debug the issue. [Figure 244](#) shows an exception example in which contains the error details and suggestion.

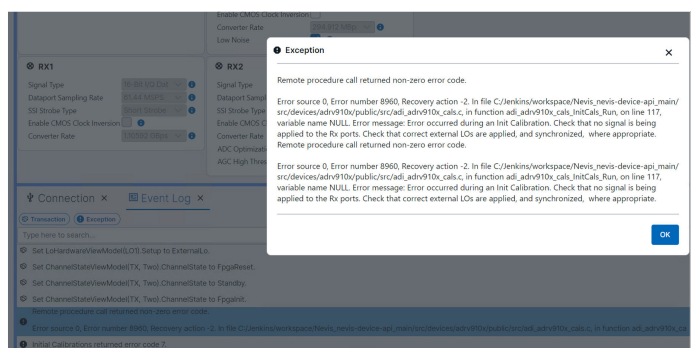


Figure 244. TES Exception event in "Event log"

Always have the exception details information ready when you need to ask the help from Analog Devices representative.

PS2 EVALUATION AND DEVELOPMENT QUICK START GUIDE

Refer to [Resources and Examples](#) for specific driver and SDK details.

PS2 Evaluation Through TES

TES PS2 Features

First, configure and connect to TES as described in [TES Configuration and Initialization](#). A JTag debugger is **not** required for the PS2 demos in TES.

After successfully connecting TES with the ADRV910x evaluation system, under the "Device Config" page, user should check the "PS2 in Data Path" to enable the PS2 subsystem, then user selects and configures the desired profile and initializes the evaluation system by clicking "Program" button.

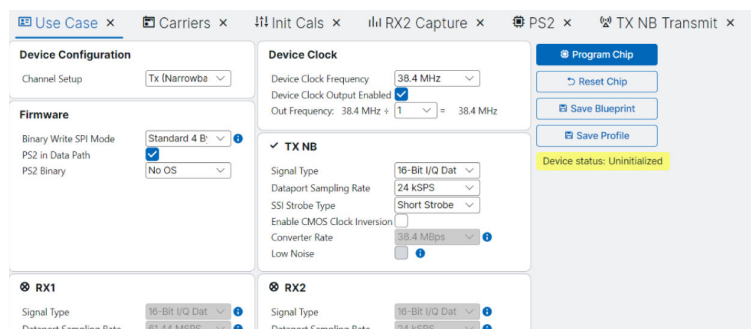


Figure 245. TES Device Config Page

A separate 'PS2' tab is used to select between the PS2 demos as shown in [Figure 246](#)

ADRV910X EVALUATION SYSTEM START GUIDE

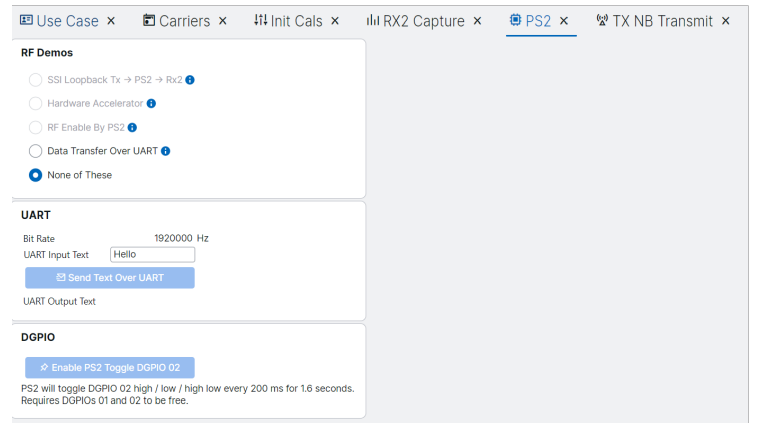


Figure 246. PS2 Page TES

PS2 UART Console

TES provides a UART Console demo use case to show the PS2 UART data communications with the FPGA. User can type any text in the “UART Input Text” box, then after clicking “Send Text Over UART”, the “UART Output Text” is then displayed, which should be the same as the “UART Input Text” if no error occurs.

