SCOPE

This user guide provides information for systems engineers and software developers using the AD9084 and AD9088 family of software defined, direct RF sampling transceivers. This family of devices consists of high performance digital-to-analog converters (DAC) and analog-to-digital (ADC) converters with configurable digital data paths in support of processing signals or RF bands of varying bandwidth (BW). These devices also support various digital features that enhance or simplify system integration. Table 1 outlines the key differences between these devices and the Features section outlines the common features shared among the devices. These devices are further delineated by the digital signal processing features and outlined in Table 2. For full specifications on the AD9084 and AD9088, refer to theAD9084 and AD9088 data sheets, which must be consulted in conjunction with this user guide to achieve successful product selection and design.

Table 1. Product Listing with Distinguishing Features

	Transmitter				Receiver				Digital Features				
Part Number	#DAC chan.	Max DAC Rate	Min Interp Ratio	Max Tx iBW/ch	#ADC chan.	Max ADC Rate	Min Dec Ratio	Max Rx iBW/ch	FSRC	Loop back	Fast Freq Hopping (PFILT/CFIR)	Dynamic configuration	Spectrum Sniffer
AD9084	4(8)[1]	28 GSPS	1x	14 GHz	4	20 GSPS	1x	10 GHz	Yes	Yes	Yes	Yes	Yes
AD9088	8	16 GSPS	1x	8 GHz	8	8 GSPS	1x	4 GHz	No	Yes	Yes	Yes	Yes

[1] 4 DACs (2 per A/B) as signal path DACs. The other 4 DACs (2 per A/B) as signal calibration injection for external dither of LO generation.

Table 2. Digital Signal Processing Feature packages

DSP Trim Package	Product Variants[1]	CDDC & CDUC	FDDC & FDUC	FFH CNCO	FFH FNCO	PFIL	CFIR	FSRC	Dynamic Reconfiguration	FFT Sniffer	Loopbacks	Active DACs
-SW1	MX-DF	X	X	X	X							4
-SW3	MX-DF	X	X	X	X	X	X	X	X			4
-SW3	TX	X	X	X	X	X	X	X	X			8
-SW5	MX-DF	X	X	X	X	X	X	X	X	X	X	8
-SW5	MX-SE	X	X	X	X	X	X	X	X	X	X	8

[1] MX-DF : Both RX and TX with Differential RX; MX-SE : Both RX and TX with Single-ended RX.

TABLE OF CONTENTS

TABLE OF CONTENTS

SYSTEM OVERVIEW

The AD9084 is a highly integrated RF mixed signal front-end (MxFE™) that features four 16-bit, 28 GSPS DAC cores (DAC_A0, DAC_A3, DAC_B0, DAC_B3) and four 12-bit, 20 GSPS ADC cores (ADC_A0,ADC_A1, ADC_B0,ADC_B1), as shown in Figure 1 and Figure 2.

The AD9088 features eight 16-bit, 16 GSPS DAC cores and eight 12-bit, 8 GSPS ADC cores, as shown in Figure 3. The AD9088 shares the same basic architecture of the AD9084 although there is a tradeoff between higher bandwidth (AD9084) and higher channel count (AD9088). The main feature differences between the two devices is the AD9088 does not support Fractional Sample Rate Conversion (FSRC).

Both ADC and DAC cores are differential broadband cores featuring an on-chip buffer input/output with 50ohm termination. The devices include an optional on-chip PLL clock multiplier (7GHz to 14GHz) for DAC and ADC sampling clock generation for less demanding jitter and/or phase noise applications. The transmit and receive digital data paths are highly configurable and support a wide range of single-band and multiband applications with varying RF bandwidth requirements. All the devices feature advanced multi-chip synchronization capabilities to ensure sample accurate pipeline delays between ADCs and DACs on the same chip as well as between multiple chips.

The AD9084 transmit and receive data paths consist of four main data paths in support of wideband signals and eight channelizers in support of narrower band signals. The AD9088 transmit and receive data paths consist of eight main data paths in support of wideband signals and sixteen channelizers in support of narrower band signals. For multiband applications with wide separation between RF bands, the channelizers can be used to process the individual RF bands to reduce the overall complex data rate needed to support narrow non-contiguous bands. Both the main and channelizer data path stages offer flexible interpolating and decimation factors to allow a more manageable data interface rate aligned to the actual signal bandwidth requirements. The numerically controlled oscillator (NCO) of each stage can be independently tuned for the flexibility of frequency placement.

Additional digital features are listed in the Features section.

The SERDES interface supports twenty-four lanes for transmit data and twenty-four lanes for receive data. Both JESD204B and JESD204C protocols are supported as well as the ability to configure quad links in each direction. The JESD204B/C data link layer is highly flexible and allows optimization of the lane count (or rate) required to support a desired data throughput rate. Multichip synchronization and internal synchronization for deterministic latency and phase alignment are supported via an external alignment signal (SYSREF) and a separate trigger input.

FEATURES

Analog Features

Key analog features for the devices include the following:

- Differential RF input/output bandwidth of 18Ghz
- ADC overvoltage protection
- On-chip PLL clock multiplier (7GHz – 14GHz)
- External RFCLK input up to 20GHz
- Single clock receiver or optional dual clock receiver for optimal phase noise

Digital Feature

Common digital features for the devices include the following:

- Transmitter digital up-converter (DUC) and receiver digital down-converter (DDC)
- 96-bit fractional Sample Rate Conversion (FSRC)
- Highly configurable 32-tap, full rate programmable filter (PFILT) supporting multiple profiles for equalizing, averaging, QEC etc.
- Low power, 16-tap programmable complex filter (CFIR) at JESD204 data rate for equalization.
- Dynamic Configuration through SPI/HSCI/GPIO without breaking JESD204 links
  - Coarse NCO/Fine NCO (CNCO/FNCO) Fast Frequency Hopping
  - Fine DDC/Fine DUC (FDDC/FDUC) 1x-64x reconfigured on the fly
  - Coarse DDC/Coarse DUC (CDDC/CDUC) 1x-12x reconfigured on the fly
  - Multiple CFIR/PFILT profiles toggled
  - FSRC setting reconfigured on the fly (SPI/HSCI only)

SYSTEM OVERVIEW

- Multiple loopback (ADC to DAC) supported
- Rx path Spectrum Sniffer/Monitor
- Transceiver gain control and power amplifier (PA) protection
- Power reduction options in Power Duty Cycling mode (TDD mode)
- General purpose input/output (GPIO)
- Multichip synchronization (MCS)

SERDES Interface

Common SERDES features for the devices include the following:

- JESD204B up to 20 Gbps and JESD204C up to 28.21 Gbps
- 24 transmit and 24 receive lanes
- Support for four independent links
- Supports real or complex digital data (8-, 12-, 16-bit)

Control Interface

- SPI 3-wire/4-wire up to 50MHz
- HSCI (high speed control interface) up to 1.6Gbps in DDR mode

DEVICE BLOCK DIAGRAMS

The block diagrams for the AD9084 (both the differential ADC input and the single-ended ADC input) and the AD9088 (with single-ended ADC input) are illustrated in Figure 1, Figure 2, and Figure 3.

SYSTEM OVERVIEW

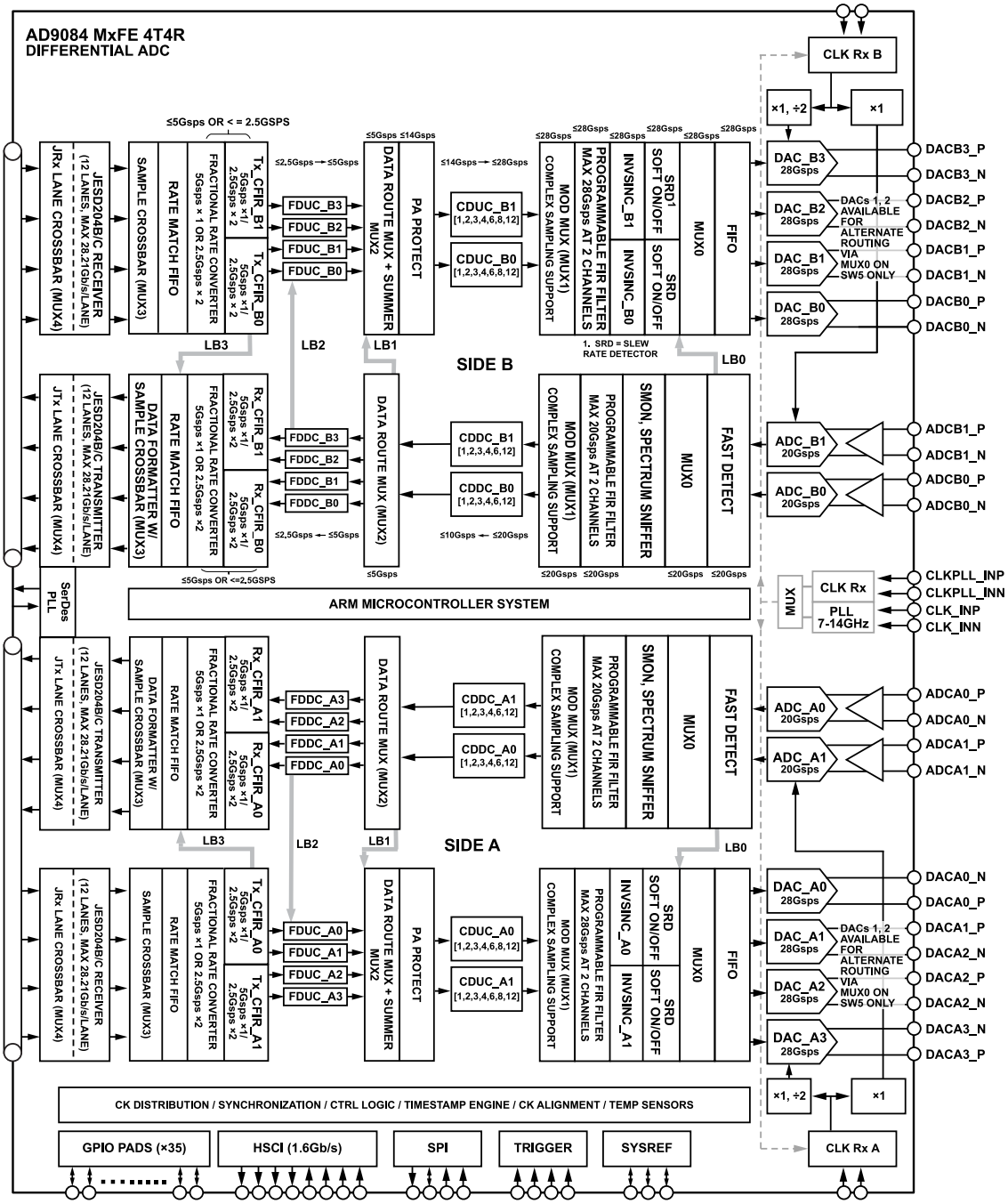

Figure 1. AD9084 Functional Block Diagram (Differential ADC)

SYSTEM OVERVIEW

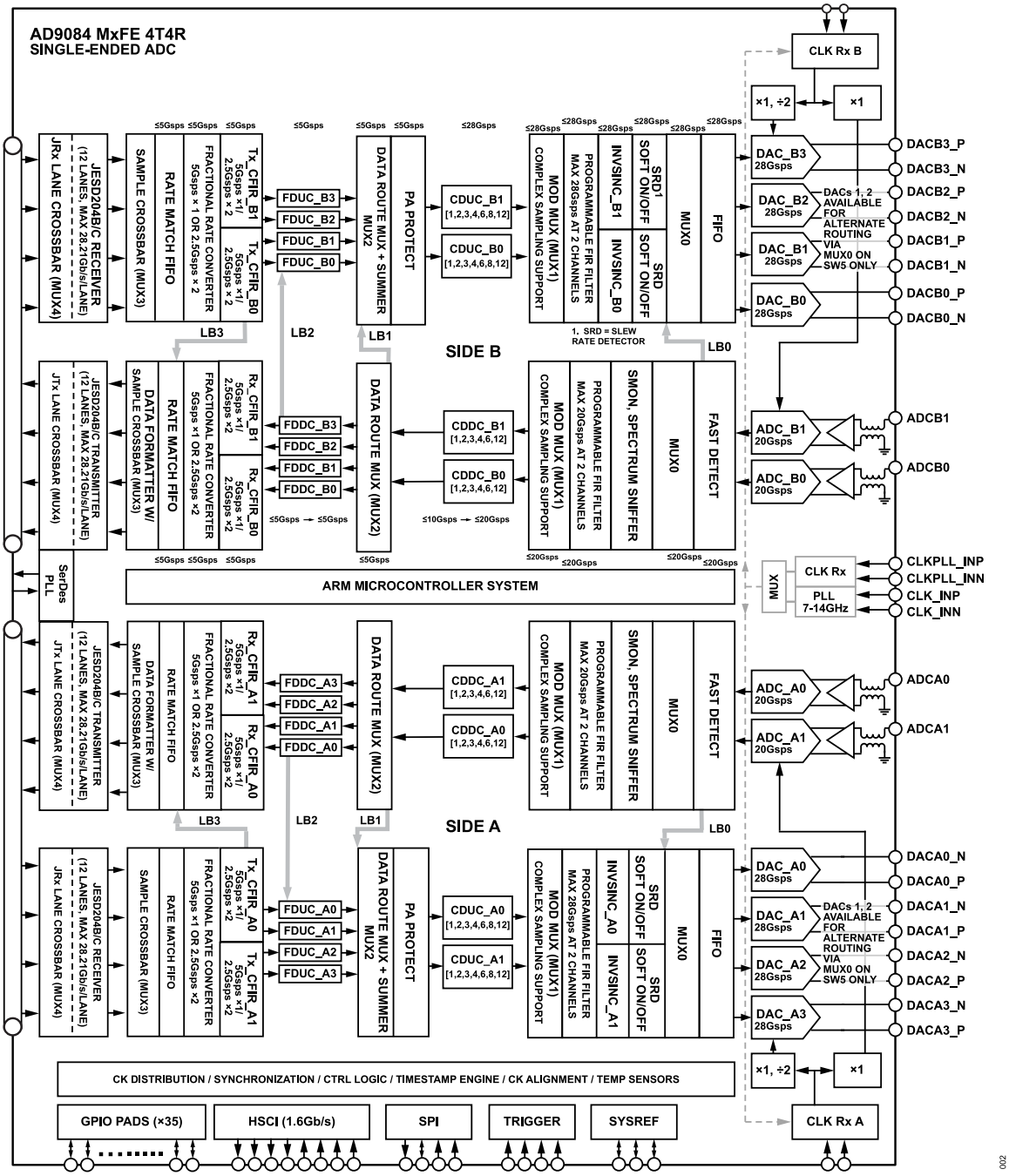

Figure 2. AD9084 Functional Block Diagram (Single-ended ADC)

SYSTEM OVERVIEW

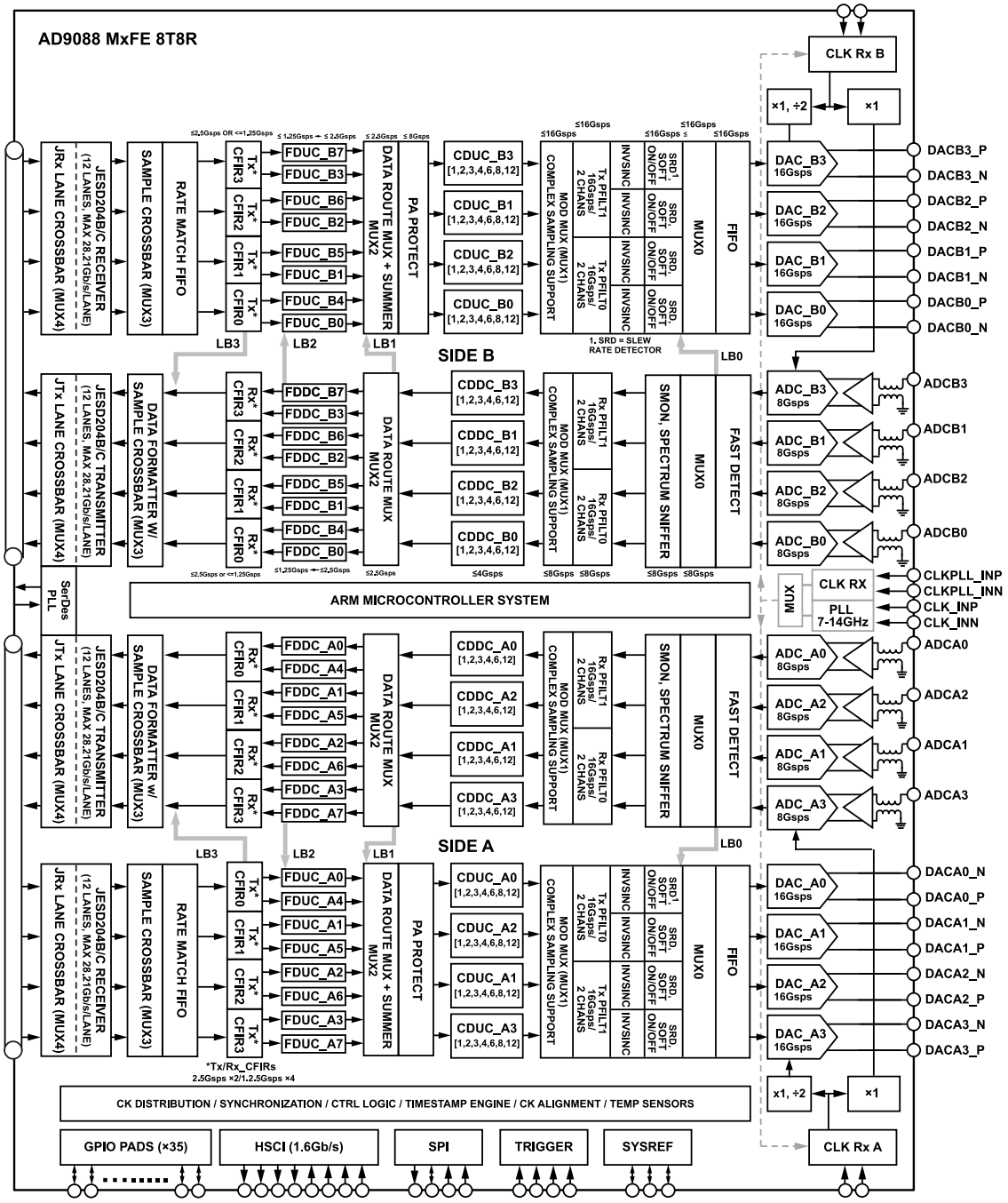

Figure 3. AD9088 Functional Block Diagram (Single-ended ADC)

DOCUMENT NOMENCLATURE

Digital Architecture

The AD9084 and AD9088 top digital is organized into Side A and Side B, and Core Digital. “Core Digital” contains the processor subsystem and MCS/retimer blocks. Side A and Side B are two separate instances of a hierarchical module called “Tx Digital” and “Rx Digital”. Side B is a mirrored version of Side A.

SYSTEM OVERVIEW

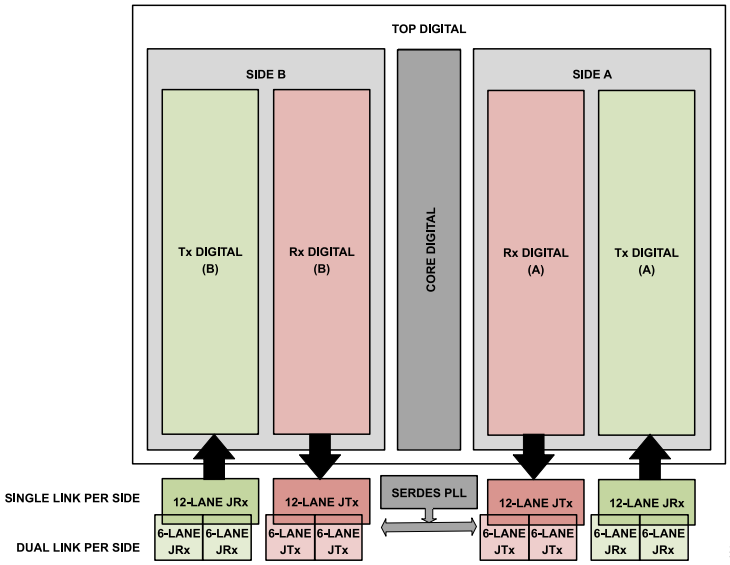

Figure 4. Digital Architecture of AD9084, and AD9088

System Terminology

There are several terms used throughout this document to describe the data flow through both the transmitter and receiver, some of which are used interchangeably. These terms are listed below and defined for both the AD9084 and the AD9088 in Figure 5 and Figure 6.

- Channel = Data path
- Sub-channel
- Data stream - corresponds to the JESD204B/C 'M' parameter. One I/Q pair = 2 data streams

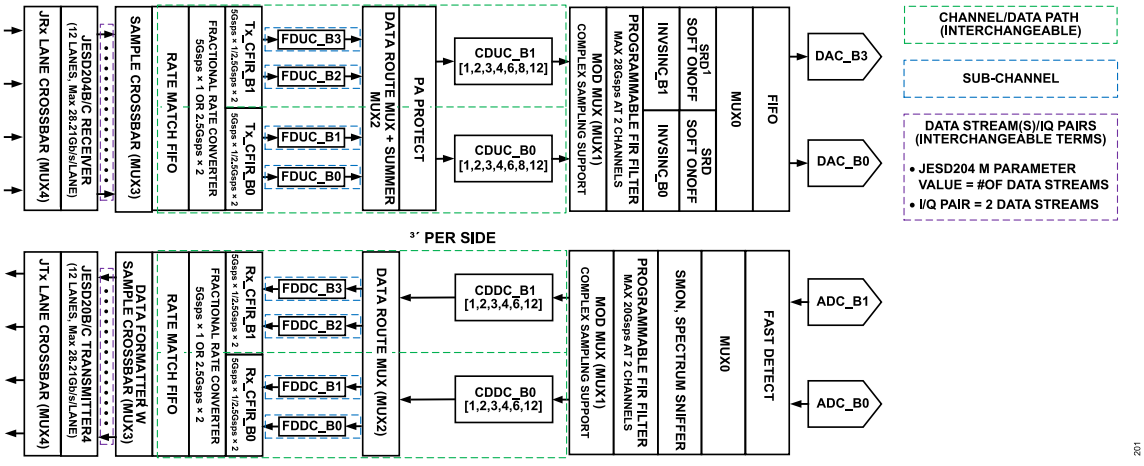

Figure 5. Data flow terminology for the AD9084

SYSTEM OVERVIEW

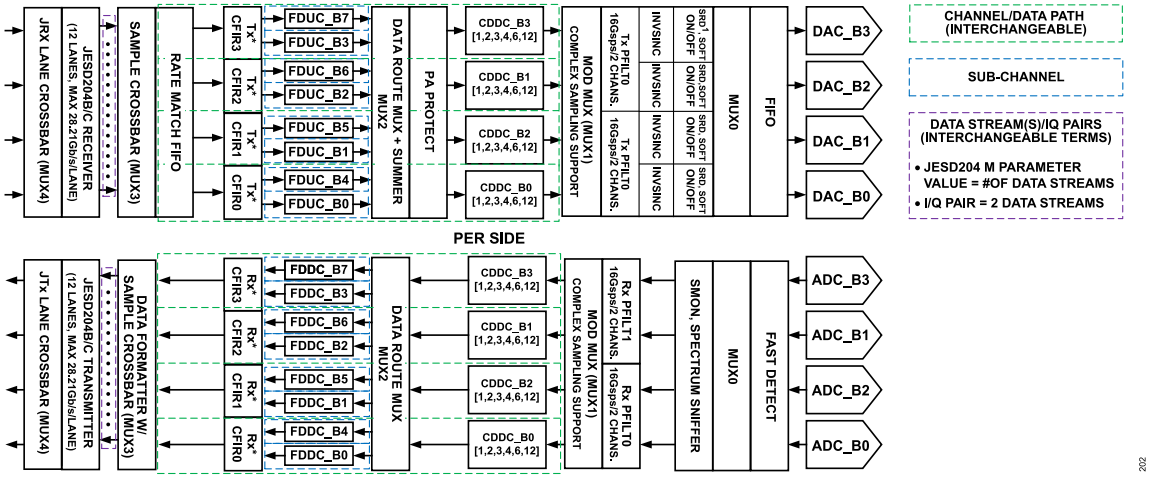

Figure 6. Data flow terminology for the AD9088

Note that the AD9084 has two transmit and two receive channels per side (A and B) and can achieve up to a 4T4R (four transmitters and four receivers) system configuration. The AD9088 has four transmit and four receive channels per side (A and B) and can achieve up to an 8T8R (eight transmitters and eight receivers) system configuration.

ADVANCED DIGITAL FEATURES OVERVIEW

Dynamic Reconfiguration

To maximize the agility of the system and switch rapidly between different RF bands, the AD9084 and AD9088 feature the fast-switching data path configuration (including FDDC/FDUC, CDDC/CDUC and FSRC (AD9084 only)) without breaking the serdes link. While reconfiguring the data path dynamically, both T0 and Tn phase coherence can be supported when FSRC is disabled. See the Dynamic Reconfiguration section for more details.

Fractional Sample Rate Converter (FSRC)

FSRC (AD9084 only) is designed to provide any arbitrary sampling ratio between 1x and 2x to generate the exact desired rates for 4G, 5G, WIFI and other standard rates such as integer multiple of 122.88MHz. See the Fractional Sample Rate Converter (FSRC) for Transmit and Receive (AD9084 Only) section for details.

Transmit and Receive PFILT

The PFILT (Programmable FILT) is a versatile filtering block running at the converter core rate. It can perform a variety of tasks such as Equalization, Crosstalk Correction and Quadrature Error Correction. A total of N=32 filter taps and programmable coefficients per side can be combined in different ways to realize multiple structures for different applications. To allow for fast profile switching, PFILT can store 4 sets of 32 coefficients at a time. The block also has scalar multiplication and bit-shift gain functionality at the outputs which provide flexibility in choosing coefficient sets. See the Transmit/Receive Programmable Filter (PFILT) section for more details.

Transmit and Receive Complex Finite Impulse Response (CFIR) Filter

Each channel of AD9084 and AD9088 at the interface data rate has a 16-tap Complex FIR filter. It can be used for equalizing the amplitude/phase variation within the band of interest caused by RF analog amplifiers, mixers, converters, or impedance mismatches. It also features 2 sets of coefficients for fast profile switching. See the Programmable Complex FIR Filter (CFIR) - Receive and Transmit section for more details.

Loopback Modes

The AD9084 and AD9088 features the following loopback paths:

- Receiver to Transmitter minimum latency Loopback (LB0 - ADC core digital output to DAC core digital input)

SYSTEM OVERVIEW

- Receive CDDC to Transmit CDUC (LB1)
- Receive FDDC to Transmit FDUC (LB2)
- JESD204B/C receiver (JRx) to JESD204B/C transmitter (JTx) Loopback (LB3)

See the Loopback Modes section for more details.

Spectrum Sniffer (FFT)

The “Spectrum Sniffer” (FFT) benefits the applications with the fast indications of where the strong and weak signals are within the digitized signal bandwidth. The FFT can be used for detecting strong signals so that action can be taken in electronic warfare, surveillance applications or RADAR systems. Each side of AD9084 and AD9088 provide two power-averaged overlapped FFTs with programable trigger thresholds that avoid the missing events in time. See the Spectrum Sniffer section for more details.

SOFTWARE OVERVIEW

SOFTWARE INTEGRATION

A typical software integration process flowchart is shown in Figure 7. More details on the integration process, along with an overview of the API and HAL requirements, are provided in subsequent sections.

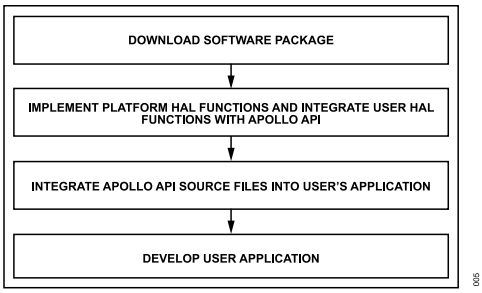

Figure 7. API Integration Overview

API Overview & Deliverables

This section provides information about the software deliverables from ADI, including the ADI-developed Application Programming Interface (API) and resource files essential for the function of the Apollo MxFE. This section outlines the overall architecture, folder structure, and method for using the Apollo MxFE API on a platform. This document will highlight how to use the Apollo MxFE API to control the AD9084 and AD9088.

Software Application Architecture

The Apollo MxFE contains dedicated signal processing blocks, ADCs, DACs, and a dual-core ARM processor. A simplified logical partitioning of a typical software architecture designed with Apollo MxFE firmware and API is in Figure 8. The firmware for the dual-core ARM processor is a pre-compiled binary.

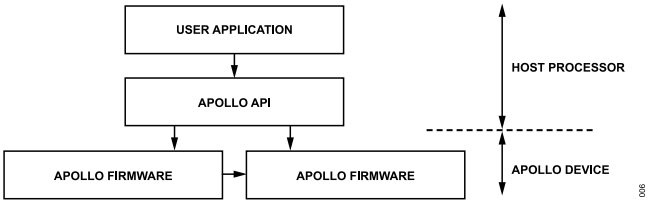

Figure 8. Software Architecture

An Apollo MxFE user application controls the Apollo MxFE through the API. The API C-source code is delivered as part of the software package. The API is processor and operating system agnostic and can be deployed on a bare-metal processor as well as a processor running an operating system.

The API does not alter the configuration or state of the Apollo MxFE on its own. It is the responsibility of the application to configure the Apollo MxFE according to the required mode of operation, poll of status, etc. using the API. The API acts only as an abstraction layer between the application and the hardware.

As an example, the user application is responsible for the following:

- Implementing platform dependent HAL functions like SPI, GPIO, Delay, LOGs, etc.
- Instantiating handles per target ADI device.
- Calling Initialization and Configuration API functions in proper sequence.

A host processor is running an Apollo MxFE based application and controls the Apollo MxFE through the API. The API integrated with the user application relies on a SPI or an HSCI interface in the host processor to interact with the Apollo MxFE chip. The Apollo MxFE firmware running on the embedded dual-core ARM processor consists of setup and calibrations vital to the Apollo MxFE device operation, controlled via the Apollo MxFE API. The API transacts with the firmware through a well-defined set of commands via mailbox interface explained in a later section. The hardware is memory mapped to the embedded dual-core ARM processor through an Advanced High-performance Bus (AHB) interface.

SOFTWARE OVERVIEW

Control commands issued to the embedded dual-core ARM processor in the Apollo MxFE are accompanied by corresponding status and error responses from the firmware. The user application retrieves the command statuses through the API mailbox functions. The user application can also monitor critical system parameters through the general-purpose interrupt status pins.

Mailbox Commands

Mailbox is a server structure within the Apollo MxFE that allows the user to send commands to the firmware of the Apollo MxFE. The Mailbox defines a set of commands that the firmware can recognize and respond to. Each Mailbox command has a separate API function associated with it. These API functions are defined in the adi_apollo_mailbox.h file. The mailbox functions are referenced throughout the user guide in the relevant sections.

Software Deliverables

The software package deliverable follows the structure illustrated in Figure 9. The software package consists of 6 main folders:

- Platform-agnostic:
  - **apollo_api –** API functions for AD9084 initialization, digital data path configuration, etc. The functions contained in this folder are not platform specific.
  - **adi_inc**– This folder contains common variables, error codes, and macros.
- Platform-specific (ADI's Evaluation Platform):
  - **adi_utils** – Generic utility functions used in the API and examples.
  - **c_src –** Non-AD9084 device APIs for CE board attach.
  - **Platforms –** custom HAL for the Evaluation Platform.
  - **Examples – C**ode for configuring and using the Evaluation Platform (Apollo MxFE Customer Evaluation board + ADS10 FPGA board).

Folder Structure

The top-level folder structure of the ADI Apollo MxFE software package is shown in Figure 9. ADI understands that the developer may desire to use a different folder structure. While ADI will provide platform specific and non-platform specific source code releases, the developer may organize the source code into a custom folder structure if desired. Modifying the content of each Apollo MxFE API source file prevents easy updates to future Apollo MxFE API releases.

The code in the 'apollo_api' folder is hardware agnostic.

SOFTWARE OVERVIEW

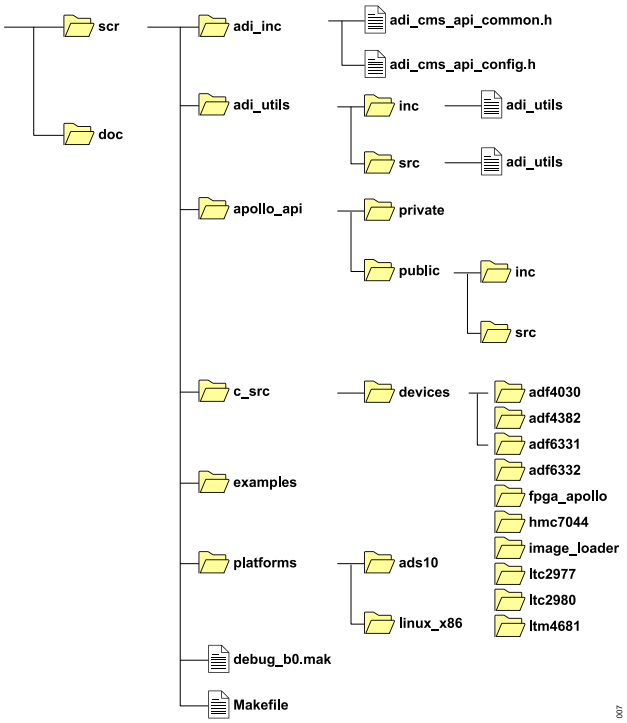

Figure 9. Apollo MxFE Software Folder Structure

Resource Files

The following are the resources required for an Apollo MxFE software package:

- Apollo MxFE firmware (Encrypted application packs)
- Device Profile

This section goes through the steps to generate resource files, which are essential for the Apollo MxFE to function. The resource files are programmed into the Apollo MxFE during the initialization phase. A brief description of the resource files is in Table 3. The resource files require approximately 625 kB of memory.

Table 3. Resource Files

Apollo MxFE Platform File	Purpose	Generation Mechanism	Size
app_signed_encrypted_B	The pre-compiled firmware binaries for the embedded dual core Arm processor in the Apollo MxFE, which mainly consists of ADI proprietary algorithms used to calibrate the Apollo MxFE	Delivered as part of the Apollo MxFE software package	600KB
Device_profile	The Apollo MxFE's configuration for a particular use case is programmed through the profile byte-array.	Generated by Analog Devices or with the device configurator	21KB

Firmware Binaries

The firmware for the embedded dual-core ARM processor in the Apollo MxFE is delivered in the form of pre-compiled binary files. The firmware in the Apollo MxFE mainly consists of ADI proprietary algorithm used for calibration, and drivers to access the Apollo MxFE hardware resources.

The firmware binaries are delivered in the software package. It is required to program the firmware binaries as part of the Apollo MxFE initialization. Refer to the programming section for information about initializing the Apollo MxFE.

Device Profile

The device profile is a file that can be generated in multiple formats (.h, .bin, .json, etc.) that contains all the parameters that are needed to configure the Apollo MxFE for a specific use case. It contains an **apollo_top_t** structure which consists of the clock rates, filter coefficients, data path configuration, and the JESD204 configuration for the use case. The API in tandem with the firmware uses the profile to configure the

SOFTWARE OVERVIEW

Apollo MxFE during initialization. Profile loading is done inside the **seq_profile_load(adi_apollo_device_t *device, adi_apollo_top_t *profile)** api located in **src/apollo_api/public/src/adi_apollo_startup.c**

The device profile is generated by ADI for initial samples. Device profiles can also be generated with the profile generator tool that is part of the ACE Evaluation Software. The same profile generator base will be exposed through Python as well. A description of how to use the profile generator can be found in the Apollo MxFE® Evaluation Board User Guide in the "Profile Generator" sub-section of the "Apollo MxFE features currently supported in ACE" section.

Documentation

Analog Devices uses Doxygen to generate a navigable (via cross-references) html document of the API using specially formatted comments within the code. In addition, a pdf document is provided as a reference manual based on the Doxygen output. Both files are included in the API package in the pkg\production\doc folder.

API Integration and Build: New Design

This section provides an overview of integration and building steps required when using the ADI's API source code with a custom hardware design.

ADI provides the full source code, and the user can customize and build the libraries as per their application. However, for a custom hardware design, users must integrate the API with a custom implementation of the defined Hardware Abstraction Layer (HAL) methods that are specific to their platform. The HAL allows translating API calls into specific Control Interface calls, as described in more detail in the Control Interfaces section.

To simplify the integration process, the API was developed using the C99 standard. The C99 standard was followed to ensure agnostic processor and operating system integration with the API code. For integration flow, please refer to Figure 1.

An example of how ADI wrote the HAL specific to using the Apollo MxFE API for the AD9084 Evaluation Platform (AD9084 Customer Evaluation board + ADS10 FPGA) is found in the ADI Evaluation System Software section.

Integrating Apollo MxFE API

There are four steps to integrate the Apollo MxFE API into an end application:

1. Implement the Platform HAL Functions

The API requires access to several platform specific, hardware and system control functions such as a system delay/sleep function, SPI bus controller functions, etc. The end user must provide and implement these functions as per the ADI device requirements.

Users will develop their own HAL functions based on their hardware-dependent platforms. Therefore, depending upon their platform, they will use different drivers for the peripherals such as the SPI and HSCI interfaces.

The Apollo MxFE API was developed such that developers can use any driver of their choosing for their platform requirements. However, there are a few platform-dependent functions in the API's HAL. Modifying the signature (or prototype) of the HAL functions is not recommended as one would then have to potentially change a lot in the API source code as well. Instead, users need to write their own platform functions based on these prototypes in "**adi_cms_api_common.h**" under the ***adi_inc*** directory for specific platform requirements.

In general, for all API specification, the following HAL members are required for correct operation of the Apollo MxFE API

- **hal_info.spi0_desc.read**, Pointer to SPI data read function for the Apollo MxFE hardware.
- **hal_info.spi0_desc.write**, Pointer to SPI data write function for the Apollo MxFE hardware.
- **hal_info.delay_us**, Pointer to delay function for the Apollo MxFE hardware.

2. Include the Apollo MxFE API Header Files

The header files under folders ***/adi_inc*** and under ***/apollo_api/inc/*** defines the interface to the Apollo MxFE API and should be included in the application.

3. Instantiate the Apollo MxFE Device Handle

For each Apollo MxFE device, the application must instantiate a unique Apollo MxFE handler reference.

SOFTWARE OVERVIEW

Refer to the **adi_apollo_device_t** section for a full description of the Apollo MxFE handler.

Notes on Apollo MxFE API Integration

Any API is not just limited to the data structures and its structure members listed in **adi_apollo_device_t**. These are the most common structures that each API share. As per the device requirement, new data structures and structure members can be added by ADI. If so, its details would be documented in device-specific API literature.

For each handler instantiated by the application, all the required members of the device handler must be initialized prior to calling any APIs with that handler as a parameter.

Along with HAL members, other peripheral interfaces should also be initialized prior to using any API functions.

Example Apollo MxFE device instantiation

```
static void configure_hal(adi_apollo_device_t *device)
{
    adi_apollo_hal_t *hal = &device->hal_info;
    adi_apollo_hal_regio_spi_desc_t *spi0_desc = &hal->spi0_desc;
    adi_apollo_hal_regio_spi_desc_t *spi1_desc = &hal->spi1_desc;
    adi_apollo_hal_regio_hsci_desc_t *hsci_desc = &hal->hsci_desc;

    /* Create platform object */
    adi_apollo_hal_config_t* ads10_platform = ads10_apollo_hal_instance();     // Allocates memory for the platform HAL data structure.

    hal->dev_hal_info = ads10_platform;
    hal->log_write = &ads10_log_write;
    hal->delay_us = &ads10_wait_us;

    /* SPI 0 Config */
    spi0_desc->is_used = 1;
    spi0_desc->spi_config.addr_inc = ADI_APOLLO_DEVICE_SPI_ADDR_INC_AUTO;
    spi0_desc->spi_config.msb = ADI_APOLLO_DEVICE_SPI_MSB_FIRST;
    spi0_desc->spi_config.sdo = ADI_APOLLO_DEVICE_SPI_SDO;
    spi0_desc->dev_obj = ads10_platform->spi0;
    spi0_desc->xfer = NULL;
    spi0_desc->read = &ads10_spi_read;
    spi0_desc->write = &ads10_spi_write;
    spi0_desc->poll_read = NULL;

    ads10_apollo_hal_config_data(device, 0, 1);     /* chip selects: SPI0=0, SPI1=1 */

    /* SPI0 config - not used */
    spi1_desc->is_used = 0;

    /* HSCI config - not used */
    hsci_desc->is_used = 0;
}
```

Figure 10.

```
adi_apollo_device_t apollo_dev = {

    .hal_info = {
        .user_data = "hello",
        .active_regio = &apollo_dev.hal_info.spi0_desc.base_regio,     // default to spi0
        .dev_hal_info = NULL,
        .hw_open = NULL,
        .hw_close = NULL,
        .reset_pin_ctrl = NULL,
        .delay_us = simple_delay_us,
        .spi0_desc = {
            .is_used = 1,
            .spi_config = {
                .addr_inc = ADI_APOLLO_DEVICE_SPI_ADDR_INC_AUTO,
                .msb = ADI_APOLLO_DEVICE_SPI_MSB_FIRST,
                .sdo = ADI_APOLLO_DEVICE_SPI_SDO
            },
            .dev_obj = "no object",
            .init = NULL,
            .xfer = &spi_xfer,
            .read = NULL,
            .write = NULL,
            .poll_read = NULL,
        },
    },
};
```

Figure 11.

4. Create the Application

Using the Apollo MxFE APIs provided in the header files under ***/src/apollo_api/public/inc/*** write the application code to initialize, configure, monitor and log the Apollo MxFE device as per the target application requirements.

Example files are in the 'src/example' folder. The ADI Evaluation System Software Examples section describes these available functions and tests.

SOFTWARE OVERVIEW

Apollo MxFE Bring-up Procedure

The steps to bring up the Apollo MxFE in a specific use case are shown in the flowchart in Figure 12.

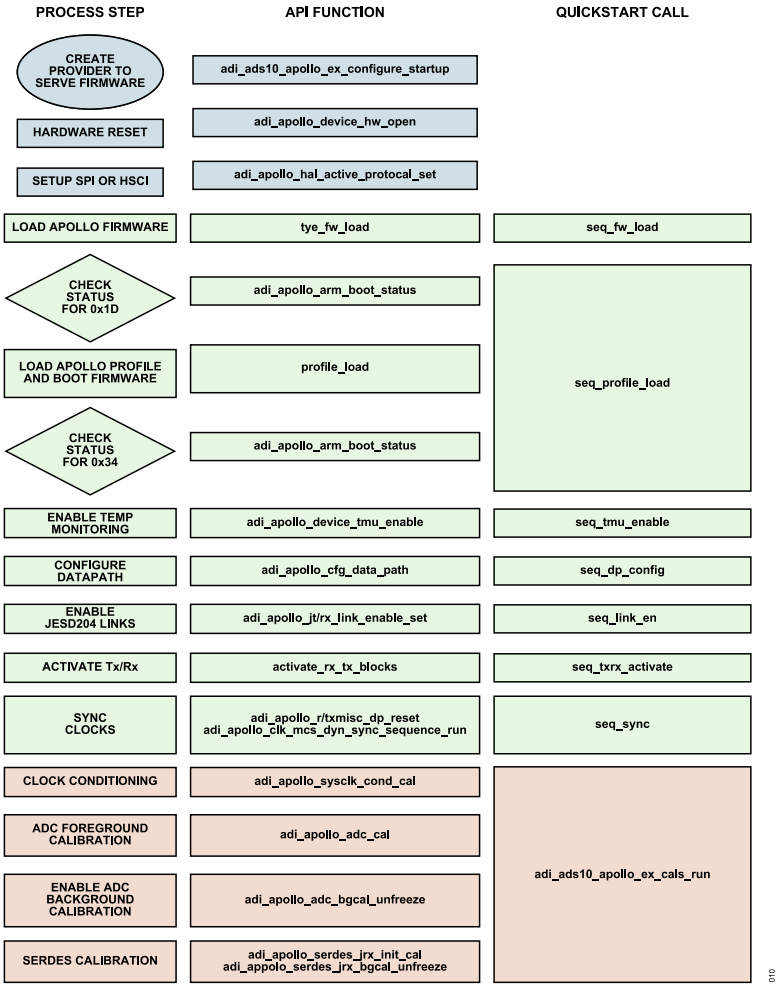

Figure 12. Apollo MxFE Bring-up Flowchart

The bring up flow chart has block descriptions on the left, API functions at center, and the quick startup calls at right (where applicable). Example bring-up can be seen in any of the example programs included in the api package at src\examples\ads10_apollo_ex_main. All these examples follow the blue and green flows above as part of main.c in the same examples directory, with some variation thereafter depending on the specific example code. The apis in blue are executed explicitly inside of main.c, and all the calls in green are executed as sub-steps of the top level **adi_ads10_apollo_ex_startup** called in main.c. **adi_ads10_apollo_ex_startup** takes the enum adi_apollo_startup_seq_type_e defined in src\apollo_api\public\inc\adi_apollo_startup_types.h, which allows the user to select which of the startup steps they would like to execute. The default enum runs all of the startup steps. Main.c will then switch to executing the specific example program, which can include any of the listed orange calibration calls. The calibrations defined by the enum adi_ads10_apollo_cal_e in src\examples\ads10_apollo_ex_common\include\adi_ads10_apollo_ex_types.h can be given as arguments to **adi_ads10_apollo_ex_cals_run**.

The definitions of green and blue apis listed in the flow above can be found at src\apollo_api\public\src\adi_apollo_startup.c. First, a provider is generated containing the details of the fw binaries to be loaded. After the hardware reset, the user needs to set the communication protocol, then load the firmware from the generated provider structure. The call to **tye_fw_load** will load the files inside src\examples\fw_images\b0\app_signed_encrypted_prod_B and an internal security engine will authenticate the firmware that is provided and encrypted by Analog Devices Inc. Any alternative application files that are loaded during this step will cause an Apollo MxFE boot failure. The call to **tye_fw_load** must complete within 60 seconds of the release of reset on the AD9084 or boot will not be permitted.

Once the firmware load has completed, the user needs to load a device profile. For more information on profiles, see the Device Profile section. After confirming the profile has been loaded, the device firmware will configure the PLLs and internal clocking, as well as the JESD204B/C

SOFTWARE OVERVIEW

transmitters and receivers, based on applicable values in the device profile. Once this firmware boot process is complete, the temp monitoring unit can be optionally enabled via **seq_tmu_enable**. Next, **seq_dp_config** takes all the settings that were specified in the device profiles and configures the DSP accordingly. **seq_link_en** brings up the jrx and jtx links that are listed as enabled in the profile, **seq_txrx_activate** powers up all dacs and adcs, and **seq_sync** syncs clocking zones. Calibrations can then be run as needed via **adi_ads10_apollo_ex_cals_run**. Clock conditioning should be run for all bring-ups. ADC foreground and background cal should be run for any modes using the ADCs. Serdes calibrations are performed based on the profile settings. Once the device has been configured to the end of the bring up flow chart, the user should not toggle any block enables as another sync sequence will need to be issued which will cause a JESD204 interruption.

All ADCs will be enabled upon start-up for optimum lifetime performance. The associated receive digital path can be disabled for unused ADC channels. Similarly, all transmit serdes lanes are enabled by default. In systems where certain ADC channels and/or serdes lanes will remain disabled for the lifetime of the system, the user can request from Analog Devices a method of placing those ADCs and serdes lanes into a deep sleep mode during boot. Contact the local ADI sales representative or send an email to ApolloSupport@analog.com.

Initial Data Path Configuration

As stated above, the initial data path configuration from the profile is handled by **adi_apollo_cfg_data_path**. This API function also serves as an example of how to configure each block for the first time.

Associating API Functions With the Apollo MxFE Hardware (or DSP) Blocks

When associating the Apollo MxFE DSP blocks with API functionality, the standard adi_apollo_<block> can be followed. For example, when looking for API functions associated with the CDUC, block would be cduc. The API function descriptions would be found in adi_apollo_cduc.h.

Some higher-level API functions will not follow the block format but the naming where block would be is explicit. For example, when looking for data path configuration API functions, the right place to look is adi_apollo_cfg.h.

Configuration Updates After Device Bring-Up

There are three categories of functions with regards to when they may be configured or reconfigured:

1. Functions only configured at device startup – digital reset and sync is required
2. Static functions that can be changed after device startup – digital reset is not required
3. Dynamically reconfigurable functions – see Dynamic Reconfiguration section

Functions That Can Only Be Configured at Device Bring-Up

Any functions not covered by the lists provided in the Functions Requiring "Dynamic Reconfiguration" or Reconfigurable Static Functions sections above must be configured at device bring-up using the parameters stored in a device profile. Note that enables need to be done before the data path resets and sync as well. Once the blocks are enabled, parameters can be changed, excluding the enable, without issuing another sync.

Reconfigurable Static Functions

The following Apollo MxFE functions can be changed after device bring-up without the need for a digital reset.

- Test mode pattern generation, including delays for loopbacks
- Test mode pattern checking
- JESD204B/C transmitter lane power down
- Receiver and transmitter enable (RXEN, TXEN)
- PFILT coefficient changes and profile hopping
- CFIR coefficient changes and profile hopping
- NCO phase and amplitude dither
- NCO Frequency Tuning Word (FTW) profile hopping
- Data path gain in the CDUC/FDUC and CDDC/FDDC can be changed but not the enable/disable
  - Receive path
    - CHB1, CTB1, FHB1
  - Transmit path:
    - FDUC input, before FHB1, GAIN block in HSDOUT, after INVSINC block

SOFTWARE OVERVIEW

Functions Requiring "Dynamic Reconfiguration"

The following data path changes require the use of Dynamic Reconfiguration when not planning on bringing the link down first.

- Decimation ratio changes using CDDCs and FDDCs
- Interpolation ratio changes using CDUCs and FDUCs
- FSRC ratio changes

ADI EVALUATION SYSTEM SOFTWARE

This section outlines the Apollo MxFE API structure when integrated with a HAL and Application layer specific to the Apollo MxFE Evaluation Platform (AD908x Customer Evaluation board + ADS10 FPGA board).

ADI EVALUATION SYSTEM SOFTWARE ARCHITECTURE

The software architecture of the Apollo MxFE ADI evaluation system is in Figure 13. The architecture of the ADI evaluation system broadly follows a client-server model, with the server residing in the microZed (uZed) on the ADS10. The client application software, such as ACE or the Apollo MxFE Python Application, can be a PC based application. A PC based application client interacts with the server through a set of C# DLLs (Dynamic-Link Library) referred to as EvalClient.

The web Remote Procedure Call (webRPC) uses a pre-compiled API object, which cannot be re-compiled to include new code or use a new API version.

The webRPC interface can be bypassed, and the user can compile their own API version directly on the host ("Embedded App" in Figure 13). There is a growing library of Example code that may be compiled and ran to directly configure the Evaluation Platform.

The host processor on the ADS10 consists of an FPGA host integrated with an embedded processor running a Linux operating system. The Apollo MxFE API is integrated within the Example the host processor software to control the Apollo MxFE. The platform HAL interface (SPI, HSCI, timer, etc.) is implemented for the ADI Apollo MxFE evaluation system, which in-turn is used by the Apollo MxFE's API to interact with the Apollo MxFE.

When running the evaluation software, a command is issued by a PC based application client to the webRPC Server running on the host processor located on the ADS10, through the webRPC transport layer interface. The command is translated into a set of calls to the API, which exists in the server as compiled C code. Each API call becomes a set of HAL function calls, to do transactions to the Apollo MxFE through a SPI or HSCI hardware interface. When a read command is issued, the read value is sent back to the application layer through the webRPC transport layer link.

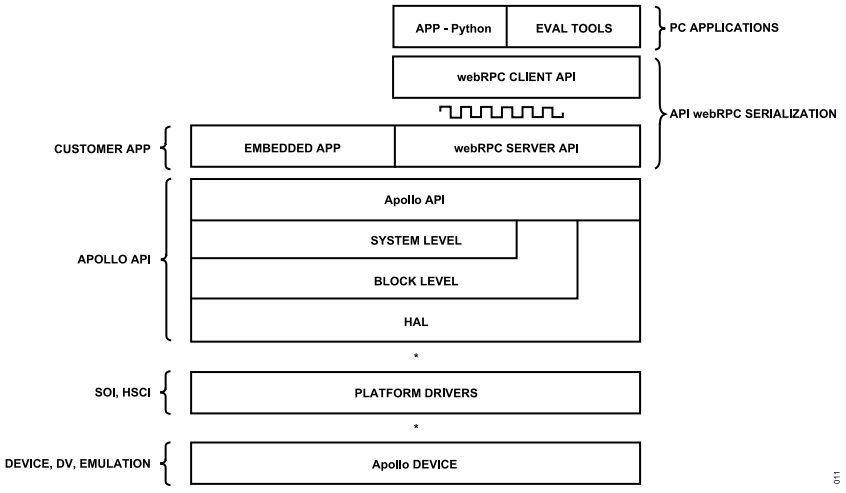

Figure 13. ADI Evaluation System Architecture

ADI EVALUATION SYSTEM SOFTWARE EXAMPLES

There is a library of examples that demonstrate how various Apollo MxFE configurations may be implemented in a system comprised of an FPGA together with an Apollo MxFE, along with the required clocking tree and power chips. These system-level examples were written and

SOFTWARE OVERVIEW

tested on the Apollo MxFE Evaluation Platform (commonly comprised of the AD9084-FMCA-EBZ evaluation board and the ADS10-EBZ FPGA board).

The examples are extremely useful to reference when the API is integrated into a new application in a custom system. The following examples, part of ads10_apollo_ex_main, are of a particular value:

1. verify JRx stability / functionality (Tx path and DAC core are inactive): ***prbs.c***
2. verify the Rx path and JTx functionality (ADC and the analog front-end are inactive): ***rx_jesd.c*** utilizing the -b command line flag to enable the AWG Mode at the ADC output.
3. verify the DAC and Tx front-end: ***tx_nco.c***
4. verify the ADC and Rx front end: ***rx_adc_nz.c***
5. verify the Apollo MxFE data paths, including DAC and ADC cores and the JESD204B/C links: ***fullchip.c***

Ads10_apollo_ex_common

The common folder contains various common functions that can be called from the main example programs. It contains bring-up functions, such as the Apollo MxFE and FPGA HAL configuration, the generation of the FW provider structure, SPI tests, and a wrapper to call the Apollo startup sequence (defined in src/apollo_api/public/src/adi_apollo_startup.c); some common block configuration functions, such as 'adi_ads10_apollo_ex_cnco_freq_set', 'adi_ads10_ex_pfilt_coeff_file_load', and 'adi_ads10_apollo_ex_fpga_capture' which can be used to capture and write data to a file; as well as status and debug functions including 'adi_ads10_apollo_ex_jesd_rx_status' for link status, 'adi_ads10_ex_dp_info_get' for datapath parameters, and register dump functions for the FPGA and Apollo. The header files inside the 'include' folder contain descriptions of the functions and their parameters to reference.

The code in this file is accessible from all examples in the ads10_apollo_ex_main folder.

Ads10_apollo_ex_main

This folder contains multiple examples and profiles that configure Apollo. For all examples, the device is setup and the JESD links are configured as part of the main.c source file. After basic bring up in main.c based on the profile and clocking parameters provided in the command line (described below) the chosen example source file (also provided in the command line along with any example-specific arguments) will be executed. In some examples, the JESD links are not brought up, like tx_nco, where links are not needed. It will run whichever example was specified in the command line. The examples are described in Table 4. Many of the below examples include setup details in their respective headers at the top of their .c file. These files will include a note to "refer to the example description" in their description in the table.

To run any of the examples, in Table 4, navigate to the 'ads10_apollo_ex_main' directory using the cd command. . Run the command "make clean all" to compile the code for the first time. After this, if you make any code edits, you just need to run the command "make" to compile any changes. Once compilation has finished, run "./debug/apollo_main". This will print out a list of the tests and use cases available for each test. Your clocking sources will need to be specified. The device clock will need to be selected as either external clocking or internal clocking from the ADF4382. The FPGA clock will need to be specified as from the FMC connector, or external to the ADS10. For further details on clocking options for the Apollo EVB hardware, please reference the EVB QuickStart guide. The print out details correct syntax to specify your desired options when running the example command. For example, to run the tx_nco test with uc id00_uc06_T and internal clocking, you would enter the command "./debug/apollo_main tx_nco id00_uc06_T -devclk=adf4382 -fpgaclk=fmc"..

The prototypes for all available examples are included in the header file apollo_examples.h. If the user creates their own example code to run out of the main folder, the associated prototype will need to be added to this header file in the same format as the rest of the prototypes for proper compilation. All available profiles are included at the top of main.c, referring to the profiles located in the examples/profiles/include folder. If the user adds their own custom profile, it will need to be included at the top of main in the same format, with the custom header file added into profiles/include. Additionally, any custom example will need to be added to the name_funcs struct and any new profile added to the name_ucs struct inside of main.c, matching formatting.

Some of our example code utilizes AD9084s CDDC BMEM features. The CDDC BMEM is a 16KB SRAM with capture, delay, and arbitrary waveform generator (AWG) functionality. In our examples, we often use it for AWG to send down the receive datapath, bypassing the ADC input. This is useful for receive datapath and JTx link tests.

SOFTWARE OVERVIEW

Table 4. Example API Functions in the ads10_apollo_ex_main Directory

Example Name	Function Description	Apollo MxFE Functions Demonstrated
bsync_tof	The example demonstrates configuration of ADF4030 as an in-system SYSREF source and how to utilize its bi-directional pads for aligning SYSREF going to multiple devices.	TDC and Firmware-based SYSREF Alignment
dp_load	This example prints the status of the jesd links and datapath configurations after bring-up of a chosen profile	Inspection functions
fmcb_aux	The example demonstrates HAL initialization for FMCB Auxiliary Devices (ADF4382, HMC7044, ADL6331 and ADL6332) and SPI read/write checks. FMCB boards require level shifters to be enabled for SPI reads on SPI2 so the appropriate level shifter must be configured with device HAL.	Setup of Software Application Architecture for EVB specific clocking and power solution.
fullchip	This example runs a simultaneous transmit and capture. To use BMEM AWG as the data source, set “use_adcs” to “false”. The example will set up the device and perform clock conditioning as well as ADC calibration. It takes captures for all converters while transmitting and saves them to ‘\home\analog\Apollo’ with the prefix "fullchip". The captures can be moved locally and viewed with Visual Analog	Simultaneous Transmit and Receive
fullchip_8t8r	Simultaneous transmit and receive for the AD9088 FMC board.	AD9088 Transmit and Receive
fullchip_fsrc_dr	Transmits and captures a tone using FSRC parameters loaded from the profile id00_uc06_F. It will then loop though other FSRC configurations declared in the file using the dynamic reconfiguration feature triggered through SPI. FDUC gain is also exercised. This test will generate I/Q files that can be copied locally from ‘\home\analog\Apollo’.	Simultaneous Transmit and Receive, SPI Trigger Based Dynamic Reconfiguration, Fractional Sample Rate Converter (FSRC) for Transmit and Receive (AD9084 Only)
fullchip_fsrc_sc1_ext_trig	Demonstrates the Apollo MxFE configured in Fullchip Dynamic Reconfiguration w/ SC1 and external trigger. See the docstring header at the top of the fullchip_fsrc_sc1_ext_trig.c file for more details.	Using Trigger for System Synchronization
fullchip_hop	Demonstrates configuration of various hopping modes for Apollo MxFE's PFILT, CFIR, CNCO and FNCO. See the docstring header at the top of the fullchip_hop.c file for more details	Simultaneous Transmit and Receive with PFILT Mode Hopping and PFILT/CFIR Hopping
fullchip_mcs_sc1_dl	Demonstrates SC1 deterministic latency for Apollo MxFE 4T4R devices using firmware alignment of internal and external SYSREF. External SYSREF to Apollo and the FPGA is provided by ADF4030, and the ADF4382 provides the Apollo device clock. The Apollo Time-to-Digital-Converter (TDC) works in concert with the ADF4382 to fine-align the internal-external SYSREFs. Phase alignment on DAC output and ADC captures relative to SYSREF can run-to-run can be validated. See the docstring header at the top of fullchip_mcs_sc1_dl.c for more details	TDC and Firmware-based SYSREF Alignment
fullchip_pfilt	Demonstrates various PFILT configurations for Rx and Tx paths. See the docstring header at the top of the fullchip_pfilt.c file for more details.	Simultaneous Transmit and Receive with PFILT Mode Hopping and PFILT/CFIR Hopping
fullchip_sc1_dl	Demonstrates SC1 deterministic latency for Apollo MxFE 4T4R devices using hardware alignment of internal and external SYSREF. See the docstring header at the top of the fullchip_sc1_dl.c file for more details.	Simultaneous Transmit and Receive in JESD204 subclass 1 mode
fullchip_sparse_cfir	Demonstrates configuration of CFIR spare mode for Rx and Tx paths. See the docstring header at the top of the fullchip_sparse_cfir.c file for more details.	Simultaneous Transmit and Receive using CFIR sparse mode (Sparse CFIR Mode)
fullchip_sr_dr	This example demonstrates dynamic reconfiguration using SPI triggering. It cycles through sets of CDDC/CDUC and FDDC/FDUC pairings, reconfiguring the datapath rates accordingly, while the jesd bit rate remains constant. The example supports both HW and SW FPGA image transport layers. The SW image implementation briefly drops the link during reconfiguration, while the HW implementation maintains the link. See the docstring header at the top of the fullchip_sr_dr.c file for more details.	CDDC/FDDC Dynamic Reconfiguration
gpio_toggle	This example sets Apollo MxFE GPIOs to communicate with the FPGA GPIOs. First, it makes Apollo MxFE GPIOs inputs and the FPGA GPIOs outputs. It will then toggle the output GPIOs in 50ms intervals. It then flips the inputs and outputs and repeats the toggling. It then sets the inputs of GPIOs 0-30 high and then flips the inputs and outputs	GPIO setup, toggling, reading, and writing (GPIOx Pin Operation)
jesd_loopback	This example sets up loopback 3 and has the FPGA's JTx send a ramp pattern to the Apollo MxFE JRx. Data is routed back to the FPGA's JRx via the Apollo MxFE JTx. Expect the A0/B0 capture files to contain an interleaved ramp incrementing by one and the A1/B1 decrementing by 1. See the docstring header at top of the jesd_loopback.c file for more details.	JESD204 (JRx to JTx) Loopback (Loopback3)

SOFTWARE OVERVIEW

Table 4. Example API Functions in the ads10_apollo_ex_main Directory (Continued)

Example Name	Function Description	Apollo MxFE Functions Demonstrated
jrx_eye_sweep	This example sets the FPGA's JESD204B/C transmitter PHY to transmit a PRBS data pattern as well as configuring the Apollo MxFE's JESD204B/C receiver to check the same PRBS test pattern. Device firmware runs a complete 2-dimensional eye sweep and stores the resulting data in a plottable array in the device memory. Vertical JRx Eye Sweeps return an array of positive and negative values for each SPO step. User can perform multiple back-to-back sweeps and the number of sweeps can be set using variable `total_sweeps`. Default is 32.At the end of each vertical sweep, for a given lane, all SPO values are saved to a file with the name and location of each file printed. See the docstring header at the top of the jrx_eye_sweep.c file for more details.	JESD204B/C Receiver PHY PRBS Testing and JESD204B/C Receiver PHY Eye Scan
lb0_bmem_delay_hop	Demonstrates BMEM delay hopping with ADC and DAC on both sides (A & B) are looped back in LB0 and side B data path has BMEM delay added. The phase shift between side A and B output can then be observed. See the docstring header at the top of the lb0_bmem_delay_hop.c file for more details.	Receiver to Transmitter Analog Loopback (Loopback0) with delay using BMEM
loopback0	Sets up loopback0 and iterates through different ADC-DAC connections. When presenting a tone to the ADC the looped-back DAC will play the tone out.	Receiver to Transmitter Analog Loopback (Loopback0)
loopback1_2	Demonstrates configuration for Loopback 1 on side A and Loopback 2 on side B. Loopback 1 has 1:1 mapping from CDDC -> CDUC and Loopback 2 has 1:1 mapping from FDDC -> FDUC. See the docstring header at the top of the loopback1_2.c file for more details.	Receive CDDC to Transmit CDUC (Loopback1) and Receive FDDC to Transmit FDUC (Loopback2)
mcs_cal	The example demonstrates the Apollo MxFE's MCS calibration procedure. The procedure consists of running MCS init calibration for coarse alignment of the Apollo MxFE's internal SYSREF against the provided external SYSREF. Following initial calibration (init cal), one can continue the adjustment by performing MCS tracking calibration to further fine tune the time difference between the internal and external SYSREF interval. The examples also shows how to configure ADF4030 as SYSREF clock source for an in-system clocking setup with a limited support for time being. See the docstring header at the top of the mcs_cal.c file for more details.	Device Synchronization with ADF4030 BSYNC (bi-directional SYSREF) (ADF4030 - Bi-directional SYSREF to Apollo MxFE)
power_readback	This example performs voltage and current measurements for the Apollo rails using on-board power monitoring chips, printing the results, which are specific to the evaluation board power network.	Power readback
prbs	This file runs a PHY PRBS (pseudorandom bit sequence) test using a PRBS-7 pattern and test with the ADS10 FPGA (JRx PHY) platform. It also executes an Apollo MxFE (JRx) PRBS-7 test with ADS10 FPGA (JTx) platform	PRBS testing for JESD204 (JESD204B/C Transmitter PHY PRBS Testing, JESD204B/C Receiver PHY PRBS Testing)
rx_adc_bmem	This example demonstrates capturing ADC data, Rx Data path, and JESD204 to FPGA capture memory. The test will run clock conditioning and the initial ADC foreground calibration. It also displays the TMU temperature sensor data. The coarse and fine NCOs are used to shift the tone. Data is driven out of the JESD204 JTx to the ADS10 FPGA. It will read the FPGA capture memory and create capture files of raw I/Q interleaved data	ADC Calibrations, Capture <add links (ADC Calibration and Specifying Nyquist Zone) >
rx_adc_cc	This example demonstrates on demand clock conditioning. Clock conditioning may need to be run after a temperature shift. No ADC input can be present during this example.	ADC Calibration and Specifying Nyquist Zone, ADC Capture, Sampling Clock and Distribution Options
rx_adc_deep	Takes a “deep” capture using the ADCs. The capture size can be set by the user in the example.	ADC Capture, Changing Capture Size
rx_adc_fd	This example demonstrates enabling fast detect on ADC inputs. The example enters a loop where when prompted it will check for fast detect and if there is a tone of approx 3dBm present would report 1 on corresponding ADC and if no tone is present it reports 0.	Rx only operation with Fast Detect (FD)
rx_adc_ms	This example demonstrates ADC mode switching and ways to configure it. ADC mode switching can be executed via firmware (slowest), regmap (slow) and GPIO (fast). The examples shows how to do it in all three ways and does ADC data capture after each mode switch.	Sampling Mode Switch
rx_adc_mux2	This example demonstrates that relative phase between ADC channels is maintained when alternating CDDC to FDDC muxing using the Receive Mux2. See the docstring header at the top of the rx_adc_mux2.c file for more details.	Receive path routing flexibility with Receive Mux2 in receive-only configuration
rx_adc_nz	Captures a tone from the ADC or using BMEM in the first and second nyquist zone. This test will generate I/Q files that can be copied locally from ‘\home\analog\Apollo’.	ADC Nyquist Zone Switching (ADC Calibration and Specifying Nyquist Zone),

SOFTWARE OVERVIEW

Table 4. Example API Functions in the ads10_apollo_ex_main Directory (Continued)

Example Name	Function Description	Apollo MxFE Functions Demonstrated
		BMEM Capture Setup, BMEM Capture (Buffer Memory (BMEM))
rx_adc_pave	The example takes two ADC inputs and averages them together with a noise improvement of up to 3dB. The PFILT ADC averaging block is used to average two input signals with an option for adding or subtracting. The filter may be bypassed or engaged. The ADC inputs are routed through the data path and out JTx to the FPGA where capture data is available. See the docstring header at the top of the rx_adc_pave.c file for more details.	ADC Averaging for Receiver PFILT
rx_adc_pfilt	Captures a tone through the ADCs while applying the PFILT functionality to loop through a high-pass filter, a low-pass filter, no filter, and a zero filter. This test will generate I/Q files that can be copied locally from '\home\analog\Apollo'.	PFILT Coeff Loading, ADC Capture (Transmit/ Receive Programmable Filter (PFILT))
rx_bmem_cfir	Exercises the CFIR block with 2 CFIR profiles - a high pass and a low pass filter. This is done using BMEM to generate a tone. This test will generate I/Q files that can be copied locally from '\home\analog\Apollo'.	CFIR Coeff Setup, BMEM AWG Mode Setup, JESD204 Capture (Buffer Memory (BMEM), Programmable Complex FIR Filter (CFIR) - Receive and Transmit)>
rx_bmem_ddc	Captures tone generated in BMEM while exercising the CDDC and FDDC gain blocks. This test will generate I/Q files that can be copied locally from '\home\analog\Apollo'.	DDC Gain block setting, BMEM Setup, JESD204 Capture<add links (Bypassable 6 dB Gain Stage)>
rx_bmem_delay	The example demonstrates configuration of BMEM sample delays. The example uses a profile that configures ADC0 input into 2 parallel data paths, both configured identically. With same input to both the Rx data path, the capture should show difference only if the data path is configured differently. Here we add sample delay to A0 data path and not on B0 data path and do data capture.	Receiver only configuration with added BMEM Sample Delay (Buffer Memory (BMEM))
rx_bmem_pfilt	Generates a tone in BMEM while applying the PFILT functionality to loop through a high-pass filter, a low-pass filter, no filter, and a zero filter. This test will generate I/Q files that can be copied locally from '\home\analog\Apollo'.	PFILT Coeff Loading, BMEM AWG Mode Setup, JESD204 Capture (Transmit/Receive Programmable Filter (PFILT), Buffer Memory (BMEM))
rx_jesd	This example demonstrates capturing ADC data, Rx Data path, and JESD204 to FPGA capture memory. Data is driven out of the JESD204 JTx to the ADS10 FPGA. It will read the FPGA capture memory and create capture files of raw I/Q interleaved data. The example can be used to do BMEM AWG capture as well which would write a tone into BMEM and would be captured via JESD. To run example in BMEM mode pass CLI arg `-b`	Receiver only configuration
rx_sniffer	FFT Sniffer example. Loops through different modes, while exposing the fft sniffer knobs in the example.	Spectrum Sniffer
tx_jesd	This file configures the Apollo MxFE JESD204 JRx and FPGA JTx. It sends complex data samples over the JESD204 link. A complex tone is created with I data on virtual converters M0 and M2, Q data on M1 and M3. The CNCO and FNCO shift the tone in frequency	Data Transmit, NCO Setting
tx_jesd_cfir	Transmits a tone generated from the FPGA while exercising the CFIR block with low pass and high pass filter CFIR profiles.	CFIR Coeff Setup, TX Transmit (Programmable Complex FIR Filter (CFIR) - Receive and Transmit)
tx_jesd_file	The example loads I and Q vector data from the file. The CLI argument `-a` (auto) will create a simple I and Q vector and save it to a file before transmitting. The CLI arg `-f`expects user to provide two paths for I and Q data files respectively. The data file should be on MicroZed and accessible to the example.	
tx_nco	Transmits a tone in NCO test mode	NCO Mode Setting, FTW Setting (Fine Digital Up-converter (FDUC) and Fine NCO (FNCO), Coarse DUC (CDUC) and Numerically-Controlled Oscillator (CNCO)
tx_nco_ffh	This example file outputs NCO tones from the DAC. It is recommended to run this test first as it does not require JESD204 links. It demonstrates changing both CNCO and FNCO frequencies programmatically	NCO Mode Setting, FTW Setting (Fine Digital Up-converter Modulator (FDUC) and Fine NCO (FNCO), Coarse DUC (CDUC) and Numerically-Controlled Oscillator (CNCO)
tx_nco_mod	The example demonstrate use of Modulus NCOs	NCO Modes

Table 4. Example API Functions in the ads10_apollo_ex_main Directory (Continued)

Example Name	Function Description	Apollo MxFE Functions Demonstrated
tx_nco_pfilt	Transmits a tone in NCO test mode and uses the PFILT functionality to loop through a high-pass filter, a low-pass filter, no filter, and a zero filter.	NCO Mode Setting, FTW Setting, PFILT Coeff Loading (Transmit/Receive Programmable Filter (PFILT))

CHIP VARIANT INFORMATION

The AD9084/8 each come in several variants as listed in Table 1. The api function adi_apollo_device_board_variant_and_rev_get() can be used to readback the variant information on a specific chip, returning a structure detailing the device ID, silicon grade, whether the chip is RX-only, TX-only, or MX, the software trim (1, 3, or 5, which enable increasing numbers of chip features as detailed in Table 2), whether it is a 9084 or a 9088, and whether it is RX differential or single ended. adi_apollo_device_board_variant_and_rev_get() is composed of individual apis to retrieve each member of the structure separately, if the user does not wish to return the full variant struct.

CONTROL INTERFACES

The Apollo MxFE can be accessed using SPI and HSCI. The maximum SPI rate is 50MHz. The HSCI can operate up to 1.6GHz with an 800MHz clock using double data rate.

SERIAL PERIPHERAL INTERFACE (SPI)

SPI Data Transfer Protocol

The Apollo MxFE is configured using a flexible, synchronous serial communications port to allow a 3-wire or 4-wire simplified interface with industry standard microcontrollers and microprocessors. An active low input signal at the CSB pin starts and gates a communication cycle used to perform a write or read operation. This input signal must remain low throughout the communication cycle and must return high before returning low again to start a new communication cycle. The SCLK serial clock pin synchronizes data to and from the device and runs the internal state machines with all data input appearing on the bidirectional SDIO pin registered on the rising edge of SCLK. All data is driven out of the SDIO pin (or SDO pin for a 4-wire interface) occurring on the falling edge of SCLK during a read operation with the pin going into a high impedance state when the CSB pin returns high. Timing specifications associated with the SPI port can be found in the device datasheet.

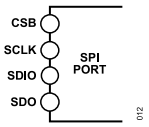

Figure 14. SPI Interface Pins

The SPI port is compatible with most synchronous transfer formats to allow a simplified write and read operation to all registers used to configure the device. Before configuring the device, the SPI interface needs to be configured. An API, ***adi_apollo_device_spi_configure(adi_apollo_device_t *device, adi_apollo_device_spi_settings_t *spi_config, uint8_t spi_num),*** is used to configure the SPI interface of the Apollo MxFE. The spi_num argument selects between SPI0 (default) and SPI1. The members of the spi_config_t struct are described in Table 5.

Table 5. SPI Configuration Parameters

Member	Description	Enumeration
sdo	3-wire or 4-wire configuration	ADI_APOLLO_DEVICE_SDO (4-wire) ADI_APOLLO_DEVICE_SDIO (3-wire)
msb	MSB/LSB bit order configuration	ADI_APOLLO_SPI_MSB_LAST (LSB First) ADI_APOLLO_SPI_MSB_FIRST (MSB First)
addr_inc	Address increment or decrement	ADI_APOLLO_DEVICE_SPI_ADDR_DEC_AUTO (Auto decremented) ADI_APOLLO_DEVICE_SPI_ADDR_INC_AUTO (Auto incremented)

The default communication cycle with MSB first consists of two phases, as shown in Figure 15. The first phase is the instruction cycle that consists of 16 SCLK cycles that define the operation type and the starting register address. The first bit, 15th bit, of the 16-bit instruction word that appears at the SDIO input defines whether the upcoming data transfer is a read or write operation (R/) The 14th bit indicates whether the SPI transaction is a Direct Access SPI transaction or Paging Access SPI transaction. The types of accesses are discussed in detail in the SPI Access Modes. The remaining 14 bits (MSB to LSB format) specify the starting register address for the read or write data transfer operation. A multibyte transfer format with an incrementing address is supported and depending on the **addr_inc** settings, the address is either incremented or decremented by 1 for every eight bits of data.

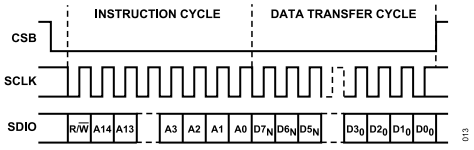

Figure 15. Serial Register Interface Timing, MSB First,

The second phase of the communication cycle consists of eight SCLK cycles and is the actual transfer of a data byte between the device and the system controller. To transfer more than one byte (or N + 1 bytes) during the transfer cycle, 8×N SCLK additional cycles are required to ensure that the last byte is transferred. Each time one of the eight clock cycles complete, the internal address index updates such that the next eight data bits transfer to the next register address.

CONTROL INTERFACES

The SPI port can also support LSB first data format, as shown in Figure 16, when the LSBFIRST bit is set. In this case, the instruction and data bits must be written from LSB to MSB with the R/$\overline{W}$ bit following the MSB (or A14) of the address word.

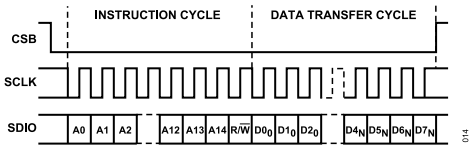

Figure 16. Serial Register Interface Timing, LSB First,

Additional details on the Analog Devices SPI interface standard can be found here.

SPI Transaction Modes

Single Transaction

In single instruction mode, the chip select line is pulled low followed by a 16-bit address during the address phase of the SPI transaction. This is followed by an 8-bit data load for the intended address. The CS is then pulled high indicating the end of the SPI instruction. See Figure 17

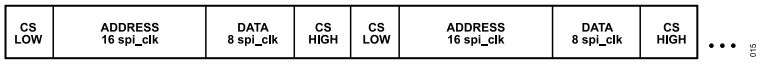

Figure 17. Single SPI Transaction

Streaming Transaction

In streaming transaction mode, the chip select line is pulled low followed by a 16-bit address during the address phase of the SPI transaction. This is followed by an 8-bit data load for the intended register. The CS remains low and more 8-bit data loads can be streamed. If the **addr_inc** is set to auto-increment, the following 8-bit data loads are used for registers that follow the register set in the address phase incrementing the register address with each 8-bit data load. If the **addr_inc** is set to auto-decrement, the following 8-bit data loads are used for registers that proceed the register set in the address phase decrementing the register address with each 8-bit data load. See Figure 18

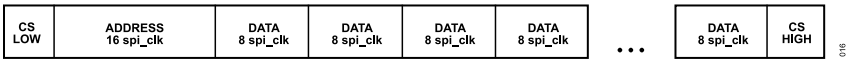

Figure 18. Streaming SPI Transaction

SPI Access Modes

There are two regions in the register map when it comes to SPI access: A direct accessed region and an indirect or paging accessed region.

Direct Access

In this mode, the 14th bit in the address phase of the SPI transaction is set to 0 and the remaining 14 bits are used for the address of the register to be accessed. Direct access SPI mode is used to access Core registers that can be addressed by 14 bits addresses.

Paging Access

In this mode, the SPI interface can access the 32-bit addressed region of the register map. To enable this mode, bit 14th needs to be set to 1. The paging access works by setting the 32-bit base register by writing to 4 8-bit registers in the direct accessed region of the register map, core registers. After setting the 32-bit base register, the following 14 bits in the address phase in the following SPI transaction are an offset that is added to the base register value. The resulting address from the sum of the offset and the base register is the address to which the data in the data phase of the SPI transaction is written. Paging mode only supports writing SPI operations. See Figure 19.

CONTROL INTERFACES

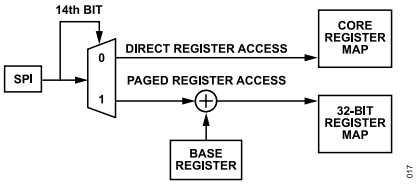

Figure 19. Access to Register Maps

Indirect Access

In this mode, the SPI can access all registers in the register map. It works by setting 32-bit register that represents the address in the register map to be written to or read from. The 32-bit register address is broken down to four 8-bit registers that can be accessed using the direct mode. The data to be written or read from this register is available at another 32-bit register that is broken down to 4 8-bit registers that can be accessed using the direct mode access. This mode supports reading from the entire register map.

SPI Read Modify Write

The SPI read-modify-write is a feature which allows SPI to write specific bitfields “only” of registers while keeping the other bitfields same as before. It does so by reading the register whose bits are needed to be written, masking the bits that need to be kept as before and then overwriting the data in the other bitfields.

HIGH SPEED SERIAL CONTROL INTERFACE (HSCI)

As an alternative to SPI, the device has a High-Speed Serial Control Interface (HSCI). The HSCI provides a faster way to interface with the AD9084 and AD9088 at up to 1.6 GHz. It is an LVDS based, point to point interface that uses source synchronous clocking to achieve high throughput. The HSCI module in the device acts as a follower and requires a corresponding HSCI controller implemented in an FPGA or ASIC.

HSCI Protocol

The information sent across the HSCI link is grouped into frames. Each frame is broken into one or more sub-frames. Each sub-frame consists of a byte of information, plus an additional parity bit (P), and an additional continuation/stop bit (C) for a total of 10 bits/sub-frame. The byte sent across the serial link is always MSB first as illustrated in the 10-bit sub-frame structure of Figure 20. A full frame example is illustrated in Figure 21.

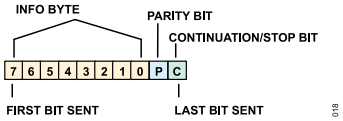

Figure 20. HSCI 10-bit sub-frame structure

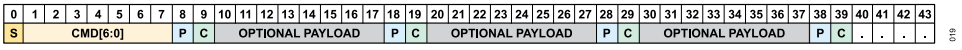

Figure 21. Example HSCI-frame structure

Table 6. HSCI 10-bit sub-frame fields

Bit/bit-field	Description
Info byte	The information byte can be an instruction, an address, data, or information.
Parity bit	The parity *(P)* bit is the parity of the previous 8 bits (info byte) plus the parity of the subsequent continuation/stop bit. If you add the total number of ones together, the parity bit is a ‘1’ if the sum is odd, or a ‘0’ of the sum is even
Continuation/Stop bit	The continuation/stop *(C)* bit is used to signify that access is not complete and that another 10-bit transfer is being sent. The bit is a ‘1’, expect another 10-bit transfer, or, if the bit is a ‘0’, that is the end of the access.

CMD Sub-frame

The first sub-frame of any access is the Command (CMD) sub-frame. As described in Table 7, the first bit is always a ‘1’. Since a ‘0’ is the IDLE state for the HSCI, the initial ‘1’ notifies the port that an access has started. After the start bit, there’s a 7-bit instruction bit-field. An HSCI CMD sub-frame timing diagram is illustrated in Figure 22.

CONTROL INTERFACES

Table 7. HSCI CMD sub-frame fields

Bit/bit-field	Description
Start bit	Always a '1' to notify the follower that a new access is beginning
Instruction [6:0]	Can be a write instruction or read instruction, etc.

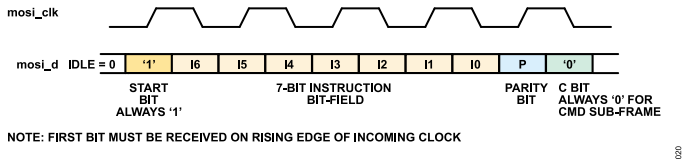

Figure 22. HSCI CMD sub-frame timing diagram

Instruction Decodes

Table 8 contains the list of instructions that can be used while doing a read, write or read modify write transaction using HSCI. This includes instructions expected to come from both the controller and the follower. The term "tsize" refers to the transfer size of the data being sent to the HSCI receiver. These transactions are defined in further detail in the HSCI Transaction Modes section.

Table 8. HSCI Instruction Decoding

Instruction [6:0]	Sender	tsize	Description
{0,1,1,0,tsize[2:0]}	Controller		Write Operation:
		000	byte transfer
		001	halfword transfer (16 bits)
		010	word transfer (32 bits)
{1,0,0,0,tsize[2:0]}	Controller	"	Read Request Operation: tsize – same as above
{1,0,0,1, tsize[2:0]}	Controller	"	Read Modify Write: tsize – same as above
{1,0,1,0,tsize[2:0]}	Follower	"	Read Acknowledge Operation: tsize – same as above
{1,1,0,0,x,x,x}	Follower	N/A	Error Encountered: ALWAYS followed by a byte value

HSCI Link-up Operation

For the HSCI interface to function properly, the clock timing must be adjusted. The HSCI follower can adjust the incoming clock with respect to the incoming data. By adjusting the clock, the sampling of the data can be optimized to ensure a robust link.

Similarly, on the transmit side, the follower can adjust the outgoing clock with respect to the outgoing data. This allows the incoming data at the controller to be sampled at the optimal time interval.

There are 2 methods for establishing the HSCI link.

- The manual method which requires SPI interaction and the user to determine which clock delay adjustment is the best to use.
- The automatic method which only requires an initial SPI access to enable the HSCI and set the auto_linkup bit and the txclk_adj_override bit.

For auto link-up, the controller must generate a continuous stream of link-up frames.

A link-up consists of a link-up CMD sub-frame, followed by a six PN sequence sub-frame, followed by an IDLE sub-frame (all 0's). The PN sequence sub-frames use the same polynomial as the JESD204 scrambler, $1 + x^{14} + x^{15}$.

The HSCI follower uses this sequence to adjust the incoming clock with respect to the incoming data. By adjusting the clock, the edge at which data is sampled can be optimized to ensure a robust link. The follower identifies a good value for receiver clock adjustment using an auto-link up table. This table is a 16 element, 1-bit array with each element corresponding to the 16 possible adjustment values. If a lock is achieved for a given receiver clock adjustment, the respective auto-link up table element is set to '1', if not, the element is set to '0'. At the end of traversing through all 16 receiver clock adjustment values, the auto-link up table contains the information needed to make a decision. The algorithm looks for a set of seven, five or three consecutive '1's' in the auto-link up table (in that order of preference) and picks a receiver clock adjustment value at the center of the consecutive set of ones. If there are no consecutive '1's' of length three, five or seven, in the follower's auto-link up table, the controller must invert the clock and re-run the procedure again until such a series is found for the auto-link up process to continue.

CONTROL INTERFACES

Similarly, on the transmit side, the follower needs to adjust the outgoing clock with respect to the outgoing data. This is enabled by the follower looping back the auto link-up frames from the leader. To enable this process a similar auto-link up table is generated on the controller. Once a good *transmitter clock adjustment* value is obtained, the controller sends the actual *HSCI transmitter clock adjustment* value on every auto link-up CMD frame during the loopback portion of the automatic link procedure. The follower receives and decodes the auto link-up CMD sub-frame and loads the HSCI transmitter clock adjustment and *HSCI transmitter clock invert* registers with the values it has received. Furthermore, it replaces the outgoing auto link-up CMD sub-frame to the controller with the current *HSCI transmitter clock adjustment* value and *HSCI transmitter clock invert* value.

The controller then verifies its *HSCI transmitter clock adjustment* values match the received values from follower before attempting to establish the link.

Figure 23 illustrates this HSCI CMD link-up sub-frame for the Apollo MxFE device.

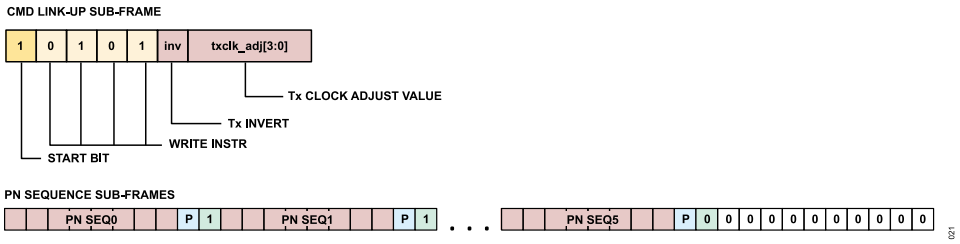

Figure 23. HSCI CMD link-up example

Instruction Sub-frame of LINKUP Frame

The instruction bits of the CMD link-up sub-frame are illustrated in Figure 24 can be used to control clock adjustments.

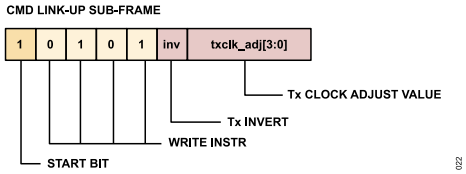

Figure 24. HSCI CMD link-up sub-frame

NOTE: *this CMD sub-frame breaks protocol, which requires a Parity bit followed by a Continuation/Stop (C/S) bit. However, for a CMD sub-frame, the C/S bit is always '0', and during auto link-up, the parity checker is disabled.*

HSCI Transaction Modes

The controller needs to be able to send the following commands - a write instruction, a read request instruction, and a read-modify-write instruction. In addition, the controller needs to be able to drive the link-up during link activation.

HSCI Write Operation

The HSCI controller needs to perform the following steps when looking to perform a write to the register space in the AD9084 and AD9088:

- The controller performs a write instruction by first sending a CMD sub-frame with the write instruction and the transfer size, tsize[2:0]. The transfer size tells the follower how to format the write data.
- The data can be formatted as bytes, halfwords (16-bits), or words (32-bits). After the CMD sub-frame, the controller sends address sub-frames containing the starting address.
- The controller can send 1, 2, 3, or 4 address sub-frames. The address sub-frames start with the lower byte and work their way up to 16 lower bits, lower 24 bits, or all 32 bits.
- If the controller only sends 1 address sub-frame, the follower assumes the upper 24 address bits are all 0's. If the controller sends only 2 address sub-frames, then the upper 16-bits are assumed to be 0's, and so on.
- The controller designates the end of the address sub-frames by setting the continuation bit to '0'. Next the controller starts sending write data sub-frames.

CONTROL INTERFACES

- There is no limit to the number of write data sub-frames the controller can send out. The follower will continue to receive the write data sub-frames, writing the data and incrementing the address automatically.

***NOTE**: The total number of write data sub-frame is dependent on the tsize variable. There are no restrictions on the number of sub-frames for byte transfers. However, for halfword transfers, the number of sub-frames must be even (divisible by 2) as this ensures an integer number of halfword. Similarly, for word transfers, the number of write data sub-frames must be divisible by 4.*

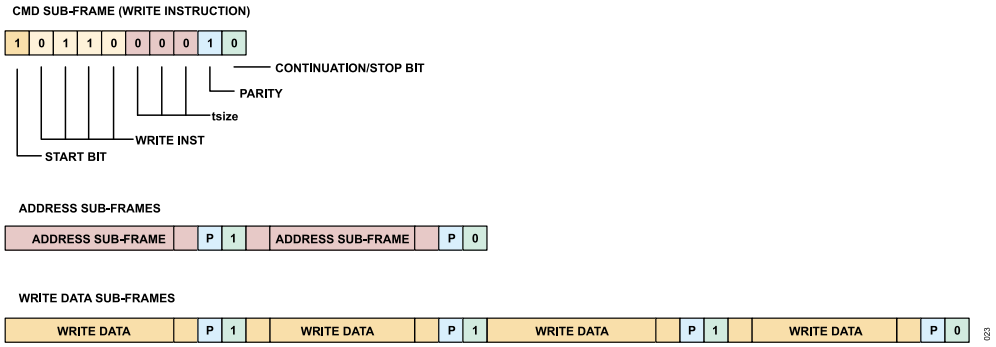

Figure 25. HSCI Write Operation Example

Figure 25 illustrates an example of a write operation. The write operation cannot be paused by the controller. The controller must be able to maintain the data throughput of the interface. If not, the write operation must be terminated, and another write operation initiated when more write data is available.

Finally, back-to-back write frames are allowed without any idle frames in between.

NOTE on the continuation/stop bit:

The continuation/stop bit does not just signify the end of an operation, but also a delineation during that operation.

Figure 26 illustrates this: for a halfword write, the c/s bit for the instruction byte is '0', and then the next sub-frame will be the address portion of the frame. Since it is a half-word address, the least significant byte (LSByte) of the address contains a '1' for the c/s bit because more address information is coming. The most significant byte (MSByte) of the address contains a '0' for the c/s bit, denoting that the next transfer will be data. Finally, the data byte will contain c/s bits that are '1' until the last data bit is sent. On the last data bit, the c/s will be a '0', signifying the end of the write operation.

Figure 26. HSCI Use of the continuation/stop bit as a delineation during an operation

HSCI Read Request Operation

Read operations are not immediate. If the HSCI controller wants to read a register or memory location, it will send out a read request. Then, HSCI follower will respond with a subsequent read acknowledge instruction along with the requested data.

- The read request starts like all other accesses with a CMD sub-frame, {1,0,0,0,tsize[2:0]}. That is followed by an address sub-frame or a series of address sub-frames.
- Once the address is sent, the next sub-frame corresponds to the number of bytes to read. The number read byte sub-frames is limited to four. This allows a read request range of 1 byte to 232 bytes.
- The number of read bytes value will be byte_num – 1. For example, if 10 read bytes are requested, then the byte_num value will be 9. Also, if only 1 byte is requested, then the byte_num value will be 0. Also, for halfword read requests, the number of bytes value must adhere to the following constraint, byte_num + 1 is even. Similarly, for word read request, byte_num + 1 must be divisible by 4.
- Note, the controller cannot send another instruction until the follower has completed the transfer of the read values back to the controller. However, the controller can send back-to-back write commands followed by a read request.

CONTROL INTERFACES

HSCI Read Acknowledge Operation (from Follower)

After the controller has performed a read request operation, the follower will respond with a read acknowledge operation. It will fetch the requested read data and drive it out to the controller. The first CMD sub-frame contains the read acknowledge instruction, {1,0,1,0,tsize[2:0]}. The next sub-frame(s) contains the byte index. The number of byte index subframes corresponds to the number of non-zero byte index bytes.

It is not necessary for the follower to send the entire read data back in a single frame. The follower may be unable to retrieve the requested read data at a rate that will keep pace with the HSCI data rate. Therefore, the follower can break-up its response into a series of read acknowledge frames. Figure 27 illustrates an example of a HSCI Read Request and Read Acknowledge.

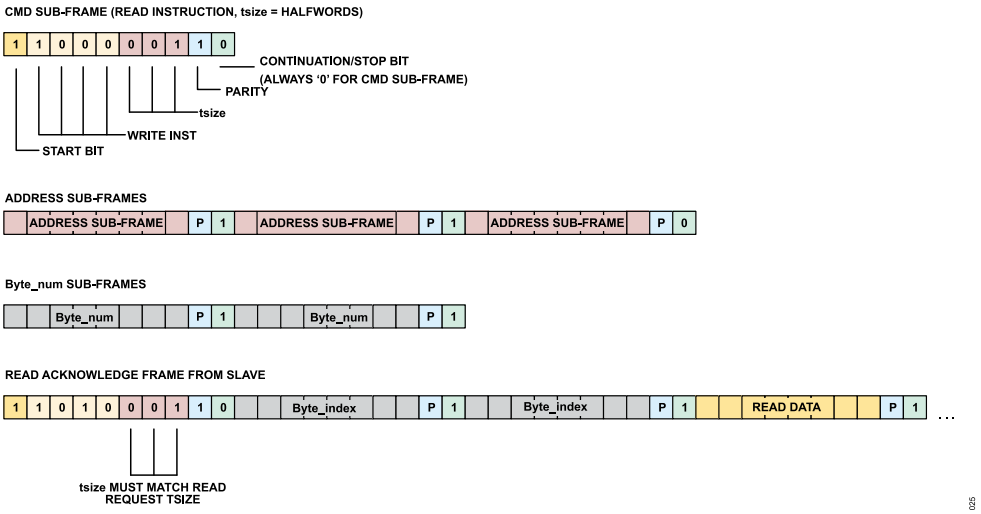

Figure 27. HSCI Read Request and Read Acknowledge Examples

HSCI Read Modify Write Operation

The read-modify-write operation is like a write operation. The controller sends the read-mod-write CMD sub-frame, followed by the address sub-frame(s). Then, the controller sends alternating write mask sub-frames and write data sub-frames.

The number of write masks and write data sub-frames must correspond to the tsize. So, if the tsize is set to bytes, then the controller will send 1 write mask sub-frame followed by 1 write data sub-frame, then repeats. If the tsize is set to words, the controller sends 4 write mask sub-frames followed by 4 write data sub-frames, then repeats.

As is the case with the write operation, there is no limit to the number of write data sub-frames to send as long as the write data can keep up with the throughput. Figure 28 illustrates an example of a HSCI read-modify write.

***NOTE:** Since the mask and write data subframes will always come in pairs, the C/S bit is ALWAYS set for the mask sub-frames*

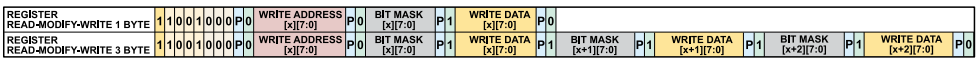

Figure 28. HSCI read-modify-write example

HSCI Error Handling

On occasions when the follower may encounter an error, it can send a "Error Encountered" frame back to the controller. The frame consists of an Error Encounter instruction followed by an error code. Table 9 includes a list of error codes.

Table 9. HSCI follower error reporting decoder table

Error bit	Error Description
0	Unknown Instruction received
1	Bad parity detected
2	Address size error (more than 4 address sub-frames received)
3	Byte_num size error (more than 4 byte_num sub-frames received)

CONTROL INTERFACES

Table 9. HSCI follower error reporting decoder table (Continued)

Error bit	Error Description
4	Write FIFO full flag
5	Read FIFO full flag

***NOTE**: These also can be accessed in the HSCI follower register map. In addition, any of these error flag can be disabled.*

The controller can also detect errors coming in from the follower. During normal operation, the controller should receive either Read Acknowledge frames or Error Encountered frames. If it isn't either of the two it would indicate either an unknown instruction or a parity error.

HSCI Hardware Implementation

The HSCI controller interface to the follower consists of 8 pins configured as 4 LVDS pairs:

- 2 pairs coming into the AD9084 or AD9088 from the controller- MOSI_CLK± & MOSI_D±
- 2 pairs going back to the controller from the AD9084 or AD9088- MISO_CLK± & MISO_D±.

The controller provides the clock that drives the interface, MOSI_CLK. The incoming clock to the controller, MISO_CLK is only a delayed version of the MOSI_CLK. The controller initiates almost all transfers across the interface while the follower can only react to an instruction from the controller.

Figure 29 shows a top-level block diagram representation of the HSCI module on AD9084 and AD9088 respectively.

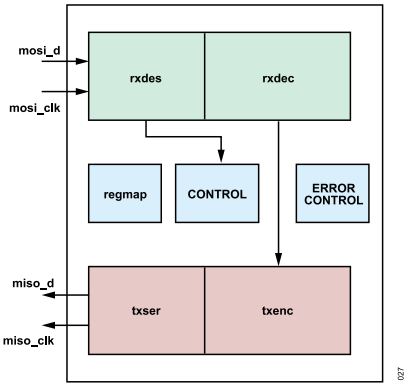

Figure 29. HSCI top level block diagram

HSCI Receive Data Path

The incoming data is sampled on the rising edge of clock. The receiver deserializer (rxdes) detects an incoming frame, deserializes the sub-frames and sends them to the receiver decoder (rxdec) block. The rxdec block decodes the incoming sub-frames from the rxdes. First it checks for an unknown instruction and then for a parity error. If either is detected, it will drive the error flag to the error control block.

For a write operation, the decoder captures the transfer size and then the starting address. Then, as the data streams in, the decoder will configure the data into bytes, halfwords or words depending on the transfer size. When a data value is ready, the decoder sends it out the req/ack bus accompanied by the destination address and transfer size.

For a read operation, the decoder captures the transfer size, the address, and the number of bytes. It then sends that information to the encoder, transmitter encoder (txenc).

For read-modify-write, the decoder captures the address, the data values and the bits to be modified. It then handles the reading of the location, determining what bits will be over-written and performing the write.

HSCI Transmit Data Path

The transmit encoder block will package up commands and data and sends them to the serializer to be sent to the controller.

CONTROL INTERFACES

For a read operation, the encoder would receive the necessary information from the rxdec. Then, it controls req/ack bus interface to fetch the appropriate read data. Once the read data is obtained, the encoder begins sending that data to the serializer with the read acknowledge instruction as the first byte. The encoder keeps track of the number of data values being sent until all the requested data has been transmitted.

If an error has occurred, the error control block will notify the encoder block. Subsequently, the encoder will drive out the error encountered, {1,1,0,0,x,x,x}, followed by the error code.

The serializer block receives byte data from the encoder along with a continuation/stop bit. It serializes the data while generating the parity and inserting the result in the parity bit location. Finally, the block sticks the c/s bit in at the end.

HSCI Error Reporting and Control

The control block handles the automatic link-up process. The error control block receives all the error flags from other block, creates the error code word and instruct the txenc to drive out the error encountered CMD along with the error code back to the controller.

HSCI FPGA/ASIC Implementation Tips

An HSCI controller implementation will need the ability to create the command sub-frame and the subsequent address and data subframes required by the HSCI protocol as well as deconstruct incoming frames from the follower to perform the read, write and read-mod-write operations.

In addition to this it also needs the ability to initiate the manual or the auto-link up procedure based on the system requirement.

For debug purposes, it is also recommended to include the ability to decode error codes that the follower may send to the controller.

Recommended Instruction Set

To make the most of the HSCI interface, the following types of transactions are recommended to be designed into the HSCI controller:

Single Transaction

A single transaction aims to access a single memory of register location on the follower.

- In the case of a write, the controller would take the address of the register to be written and the data to be written to it as inputs, generate the required HSCI frames and send them to the follower.
- In the case of a read modify write, the controller would take the address of the register to be written, the data to be written and a write mask as inputs and generate the required HSCI frames.
- In the case of a read, the controller would take the address of the register as an input and receive the data stored in that register location.

Continuous Transaction

A continuous transaction accesses a continuous register range with successive writes, successive reads, or successive read-modify-writes. While only a single type of operation can be performed at a time, the primary goal of this is to reduce the over-head incurred on doing multiple single transactions.

- In the case of a write, the controller would take the starting address of the register range to be written and all the data to be written to it as inputs, generate the required HSCI frames and send them to the follower one after another as one big write.
- In the case of a read modify write, the controller would take the starting address of the register range to be written, the data to be written to each register and a write mask for each register as inputs and generate the required HSCI frames.
- In the case of a read, the controller would take the starting address of the register range as an input and receive the data stored in each of those locations one at a time.

Dis-contiguous Transaction

A Dis-contiguous transaction aims to access different addresses at distinct locations in the follower register map with a series of transfers - writes, reads, or read-mod-writes. While only a single type of operation can be performed at a time, the primary goal of this is to reduce the over-head incurred on doing multiple single transactions even when the registers aren't all continuous address spaces.

- In the case of a write, the controller would take pairs of the address and its corresponding data as inputs, generate the required HSCI frames and send them to the follower one after another as one big write.

CONTROL INTERFACES

- In the case of a read modify write, the controller would take sets of the address, its corresponding data and write mask as inputs and generate the required HSCI frames.
- In the case of a read, the controller would take a list of addresses as an input and receive the data stored in each of those register locations one at a time.