This demonstrates the following operations:

- ▶ FPGA reads “Hello World!” user inputs in the TES
- ▶ FPGA UART sends “Hello World!” to ADRV910x PS2 UART
- ▶ ADRV910x PS2 receives “Hello World!” through PS2 UART
- ▶ ADRV910x PS2 transmits “Hello World!” to FPGA through PS2 UART
- ▶ FPGA UART receives “Hello World!” and then it is displayed in TES

Note: the UART bit rate in TES is fixed at 1920000 bps under current software support, which is not configurable by user.

Hardware Accelerator Demo

The hardware accelerator demo shows an example use case for signal conditioning. The data is transmitted or received by PS1 and the data is processed by PS2 HWA. The full details of the HWA in this example are given in [Example: Signal Conditioning for an FSK Application](#).

Data Transfer Over UART

The data transfer over UART functions sends and receives IQ data over the UART bus instead of the dedicated SSI lines. This is described in [UART For IQ Transfer](#).

Enable PS2 Toggle DGPIIO

TES provides the PS2 toggling DGPIIO demo function. A mailbox command would be sent to ADRV910x SoC and its PS2 will start a 200ms timer and toggle the DGPIIO2 when timer is expired, there will be 8 toggles on DGPIIO2 each time the button is clicked. A scope probe could be attached to DGPIIO2 on the evaluation board to inspect the GPIO activity.

Scalable ARM Clock

TES provides an example of scaling the PS2 ARM clock for a power saving example as described in [Clock Scaling Examples and Resources](#).

SSI Loopback

The SSI Loopback function performs a loopback from the Rx SSI to the Tx SSI. This uses the to/from memory DMAs as shown in [Figure 181](#). Ensure that both Tx and Rx are enables to observe the full data flow.

ADRV910X EVALUATION SYSTEM START GUIDE

PS2 Prototyping and Production Development

JTAG Debugger

A JTAG debugger is required for PS2 development. The JTAG should be connected to the ADRV910x evaluation board on P502. As described in [Hardware Kit](#), an adapter might be required. The supported JTAG debuggers are:

- CodeFusion studio: Any compatible JLink can be used.
- IAR System: Any compatible JLink or an IJET can be used.



Figure 247. JLink Debugger with Adapter Connection

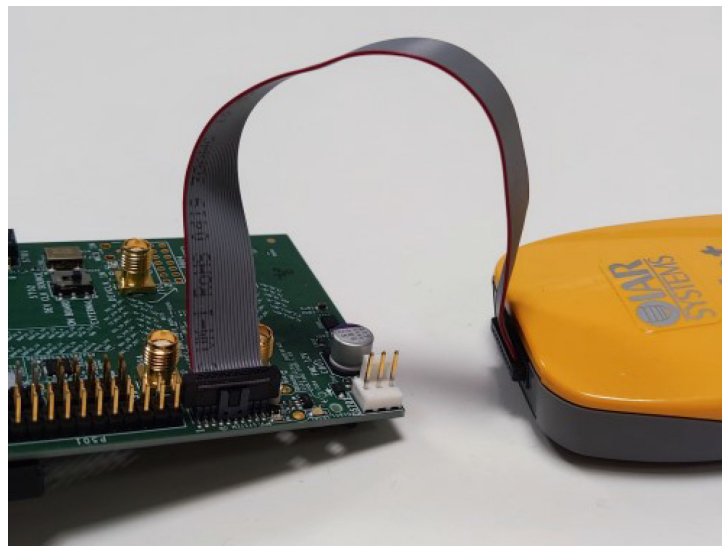


Figure 248. IJet Debugger Connection

Software Development Using Code Fusion Studio (CFS)

CodeFusion Studio (CFS) is an embedded software development platform based on Microsoft's Visual Studio Code (VS Code). CodeFusion Studio provides best in class development tooling for embedded processors and MCUs by providing intuitive tools for newcomers while enabling advanced features for expert embedded developers.