NOTE: *Each of these transaction types need to maintain the requirements of being an even number (multiple of 2) for half-word transfers and multiples of 4 for word transfers.*

An HSCI Controller implementation (RTL and drivers) is available to customers as part of the AD9084/AD9088 FPGA Reference Designs. Users can also request access to a HSCI FPGA IP Reference Guide that includes additional details on implementing a HSCI Controller in an FPGA.

HSCI API

The HSCI follower module on the device will be supported by API functions that enable the user to configure the HSCI interface and switch between HSCI and SPI for control.

The HSCI interface is configured using the **adi_apollo_device_hsci_configure()** API function on device initialization. The function accepts a configuration object **adi_apollo_device_hsci_settings_t** which contains members for choosing the address incrementation mode, and for en/disabling auto-linkup and loopback mode.

If the adi_apollo_device_hsci_configure() API isn't called explicitly and the user relies on adi_apollo_device_hw_open() to set up the HSCI interface, then adi_apollo_device_t.hal_info.hsci_desc.is_used needs to explicitly be set to 1.

The use of the HSCI interface is enabled by the adi_apollo_hal_active_protocol_set() API function, which uses the adi_apollo_hal_protocol_e enumeration to select between HSCI and SPI interfaces.

SAMPLING CLOCK AND DISTRIBUTION OPTIONS

CLOCK DISTRIBUTION OVERVIEW

The clock distribution architecture supports sample accurate synchronization between ADC and DAC cores on the same IC as well as between multiple IC's. It also supports single or dual clocking modes, as shown in Figure 30, where an external clock and SYSREF pair is driven into the "center" of the IC (single clock mode) or two pairs of these signals are driven into opposite "A" and "B" sides of the device (dual clock mode). Both A and B-side DACs operate in DDR mode where DAC samples are captured on both the rising and falling edges of the input clock to achieve a max sampling rate of 28Gsps. The max ADC sample rate is 20Gsps.

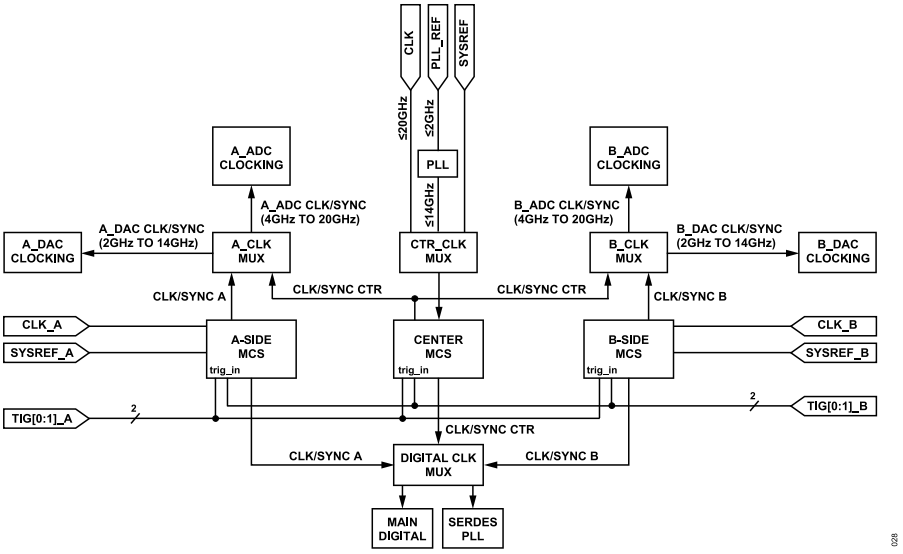

Figure 30. Clock Distribution Block Diagram and Options

SINGLE AND DUAL CLOCK MODES

For single clocking mode only, an optional on-chip PLL clock multiplier can be used to generate the high frequency converter clock. As indicated in Figure 30, the PLL is limited to 14GHz maximum output frequency. Dual clocking mode is intended for the most demanding phase noise requirements where separate uncorrelated in-phase clocks to each side (A and B) of the IC reduces the correlated jitter across converters compared to single clocking mode. This is beneficial for applications such as beam forming. Dual clocking mode also allows for the flexibility to interleave between A and B-side ADC thus providing the highest possible ADC sample rate with the user supplying two clocks that are 180^{o} out-of-phase.

A dedicated multi-chip synchronization (MCS) block is assigned to each clock and SYSREF pair. The MCS block is responsible for generating clocks to the mixed signal, digital, and SERDES PLL circuits to keep these different clock domains synchronized via internal "sync" signals. The MCS block is discussed in detail in the Device Synchronization section. Four single-ended trigger inputs are used to allow sample accurate timing of internal digital functions. Each trigger can be used to provide synchronization events to any or all of the four digital data paths (Receive Data paths A and B, Transmit Data paths A and B). Triggers are retimed to internal SYSREF signals to provide these sync events. In single clocking mode all four data paths are synchronized to the center internal SYSREF. For Dual clocking mode, the A-side digital data paths are synchronized to the A-side internal SYSREF, and the B-side digital data paths are synchronized to the B-side internal SYSREF. The user can select which of the 4 input triggers is used to control each data paths' events within the respective MCS block. See the Trigger Receiver Input section for more details on using the trigger inputs.

Many applications may prefer the simpler single clocking mode with about 1-2 dB degraded phase noise (and jitter) performance at lower frequency offsets. In this mode, an RF differential clock with 8 to 20 GHz range can be provided to the CLK_P and CLK_N balls of the AD9084 (range is 5 to 8 GHz for the AD9088) or a PLL reference clock up to 1 GHz applied to the PLLREF_CLK_P and PLLREF_CLK_N balls. For subclass 1 operation where MCS and/or deterministic latency is desirable, a differential SYSREF signal must also be applied to the SYSREF_P and SYSREF_N pins since this is a required input for subclass1. SYSREF and subclass 1 operation are discussed in detail in the Device Synchronization section. A multiplexer (or Mux) is used to select between the two RF clock generation options (Apollo MxFE internal PLL or Direct clocking) with its output driving the "A" and "B" clock paths as well as the "Center" MCS. The "Center" MCS will generate internal sync signals aligned to the internal SYSREF for the mixed signal converters of "A" and "B" sides. It will also pass a clock and sync signal to digital data path along with a reference clock signal for the SERDES PLL whose frequency will depend on the SERDES configuration.

SAMPLING CLOCK AND DISTRIBUTION OPTIONS

The phase-aligned converter clock and sync signals (per "A" and "B" sides) are then split into DAC and ADC specific clock and sync signals. The block diagrams for the DAC and ADC clocking are illustrated in Figure 31 and Figure 32. The DAC clocks can be divided-by-2 to allow the DAC sample rate to operate at 2x or 1x the ADC sample rate. The ADC's clocking circuit has the option for clock inversion per ADC to enable interleaving. Since the DAC core uses both the positive and negative edge of the clock signal for sample reconstruction, a duty-cycle, skew correction block is included to ensure near 180^{o} phase offset between sampling edges. It is worth noting that since the DAC operates on both edges of its input clock, its sample rate will be 2x the sample rate of the ADC if the DAC divider is set to divide-by-1. For equal sample rates, the DAC divider must be set to divide-by-2.

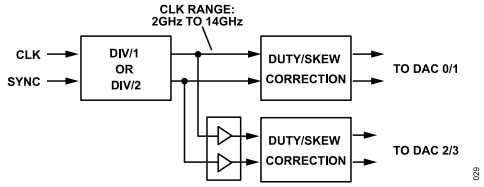

Figure 31. DAC Clocking.

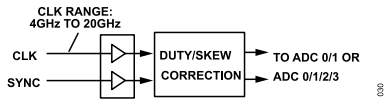

Figure 32. ADC Clocking.

Table 10. Supported ADC/DAC Sampling Rate Ratio

Option	Sampling rate ratio	DAC CLK DIV
1	$F_{ADC} = F_{DAC}$	Div/2
2	$F_{ADC} = ½ * F_{DAC}$	Div/1

In Dual clocking mode, the "A" and "B" sides are driven separately with external clock and SYSREF signals and separate "A" and "B" MCS blocks generate converter and digital Sync signals. Between the "A" and "B" sides of the device, the external "A" and "B" clocks must be the same frequency and the "A" and "B" SYSREF signals must be the same frequency. From an IC perspective, a shorter clock tree typically improves jitter and phase noise performance of the clock signal that drives the ADC and DAC cores. For digital beamforming applications, supporting two separate clock inputs enables improved system level phase noise performance in the frequency offset regions where the input clock sources phase noise remain decorrelated as discussed in System-Level LO Phase Noise Model for Phased Arrays with Distributed Phase-Locked Loops.

Clock Configuration

Clock configuration is performed by on-device firmware based on parameters found in the device profile. These parameters are found in the **adi_apollo_clk_cfg_t** structure and described in Table 11.

Table 11. Clock Configuration Structure and Enumerations

Type	Profile Parameter Name	Description
bool	single_dual_clk_sel	FALSE: Single (Center) clocking, TRUE: Dual clocking
bool	clk_path_sel	FALSE: On-chip PLL, TRUE: External clock
uint32_t	ref_clk_freq_kHz	Clock PLL's input reference frequency. Must be less or equal to 1 GHz. Note, ensure that the phase-frequency detector (PFD) frequency is less than or equal to 500 MHz. The PFD frequency is ref_clk_freq_Khz / ref_clk_div, where ref_clk_div is programmed in the clock PLL configuration portion of device profile.
bool	clocking_mode	divG clocking mode ADI_APOLLO_CLOCKING_MODE_SDR_DIV_8 = 4T4R DAC Fs = ADC Fs ADI_APOLLO_CLOCKING_MODE_SDR_DIV_4 = 8T8R DAC Fs = ADC Fs ADI_APOLLO_CLOCKING_MODE_DDR_DIV_4 = 4T4R DAC Fs = 2* ADC Fs ADI_APOLLO_CLOCKING_MODE_SDR_DIV_4_DDR_DIV_2 = 8T8R DAC Fs = 2* ADC Fs
bool	dac_divby2 [ADI_APOLLO_NUM_SIDES];	Divider control for DAC clock path. TRUE: Divide by 2

SAMPLING CLOCK AND DISTRIBUTION OPTIONS

Table 11. Clock Configuration Structure and Enumerations (Continued)

Type	Profile Parameter Name	Description
bool	adc_inclk_invert0 [ADI_APOLLO_NUM_SIDES];	Enable ADC 0 clock inversion. TRUE: invert clockIt configures ADC Side A Channel 0 and Side B Channel 0
bool	adc_inclk_invert1 [ADI_APOLLO_NUM_SIDES];	Enable ADC 1 clock inversion. TRUE: invert clockIt configures ADC Side A Channel 1 and Side B Channel 1
uint32_t	dev_clk_freq_kHz	Device clock frequency. It is either Direct clock or clock PLL output frequency.
uint8_t	arm_clk_div	Arm clock divider. Ensure dev_clk_freq_kHz / divG_modulus / arm_clk_div <= 500 MHz, where divG_modulus is 8 if clocking_mode is ADI_APOLLO_CLOCKING_MODE_SDR_DIV_8. Otherwise, divG_modulus is 4.
uint8_t	serdes_clk_div	Divider for SerDes PLL reference clock generator. The reference frequency is dev_clk_freq_kHz / divG_modulus / serdes_clk_div, where divG_modulus is 8 if clocking_mode is ADI_APOLLO_CLOCKING_MODE_SDR_DIV_8. Otherwise, divG_modulus is 4. A value of 1 is not supported.

Using ADC Clock Inversion for Time-Interleaving ADC Samples

The adc_inclk_invert0 and/or adc_inclk_invert1 parameters described in Table 11 can be used to invert the clock to one of the ADC's on the A and/or B sides of the Apollo MxFE in order to time interleave the ADC samples and effectively achieve twice the sample rate of a single ADC. API functions including example code will be supported in a future revision of the Apollo MxFE API package. The Apollo MxFE user guide will fully document this feature once the API is available.

HIGH SPEED CLOCK RECEIVER INPUT

Figure 33 shows a simplified diagram of the clock receiver used for the single as well as dual clock options. Note these receivers are for direct RF clock while the on-chip PLL has its dedicated input for its reference clock. The clock receiver supports operation between 4 to 20 GHz with input power levels between -1 to +5 dBm. The Apollo MxFE includes an on-chip clock power detector that senses whether the input drive level falls within these thresholds. The API function **adi_apollo_clk_mcs_input_power_status_get()** reads back the detector and returns the status via the adi_apollo_clk_input_power_status_e enumeration. The clock receiver input (shown in Figure 33) is self-biased with a nominal common-mode voltage (V_{CM}) of 0.65 V and a differential impedance of 50 Ω across the input pins, CLKP and CLKN. AC coupling of the external clock source is required. The "A" and "B" clock receivers are enabled when dual clocking mode is used while the center clock receiver is disabled and vice-versa for single clocking mode.

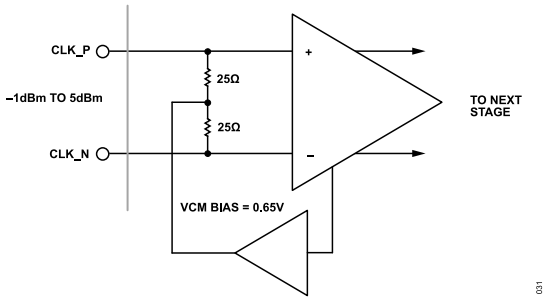

Figure 33. Clock Receiver Input Simplified Equivalent Circuit

The additive jitter and phase noise contribution from the clock receiver depends on the input slew rate and input voltage level. This additive jitter can limit the achievable noise floor performance of a DAC or ADC when operating under large signal conditions with high frequency content. To improve the phase noise performance, use a higher slew rate clock input signal.

Figure 34 shows the combined phase noise from the clock receiver to the DAC's output for different clock input sinusoidal wave drive levels with the DAC operating at 20 GSPS. The phase noise is measured with the DAC output reconstructing a 2.6 GHz sinusoidal signal with a full-scale of -1 dBFS. The phase noise of the 20 GHz clock source (normalized to 2.6 GHz) is also provided to show the additive phase noise from the device. Note that the labeled clock powers correspond to the power setting on the clock source signal generator and do not account for loss from the clock source to the Apollo MxFE clock pins. The plot shows that the drive level mostly impacts the high frequency offset phase noise (> 1 MHz) with drive levels of 16 dBm and above, resulting in the optimal wide offset performance. The clock peak detector reported as valid range the values of 16 dBm to 21 dBm.

SAMPLING CLOCK AND DISTRIBUTION OPTIONS

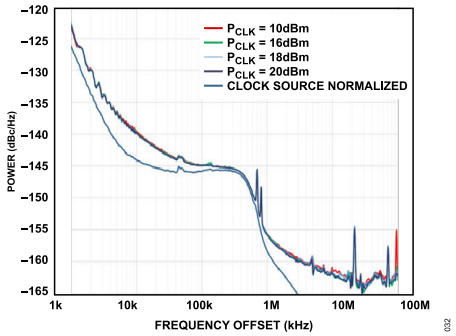

Figure 34. Single Sideband Phase Noise vs. Frequency Offset for Different Clock Input Power (Pclk), f_{OUT} = 2.6 GHz, External 20 GHz Clock

The quality of the clock source and the interface to the CLKP pin and CLKN pin directly impact ac performance. Ensure that the external clock path remains clean of any power supply or printed circuit board (PCB) coupling induced noise (both Differential and Common-mode) and select the phase noise and spur characteristics of the clock source to meet the target application requirements. Figure 35 shows a simple single-ended interface when using an RF driver amplifier specified for low phase noise. Such an amplifier may be used in a multi-chip clock distribution application where a power divider may follow the amplifier. An optional attenuator may be required to limit the input power into the clock receiver input. The clock receiver can correct for up to 4pS P/N skew and will correct for duty cycle errors incurred by low amplitude or edge rate.

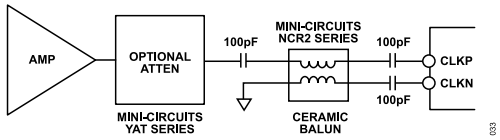

Figure 35. Balun Coupled Differential Clock

The ADF4382 as shown in Figure 36 is the recommended direct RF clock source since it provides industry leading phase noise and jitter extending to 22 GHz operation. It provides two differential CML outputs with each of the single-ended outputs having a 50 Ω load resistor to its internal 3.3 V supply. DC blocking capacitors are required to couple the differential output to clock receiver input. Low value RF inductors connected to the 3.3 V supply can be used to improve matching. Due to the high frequency output capabilities of the ADF4382, it is highly advised that simulation using the S parameters of the ADF4382 output stage, the clock receiver input stage, the passive components, and extracted differential PCB trace be used to optimize the match such that sufficient signal level is delivered to the clock input receiver. Please note that the output termination of the ADF4382 is 100ohms differential while the apollo clock input is 50 ohms differential. Care should be taken to minimize reflections on the trace between the two.

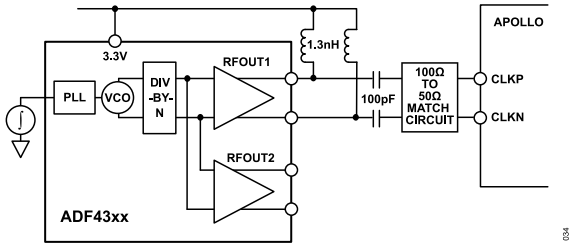

Figure 36. ADF4382 Differential CML Sample Clock

Figure 37 shows the differential input return loss curve for the clock inputs up to 22 GHz with a reference impedance of 50 Ω differential. The S-parameters will be made available for download on the AD9084 and AD9088 product page. For best RF performance, use an ac analysis model when optimizing the frequency response to an external component (such as a balun or the ADF4382), along with an extracted PCB layout model. Keysight ADS models are available from ADI, which include an ac analysis model and the S-parameters of the CLK input.

SAMPLING CLOCK AND DISTRIBUTION OPTIONS

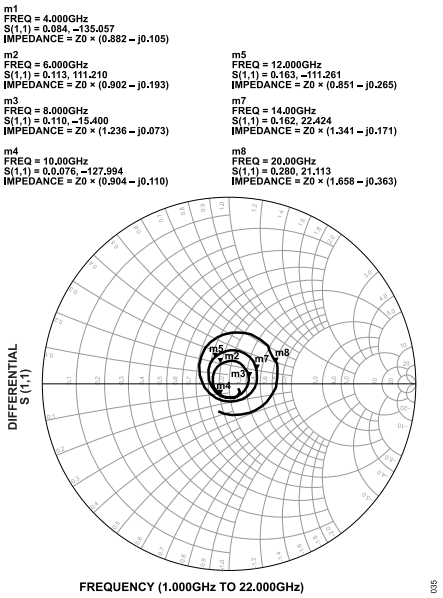

Figure 37. Clock Receiver Differential Input Return Loss

PLL CLOCK MULTIPLIER

An optional PLL Multiplier can be used in applications where reduced phase noise performance as well as ≤ 14 GHz ADC operation is still sufficient to meet the target applications specifications. The PLL clock multiplier shown in Figure 38 is an integer-N type with dual core octave VCO core and on-chip programmable loop filter. The maximum phase detection frequency, f_{PFD}, supported is 500 MHz. An on-chip divider, D, is used to generate center clock frequencies below the VCO's lower 7 GHz limit. The PLL multiplier is automatically powered down when the external RF clock input option is selected. The PLL is configured using the profile settings shown in Table 12 and calibrated during the initialization process via clock configuration firmware. High level profile settings are passed to the clock configuration firmware to configure and calibrate it.

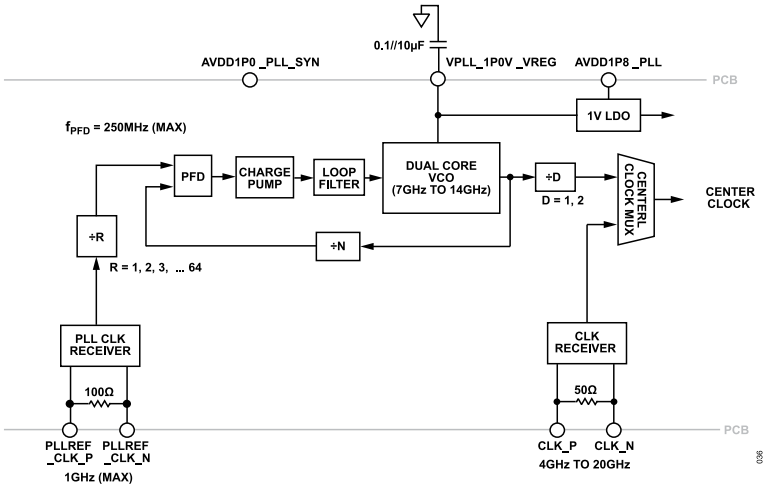

Figure 38. Block diagram of on-chip PLL Clock Multiplier

The PLL clock receiver is a self-biasing differential input with on-chip 100-ohm termination. It must be externally driven with an ac coupled input signal between 0.2 and 1 Vp-p. High slew rate input signals such as PECL or CML is preferred to reduce any phase noise contribution from the PLL clock receiver. The maximum input frequency must be limited to 1 GHz.

SAMPLING CLOCK AND DISTRIBUTION OPTIONS

CLOCK PLL CONFIGURATION

The clock PLL configuration is performed by on-device firmware based on parameters found in the device profile. These parameters are found in the **adi_apollo_clk_pll_cfg_t** structure of the API and are described in Table 12. These values are auto-calculated and populated in the device profile according to the user clocking requirements by the profile generator. Changing the clocking parameters requires the user to change the device profile via the profile generator tool that is part of the ACE Evaluation Software. Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 12. CLK PLL Profile Settings

Type	Profile Parameter Name	Description
uint32	loop_bandwidth	PLL Loop bandwidth. (valid range: 1kHz - 10 MHz)
uint8	phase_margin	PLL Phase Margin in degrees. (valid range 0 - 80)
uint8	div_range	when high, enables vco divby1 from root divider
uint8	div2	when high, enables vco divby2 from root divider - div_range_mode must be zero
uint8	power	PLL Power setting
uint8	ref_clk_div	PLL ref clock divider. (valid range: 0 - 31)
uint8	i_bleed_en	PLL bleed ramp enable
uint32	feedback_int	Integer portion of feedback factor
uint32	feedback_frac	Fractional portion of feedback factor

DEVICE SYNCHRONIZATION

SYNCHRONIZATION NOMENCLATURE

Table 13 defines common terms and variables associated with device synchronization.

Table 13. Device Synchronization Terminology

Term	Description
Internal SYSREF	An internal clock divided down from the sample clock used as the synchronization timing reference for the Apollo MxFE device synchronization and event handling. Period must be programmed, regardless of JESD204B/C subclass mode Apollo MxFE has 1 (all clocks) or 2 (A/B) depending on Single or Dual clocking mode respectively In subclass 0 mode, it is not aligned to external SYSREF and frequency is not limited by the LMFC/LEMC. Is aligned to external SYSREF for subclass 1 operation. Max Frequency 156.250MHz (can be higher than LMFC/LEMC frequency)
External SYSREF	Board/User Level SYSREF used to align internal SYSREF to system timing. Applied to Apollo MxFE at 1 or 2 inputs depending on the clocking mode Periodicity - Periodic SYSREF period must be a multiple of the internal SYSREF period and can be between Internal SYSREF and JESD204B/C SYSREF (sub-multiple of the least common multiple (LCM) of the LMFC/LEMC of the JESD204B/C transmitter and receiver) Gapped Periodic SYSREF – Shortest time between two adjacent rising edges of gapped periodic waveform must equal internal SYSREF period
BSYNC	Bi-directional Sync. A sub-type of “External SYSREF”, sometimes referred to as bi-directional SYSREF
JESD204B/C SYSREF	Legacy JESD204 synchronization signal applied to converters or logic devices for the purpose of implementing subclass 1 (and device) synchronization. Frequency is an integer sub-multiple of the least common multiple (LCM) of the receiver and transmitter LMFCs (JESD204B) or the receiver and transmitter LEMCs (JESD204C)
SYSREF alignment	The process of aligning the internal SYSREF to an external or JESD204B/C SYSREF input signal
SYSREF averaging	Averaged sampled SYSREF allows for averaging of the SYSREF noise but not programmable time offset
MCS Block	Multi-chip Synchronization block – sometimes referred to as “MCS”. There are 3 instantiations of the MCS block: Central is used for single clock configurations A-side and B-side are used for dual clock configurations
MCLK	MCLK is the system “main source clock” (aka device clock) and can be sourced externally (direct clocking) or from the on-chip PLL (derived clocking). Frequency is equal to the highest clock frequency needed for converter sampling in the device (either ADC sample rate or ½ * DAC sample rate). Frequency range is 8-20 GHz for AD9084 , and 5-8GHz for AD9088.
TDC	An averaging Time-to-Digital Converter used for measuring the time difference between External and Internal SYSREF rising edges. The use of the TDC allows for synchronizing to a programmable time offset and sub-MCLK accurate multi-chip alignment. This block is used by MCS Init and Tracking Calibrations.
Sysref_Align	An alignment phase wherein the Internal SYSREF is synchronized to the External SYSREF within ± ½ clock cycle of a programmable time offset.
Trigger	Externally applied signal captured relative to Apollo MxFE’s internal SYSREF for the purposes of DSP synchronization and high-speed event timing. Combination of Trigger and Internal SYSREF replaces need for (external) JESD204B/C SYSREF if internal SYSREF frequency > JESD204B/C SYSREF (the LMFC/LEMC frequency) - See Trigger Receiver Input section.
Sync Pulse	Internally generated signal distributed to various circuits to achieve device synchronization. Not synonymous with SYNCIN or SYNCOUT.
jtx_conv_clk	= f_{ADC}/(DCM × jtx_ns), DCM is total decimation,
jrx_conv_clk	= f_{DAC}/(INTERP × jrx_ns), INTERP is total interpolation
jtx_ns	number of samples processed per the jtx_conv_clk (auto calculated based on DCM, JESD mode S, and other values)
jrx_ns	number of samples processed per the jrx_conv_clk (auto calculated based on INTERP, JESD mode S, and other values)
f_{ADC}	ADC sample clock frequency
f_{DAC}	DAC sample clock frequency

DEVICE SYNCHRONIZATION

SYNCHRONIZATION PROCESS OVERVIEW

This section gives an overview of the device synchronization processes used on the Apollo MxFE.

The goal of synchronization is to align, in time, a SYSREF clock internal to the Apollo MxFE ("system divider") with an external SYSREF/BSYNC reference clock and maintain the alignment to a specified tolerance. All ADC and DAC sampling timing will be reset relative to this system divider. Since the ADC and DAC cores are located at the ends of clock distribution trees, their sample timing will incur a delay relative to the system divider. This delay may also vary across devices. Therefore, multiple devices must be synchronized.

Once aligned, the internal SYSREF can serve as the "heartbeat" of the device, and numerous digital events within the Apollo MxFE DSP may be aligned to its clock edge, distributed internally as trigger signals to the various DSP blocks.

There are three phases to device synchronization process:

- Phase 0 (sysref_align): SYSREF configuration and the internal SYSREF alignment to an external SYSREF edge which is required in JESD204 subclass 1, or the dual clock mode even in JESD204 subclass 0.
- Phase 1 (clock_divider_sync): The alignment of all the internal clock domains to the internal SYSREF, including analog, digital path, JESD clocks and FIFOs synchronization. This phase is required for all the synchronization scheme.
- Phase 2 (trig_sync): JESD (LMFC/LEMC), Rx and Tx datapath clocks re-synchronization and Time Stamp reset.

This three-phase synchronization procedure is illustrated in Figure 39.

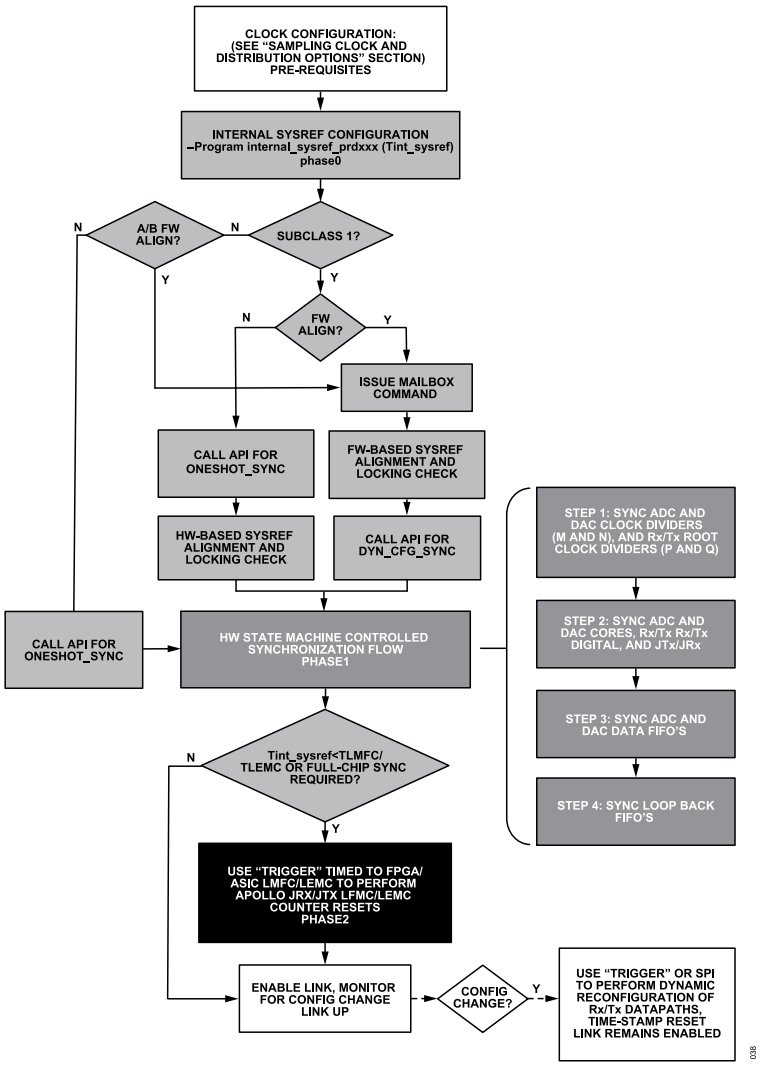

Figure 39. Device Synchronization Process Flow

DEVICE SYNCHRONIZATION

Phase 0 Overview

Phase 0, or sysref_align, performs the alignment between the internal SYSREF and the external SYSREF, before the alignment happens, the internal SYSREF frequency needs to be programmed (details refers to SYSREF Configuration). The sysref_align is required in JESD204 subclass 1 where the internal SYSREF(s) need to be aligned to the external SYSREF(s) applied to the part. Internal SYSREF alignment is also necessary if the internal SYSREFs of the A and B-sides in dual clock mode need to be aligned, even when operating in JESD204 subclass 0 mode. If the user has little interest in aligning internal SYSREF with external then Phase 0 can be skipped (e.g. single clocking mode in subclass 0). Sysref_align can be performed by either of the two methods below:

- Hardware based SYSREF alignment which supports subclass 1 in the Oneshot_sync mode. Refer to Hardware Based SYSREF Alignment
- TDC and Firmware based SYSREF alignment (refer to TDC and Firmware-based SYSREF Alignment) which supports both subclass 1 in single and dual clock modes and subclass 0 in dual clock mode. It uses a time-to-digital converter (TDC) which averages several thousand SYSREF pulses to measure the deltas between SYSREFs on the picosecond scale. It can perform repeatable alignments from startup to startup.

Phase 1 Overview

Phase 1, or clock_divider_sync, is performed by the hardware state machine to align all the converters (includes their FIFOs), all the digital clocks, Rx/Tx datapath, JESD and the loopback FIFOs to the internal SYSREF. This phase is required for all the synchronization scheme and its routine can be initiated by the ways shown below:

- Calling API function - **adi_apollo_clk_mcs_sync_only_set()** to kick off the synchronization state machine. The state machine can be readback by calling API function - **adi_apollo_clk_mcs_sync_only_get()**. Alternatively, API function - **adi_apollo_clk_mcs_dyn_sync_sequence_run** can be called. This function intends to call the synchronization to the different blocks sequentially to mitigate the power peak during this phase.
- The clock_divider_sync is part of the Oneshot_sync no matter that it's subclass 0 or subclass 1. In this case API function - **adi_apollo_clk_mcs_oneshot_sync()** triggers the state machine to synchronize individual blocks after the SYSREFs alignment for subclass 1.
- If the sysref_align is performed by the firmware-based alignment, clock_divider_sync is automatically performed at the end of sysref_align. The user doesn't need to run this phase 1 but need to follow the MCS programming flow in TDC and Firmware-based SYSREF Alignment

Phase 2 Overview

Phase 2, or trig_sync, is the phase to re-synchronize Rx and Tx datapath clocks, JESD (LMFC/LEMC), and reset the Time Stamp using either internal or external trigger. This phase is necessary under the following conditions:

- In dual clock mode where A and B-sides need to be synchronized on single device.
- The internal SYSREF period is shorter than the LMFC/LEMC of JESD204B/C. Programming the internal SYSREF with shorter periodicity allows for faster response time to synchronize a data-path reconfiguration event. If the internal SYSREF period programmed during phase 0 is longer than LMFC/LEMC of the JESD204, JESD and Serdes (both Jtx and Jrx) could be synchronized in Phase 1.
- Multi-chip synchronization where the system needs to synchronize over multiple devices which needs external trigger from the hardware pins.
- Dynamic reconfiguration in which scenarios, CDDC/FDDC/FSRC or CDUC/FDUC/FSRC datapath reconfiguration happens in run-time.
- Hopping cases which refers to CNCO, FNCO, CFIR, PFILR, BMEM delay hopping.
- When a system re-synchronization is needed.

The trig_sync can be actuated by either using the trigger inputs (hardware pins) or SPI writes. However, SPI actuation of these resets does not ensure any level of synchronization from device to device because SPI writes could happen at different time from device to device. If the multiple devices synchronization is required, the hardware trigger inputs can be used to receive the same trigger pulse from the system to all the devices simultaneously, each device will sample its trigger on its internal SYSREF rising edge and issue the reset at the following internal SYSREF rising edge to the blocks which are configured to be reset.

SYNCHRONIZATION HARDWARE

SYSREF Input and Setup/Hold Detector

SYSREF Hardware Input

DEVICE SYNCHRONIZATION

There are 3 sets of SYSREF/CLK inputs which includes **single SYSREF** and clocking for the whole chip or **dual SYSREFs** and clock inputs to the A and B-sides individually. Refer to the Sampling Clock and Distribution Options section for details on the clocking architecture.

The common mode voltage on the SYSREF inputs is set by the SYSREF source device because there is no internal common mode bias in Apollo MxFE. The SYSREF receiver supports a wide common mode input range between 0.4 - 1.4 Volts. The cm_above_900mv parameter in the adi_apollo_sysref_inp_cfg_t structure in the device profile is used to configure the Apollo MxFE whether the input SYSREF common mode voltage is above 900mV (by a value of "TRUE") or below 900mV (by a value of "FALSE"). See the AD9084 data sheet for detailed input and output electrical specifications on the SYSREF interface.

To achieve repeatable synchronization, the proper setup and hold timing need to be met between SYSREF and device clock (MCLK), the SYSREF needs to be stable during the keep out window (KOW = hold time (t_{HOLD}) + setup time (t_{SETUP})). The KOW for Apollo is 28 picoseconds as illustrated in Figure 40 , Note that as shown in the figure, internal path differences between the device clock and external SYSREF can result in KOW appearing at an offset (13 picoseconds) of MCLK.

The user can use the function of SYSREF Setup Hold Detector to check the timing of setup and hold and adjust the external SYSREF to meet the KOW timing requirement. The detail is described in the next section.

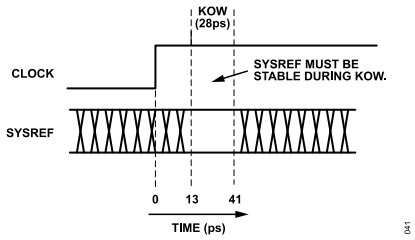

Figure 40. SYSREF setup and hold time KOW

SYSREF Setup Hold (SH) Detector

The Apollo MxFE has **sh_before** and **sh_after** parameters that can be used to check if a potential setup or hold time condition exists on the SYSREF_x_P/N input. As of the writing of this document, The effectiveness of this method for determining robust SYSREF capture has not been fully characterized but has been designed to be effective for an MCLK up to 20GHz . To validate that the SYSREF signal is being captured reliably, the sysref_phase parameter can be read (see **SYSREF Monitor Mode** in Hardware Based SYSREF Alignment). Multiple readbacks with the same value (clearing the readback value in-between readbacks) are indicative of a successful SYSREF capture. Recall, if using TDC and Firmware-based SYSREF Alignment, phase 0 is performed with the MCS's TDC block, which is not the same alignment procedure executed by oneshot_sync in subclass 1. Namely, the sysref_phase readback value will be dependent on the alignment method and offset between external and internal SYSREF.

The sh_before and sh_after parameters act as pseudo thermometer indicators (4 bits) of the potential for a timing error as indicated in Table 14. These parameters are returned when calling the TBD sub-function. If the returned value indicates a potential timing violation, adjust the SYSREF phase until the returned value is in the safe region. A TBD function will abstract the sh_before and sh_after parameters for the user to easily determine the likelihood of a timing violation as illustrated in Table 15.

This table will be supplanted by an API readback:

Table 14. SH Detector Parameter Decode

sh_before [1:0]	sh_after [1:0]	Conditions
11	00	Likelihood to have setup or hold violations. External SYSREF in KOW. Adjust the external SYSREF phase to avoid this region.
10	00	Setup violation unlikely but closer to KOW region. It is advised to advance the External SYSREF with respect to MCLK (Delay MCLK with respect to External SYSREF)
00	00	Safe region: External SYSREF between KOW and optimum sampling point.Optionally, consider advancing the External SYSREF with respect to MCLK (Delay MCLK with respect to External SYSREF)
00	10	Safe region: External SYSREF between KOW and optimum sampling point, closer to the ideal sampling point. Optionally, consider advancing the External SYSREF with respect to MCLK (Delay MCLK with respect to External SYSREF)
00	11	Safe region: Optimal External SYSREF Sampling Point
01	11	Safe Region: External SYSREF between KOW and optimum sampling point, closer to the KOW.Optionally, consider delaying the External SYSREF with respect to MCLK (Advance MCLK with respect to External SYSREF)

DEVICE SYNCHRONIZATION

Table 14. SH Detector Parameter Decode (Continued)

sh_before [1:0]	sh_after [1:0]	Conditions
11	11	Safe Region: External SYSREF between KOW and optimum sampling point.Optionally, consider delaying the External SYSREF with respect to MCLK (Advance MCLK with respect to External SYSREF)
11	01	Hold violation unlikely but closer to KOW region. It is advised to delay the External SYSREF with respect to MCLK (Advance MCLK with respect to External SYSREF)

Table 15. SH Detector API Output Decode

API Output SH_Value	Conditions
11	Good Region
10	Good region but closer to hold side violation. ~8-15pS away.
01	Bad Region ~ (26pS wide)
00	Good region but closer to setup side violation. ~8-15pS away

SYSREF Configuration APIs

The preliminary information includes the API function names mentioned in the SYSREF related section above as well as the parameters described in Table 16. Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 16. SYSREF Related Functions

API Function	Description
adi_apollo_clk_mcs_sysref_en_set(adi_apollo_device_t* device, uint8_t enable)	Enable SYSREF receiver on hardware
adi_apollo_clk_mcs_sysref_en_get(adi_apollo_device_t* device, uint8_t enable)	Get the SYSREF receiver enable state
adi_apollo_clk_mcs_sysref_internal_term_en_set(adi_apollo_device_t *device, uint8_t enable)	Enable the SYSREF internal termination resistor

MCS Hardware Block

As described fully in the Sampling Clock and Distribution Options and Single and Dual Clock Modes sections, the clocking and synchronization for the Apollo MxFE is highly configurable. The description to follow is meant to abstract the information given in the those sections to facilitate a system-level understanding on how to configure the Apollo MxFE according to the needs of the end application.

As illustrated in Figure 30 figure, there exists an MCS block for each of the clocking regions to support single (center region) or dual (A-side and B-side) clock modes. Figure 41 shows the elements of the MCS block which contains:

- A global MCS divider designated as "divG"
  - responsible for generating the root clock used by MCS
  - Note that divG retimes each of the signals going through it in Figure 41
- A sync logic block
  - This block is responsible for generating the internal SYSREF and sync pulses.
  - It also aligns the internal SYSREF with an external SYSREF when in JESD204 subclass 1 using oneshot_sync.
- A Time-to-Digital converter "TDC"
  - This block is responsible for precise relative measurements between the external SYSREF and the divG-retimed internal SYSREF.

The clock output of the MCS block is distributed along with the sync pulse where they can be used to fully synchronize the device according to the synchronization mode that is selected. That is, single or dual clock mode and JESD204 subclass 0 or subclass 1 mode.

DEVICE SYNCHRONIZATION

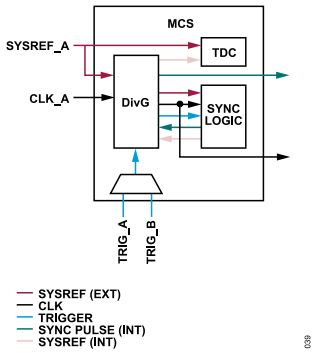

Figure 41. MCS Block Diagram

Trigger Receiver Input

As illustrated the Figure 30, the Apollo MxFE has 4 trigger inputs, TRIG_0_A, TRIG_1_A, TRIG-0_B, and TRIG1_B. All of these are routed to a trigger mux within each of the MCS blocks such that each can independently choose which trigger input to use when a "trigger sync" is required. Regardless of single or dual clocking mode, each trigger can be used to provide synchronization events to any or all of the four digital data paths(Receive Data paths A and B, Transmit Data paths A and B).

When necessary for the user system implementation, the mapping of the appropriate trigger input to the MCS blocks being used (as determined by the single_dual_clk_sel parameter) is set by the variables inside the **adi_apollo_mcs_trigger_cfg_t** structure of device profile. The mapping will happen during device startup and when using the TDC and Firmware-based SYSREF Alignment. The possible values the configuration variables can take are 0 to 3 inclusive, as defined inside the **adi_apollo_trig_pin_e** enumeration. Optionally, the user can call **adi_apollo_clk_mcs_sync_trig_map()** API function to set the trigger mapping in the cases where MCS Init is not used, as executing MCS Init will re-program MCS digital's trigger configuration based on device profile settings.

The API function will handle single and dual clock modes. The parameters in Table 17 are used to route the Pins through the trigger select mux.

Table 17. Trigger source selection parameters

rx_tx_select parameter (uint16_t)	Description	adi_apollo_trig_pin_e enumerations
ADI_APOLLO_RX_TX_ALL = 0x0F	Trigger pin selection control for transmitter B, transmitter A, receiver B, and receiver A	0: trig pin a0 1: trig pin a1 2: trig pin b0 3: trig pin b1
ADI_APOLLO_TX_DIG_B = 0x08	Trigger pin selection control for transmitter B	Same as above
ADI_APOLLO_TX_DIG_A = 0x04	Trigger pin selection control for transmitter A	Same as above
ADI_APOLLO_RX_DIG_B = 0x02	Trigger pin selection control for receiver B	Same as above
ADI_APOLLO_RX_DIG_A = 0x01	Trigger pin selection control for receiver A	Same as above

SYNCHRONIZATION MODES

The Synchronization flow at the system level can fall into three classes roughly following the use of the three synchronization modes described in the following sections and depending on the system clocking solution. For the highest degree of accuracy, it is recommended to use the Apollo MxFE companion clocking devices, the ADF4030 and ADF4382. These clocking solutions are described in the "More Applications Information's System Clocking Solutions" section and can achieve deterministic latency and multi-chip synchronization within less than 10 picoseconds.

A "legacy" synchronization flow, called oneshot_sync, is supported but is not the preferred method of synchronization using the Apollo MxFE since the level of accuracy for deterministic latency and multi-chip synchronization are less than optimal. Oneshot sync is controlled by the MCS Digital within each of the MCS blocks and can align the divG output to within ±0.5 MCLK cycles when the external SYSREF meets the setup and hold time requirements. However, as for the alignment of converters across multiple devices, there is an added uncertainty due to part-to-part variation in the delay of the input clock and SYSREF paths which could amount to as much as tens of picoseconds.

Synchronization modes are used to control how the device synchronization flow (the hardware state machine-controlled synchronization flow) is accomplished.

DEVICE SYNCHRONIZATION

The synchronization modes are programmed and executed with API functions located inside **adi_apollo_clk_mcs.c** as described in the subsections below.

Oneshot_sync Mode

Oneshot_sync mode basically executes two steps of synchronization, the first step is to align the internal sysref to external SYSREF for subclass 1 which is also called as hardware-based alignment (refer to Hardware Based SYSREF Alignment), and it is skipped by the subclass 0. The second step is to run the internal hardware synchronization which controlled by hardware state machine and required by all the synchronization schemes.

Oneshot_sync mode can be used by both single clocking mode and dual clocking mode. In single clocking mode, all the synchronization are controlled by the central MCS. In dual clocking mode, A side and B side synchronization are controlled by A side and B side MCS respectively, so in this case the A side and B side are not aligned. If A side and B side alignment is required, refer to TDC and Firmware-based SYSREF Alignment which provides the dual clocking internal alignment.

The key fact to keep in mind is that this SYSREF alignment method happens within the MCS Digital which is different from the TDC that is used by MCS Init and Tracking Calibrations.

There are some limitations of using oneshot_sync mode. For example:

- Incompatibility with ADF4030 BSYNC calibration, as oneshot_sync is unable to measure the time difference between external and internal SYSREF with better than one high-speed clock cycle of resolution.
- Incompatibility with ADF4382 Clock Alignment (MCS Tracking Calibration), as clock alignment uses MCS's TDC as means to precisely measure the time difference between the external and internal SYSREF, which is not the measuring technique used by MCS Digital.
- Inability to report External SYSREF's margin with respect to the keep-out window imposed by the retiming against the high-speed clock signal.
  - In other words, when using oneshot_sync at high SYSREF frequencies, the user will have difficulties obtaining repetitive synchronization latencies, as jitter on the external SYSREF can result in MCS Digital aligning the internal SYSREF to different points in time.
- It forces users to drive External SYSREF to both Side A and Side B when in dual clock mode.

The following steps are used to configure MCS Digital to perform internal SYSREF alignment when executing oneshot_sync in subclass 1:

- Enable the external SYSREF receiver with **adi_apollo_clk_mcs_sysref_en_set()** and argument ADI_APOLLO_ENABLE
- Wait 50 microseconds for the SYSREF receiver to come online
- Call **adi_apollo_clk_mcs_subclass_set()** with subclass argument set to 1

If working in subclass 0 or if SYSREF alignment is not desired, ensure that MCS is set to subclass 0 and the SYSREF receiver is disabled.

- Call **adi_apollo_clk_mcs_oneshot_sync()** to execute oneshot_sync.

Note, oneshot_sync in subclass 0 performs equivalently as dyn_cfg_sync.

Dyn_cfg_sync Mode

Dynamic configuration synchronization (dyn_cfg_sync) is basically used to trigger the hardware state machine to synchronize the internal clocks (includes all the converters (includes their FIFOs), all the digitalclocks, Rx/Tx datapath, JESD and the loopback FIFOs) without adjusting internal SYSREF regardless of subclass 0 or 1. As you can see, this mode is equivalent to oneshot_sync mode in subclass 0. In the meanwhile, this synchronization mode is required after internal SYSREF has been aligned by TDC and Firmware-based SYSREF Alignment in subclass 1 case.

If the Internal SYSREF is the same period as the JESD204 SYSREF, then dyn_cfg_sync can be used to align A and B-side clocks including LEMC but only in subclass 1 mode after sysref_align has been done, or in subclass0 after sysref_align has been done between A and B-sides.

To execute dyn_cfg_sync, use API function:

- adi_apollo_clk_mcs_sync_only_set()

In the case the supply network for the MxFE shows excessive transients or glitches that affect performance during the dyn_cfg_sync required during initialization, the user can use the following function to stagger the initialization of the different digital sections of the device. The function

DEVICE SYNCHRONIZATION

will automatically issue a dyn_cfg_sync after each block is allowed to be reset with a sync pulse. The blocks handled independently by the function are the digital data path receiver and transmitter of side A and B, JESD204B/C transmitter (JTx) and JESD204B/C receiver (JRx) of Side A and B.

- adi_apollo_clk_mcs_dyn_sync_sequence_run()

Trig_sync Mode

Trigger synchronization (trig_sync) configures MCS Sync Logic to receive an external trigger signal presented at the TRIG_A and/or TRIG_B input pin to initiate the synchronization process. An internal sync pulse is distributed to the appropriate function blocks for JESD204B/C, NCO, FSCR as well as for the dynamic reconfiguration including the frequency hopping and profile hopping. Since the internal SYSREF has an integer timing relationship with the LMFC/LEMC, all clock resets are deterministic across power cycles.

Deterministic triggering is sourced by the system logic device, based on source-synchronous timing from the System Clocking Solution. A simplified block diagram is illustrated in Figure 42. Triggers from the logic device to itself and to the Apollo MxFE are repeatably spaced. The "sync time" for the Apollo MxFE and the logic device for clock alignment after the trigger event is also repeatable. Timing for the system trigger event is illustrated in Figure 43.

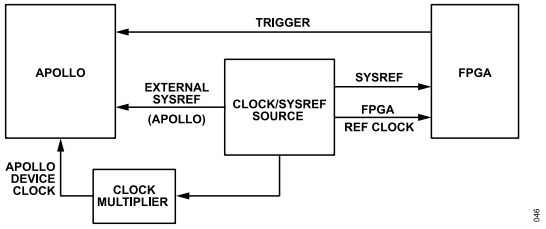

Figure 42. System Clocking w/Trigger Block Diagram

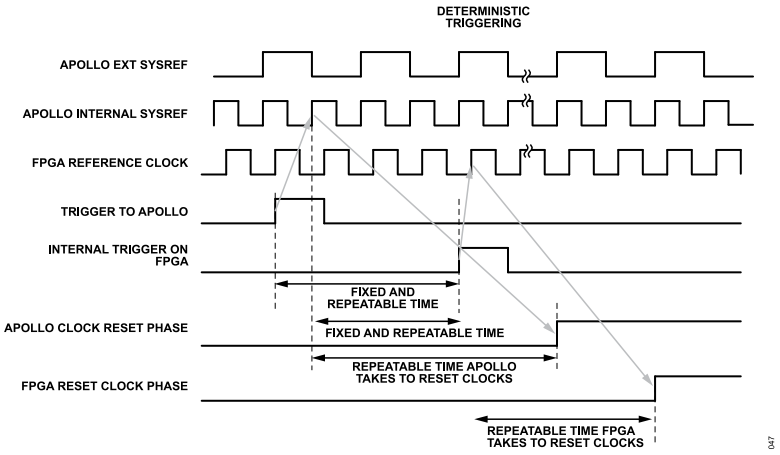

Figure 43. Deterministic Trigger Event Timing

Subsequent triggers after initial system synchronization can be used to change configuration using dynamic reconfiguration where certain blocks can be selected for reconfiguration without the need to reset the JESD204B/C links. Example timing for subsequent trigger events is illustrated in Figure 44. More details can be found in the Dynamic Reconfiguration section.

DEVICE SYNCHRONIZATION

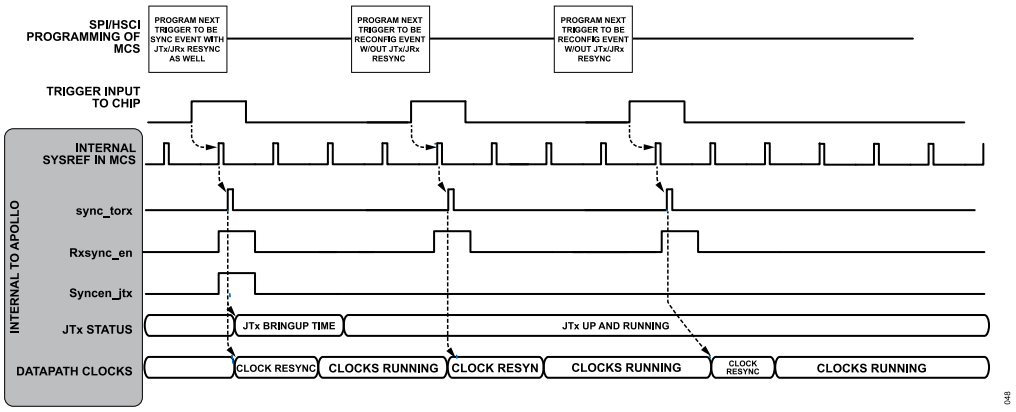

Figure 44. Example timing for subsequent trigger events

Trigger Monitor Mode for Trigger Timing Adjust

The device enters trigger monitor mode automatically when the trig_sync completes. The **adi_apollo_clk_mcs_trig_phase_get()** API function returns variables phase_0 and phase_1. The phase_0 corresponds to center MCS while phase_1 is zero in single clock mode. In dual clock mode, the phase_0 and phase_1 correspond to A- and B-side MCS respectively. This function provides the phase relationship between the trigger signal phase relative to the internal SYSREF. Note that the value which reads back after trig_sync completes reflects the phase of the most recent trigger rising edge occurring at the trigger input pin prior to the read-back.

A read-back value of 0 indicates that the trigger signal and internal SYSREF are aligned. A nonzero value such as 10, for example, indicates that the trigger rising edge is 10 divG cycles (MCLK/divG) later than the internal SYSREF's rising edge. In other words, to obtain the delay in MCLK cycles, the value should be multiplied by 8 if the clocking mode (Parameter "clocking_mode" in device profile) is 0, or 4 for the other clocking modes which is described in Table 19.

This trigger monitor mode can be used to adjust the timing of the trigger signal from FPGA to achieve the deterministic latency or multi-devices synchronization.

- Deterministic behavior is measured with respect to the "sync" pulse generated by the internal SYSREF rising edge that happens after the device receives the trigger signal from the system. If the trigger is too close to the rising edge of the internal SYSREF, the jitter on the trigger path may cause the latency varying +/-1 internal SYSREF clock cycle. Therefore, it's recommended that the trigger rising edge occurs in the middle of an internal SYSREF cycle.
- When read-back across multiple devices (or between side-A and side-B within the same device), the difference between the multiple phase_0 and phase_1 values can be used to make timing adjustments.

Relevant API Functions

Table 18 contains descriptions of API functions that are used to configure and interact with the synchronization modes.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 18. API Functions for the Synchronization Modes

API function	Description
adi_apollo_clk_mcs_subclass_set(adi_apollo_device_t *device, uint32_t subclass)	Configure the MCS subclass 0 or 1.
adi_apollo_clk_mcs_oneshot_sync(adi_apollo_device_t *device)	Synchronization with oneshot_sync mode.
adi_apollo_clk_mcs_sync_only_set(adi_apollo_device_t *device)	Issue the synchronization without the internal SYSREF alignment, the synchronization basically run in Dyn_cfg_sync mode.
adi_apollo_clk_mcs_dyn_sync_sequence_run(adi_apollo_device_t *device)	Synchronization run in Dyn_cfg_sync mode but it sequentially happens to each block to avoid the transients or glitches on the power supply.
adi_apollo_clk_mcs_sync_trig_map(adi_apollo_device_t *device, uint16_t rx_tx_select, adi_apollo_trig_pin_e trig_pin)	Configure which of the 4 trigger pins is mapped to reset each of Side A receiver, Side B receiver, Side A transmitter, Side B transmitter digital sections of the device.

DEVICE SYNCHRONIZATION

Table 18. API Functions for the Synchronization Modes (Continued)

API function	Description
adi_apollo_clk_mcs_trig_sync_enable(adi_apollo_device_t *device, uint8_t enable)	When enable = 1, it programs MCS Sync Logic to issue a sync pulse on the next internal SYSREF rising edge whenever the user sends a trigger signal to the appropriated trigger pin of the AD9084
adi_apollo_clk_mcs_trig_phase_get(adi_apollo_device_t *device, adi_apollo_trig_pin_e trig, uint16_t *phase_0, uint16_t *phase_1)	Reads the internal SYSREF phase at which the external trigger signal was sampled by the MCS Sync Logic digital. The units are MCLK/divG. The user is recommended to maintain the trigger phase close to internal_sysref_prd_digclk_cycles/2
adi_apollo_clk_mcs_trig_reset_disable(adi_apollo_device_t *device)	Disables reset of all phase 1 dividers and clock generators. This ensures that trig_sync's sync pulse does not reset converter dividers, data path clock generators, as well as FIFO clocks.
adi_apollo_clk_mcs_trig_reset_dsp_enable(adi_apollo_device_t *device)	Allows for the reset of receiver and transmitter digital clock generators to be reset on the next sync pulse

SYSREF CONFIGURATION

As implied in Figure 39, to start device synchronization, device clock must be configured and PLLs must be locked. Device configuration (for example, serial interfaces, converters, data paths, etc.) must be programmed. In the meanwhile the internal SYSREF period configuration is required no matter that the device operates in JESD204B/C subclass 0 or subclass 1 mode.

The internal SYSREF on Apollo is used as a timing basis to trigger the events within Apollo. It's essential for all of the internal timing control and synchronization events including the reset of the LEMC/LMFC, the reset of the NCO/FSRC counters, hopping between PFILT/ CFIR configurations or NCO frequencies, the reset of the time stamp counters, or the reconfiguration of the data path, etc.

The period of the internal SYSREF governs how often these events can be signaled so it needs to be programmed appropriately. The units for programming the SYSREF period are determined by the MCS clock ratio which depends on the device configuration. These configurations are provided in Table 19. The parameters used for setting the internal SYSREF period are part of the device profile (see Programming the Internal SYREF Period Parameters section).

Table 19. Clocking and Synchronization Settings per Device Configuration

Device Configuration	Sample Clock Ratio (f_{ADC} : f_{DAC})	Max ADC Fs (Gsps)	Max DAC Fs (Gsps)	MCLK (Gsps)	ADC_CLK divider	DAC_CLK divider	MCS MCLK Divider (/G)	Clocking _mode	MCS Clock Ratio
8t8r	1:1	8	8	8	/1	/2	4	1	MCLK/4
8t8r	1:2	8	16	8	/1	/1	4	3	MCLK/4
4t4r	1:1	20	20	20	/1	/2	8	0	MCLK/8
4t4r	1:2	14	28	14	/1	/1	4	2	MCLK/4

Guidelines for Setting the Internal SYSREF Period

1. The minimum SYSREF period allowed is 16 output cycles of the digital divider (divider G); while the maximum period is 65534 cycles of the divider G's output.
2. This setting is programmed with the device profile's mcs_cfg section using variables internal_sysref_prd_digclk_cycles_center , internal_sysref_prd_digclk_cycles_side_a, and internal_sysref_prd_digclk_cycles_side_b, which must all be programmed to the same value.
3. As shown in Table 19, the divider G's modulus is either 4 or 8 depending on the device configuration. In the case of a 4t4r device configuration with MCLK equals 20 GHz, the internal SYSREF periods can range from 16*8*50ps = 6400ps to 65534*8*50ps = 26213600ps.
4. If in traditional subclass 1 mode using a JESD204B/C SYSREF, the period of the internal SYSREF must be an integer multiple of the LCM of the following:
   - ADC sample rate/32 or DAC sample rate/32 (if operating DAC only) and the LMFC/LEMC:
   - LMFC/LEMC period of the DAC path's JESD204B/C receiver as determined by the JESD204B/C mode, DAC sample rate, and interpolation ratio. See the JESD204B/C Receiver Clock Relationships section.
   - LMFC/LEMC period of the ADC path's JESD204B/C transmitter as determined by the JESD204B/C mode, ADC sample rate, and decimation ratio. See JESD204B/C Transmitter Clock Relationships section.

DEVICE SYNCHRONIZATION

5. The period of the internal SYSREF may be less than an LMFC/LEMC period when using trig_sync or in subclass 0 operating mode but must adhere to the following constraints. Definitions for related variables are specified in Table 13.
   - Value of device profile's mcs_cfg. internal_sysref_prd_digclk_cycles_* variables:
     - Must be larger or equal to 16 and less or equal to 65534.
     - Must be an even number.
   - The period of the Center MCS' internal SYSREF is mcs_cfg. internal_sysref_prd_digclk_cycles_center * G (where G = 8 for 4t4r 1:1 mode, and = 4 for rest modes). The same equation applies for side A and side B MCS using the internal_sysref_prd_digclk_cycles_side_a and internal_sysref_prd_digclk_cycles_side_b variables.
6. Period of internal SYSREF must be integer multiple of the receiver root clock, the transmitter root clock, JTx conv_clk, JRx conv_clk, and fdac/32 (if the receiver BMEM is used).
7. The relationship of Rx/Tx root clock to MCLK can be found in Table 19 and conv_clk can be calculated by below formulas:
   - Jtx conv_clk = fadc/(DCM * Jtx_ns)
   - Jrx conv_clk = fdac/(INTERP * Jrx_ns)

JTX_NS and JRX_NS

As defined in the Table 13 table, jtx_ns and jrx_ns are the number of samples processed per jtx_conv_clk and jrx_conv_clk cycles, respectively. These values are calculated internally based on the number of samples per converter per frame ('S'), decimation/interpolation ratios and various other settings. For cases where FSRC is enabled and link_total_ratio = min(link_ddc_dec across all configs), the jtx_ns_override parameter is set and the computed jtx_ns_cfg value is overridden by on-chip firmware. (see the Fractional Sample Rate Converter (FSRC) for Transmit and Receive (AD9084 Only) section for details. The JRX_NS values per mode are listed in the JESD204B/C Receiver Mode Tables. JTX_NS is set based on the 'S' parameter and the DDC_NS value as governed by this equation:

JTX_NS = MAX(S, DDC_NS) (1)

S is the JESD204 parameter for number of samples per converter per frame.

DDC_NS is determined by:

- Total decimation ratio (DCM)
- The low_samp parameter, where:
  - Low_samp = 1 if Fs < 10GSPS in 4t4r mode or Fs < 5GSPS in 8t8r mode
- The chip configuration (4T4R vs 8T8R),
- and if the fine DDC is enabled (fine_en).
  - The fine DDC is enabled when the fine decimation value is greater than one or if the fine NCO (FNCO) is enabled.

The DDC_NS values are given in Table 20.

Table 20. DDC_NS Value for Determining JTX_NS

8T8R_en	fine_en	low_samp	DCM=1	DCM=2	DCM=3	DCM=4	DCM=6	DCM=8	DCM=12	DCM=16	DCM=24	DCM=32	DCM=48	DCM=64	DCM=96	DCM=128	DCM=192	DCM=256	DCM=384	DCM=768
0	0	0	32	16	8	8	4	NA	2	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
0	0	1	16	8	8	4	4	NA	2	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
0	1	0	8	8	8	8	8	4	4	2	2	1	1	1	1	1	1	1	1	1
0	1	1	8	4	8	4	4	2	2	1	1	1	1	1	1	1	1	1	1	1
1	0	0	16	8	4	4	2	NA	1	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
1	0	1	8	4	4	2	2	NA	1	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
1	1	0	4	4	4	4	4	2	2	1	1	1	1	1	1	1	1	1	1	1
1	1	1	4	2	4	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1

Internal SYSREF Programming Rules Per Device Configuration (Table 19)

The device profile parameters "internal_sysref_prd_digclk_cycles_center", "internal_sysref_prd_digclk_cycles_side_a," and "internal_sysref_prd_digclk_cycles_side_b" set the values of the internal SYSREF divider during the clock configuration portion of the Apollo MxFE Bring-up Procedure. In the following, internal_sysref_prd_digclk_cycles refer to all three device profile parameters. The device profile generator will compute the values for these parameters and populate the device profile appropriately. The computation is based on the following rules:

DEVICE SYNCHRONIZATION

For 4t4r 1:1 mode: fdac = fadc = fclk, divG = 8, receiver root clk = fadc/8, transmitter root clk = fdac/8,

- Internal SYSREF period = internal_sysref_prd_digclk_cycles * 8
  - = integer multiple of (8, DCM*Jtx_ns, INTERP*Jrx_ns) if receiver BMEM is not used or
  - = integer multiple of (8, DCM*Jtx_ns, INTERP*Jrx_ns, 32) if receiver BMEM is used

For 4t4r 1:2 mode: ½ * fdac = fadc = fclk, divG = 4, receiver root clk = fadc/8, transmitter root clk = fdac/8,

- Internal SYSREF period = internal_sysref_prd_digclk_cycles * 4
  - = integer multiple of (8, 4, DCM*Jtx_ns, INTERP*Jrx_ns/2) if the receiver BMEM is not used or
  - = integer multiple of (8, 4, DCM*Jtx_ns, INTERP*Jrx_ns/2, 32) if the receiver BMEM is used

For 8t8r 1:1 mode: fdac = fadc = fclk, divG = 4, receiver root clk = fadc/4, transmitter root clk = fdac/4,

- Internal SYSREF period = internal_sysref_prd_digclk_cycles * 4
  - = integer multiple of (4, DCM*Jtx_ns, INTERP*Jrx_ns) if the receiver BMEM is not used or
  - = integer multiple of (4, DCM*Jtx_ns, INTERP*Jrx_ns, 32) if the receiver BMEM is used

For 8t8r 1:2 mode: ½ * fdac = fadc = fclk, divG = 4, receiver root clk = fadc/4, transmitter root clk = fdac/4,

- Internal SYSREF period = internal_sysref_prd_digclk_cycles * 4
  - = integer multiple of (4, 2, DCM*Jtx_ns, INTERP*Jrx_ns/2) if the receiver BMEM is not used or
  - = integer multiple of (4, 2, DCM*Jtx_ns, INTERP*Jrx_ns/2, 32) if the receiver BMEM is used

Programming the Internal SYREF Period Parameters

The internal SYSREF period is set using the parameters in the **adi_apollo_mcs_cfg_t** structure of the device profile. The units used in the profile are MCLK/divG cycles and the parameters are described in Table 21. Refer to the “apollo_api.chm" file for more details on the device API functions, structures and parameters. This file is included in the API package in the \apollo-sdk\pkg\production\doc folder.

Table 21. “adi_apollo_mcs_cfg” Structure Parameters

Parameter	Description	Type	Comments
internal_sysref_prd_digclk_cycles_center	SYSREF period in MCLK/divG cycles when in single clock mode	uint16_t	divG equals 8 when clk_cfg. clocking_mode is 0. Otherwise, divG is 4
internal_sysref_prd_digclk_cycles_side_a	SYSREF period in MCLK/divG cycles for the A-side when in dual clock mode	uint16_t	divG equals 8 when clk_cfg. clocking_mode is 0. Otherwise, divG is 4
internal_sysref_prd_digclk_cycles_side_b	SYSREF period in MCLK/divG cycles for the B-side when in dual clock mode	uint16_t	divG equals 8 when clk_cfg. clocking_mode is 0. Otherwise, divG is 4

Internal SYSREF Programming Examples

The internal_sysref_prd_digclk_cycles parameters expect the internal SYSREF in MCLK/divG cycles. MCLK is the max value between the ADC clock and ½ DAC clock. divG is the modulus of the global digital divider, which can be either 8 or 4 depending on the clocking_mode setting. This example assumes the period is set to have an integer relation to the LMFC/LEMC period for the JESD204B/C mode being used. The formulas for LEMC and LMFC are as follows:

$$LMFC = LEMC = (DACCLK)/(S \times K \times Interp)$$

or

$$LEMC = (ADCCLK)/(S \times K \times Decimation)$$

The example use case in Table 22 uses JESD204C mode 47 with ref_clk = DAC_clk = 20 GHz and 4x4 coarse x fine interpolation, (i.e., Interp = 16).

Table 22. Example Calculation of LEMC Period

JESD204 Mode Number	L	M	F	S	N’	K (204B)	K (204C)	E (204C)	clocking_mode	divG
47	8	4	1	1	16	32	256	1	0	8

DEVICE SYNCHRONIZATION

So, LEMC = 20GHz/(1*256*16) = 4.8828125 MHz. MCLK, in this case is 20 GHz, so LEMC in MCLK cycles is 20GHz/4.8828125 MHz = 4096; Hence, the value for the internal_sysref_prd_digclk_cycles parameters part of the device profile would be 4096/divG = 4096/8 = 512. Note that with trig_sync, the "integer relation" mentioned above can be an integer sub-multiple as well as multiple. Namely, the internal SYSREF period can be an integer sub-multiple (higher frequency) than LMFC/LEMC. See trig_sync section.

SYSREF ALIGNMENT MODES

There are two methods, called alignment modes, by which the Apollo MxFE synchronizes its clocking, converters, FIFOs, and JESD204 blocks, including the alignment of its internal SYSREF to an external SYSREF. The primary and recommended method for alignment is through internal MCS calibrations, which uses the MCS block's TDC to align the internal SYSREF as well as align the A/B-sides in dual clock mode. The alignment process is divided in two steps. The first step is a coarse alignment, MCS Init Cal, which aligns the internal SYSREF based on an MCLK period adjustment basis. The second step is a fine alignment, MCS Tracking Cal, which interacts with the ADF4382 clocking device to achieve sub-picosecond alignment accuracy. Alternatively, there is a hardware-based method using an on-board state-machine. The hardware method is used when oneshot_sync is used in subclass1 and results in ± ½ MCLK cycle accuracy when setup/hold requirements are met between MCLK and the external SYSREF.

Hardware Based SYSREF Alignment

Hardware-based SYSREF alignment is, as described in Oneshot_sync Mode is the first step of the Oneshot_sync mode in subclass 1. Basically Apollo devices provide the hardware to sample the external SYSREF from the SYSREF inputs and then the internal SYSREF is synchronized to the captured SYSREF. Two options allowed for the users to configure how to capture the external SYSREF, **the single SYSREF mode** and **the averaged SYSREF mode**.

The single SYSREF and the averaged SYSREF synchronization require the following:

- Synchronous sampling of a single SYSREF pulse.
- Meets setup and hold time requirements for reliable synchronization, although these requirements are increasingly difficult to achieve as the sample rate increases.
- For the single SYSREF mode, SYSREF input jitter must be less than half of the difference between the CLK± input period and the SYSREF keep out window (KOW) of SYSREF. (Details refer to SYSREF Input and Setup/Hold Detector)

Single SYSREF Mode and Pulse Counting

This mode is the traditional SYSREF sampling mode. It is incorporated in oneshot_sync and allows the internal SYSREF alignment to the external SYSREF occurs on either the first SYSREF pulse after oneshot_sync is enabled or on the N-th SYSREF pulse if pulse counting is enabled. The pulse counting basically configures the device to wait a predetermined number of pulses before using one as a proper phase sample of the internal counter. The API function **adi_apollo_clk_mcs_sysref_count_set()** is the low-level function to configure the predetermined number of pulses to skip.

The single SYSREF mode is triggered by calling the function **adi_apollo_clk_mcs_subclass_set()** with argument subclass set to 1, followed by the **adi_apollo_clk_mcs_oneshot_sync()** function. This function enables the entire flow to occur including aligning internal SYSREF to external SYSREF and then triggering the hardware state machine-controlled sync.

Averaged SYSREF Mode

In the averaged SYSREF mode, the averaging function determines the mean phase of the SYSREF that is used to align the internal SYSREF and effectively filters SYSREF and clock Jitter. For Multi-chip synchronization with a high SYSREF jitter, the averaged SYSREF mode is preferred. The API function TBD is the low-level function that enables averaging mode and sets the number of SYSREF occurrences that are averaged using the "pulses" parameter as described in Table 16. This mode will be fully supported in future revisions of the Apollo MxFE API.

Figure 45 shows how the SYSREF averaging function insulates the synchronization logic from the effects of a miss sampled SYREF input signal. The number of SYSREF occurrences that are averaged is 2^N, where N is the value in the TBD parameter. Averaged SYSREF mode works the same way as single SYSREF mode, except for the position of the SYSREF is considered to be the mean of several SYSREF phases. The following conditions must be met to employ SYSREF averaging mode, and are increasingly difficult to achieve as the sample rate increases:

- Synchronous sampling of the mean SYSREF phase.
- Mean SYSREF location must meet setup and hold time requirements for reliable synchronization.
- Only support external SYSREF frequencies less than or equal to 39.0625MHz.

DEVICE SYNCHRONIZATION

Note that averaging SYSREF mode is only available for oneshot_sync and is not available for SYSREF monitor mode.

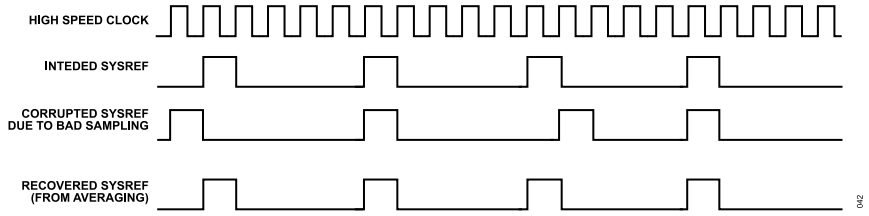

Figure 45. JESD204C Conceptual Illustration of Recovered SYSREF

Average SYSREF Mode with Pulse Counting

Users may enable pulse counting while in averaged SYSREF alignment mode. In this case, the first n SYSREF pulses will be ignored before averaging begins. API support for this is TBD

Internal SYSREF Delay

Users may add up to 255 MCLK cycles worth of delay between the external SYSREF and the aligned internal SYREF using the **TBD** API function.

SYSREF Monitor Mode

The device enters SYSREF monitor mode automatically when using the hardware-based alignment method after oneshot_sync completes. This mode can be exercised with the **adi_apollo_clk_mcs_sysref_phase_get()** API function. This readback value verifies the phase relationship between the incoming SYSREF signal relative to the internal SYSREF and the LMFC/LEMC boundaries. Note that the value read back after oneshot_sync completes reflects the phase of the most recent SYSREF leading edge occurring at the MCS' sync logic prior to initiating the readback.

A readback of 0 indicates that internal SYSREF (and LMFC/LEMC) are aligned to the external SYSREF. A nonzero value such as 10, for example, indicates that the SYSREF rising edge is 10 MCLK cycles later than the internal SYSREF.

SYSREF Error Window

Alignment between the external SYSREF and internal SYSREF can be monitored using the TBD API function. The SYSREF_ERR_WINDOW parameter sets how much SYSREF jitter or drift can be tolerated by the system. This parameter is set in number of MCLK cycles. Figure 46 shows the relation between the SYSREF_ERROR_WINDOW setting, the MCLK, and the SYSREF (averaged or sampled). If the external SYSREF is aligned to the internal SYSREF reference to within the limits set by the SYSREF_ERR_WINDOW parameter, no SYSREF alignment errors will be reported. The SYSREF_WITHIN_ERRWINDOW parameter is returned by the TBD function. A value of 1 indicates a SYSREF alignment error.

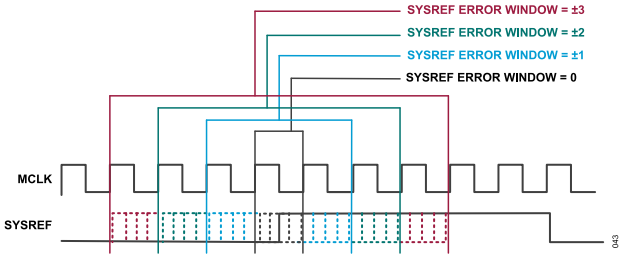

Figure 46. SYSREF_ERROR_WINDOW Setting Example

Avoiding the SYSREF Keep-Out Window (KOW)

When a high sampling clock is required, the SYSREF setup hold timing is difficult to meeting, the MCS Init alignment method ensures that the MCS and DL ambiguity associated with clocking the external SYSREF in the KOW is avoided. The details will be discussed in next session.

DEVICE SYNCHRONIZATION

TDC and Firmware-based SYSREF Alignment

As described in MCS Hardware Block, each MCS block has one TDC (Time to Digital Converter) for precise relative measurements between the external SYSREF and the divG-retimed internal SYSREF. The main process of this TDC and FW based SYSREF alignment is,

TDC block captures and samples two input signals (usually one is external SYSREF, the other is internal SYSREF) and sends the timing data to MCS block which calculates and accumulates the two signals' period and the time difference between the two signals' rising edge.
Firmware reads back time difference information from MCS and calculates the adjustment needed to adjust internal SYSREF, then issues command to adjust internal SYSREF which is executed by MCS Init calibration with a precision of ± 0.5 MCLK cycles (A coarse alignment).
When a fine alignment is required, the firmware implements MCS Tracking Calibration which interacts with the external clocking device (ADF4382) to achieve sub-picoseconds alignment accuracy.

MCS Init Calibration provides SYSREF alignment with a precision of ± 0.5 MCLK cycles and can be executed with AD9084 alone without the need of ADF4382. This calibration adjusts the internal SYSREF only once and does not track external and internal SYSREF time-difference drift. The alignment is performed by adjusting MCS's divG divider, which in turns adjusts the AD9084's internal SYSREF generated by MCS digital. Measurements are done by the MCS's TDC, and the desired time difference between internal and external SYSREF can be adjusted in steps of one MCLK cycle, as this is the native step size of divG's adjustment.

MCS Tracking Calibration requires at least one instance of ADF4382 generating the main clock for AD9084 in the user's system. This calibration provides a few picoseconds to sub-picosecond of precision depending on SYSREF period and MCS Tracking decimation setting. Unlike MCS Init, MCS Tracking can adjust AD9084's clocks to track and correct changes in the time difference between AD9084's internal and external SYSREF due to environmental factors. The user can also specify a finer time-difference offset to account for SYSREF trace length differences between AD9084 devices. The resolution of this adjustment is in the order of a few hundreds of femtoseconds, depending on ADF4382 and MCS tracking configuration.

The example of Firmware-based SYSREF Alignment Flow

The Apollo API package provides the example codes in **examples/ads10_apollo_ex_main/mcs_cal.c** file to demonstrate the programing and utilization of AD9084's MCS calibration process with AD9084 and ADF4382. The example codes execute MCS Init calibration for coarse alignment of the AD9084's internal SYSREF against the external SYSREF. The codes, then perform the MCS Tracking calibration for the fine tuning of the internal SYSREF alignment to the external SYSREF.

MCS Configurations: The configurations to the MCS Init and Tracking Calibrations are included in the device profile, and they are configured to the device in the boot-up sequence. The default programming in the example code is for the following system configurations:

- Single AD9084 device running in single clock mode.
- Single ADF4382 device driving AD9084's center clock receiver.
- AD9084's GPIO pins 15 and 16 drive ADF4382's DELSTR and DELADJ pins, respectively.
- External SYSREF frequency matches that of the AD9084's internal SYSREF frequency.
  - With device profile id00_uc06, it is 4882812.5 Hz.
- External SYSREF is a periodic signal. Namely, it is not missing cycles at random, i.e., gapped periodic.
- This signal is present at the AD9084's center SYSREF receiver.

The user should configure their system to provide the required SYSREF signal to successfully execute the example code.

Stage 1: MCS Init Calibration,

Note that the mcs_cal example code programs both AD9084 and ADF4382 in a particular order to ensure correct interoperability of the devices. After such configuration, the mcs_cal example code:

1. Triggers MCS Init with function **adi_apollo_mcs_cal_init_run().**
   - The user can adjust the desired final time difference between external and internal SYSREF in femtoseconds with the offset_C_femtoseconds variable. It is important to note that MCS Init can only adjust in steps of one MCLK cycle.
   - For initial evaluation, the user is recommended to leave offset_C_femtoseconds as 0. This will tell MCS Init to obtain an estimate of the intrinsic time difference between external and internal SYSREF as measured by the MCS's TDC during the beginning of the alignment process. The estimates are then used as offset_C_femtoseconds. The estimated time difference is a value between ± 0.5 MCLK cycles.

DEVICE SYNCHRONIZATION

Any integer factors of MCLK are removed. The estimate is reported as recommended_offset_C_femtoseconds within the MCS Init status structure. The time difference can arise due to system layout differences between the High-Speed clock path and SYSREF path.

- If a non-zero value is provided for offset_C_femtoseconds, MCS Init will bypass the early estimation of offset_C_femtoseconds and will use the user-provided offset_C_femtoseconds value instead. Therefore, a recommended_offset_C_femtoseconds value of zero is expected in this case. MCS Init will only succeed if it can align Internal and External SYSREF ± 0.4 MCLK cycles around the provided offset_C_femtoseconds.
- The example codes to configure offset_C_femtoseconds are included in the function of **adi_ads10_apollo_ex_mcs_init_cal_setup()**, and the absolute value of offset_C_femtoseconds must be less than 25% of the internal sysref period; otherwise, an error will be reported by MCS Init.

2. Retrieves MCS Init results, which contain TDC measurements with function **adi_apollo_mcs_cal_init_status_get().**
   - Similar to recommended_offset_C_femtoseconds, the variable diff_C_After_femtoseconds contains the residual time difference calculated from the last measurement performed by the TDC. Note that this value could be different from the recommended_offset_C_femtoseconds since both are based from different TDC measurements. The time difference is defined as the external SYSREF rising edge minus the internal SYSREF rising edge. Therefore, a negative time difference means that the internal SYSREF edge is occurring later than the external SYSREF edge. If the recommended_offset_C_femtoseconds or diff_C_After_femtoseconds variables are too close to +/- 0.5* MCLK period, it is recommended to shift the external sysref or MCLK source by at least +/-0.25* MCLK period to avoid variations in the final alignment of the internal sysref. This adjustment can be skipped when using MCS Tracking, as it can compensate by steps smaller than 1 MCLK cycle.
3. Check if MCS Init calibration is executed appropriately with function **adi_ads10_apollo_ex_mcs_init_cal_validate().**
   - Variable is_C_Locked is set to 1 if the absolute value of diff_C_After_femtoseconds minus the recommended_offset_C_femtoseconds (if the user did not provide a desired offset, or offset_C_femtoseconds if the user provided a non-zero value) is less than or equal to 0.4 times the clock period.

Stage 2: MCS Tracking Requirements,

After successfully running MCS Init, the mcs_cal example code runs MCS Tracking. MCS Tracking requires the following steps to be taken before proceeding with clock adjustments:

1. Within the adi_ads10_apollo_ex_mcs_tracking_cal_setup() function:
   - Update the MCS Digital's TDC decimation setting. A larger decimation value improves precision at the expense of longer TDC measurement time
2. Enable MCS Tracking calibration. This will set AD9084 to run one full correction cycle (i.e., TDC measurement + Adjustment) of the MCS Tracking flow every second; however, with the mcs_cal defaults, no adjustments will be sent until manually enabled by the mcs_cal example. The execution flow of MCS Tracking is dependent on user-defined configuration.
3. If not previously set during the configuration stage, the code sets MCS Tracking to execute the initialization phase. This indicates MCS tracking to enable MCS Digital's TDC hardware as well as AD9084's GPIOs.
4. Configure ADF4382 to accept adjustment commands sent by the Apollo MxFE's MCS Tracking through the two-wire interface.

After the above steps, the user has two ways to let MCS Tracking adjust clocks if needed. One option lets MCS Tracking send adjustments, as needed, every one second to ADF4382. The other option is a user-driven adjustment.

Stage 3: MCS Tracking Background Calibration,

The user can run the following sequence if there are no special considerations required for AD9084 clock alignment.

1. MCS Tracking Foreground with function **adi_apollo_mcs_cal_fg_tracking_run()**.
   - This indicates MCS tracking to quickly adjust ADF4382 output clock. MCS tracking will send packs of strobe signals to ADF4382 until the time difference between internal SYSREF and external SYSREF is within the track_win amount in femtoseconds, or when there are a few strobes left.
   - The Foreground approach is intended for correction of large timing errors, and it can be run only once. It is recommended to run it post MCS Init calibration.
2. MCS Tracking Background with function **adi_apollo_mcs_cal_bg_tracking_run()**.
   - This will indicate MCS Tracking to perform one TDC measurement and send one strobe, if needed, to ADF4382.

DEVICE SYNCHRONIZATION

- If the latest time difference error defined as (TDC measurement – track target) and reported as current_measurement is within plus or minus the tracking window amount, no strobe is sent. The user can set the desired tracking window and track target in femtoseconds with the variables track_win and track_target, respectively.
- Note that track_target is independent of offset_C_femtoseconds in the configuration structure; however, they both signify the desired time difference between external and internal SYSREF seen at the MCS' TDC.

3. The mcs_cal example code verifies every ten seconds that ADF4382 and AD9084's MCS Tracking are in synchronicity by comparing the correction values reported by MCS Tracking and the values present in ADF4382. The user can change this verification interval as needed.
   - Note, MCS Tracking must be frozen and unfrozen before and after retrieving MCS tracking data with functions **adi_apollo_mcs_cal_bg_tracking_freeze()** and **adi_apollo_mcs_cal_bg_tracking_unfreeze()** respectively.
   - mcs_cal retrieves MCS tracking data with function **adi_apollo_mcs_cal_tracking_status_get()**

One note is that MCS Tracking can avoid adjusting ADF4382's clocks if it is expecting a coarse correction transition within ADF4382 correction DAC. This behavior is controlled with the variable allow_adf4382_coarse_adjustment. ADF4382's coarse transition can introduce a timing jump of a few tens of picoseconds that might be too large in some systems. In such cases, the user can allow for a coarse adjustment at appropriate times by allowing a coarse jump with the function **adi_apollo_mcs_cal_coarse_jump_set().**

Stage 4: (Optional) Forced MCS Tracking Calibration,

If the user's system is sensitive to clock adjustment, then the following routine can be followed:

1. MCS Tracking Force Foreground with function **adi_apollo_mcs_cal_force_fg_tracking_run().**
   - Unlike the normal Tracking Foreground, this can be triggered multiple times, irrespective of the freeze state of MCS Tracking.
   - The underlying FW algorithm is the same as that of Normal Tracking Foreground.
2. MCS Track Force Foreground with function **adi_apollo_mcs_cal_force_bg_tracking_run()**.
   - Unlike normal Tracking Background, the Force Background executes one single clock adjustment, if needed, on the next run of MCS Tracking. There are no successive adjustments every second, until a new Force Tracking signal is received by MCS Tracking FW.
3. To disable MCS Tracking, the user can send an abort command with function **adi_apollo_mcs_cal_bg_tracking_abort(),** followed by a disable of the MCS Tracking calibration with function **adi_apollo_mcs_cal_tracking_enable()** with an enable argument of 0.
4. The abort command will instruct MCS Tracking to disable the TDC hardware. The user would need to reinitialize after such command.

SYNCHRONIZATION CATEGORY AND EXAMPLES

In Table 23, the synchronization is categorized as JESD204 subclass 0 and subclass 1. As well known, JESD204 subclass 1 needs to align the internal SYSREF to external SYSREF but subclass 0 does not. For JESD204 subclass 1, as discussed in the above sections, the hardware-based SYSREF alignment provides +/- ½ MCLK cycle of alignment accuracy with less setup and configuration complexity. If the system requires accurate alignment, the TDC and firmware-based SYSREF alignment is the approach to provide more accurate than the hardware approach. In this section several examples with the synchronization procedures are discussed for the users to better understand these synchronization schemes and approaches provided by Apollo device.

Table 23. Synchronization Subclass and Approach Summary

Subclass	Clock Mode	Alignment Accuracy	Synchronization Mode	SYSREF Alignment Mode	mcs_cfg requirements
0	single	N/A	dyn_cfg_sync	N/A	N/A
0	dual A/B not aligned	N/A	dyn_cfg_sync	N/A	N/A
0	Dual A/B aligned	Coarse/fine	dyn_cfg_sync	TDC and Firmware-based	aside_sysref: sysref_present = false
					bside_sysref: sysref_present = false
1	single	coarse	oneshot_sync	Hardware-based	N/A
1	single	coarse/fine	dyn_cfg_sync or trig_sync	TDC and Firmware-based	center_sysref: sysref_present = true
1	dual A/B not aligned	Coarse	oneshot_sync	Hardware-based	N/A
1	Dual A/B aligned	Coarse/fine	dyn_cfg_sync or trig_sync	TDC and Firmware-based	Option 1: aside_sysref: sysref_present = true bside_sysref: sysref_present = true
					Option 2: aside_sysref: sysref_present = true

DEVICE SYNCHRONIZATION

Table 23. Synchronization Subclass and Approach Summary (Continued)

Subclass	Clock Mode	Alignment Accuracy	Synchronization Mode	SYSREF Alignment Mode	mcs_cfg requirements
					bside_sysref: sysref_present = false
					Option 3: aside_sysref: sysref_present = false bside_sysref: sysref_present = true

Subclass 0 Synchronization

For JESD204 subclass 0 which doesn't need SYSREF alignment, the synchronization process flow which is illustrated in Figure 39 is required to follow and it is recommended to program the internal SYSREF period according to the " Guidelines for Setting the Internal SYSREF Period" . The SYSREF alignment portion of phase 0 can be skipped (unless A-side and B-side are required to be aligned in dual clock mode, refer to example Use Case 0: Subclass 0, Dual Clocks, A/B-sides Alignment). Phase 2 of the synchronization process can be bypassed and the JESD link can be enabled directly after phase 1 for subclass 0.

Configuration and API Functions for Subclass 0

The configurations required for subclass 0 operation are described in Table 21 and Table 24. Programming of these parameters in the device profile and calling the API function **adi_apollo_clk_mcs_sync_only_set()** in the bring-up procedure are required to achieve subclass 0 synchronization.

Table 24. "adi_apollo_jesd_common_cfg" Structure Parameter for Subclass

Parameter	Description	Enumeration	Comments
subclass	Sets the JESD204B/C subclass mode	ADI_APOLLO_SUBCLASS_0	Operate in subclass 0 (no deterministic latency)
		ADI_APOLLO_SUBCLASS_1	Operate in subclass 1

Use Case 0: Subclass 0, Dual Clocks, A/B-sides Alignment

In dual clocking mode, it is necessary to align the A- and B-side clocks to ensure the internal channels' alignment. The A- and B-side SYSREF alignment is performed by the TDC in the central MCS block and its two inputs to TDC are the internal SYSREFs from both A side and B side MCS block, the TDC measures the time difference and adjust either A side or B side internal SYSREF so that A side and B side internal SYSREFs are aligned. This alignment process is implemented in the MCS Init Calibration which can be called by API functions **adi_apollo_mcs_cal_config_set()** and **adi_apollo_mcs_cal_init_run().**

Configurations for Use Case 0:

- Device configuration (**adi_ads10_apollo_ex_startup()**) using device profile is set for subclass 0.
- Appropriate clock is provided according to the device profile.
- In the device profile version 9 or later, all the variables of "sysref_present" variables in data structure "mcs_cfg" are set to false.
- If using gaped periodic SYSREF, the TDC decimation rate (**measurement_decimation_rate** in data structure **adi_apollo_mcs_cal_config_t**) should be kept below 32768.

Figure 47 illustrates the synchronization process. As discussed above, the firmware-based SYSREF alignment (MCS Init Calibration) is used which can be initiated by API function **adi_apollo_mcs_cal_init_run()**. Then API function **adi_apollo_mcs_cal_init_status_get()** reads back the calibration status which is included in the data structure **adi_apollo_mcs_cal_data_t**:

- is_*_Locked: if the value is 1, MCS Init is succeeded. If the value is 0, it means the SYSREF alignment is failed, for the failure case, check if the internal SYSREF is configured to the correct frequency or increase the TDC decimation rate setting which provides better noise rejection.
- internal_period_*_femtoseconds: estimate of the internal SYSREF frequency. If the value is not the programmed frequency, it indicates either, a misconfiguration of data structure**adi_apollo_mcs_cal_config_t**, or, an incorrect clock or internal SYSREF frequency.
- diff_*_After_femtoseconds: The final time difference between the A/B-side internal SYSREFs after MCS Init.
- recommended_offset_*_femtoseconds: It represents the time difference between the internal SYSREFs that MCS Init Calibration cannot correct. This offset can be used as offset_*_femtoseconds in later runs of MCS Init Calibration.

When the MCS Init Calibration is completed the alignment of the A and B-side internal SYSREFs to less than ± ½ MCLK period is achieved. Then, the process continues to perform Phase 1 (dyn_cfg_sync) to synchronize the internal blocks of both sides to their internal SYSREFs

DEVICE SYNCHRONIZATION

respectively. Until now, A-side and B-side of the device are aligned if the internal SYSREF period is same as (or longer than) the LMFC/LEMC of JESD204B/C. Otherwise, the Phase 2 (trig_sync) is required if any of the conditions in Phase 2 Overviewis violated.

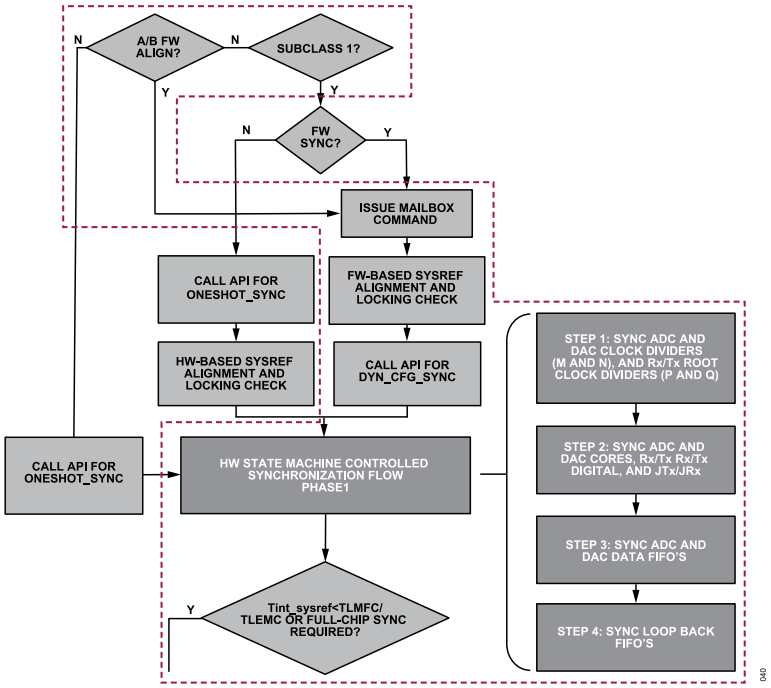

Figure 47. Subclass 0, Dual-clock, Synchronization Process Flow

Subclass 1 Synchronization

Subclass 1 specification requires deterministic latency which utilizes system reference (SYSREF) to synchronize all the timing in the system. Apollo MxFE device takes the external SYSREF inputs and aligns the internal SYSREF with it, then using the internal SYSREF to synchronize the internal timing to provide the deterministic latency. In this sections a few examples are provided to demonstrate the subclass 1 synchronization, and the users can find the examples in the software package in the folder **examples\ads10_apollo_ex_main**. These examples can be followed by any systems which require subclass 1 JESD204B/C operation. These examples include the proper order of the operation and the configurations of the entire Apollo MxFE device, including the appropriate JESD204B/C configurations for deterministic latency across all power cycles. The setting of these parameters forms the basis for multi-chip synchronization as well.

All the examples assume the conditions as,

- Device configuration (**adi_ads10_apollo_ex_startup**()) using device profile is set for subclass 1.
- Appropriate clocking provided according to device profile.
- Calibration of the JRX phase adjustment has been performed and included in the device profile. Refer to Transmit Path Deterministic Latencyfor the details.

Use Case 1: Subclass 1, Single Clock, Hardware-based Alignment

This use case is the example illustrated in the **examples\ads10_apollo_ex_main\fullchip_sc1_dl.c**. In this example the hardware-based SYSREF alignment is applied so it's recommended to follow the timing described in the Figure 40 to ensure that the external SYSREF is correctly sampled by MCLK.

In Figure 48 the high level procedure for this example,

- Configure internal SYSREF which is done by the whole device configuration during the bootup,
- Call **adi_apollo_clk_mcs_subclass_set(device, 1)**to set the subclass 1,
- Call **adi_apollo_clk_mcs_sysref_en_set(device, ADI_APOLLO_ENABLE)** to enable the SYSREF hardware inputs,
- Wait 50 microseconds,

DEVICE SYNCHRONIZATION

- Call **adi_apollo_clk_mcs_oneshot_sync()** to start the hardware-based alignment which as talked in the previous sections includes the phase 1 synchronization.

For multi-chip synchronization across more than one Apollo MxFE device, apply this procedure across all Apollo MxFE devices, and followed by phase 2 synchronization to use trig_sync mode to achieve multi-chip synchronization.

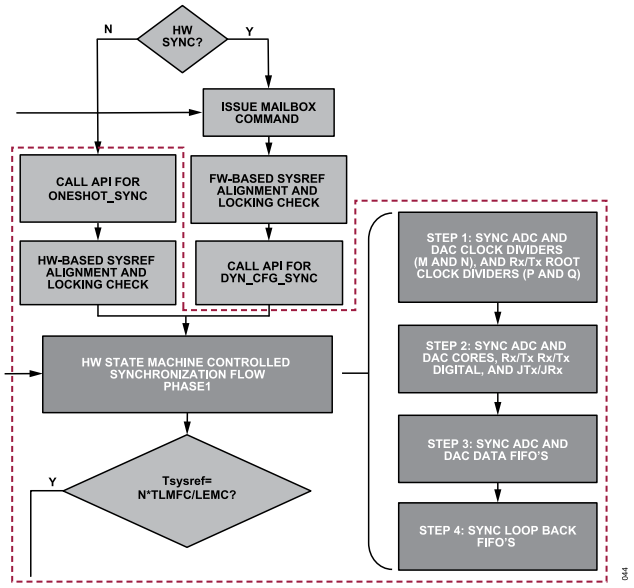

Figure 48. Subclass 1, Single Clock, hardware-based alignment flow

Use Case 2: Subclass 1, Single Clock, Firmware-based Alignment

The example for this use case is located in **examples\ads10_apollo_ex_main\mcs_cal.c**. The basic configuration for this use case,

- In device profile version 9, the **sysref_present** in **center_sysref** structure is set to true, while the **sysref_present** in both **aside_sysref** and **bside_sysref** structure should be set to false.
- As shown in Figure 49, the step of "FW-BASED SYSREF ALIGNMENT AND LOCKING CHECK" is where the MCS Init Calibration is called and executed by the firmware. Once it's done, the Phase 1 alignment using DYN_CFG_SYNC is followed to synchronize the internal timing of Apollo device.
- If using gapped periodic SYSREF, the TDC decimation rate setting should be kept below 32768.

Once MCS Init Calibration is completed, the calibration status can be read back by calling **adi_apollo_mcs_cal_init_status_get( )**, the information of the status structure is shown below,

- is_C_Locked: if the value is 1, it means MCS Init succeeded. If the value is 0, it means the SYSREF alignment has failed. Check if the external and internal SYSREF are configured to the correct frequency, or increase the TDC decimation rate setting (measurement_decimation_rate) to provides better noise rejection.
- internal_period_C_femtoseconds: estimate of the internal SYSREF frequency. If the value is not the programmed period, it may indicate either, a misconfiguration of MCS Init structure **adi_apollo_mcs_cal_config_t**, or, an incorrect clock or external SYSREF frequencies.
- diff_C_After_femtoseconds: The final time difference between the internal and external SYSREFs after MCS Init.
- recommended_offset_C_femtoseconds: It represents the time difference between the external and the internal SYSREF that MCS Init cannot correct. This offset can be used as offset_C_femtoseconds in later runs of MCS Init Calibration.

Only running MCS Init Calibration as discussed in the previous section the alignment accuracy is ± ½ MCLK cycle. In order to track the SYSREF drift over temperature or more accurate alignment, the MCS tracking Calibration is recommended after the MCS Init Calibration. However, it needs to interact with the clock device, as shown in the example, ADF4382 is used to provide the device clock and periodically received the correction strobes from the MCS tracking calibration to correct the clock drift over temperature. The details can be found in the section of TDC and Firmware-based SYSREF Alignment.

DEVICE SYNCHRONIZATION

For multi-chip synchronization across more than one Apollo MxFE device, apply this procedure across all Apollo MxFE devices, and followed by phase 2 synchronization to use trig_sync mode to achieve multi-chip synchronization.

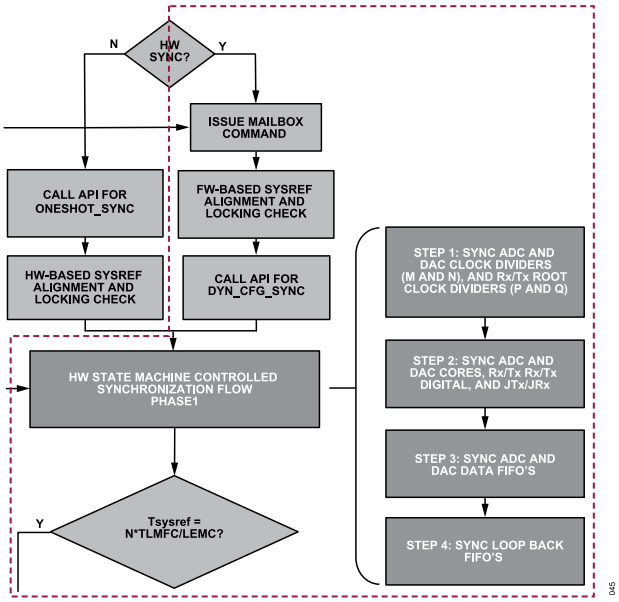

Figure 49. Subclass 1, Single Clock, Firmware-based alignment flow

Use Case 3: Subclass 1, Dual Clock, MCS Init

The programming example for this use case is TBD. In addition to the above pre-conditions, ensure that in device profile version 9, the center_sysref sysref_present is set to false, while both aside_sysref sysref_present and bside_sysref sysref_present are set to reflect which SYSREF receiver is being driven by an external SYSREF. The configuration flow from here is identical to the single clock use case above.

Use Case 4: Subclass 1, Single Clock, MCS with ADF4030 and ADF4382

As discussed in the previous sections, the highest degree of accuracy for deterministic latency and multi-chip synchronization is achieved by using the Apollo MxFE companion clocking devices, the ADF4030 and ADF4382. These clocking solutions are described in the System Clocking Solution section and can achieve deterministic latency and multi-chip synchronization to be within less than 10 picoseconds.

The Apollo MxFE includes a bi-directional SYSREF that can be paired with the ADF4030 clock chip to facilitate the alignment of the internal SYSREF signals of multiple Apollo MxFE devices to be within ± 10 picoseconds. For the stable voltage and temperature conditions, and sufficiently large TDC decimation, the SYSREF alignment precision is greatly improved and achieving a few picoseconds.

The internal SYSREF is used as a reference for all the timing in the Apollo MxFE device. However, the clock distribution inside the device which varies from part to part as well, causes that the sampling instance of the DAC and ADC differing from the internal SYSREF timing. To achieve more precision of synchronization in a system, it is recommended to pair AD9084 and ADF4030 with the ADF4382. As discussed in the previous sections, the MCS Tracking Calibration on Apollo MxFE allows for fine tuning of the remaining internal SYSREF timing mismatch back to the clock source (ADF4382) to achieve less than 1 picoseconds of precision.