Pre-requisites:

- [Visual Studio Code](#)
- [ADRV910x Transceiver Evaluation Studio \(TES\)](#)
- [CodeFusion Studio™ | Analog Devices](#)

ADRV910X EVALUATION SYSTEM START GUIDE

For CodeFusion Studio setup, please refer to [User guide - CodeFusion Studio](#).

Software Development Using IAR

A Common Microcontroller Software Interface Standard (CMSIS) pack “AnalogDevices.ADRV910X-PS2FW.X.Y.Z.pack” is included in the SDK release package under “adrv910x-sdk\pkg\production” directory. It provides user an example PS2 IAR project for in depth PS2 evaluation and an easier path to start PS2 software development by using the provided PS2 hardware driver APIs. Note the current example project requires an IAR version of 9.32.1 to work properly.

CMSIS Pack Usage

To import the IAR project from the CMSIS pack, follow the steps below (Note the following example uses SDK4 CMSIS pack as an example. Similar procedure should be followed for future SDKs):

1. Open IAR v9.32.1 and create a new workspace under “File” tab.

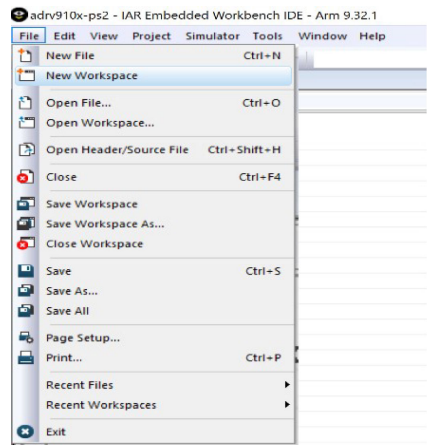


Figure 249. Create a New Workspace

2. Select option “CMSIS-Pack Manager” under “Project” tab.

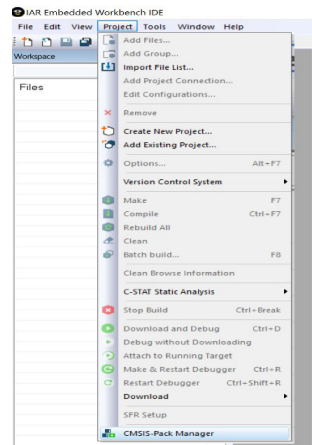


Figure 250. Open CMSIS-Pack Manager

3. Save the workspace when prompted by the IAR tool.

ADRV910X EVALUATION SYSTEM START GUIDE

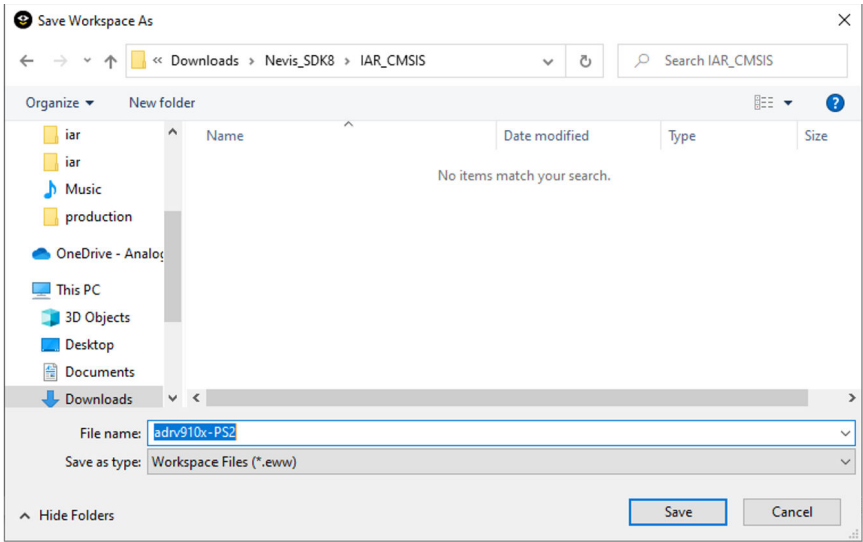


Figure 251. Save the Workspace as Prompted

4. Once the workspace is saved, CMSIS manager screen appears, where user can press on “Import Existing Packs” option on “CMSIS Manager” tab.