High-level Synchronization Flow for Multi-Chip Synchronization Using ADF4030 and ADF4382

An example is provided in **examples\ads10_apollo_ex_main\fullchip_mcs_sc1_dl.c**. In this example, ADF4030 generates the SYSREF for both Apollo MxFE and FPGA, and the time-of-flight (TOF) measurement of SYSREF (BSYNC of ADF4030, refer to example in examples\ads10_apollo_ex_main\bsync_tof.c) is used to align the arrival time of SYSREF to all the Apollo devices and FPGA. After the external SYSREF alignment, the example demonstrates the MCS Init and Tracking calibration to align and maintain the synchronization between the internal SYSREF and the external SYSREF, in the meanwhile as mentioned above, ADF4382 was tuned by the tracking calibration to achieve the most accuracy synchronization. The basic configurations are shown below,

- Configure AD9084 devices as single clock mode.
- One ADF4382 drives each AD9084's center clock receiver.

DEVICE SYNCHRONIZATION

- Each AD9084's GPIO pins 15 and 16 drive it's ADF4382's DELSTR and DELADJ pins respectively.
- The external SYSREF frequency provided by ADF4030 matches that of the AD9084's internal SYSREF frequency.
  - With device profile id00_uc08, it is 9,765,625 Hz.
- External SYSREF is a periodic signal. Namely, it is not missing cycles at random, i.e., gapped periodic.
  - This signal presents at each AD9084's center SYSREF receiver
- HMC7044 provides reference clocking to ADF4030 and FPGA
- HMC7044 and ADF4382 use a shared reference clock (e.g. 125MHz distributed via LTC6955)

The high-level procedure is summarized as,

1. Setup Host and boot up FPGA (ADS10) in the example,
2. Configure clock by calling **adi_ads10_apollo_ex_configure_profile_clks()** to bring up the clock devices ADF4382 and HMC7044 for each set in the system,
3. Bring up each AD9084 with a SC1 (subclass 1) profile by calling **adi_ads10_apollo_ex_startup(),**
   - Note that the above 3 steps can be found in the **examples\ads10_apollo_ex_main\main.c**
4. Configure ADF4030 with **adi_ads10_apollo_ex_adf4030_startup()** to configure the reference frequency,
5. Set BSYNCs as inputs or outputs on ADF4030:
   - BSYNCs going to Apollos and the FPGA are configured as outputs: **adi_ads10_apollo_ex_adf4030_bsync_output_set()**
   - One BSYNC accepts the reference input from HMC7044: **adi_ads10_apollo_ex_adf4030_bsync_input_set()**
   - Align all the outputs to the reference input from HMC7044: **adi_ads10_apollo_ex_adf4030_align_bsync_out()**
6. Ideally, wait for system temperature to stabilize,
7. Call **adi_ads10_apollo_ex_mcs_init_cal_setup()** to configure each ADF4382/AD9084 pair,
8. Call **adi_ads10_apollo_ex_mcs_adf4030_apollo_path_delay_measurement()** to perform BSYNC Time-of-Flight (TOF) with ADF4030 to measure the path delays between ADF4030 and each AD9084 SYSREF input pins, as well as the delay between ADF4030 and the FPGA SYSREF,
9. TOF calibration applies offsets to correct these varying path delays by calling **adi_ads10_apollo_ex_mcs_adf4030_apollo_path_delay_offset()**, so that SYSREF arrives to each AD9084 and FPGA at the same time,
10. Perform MCS Init Calibration for SYSREF alignment by calling **adi_apollo_mcs_cal_init_run(),**
    - This will align the Apollo MxFE Internal SYSREF against the external SYSREF within ± 0.5 high-speed clock cycles.
11. Initialize ADF4382 to do the fine phase adjustment by calling **adi_ads10_apollo_ex_mcs_tracking_cal_setup()** and **adi_adf4382_phase_adjust_auto_align_enable(),**
12. Perform the initial adjustment to ADF4382 using **adi_apollo_mcs_cal_fg_tracking_run()** and optionally enable periodic background phase measurement and adjustment by calling **adi_apollo_mcs_cal_bg_tracking_run(),**
    - This will ensure Apollo MxFE's internal SYSREF tracks its external SYSREF within the desired time precision ranging from tens of picoseconds to hundreds of femtoseconds.
    - Precision is dependent on the time difference between the internal and external SYSREFs, depends on the SYSREF period, and TDC decimation setting.
13. Trig_sync is required for multi-chip synchronization or dynamic reconfiguration which isn't demonstrated in this example codes but the steps below can be followed to continue the Phase 2 (trig_sync) synchronization which is part of the example in **examples\ads10_apollo_ex_main\fullchip_fsrc_sc1_ext_trig.c**.
    - Setup trigger pin mapping and check for Trigger margin with respect to the internal SYSREF's KOW, by calling API **adi_apollo_clk_mcs_sync_trig_map()** and **adi_apollo_clk_mcs_trig_phase_get()**,
    - Configure the MCS Digital to internally generate the **Sync Pulses** to the desired digital sections of the device by calling **adi_apollo_clk_mcs_trig_reset_dsp_enable()** and then calling **adi_apollo_clk_mcs_trig_sync_enable(device, 1)** to enable the trig_sync mode,
    - Proceed to receive the trigger signal from the external trigger pins and then execute the **trig_sync** synchronization.
    - At the end call **adi_apollo_clk_mcs_trig_sync_enable(device, 0)** to clear the trig_sync as it isn't a self-clearing bit.

JESD204B/C INTERFACE FUNCTIONAL OVERVIEW

The device employs serial interfaces that comply to the JESD204C standard for the ADC and DAC paths, including the JESD204B backward compatible option.

The main differences introduced in the JESD204C standard as employed on this device are the additional 64-bit/66-bit encoding scheme, the respective synchronization process (eliminating the need for the SYNCOUTB_xx and SYNCINB_xx pins), and the recommended operating link rates.

If the 8-bit/10-bit link layer option is selected, the link operation complies to both the JESD204B and JESD204C standards and the link lane rates can be up to 20 Gbps. If the 64-bit/66-bit link layer option is selected, the link operation complies to the JESD204C standard, including the new synchronization process (SYNCOUTB_xx and SYNCINB_xx pins not used), and the link lane rates can be up to 28.21 Gbps. The high-level differences between using the 8-bit/10-bit and 64-bit/66-bit link layers are shown in Table 25.

This section of the user guide focuses on the common requirements for the ADC and DAC paths.

DEVICE ARCHITECTURE IMPLICATIONS ON THE JESD204B/C INTERFACE

As described in the Document Nomenclature section, the AD9084 and AD9088 top digital is organized into Side A and Side B, and Core Digital. Figure 50 illustrates that the A and B-sides are two identical and independent sides, each with its own JESD204B/C transmitter and receiver link interfaces that connect to the data path digital. Each of these JESD204B/C interfaces can be configured as single or dual link. So, when implementing a system design with both transmit and receive functions enabled, there will be either two or four JESD204B/C transmitter links (for the receive path) and either two or four JESD204B/C receiver links (for the transmit path). Each single link configuration supports up to 12 lanes while dual link configurations support up to six lanes of serial data.

The two sides of the device can be operated off independent clock and SYSREF signals (dual clocking). Alternatively, both sides of the device can be operated from a single clock and SYSREF signal. The SERDES lanes on both sides of the device are run from a common clock imposing a requirement that lanes rates must be related between the A and B-sides of the device even in a dual clocking configuration. Details for clocking can be found in the Sampling Clock and Distribution Options section. Details for SYSREF can be found in the Device Synchronization section.

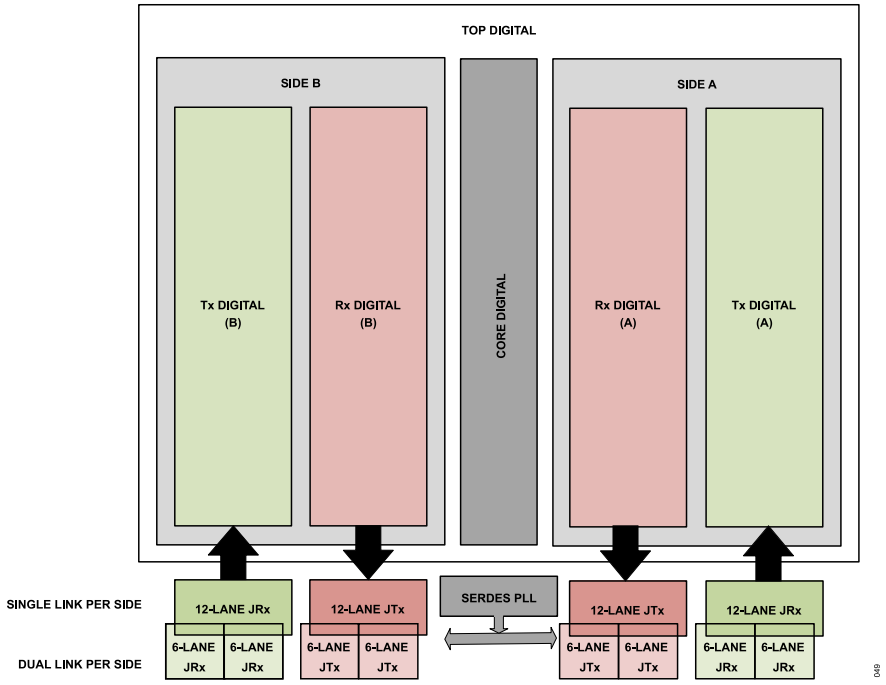

Figure 50. Digital Architecture with JESD204B/C Links Block Diagram

NEW FEATURES IN THE JESD204C STANDARD

The following subsections contain an overview of JESD204C specifications that are new or updated when compared to the those in the JESD204B standard.

JESD204B/C INTERFACE FUNCTIONAL OVERVIEW

Terminology and Parameters

There are new terms and configuration parameters introduced in the JESD204C standard that are used to describe the functions associated with the 64-bit/66-bit link layer (see Table 26). These terms are detailed throughout the document in the context of the JESD204C transmitter and JESD204C receiver physical, link, and transport layers.

Table 25. Differences Between 8-Bit/10-Bit and 64-Bit/66-Bit Link Layer Operations

Function/Attribute	8-bit/10-bit Encoding	64-bit/66-bit Encoding
Payload Delivery Efficiency	80% encoding efficiency	96.97% encoding efficiency
SYNCOUTB_xx and SYNCINB_xx	Yes, from JESD204B receiver to JESD204B transmitter	Not used, entirely feed forward
Link Initialization	Code group synchronization (CGS) + initial lane alignment sequence (ILAS)	Synchronization header alignment, multiblock alignment, and extended multiblock alignment achieved using embedded synchronization header stream
Scrambling	Optional (recommended)	Required
Error Monitoring	8-bit/10-bit disparity, not in table (NIT), and unexpected K-characters (UEKC) errors are detected	Cyclic redundancy check (CRC) checks per multiblock of data (2048 bits), Sync Header, Multiblock Header, and Extended multiblock header error tracking
Deterministic Latency and Multichip Sync	Aligned to local multiframe clock (LMFC)	Aligned to a local extended multiblock clock (LEMC)
Lane Rate	5 Gbps ≤ lane rate ≤ 20 Gbps	1 Gbps ≤ lane rate ≤ 28.21 Gbps

Table 26. New Terms and Parameters Defined in JESD204C

Term	Definition
Block	A structure starting with a 2-bit synchronization header containing 66 bits
BkW	Block width, the number of bits in a block (always 66 bits for the device).
cmd	Command, as related to the command channel.
Command Channel	Data stream using extra bandwidth afforded from synchronization headers.
E	The number of multiblocks in an extended multiblock.
EMB_LOCK	A state that asserts extended multiblock alignment is achieved.
EoEMB	End of extended multiblock identifier bit (Bit 22 of the synchronization word).
EoMB	End of multiblock sequence (00001), decoded from the synchronization header stream.
Extended Multiblock	A set of data containing one or more multiblocks.
LEMC	Local extended multiblock clock.
Multiblock	A set of data containing 32 blocks.
PCS	Physical coding sublayer.
SH_LOCK	A state that asserts synchronization header alignment is achieved.
Synchronization Header (SH)	Two bits that guarantee a transition precede every block.
Synchronization Transition Bit	Decoded synchronization header (2b'10 = 0, 2b'01 = 1).
Synchronization Word	32 synchronization transition bits from a multiblock.

Physical Layer Updates

The JESD204C physical layer specification and the implementation on the device supports operation with the 8-bit/10-bit (JESD204B) and 64-bit/66-bit (JESD204C) link layers.

JESD204C introduces data interface classes and defines two categories of classes, Category B and Category C. There are three classes defined for each category. Table 27 lists the JESD204C lane rates associated with each category (not necessarily the Apollo MxFE Specification). For Category C, there are three subclasses defined to minimize link power dissipation for a variety of channel types: C-S (short), C-M (medium) and C-R (reflective). Each class is a superset of the previous class. Table 28 lists the architectural differences between the classes. The device implements a class C-M interface on both the ADC and DAC paths, although the lane rate is limited to 1 Gbps on the low end (when employing 8-bit/10-bit encoding) and 28.21 Gbps on the high end (when employing 64-bit/66-bit encoding).

JESD204B/C INTERFACE FUNCTIONAL OVERVIEW

Table 27. Lane Data Rates for Data Interface Classes

Data Interface Class	Minimum Data Rate(Gbps)	Maximum Data Rate(Gbps)
B-3	0.3125	3.125
B-6	0.3125	6.375
B-12	6.375	12.5
Category C	6.375	32.45

Table 28. JESD204C 32 Gbps Interface Device Class Features

Class	Relative Power	Tx FFE (dB)	Rx CTLE (dB)	Rx DFE (No. of Taps)
C-S	Low	9.5	6	0
C-M	Medium	9.5	9	3
C-R	High	9.5	12	14

JESD204C Class C compliance is assessed by calculating a figure of merit called a channel operating margin (COM). Most EDA tools support COM measurements.

A JESD204C compliant link has a COM higher than 2dB. Incorporating channel compliance using COM affords greater flexibility in managing the trade-offs of channel impairments and SERDES architectural complexity.

Transport and Link Layer

The transport layer provides mapping between converter samples and octets. The 8-bit/10-bit and the 64-bit/66-bit link layers use the same octet format and there is no difference in the transport layer that depends on the encoding scheme.

The only difference between using the two encoding schemes is that the octets sent to the 64-bit/66-bit link layer must be scrambled. For 8B/10B, scrambling is optional.

When using the 8-bit/10-bit link layer option of JESD204C, the device is fully compatible with the JESD204B specification and all that the specification implies. These implications include the use of K.28 characters for CGS, ILAS, and character replacement as well as the SYNCOUTB_xx and SYNCINB_xx pins used to initiate synchronization and report errors from the receiver back to the transmitter.

When operating with 64-bit/66-bit encoding, the use of the SYNCOUTB_xx and SYNCINB_xx pins is eliminated and there is no compatibility with JESD204B. There is no encoding of the octets. The octets are packed into a 64-bit block of data. The entire block is then scrambled and has a 2-bit synchronization header appended. This format is shown in Figure 51, where D[0:7] represents the eight data octets, S[0:7] represents the scrambled octets, and SH is the 2-bit synchronization header.

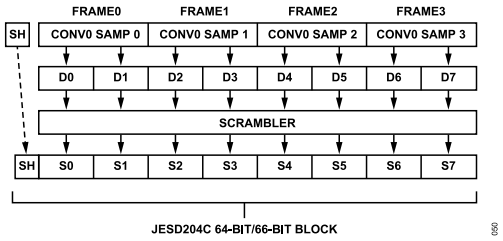

Figure 51. 64-Bit/66-Bit Block Format Example for LMFS = 1.1.2.1, N = N' = 16

The synchronization header is a 2-bit, unscrambled value at the beginning of each block. The header contents are interpreted to decode a single synchronization transition bit. The synchronization header bits must be either a 0 to 1 sequence to indicate a Logic 1 or a 1 to 0 sequence to indicate a Logic 0.

Table 29 shows the synchronization header and synchronization transition bit values.

Table 29. Synchronization Header Bit Values

Sync Header Bits[0:1]	Sync Transition Bit Value
00	Invalid
01	1
10	0

JESD204B/C INTERFACE FUNCTIONAL OVERVIEW

Table 29. Synchronization Header Bit Values (Continued)

Sync Header Bits[0:1]	Sync Transition Bit Value
11	Invalid

Multiblocks (MB) and Extended Multiblocks (EMB)

There are 32 blocks in a JESD204C multiblock. The 32 synchronization transition bits in each multiblock make up a 32-bit synchronization word. The functions within the synchronization word are described in the Synchronization Word section. An extended multiblock is a container of E multiblocks and must contain an integer number of frames. When a multiblock does not contain an integer number of frames, E must be >1. The format of the multiblock and extended multiblock are shown in Figure 52.

The JESD204C standard supports a multiblock that is either 2112 (66 32-bit blocks) bits or 2560 (80 32-bit blocks) bits, depending on which 64-bit encoding scheme is used. A multiblock in the AD9084 and AD9088 is always 2112 bits because 64b/80b encoding is not supported. For most implementations and configurations, an extended multiblock is one multiblock.

The E parameter is introduced in JESD204C and determines the number of multiblocks in the extended multiblock. The default value for E is 1. E must be >1 for configurations where the number of octets in the frame (F) is not a power of two and is typically associated with modes where NP = 12.

This requirement ensures that the extended multiblock boundary coincides with a frame boundary.

The equation for E is given as the following:

$$E = (K \times F)/256 \quad (2)$$

when 256 mod F does not equal to 0

E must be an integer and the number of frames in a multiframe (K) must be set appropriately.

LEMC is the local extended multiblock counter and is similar to the LMFC in the 8-bit/10-bit link layer. The SYSREFN and SYSREFP input signal aligns all LEMCs in a system and the LEMC boundaries are used to determine synchronization and lane alignment.

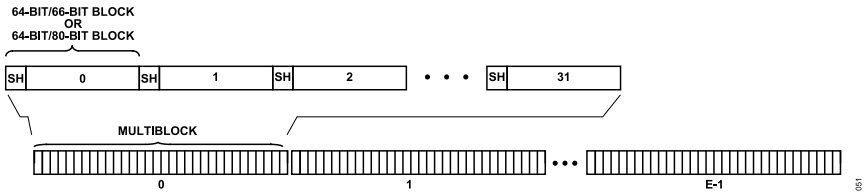

Figure 52. JESD204C Multiblock and Extended Multiblock Format

Synchronization Word

The 32-bit synchronization word is constructed from each of the sample headers from the 32 blocks within the multiblock where Bit 0 is transmitted first. The synchronization word is used to enable lane synchronization, error detection, and deterministic latency.

There are seven bits (CMD[6:0]) that provide a command channel for the transmitter to communicate to the receiver. However, this command channel is not supported on the AD9084 and AD9088 and these bits are always zeros for the device.

Table 30 describes the different synchronization word fields and functions.

Table 30. Synchronization Word Fields and Functions

Sync Word Bit	Field Name	Function
0	CRC11	Bits[11:9] of the 12-bit CRC check applicable to the previous multiblock.
1	CRC10	
2	CRC9	
3	1	Always 1.
4	CRC8	Bits[8:6] of the 12-bit CRC check applicable to the previous multiblock.
5	CRC7	

JESD204B/C INTERFACE FUNCTIONAL OVERVIEW

Table 30. Synchronization Word Fields and Functions (Continued)

Sync Word Bit	Field Name	Function
6	CRC6	
7	1	Always 1.
8	CRC5	Bits[5:3] of the 12-bit CRC check applicable to the previous multiblock.
9	CRC4	
10	CRC3	
11	1	Always 1.
12	CRC2	Bits[2:0] of the 12-bit CRC check applicable to the previous multiblock.
13	CRC1	
14	CRC0	
15	1	Always 1.
16	Cmd6	Bits[7:5] of the 7-bit command channel (not supported, always 0).
17	Cmd5	
18	Cmd4	
19	1	Always 1.
20	Cmd3	Bit 3 of the 7-bit command channel.
21	1	Always 1.
22	EoEMB	End of extended multiblock bit.
23	1	Always 1.
24	Cmd2	Bits[2:0] of the 7-bit command channel.
25	Cmd1	
26	Cmd0	
27	0	End of multiblock pilot signal.
28	0	
29	0	
30	0	
31	1	

CRC-12 Encoder

The CRC-12 encoder in the JESD204C transmitter takes in the 2048 scrambled data bits of each multiblock and computes 12 parity bits. These parity bits are transmitted to the receiver during the subsequent multiblock.

The receiver computes 12 parity bits from each multiblock of data received. The 12 bits are compared to the bits that were received over the command channel. If the parity bits do not match, there is at least one error in the received data. See the 64-Bit/66-Bit Error Monitoring and Resynchronization section for details.

Forward Error Correction

This feature will be documented on a future revision of this document.

8-BIT/10-BIT LINK ESTABLISHMENT OVERVIEW

When using the 8-bit/10-bit link layer, the link establishment process follows the protocol established in the original JESD204B/C standard (and subsequent versions). Using K28 characters and the SYNC~ signals, the link first establishes CGS, then frame synchronization (FS) and ILAS prior to transmitting sample data in the user data phase.

During the user data phase, character replacement (inserting K28.x characters) is used to monitor frame and multiframe alignment while an error checking circuit in the JESD204B/C receiver monitors incoming data for 8-bit/10-bit errors (running disparity, NIT, UEKC). Details are not provided because this protocol is well established. For more details, refer to the Analog Devices webcast on the JESD204B data link layer here.

JESD204B/C INTERFACE FUNCTIONAL OVERVIEW

64-BIT/66-BIT LINK ESTABLISHMENT OVERVIEW

The link establishment process when using the 64-bit/66-bit link layer starts automatically when the link is powered on. The SYNC~ signal, or synchronization request, is not required. The process begins with synchronization header synchronizations, then progresses to extended multiblock synchronization, and then to extended multiblock alignment.

SERDES PLL AND CONFIGURATION

Because the JESD204B/C receiver and transmitter share the SERDES PLL, consider the following during PLL configuration. The JESD204B/C receiver and transmitter modes must be selected such that the corresponding lane rates remain equal or that the lane rate of the JESD204B/C transmitter and receiver have a power of 2 relationship between them. If using different lane rates, the JESD204B/C transmitter and receiver lane rate adaptation functions will be needed. (see the JESD204B/C Transmitter Lane Rate Adaptation and JESD204B/C Receiver Lane Rate Adaptation sections for details).

SERDES PLL Configuration API

Configuration of the SERDES PLL is performed by a combination of on-chip firmware and the **adi_apollo_serdes_pll_config()** API function. Both firmware and software use the appropriate parameters from the device profile after the **adi_apollo_cfg_data_path()** API function is called as part of the Apollo MxFE Bring-up Procedure. The parameters for the SERDES PLL are contained in the **adi_apollo_serdes_pll_cfg_t** structure and are described in Table 31. Other parameters used for SERDES PLL configuration are dev_clk_freq_kHz and serdes_clk_div which are part of the adi_apollo_clk_cfg_t structure.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 31. SERDES PLL Parameters

Parameter	Description	Datatypes	Valid Range
loop_bandwidth_hz	PLL Loop bandwidth	Integer in Hz	1kHz - 10 MHz (1000 – 10,000,000)
phase_margin	PLL Phase Margin	Integer in degrees	0 - 80
div_range	SERDES only Div/1	Boolean	1= div/1 from root divider enabled
div2	SERDES only Div/2	Boolean	1= div/2 from root divider enabled
power	PLL Power setting	ADI_APOLLO_SERDES_PLL_POWER_DOWN ADI_APOLLO_SERDES_PLL_POWER_UP	0 = PLL Power Down 1 = PLL Power Up
ref_clk_div	PLL reference clock divider (/R)	Integer	0 - 31 Default is 2
i_bleed_en	PLL bleed ramp	Integer	0 = PLL Bleed Disable 1 = PLL Bleed Enable
serdes_pll_odiv	SERDES output divider value	Integer	0 - 63 Default is 33 Set based on use case
feedback_int	Integer portion of feedback factor	Integer	8 - Output frequency/50MHz
feedback_frac	Fractional portion of feedback factor	Integer	Always 0

APOLLO MXFE RECEIVER

The high-level block diagram for the Apollo MxFE receiver of the AD9084 and AD9088 are illustrated in Figure 53 and Figure 54. Note that the AD9088 has 2 additional ADCs per side and does not have the Fractional Sample Rate Converter functions. The functional overview and programming details for each of the sub-blocks are contained in the following sub-sections.

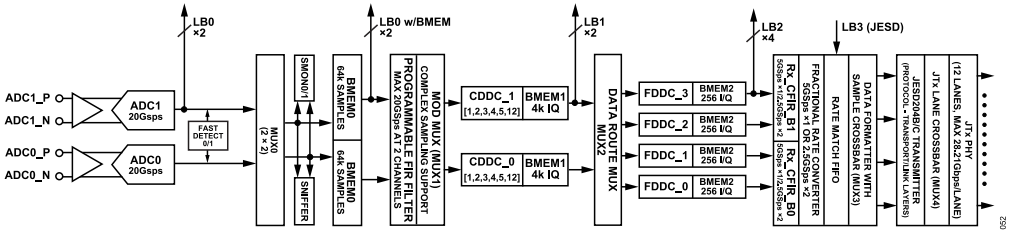

Figure 53. AD9084 Receive Path Block Diagram (per side)

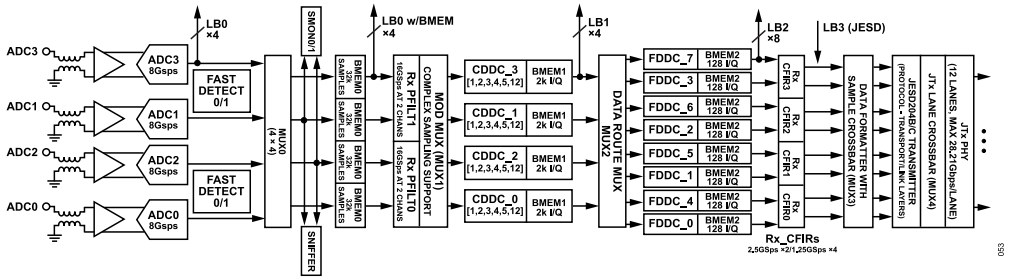

Figure 54. AD9088 Receive Path Block Diagram (per side)

ADC ARCHITECTURE AND CALIBRATION

The AD9084 consists of *four* ADCs supporting sample rates from 8 GSPS to 20 GSPS while the AD9088 consists of eight ADCs supporting sample rates from 4 GSPS to 8 GSPS. Referring to Figure 55 and Figure 56, the ADCs feature a track-and-hold amplifier (THA) with some variants including an on-chip balun to simplify the interface to single-ended signal sources. To achieve higher sample rate, AD9084's ADC Channel consists of two interleaved ADC cores sampling at 180-degree offsets while the lower sample rate AD9088's ADC Channel consists of a single ADC core. A clock generator is used to provide precise timing for each sub-block while including synchronization capability to ensure a repeatable pipeline delay for the ADC. A digital backend providing a 12-bit output is used to re-assemble the outputs from each sub-ADC while including foreground and background calibration for enhanced spurious and noise performance. For fast automatic gain control (AGC) against ADC input overvoltage, a Fast Detect, FDx signal can be routed to an external pin. See the Receive AGC Assist Functions section for more details on the Fast Detect function.

The ADC core itself is an interleaved architecture consisting of four sub-ADCs driven by an 18 GHz input bandwidth THA. The THA provides uniform sampling of the input signal during track mode and a steady output during hold mode for the four sub-ADCs. Uniform sampling prior to digitization by the four sub-ADCs significantly mitigates any interleaving images that would otherwise exist from timing mismatches between these sub-ADCs operating out-of-phase. However, timing calibration for the AD9084 remains critical to suppress the interleaving image since it employs two THA's. The AD9084 ADC cores can be operated in sequential or random mode up to an interleaved rate of 20 GSPS. For the AD9088, the ADC cores are run in random mode.

In sequential mode, the sub-ADCs operate at ¼ the THA sample rate with each sub-ADC operating 90º out-of-phase relative to each other. The order of digitization is fixed starting with sub-ADC0 and ending with sub-ADC3 before repeating itself again. In this mode, low-level, signal-independent spurs appear at ¼ the THA sample rate due to the residual dc offsets associated with each sub-ADC. Surrounding these spurs is narrowband colored noise attributed to the 1/f flicker noise associated with the interleaved sub-ADCs. Additionally, residual gain and timing errors between the two THA and sub-ADCs will appear at F_{ADC} /8 ± F_{IN} spurs, where F_{ADC} is the Interleaved ADC sampling rate and F_{IN} is the fundamental frequency of an input sinusoid. Note that background calibration is used to calibrate out dc offsets, gain, and timing mismatches between the THA and sub-ADC's. In random mode, the order of digitization is pseudo-random consisting of four sub-ADC's operating in the range of ⅓ to ¼ the THA sample rate and 120º to 90º out-of-phase relative to each other. By randomizing the selection order of the sub-ADC's, any residual gain and offset mismatch errors between sub-ADC's (after background calibration) as well as the flicker noise is spread into the noise floor. Figure 58 and Figure 59 show the difference in spur levels when operating the AD9084 at 16 GSPS operation.

APOLLO MXFE RECEIVER

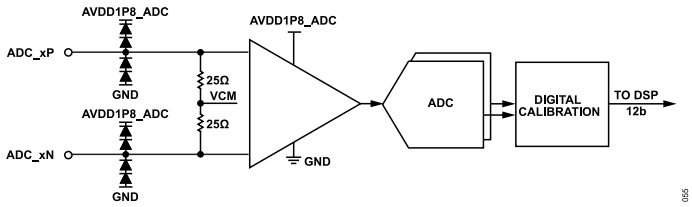

Figure 55. Differential ADC Architecture of AD9084

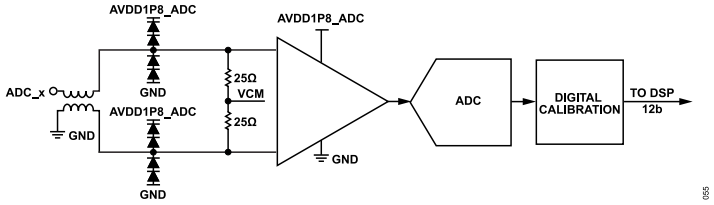

Figure 56. Single-Ended ADC Architecture of AD9084 and AD9088

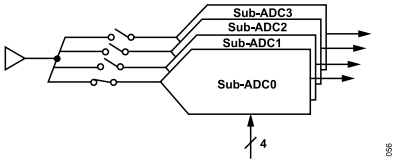

Figure 57. AD9084's Single Core ADCs consists of 4-sub-ADCs sampling at either at 90º or 120º relative to each other when operating in sequential or random mode respectively

The THA consists of a differential input buffer with a –3dB bandwidth of 18Ghz, a sampling network and an amplifier to drive the sub-ADCs. The input buffer isolates the signal source from the sampling network's charge kick-back while providing a nominal 50 ohm differential input impedance with a dc common-mode voltage of 0.9 V. The sampling network tracks the buffered input signal and then holds it for the duration of track time required by the 1st stage of the selected sub-ADC before switching back to the tracking mode for the next selected sub-ADC. The final amplifier isolates the sampling network from the sub-ADCs while driving the designated sub-ADCs during its track period. The AD9088's THA consists of a single sampling network to support 8 GSPS operation while the AD9084's THA consists of two out-of-phase sampling networks in ping-pong operation to support 20 GSPS sample rate. In the latter case, residual image spur falls at $F_{ADC}/2 - F_{IN}$ with its magnitude-dependent on the timing mismatch error between the two THAs as well as gain variation between the two ADC cores.

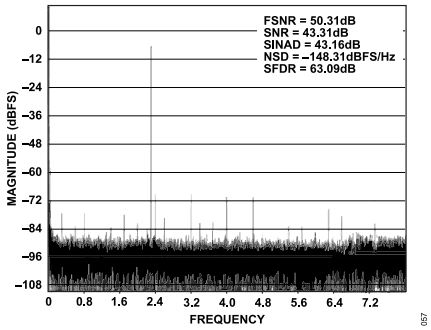

Figure 58. FFT of AD9084 operating at 16 GSPS in sequential mode with –7 dBFS input at 2.3 GHz. Note full-scale SNR-dBFS is 50.3 with residual interleaving artifacts appearing as spurs.

APOLLO MXFE RECEIVER

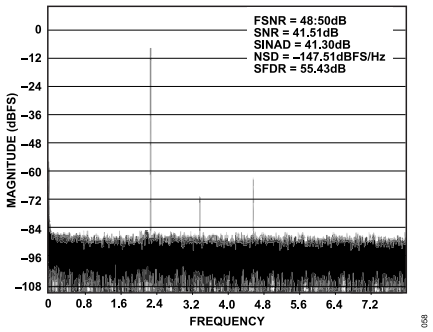

Figure 59. Same conditions as Figure 58 but with AD9084 operating at 16 GSPS in random mode. Note full-scale SNR-dBFS reduces to 48.5 dBFS as interleaving spurs are spread into noise floor

Sampling Mode Switch

The Apollo MxFE ADC sampling mode is configured with the device profile. In the device profile version 9, the configuration variable, called adc_slice_mode, is located within the adc_config structure. adc_slice_mode is instantiated twice, one for side A and the other for side B, and it is an array of four integers corresponding to the four ADC Channels per side of the AD9088. Recall that AD9084 only has two ADC channels per side. Table 32 shown below describes the possible values that adc_slice_mode can take.

Table 32. Details of ADC Slice Mode Enumeration, adi_apollo_adc_slc_mode_e

Name	Description
ADI_APOLLO_ADC_SLICE_PPRAND3P1 = 6	Random sub-ADC sampling order
ADI_APOLLO_ADC_SLICE_PPSEQ4 = 7	Sequential sub-ADC sampling order

The device profile defines the desired initial sampling mode of the sub-ADCs. Randomized sampling provides improved spurious-free dynamic range (SFDR) with a minor increase in noise floor, while sequential sampling provides optimum noise floor with a small detriment in SFDR. In some cases, it might be beneficial to switch between sampling modes while the ADC is running to take full advantage of the benefits provided by both sampling modes. Three mode switching methods are provided,

Sampling Mode Switch by Command, this method allows for seamless transition between random to sequential sampling mode and vice versa without disrupting the normal execution flow of the Apollo MxFE Calibrations. API sends a command to Apollo MxFE and the firmware executes the switch whenever it is most appropriate with respect of the enabled background calibrations.

Sampling Mode Switch by Register Map, this method allows for faster execution of the sampling mode switch. This is achieved by commanding Apollo MxFE to serve the sampling mode switch request as soon as possible. This means that the normal execution of enabled background calibrations may be halted immediately while the request is handled.

Sampling Mode Switch by GPIO, this way allows for the fastest execution time. This is achieved by allowing the host FPGA to toggle one of the Apollo MxFE GPIOs to trigger the mode switch, bypassing the need of using SPI to communicate with Apollo MxFE. Internally, Apollo MxFE will handle the request as if Sampling Mode Switch by Register map was issued. Once the mode switch is configured as by GPIO, to trigger the switch, the host FPGA must toggle the appropriate GPIO.

To configure the sampling mode switch, the user can follow the example code located in \examples\ads10_apollo_ex_main\ rx_adc_ms.c. In all cases, API will command the ADC to prepare and load appropriate calibration coefficients based on the target sampling mode, trigger the sampling mode switch, and restore ADC operation.

1. Enables ADC Slice Mode switch by calling API **adi_apollo_adc_slice_mode_switch_enable_set()**,
   - Note, this function must be called before loading device profile to Apollo MxFE. It is done within examples\ads10_apollo_ex_main\main.c
2. Configure Apollo MxFE GPIOs for **Mode Switch by GPIO** by calling **adi_apollo_gpio_cmos_func_mode_set()** to select **ADI_APOLLO_FUNC_ADC_MODE_SWITCH**,
   - Note, this step isn't necessary for the other two switch methods,
3. Configure the BMEM for ADC data capture by calling adi_apollo_bmem_hsdin_capture_config()
4. Run and enable calibrations by calling adi_ads10_apollo_ex_run_cals()
5. Check the status of the ADC calibration by calling adi_ads10_apollo_ex_inspect_adc_all()

APOLLO MXFE RECEIVER

6. Capture ADC data by calling adi_ads10_apollo_ex_bmem_capture()
7. Prepare the Apollo MxFE ADC for sampling mode switch by calling API **adi_apollo_adc_mode_switch_prepare()**
   - Internally, this API gracefully freezes ADC Calibrations.
8. Execute Mode Switch by calling API **adi_apollo_adc_mode_switch_execute()**,
   - For **Mode Switch by Command**, select the enumeration value to be "**ADI_APOLLO_ADC_MODE_SWITCH_BY_COMMAND**",
   - For **Mode Switch by Register Map**, select the enumeration value to be "**ADI_APOLLO_ADC_MODE_SWITCH_BY_REGMAP**",
   - For **Mode Switch by GPIO**, toggle the GPIO assigned in step 2 from the host FPGA side,
9. Restores Apollo MxFE ADC operation by calling API **adi_apollo_adc_mode_switch_restore()**
   - Internally, this API function gracefully re-enables ADC background calibration if the argument "enable_bg_cal" is set to true.

ADC Calibration and Specifying Nyquist Zone

ADC calibration is used to reduce spurious artifacts common among interleaving ADC architectures due to sub-ADC timing, gain, and offsets mismatches. On Apollo MxFE the ADC calibration includes initial calibration (or called as foreground calibration) and background calibration which usually runs during the real operation and deals with the variations in the system.

ADC calibration depends on the knowledge of the Nyquist zone being odd or even, which depends on the ADC input frequency (F_{IN}) and sample rate (F_{ADC}), as defined in the following equation and Figure 60:

$$\text{Nyquist Zone} = \text{FLOOR}(F_{IN}/(F_{ADC}/2)) + 1 \quad (3)$$

To configure the Nyquist zone of a given ADC, call the API **adi_apollo_adc_nyquist_zone_set()**. To read the Nyquist zone of a given ADC, call the API **adi_apollo_adc_nyquist_zone_get()**. The parameters for these functions are described in Table 33. Without specifying the Nyquist zone, the default is the first Nyquist zone.

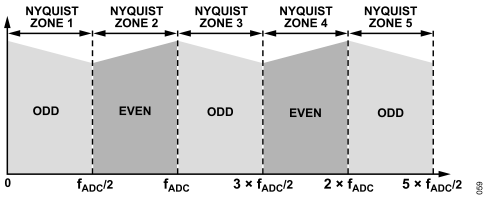

Figure 60. Relationship Between Nyquist Zone Number vs. Odd or Even

Table 33. Nyquist Zone Parameters

Parameter	Description	Enumerations or Settings	Comment
adcs	target ADC for the Nyquist_zone setting	ADI_APOLLO_ADC_NONE	0x0, No ADC
		ADI_APOLLO_ADC_A0	0x1, ADC0 of SIDE A
		ADI_APOLLO_ADC_A1	0x2, ADC1 of SIDE A
		ADI_APOLLO_ADC_A2	0x4, ADC2 of SIDE A
		ADI_APOLLO_ADC_A3	0x8, ADC3 of SIDE A
		ADI_APOLLO_ADC_B0	0x10, ADC0 of SIDE B
		ADI_APOLLO_ADC_B1	0x20, ADC1 of SIDE B
		ADI_APOLLO_ADC_B2	0x40, ADC2 of SIDE B
		ADI_APOLLO_ADC_B3	0x80, ADC3 of SIDE B
		ADI_APOLLO_ADC_ALL	0xFF, ALL ADCs
		ADI_APOLLO_ADC_A_ALL	0x0F, ALL ADCs of SIDE A
		ADI_APOLLO_ADC_B_ALL	0xF0, ALL ADCs of SIDE B
nyquist_zone	Selects Nyquist zone the target ADC will operate in	1	first Nyquist zone
		2	second Nyquist zone

Types of Foreground Calibration

APOLLO MXFE RECEIVER

The Apollo MxFE ADCs are factory-trimmed for specific clock rates such as 20 GSPS. During this trimming, the ADC calibrations are executed and allowed to converge for an input signal in the first Nyquist zone. Then, the calibration coefficients are saved in the non-volatile memory (NVM). These calibration coefficients provide good initial ADC performance. However, since differences in the operating conditions exist between factory test and the real system board in which the device resides, the foreground calibration at the user specified Nyquist zone is required again during the device initialization process.

Additionally, the Apollo MxFE software infrastructure allows the user to extract the optimal coefficients to be stored and re-loaded at the users' discretion. This allows users to load good and known coefficients as well as to skip the calibration execution and the settling time associated with the foreground and background calibrations. It is possible to configure the foreground calibration mode before loading the device profile to the Apollo device. Table 34 shows the available options when executing ADC foreground calibration together with a description of each.

Table 34. Supported Foreground Calibration Options

Name	Value	Description
ADI_APOLLO_INIT_CAL_DISABLED	0	No ADC Foreground calibration algorithms to be executed.Start ADC with Apollo MxFE uncalibrated calibration data. Note, this calibration data is not the data that was saved in NVM during device trimming. Please note, although in this mode the ADC foreground calibration is bypassed, the Nyquist zone is reset to the first zone.
ADI_APOLLO_INIT_CAL_DISABLED_WARMBOOT_FROM_NVM	1	No ADC Foreground calibration algorithms to be executed in this mode. Calibration coefficients stored in NVM will be used. Please note, although in this mode the ADC foreground calibration is bypassed, the Nyquist zone is reset to the first zone.
ADI_APOLLO_INIT_CAL_DISABLED_WARMBOOT_FROM_USER	2	No ADC Foreground calibration algorithms to be executed. Calibration coefficients provided by the user will be loaded to the hardware. In other words, the user is required to run an appropriate foreground calibration mode followed by the ADC background calibration. Then, read the calibration coefficients and store the data elsewhere for being used in this mode later on. How to read and reload calibration coefficients is discussed below. Please note, although in this mode the ADC foreground calibration is bypassed, the Nyquist zone is reset to the first zone.
ADI_APOLLO_INIT_CAL_ENABLED	3	ADC Foreground calibration algorithms will be executed without any pre-calibrated data. In this mode, the Nyquist zone is reset to the first zone, so present a CW tone with -7 dBFS power level near the center of the first Nyquist zone. If the desired band is at the second Nyquist zone, the modes 4 or 5 are recommended.
ADI_APOLLO_INIT_CAL_ENABLED_WARMBOOT_FROM_NVM	4	ADC Foreground calibration algorithms will be executed with the NVM calibration data. In this mode, the calibration is executed in the specified Nyquist zone. Calling API adi_apollo_adc_nyquist_zone_set() before this calibration. Present a CW tone with -7 dBFS power level near the center of the specified Nyquist zone.
ADI_APOLLO_INIT_CAL_ENABLED_WARMBOOT_FROM_USER	5	ADC Foreground calibration algorithms will be executed with the User-Defined calibration data. In this mode, the calibration is executed in the specified Nyquist zone. Calling API adi_apollo_adc_nyquist_zone_set() before this calibration. Present a CW tone with -7 dBFS power level near the center of the specified Nyquist zone.

The recommended foreground calibration is mode 2 (ADI_APOLLO_INIT_CAL_DISABLED_WARMBOOT_FROM_USER) as it provides optimum performance with minimum execution time. In this mode, the ADC foreground calibration algorithm is bypassed. Instead, the calibration coefficients provided by the user will be used to provide a decent ADC initial performance. In other words, the user is required to run an appropriate foreground mode, such as ADI_APOLLO_INIT_CAL_ENABLED_WARMBOOT_FROM_NVM, followed by the ADC background calibration. After the calibrations are settled, read back the ADC calibration coefficients and store the data elsewhere for later usage in mode 2. The ADC background calibration post ADI_APOLLO_INIT_CAL_DISABLED_WARMBOOT_FROM_USER is recommended as well to account for system drift with respect to the state at which the user-provided calibration data was read.

Reading and Loading ADC Calibration Data

APOLLO MXFE RECEIVER

The user can extract the current ADC Calibration data for later usage. For example, if the user knows the final operating state of Apollo MxFE after system bring up, the specific calibration data to such state can be stored and loaded later on instead of using the trimmed NVM data. The ADC Calibration data is stored in the Apollo MxFE memory, and it can be read by the user. An example procedure is provided in the following steps:

- Execute ADC Foreground calibration on the selected ADC Channels. By default, it runs on ADC Channels A0, A1, B0, and B1.
- Call API **adi_apollo_adc_cal()** which starts ADC foreground calibration with the data from NVM.
- Logs Apollo MxFE Temperature post ADC Foreground calibration.
- Enables ADC background calibration by calling API **adi_apollo_adc_bgcal_unfreeze()**.
- Waits 10 seconds for calibration adaptations.
- Waits for Apollo MxFE to reach a user-selectable target temperature with a default tolerance of 1 degree. This is to ensure that the ADC calibration data is read for the desired target temperature.
- If the desired temperature is reached, reads back the ADC calibration data by calling API **adi_ads10_apollo_ex_adc_cal_data_dump_to_file()**.
  - Note that this API function will freeze ADC calibrations if they are running before reading the calibration data.

This file can be stored in the system memory for the later usage. When it's required, after the system boot up and waits for Apollo MxFE to reach to a user-defined temperature, calling,

- API **adi_ads10_apollo_ex_adc_cal_data_reload_from_file()** to load the calibration coefficients from the file to the device.
  - Note that this API function will freeze ADC calibrations if they are running before loading the new data.
- Then, after loading the data, call API **adi_apollo_adc_init_cal()** to run foreground calibration with ADI_APOLLO_INIT_CAL_DISABLED_WARMBOOT_FROM_USER.

Background Calibration

The ADC Background calibration is intended to be enabled post execution of the foreground calibration. The background calibration allows for a good ADC performance to be maintained in the system. This implies that the ADC calibration coefficients will be adapted to the varying system conditions. This is particularly important on systems that execute fast startup of Apollo MxFE in a cold condition, as the Apollo MxFE temperature can take seconds or minutes to settle to a steady temperature state depending on the user's thermal solution. In other words, the calibration coefficients of a given ADC channel are unique to a particular moment in time and may not hold for other temperatures, voltages, ADC channels, Apollo MxFE devices, therefore, the background calibration is recommended to be enabled during the system operation.

The Procedure to run the Foreground and Background Calibration

An example API code that exercises the ADC calibration procedure can be found in \examples\ads10_apollo_ex_main\rx_adc_nz.c. The example code executes the following steps:

1. Configures Rx BMEM Captures by calling adi_apollo_bmem_hsdin_capture_config().
2. Writes Nyquist Zone by calling API **adi_apollo_adc_nyquist_zone_set()**.
3. Reads Nyquist Zones by calling API **adi_apollo_adc_nyquist_zone_get()**
   - Checks that the Nyquist zone readback is correct.
4. Calls adi_ads10_apollo_ex_run_cals() to execute:
   - Runs Clock Conditioning by calling API adi_ads10_apollo_ex_clock_condition_cal().
     - Note, this must be done before executing any other type of calibration.
   - Runs ADC Foreground Calibration by calling API adi_apollo_adc_cal()
     - The user is recommended to execute all types of foreground (initialization) calibrations before enabling any type of background calibration.
     - If the user would like to run ADC foreground calibration again, ADC background calibrations must first be disabled by freezing them with API function **adi_apollo_adc_bgcal_freeze()**
   - Enables ADC Background Calibration by calling API **adi_apollo_adc_bgcal_unfreeze()**.
   - Execute oneshot_sync

5. Captures ADC samples with Rx BMEM, and saves the ADC output samples into a file with API adi_ads10_apollo_ex_bmem_capture()

Calibration Considerations

As discussed in the previous section, the foreground calibration requires a CW tone with -7 dBFS power presenting to ADC input at nearly the center of the desired Nyquist zone, to allow the calibration to quickly converge to a decent performance. Likewise, some background calibrations require an input signal to be present at the ADC's input to run effectively. These calibrations are referred to as input-dependent calibrations.

The effectiveness of these calibrations depends on the characteristics of the input signal. Certain input signal characteristics can interfere with the operation of input-dependent calibrations. To prevent adverse calibration behavior caused by undesirable input characteristics, the statistics of the digitized input signal are monitored. If the statistics of the input signal fail to meet certain criteria, the input-dependent calibrations are paused automatically until the input signal statistics become acceptable again. There are several cases as discussed below,

- **Over-ranged Signal**, if the ADC input is overloaded, the data-path samples get corrupted and the calibration algorithms can no longer converge. A hardware detector between the ADC output and the digital calibration block monitors the power of the data-path. If the monitored power is higher than the over-range threshold, all the background calibrations are automatically paused. The details are discussed in Over-range (OVR) Calibration Gating.
- **Low-power Signal**, if the input signal does not have enough energy (input power < -20dBFS), the interleaving gain, offset and timing mismatch calibrations are paused until higher input signal power (input power > -20dBFS) is detected at the ADC input.
- **Diversity of the Input Amplitude**, the diversity of the amplitude of the input signal is monitored in the background. If the diversity of the amplitudes of the samples falls below a predetermined threshold, the interleaving calibrations pause until the amplitude diversity of the input samples becomes rich again. For example, a DC signal or a two-level signal would halt the interleaving calibrations.
- **Single Tone at f_{ADC}/N**, background calibrations are automatically paused when CWs at f_{ADC}/N, with N = 1, 2, 3, ..., are detected.

Despite the safeguards mentioned above, if an input tone falls on top of the architecturally dictated interleaving mismatch spurs of the ADC, the corresponding interleaving calibrations will react to the presence of such tone and the convergence accuracy will be impacted. For example, the interleaving gain and timing mismatch calibrations are adversely affected if there is a two-tone input (or any pair of multiple tones) where one of the tones falls on top of the two-way interleaving spur location of the other spur (like, f1 + f2= $f_{ADC}/2$). This restriction only applies to AD9084.

To prevent performance degradation while processing such input signals, all input-dependent calibrations can be disabled or interleaving gain and timing calibrations can be disabled.

Moreover, under certain conditions, the users may choose to apply their gain, offset or timing mismatch calibrations. In such cases, the ADC background calibrations which correct for the same errors can be disabled to prevent any interference between such calibrations.

The API function **adi_apollo_adc_bg_cal_grp_gate_set()** is provided to disable one or multiple calibrations. Before execution of this function, it is required that the calibrations of the target ADC are halted by using the **adi_apollo_adc_bgcal_freeze()** function. The difference between both API functions is that **adi_apollo_adc_bg_cal_grp_gate_set()** requires the argument cal_gates to allow the user to disable the partial calibrations. The values of cal_gates are defined inside the enumeration adi_apollo_adc_cal_group_gate_e. The enumerations are described in Table 35.

Table 35. adi_apollo_adc_cal_group_gate_e enum

Enumeration	Target ADC Calibration
ADI_APOLLO_ADC_CAL_GROUP_ALL_INP_DEP	All input-dependent calibrations
ADI_APOLLO_ADC_CAL_GROUP_IL_GAIN_0	A sub-group of interleaving gain calibration
ADI_APOLLO_ADC_CAL_GROUP_IL_GAIN_1	All interleaving gain calibrations
ADI_APOLLO_ADC_CAL_GROUP_IL_OFFSET	Interleaving offset calibration
ADI_APOLLO_ADC_CAL_GROUP_IL_TIMING	Interleaving timing calibration
ADI_APOLLO_ADC_CAL_GROUP_FE_NLE	Front-End non-linearity calibrations

The user is free to select the group of calibrations to disable based on the target application. For the cases where the ADC can be exposed to signals of unknown characteristics, it is recommended to disable all input-dependent calibrations as shown below,

- adi_apollo_adc_bg_cal_grp_gate_set(device, ADI_APOLLO_ADC_A0, ADI_APOLLO_ADC_CAL_GROUP_ALL_INP_DEP).

APOLLO MXFE RECEIVER

ADC Fast Detect Configuration

The Apollo MxFE ADC provides means to configure signal detection for fast automatic gain control (AGC) at the system level. For a description and API functions, see the section Receive AGC Assist Functions.