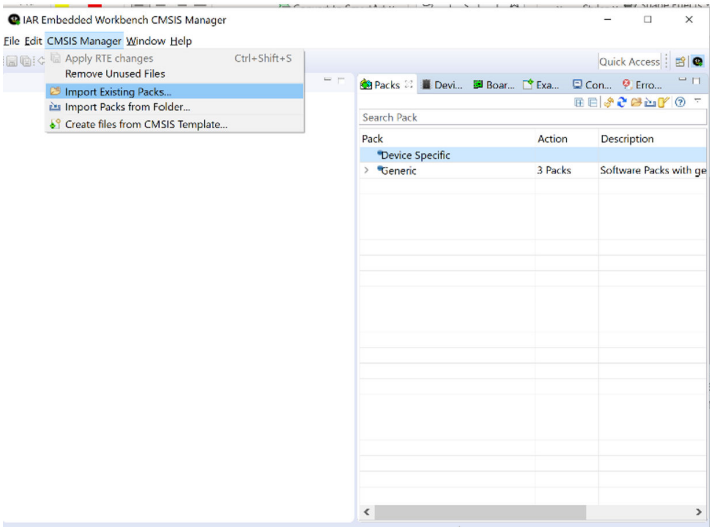


Figure 252. Import Existing Packs

5. Navigate to the folder where the “AnalogDevices.ADRV910X-PS2FW.X.Y.Z.pack” is saved, select and open the pack file.

ADRV910X EVALUATION SYSTEM START GUIDE

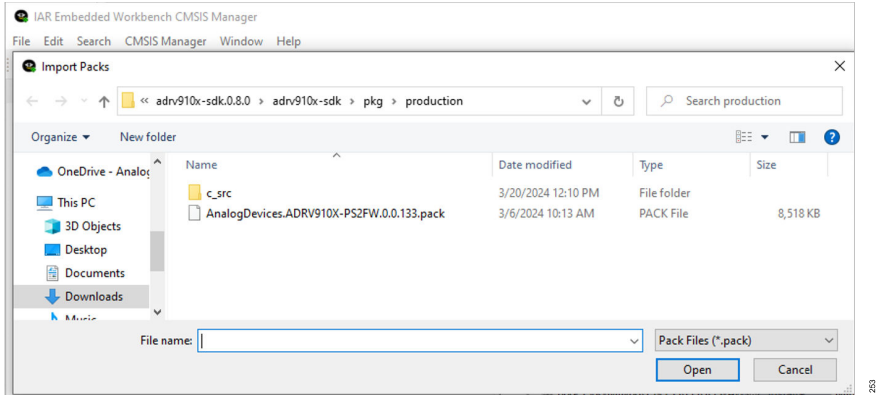


Figure 253. Open the Pack File

6. Read, agree and press OK on the license agreement to unpack the pack file.

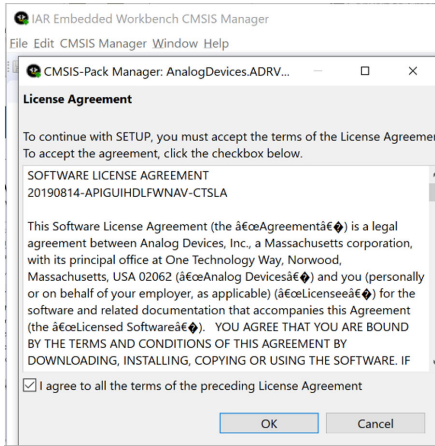


Figure 254. CMSIS-Pack License Agreement

7. Observe the pack file is unpacked

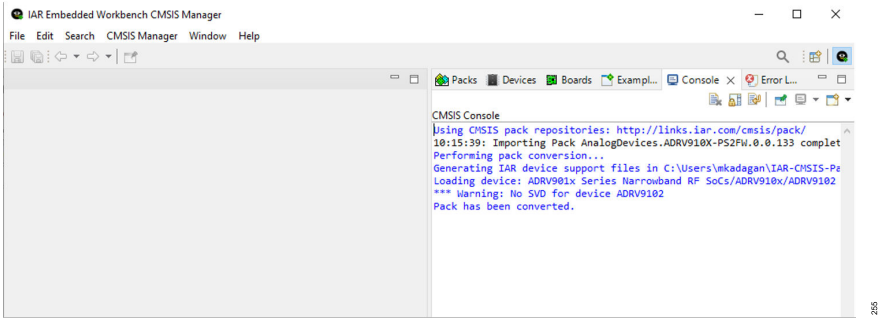


Figure 255. Unpacking CMSIS-Pack

8. Navigate to the "Packs" tab to observe the ADRV910x-PS2FW package in the list of packages under **Device Specific** section (This might occasionally take a few moments to update).

ADRV910X EVALUATION SYSTEM START GUIDE

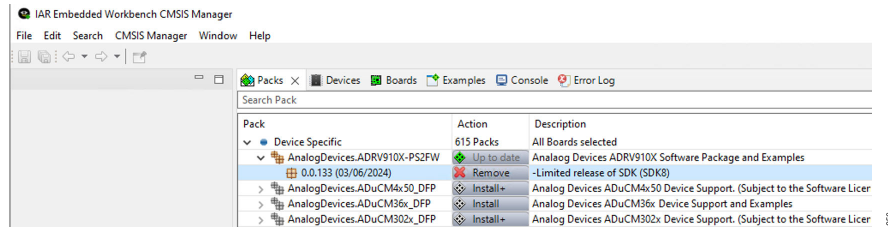


Figure 256. Observe the ADRV910x-PS2FW Package

9. Right click on the pack (with the correct date and pack number) under AnalogDevices.ADRV910X-PS2FW to see the pack's properties.

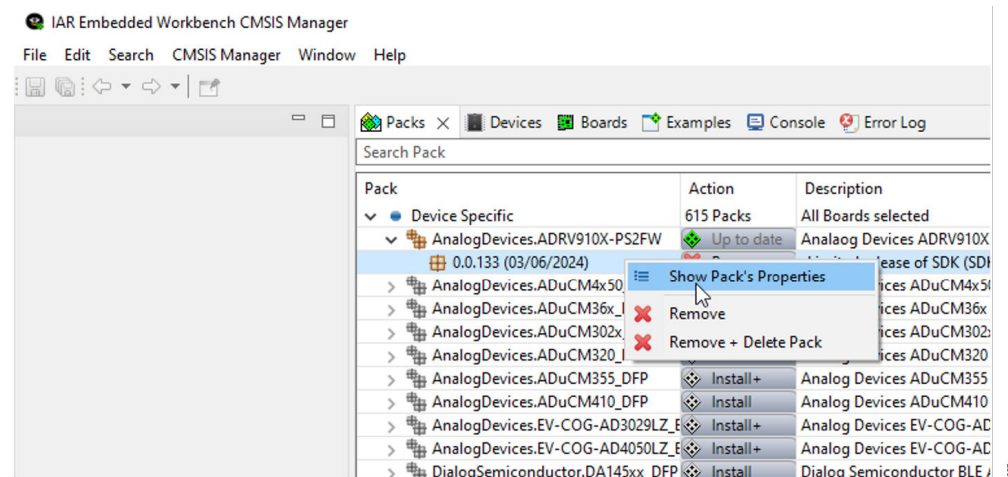


Figure 257. Show Pack's Properties

10. On clicking upon "Show Pack's Properties" button, observe the Packs properties shown as below. Right click and press "Copy" button on ADRV910X PS2 FW Example.