Relevant API Functions

Table 36 contains the descriptions of API functions that are used to configure and interact with ADC calibrations. Table 37 provides the API functions to switch the sub-ADC switching mode between the random mode and sequential mode.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 36. API Functions for ADC Calibrations

API function	Description
adi_apollo_adc_init_cal(adi_apollo_device_t *device, adi_apollo_blk_sel_t adcs, adi_apollo_init_cal_cfg_e init_cal_cfg)	Execute an ADC init (foreground) calibration with config option.
adi_apollo_adc_cal(adi_apollo_device_t *device, adi_apollo_blk_sel_t adcs)	Execute an ADC init (foreground) calibration with either ADI_APOLLO_INIT_CAL_ENABLED_WARMBOOT_FROM_NVM, or ADI_APOLLO_INIT_CAL_ENABLED.
adi_apollo_adc_bgcal_freeze(adi_apollo_device_t* device, adi_apollo_blk_sel_t adcs)	This function is used to halt execution of all ADC calibrations at the Apollo MxFE level for one of more ADCs. It is important to note that theApollo MxFE will not attempt to run ADC calibrations regardless of the calibration configuration of each individual ADC. After this function iscalled, the target ADC(s) will stop tracking of environmental changes such as voltage and temperature. If the input signal drastically changescarrier frequency, the spurious performance may degrade slightly.
adi_apollo_adc_bgcal_unfreeze(adi_apollo_device_t* device, adi_apollo_blk_sel_t adcs)	This function is used to restart the periodic execution of ADC calibrations for one or more ADC at the Apollo MxFE level. This means thatspecific configurations for a given ADC will be considered, and the target ADC(s) will continue to track environmental and signal changes.
adi_apollo_adc_bg_cal_grp_gate_set(adi_apollo_device_t *device, adi_apollo_blk_sel_t adcs, uint32_t cal_gates)	This function is used to configure the subset of ADC calibrations that run for one, and only one specific ADC channel at a time. Beforeexecution of this function, it is required that the calibrations of the target ADC are halted by using the adi_apollo_adc_bgcal_freeze() function.
adi_apollo_adc_bg_cal_grp_gate_get(adi_apollo_device_t *device, adi_apollo_blk_sel_t adc, uint32_t *cal_gates)	This function reads which calibration groups are disabled for one and only one ADC channel at a time. Before execution of thisfunction, it is required that the calibrations of the target ADC are halted by using the adi_apollo_adc_bgcal_freeze() function. Unlikeadi_apollo_adc_bg_cal_grp_gate_set(), cal_gates is a pointer used to return the status of the calibration groups. The user is required to parsethe returning 32bits and map them to the corresponding adi_apollo_adc_cal_group_gate_e enum.
adi_apollo_adc_nyquist_zone_set(adi_apollo_device_t *device, adi_apollo_blk_sel_t adcs, uint32_t nyquist_zone)	Configure the ADC Nyquist zone where the calibrations are executed.
adi_apollo_adc_nyquist_zone_get(adi_apollo_device_t *device, adi_apollo_blk_sel_t adc, uint32_t *nyquist_zone)	Read back the configured Nyquist zone.
adi_apollo_adc_status_get(adi_apollo_device_t *device, adi_apollo_adc_status_t *status)	Retrieve ADC Calibration status information. An API example code that shows its utilization of this API function. It is located in the file \examples\ads10_apollo_ex_common\src\adi_ads10_apollo_ex_inspect.c inside function adi_ads10_apollo_ex_inspect_adc_all().

Table 37. API Functions for Sub-ADC Sampling Mode Switch

API Functions	Descriptions
adi_apollo_adc_mode_switch_enable_set(adi_apollo_device_t *device, uint8_t enable)	enable: set to 1 to enable slow switch mode; Default value is 0 on device power up. Call this function before loading the device profile.
adi_apollo_adc_mode_switch_prepare(adi_apollo_device_t *device, adi_apollo_blk_sel_t adcs)	Perform setup sequence for the ADC Mode switch for the selected ADCs.
adi_apollo_adc_mode_switch_execute(adi_apollo_device_t *device, adi_apollo_blk_sel_t adcs, adi_apollo_adc_mode_switch_method_e method)	Execute ADC Slice Mode Switch for selected ADCs. For fast switch, this API is replaced by GPIO toggle with GPIO func mode

APOLLO MXFE RECEIVER

Table 37. API Functions for Sub-ADC Sampling Mode Switch (Continued)

API Functions	Descriptions
	ADI_APOLLO_FUNC_ADC_MODE_SWITCH. An example is shown in the section Sampling Mode Switch.
adi_apollo_adc_mode_switch_restore(adi_apollo_device_t *device, adi_apollo_blk_sel_t adcs, uint8_t enable_bgcal)	Re-enables ADC background calibration if the argument “enable_bg_cal” is set to true.

ADC INPUT INTERFACE DESIGN

Input Interface Considerations for ADCs Having Differential Inputs

Optimum ac linearity performance is achieved when driving the ADC input differentially since this reduces the amount of even order harmonics. The reduction in even order harmonics depends on the amplitude and phase balance over the frequency passband of interest. The maximum specified input drive level into any of the ADC inputs is limited to 10 dBm which corresponds to 2 Vpk-pk at the input. Note, this max drivel level corresponds to a differential input signal which is typically the case but also applies to any single-ended input signal where the other differential input is not driven.

For applications that prefer to use a fixed RF gain block (such as the HMC8411 or ADL8121) with single-ended output, an RF balun is used to provide a differential signal to the ADC input as shown in Figure 61. Note the AD9084-DF EVB uses the Marki BAL-0020SLG since it provides excellent amplitude and phase balance up to 20 GHz. If the signal level of the RF gain block can exceed 10 dBm plus the insertion loss of the balun (during a fault condition), it is recommended that a limiter such as the Marki DLM-10SM be included in the signal path with its compression point adjusted by applying a DC level applied to $V_{CONTROL}$. The DC level will depend on the P_{SAT} of the RF gain block and how much attenuation (or limiting) is required at the frequency range of interest so that 10 dBm level into the ADCs input is not exceeded. Normally a DC value between 0 V to 0.5 V should be adequate. A simple RC voltage divider with filter can be used to generate the clean DC bias. Lastly, an LTCC low or bandpass filter from Mini-Circuits (not shown) after the RF gain block can also be included to provide alias rejection while filtering out the harmonics of the RF Amp.

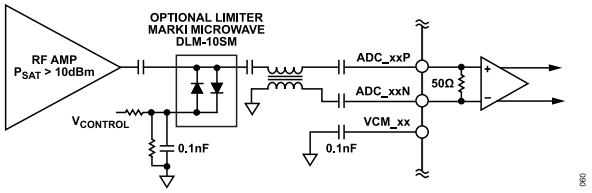

Figure 61. AC coupled Balun interface

Applications requiring a receiver variable gain amplifier (RxVGA) may consider the ADL6332 as shown in Figure 62 which can support a frequency range of 0.4 to 12 GHz while providing a seamless interface to the ADC input with fast gain reduction for ADC overload protection. In the circuit shown in Figure 62, the fast detect signal whose threshold is programmable is sent to the ADL6332's ATTSEL0 pin to reduce the gain by a programmable value within 1 usec.

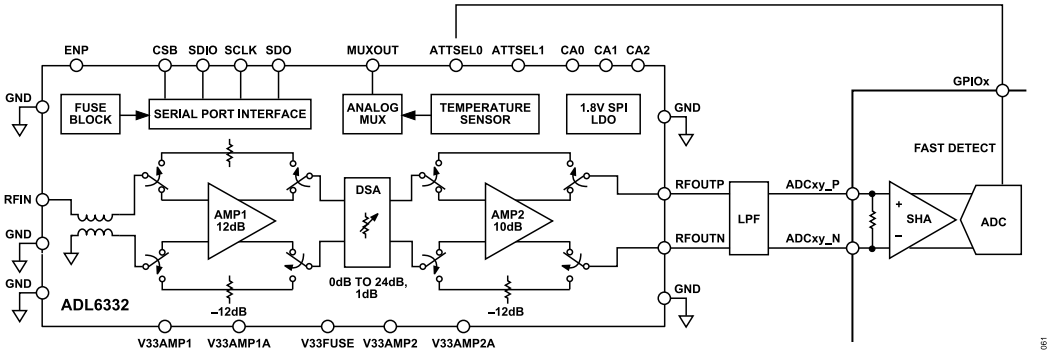

Figure 62. ADL6332 RxVGA to ADC interface for AD9084-DF

APOLLO MXFE RECEIVER

Input Interface Considerations for ADCs Having Single-ended Inputs

An on-chip balun is included with the AD9084-SE and AD9088; hence, a single-ended driver is most suitable as shown in Figure 63. The circuit is the same as what is shown in Figure 61 with the exception that it does not include the external balun but includes an LTCC antialiasing filter.

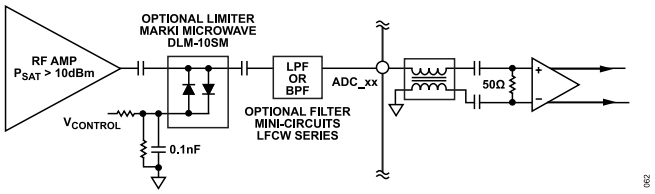

Figure 63. AC coupled interface to the AD9084-SE and AD9088

Simulating the ADCs AC Frequency Response With External Interface Circuitry

It is highly advisable to use simulation tools when optimizing the overall Rx frequency response to meet a target applications requirement. For this reason, a simulation package containing S parameter used to create an equivalent ac analysis models of the ADC inputs will be available prior to the product release. Included in the simulation package will be balun models from various vendors. Designers are encouraged to simulate with these models combined with models of the selected RF components and PCB layout models. Figure 64 and Figure 65 show the return loss for the single-ended and differential ADCs for the AD9088 and AD9084.

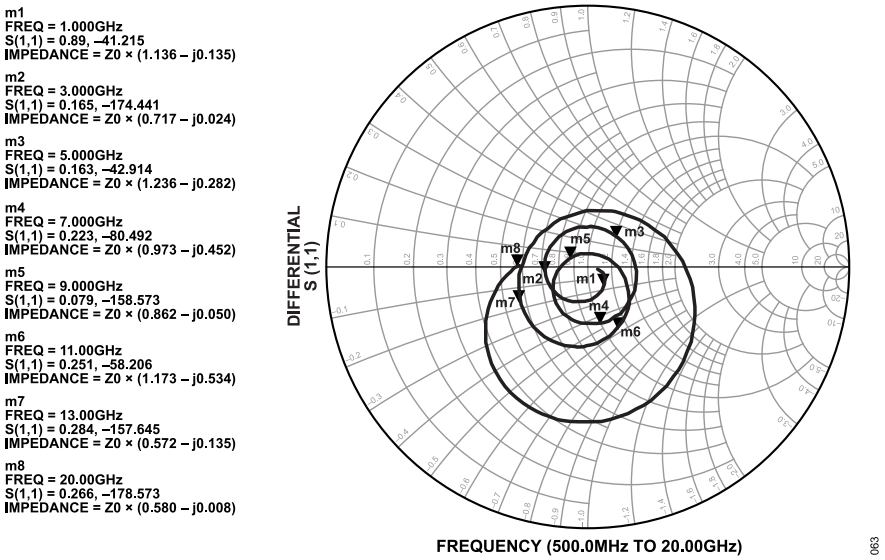

Figure 64. AD9088 ADC' Single-ended Return Loss (S11)

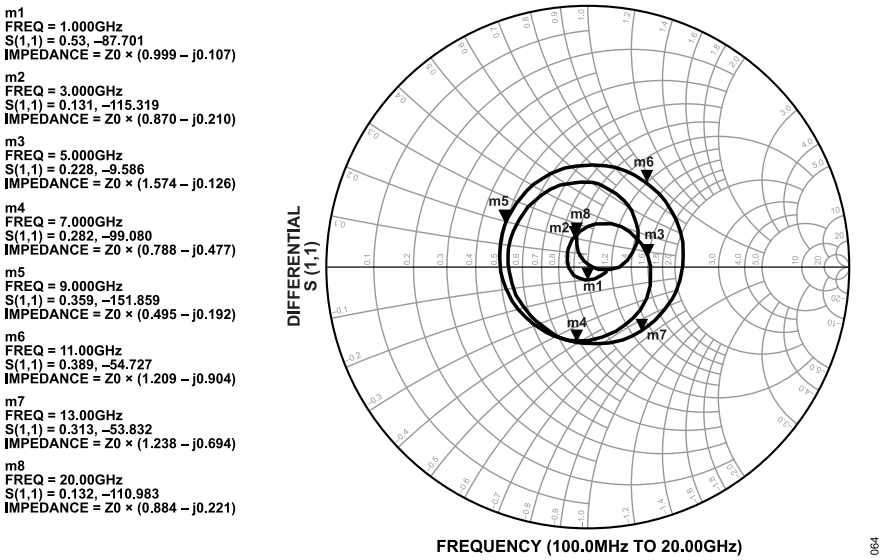

Figure 65. AD9084 ADCs Differential Return Loss (S11dd) for no balun product variant

RECEIVE DIGITAL DATA PATH OVERVIEW

AD9084 product variants share the same receive digital data path regardless of whether the product variant ADC input is single-ended 50ohm or differential 50ohm (with the noted exceptions highlighted in Table 1. It supports up to max 20Gsps sampling rate. While AD9088 supports up to 8Gsps sampling rate.

AD9084 receive digital data path (Side A or Side B) shown in Figure 53 consists of PFILT, CNCO, CDDC filters, FNCO, FDDC filters, FSRC and CFIR.

AD9088 receive digital data path (Side A or Side B) shown in Figure 54 consists of PFILT, CNCO, CDDC filters, FNCO, FDDC filters and CFIR.

The AD9084 has 4 coarse digital down-conversion (CDDC) blocks with 2 CDDCs per B-side/A-side. The AD9088 has 8 coarse digital down-conversion (CDDC) blocks with 4 CDDCs per B-side/A-side.

Each CDDC block consists of an optional, digital quadrature down-converter with a 32-bit Integer & 32-bit Modulus NCO followed by a decimation filter supporting factors of 2, 3, 4, 6, and 12.

The AD9084 has 8 fine digital down-conversion (FDDC) blocks (4 per side) that offer additional digital down-conversion capacity with a decimation filter supporting factors of 2, 4, 8, 16, 32, and 64. The AD9088 has 16 fine digital down-conversion (FDDC) blocks (8 per side) that offer additional digital down-conversion capacity with a decimation filter supporting factors of 2, 4, 8, 16, 32, and 64.

A data router multiplexer is used to select the desired data outputs (at different stages of the receive data path) that are aggregated for the JESD204B/C link.

The AD9084 and AD9088 FDDC provides additional digital down-conversion stages and higher decimation factors. The additional down-conversion stages enable multiband applications where two or more smaller RF bands can be down-converted separately and represented at the lowest possible sample rate to reduce the overall throughput and post digital signal processing requirements of the host processor.

A 2×4 crossbar multiplexer per B-side and A-side of AD9084 allows up to 4 FDDC blocks to be mapped to the output of a CDDC block with the maximum complex data interface rate limited to 5000 MSPS.

A 4×8 crossbar multiplexer per B-side and A-side of AD9088 allows up to 8 FDDC blocks to be mapped to the output of a CDDC block with the maximum complex data interface rate limited to 2500MSPS.

The B-side/A-side receive data path offers considerable flexibility and the following auxiliary features to simplify system design:

- Support for two independent JESD204B/C transmitter links that each have different receive data path configurations. Note that since the JESD204B/C links all share the same PLL, all of the link lane rates must be related by a power of 2.
- Fast detection at the ADC output with high and low programable thresholds to facilitate external AGC implementations.

APOLLO MXFE RECEIVER

- Signal Monitoring (SMON) and spectrum sniffer monitoring capability.
- Programmable digital filter (PFILT) that allows the user to provide customized digital filtering and/or equalization directly to the wideband signal content represented at the ADC(s) output(s).
- Complex programmable digital filter (CFIR) that allows the user to provide customized digital filtering and/or equalization to the baseband signal content represented at the CDDC/FDDC(s) output(s) up to 5GSPS rate.
- Programable integer delay per ADC output to compensate for any RF channel delay mismatch.
- Optional up sampler to enable different sample rates among receive data paths that share the same JESD204B/C link.
- Fractional sample rate converter(FSRC) provides the flexibility of converting to the exact desired output data rate.
- Optional 6 dB gain enhancement when down-converting a real signal. 6 dB gain can also be used for complex signals with low input gain.
- CNCO/FNCO FFH with up to 16/32 preassigned hop frequencies.

Receive Data Path Configuration Considerations

Proper selection of the CDDC factor (M_{RX}) and FDDC factor (N_{RX}) for each receive data path must be considered, as well as whether an additional JESD204B/C link is required when data path requirements may differ based on the usage case. An optimal configuration is one where all desired RF signal channels are supported while operating at the lowest possible data rate. Lowering the data rate into the host processor reduces any post digital filtering and the number of JESD204B/C lanes required to transfer the data.

The simplest case occurs when all receive data paths can be configured with the same total decimation factor with matching values for M_{RX} and N_{RX} to require one JESD204B/C link with all output data rates from the receive data paths matched. This scenario is often the case in single-band communication applications that consist of multiple antennas with the target RF bands common among all paths. A more complicated case can occur if an additional RF band needs to be supported with different bandwidth requirements, which requires a different decimation factor that supports the data rate of the added RF band. In this case, the output data rates from the receive data paths may not match and require the use of either an additional JESD204B/C link or the up-sampler shown in Figure 73 to match the lower rate receive data paths to the highest rate data path such that one JESD204B/C link can still be used. Using the up-sampler to match data rates requires that all data rates in the JESD204B/C link be related by any factor in the form of 2^N.

A more complicated case can arise in a communication application where one (or more) of the receive data paths associated with an ADC must be quickly repurposed with a different configuration, which results in a different output data rate. In this case the JESD204B/C link rate should be chosen to reflect the widest bandwidth (lowest decimation) mode that will be used. Narrower bandwidth modes can use the up-sampler to maintain the JESD link rate and keep the link running without interruption.

The receive digital data path contains two separate stages of decimation following the complex down-conversion NCO in the CDDC and FDDC data paths. The usable bandwidth after the decimation filters is 80% of the IQ data output rate (f_{IQ_OUT}). The M_{RX} can be set to 1, 2, 3, 4, 6, or 12 in the COARSE_DEC_SEL parameter and the N_{RX}, can be set to 1, 2, 4, 8, 16, 32 or 64 in the FINE_DEC_SEL parameter. Each channelizer can also be bypassed in the FINE_BYPASS parameter to set N_{RX} to 1. These parameters are described in Table 42. The following equation shows the total decimation factor as a function of $M_{RX} \times N_{RX}$ as well as the ADC rate, f_{ADC} , and f_{IQ_OUT} (with both values represented in either MSPS or GSPS)

$$Total\ Decimation = f_{ADC}/f_{IQ\_OUT}$$

$$Total\ Decimation = M_{RX} \times N_{RX}$$

The up-sampler requires knowledge of the total decimation factor for each receive data path as well as the lowest decimation factor of all data paths assigned to a JESD204B/C link. The total decimation factor for each receive data path is calculated automatically.

Selecting the optimum M_{RX} and N_{RX} values for each receive data path depends on the following factors and trade-offs:

- For the complex data f_{IQ_OUT} required to represent the signal bandwidth of interest, this data rate must exceed the bandwidth by at least a factor of 1.25 to allow the signal to fall within the digital filter's bandwidth. For instance, if the desired signal bandwidth was 100 MHz, the minimum f_{IQ_OUT} should be no less than 125 MSPS.
- The optimum f_{ADC} results in the desired RF band(s) of interest to meet the required spurious content because of ADC performance limitations that manifest as harmonics or digital induced image spurs. Ensure that these band(s) fall well within a Nyquist zone, which requires some degree of frequency planning to determine a suitable range of operation within the maximum specified range. Large signal bandwidths (including multiband support) typically benefit from operating with higher ADC clock rates. In practice, a suitable ADC clock rate within 75% to 100 % of the maximum ADC clock rate often exists.

APOLLO MXFE RECEIVER

- Recognition of whether the application requires multiband support and if so, whether the spacing between RF bands (with consideration to the signal bandwidths) would benefit from using the channelizer FDDC capability to create two or more separate receive data paths. For instance, consider the dual band case that consists of one 75 MHz band centered at 1.7475 GHz and another 70 MHz band centered at 2.535 GHz. Both bands can be represented with a f_{IQ_OUT} of 92.16 MSPS if each band is assigned an individual channelizer path, which results in two separate receive data paths. If a single receive data path is used to process both bands simultaneously (occupying an edge-to-edge bandwidth of 860 MHz), the f_{IQ_OUT} must exceed 1057 MSPS, which is a 5.7 factor increase in data throughput rate that must be supported by the JESD204B/C link as well as host processor.
- Compare the ratio of the edge-to-edge bandwidth required to support all bands of interest to the sum of all RF band occupied bandwidths to indicate whether to use channelizers. If this ratio exceeds 2, consider using the FDDCs.
- Recognition that the maximum data rate of AD9084 and AD9088 to FDDC(s) is limited to 5000 MSPS/2500MSPS places restrictions on the minimum value of M_{RX} when using the channelizers such that $M_{RX} > f_{ADC}/5000$ or $f_{ADC}/2500$ where f_{ADC} is specified in MHz. The usable bandwidth is limited to 4000 MHz/2000 MHz because of the digital filter passband response. Multiband applications where the edge-to-edge bandwidth exceeds this limit require an additional coarse digital down-converter (CDDC) stage to process one of the outer bands.
- To benefit from the up-sampler ability to enable the use of one JESD204B/C link for receive data paths with different f_{IQ_OUT} rates, keep all data rates related by a factor of 2^N.

Receive Data Path Configuration API

Configuration of the Receive digital data path is performed automatically by Apollo MxFE during boot using the appropriate parameters from the device profile and initiated by the **adi_apollo_cfg_data_path()** API function as shown in the Figure 12 in the SOFTWARE OVERVIEW section. The parameters associated with configuring the receive data path are automatically generated and inserted into the device profile by the Device Profile generator tool that is included with ADI's ACE evaluation software for Apollo MxFE. The parameters associated with configuring the receive data path are described in each of the receive path functional block subsections that follow. There are numerous examples of how the API is used to configure the receive data path. Among those are the **fullchip_*.c** and others in the examples\ ads10_apollo_ex_main folder. See the ADI Evaluation System Software Examples section for more examples and how those can be implemented.

Configuration changes should be accomplished by creation of a new profile unless Dynamic Reconfiguration is required by the end application. The Dynamic Reconfiguration section explains which functional blocks and their parameters can be dynamically reconfigured without bringing the JESD204 link down. The Configuration Updates After Device Bring-Up section summarizes what functions can be configured after start-up and how, as well as those functions that are only configured at device bring-up.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

RECEIVE MUX0

Mux0 is a crossbar multiplexer that takes the form of either a 2×2 input to output crossbar for the AD9084 (with two ADCs per Side B/ Side A) or a 4×4 input to output crossbar for the AD9088 (with four ADCs per Side B/ Side A). Mux1 allows the mapping of any ADC physical output to any PFILT input of AD9084/AD9088. If the PFILT filter is bypassed, the Mux1 outputs become the inputs to the following stage. These multiplexer outputs are referred to as logical ADC outputs with the suffix _x added to distinguish these outputs from the physical data output directly from the ADC. Figure 66 shows the input to output relationship for the 2×2 and 4×4 Mux1 crossbars usage cases.

The logical ADC outputs can be mapped to any physical ADC data input. Figure 66. show the mapping settings for the 2×2 and 4×4 multiplexers. Two or more logical ADCs can be mapped to the same physical ADC. Mapping the same logical numeric ADC to the physical counterpart is typically sufficient.

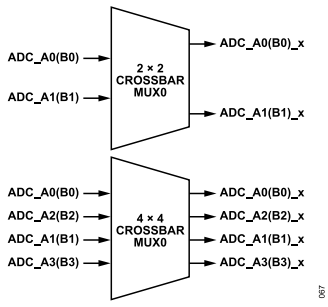

Figure 66. 2×2 and 4×4 Crossbar Mux0 for AD9084 and AD9088

APOLLO MXFE RECEIVER

Receive MUX0 Configuration API

The API library fully supports configuration of Mux0. Mux0 is configured using the **adi_apollo_rxmux_xbar1_set()** API function. The function is used with the API example codes examples\ads10_apollo_ex_main\rx_adc_bmem.c and \examples\ads10_apollo_ex_main\rx_adc_pave.c within their respective rx_mux1_config() functions. The Mux0 (rxmux_xbar1) specific parameters and the possible values set by this API function are listed in Table 38.

Table 38. Mux 0 Configuration parameters

Parameter	Description	Enumerations
side_sel	Selects which of the Apollo sides to configure	ADI_APOLLO_SIDE_NONE = 0x0, ADI_APOLLO_SIDE_A = 0x1 ADI_APOLLO_SIDE_B = 0x2 ADI_APOLLO_SIDE_ALL = 0x3
cbout_from_adc_sel	List of selection settings. Each setting determines which ADC is seen at the crossbar output	ADI_APOLLO_4T4R_CB_OUT_0_FROM_ADC0 = 0 ADI_APOLLO_4T4R_CB_OUT_0_FROM_ADC1 = 1 ADI_APOLLO_4T4R_CB_OUT_1_FROM_ADC1 = 0 ADI_APOLLO_4T4R_CB_OUT_1_FROM_ADC0 = 1 ADI_APOLLO_8T8R_CB_OUT_FROM_ADC0 = 0 ADI_APOLLO_8T8R_CB_OUT_FROM_ADC1 = 2 ADI_APOLLO_8T8R_CB_OUT_FROM_ADC2 = 1 ADI_APOLLO_8T8R_CB_OUT_FROM_ADC3 = 3
length	Defines how many configuration settings there are within **cbout_from_adc_sel**	ADI_APOLLO_RX_MUX0_NUM_4T4R = 2, ADI_APOLLO_RX_MUX0_NUM_8T8R = 4

DSP FUNCTIONS (@ ADC SAMPLE RATE)

The Mux1 block outputs can be directed to either the Mux1 block input or to the full-bandwidth digital signal processing (DSP) block. The full-bandwidth DSP functions reside prior to any receive data path decimation stages and consist of the receive AGC assist block for fast detect and signal monitoring, a spectrum sniffer, and a programmable FIR filter (PFILT) with an integer delay function.

Note that this user guide and some API functions will sometimes refer to an “HSDIN” block. The HSDIN block encapsulates the sample data buffers, sample crossbar (Mux1), and functions performed on the sample – fast detect, signal monitor and the spectrum sniffer.

Receive AGC Functions

External Automatic Gain Control (AGC) circuitry can utilize signals from the Signal Monitor (SMON) and Fast Detect (FD) circuits on AD9084 and AD9088. The signals from SMON and FD are available on GPIO pins or can be inserted into the JESD204 data stream. More information on SMON and FD are found in the Signal Monitor (SMON) and Fast Detect (FD)subsections in the More Auxiliary Features section.

Buffer Memory (BMEM)

The Apollo device features a random-access memory located at different places of the receive digital data path. One of the operational modes of this memory is that of capturing and storing data acting as a buffer, which the user can read for off-chip processing. The BMEM also allows for introduction of data path in-situ programmable delays which can adjust the latency through the system. Moreover, the BMEM also allows for its Buffer’s data to be injected as signal in the data path. Namely, the BMEM can act as an arbitrary waveform generator. Refer to the Buffer Memory (BMEM) section for details.

Spectrum Sniffer

The Spectrum Sniffer provides quick full bandwidth pre-DSP signal information for advance analysis and response to signals appearing in the digitized spectrum. Please see the Spectrum Sniffer subsection of the Advanced Digital Features For System Applications section for more information on the Spectrum Sniffer.

APOLLO MXFE RECEIVER

Receive Path PFILT

The auxiliary DSP block includes a programmable (and bypassable) finite impulse response filter whose output feeds into the Mux1 block. The PFILT is configured at startup using parameters stored in the device profile. Please see the Transmit/Receive Programmable Filter (PFILT) section for more details on the configurations that are supported and how to update the configuration after start-up.

RECEIVE MUX1

Mux1 is a crossbar multiplexer that defines the connection between the logical ADCs and the CDDC in the main data path, which can process real or complex data. As a result, this connection can support either real data from a single logical ADC or complex data from a pair of logical ADCs. MUX1 is not user-controllable; it is configured automatically based on whether the data is complex or real.

RECEIVE COARSE DIGITAL DATA PATH

Coarse NCO(CNCO)

Each AD9084 and AD9088 Coarse NCO has the Frequency Tuning Word (FTW) and phase value inputs taken through 16 profiles. Profile selection can be done through Trigger based hopping and direct hopping (using SPI or GPIO pins). On selecting a new profile, both FTW and phase offset values from the active profile is used for signal generation. AD9084 and AD9088 coarse NCO doesn't support independent hopping between Frequency and phase offset.

The FTW bit width is 32-bit. The Numerator and Denominator bit width of the Modulus A/B phase correction is also 32-bit.

The digital mixer supports SPI based selection between real and complex mixing.

Fs/4 power efficient mixing mode can be enabled which bypasses the NCO output for frequency translation using fs/4 frequency, thereby reducing complexity and latency requirements.

Mixer supports programmable DC test mode. In this mode the NCO output is multiplied by a fixed DC value (bypassing the ADC input) which can be programmed. By default, NCO output is multiplied by 0.

This NCO also supports Mixer bypass mode. In this mode, the ADC input is directly sent to the output without mixing with the NCO output.

Each NCO has an NCO Phase Coherence (NPC) counter to maintain NCO hopping phase coherence across all devices in a multichip application.

Coarse NCO API Functions

The Coarse NCO is controlled in a similar fashion as those in the Tx data path. The main difference is that the API's argument "terminal" must be set as ADI_APOLLO_RX. The following is a list of relevant API functions to control the Rx NCOs. Some of them are exercised within the **adi_apollo_rx_cddc_configure()** function. Other functions are used within the \examples\ads10_apollo_ex_main\ rx_mcs_dl.c example code.

- To program a new configuration for a given NCO, call **adi_apollo_cnco_pgm()**
- To program the NCO profile's frequency and phase offset, call **adi_apollo_cnco_profile_load()**
- To tune the FTW, call **adi_apollo_cnco_ftw_set()**
- To tune the Phase Offset Word, call **adi_apollo_cnco_pow_set()**
- To enable and configure NCO Hop, call **adi_apollo_cnco_hop_enable()**
- To perform a Coarse NCO hop, follow the code in \examples\ads10_apollo_ex_main\fullchip_hop.c
  - Set variable run_cnco = true and follow the example code for configuration and triggering of the coarse hop using SPI or trigger module as described in the Dynamic Reconfiguration -> CNCO/FNCO Hopping section.
- To select the NCO Mixer mode, call **adi_apollo_cnco_mode_set()**. Possible settings are:
  - ADI_APOLLO_MXR_VAR_IF_MODE = 0, /*!< Variable IF Mode */
  - ADI_APOLLO_MXR_ZERO_IF_MODE = 1, /*!< Zero IF Mode */
  - ADI_APOLLO_MXR_FS_BY_4_MODE = 2, /*!< Fs/4 Hz IF Mode */
  - ADI_APOLLO_MXR_TEST_MODE = 3 /*!< Test Mode */

APOLLO MXFE RECEIVER

- To select between complex and real mixing, call **adi_apollo_cnco_mixer_set()**. Possible options are:
  - ADI_APOLLO_DRC_MIXER_REAL = 0x0, /*!< Real Mixing Enable */. DRC (or drc) stands for data rate conversion and can mean decimation or interpolation, depending on the context.
  - ADI_APOLLO_DRC_MIXER_COMPLEX = 0x1, /*!< Complex Mixing Enable */

Coarse Digital Down-converter (CDDC) Filters

The cascaded decimation filter line up shown in Figure 67 provides a selectable decimation factor of 2, 3, 4, 6 and 12, as well as a bypass option (or 1). Using the fine_bypass_parameter in adi_apollo_cddc_cfg structure of the device profile both the FNCO and the filter chain can be bypassed. Table 39 shows the characteristics of each filter pair relative to the decimated f_{IQ_OUT}. The usable pass band is the frequency band over which the response maintains a pass-band ripple of less than 0.001 dB with an alias (or image) rejection greater than 100dB. Note that the pass band for a single filter when driven with a real input is half of the complex passband.

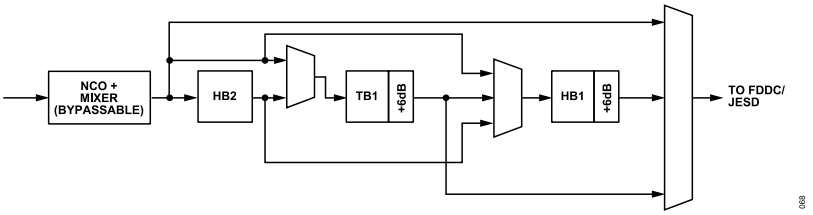

Figure 67. CDDC Decimation Filter Lineup

Table 39. CDDC Decimation Filter Characteristics

Filter Name	Decimation Ratio	Pass Band (% of f_{IQ_OUT})
HB2	2	40
HB1	2	80
TB1	3	80

CDDC Variable IF NCO Operating Modes

The NCO can be configured for different modes of operation if variable IF mode is selected. Figure 68 shows a block diagram of a digital quadrature NCO and the other stages within the CDDC. The gray lines in Figure 68 represent the SPI control lines. The 32-bit complex NCO supports the following modes of operation:

- Dual modulus mode for higher frequency resolution where the modulus is set by the phase increment numerator (nco_phase_inc_frac_a parameter or PIFA, for short) and phase increment denominator (nco_phase_inc_frac_b parameter or PIFB, for short) words.
- Integer-N mode where 32-bit frequency and phase settings are set by the phase increment word (nco_phase_inc_words parameter or PIW for short) and phase offset word (POW). This mode also supports phase coherency when switching among up to 16 different frequency assignments where the phase is referenced to a single synchronization event at Time 0. See the Receive Main and Channelizer Path FFH NCO section for more information.

APOLLO MXFE RECEIVER

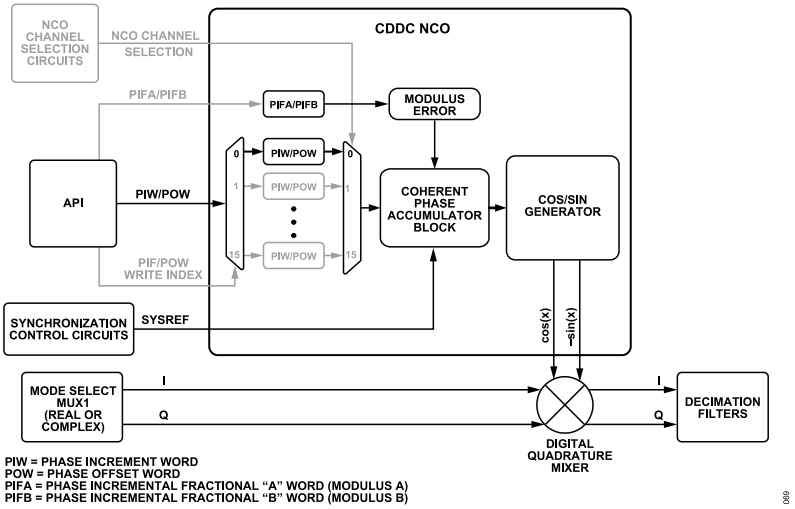

Figure 68. NCO Block Diagram with CDDC Stage

The NCO frequency value is determined by the following settings:

- A two's complement number entered in the PIW that represents the phase increment word, which is also known as the FTW.
- An unsigned number entered in the PIFA that represents the phase increment fractional numerator word.
- An unsigned number entered in the PIFB that represents the phase increment fractional denominator word.

The NCO initial phase value is determined by a two's complement phase offset word (POW). This value can create a known phase relationship between multiple chips or between individual DDC channels inside the chip.

To allow all NCOs to simultaneously update in the main receive data path, a main/subordinate implementation is used to transfer the PIW, POW, PIFA, and PIFB settings. Transfer control can be either under software control (via API) or hardware control (via the GPIO pin). Hardware control allows different profile settings to be loaded quickly with more precise timing accuracy.

Frequencies between −fADC/2 and +fADC/2 (fADC/2 is excluded) are represented using the following values for the 32-bit NCO configuration:

- PIW = 0x8000_0000 and PIFA = 0x0000_0000 represent a frequency of –fADC/2.
- PIW = 0x0000_0000 and PIFA = 0x0000_0000 represent dc (frequency is 0 Hz).
- PIW = 0x7FFF_FFFF and PIFA = 0x0000_0000 represent a frequency of just below +fADC/2.

Each NCO contains a separate phase accumulator word (PAW). The initial reset value of each PAW is set to zero and incremented every clock cycle. The instantaneous phase of the NCO is calculated using the PAW, PIW, PIFA, PIFB, and POW. This architecture allows the PIW and POW registers to be updated at any time and still maintain deterministic phase results in the PAW of the NCO while in programmable dual modulus or integer-N mode. However, in the dual modulus mode, the user must carry out the following procedure to update the PIFA and/or the PIFB registers to ensure proper NCO operation:

- Write to the PIFA and PIFB for the NCO using API
- Perform NCO profile update using API

CDDC NCO Synchronization Options

If synchronization of the NCOs across multiple Apollo MxFE devices is required, the trig_sync MCS synchronization is required. Refer to the Device Synchronization section. If only using a single device, the oneshot_sync synchronization is adequate for synchronizing the NCOs within a single Apollo MxFE.

NCO Setting Consideration for Homodyne Transmit-to-Receive Loopback Applications

Applications such as digital predistortion, where the transmitted signal is observed by the receiver, require absolute phase stability, such that the recovered signal from a transmitted unmodulated CW carrier experiences no phase drift due to transmit and receive NCO settings that are slightly misaligned. Misalignment can occur if the ratio between the DAC and ADC sample rate is not considered when setting the NCO

frequency settings in the transmit and receive data paths. If the ADC sample rate is a factor of two (or four) less than the DAC rate, set the LSB (or lower two LSBs) of the receive NCO integer-N setting to 0 to allow the tuning resolutions relative to the DAC rate to remain matched.

NCO Dual Modulus Mode

This mode operates in a fractional-N mode and automatically enables when the nco_phase_inc_frac_a (PIFA) and nco_phase_inc_frac_b (PIFB) parameters are set to a nonzero value to provide a tuning accuracy resolution of >32 bits. An example of a rational frequency synthesis requirement that requires >32 bits of accuracy is a carrier frequency of 1/3 the sample rate. When a frequency accuracy of ≤32 bits is acceptable, integer-N mode is more suitable because this mode also supports phase coherent switching when changing NCO frequencies. For programable dual modulus mode, set the PIFA to a nonzero value (not equal to 0x0000_0000_0000) and use the following equations (for more information on the programable modulus feature, see the AN-953 Application Note, *Direct Digital Synthesis (DDS) with a Programmable Modulus*):

$$\frac{mod(f_C, f_{ADC})}{f_{ADC}} = \frac{M}{N} = \frac{PIW + \frac{PIFA}{PIFB}}{2^{32}} \tag{4}$$

$$FTW = floor\left(2^{32}\frac{mod(f_C, f_{ADC})}{f_{ADC}}\right) \tag{5}$$

$$PIFA = \mathrm{mod}\left(2^{32} \times M, \quad N\right) \tag{6}$$

$$PIFB = N \tag{7}$$

where:
f_C is the desired carrier frequency.
M is the integer representing the rational numerator of the frequency ratio.
N is the integer representing the rational denominator of the frequency ratio.

Note that Equation 1 to Equation 4 apply to the aliasing of signals in the digital domain, that is, the aliasing introduced when digitizing analog signals. M, N, PIFA, and PIFB are integers reduced to the lowest terms.

For example, if the f_{ADC} is 3000 MSPS and f_C is 1001.5 MHz,

$$\frac{\mathrm{mod}(1001.5, \quad 3000)}{3000} = \frac{M}{N} = \frac{2003}{6000} \tag{8}$$

$$PIW = floor\left(2^{32}\frac{mod(1001.5,3000)}{3000}\right) = 0x0000\_5576\_19F0 \tag{9}$$

$$PIFA = \mathrm{mod}\left(2^{32} \times 2003, \quad 6000\right) = \mathrm{0x0000\_0000\_1700} \tag{10}$$

$$PIFB = 0x0000\_0000\_1770 \tag{11}$$

To calculate the actual carrier frequency (f_{C_ACTUAL}) in this case, use the following equation:

$$f_{C\_ACTUAL} = \frac{\left(PIW + \frac{PIFA}{PIFB}\right) \times f_{ADC}}{2^{32}} \tag{12}$$

In the case of this example, the f_{C_ACTUAL} is as follows:

$$f_{C\_ACTUAL} = \frac{\left(0x557619F0 + \frac{0x1700}{0x1770}\right) \times 0x0BB8}{2^{32}} = 1001.5MHz \tag{13}$$

NCO Integer-N Mode

Integer-N mode is the default NCO setting. This mode is automatically enabled when the NCO PIFA setting is set to all 0s. In this mode, calculate the NCO PIW with the following equation in the 32-bit NCO configuration:

$$PIW = \mathrm{round}\left(2^{32}\frac{\mathrm{mod}(f_C, \quad f_{ADC})}{f_{\mathrm{ADC}}}\right) \tag{14}$$

APOLLO MXFE RECEIVER

Note that Equation 5 applies to the aliasing of signals in the digital domain, that is, the aliasing introduced when digitizing analog signals.

For example, if the f_{ADC} is 6000 MSPS and the f_C is 833.334 MHz,

$$PIW = \text{round}\left(2^{32}\frac{\text{mod}(833.334,\ \ 6000)}{6000}\right) = 0\text{x}238\text{E\_}3\text{AC1} \quad (15)$$

To calculate the f_{C_ACTUAL} in this case, use the following equation:

$$f_{C\_ACTUAL} = \frac{PIW \times f_{ADC}}{2^{32}} \quad (16)$$

where the resulting f_{C_ACTUAL} value is as follows:

$$f_{C\_ACTUAL} = \frac{0\text{x}238\text{E\_}3\text{AC1} \times 6000}{2^{32}} = 833.33400032 \ \ \text{MHz} \quad (17)$$

APOLLO MXFE RECEIVER

Coarse Data Path Decimation Stage

Table 40 shows the characteristics of each filter pair relative to the decimated f_{IQ_OUT}. The usable pass band is the frequency band over which the response maintains a pass-band ripple of less than ±0.001 dB with an alias (or image) rejection greater than 100dB. Note that the pass band for a single filter when driven with a real input is half of the complex pass band. Table 41 shows the filter line up used to achieve the different decimation factors with the resulting I/Q output data rate and usable pass band assuming an ADC sample rate of 18000 MSPS.

Table 40. Main Data path Decimation Filter Characteristics

Filter Name	Decimation Ratio	Pass Band (% of f_{IQ_OUT})
HB2	2	40
HB1	2	80
TB1	3	80

Table 41. I/Q Output Data Rate and Usable Complex Pass Band vs. Decimation Ratio with f_{ADC} = 18000 MSPS

Decimation Ratio	Filter Line Up	FIQ_OUT (MSPS)	Pass Band (MHz)
2	HB1	9000	7200
3	TB1	6000	4800
4	HB2 + HB1	4500	3600
6	HB2 + TB1	3000	2400
12	HB2 + TB1+HB1	1500	1200

Optional 6 DB Gain Stage

Each CDDC contains an optional, controlled gain stage block.

The gain is selectable as either 0 dB or 6 dB and compensates for the 6 dB loss in signal level incurred when down-converting a real input signal because the undesired sideband image is typically filtered by the decimation filters (assuming it falls in the stopband region) that follow the mixing stage. Only enable this gain stage if the FDDC data path is bypassed. If the FDDC data path is also enabled, enable this similar functional block in this stage instead to prevent possible digital overflow because of excessive gain if 6 dB gain is applied in both stages.

Receive Coarse Data Path Device Profile

CDDC is configured using the **adi_apollo_cddc_cfg_t** data structure in the device profile. The parameters defined in the **adi_apollo_cddc_cfg_t** structure are listed in Table 42. Many of the parameters for the CDDC can also be changed after startup or part of a dynamic reconfiguration. Refer to the Dynamic Reconfiguration and Configuration Updates After Device Bring-Up sections for more information.

Table 42. adi_apollo_cddc_cfg_t parameters

Parameter	Description	Enumerations
nco	CNCO configuration	See **adi_apollo_cnco_cfg_t** structure described in Table 43 for more details.
drc_ratio	CDDC decimation ratio	adi_apollo_coarse_ddc_dcm_e
fine_bypass	FDDC bypass	0: FDDC bypassed 1: FDDC enabled
link_num	JESD link number the CDDC is connected to (only used when FDDC is bypassed)	JESD link number
hb1_filt_dly_en	HB1 filter delayed input enable	0: Delayed input is not used on HB1 1: HB1 filter uses a delayed input
hb2_filt_dly_en	HB2 filter delayed input enable	0: Delayed input is not used on HB2 1: HB2 filter uses a delayed input
tb1_filt_dly	TB1 filter delay selection	adi_apollo_cdrc_tb1_filt_dly_e
hb1_gain_6db_en	HB1 6dB gain block enable	0: HB1 6dB gain block disabled 1: HB1 6dB gain block enabled
tb1_gain_6db_en	TB1 6dB gain block enable	0: TB1 6dB gain block disabled 1: TB1 6dB gain block enabled
trig_mst_en	Trigger main enable	0: Trigger main disabled

APOLLO MXFE RECEIVER

Table 42. adi_apollo_cddc_cfg_t parameters (Continued)

Parameter	Description	Enumerations
		1: Trigger main enabled
trig_mst_period	Trigger main period	Trigger main period, in units of Fs (unsigned 64-bit integer). This parameter is to be used with profile select timer.
trig_mst_offset	Trigger main offset	Trigger main offset, in units of Fs (unsigned 64-bit integer). This parameter is to be used with profile select timer.
debug_cddc_clkoff_n	CDDC clocks enable	0: CDDC clocks disabled 0xFF: CDDC clocks enabled (default)

Table 43. adi_apollo_cnco_cfg_t parameters

Parameter	Description	Enumerations
drc_en	CDDC Enable	Boolean
nco_mode	CDDC NCO mode	adi_apollo_nco_mode_e
nco_if_mode	Select NCO Mixer Mode for CDDC (variable IF, zero IF, Fs/4, DC test)	adi_apollo_nco_mixer_mode_e
drc_mxr_sel	Select real or complex mixer	adi_apollo_drc_mixer_sel_e
cmplx_mxr_mult_scale_en	Set NCO complex multiplication scale	adi_apollo_nco_cmplx_mult_scale_e
dc_testmode_value	Sets the DC value used to multiply the Coarse NCO when it is in test mode.	The upper 12 bits of values between 0 and 2^13-1 are used. 2^13-1 is mapped to 0.999 full scale.
nco_phase_inc	NCO phase increment word	NCO phase increment word (unsigned 32-bit integer). See Receive course digital data path for more details).
nco_phase_offset	NCO phase offset word	NCO phase offset word (unsigned 32-bit integer). See Receive course digital data path for more details).
nco_phase_inc_frac_a	Modulus phase increment numerator	NCO modulus phase increment numerator (unsigned 32-bit integer). See Receive course digital data path for more details).
nco_phase_inc_frac_b	Modulus phase increment denominator	NCO modulus phase increment denominator (unsigned 32-bit integer). See Receive course digital data path for more details).
nco_phase_inc_words[ADI_APOLLO_CNCO_PROFILE_NUM]	NCO phase increment words	NCO phase increment words for CNCO profile (unsigned 32-bit integer array). See Receive course digital data path for more details).
nco_phase_offset_words[ADI_APOLLO_CNCO_PROFILE_NUM]	NCO phase offset words	NCO phase offset words for CNCO profile (unsigned 32-bit integer array). See Receive course digital data path for more details).
amp_dither_en	If True, Amplitude dither for NCO mixing is enabled	Boolean
phase_dither_en	If True, Phase dither for NCO mixing is enabled	Boolean
nco_profile_sel_mode	Set NCO profile selection mode. This parameter is to be used with the NCO profile hopping feature.	adi_apollo_nco_profile_sel_mode_e
nco_auto_inc_dec	Increment or decrement profile selection for automatic hopping. This parameter is to be used with the NCO profile hopping feature.	adi_apollo_nco_auto_hop_dir_sel_e
debug_cdrc_clkoff_n	Debug. If set to True (default setting), the NCO clocks are running. If set to False, the NCO clocks are disabled.	Boolean

The enumerations used by the data structures in the device profile to configure the CDDC are listed in Table 44.

Table 44. Enumerations associated with CDDC

Enumerations	Parameters	Description
adi_apollo_coarse_ddc_dcm_e	ADI_APOLLO_CDDC_DCM_1	0: Decimate by 1
	ADI_APOLLO_CDDC_DCM_2	1: Decimate by 2
	ADI_APOLLO_CDDC_DCM_3	2: Decimate by 3
	ADI_APOLLO_CDDC_DCM_4	3: Decimate by 4

APOLLO MXFE RECEIVER

Table 44. Enumerations associated with CDDC (Continued)

Enumerations	Parameters	Description
	ADI_APOLLO_CDDC_DCM_6	4: Decimate by 6
	ADI_APOLLO_CDDC_DCM_12	6: Decimate by 12
adi_apollo_nco_mode_e		Programming nco_phase_inc_frac_a to 0 puts the NCO in Integer mode.
	ADI_APOLLO_NCO_MOD_INTEGER	NCO integer-N mode
	ADI_APOLLO_NCO_MOD_INT_PLUS_MODULUS	NCO dual modulus mode
adi_apollo_nco_mixer_mode_e	ADI_APOLLO_FINE_MXR_VAR_IF_MODE	0: Variable IF Mode
	ADI_APOLLO_FINE_MXR_ZERO_IF_MODE	1: Zero IF Mode
	ADI_APOLLO_FINE_MXR_FS_BY_4_MODE	2: Fs/4 Hz IF Mode
	ADI_APOLLO_FINE_MXR_TEST_MODE	3: DC Test Mode
adi_apollo_drc_mixer_sel_e	ADI_APOLLO_DRC_MIXER_REAL	0: Real Mixing Mode
	ADI_APOLLO_DRC_MIXER_COMPLEX	1: Complex Mixing Mode
adi_apollo_nco_cmplx_mult_scale_e	ADI_APOLLO_CMPLX_MULT_SCALE_0p7	0: Enable scaling
	ADI_APOLLO_CMPLX_MULT_SCALE_1	1: Disable scaling
adi_apollo_nco_profile_sel_mode_e	ADI_APOLLO_NCO_CHAN_SEL_TRIG_AUTO	0x0: Enable trigger-based hopping. Automatic hopping mode..
	ADI_APOLLO_NCO_CHAN_SEL_TRIG_REGMAP	0x1: Enable trigger-based hopping based on register map
	ADI_APOLLO_NCO_CHAN_SEL_TRIG_GPIO	0x2: Enable trigger-based hopping based on GPIO
	ADI_APOLLO_NCO_CHAN_SEL_DIRECT_GPIO	0x3: Direct GPIO profile select. All parameters hop together
	ADI_APOLLO_NCO_CHAN_SEL_DIRECT_REGMAP	0x4: Direct SPI/HSCI NCO profile select. All parameters hop together
adi_apollo_nco_auto_hop_dir_sel_e	ADI_APOLLO_NCO_AUTO_HOP_DIR_DECR	0x0: Decrement profile number on trigger
	ADI_APOLLO_NCO_AUTO_HOP_DIR_INCR	0x1: Increment profile number on trigger

Coarse DDC API Functions

The API library fully supports configuration of CDDC. The CDDC is configured using the following API functions based on the settings in the device profile. The CDDC specific parameters and the possible values set by these API functions are listed in the Receive Coarse Data Path Device Profile section above.

The following is a list of API functions used to configure the coarse DDC.