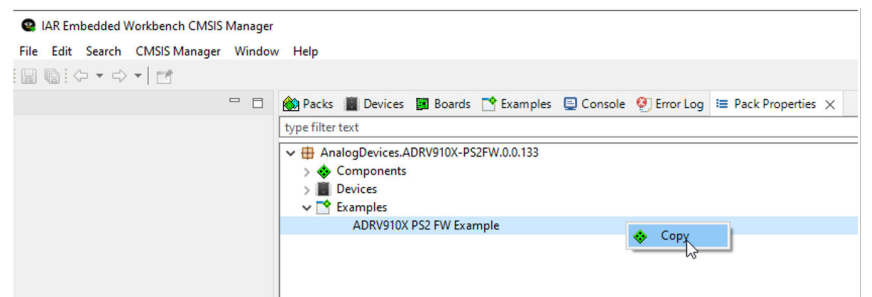


Figure 258. Copy ADRV910x PS2 FW Example

11. Now navigate to the file location where the workspace is created in the second step and observe the example IAR project is unpacked and copied.

ADRV910X EVALUATION SYSTEM START GUIDE

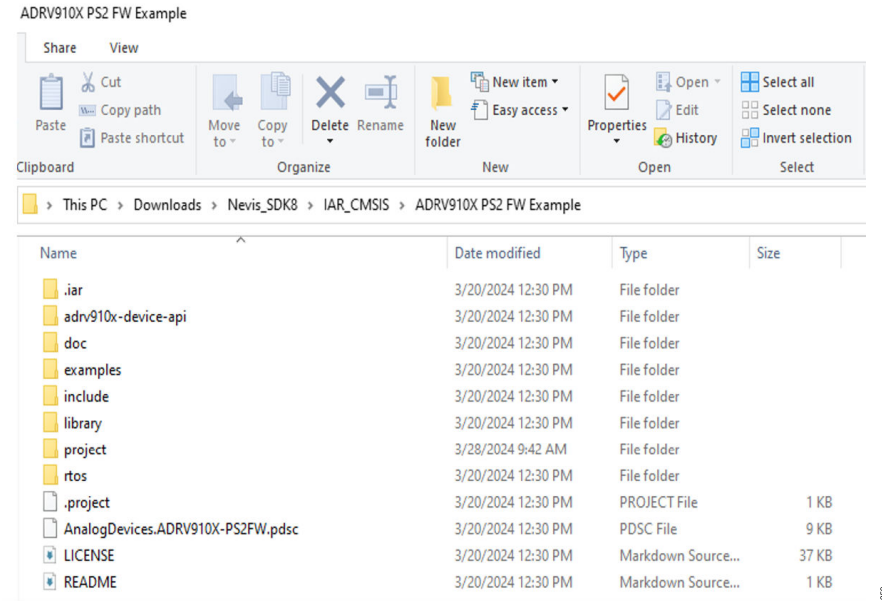


Figure 259. Locate and Observe the Example IAR Project

12. Open ADRV910X PS2 FW Example ->project->iar->adrv910x-pw2.eww to compile and build the project. User may modify the IAR project to add their source code.