- To program the DDC configuration, call **adi_apollo_rx_cddc_configure().**
  - Within this function, API will call **adi_apollo_cddc_pgm()** to program a set of defaults into the coarse DDC.
- To change the decimation setting, call **adi_apollo_cddc_dcm_set ()**
- To set the DDC gain call **adi_apollo_cddc_gain_enable_set()**

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

RECEIVE MUX2

Mux2 is a crossbar multiplexer shown in Figure 69 that defines the mapping of complex data transferred between the two CDDC outputs and the four FDDC inputs of AD9084 side B or side A.

Mux2 is a crossbar multiplexer shown in Figure 70 that defines the mapping of complex data transferred between the four CDDC outputs and the eight FDDC inputs of AD9088 side B or side A.

APOLLO MXFE RECEIVER

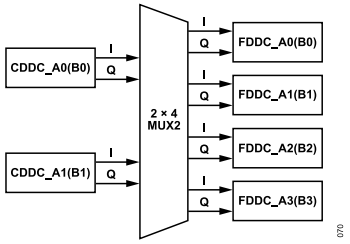

Figure 69. Side B/Side A MUX2 of AD9084

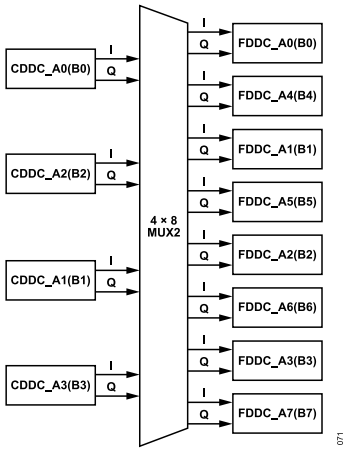

Figure 70. Side B/Side A MUX2 of AD9088

Receive Mux2 API

The API library fully supports configuration of Mux2. Mux2 is configured using the adi_apollo_rxmux_xbar2_set **adi_apollo_rxmux_xbar2_set()** API function based on the settings in the device profile. The Mux2 specific parameters and the possible values set by this API function are listed in Table 45.

Table 45. Mux 2 Configuration parameters

Parameter	Description	Enumerations
side_sel	Defines which side of Apollo to configure	ADI_APOLLO_SIDE_NONE = 0x0 ADI_APOLLO_SIDE_A = 0x1 ADI_APOLLO_SIDE_B = 0x2 ADI_APOLLO_SIDE_ALL = 0x3
coarse_to_fine_xbar_sel	List of four Enumerations. Each contains the Configuration selection for each of the Fine DDCs	Refer to adi_apollo_rx_mux2_sel_e for details on valid configurations in 4T4R and 8T8R devices.
length	Number of elements inside the coarse_to_fine_xbar_sel list.	This value must be set to 4

RECEIVE FINE DIGITAL DATA PATH

Fine NCO (FNCO)

The FNCO is used for carrier generation along with summers and multipliers to result in frequency translation and side band cancellation. The FNCO supports the following modes of operation:

- **Integer N-mode** : The FTW along with the initial phase offset control the frequency and phase offset respectively. The frequency resolution is limited to a single LSB of the FTW bit width.
- **Dual Modulus mode**: This mode allows a higher frequency resolution that is otherwise not possible using the Integer-N mode. In this mode the FTW sets the initial NCO output frequency, while the remaining fractional frequency step is set using the A/B ratio.

APOLLO MXFE RECEIVER

The NCO output frequency can be expressed as a ratio M/N with respect to the running clock rate of the FNCO (which is data rate at the input of the FNCO) as

$$\frac{f_{CARRIER}}{f_{NCO,CLK}} = \frac{M}{N} = \frac{X + \frac{A}{B}}{2^{Bits}} \quad (18)$$

Where:

- X-> Integer N value
- A-> PIFA
- B-> PIFB
- Bits-> FTW resolution
- $F_{carrier}$ -> NCO output frequency
- $F_{NCO,CLK}$-> NCO clock frequency i.e. the data rate at the input of FNCO

FNCO-MIXER MODES

The NCO-mixer supports selection between real and complex mixing. Parameter based Frequency shift in Mixer is also added. For RX the I,Q Mixer data out has negative frequency shift (Input tone – NCO tone) and TX I,Q data has positive frequency shift (Input tone + NCO tone).

The NCO – Mixer supports programmable DC test mode. In this mode the NCO output is multiplied with a fixed DC value (bypassing the ADC input) which can be programmed. By default, NCO output is multiplied with 0 (no output), so a positive or negative value must be applied to develop a test tone.

The NCO also supports Mixer bypass mode. In this mode, the FNCO Mixer input is directly sent to the FNCO Mixer output without mixing with the NCO output. There is also a special fs/4 if mode which bypasses the NCO output for frequency translation using fs/4 frequency, thereby reducing complexity and latency requirements. Each NCO has an NCO Phase Coherence (NPC) counter to maintain phase coherence across chips and data path reconfigurations.

Fine NCO API Functions

The Fine NCOs share a similar architecture as that of the coarse NCOs. This lends to similar controlling functions in API. The main functions to control the Fine NCOs have the same name as that of the Coarse NCOs. The main difference being that instead of *_cnco_*in their name, they have *_fnco_*. Refer to the Coarse NCO API Functions for a quick description of such functions.

FAST FREQUENCY HOPPING (FFH)

FTW and phase value inputs are taken through 32 profiles when hopping is enabled. Profile selection can be done through Trigger based hopping or direct hopping (using API and/or GPIO pins). On selecting new profile, both FTW and phase offset values from the active profile are used for signal generation. AD9084 and AD9088 Fine NCO supports independent hopping of frequency and phase offset using triggers.

Fine Digital Down-converter (FDDC) Filters

The cascaded decimation filter lineup shown in Figure 71 provides a selectable decimation factor of 2, 4, 8, 16, 32 and 64 as well as a bypass option (or 1) using the fine_bypass parameter in **adi_apollo_cddc_cfg** structure of the device profile, both the FNCO and the filter chain can be bypassed. The maximum f_{IQ_IN} into this decimation stage is limited to 5000 MSPS/2500MSPS on AD9084/AD9088.

The usable pass band is the frequency band over which the response maintains a pass-band ripple of less than ±0.001 dB with an alias (or image) rejection of >100dB. Note that the pass band for a single filter when driven with a real input is half of the complex pass band.

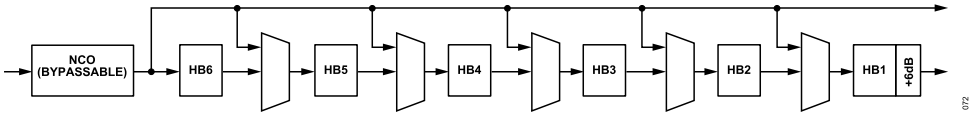

Figure 71. Receive Fine Data path Decimation Filter Lineup

APOLLO MXFE RECEIVER

Table 46. Channelizer Decimation Filter Characteristics

Filter Name	Decimation Ratio	Pass Band (% of f_{IQ_OUT})
HB6	2	2.5
HB5	2	5
HB4	2	10
HB3	2	20
HB2	2	40
HB1	2	80

FDDC Variable IF NCO Operating Modes

The NCO in the FDDC shown in Figure 71 is the same core design used for the CDDC, and the operation is the same except that a different set of API functions are used to configure the FDDC. Refer to the CDDC Variable IF NCO Operating Modes section for a more detailed version of the description that follows. Note that the data rate into the receive channelizer/fine data path (f_{FDDC_IN} or f_{IQ_IN}) is equal to the data rate out of the receive main/coarse data path (f_{CDDC_OUT} or f_{IQ_OUT}), which is also equal to f_{ADC}/M_{RX}. For this reason, the tuning range for the NCO in the FDDC is specified to be between $-f_{IQ\_IN}/2$ and $+f_{IQ\_IN}/2$ with the equations used to determine the NCO setting values based on f_{IQ_IN}.

The complex NCO supports integer-N and dual modulus, where PIW and POW words are used to define the frequency tuning and phase offset setting, and PIFA and PIFB are used for the modulus setting when operating in dual modulus mode. Note that a main/subordinate implementation is also used to update all receive channelizer NCOs simultaneously using API (SPI control).

Table 47 shows the FNCO resolution with different configurations.

Table 47. FNCO Resolution

Part configuration	FNCO PIW Resolution	PIFA/PIFB Resolution	POW Resolution
Frequency Hopping Disabled	48	24	48
Frequency Hopping Enabled	32	N/A	16

FNCO Synchronization Options for FDDC

If synchronization of the NCOs across multiple Apollo MxFE devices is required, the trig_sync MCS synchronization is required. Refer to the Device Synchronization section. If only using a single device, dyn_cfg_sync synchronization should be used for synchronizing the NCOs within a single Apollo MxFE.

FNCO Dual Modulus Mode for FDDC

This mode of operation provides a tuning accuracy resolution of >48 bits (for the non-hopping case) by operating in a fractional-N mode and is automatically enabled when the PIFA is set to a nonzero value. For programmable dual modulus mode, set the PIFA to a nonzero value (not equal to 0x00_0000) and satisfy the following four equations (for more information on the programmable modulus feature, see the AN-953 Application Note, *Direct Digital Synthesis (DDS) with a Programmable Modulus*):

$$\frac{mod\left(f_C, f_{IQ\_IN}\right)}{f_{IQ\_IN}} = \frac{M}{N} = \frac{PIW + \frac{PIFA}{PIFB}}{2^b} \tag{19}$$

where b is the resolution in bits

For the non-hopping case:

$$FTW = floor\left(2^{48}\frac{mod\left(f_C, f_{IQ\_IN}\right)}{f_{IQ\_IN}}\right) \tag{20}$$

$$PIFA = \mathrm{mod}\left(2^{48} \times M, \quad N\right) \tag{21}$$

$$PIFB = N \tag{22}$$

For the hopping case:

$$FTW = floor\left(2^{32}\frac{mod(f_C, f_{IQ\_IN})}{f_{IQ\_IN}}\right) \quad (23)$$

$$PIFA = \text{mod}\left(2^{32} \times M, \quad N\right) \quad (24)$$

$$PIFB = N \quad (25)$$

Note: N should be chosen such that it never exceeds 2^{24} for both the non-hopping and hopping cases.

It is not recommended that Dual Modulus mode be used with profile hopping. Profiles in profile hopping do not support PIFA and PIFB parameters and coherence can not be maintained in Dual Modulus mode.

FNCO Integer-N Mode for FDDC

Integer-N mode is the default setting for the FDDC NCOs and is automatically applied when the NCO PIFA setting is set to all 0s. In this mode, calculate the NCO PIW with the following equation:

$$PIW = \text{round}(2^{48}\frac{\text{mod}(f_c, f_{IQ\_IN})}{f_{IQ\_IN}}) \quad (26)$$

Optional 6 DB Gain Stage

Each receive FDDC path contains a 6 dB gain stage. This block can be bypassed if not needed and can be independently controlled. These features can be controlled using device profile.

The gain is selectable as either 0 dB or 6 dB and compensates for the 6 dB loss in signal incurred when down-converting a real input signal with the associated sideband image sufficiently filtered via the decimation filter(s) in the signal path. To prevent possible overflow, only apply the 6 dB compensation to the receive channelizer and disable the compensation to the receive main data path.

Receive Fine Digital Data Path Device Profile

FDDC is configured using the **adi_apollo_fddc_cfg_t** data structure in the device profile. The parameters defined in the **adi_apollo_fddc_cfg_t** structure are listed in Table 48.

Table 48. adi_apollo_fddc_cfg_t parameters

Parameter	Description	Enumerations
nco	FNCO configuration	See **adi_apollo_fnco_cfg_t** structure described in Table 49 for more details.
drc_ratio	FDDC decimation ratio	adi_apollo_fddc_ratio_e
link_num	JESD link number	JESD link number FDDC is connected to (8-bit unsigned integer)
debug_fddc_clkoff_n	FDDC clocks enable	0: FDDC clocks disabled 0xFF: FDDC clocks enabled (default)
hb1_filt_dly_en	Enable HB1 sample delay	TRUE/FALSE
hb2_filt_dly_en	Enable HB2 sample delay	TRUE/FALSE
hb3_filt_dly_en	Enable HB3 sample delay	TRUE/FALSE
hb4_filt_dly_en	Enable HB4 sample delay	TRUE/FALSE
hb5_filt_dly_en	Enable HB5 sample delay	TRUE/FALSE
hb6_filt_dly_en	Enable HB6 sample delay	TRUE/FALSE
hb1_gain_6db_en	Enable 6dB gain	TRUE/FALSE

Table 49. adi_apollo_fnco_cfg_t parameters

Parameter	Description	Enumerations
drc_en	FDDC enable	0: FDDC disabled 1: FDDC enabled
nco_mode	FDDC NCO mode	**adi_apollo_nco_mode_e** (See enumerations in CDDC section for details)

APOLLO MXFE RECEIVER

Table 49. adi_apollo_fnco_cfg_t parameters (Continued)

Parameter	Description	Enumerations
dc_testmode_value	FDDC mixer test mode value (14-bit)	Mixer test mode value (16-bit unsigned integer)
nco_if_mode	Select NCO Mixer Mode for CDDC (variable IF, zero IF, Fs/4, DC test)	**adi_apollo_nco_mixer_mode_e** (See enumerations in CDDC section for details)
drc_mxr_sel	Select real or complex mixer	**adi_apollo_drc_mixer_sel_e** (See enumerations in CDDC section for details)
cmplx_mxr_mult_scale_en	Set NCO complex multiplication scale	**adi_apollo_nco_cmplx_mult_scale_e** (See enumerations in CDDC section for details)
nco_phase_inc	NCO phase increment word	NCO phase increment word (unsigned 64-bit integer). See Receive course digital data path for more details).
nco_phase_offset	NCO phase offset word	NCO phase offset word (unsigned 64-bit integer). See Receive course digital data path for more details).
nco_phase_inc_frac_a	Modulus phase increment numerator	NCO modulus phase increment numerator (unsigned 64-bit integer). See Receive fine digital data path for more details).
nco_phase_inc_frac_b	Modulus phase increment denominator	NCO modulus phase increment denominator (unsigned 64-bit integer). See Receive fine digital data path for more details).
hop_mode_en	Enable frequency hopping for FNCO	0: Disable FNCO frequency hopping 1: Enable FNCO frequency hopping
hop_mode	FNCO Hop Mode	Set to Zero for standard hopping
nco_phase_inc_words[ADI_APOLLO_FNCO_PROFILE_NUM]	NCO phase increment words	NCO phase increment words for FNCO profile (unsigned 64-bit integer array). See Receive course digital data path for more details).
nco_phase_offset_words[ADI_APOLLO_FNCO_PROFILE_NUM]	NCO phase offset words	NCO phase offset words for FNCO profile (unsigned 64-bit integer array). See Receive course digital data path for more details).
nco_profile_sel_mode	Set NCO profile selection mode. This parameter is to be used with the NCO profile hopping feature.	**adi_apollo_nco_profile_sel_mode_e** (See enumerations in CDDC section for details)
nco_auto_inc_dec_freq	Increment or decrement frequency profile selection for automatic hopping. This parameter is to be used with the NCO profile hopping feature.	**adi_apollo_nco_auto_hop_dir_sel_**e (See enumerations in CDDC section for details)
nco_auto_inc_dec_phase	Increment or decrement phase profile selection for automatic hopping. This parameter is to be used with the NCO profile hopping feature.	**adi_apollo_nco_auto_hop_dir_sel_e** (See enumerations in CDDC section for details)
nco_trig_hop_sel	Select parameters (phase and/or frequency) to hop	adi_apollo_fnco_trig_hop_sel_e
debug_fdrc_clkoff_n	FDDC NCO clocks enable	0: FDDC NCO clocks disabled 1: FDDC NCO clocks enabled (default)

The enumerations used by the data structures in the device profile to configure the FDDC are listed in Table 50.

Table 50. Enumerations associated with FDDC

Enumerations	Parameter	Descriptions
adi_apollo_fddc_ratio_e	ADI_APOLLO_FINE_DRC_RATIO_1	0: No decimation
	ADI_APOLLO_FINE_DRC_RATIO_2	Decimate by 2
	ADI_APOLLO_FINE_DRC_RATIO_4	Decimate by 4
	ADI_APOLLO_FINE_DRC_RATIO_8	Decimate by 8
	ADI_APOLLO_FINE_DRC_RATIO_16	Decimate by 16
	ADI_APOLLO_FINE_DRC_RATIO_32	Decimate by 32
	ADI_APOLLO_FINE_DRC_RATIO_64	Decimate by 64
adi_apollo_fnco_trig_hop_sel_e	ADI_APOLLO_FNCO_TRIG_HOP_FREQ	0x01: Hop frequency profile in trigger-based hopping
	ADI_APOLLO_FNCO_TRIG_HOP_PHASE	0x04: Hop phase profile in trigger-based hopping

APOLLO MXFE RECEIVER

Table 50. Enumerations associated with FDDC (Continued)

Enumerations	Parameter	Descriptions
	ADI_APOLLO_FNCO_TRIG_HOP_FREQ_PHASE	0x05: Hop both frequency and phase profiles in trigger-based hopping

Fine DDC API Functions

The API library fully supports configuration of FDDC. The FDDC is configured using the following API functions based on the settings in the device profile. The FDDC specific parameters and the possible values set by these API functions are listed in this section. Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

The following is a list of API functions used to configure the fine DDC. They resemble those functions used to configure the Coarse DDCs with a small change in their name. They mention *_fddc_* rather than *_cddc_*

- To program the DDC configuration, call adi_apollo_rx_fddc_configure(). See the following table for required arguments.
  - Within this function, API calls adi_apollo_fddc_pgm() to program, the configuration for the desired Fine DDCs
- To change the decimation setting, call adi_apollo_fddc_dcm_set ()
- To set the DDC gain call adi_apollo_fddc_gain_enable_set()

Table 51. adi_apollo_rx_fddc_configure function configuration parameters

Parameter	Description	Enumerations
side	Apollo MxFE device side to be configured	**adi_apollo_sides_e** (See enumerations in CDDC section for details)
idx	FDDC index to be configured	adi_apollo_fddc_idx_e
config	FDDC configuration parameters.	**adi_apollo_fddc_cfg_t** data structure. See **adi_apollo_fddc_cfg_t** table for more details.

The enumerations used by the API functions to configure the FDDC are listed in Table 52.

Table 52. Enumerations associated with FDDC

Enumerations	Parameters	Description
adi_apollo_fddc_idx_e	ADI_APOLLO_FDDC_IDX_0	Configure FDDC index 0
	ADI_APOLLO_FDDC_IDX_1	Configure FDDC index 1
	ADI_APOLLO_FDDC_IDX_2	Configure FDDC index 2
	ADI_APOLLO_FDDC_IDX_3	Configure FDDC index 3
	ADI_APOLLO_FDDCS_PER_SIDE	Configure all FDDCs

DSP FUNCTIONS (@ INTERFACE DATA RATE)

Receive Path CFIR

Each receive channel has a 16-tap CFIR for equalizing the roll off within the interested band that caused by the analog components , such as amplifier, mixer, converter, or the reflection ripples due to the impedance mismatching. See the Programmable Complex FIR Filter (CFIR) - Receive and Transmit section for details.

RECEIVE PATH DATA RATE CONVERSION

Fractional Sample Rate Converter (FSRC, AD9084 Only)

The Fractional Sample Rate Converter on the ADC data path provides the AD9084 with the flexibility to support output data rates that are not an integer fraction of the ADC sample rate. The same JESD204B/C mode can be used for multiple fractional rates on a selected data path without the need to reinitialize the link. To support this flexibility, invalid sample insertion is used to correct the resultant data rate mismatch between the ADC data path and the JTX block. It has support to operate on two streams of independent data. The AD9084 has one FSRC per A-side, B-side. Please see the Fractional Sample Rate Converter (FSRC) for Transmit and Receive (AD9084 Only) section for more details.

APOLLO MXFE RECEIVER

Up-sampler

When the FSRC is not in use, an up-sampler block resides after the receive channelizers to match the IQ data rates of all channelizer outputs assigned to a JESD204B/C link. This feature allows the host processor to still benefit from the on-chip decimation filter capability in cases where different decimation factors are assigned to the channelizers, provided that the decimation factors are related by a factor of 2^N. The up-sampler repeats the samples of the lower data rate channels by 2^{N-1} to match the highest data rate channel and the host processor performs the opposite action of removing these redundant samples to reduce the data rate back to the original rate. For more details on the up-sampler operation, see the CDDC/FDDC Dynamic Reconfiguration section.

NS Converter

When the DDCs are enabled in the data path the number of active samples per cycle depends on the decimation ratio that has been selected. In cases where the user needs to support lower data rates, the data needs an additional level of processing before being passed on to the JTX block. The NS_CONV block is used to convert the number of parallel samples in the AD9084 and AD9088 digital to be converted the desired format per cycle which then makes it usable further along the data path.

Invalid Sample Insertion

Invalid samples are inserted into the data stream side to support the fractional sample rate conversion. An invalid sample is a negative full scale (-FS) value sample. The valid and invalid samples are combined to ensure the data rate is converted from what is received from the ADC data path to the rate that the JTX is expecting. See the Invalid Samples section for more details.

DATA FORMAT

The Data Format (DFORMAT) block is used to format the incoming data from the different sources such as fine stage DDC, and test mode blocks, and multiplex the output data and control signals to the JESD Transmitter Interface (JTx). Two instances of DFORMAT block have been used in AD9084 and AD9088 to support two JTx links and programming of these two blocks is independent to each other. See JESD204B/C Transmitter Sample Crossbar (Receive MUX3) and Fractional Sample Rate Converter (FSRC) for Transmit and Receive (AD9084 Only) sections for more details.

DFORMAT CONFIGURATION API

The following API functions configure the DFORMAT block:

- adi_apollo_dformat_pgm()
- adi_apollo_dformat_res_sel_set()
- adi_apollo_dformat_format_sel_set()
- adi_apollo_dformat_ctrl_bit_sel_set()
- adi_apollo_dformat_conv_test_mode_enable_set()
  - This function puts the target link in test mode
- adi_apollo_dformat_overflow_status_get()
  - This function reads the overflow status of one of the JESD links. The function can clear the status.
- adi_apollo_dformat_overflow_status_clear()
  - This function can be used to manually clear the over status

The DFORMAT Configuration API parameters for these functions are described in, Table 53, Table 54, Table 55, and Table 56. Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 53. (adi_apollo_dformat_pgm_t)

Parameter	Description	Enumerations or Settings	Comment
dfor_sel	Enumerates Digital ADC Data Format selection	adi_apollo_dformat_sel_e (dfor_sel)	
		ADI_APOLLO_DFOR_COMPLEMENT_2	2's complement
		ADI_APOLLO_DFOR_OFFSET_BINARY	Offset Binary

APOLLO MXFE RECEIVER

Table 53. (adi_apollo_dformat_pgm_t) (Continued)

Parameter	Description	Enumerations or Settings	Comment
		ADI_APOLLO_DFOR_GRAY_CODE	Gray Code
dfor_inv	Digital ADC Sample Invert on link	0 1	ADC sample data is NOT inverted ADC sample data is inverted
dfor_ddc_dither_en	Dformat dither enable for DDC mode for link	0 1	dformat dither disable dformat dither enable
dfor_res	Enumerates Chip Output Resolution	adi_apollo_chip_output_res_e ADI_APOLLO_CHIP_OUT_RES_16BIT ADI_APOLLO_CHIP_OUT_RES_15BIT ADI_APOLLO_CHIP_OUT_RES_14BIT ADI_APOLLO_CHIP_OUT_RES_13BIT ADI_APOLLO_CHIP_OUT_RES_12BIT ADI_APOLLO_CHIP_OUT_RES_11BIT ADI_APOLLO_CHIP_OUT_RES_10BIT ADI_APOLLO_CHIP_OUT_RES_09BIT ADI_APOLLO_CHIP_OUT_RES_08BIT	 16Bit 15Bit 14Bit 13Bit 12Bit 11Bit 10Bit 9Bit 8Bit
link_en	Enables or disables link	1 0	Enable Disable
dcm_ratio	The least overall decimation among DDCs present in link	Managed by the API function adi_apollo_dformat_pgm()[Should be reviewed by API team]	Within a link, all the DDCs should have same data rate. When DDCs are programmed to have different decimation (within the same link), then rates will be different from each other.Before entering the JTX block, DDC rates should match. The LINK_DDC_DEC_0 and LINK_DDC_DEC_1 parameters are defined for this purpose and hold the least decimation value across the DDCs (in a given link). All the DDCs are "up-sampled" as per the LINK_DDC_DEC_0 and LINK_DDC_DEC_1 parameters so the final data rates of the DDCs are matching and the rate correspond to the values in these parameters. (If any DDC ratio is already matching the ratio in the LINK_DDC_DEC_x parameter, then up-sampling is not done for that DDC).
total_dcm	Link Total Decimation	Managed by the API function **adi_apollo_dformat_pgm()**[Should be reviewed by API team]	Specified in parameter link_total_ratio_0 for link0, parameter link_total_ratio_1 for link1 link_total_ratio_0 and link_total_ratio_1 are programmed such that link_total_ratio_x*NS/8 = floor(LINK_DDC_DEC*FSRC_DEC *NS/8) ; where NS is the JTX's NS parameter corresponding to each mode
invalid_en	Invalid Sample Insertion enable	0 or 1	Used with FSRC
sample_repeat_en	Sample repeat Enable	0 or 1	

Table 54. DFORMAT Configuration API Parameters (adi_apollo_dformat_res_sel_set)

Parameter	Description	Enumerations or Settings	Comment
Device	Pointer to device data structure		
Links	Target link select	adi_apollo_jesd_link_select_e ADI_APOLLO_LINK_NONE ADI_APOLLO_LINK_A0 ADI_APOLLO_LINK_A1 ADI_APOLLO_LINK_B0 ADI_APOLLO_LINK_B1 ADI_APOLLO_LINK_ALL	 No Link A-side Link 0 A-side Link 1 B-side Link 0 B-side Link 1 All Links
Resolution	Chip output resolution	adi_apollo_chip_output_res_e	

APOLLO MXFE RECEIVER

Table 54. DFORMAT Configuration API Parameters (adi_apollo_dformat_res_sel_set) (Continued)

Parameter	Description	Enumerations or Settings	Comment
		ADI_APOLLO_CHIP_OUT_RES_16BIT	16Bit
		ADI_APOLLO_CHIP_OUT_RES_15BIT	15Bit
		ADI_APOLLO_CHIP_OUT_RES_14BIT	14Bit
		ADI_APOLLO_CHIP_OUT_RES_13BIT	13Bit
		ADI_APOLLO_CHIP_OUT_RES_12BIT	12Bit
		ADI_APOLLO_CHIP_OUT_RES_11BIT	11Bit
		ADI_APOLLO_CHIP_OUT_RES_10BIT	10Bit
		ADI_APOLLO_CHIP_OUT_RES_09BIT	9Bit
		ADI_APOLLO_CHIP_OUT_RES_08BIT	8Bit

Table 55. DFORMAT Configuration API Parameters (adi_apollo_dformat_format_sel_set)

Parameter	Description	Enumerations or Settings	Comment
Device	Pointer to device data structure		
Links	Target link select	adi_apollo_jesd_link_select_e	
		ADI_APOLLO_LINK_NONE	No Link
		ADI_APOLLO_LINK_A0	A-side Link 0
		ADI_APOLLO_LINK_A1	A-side Link 1
		ADI_APOLLO_LINK_B0	B-side Link 0
		ADI_APOLLO_LINK_B1	B-side Link 1
		ADI_APOLLO_LINK_ALL	All Links
Format	Data Format	0	2's complement
		1	offset binary
		2	gray code

Table 56. DFORMAT Configuration API Parameters (adi_apollo_dformat_ctrl_bit_sel_set)

Parameter	Description	Enumerations or Settings
Device	Pointer to device data structure	
Links	Target link select	adi_apollo_jesd_link_select_e
		ADI_APOLLO_LINK_NONE = 0x0, (No Link)
		ADI_APOLLO_LINK_A0 = 0x1, (A-side Link 0)
		ADI_APOLLO_LINK_A1 = 0x2, (A-side Link 1)
		ADI_APOLLO_LINK_B0 = 0x4, (B-side Link 0)
		ADI_APOLLO_LINK_B1 = 0x8, (B-side Link 1)
		ADI_APOLLO_LINK_ALL = 0xf, (All Links)
bit0_sel	Bit0 control bit function	ADI_APOLLO_DFOR_CTRL_SEL_0 = 0x0, (Selects Overflow as status bit at the output)
		ADI_APOLLO_DFOR_CTRL_SEL_1 = 0x1, (Selects tie low as status bit at the output)
		ADI_APOLLO_DFOR_CTRL_SEL_2 = 0x2, (Selects SMON as status bit at the output)
		ADI_APOLLO_DFOR_CTRL_SEL_3 = 0x3, (Selects Fast Detect as status bit at the output)
		ADI_APOLLO_DFOR_CTRL_SEL_4 = 0x4, (Reserved)
		ADI_APOLLO_DFOR_CTRL_SEL_5 = 0x5, (Selects SYSREF as status bit at the output (full bw mode only))
		ADI_APOLLO_DFOR_CTRL_SEL_6 = 0x6, (Selects Invalid data status as status bit at the output
		ADI_APOLLO_DFOR_CTRL_SEL_7 = 0x7, (Selects negative full scale as status bit at the output
Bit1_sel	Bit1 control bit function	Same as bit0_sel
Bit2_sel	Bit2 control bit function	Same as bit0_sel

JESD204B/C TRANSMITTER FUNCTIONAL OVERVIEW

There are 24 JESD204B/C transmit data lanes (12 lanes per A/B-sides) available to transmit the serialized sample data to a logic device. The 12 JESD204B/C lanes on either side of the device can be combined to form either one (single-link) or two (dual-link) links. So, the device supports up to four independent links from the AD9084 and AD9088 devices to a single or multiple logic devices.

APOLLO MXFE RECEIVER

Each link can receive data from unique data formatting blocks per link. Both single-link and dual-link JESD204B/C modes align individual sync(local) clocks to the same system reference (SYSREF_x_P/N) and device clock (CLK_X_N/P) signals.

When operating with the 8-bit/10-bit link layer (JESD204B enabled), each of the SYNCINB_xx_P/N signals are specific to the respective JESD204B link (up to two links per side), and in dual-link mode the two links can operate independently from one another. For example, one link can be powered down while the other link is running. If the 8-bit/10-bit link layer option is selected, the link operation complies to the JESD204B and JESD204C (8-bit/10-bit link layer) standards and the link lane rates can be between 5 Gbps and 20 Gbps.

The two links can also operate independently from one another when operating with the 64-bit/66-bit link layer (JESD204C enabled) in dual-link mode. If the 64-bit/66-bit link layer option is selected, the link operation complies to the JESD204C standard including the new synchronization process (SYNCINB_xx_P/N pins are not used) and the link lane rates can be between 1 Gbps and 28.21 Gbps.

The JESD204B/C serial interface hardware is grouped into three layers: the physical layer, the data link layer, and the transport layer. The functional block diagram of the JESD204B/C transmitter is illustrated in Figure 72.

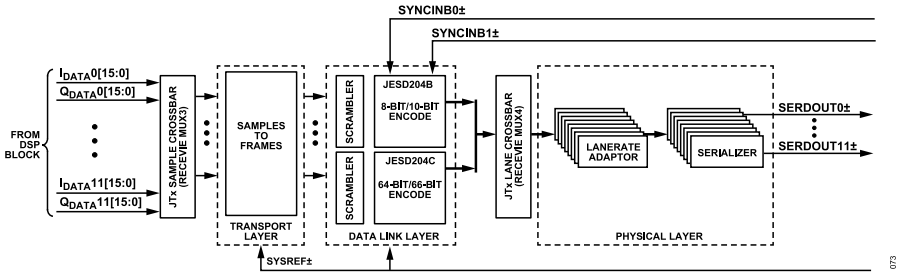

Figure 72. Functional Block Diagram of the JESD204B/C Transmitter

Dual Link Support

All modes in the JESD204B/C Transmitter Mode Tables are supported for link 0. Modes with M<=8 and L<=6, are also supported for dual link configurations (both link 0 and link 1 support these modes). The JESD204B/C Transmitter Mode Table indicates dual link mode support with a "yes" in the "dual link" column of the table. The number of links is set by the dual_link parameter in the device profile.

In a dual link configuration (dual_link = TRUE), the lane rate for one link should be a power of 2 multiple of the other. This is required since the SERDES PLL runs at same clock rate for all links (links A[0:1] and B[0:1]). If one lane rate differs from the other, then lane rate adaptation can be used to ensure the effective bit rate is the same. Refer to the JESD204B/C Transmitter Lane Rate Adaptation section for details. The total number of lanes used, whether in single link or dual link mode, cannot exceed 12. Also note that the total number of virtual converters (M_link0 + M_link1) must be less than the max value of M for the device configuration (8 for 4T4R, 16 for 8T8R).

JESD204B/C Transmitter Clock Relationships

The following clock rates are used throughout the rest of the JESD204B/C section. The relationship between any of the clocks can be derived from the following equations:

- Data Rate = ADC rate/Total Decimation Ratio
- Lane Rate = (M/L)×NP×(10/8)×Data Rate for 8-bit/10-bit encoding
- Lane Rate = (M/L)×NP×(66/64)×Data Rate for 64-bit/66-bit encoding
- PCLK Rate = Lane Rate/40 for 8-bit/10-bit encoding. PCLK = processing clock and is the rate at which samples are processed in quad-byte encoders/decoders
- PCLK Rate = Lane Rate/66 for 64-bit/66-bit encoding
- PCLK Factor = 4/F

Where:

- M is the JESD204B/C parameter for converters per link, which is the effective number of converters as seen by the JESD204B/C interface (not necessarily equal to the number of ADC cores).
- L is the JESD204B/C parameter for lanes per link.
- F is the JESD204B/C parameter for octets per frame per lane.

APOLLO MXFE RECEIVER

- NP is the JESD204B/C parameter for the total number of bits per sample.

JESD204B/C Transmitter Sample Crossbar (Receive MUX3)

The JESD204B/C Transmitter Sample Crossbar is programmed to ensure the M[0..n] samples are routed to the transport layer appropriately for the given device configuration. Each JESD204B/C Transmitter (A/B) can be configured for a maximum of 16 virtual converters (M=16) in an 8T8R configuration (AD9088) and 8 virtual converters (M=8) in a 4T4R configuration (AD9084). Each one of the virtual converters being used in a particular JESD204B/C transmitter configuration has its samples routed through JESD204B/C Transmitter Sample Crossbar. Beyond the number of transmit and receive channels in the device configuration and the JESD204B/C 'M' parameter, the data path configuration (decimation mode) also effects the routing into the transport layer. The sample_xbar_sel parameter index in the **adi_apollo_jesd_tx_link_cfg_t** structure of the device profile is derived from these configuration parameters to configure the mux appropriately.

4T4R MUX3 Sample Data Routing

When using a 4T4R configured device, the routing through MUX3 is straight forward and can be left at the default settings. This is illustrated for three cases in Figure 73.

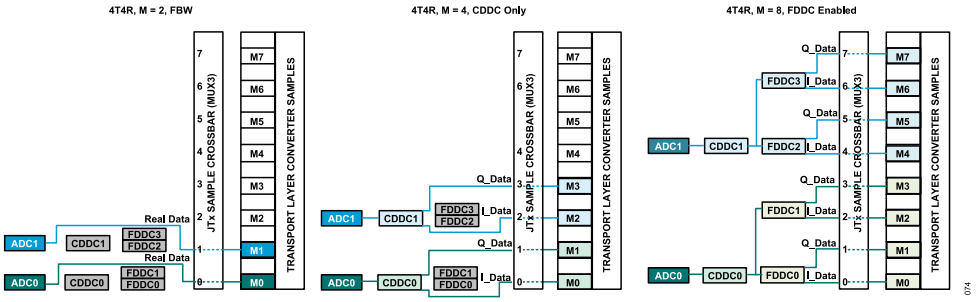

Figure 73. MUX3 sample data routing examples for 4T4R device configuration

8T8R MUX3 Sample Data Routing

When using an 8T8R configured device, the routing through MUX3 is not straight forward and the mux is configured based on the sample_xbar_sel parameter index that is part of the **adi_apollo_jesd_tx_link_cfg_t** structure in the device profile. Figure 74 illustrates three use cases. The parameters are described in more details in the Receive MUX3 Configuration section.

For details on which DDC filters are used in which configuration, refer to the Receive Coarse Digital Data Path and Receive Fine Digital Data Path sections as well as the Data Format section. These sections describe which CDDCs and FDDCs are used for the supported decimation ratios in both 4T4R and 8T8R configurations.

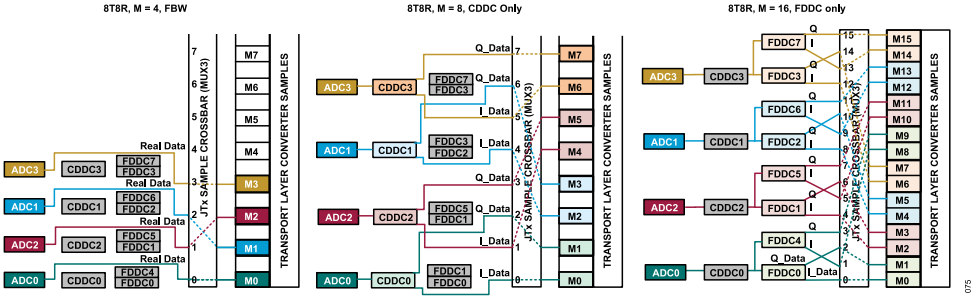

Figure 74. MUX3 sample data routing examples for 8T8R device configuration

Receive MUX3 Configuration

The device profile contains the Receive MUX3 configuration information in the **adi_apollo_jesd_tx_link_cfg_t** structure. The parameter and enumeration are described in Table 57. The firmware will power down any unused PHY lanes.

APOLLO MXFE RECEIVER

Table 57. JESD204C Transmitter Crossbar Parameter per Virtual Converter

Parameter	Description	Enumerations
sample_xbar_sel	Index that sets the source sample data for each of the virtual converters	ADI_APOLLO_JESD_FRM_SPLXBAR_RX[0..7]_BAND_[0,1]_DATA_[I,Q]ADI_APOLLO_JESD_FRM_SPLXBAR_0RX[0..7]_BAND_[0,1]_DATA_[I,Q]

Configuring the JESD204B/C Transmitter Link

Configuration of the JESD204B/C Transmitter is performed using the appropriate parameters from the device profile after the device profile is loaded as part of the "Load Apollo Profile and Firmware Boot" step of the of the Figure 12. Most of the example code described in the ADI Evaluation System Software Examples section provide an example of how to use this function except for the examples demonstrating loopbacks or NCO test modes.

The link is enabled via the **adi_apollo_jtx_link_enable_set()** API command during the "Enable Apollo JESD Links" portion of the Apollo MxFE Bring-up Procedure flow.

The parameters for the JESD204B/C Transmitter contained in the device profile in a few different structures. They are listed per block function in the relevant subsections below along with any relevant API functions.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

JESD204B/C Transmitter Transport Layer

The transport layer handles packing the data that consists of samples and optional control bits into JESD204B/C frames that are mapped to 8-bit octets. These octets are sent to the data link layer. The transport layer mapping is controlled by rules derived from the link parameters.

For more information on the transport layer, refer to the Analog Devices webcast, the JESD204B Transport Layer here.

JESD204B/C Transmitter Transport Layer Configuration Parameters

The parameters for the transport layer in the device profile are determined by the Quick Configuration Value as described in the JESD204B/C Mode table (Table 65). Alternately, the JESD204B/C mode parameters can be set individually if the quick_cfg_en parameter is not set. The most relevant parameters related to the JESD transport layer are described in Table 58 . These parameters are found in the **adi_apollo_jesd_tx_link_cfg_t** structure of the device profile.

Table 58. JESD204B/C Transmitter Transport Layer Parameters

JESD Parameter	Description	Enumerations	Comments
quick_cfg_en	Enable Quick configuration mode	Boolean (0 or 1)	0 = configure JESD using JESD204 parameter values 1= configure JESD using quick_mode_id
quick_mode_id	Quick configuration mode ID	uint8_t	See Table 65
l_minus1	number of lanes per link, values supported are 1, 2, 3, 4, 6, 8, or 12	ADI_APOLLO_LANES_PER_LINK_[1,2,3,4,6,8,12]	Enum 1 = 1 laneetc
m_minus1	number of virtual converters per link, values supported are 1, 2, 3, 4, 6, 8, 12, or 16	ADI_APOLLO_CONV_PER_LINK_[1,2,3,4,6,8,12,16]	Enum 1 = 1 virtual converter, etc.
f_minus1	octets per lane per frame, values supported are 1, 2, 3, 4, 6, 8, 12, 16, 24, or 32	ADI_APOLLO_OCT_PER_FRM_[1,2,3,4,6,8,12,16,24,32]	Enum 1 = 1 octet per frame, etc.
s_minus1	samples per virtual converter per frame cycle. Values supported are 1, 2, 3, 4, 6, 8, 12, or 16 (S = (8*L*F)/(M*N'))	ADI_APOLLO_SAMP_CONV_FRM_[1,2,3,4,6,8,12,16]	Enum 1 = 1 sample per converter, etc.
np_minus1	number of bits per JESD word, values supported are 8, 12, or 16.	ADI_APOLLO_BITS_PER_SAMP_[8,12,16]	Enum _8 = JESD word size is 8, etc.
k_minus1	number of frames per multiframe, (must be ≤ 256)	uint8_t	= 31 for 204B and K=E*(256/F) for 204C
e_minus1	Number of multiblocks in an extended multiblock (204C only)	uint8_t	0 = 1 multiblock E =LCM(F,256)/F

APOLLO MXFE RECEIVER

Table 58. JESD204B/C Transmitter Transport Layer Parameters (Continued)

JESD Parameter	Description	Enumerations	Comments
Cs	number of control bits per sample, values supported are 0, 1, 2, or 3	ADI_APOLLO_CONT_BITS_PER_SAMP_[0,1,2,3]	Enum 0 = no control bits, etc.

JESD204B/C Transmitter Data Link Layer Selection

The JESD204B/C transmitter in the device receiver can operate using the 8-bit/10-bit link layer (JESD204B) or the 64-bit/66-bit link layer (JESD204C). The ver parameter in the **adi_apollo_jesd_common_cfg_t** structure of the device profile is used to make this selection. Table 59 describes this parameter and its enumerations. Note that this parameter (along with all parameters in the **adi_apollo_jesd_common_cfg_t** structure) apply to both the JESD204B/C transmitter and receiver.

Table 59. JESD204B/C Transmitter Data Link Layer Selection Parameter

Parameter	Description	Enumerations	Comments
ver	Selects the JESD204 version	ADI_APOLLO_JESD_204B ADI_APOLLO_JESD_204C	version is 204Bversion is 204C

JESD204B TRANSMITTER 8-BIT/10-BIT LINK LAYER

The 8-bit/10-bit data link layer is responsible for the low-level functions available to pass data across the link. These functions include optionally scrambling the data, encoding 8-bit octets into 10-bit symbols, and inserting control characters for multichip synchronization, lane alignment, and monitoring.

The data link layer is also responsible for sending the ILAS, which contains the link configuration data used by the receiver to verify the settings in the transport layer.

SYNCINB_xx± Interface Options

The SYNCINB_xx receiver input supports dc-coupled, single-ended CMOS logic sources and differential LVDS sources which are selectable by passing the appropriate syncb_lvds_mode parameter value (described in Table 60) to the **adi_apollo_gpio_sync_pad_cmos_lvds_mode_set()** function in the Apollo API. The value of ADI_APOLLO_JESD_FRM_SYNCB_LVDS_NO_INTL_TERM is recommended which means to use an external 100 Ω termination on the hardware for the differential LVDS mode. The SYNCINB_xx receiver is only required in the establishment of the JESD204B link (or possible use as a GPIO input) and is powered down by default. Ensure that the logic input level into SYNCINB_xx+ does not exceed the DVDD1P8 supply. If this is not possible, a voltage divider or level translator must be used to reduce the maximum logic voltage input such that it does not exceed DVDD1P8 supply.

For single-ended CMOS operation, the syncb_lvds_mode parameter is set to ADI_APOLLO_JESD_FRM_SYNCB_CMOS . In this case, ensure that the logic input level is referenced to the same DVDD1P8 supply used by the digital CMOS input/outputs on the device.

The SYNCINB signal can be routed from one, or both of the SYNCINB pins using the syncb_in_sel parameter that is also stored in the device profile per A/B-side. These parameters are described in Table 60.

Both the syncb_in_sel and syncb_lvds_mode parameters are stored in the device profile.

Table 60. JESD204B/C Transmitter SYNCINB Configuration Parameters

Parameter	Description	Enumerations	Comments
syncb_in_sel	Selects SYNCINB pin	0 = ADI_APOLLO_JESD_FRM_PWR_OFF_ALL_SYNC_PADS	No SYNCINB
		1 = ADI_APOLLO_JESD_FRM_PWR_ON_SYNC_PAD0	Selects SYNCINB_A0/B0 pin
		2 = ADI_APOLLO_JESD_FRM_PWR_ON_SYNC_PAD1	Selects SYNCINB_A1/B1 pin
		3 = ADI_APOLLO_JESD_FRM_PWR_ON_ALL_SYNC_PADS	Selects both A0/B0 and A1/B1 pins
syncb_lvds_mode	Selects the SYNCINB mode	0 = ADI_APOLLO_JESD_FRM_SYNCB_CMOS	CMOS output
		1 = ADI_APOLLO_JESD_FRM_SYNCB_LVDS_WITH_INTL_TERM	LVDS w/termination
		2 =ADI_APOLLO_JESD_FRM_SYNCB_LVDS_NO_INTL_TERM	LVDS no termination

APOLLO MXFE RECEIVER

8-Bit/10-Bit Encoder

The 8-bit/10-bit encoder converts 8-bit octets into 10-bit symbols and inserts control characters into the stream when needed. The control characters used in JESD204B are shown in Table 61. The 8-bit/10-bit encoding uses the same number of ones and zeros across multiple symbols to ensure that the signal is dc balanced.

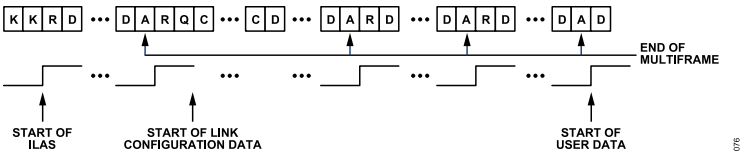

Figure 75. Initial Lane Alignment Sequence

Table 61. Control Characters Used in JESD204B

Abbreviation	Control Symbol	8-Bit Value	10-Bit Value, RD[1] = −1	10-Bit Value, RD[1] = +1	Description
/R/	/K28.0/	000 11100	001111 0100	110000 1011	Start of multiframe
/A/	/K28.3/	011 11100	001111 0011	110000 1100	Lane alignment
/Q/	/K28.4/	100 11100	001111 0100	110000 1101	Start of link configuration data
/K/	/K28.5/	101 11100	001111 1010	110000 0101	Group synchronization
/F/	/K28.7/	111 11100	001111 1000	110000 0111	Frame alignment

[1] RD means running disparity.

JESD204B (8-Bit/10-Bit) Link Establishment

The device's JESD204B transmitter interface can operate in subclass 0 or subclass 1, as defined in JESD204B. The link establishment process is divided into the following steps: code group synchronization (CGS), ILAS, and user data and error correction.

CGS

The CGS is the process by which the JESD204B receiver finds the boundaries between the 10-bit symbols in the stream of data. During the CGS phase, the JESD204B transmit block transmits /K28.5/ characters. The receiver must locate /K28.5/ characters in the input data stream using CDR techniques.

The receiver asserts the SYNCINB_xx± pin of the device low to issue a synchronization request, and then the JESD204B transmitter begins sending /K/ symbols. The receiver is synchronized once it properly decodes at least four consecutive /K/ symbols and then the SYNCINB_xx± pin is de-asserted. The device transmits an ILAS on the following LMFC boundary.

ILAS

The ILAS phase follows the CGS phase and begins on the next LMFC boundary. The ILAS consists of four multiframes with an /R/ character marking the beginning and an /A/ character marking the end. The ILAS first sends an /R/ character followed by 0 to 255 ramp data for one multiframe. On the second multiframe, the link configuration data is sent starting with the third character. The second character is a /Q/ character to confirm that the link configuration data follows. All undefined data slots are filled with ramp data. The ILAS sequence is never scrambled.

The ILAS sequence construction is shown in Figure 75. The four multiframes include the following:

- Multiframe 1: begins with an /R/ character (/K28.0/) and ends with an /A/ character (/K28.3/).
- Multiframe 2: begins with an /R/ character followed by a /Q/ character (/K28.4/), followed by link configuration parameters over 14 configuration octets (see Table 61) and then ends with an /A/ character. Many parameter values are of the value – 1 notation.
- Multiframe 3: begins with an /R/ character (/K28.0/) and ends with an /A/ character (/K28.3/).
- Multiframe 4: begins with an /R/ character (/K28.0/) and ends with an /A/ character (/K28.3/).

APOLLO MXFE RECEIVER

User Data and Error Detection

When the ILAS is complete, the user data is sent. Typically, all characters are considered user data within a frame. However, to monitor the frame clock and multiframe clock synchronization, there is a mechanism that replaces characters with /F/ or /A/ alignment characters when the data meets certain conditions. These conditions are different for unscrambled and scrambled data. The scrambling operation is enabled by default but can be disabled by setting the jesd_scr parameter to 0. The scr parameter is part of the **adi_cms_jesd_param_t** structure and is stared in the device profile. It is enabled by default (jesd_scr = 1).

8b/10b Character Replacement

For scrambled data, any 0xFC character at the end of a frame is replaced by an /F/, and any 0x7C character at the end of a multiframe is replaced with an /A/. The JESD204B receiver checks for /F/ and /A/ characters in the received data stream and verifies that these characters only occur in the expected locations. If an unexpected /F/ or /A/ character is found, the receiver handles the situation using dynamic realignment or by asserting the SYNCINB_xx± signal for more than four frames to initiate a resynchronization. For unscrambled data, if the final character of two subsequent frames are equal, the second character is replaced with an /F/ if the character is at the end of a frame and an /A/ if the character is at the end of a multiframe.

There are some optional settings for the JESD204B link layer that can be set within the device profile. One of these, the frame alignment character insertion (FACI), is enabled by default. The user can also program the device to insert multiframe characters (/A/, K28.3) as well by setting the tx_dl_204b_lsync_en_cfg parameter to 1 using the TBD API function By default, the multiframe character insertion is disabled.

JESD204C TRANSMITTER 64-BIT/66-BIT LINK LAYER

When using the 64-bit/66-bit link layer, scrambling is always enabled. As described in Table 30, the only synchronization word function beyond the EoEMB function is the 12-bit CRC function (CRC-12). The CRC-12 function is always enabled and carries the 12-bit CRC value of the data transmitted during the previous multiblock.

The ability to reverse the bit order of the 12-bit CRC value is available in Apollo using the metaword_rev parameter in the **adi_apollo_jesd_tx_link_cfg_t** structure of the device profile. This parameter is described in Table 62. Scrambling when using the 64-bit/66-bit link layer is performed on a per block per lane basis. All 8 octets of each lane block are scrambled (64 bits of scrambled data followed by an unscrambled synchronization header).

For details on the 64-bit/66-bit Link Layer Synchronization, refer to the JESD204C Receiver 64-bit/66-bit Link Layer and Synchronization section.