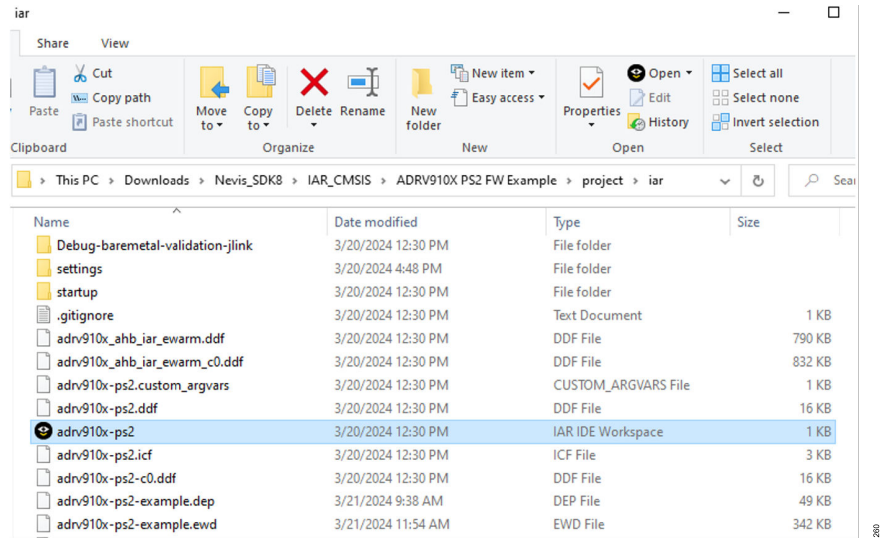


Figure 260. The Example IAR Project File

Compile and Debug the IAR Projects

There are five projects included in the SDK. Each of these projects are described below. The projects can be switched between in the dropdown from the 'workspace'.

ADRV910X EVALUATION SYSTEM START GUIDE

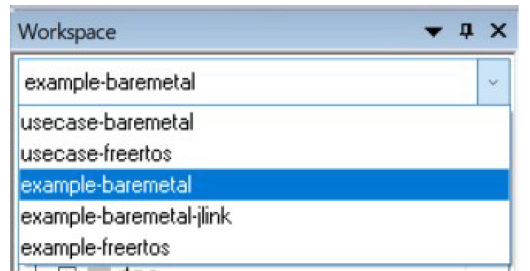


Figure 261. PS2 Projects

To run or debug the binary, user should first initialize the ADRV910x SoC with PS2 enabled if not already done. This could be accomplished by programming any profile with “PS2 in Data Path” selected in TES. Then, user can download the binary to the PS2 memory through JTAG interface and start running/debugging the example use cases. Note a JTAG should be properly connected to the ADRV910x Evaluation Board in advance.

- ▶ **Usecase-baremetal:** This is the source code for the project used for PS2 operation in TES when the ‘No OS’ option is selected. The ‘use-case’ projects serves as a reference file for TES demo use cases, mainly for the datapath DMA and the UART console operations, HWA example use cases, etc. The demo use case should be executed from TES and it involves FPGA operations. Therefore, the “demo_main.c” file is mainly for user’s reference to understand the API calling procedures for those demo use cases. It is not intended for user to test and debug from the IAR project directly.
- ▶ **Usecase-freertos:** This is the source code for the project used for PS2 operation in TES when the ‘Free RTOS’ option is selected. Otherwise, this is similar to usecase-baremetal above.
- ▶ **Example-baremetal:** includes bare metal examples which explain how to use the driver APIs provided. Several examples of UART, Timer, GPIOs, HWA etc. are provided. Using these examples, user can debug and see how the APIs respond when they are called with different configurations. This can be used and modified directly from IAR. This project can be used only with I-jet.
- ▶ **Example-baremetal-jlink:** Same as ‘Example-baremetal’ but allows jlink debugger
- ▶ **Example-freertos:** This project gives an example of how the PS2 baremetal drivers can be used in a Free RTOS example. This project does not show an exhaustive example of all drivers and is intended as an example of Free RTOS operation only.

Note: The executable generated by the Example-baremetal project “adrv910x-ps2.bin” is generated under “ADRV910X PS2 FW Example\project\iar\debug-baremetal-example\Exe” directory. This might be needed for further prototyping and deployment.

NOTES

**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. All Analog Devices products contained herein are subject to release and availability. 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