Table 62. JESD204C CRC Option Parameter

Parameter	Description	Enumerations	Comments
metaword_rev	Reverses the order of the CRC/FEC meta word	TRUE	Reverse the bit order
		FALSE	Don't reverse the order

RECEIVE MUX4 (JESD204B/C TRANSMITTER CROSSBAR)

Receive Mux4 is the JESD204B/C transmitter data crossbar multiplexer that maps a specified logical lane to one of the 12 available JESD204B/C transmitter physical lanes. Each logical lane has to be mapped to a separate physical lane (PHY_lane[*] in diagram below) for both links to be active. Only the first L logical lanes will be active for each link. These can be mapped to any of the physical lanes. Care should be taken so that 2 or more logical lanes are not mapped to same physical lane. If in dual link mode, only modes with 6 lanes or fewer and 8 virtual converters or fewer (M≤8 and L≤6) are supported on link1. The total number of lanes uses must be ≤12. In dual link mode, any of the 12 physical lanes can transmit data from any of the logical lanes from either link.

Unused lanes may be left at their default values. For example, if link0 has L=6 and link1 has L=3, logical lanes 0-5 of link0 are active and logical lanes 0-2 of link1 are active. All other logical lanes are inactive. The example shown in Figure 76 illustrates this configuration where link0 uses PHY lanes 1, 2, 4, 5, 0, and 8 in that order (routing shown in green). Link1 logical lanes are routed to PHY lanes 6, 9, and 3 in that order (routing shown in blue). The unused PHY lanes (7, 10, and 11) should be powered down. This function is taken care of internally based on the L parameter setting and the mux4 settings.

APOLLO MXFE RECEIVER

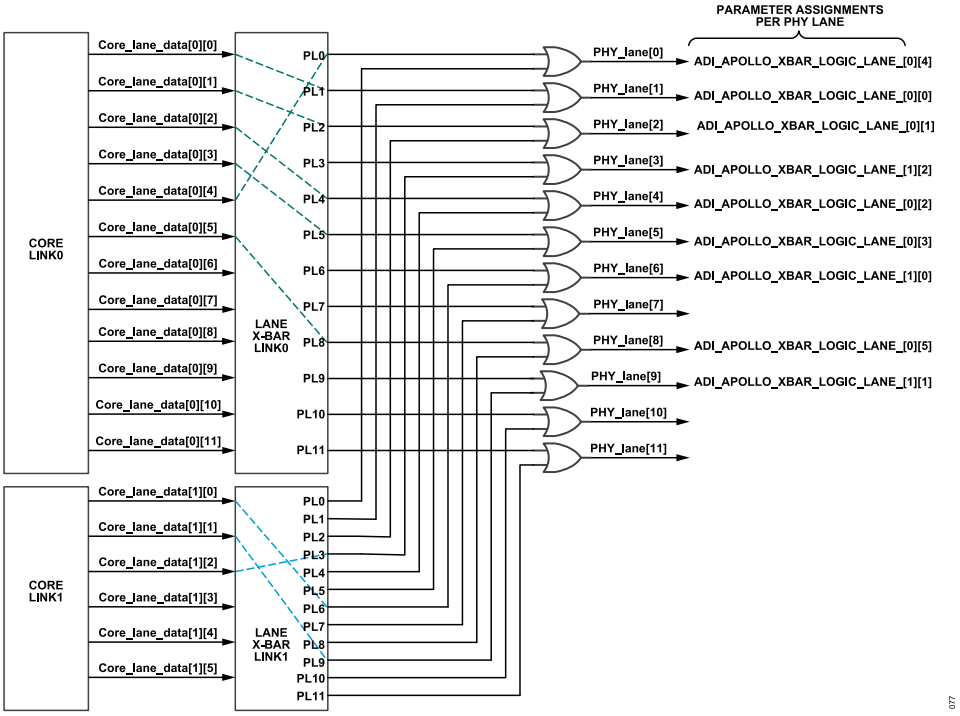

Figure 76. Dual Link JESD204B/C Transmit Crossbar Routing Example

Receive MUX4 Configuration

The device profile contains the Receive MUX4 configuration information in the **adi_apollo_jesd_tx_link_cfg_t** structure. The parameter and enumeration are described in Table 63. Any unused PHY lanes will be powered down. In dual link mode, these assignments must be made for each link. Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 63. JESD204C Transmitter Crossbar Parameter per PHY lane

Parameter	Description	Enumerations	Comments
lane_xbar	Index that sets the source logical lane for each of the physical lanes	ADI_APOLLO_XBAR_LOGIC_LANE_[0..11]	0= logical lane0, etc.

JESD204B/C TRANSMITTER PHYSICAL LAYER

As shown in Figure 77, the physical layer consists of the lane rate adaptor, serializer, feed forward equalization (FFE), and the output driver. In this layer, parallel data is converted into up to 12 lanes of differential serial data per A/B-side. The differential digital outputs are powered up by default. The drivers use a dynamic, 100 Ω internal termination to reduce unwanted reflections.

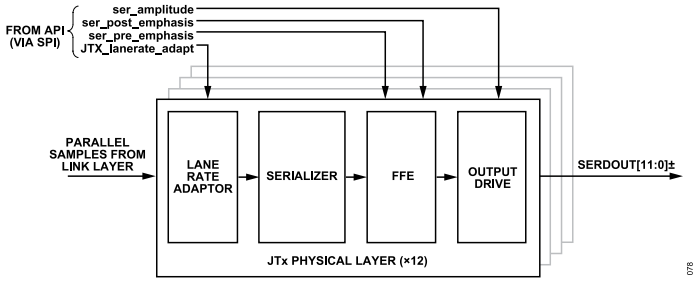

Figure 77. JESD204B/C Transmit Physical Layer Block Diagram

APOLLO MXFE RECEIVER

JESD204B/C Transmitter PHY Configuration

Configuration of the JESD204B/C Transmitter PHY is performed by the on-chip firmware using the appropriate parameters from the device profile as found in the **adi_apollo_jesd_ser_lane_t** structure. Table 64 lists the parameters set by firmware.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 64. JESD204B/C Transmitter PHY Parameters (per lane)

Parameter	Description	Enumerations	Comments
lane	JTx PHY lane number	Lane number to which the settings apply	
ser_amplitude	JTx PHY lane output swing level	ADI_APOLLO_JESD_DRIVE_SWING_VTT_100	1.0*VTT
		ADI_APOLLO_JESD_DRIVE_SWING_VTT_85	0.850*VTT
		ADI_APOLLO_JESD_DRIVE_SWING_VTT_75	0.750*VTT
		ADI_APOLLO_JESD_DRIVE_SWING_VTT_50	0.500*VTT
ser_pre_emphasis	JTx PHY lane pre-emphasis	ADI_APOLLO_JESD_PRE_TAP_LEVEL_0_DB	0dB
		ADI_APOLLO_JESD_PRE_TAP_LEVEL_3_DB	3dB
		ADI_APOLLO_JESD_PRE_TAP_LEVEL_6_DB	6db
ser_post_emphasis	JTx PHY lane post-emphasis	ADI_APOLLO_JESD_POST_TAP_LEVEL_0_DB	0dB
		ADI_APOLLO_JESD_POST_TAP_LEVEL_3_DB	3dB
		ADI_APOLLO_JESD_POST_TAP_LEVEL_6_DB	6dB
		ADI_APOLLO_JESD_POST_TAP_LEVEL_9_DB	9dB
		ADI_APOLLO_JESD_POST_TAP_LEVEL_12_DB	12dB
ser_invert_lane_polarity	Serializer lane PN inversion select	boolean	TRUE = invert
			FALSE = don't invert
lanerate_adapt	Lane rate adaptation value	boolean	Valid settings are 1, 2, 4, 8, 16, or 32

JESD204B/C Transmitter Lane Rate Adaptation

There are some conditions that require enabling the lane rate adaptation feature. Lane rate adaptation automatically adjusts the clock rate of the serializer to support the device profile use case while still operating within the device operating parameters. For example, since the minimum clock rate of the Apollo serializer is 4.0625Gbps, lane rate adaptation is required to achieve an effective lane rate of 1.015625Gbps. This is one use case where lane rate adaptation is required. That is, if a lane rate of < 4.0625Gbps is desired. Since all lane rates are derived from the same PLL, lane rate adaptation is also needed if the links in a dual link mode of operation have different lane rates. Lane rate adaptation is also required if the JESD204B/C transmitter and receiver require different lane rates. In both cases, the lane rate differences must be a power of 2 multiple of each other.

Lane rate adaptation is set as needed in the device profile using the lanerate_adapt parameter that is part of the **adi_apollo_jesd_ser_lane_t** structure and also described in Table 64.

FFE

The FFE consists of both pre-tap and post-tap de-emphasis and can enable the receiver eye diagram mask to be met in conditions where the interconnect insertion loss does not meet the JESD204B specification, for which the eye mask specification applies. It also can assist in opening up the eye at the JESD204B/C receiver for the higher lane rates supported by JESD204C.

Only use FFE settings that enable de-emphasis when the receiver is unable to recover the clock because of excessive insertion loss. For links with well-designed PCB channels that have low insertion loss, disable de-emphasis to conserve power.

Use nonzero de-emphasis settings with caution because these settings can increase electromagnetic interference (EMI).

Relevant parameters (ser_pre_emphasis and ser_post_emphasis) for configuring the FFE are described in Table 64.

APOLLO MXFE RECEIVER

Output Driver

The amplitude of the JESD204B/C transmitter output is set relative to the VTT supply. The ser_amplitude parameter is used to set this value as indicated in Table 64.

DIGITAL OUTPUTS, TIMING, AND CONTROLS

Place a 100 Ω differential termination resistor at each receiver input to achieve a nominal 0.85×DRVDD1 V p-p swing at the receiver, as shown in Figure 78. The swing is adjustable using the ser_amplitude parameter which is part of the **adi_apollo_jesd_ser_lane_t** structure described in Table 64. AC coupling is recommended to connect to the receiver.

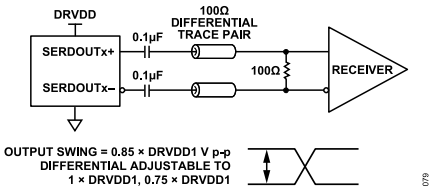

Figure 78. AC-Coupled Digital Output Termination Example

The device digital outputs can interface with custom application specific integrated circuits (ASICs) and field programmable gate array (FPGA) receivers and provide superior switching performance in noisy environments. Single point to point network topologies are recommended with a single differential 100 Ω termination resistor placed as close to the receiver inputs as possible.

RECEIVE PATH DETERMINISTIC LATENCY

JESD204B/C links contain various clock domains distributed throughout each system. Data traversing from one clock domain to a different clock domain can lead to ambiguous delays in the JESD204B/C link. These ambiguities lead to non-repeatable latencies across the link from one power cycle or link reset to the next. The Apollo MxFE JESD204B/C transmitter supports JESD204B/C Subclass 0 (deterministiclatency across power cycles not required) and Subclass 1 operation (using SYSREF to ensure deterministic latency across power cycles).

Subclass 0 Operation

If there is no requirement for multi-chip synchronization while operating in Subclass 0 mode, the SYSREF input can be left disconnected. Even so, it is still required to program the internal SYSREF according to the information provided in the Subclass 0 Synchronization.

Subclass 1 Operation

The JESD204B/C protocol organizes data samples into octets, frames, and multiframes (or multiblocks), as described in the JESD204B/C Transmitter Transport Layer section. The LMFC/LEMC is synchronous with the beginning of these multiframes/multiblocks. In Subclass 1 operation, the internal SYSREF, and therefore, the LMFC/LEMCs for each device are aligned to the external SYSREF. This is necessary across all links in all the devices within the system that need to be synchronized. Within the Apollo MxFE Device, the SYSREF signal also synchronizes the internal sample dividers. This synchronization is shown in Figure 79. The JESD204B receiver uses the multiframe boundaries and buffering to achieve consistent latency across lanes (or across multiple devices) and to achieve a fixed latency across the power cycles and link reset conditions. Similarly, the JESD204C receiver uses the extended multiblock boundaries and buffering to achieve consistent latency.

Deterministic Latency Requirements

Key factors that are required for achieving deterministic latency in a JESD204B/C Subclass 1 system include the following:

- SYSREF signal distribution skew within the system must be less than the desired uncertainty for the system.
- SYSREF setup and hold time requirements must be met for each device in the system. When using averaged SYSREF mode, the setup and hold time requirements are eased for the externally applied SYSREF signal. Refer to Hardware Based SYSREF Alignmentfor the details.
- The total latency variation from one power cycle to another across all lanes, links, and devices must not exceed the lane deskewing capability of the JESD204B/C receiver. For the Apollo MxFE this limit is three PCLK cycles. (120 UI for JESD204B and 198 UI for JESD204C) (see Figure 79). This variation includes both the variable delays and the fixed delays from lane to lane, link to link, and device

APOLLO MXFE RECEIVER

to device in the system. For more requirements regarding latency variation related to the device JESD204B/C receiver, see the Deterministic Latency Requirements section.

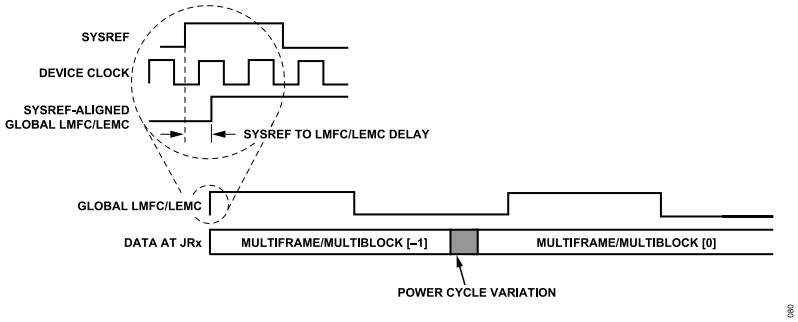

Figure 79. SYSREF and LMFC/LEMC

Deterministic Latency Programmability Options in JESD204B/C Transmitter

The JESD204B/C receive buffer in the logic device buffers data. If the total link latency in the system is near an integer multiple of the LMFC/LEMC period, the data arrival time of the incoming LMFC/LEMC boundary at the receive buffer can straddle the JESD204B/C receiver's local LMFC/LEMC boundary from one power cycle to the next. To ensure deterministic latency in this case, perform a phase adjustment of the LMFC/LEMC at either the JESD204B/C transmitter or JESD204B/C receiver. Typically, adjustments to accommodate the receive buffer are made to the receiver LMFC/LEMC. In the JESD204B/C transmitter of Apollo, this adjustment can be made by passing the **phase_adjust** parameter to the **adi_apollo_jtx_phase_adjust_set()** function. This value is 0 by default. Typically, this parameter, if used, is called phase_adjust and will be stored in the device profile (as part of the **adi_apollo_jesd_tx_link_cfg_t** structure) and set during device startup. The step size for this adjustment is in the jtx_conv_clk cycles, where

- jtx_conv_clk = f_{ADC}/(DCM × NS)
- f_{ADC} = ADC sample clock
- DCM = total decimation (parameter name link_total_ratio)
- NS = number of samples processed per the jtx_conv_clk in the data path (parameter name jtx_ns_cfg). This value is auto-calculated by Apollo based on the JESD204B/C mode and the total decimation.
- For cases where FSRC is enabled and link_total_ratio = min(link_ddc_dec across all configs), the jtx_ns_override parameter is set and the computed jtx_ns_cfg value is overridden by Apollo.

Figure 80 shows that when the link latency is near an LMFC/LEMC boundary, the local LMFC/LEMC of Apollo can be delayed allowing all instances of the data arrival time at the receiver to occur within the same LMFC/LEMC cycle. Figure 81 shows how a delay of the LMFC/LEMC in the receiver accommodates the receive buffer timing. Consult the applicable JESD204B/C receiver user guide of the logic device being used for details on making this adjustment. If the total latency in the system is not near an integer multiple of the LMFC/LEMC period or if the appropriate adjustments have been made to the LMFC/LEMC phase at the clock source, variable latency from one power cycle to the next is still possible.

Tips and tools for ensuring deterministic latency can be achieved:

- If the hardware SYSREF alignment method is used, check for the possibility that the setup and hold time requirements for the SYSREF signal are not met. Refer to the SYSREF Input and Setup/Hold Detector section.
  - If the SYSREF setup and hold (SH) detector indicates a potential timing problem, the phase of the SYSREF signal supplied to Apollo must be adjusted until the sh_before and sh_after parameters indicate that there is no potential timing problem.
- Regardless of alignment method, SYSREF_PHASE can be read to determine if SYSREF alignment has been achieved. This can be done by calling the **adi_apollo_clk_mcs_sysref_phase_get()** function.
  - This read indicates the phase relationship between the incoming SYSREF signal relative to the internal SYSREF and the LMFC/LEMC boundaries.
  - A readback of 0 indicates that internal SYSREF (and LMFC/LEMC) are aligned to the external SYSREF. A nonzero value such as 10, for example, indicates that the SYSREF rising edge is 10 MCLK cycles later than the internal SYSREF.

APOLLO MXFE RECEIVER

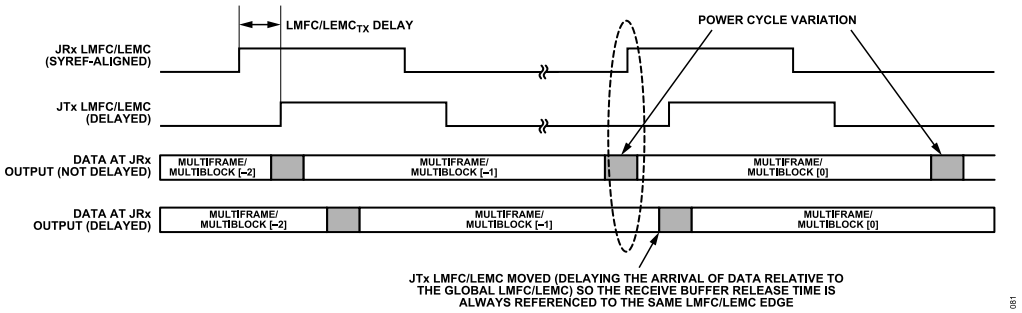

Figure 80. Adjusting the JESD204B/C Transmitter LMFC/LEMC in the AD9084 and AD90888

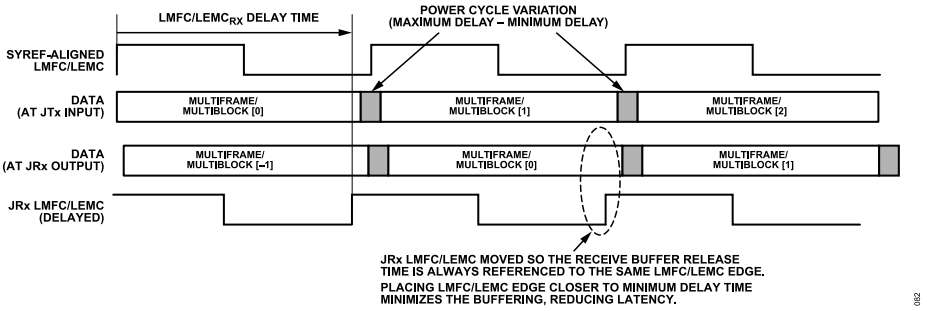

Figure 81. Adjusting the JESD204B/C Receiver LMFC/LEMC in the Logic Device

JESD204B/C TRANSMITTER MODE TABLE

The device JESD transmitter (ADC path) supports many JESD204B and JESD204C modes, as described in Table 65. The Fine x Coarse decimation modes supported per JESD204B/C mode number depends on whether the device is operating in 204B or 204C mode as indicated in Table 66. Decimation modes supported by JESD204B for a given JESD204 receiver mode are marked with a “B”. Decimation modes supported by JESD204C for a given JESD204 receiver mode are marked with a “C”. If the decimation mode is supported by both 204B and 204C, it is marked with “B, C”.

Table 65. ADC Path Supported JESD204B/C Modes per A and B-sides

JESD204 Mode Number	L	M	F	S	N’	K (204B)	K (204C)	E (204C)	Dual link support per A/B	Mode Support Conditions	
										8T8R:AD9088	4T4R:AD9084
0	1	16	32	1	16	32	8	1	NO	when decimation Ratio>2x	Not Supported
1	1	16	24	1	12	32	32	3	NO	when decimation Ratio>2x	Not Supported
2	1	8	16	1	16	32	16	1	YES	when decimation Ratio>1x	when decimation Ratio>2x
3	1	8	12	1	12	32	64	3	YES	when decimation Ratio>1x	when decimation Ratio>2x
4	1	4	8	1	16	32	32	1	YES	No restrictions	when decimation Ratio>1x
5	1	4	6	1	12	32	128	3	YES	No restrictions	when decimation Ratio>1x
6	1	2	4	1	16	32	64	1	YES	No restrictions	No restrictions
7	2	16	16	1	16	32	16	1	NO	when decimation Ratio>2x	Not Supported
8	2	16	12	1	12	32	64	3	NO	when decimation Ratio>2x	Not Supported
9	2	8	8	1	16	32	32	1	YES	when decimation Ratio>1x	when decimation Ratio>2x
10	2	8	6	1	12	32	128	3	YES	when decimation Ratio>1x	when decimation Ratio>2x
11	2	4	4	1	16	32	64	1	YES	No restrictions	when decimation Ratio>1x
12	2	4	3	1	12	32	256	3	YES	No restrictions	when decimation Ratio>1x
13	2	2	2	1	16	32	128	1	YES	No restrictions	No restrictions
14	2	1	1	1	16	32	256	1	YES	when decimation Ratio=1x	when decimation Ratio=1x
15	3	12	8	1	16	32	32	1	NO	when decimation Ratio>2x	Not Supported

APOLLO MXFE RECEIVER

Table 65. ADC Path Supported JESD204B/C Modes per A and B-sides (Continued)

JESD204 Mode Number	L	M	F	S	N'	K (204B)	K (204C)	E (204C)	Dual link support per A/B	Mode Support Conditions	
										8T8R:AD9088	4T4R:AD9084
16	3	8	4	1	12	32	64	1	YES	when decimation Ratio>1x	when decimation Ratio>2x
17	3	6	4	1	16	32	64	1	YES	when decimation Ratio>1x	when decimation Ratio>2x
18	3	4	2	1	12	32	128	1	YES	No restrictions	when decimation Ratio>1x
19	3	3	2	1	16	32	128	1	YES	when decimation Ratio=1x	Not Supported
20	3	2	1	1	12	32	256	1	YES	No restrictions	No restrictions
21	3	1	1	2	12	32	256	1	YES	when decimation Ratio=1x	when decimation Ratio=1x
22	4	16	8	1	16	32	32	1	NO	when decimation Ratio>2x	Not Supported
23	4	16	6	1	12	32	128	3	NO	when decimation Ratio>2x	Not Supported
24	4	8	4	1	16	32	64	1	YES	when decimation Ratio>1x	when decimation Ratio>2x
25	4	8	3	1	12	32	256	3	YES	when decimation Ratio>1x	when decimation Ratio>2x
26	4	4	2	1	16	32	128	1	YES	No restrictions	when decimation Ratio>1x
27	4	4	3	2	12	32	256	3	YES	No restrictions	when decimation Ratio>1x
28	4	4	1	1	8	32	256	1	YES	No restrictions	when decimation Ratio>1x
29	4	2	1	1	16	32	256	1	YES	No restrictions	No restrictions
30	4	2	3	4	12	32	256	3	YES	No restrictions	No restrictions
31	4	2	1	2	8	32	256	1	YES	No restrictions	No restrictions
32	4	1	1	2	16	32	256	1	YES	when decimation Ratio=1x	when decimation Ratio=1x
33	4	1	3	8	12	32	256	3	YES	when decimation Ratio=1x	when decimation Ratio=1x
34	4	1	1	4	8	32	256	1	YES	when decimation Ratio=1x	when decimation Ratio=1x
35	6	12	4	1	16	32	64	1	NO	when decimation Ratio>2x	Not Supported
36	6	8	2	1	12	32	128	1	YES	when decimation Ratio>1x	when decimation Ratio>2x
37	6	6	2	1	16	32	128	1	YES	when decimation Ratio>1x	when decimation Ratio>2x
38	6	4	1	1	12	32	256	1	YES	No restrictions	when decimation Ratio>1x
39	6	3	3	4	12	32	256	3	YES	when decimation Ratio=1x	Not Supported
40	6	3	1	1	16	32	256	1	YES	when decimation Ratio=1x	Not Supported
41	6	2	1	2	12	32	256	1	YES	No restrictions	No restrictions
42	6	1	1	4	12	32	256	1	YES	when decimation Ratio=1x	when decimation Ratio=1x
43	8	16	4	1	16	32	64	1	NO	when decimation Ratio>2x	Not Supported
44	8	16	3	1	12	32	256	3	NO	when decimation Ratio>2x	Not Supported
45	8	8	2	1	16	32	128	1	NO	when decimation Ratio>1x	when decimation Ratio>2x
46	8	8	3	2	12	32	256	3	NO	when decimation Ratio>1x	when decimation Ratio>2x
47	8	4	1	1	16	32	256	1	NO	No restrictions	when decimation Ratio>1x
48	8	4	3	4	12	32	256	3	NO	No restrictions	when decimation Ratio>1x
49	8	4	1	2	8	32	256	1	NO	No restrictions	when decimation Ratio>1x
50	8	2	1	2	16	32	256	1	NO	No restrictions	No restrictions
51	8	2	3	8	12	32	256	3	NO	when decimation Ratio = 1x	No restrictions
52	8	2	1	4	8	32	256	1	NO	No restrictions	No restrictions
53	8	1	1	4	16	32	256	1	NO	when decimation Ratio=1x	when decimation Ratio=1x
54	8	1	3	16	12	32	256	3	NO	when decimation Ratio=1x	when decimation Ratio=1x
55	8	1	1	8	8	32	256	1	NO	when decimation Ratio=1x	when decimation Ratio=1x
56	12	16	2	1	12	32	128	1	NO	when decimation Ratio>2x	Not Supported
57	12	12	2	1	16	32	128	1	NO	when decimation Ratio>2x	Not Supported
58	12	12	3	2	12	32	256	3	NO	when decimation Ratio>2x	Not Supported
59	12	8	1	1	12	32	256	1	NO	when decimation Ratio>1x	when decimation Ratio>2x
60	12	6	1	1	16	32	256	1	NO	when decimation Ratio>1x	when decimation Ratio>2x

APOLLO MXFE RECEIVER

Table 65. ADC Path Supported JESD204B/C Modes per A and B-sides (Continued)

JESD204 Mode Number	L	M	F	S	N'	K (204B)	K (204C)	E (204C)	Dual link support per A/B	Mode Support Conditions	
										8T8R:AD9088	4T4R:AD9084
61	12	4	1	2	12	32	256	1	NO	No restrictions	when decimation Ratio>1x
62	12	4	1	3	8	32	256	1	NO	when decimation Ratio=1x	Not Supported
63	12	3	1	2	16	32	256	1	NO	when decimation Ratio=1x	Not Supported
64	12	3	3	8	12	32	256	3	NO	when decimation Ratio=1x	Not Supported
65	12	3	1	4	8	32	256	1	NO	when decimation Ratio=1x	Not Supported
66	12	2	1	4	12	32	256	1	NO	No restrictions	No restrictions
67	12	2	1	6	8	32	256	1	NO	when decimation Ratio = 1x	when decimation Ratio = 1x
68	12	1	1	8	12	32	256	1	NO	when decimation Ratio=1x	when decimation Ratio=1x
69	12	1	1	12	8	32	256	1	NO	when decimation Ratio=1x	when decimation Ratio=1x
70	4	4	2	2	8	32	128	1	YES	only 204B and 3x	only 204B and 3x
71	4	2	2	2	16	32	128	1	YES	only 204B and 3x	only 204B and 3x
72	6	4	2	2	12	32	128	1	YES	only 204B and 3x	only 204B and 3x
73	8	4	2	2	16	32	128	1	NO	only 204B and 3x	only 204B and 3x
74	12	8	2	2	12	32	128	1	NO	only 204B and 3x	only 204B and 3x
75	12	6	2	2	16	32	128	1	NO	only 204B and 3x	only 204B and 3x
76	12	1	1	6	16	32	256	1	NO	only 204C and 1x	only 204C and 1x
77	12	2	1	3	16	32	256	1	NO	only 204C and 1x/2x	only 204C and 1x/2x

Table 66. Supported Decimation Modes per JESD204B/C Modes

Total Decimation	1	2	3	4	6	8	12	16	24	32	48	64	96	128	192	256	384	768
	Fine x Coarse Decimation Modes																	
JESD204 Mode Number	1x1	1x2	1x3	1x4 2x2	1x6 2x3	2x4 4x2	1x12 2x6 4x3	4x4 8x2	2x12 4x6 8x3	8x4 16x2	4x12 8x6 16x3	16x4 32x2	8x12 16x6 32x3	32x4 64x2	16x12 32x6 64x3	64x4	32x12 64x6	64x12
0									C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C
1									C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C
2								C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C
3							C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B
4					C	C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B
5					C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B
6			C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	
7								C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C
8							C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B
9					C	C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B
10					C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B
11			C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	
12			C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	
13		C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B		
14	C																	
15					C	C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B
16			C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	
17			C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	
18		C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B		
19	C																	
20	C																	

APOLLO MXFE RECEIVER

Table 66. Supported Decimation Modes per JESD204B/C Modes (Continued)

Total Decimation	1	2	3	4	6	8	12	16	24	32	48	64	96	128	192	256	384	768
	Fine x Coarse Decimation Modes																	
JESD204 Mode Number	1x1	1x2	1x3	1x4 2x2	1x6 2x3	2x4 4x2	1x12 2x6 4x3	4x4 8x2	2x12 4x6 8x3	8x4 16x2	4x12 8x6 16x3	16x4 32x2	8x12 16x6 32x3	32x4 64x2	16x12 32x6 64x3	64x4	32x12 64x6	64x12
21	C																	
22					C	C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B
23					C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B
24			C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	
25			C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	
26		C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B		
27		C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B		
28	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B				
29	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B				
30	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B					
31	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B						
32	C																	
33	B, C																	
34	B, C																	
35			C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	
36		C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B		
37		C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B		
38	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B				
39	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B					
40	C																	
41	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B						
42	B, C																	
43			C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	
44			C	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	
45		C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B		
46		C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B		
47	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B				
48	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B					
49	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B						
50	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B						
51	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B						
52	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B								
53	B, C																	
54	B, C																	
55	B, C																	
56		C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B		
57		C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B		
58		C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B		
59	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B				
60	C	C	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B				
61	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B						
62	B, C																	

APOLLO MXFE RECEIVER

Table 66. Supported Decimation Modes per JESD204B/C Modes (Continued)

Total Decimation	1	2	3	4	6	8	12	16	24	32	48	64	96	128	192	256	384	768
	Fine x Coarse Decimation Modes																	
JESD204 Mode Number	1x1	1x2	1x3	1x4 2x2	1x6 2x3	2x4 4x2	1x12 2x6 4x3	4x4 8x2	2x12 4x6 8x3	8x4 16x2	4x12 8x6 16x3	16x4 32x2	8x12 16x6 32x3	32x4 64x2	16x12 32x6 64x3	64x4	32x12 64x6	64x12
63	C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B						
64	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B	B						
65	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B								
66	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B, C	B	B								
67	B, C																	
68	B, C																	
69	B, C																	
70			B															
71			B															
72			B															
73			B															
74			B															
75			B															
76	C																	
77	C	C																

APOLLO MXFE TRANSMITTER

JESD204B/C RECEIVER

Twenty-Four JESD204B/C receive (JRx) data lanes (twelve lanes per A/B-sides) are available to receive the input sample data sent to the device. The twelve JESD204B/C lanes within A/B-sides can be combined to form either one (single-link) or two (dual-link) identical links.

Each link can provide data to an individual data path, each with a unique set of FDUCs. Both single-link and dual-link JESD204B/JESD204C modes align the individual (local) clocks to the same system reference (SYSREF_x_P/N) and device clock CLK_x_N/P) signals.

When operating with the 8-bit/10-bit link layer (JESD204B enabled) the SYNCOUTB_Ax and SYNCOUTB_Bx signals are specific to the respective JESD204B link, and in dual-link mode (Register 0x0596, Bit 3 = 1) the two links can operate independently from one another. For example, one link can be powered down while the other link is running. If the 8-bit/10-bit link layer option is selected, the link operation complies to both the JESD204B and JESD204C (8-bit/10-bit link layer) standards and the link lane rates can be between 5 Gbps and 20 Gbps.

Similarly, the two links can also operate independently from one another when operating with the 64-bit/66-bit link layer (JESD204C enabled) in dual-link mode. If the 64-bit/66-bit link layer option is selected, the link operation complies to the JESD204C standard, including the new synchronization process (SYNCOUTB_xx pin is not used) and the link lane rates can be between 1 Gbps and 28.21 Gbps.

The JESD204B/C serial interface hardware is grouped into three layers: the physical layer, the data link layer, and the transport layer. The functional block diagram of the JESD204B/JESD204C device receiver is shown in Figure 82.

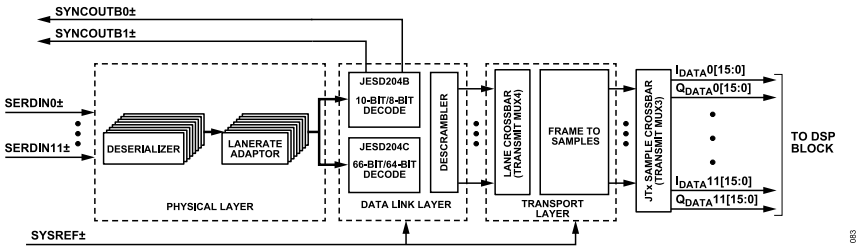

Figure 82. Functional Block Diagram of the JESD204B/C Receiver of A/B-sides

JESD204B/C Receiver Clock Relationships

The following clock rates are used throughout the rest of the JESD204B/C section. The relationship between any of the clocks can be derived from the following equations:

- Data Rate = DAC Rate/ Total Interpolation
- PCLK Factor = 4/F

For 8-bit/10-bit encoding:

- Lane Rate = (M/L) × NP × (10/8) × Data Rate
- PCLK Rate = Lane Rate/40

For 64-bit/66-bit encoding:

- Lane Rate = (M/L) × NP × (66/64) × Data Rate
- PCLK Rate = Lane Rate/66

The Data Rate is the rate at which data is sent to the JESD204B Receiver from the JESD204B Transmitter, in samples per second (SPS)

The Lane Rate, or the bitrate is the rate at which sample bits are sent across the physical lanes (SRXx_x pins)

PCLK Rate is the rate of the processing clock that is used for the quad-byte decoder (204B) or block clock (204C).

Configuring the JESD204B/C Receiver

JESD204B/C Receiver is configured using the appropriate parameters from the device profile after the device profile is loaded as part of the "Load Apollo Profile and Firmware Boot" step of the of the Figure 12. Most of the example code described in the ADI Evaluation System Software Examples section provide an example of how to use this function. The exceptions being any examples demonstrating loopbacks or NCO test modes:

APOLLO MXFE TRANSMITTER

The link is enabled via the **adi_apollo_jesd_rx_link_enable_set()** API command during the “Enable Apollo JESD Links” portion of the Figure 12.

The parameters for the JESD204B/C Receiver are contained in the device profile in a few different structures. They are listed per block function in the relevant subsections below along with any relevant API functions.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

JESD204B/C Receiver Physical Layer

JESD204B/C data is input to the device at the physical interface (referred to as the deserializer) via the SRXx_x differential input pins. The class C-M physical layer has 12 identical channels on each side (A/B) of the device. Each channel consists of the termination, a continuous time linear equalizer (CTLE), a clock and data recovery (CDR) circuit, and a demultiplexer function, as shown in Figure 83.

To optimize power and performance of the JESD204B/C receiver PHY, several DESER_CBUS registers must be written. These register writes are handled by device firmware using the setting found in the device profile.

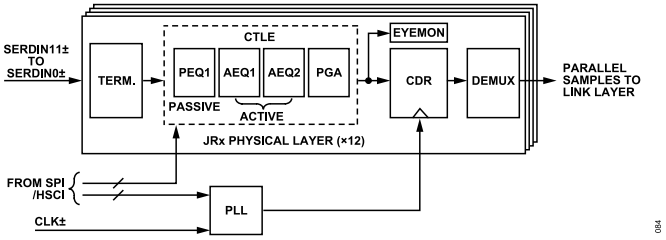

Figure 83. Deserializer Block Diagram

JESD204B/C Receiver PHY Modes

There are three modes of operation of the PHY based on the lane rate of the use case being employed. Full-rate mode is for lane rates below 8Gbps. Half-rate is for lane rates ≥ 8Gbps and below 16Gbps. Quarter-rate mode is for lane rates ≥ 16Gbps and up to the maximum lane rate (28.21 Gbps). Quarter-rate mode is not applicable when using JESD204B modes. The mode setting is determined by the rx_des_qhf_rate parameter in the **adi_apollo_jesd_deser_lane_t** structure of the device profile. The device firmware uses this information to configure the equalizer. The rx_des_qhf_rate parameter is described in Table 69.

Power Down Unused PHY Lanes

Any unused physical lanes (SRXx_x) that are left enabled consume extra power. Unused lanes are powered off by the Apollo MxFE based on the L parameter (number of lanes) in JESD204B/C Receiver mode as set by the device profile as well as the transmit mux4 configuration, The powered-on lanes are reflected by the lane_enables parameter in the device profile. This is a12-bit parameter where each bit represents the power state of its respective lane within the A and B-sides. A value of 4095 means all 12 PHY lanes are powered-on.

Inverting PHY Lane Data

It may be convenient for PCB layout purposes to swap the polarity on some of the differential SRXx_x inputs, which is equivalent to swapping the SRXx_x_P with the SRXx_x_N in the PCB layout for a particular lane. The des_inv_lane_pol parameter in the **adi_apollo_jesd_deser_lane_t** structure of the device profile is used for this purpose. This parameter is described in Table 69.

Very-Short Reach (VSR) Low-Power PHY

For low-loss PCB channels (max insertion loss 8dB) a low-power VSR interface option is available on the Apollo MxFE. The VSR is selected using the usr_sel parameter located in the device profile and is configured as part of the device bring-up. This feature will be enabled on a future release of the Apollo MxFE API (TBD). The VSR receiver PHY uses the same pins as the class C-M PHY at approximately one-half of the power usage. The enabling parameters for the VSR PHY are described in Table 69. Once enabled, the PHY is fully configured automatically by device. Preliminary JESD204 receiver PHY power numbers, based on design targets, are shown in Table 67.

Configurability options for the VSR interface are TBD and will be included in a future revision of the API.

APOLLO MXFE TRANSMITTER

Table 67. JESD204B/C Receiver PHY Power Consumption per Class (Preliminary)

JESD204 Receiver PHY Class	Lane Rate (Gbps)	Power consumption (pj/bit)
C-M (medium reach)	25	4.1
	32	4.9
VSR (Very-short reach)	25	2.12
	32	2.55

Equalizer (CTLE)

Figure 83 shows the four stages of the CTLE. The first stage is a passive equalizer (PEQ). The PEQ is configured to provide a filter with the inverse frequency response of the PCB channel. This conditions the signal and improves the performance of two active equalization stages (AEQ) that follow. The final stage is a programmable gain amplifier (PGA). The PGA is required since the DC gain of the equalization stages are less than one. The increased amplitude given by the PGA assists the Clock and Data Recovery (CDR) stage of the PHY recover the best phase for data capture.

Apollo MxFE device firmware does most of the heavy lifting for configuring the Equalizer without user interactions. When operating at full rate and half-rate PHY modes (lane rates below 16Gbps), there is no need for special user input or API commands. In full-rate mode, insertion loss up to 8dB can easily be compensated for. This equates to up to 20 inches of a low-loss FR4 channel. In half-rate mode, up to 20dB of insertion loss can be compensated for. When operating in quarter-rate mode (16-28.21Gbps), it is necessary to set the high_boost_options parameter based on the amount of insertion loss in the PCB channel and the lane rate range. The profile generator derives the high_boost_options based on the insertion loss of the PCB channels. The insertion loss is entered into the profile generator and the high_boost_options parameter is calculated and entered into the device profile. Table 68 lists the possible enumeration for the profile generator insertion loss input. Users will enter the enumeration that is appropriate for each of their instantiated JRx lanes based on their insertion loss into the profile generator tool. For insertion loss greater than 13dB, the FFE settings of the logic device will likely need to be increased. The JESD204B/C Receiver PHY Eye Scan section capability of the Apollo MxFE JRx can be used to optimize these settings.

Table 68. Enumeration for insertion loss (quarter-rate PHY mode only)

Enumeration	Description
IL_5db_or_less	Lane insertion loss is 5dB, or less
IL_5dB_to_9dB	Lane insertion loss is >5dB and ≤ 9dB
IL_9dB_to_13dB	Lane insertion loss is > 9dB
Default	Lane insertion loss is based on AD9084-FMCA-EBZ

JESD204B/C Receiver PHY Calibration

Once the JESD204B/C receiver link is enabled in half-rate or quarter-rate mode (lane rate ≥ 8Gbps), the Apollo MxFE will perform a foreground calibration of each active lane based on the CTLE settings and the electrical characteristics of the incoming data. In half-rate mode this calibration is fairly quick (< 1 second). In quarter-rate mode, the foreground calibration typically takes TBD seconds.

To ensure the highest integrity of the link over temperature when in quarter-rate mode, the Apollo MxFE also runs a background calibration. While the background calibrations are running, do not attempt to access register associated with the deserializer PHY. Before accessing these registers, first disable background calibrations, then access the deserializer PHY registers. After the PHY adjustments are made, re-enable the background calibrations. Typically, any PHY adjustments are made by the device or API where these precautions are already taken.

JESD204B/C Receiver PHY Warm-boot

If operating in quarter-rate mode, there may be an application requirement to reduce the Apollo MxFE boot-up time. For those use cases, the Apollo MxFE API will support a JESD204B/C Receiver PHY warm-boot procedure. This procedure will require the user to perform a one-time foreground calibration using the TBD function which will store the calibration values in the TBD file. This will allow the user, on subsequent start-ups to use the TBD function to recall the previously stored calibration values at start-up to achieve up to msec of start-up time savings. This feature will be supported in a future revision of the Apollo MxFE API.

JESD204B/C Receiver Lane Rate Adaptation

There are some conditions that require enabling the lane rate adaptation feature. Lane rate adaptation automatically adjusts the clock rate of the deserializer to support the device profile use case while still operating within the device operating parameters. For example, since the minimum lane rate of the Apollo MxFE deserializer is 4.0625Gbps, lane rate adaptation is required to achieve an effective lane rate of

APOLLO MXFE TRANSMITTER

1.015625Gbps. This is one use case where lane rate adaptation is required. That is, if a lane rate of < 4.0625Gbps is desired. Since all lane rates are derived from the same PLL, lane rate adaptation is also needed if the links in a dual link mode of operation have different lane rates. Lane rate adaptation is also required if the JESD204B/C transmitter and receiver require different lane rates. In both cases, the lane rate differences must be a power of 2 multiple of each other.

Lane rate adaptation is set as needed in the device profile using the jrx_lr_adapt parameter as described in Table 69.

Table 69. JESD204B/C Receiver PHY Configuration Parameters per PHY lane

Parameter	Description	Enumerations	Comments
rx_des_qhf_rate	Sets the JESD204B/C Receiver PHY Mode based on lane rate.	ADI_APOLLO_DESER_RATE_MODE_FULL_RATE ADI_APOLLO_DESER_RATE_MODE_HALF_RATE ADI_APOLLO_DESER_RATE_MODE_QUARTER_RATE	Lane rate < 8Gbps 8Gbps ≤ Lane rate < 16Gbps Lane rate ≥ 16Gbps
des_inv_lane_pol	Inverts the P/N lane polarity	Boolean	TRUE = invert FALSE = don't invert
JesdRxInsertionLoss	Sets CTLE parameters for QR mode (See Table 68)	Default Low_5dB_or_less Medium_5dB_to_9dB High_9dB_to_12dB	Loss based on typical ADI EVB Less than 5dB Greater then 5dB less then 9dB Greater than 9dB less than 12dB
jrx_lr_adapt	Lane rate adaptation value	Boolean	Valid settings are 1, 2, 4
usr_sel	Enable USR digital logic.	Boolean	FALSE = Main PHY digital path select TRUE = USR PHY digital path selected
usr_slice[1:0]_pd	USR Slice [1:0] power down	Boolean	FALSE = USR slice on TRUE = USR slice off
usr_pd	USR PHY power down.	Boolean	FALSE = USR PHY on TRUE = USR PHY off

Dual Link Support

All modes in the JESD204B/C Receiver Mode Tables are supported for link 0. Modes with M<=8 and L<=6, are also supported for dual link configurations (both link 0 and link 1 support these modes). The Table 81 table indicates dual link mode support with a "yes" in the "dual link" column of the table. Dual link mode is enabled using the dual_link parameter as defined in the **adi_apollo_jesd_common_cfg_t** structure which is part if the device profile.

In a dual link configuration (dual_link = TRUE), the lane rate for one link should be a power of 2 multiple of the other. This is required since the SERDES PLL runs at same lane rate for all links (links A[0:1] and B[0:1]). If one lane rate is a power of 2 of the other, then lane rate adaptation can be used to ensure the effective bit rate is the same. Refer to the JESD204B/C Receiver PHY Calibration section for details. The total number of lanes used, whether in single link or dual link mode, cannot exceed 12. Also note that the total number of virtual converters (M_link0+ M_link1) must be less than max value of M for the device configuration.

Link 0 must be used when in single link mode and link 1 must be disabled.

Transmit Mux4 (JESD204B/C Receiver Lane Crossbar)

Transmit Mux4 is the JESD204B/C receiver data crossbar multiplexer that maps a specified logical lane to one of the 12 available JESD204B/C receiver physical lanes (SRXx_x). Each logical lane has to be mapped to a separate physical lane for links to be active. Only the first L logical lanes will be active for each link. These can be mapped to any of the physical lanes. If in dual link mode, only modes with 6 lanes or fewer and 8 virtual converters are fewer (M≤8 and L≤6) are supported on link1. The total number of lanes used must be ≤12.

Only the first L logical lanes will be active for each link. For example, if link0 has L=6 and link1 has L=3, logical lanes 0-5 of link0 are active and logical lanes 0-2 of link1 are active. All other logical lanes are inactive. The example shown in Figure 84 illustrates this configuration where link0 uses PHY lanes 1, 2, 4, 5, 0, and 8 in that order (routing shown in green). Link1 logical lanes are routed to PHY lanes 6, 11, and 9 in that order (routing shown in blue). The unused PHY lanes (3, 7, and 10) should be powered down. This function is taken care of by the device based on the L parameter setting and the mux4 settings

APOLLO MXFE TRANSMITTER

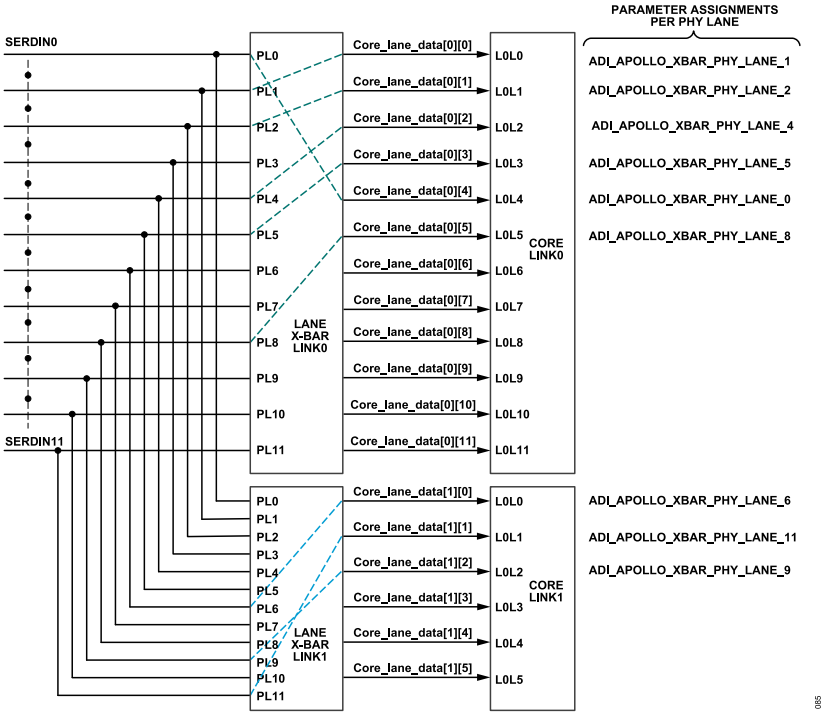

Figure 84. Transmit Mux4 configuration example

Transmit MUX4 Configuration

The device profile contains the Transmit MUX4 configuration information in the **adi_apollo_jesd_rx_link_cfg structure_t**. The parameter and enumeration are described in Table 70. The device will power down any unused PHY lanes. In dual link mode, these assignments must be made for each link.

Table 70. JESD204C Receiver Crossbar Parameter per PHY lane

Parameter	Description	Enumerations	Comments
lane_xbar	Index that sets the source logical lane for each of the physical lanes	ADI_APOLLO_XBAR_PHY_LANE_[0..11]	0= PHY lane0, etc.

JESD204B/C Receiver Data Link Layer Selection

The JESD204B/C receiver in the device transmitter can operate using the 8-bit/10-bit link layer (JESD204B) or the 64-bit/66-bit link layer (JESD204C). The ver parameter in the **adi_apollo_jesd_common_cfg_t** structure of the device profile is used to make this selection. Table 71 describes this parameter and its enumerations. Note that this parameter (along with all parameters in the **adi_apollo_jesd_common_cfg_t** structure) apply to both the JESD204B/C transmitter and receiver.

Table 71. JESD204B/C Receiver Data Link Layer Selection Parameter

Parameter	Description	Enumerations	Comments
ver	Selects the JESD204 version	ADI_APOLLO_JESD_204B	version is 204B
		ADI_APOLLO_JESD_204C	version is 204C

JESD204B Receiver 8-bit/10-bit Link Layer

The 8-bit/10-bit link layer of the device accepts the deserialized data from the PHYs and deframer and descrambles the data so that data octets are presented to the transport layer to be recombined into the original data samples ahead of the DAC core.

The device can be set up to receive data from either a single link or from two links. When operating in dual-link mode, the data link layer abstracts the interface to appear as two independent JESD204B links to the user, each occupying a maximum of six lanes.

APOLLO MXFE TRANSMITTER

In either mode, all JESD204B interface lanes independently handle link layer communications such as CGS, FS, and ILAS. The 8-bit/10-bit link layer decodes 8-bit/10-bit control characters, which mark the edges of the frame and help maintain alignment between serial lanes. Each link can issue a synchronization request by setting the SYNCOUTB_xx signal low. The synchronization protocol follows the JESD204B and JESD204C standards for 8-bit/10-bit link layer operation. When a stream of four consecutive /K/ symbols is received on any enabled JESD204B receiver lane, the device deactivates the synchronization request by setting the SYNCOUTB_xx signals high at the next internal LMFC rising edge. Then, the 8-bit/10-bit link layer waits for the transmitter to issue an ILAS.

During the ILAS, all lanes are aligned using the /A/ to /R/ character transition, as described in the JESD204B /C standards. Figure 85 illustrates how the elastic buffers hold early arriving lane data until the alignment character of the latest lane arrives. At this point, the buffers for all lanes are released and all lanes are aligned.

Lane First In/First Out (FIFO)

The FIFOs in front of the crossbar switch and deframer adjust the phase of the incoming data to synchronize the samples sent on the high-speed serial data interface with the deframer clock.

Lane FIFO Interrupt Request Operation (IRQ)

An aggregate lane FIFO error bit is also available as an IRQ event along with other JESD204 receiver status IRQ's when enabled using the **adi_apollo_jrx_j204c_irq_enable_set()** function. Enumerations for setting this and other JESD204C status IRQ's can be found in **adi_apollo_jrx_j204c_irq_e** found in the adi_apollo_jrx.h header file.

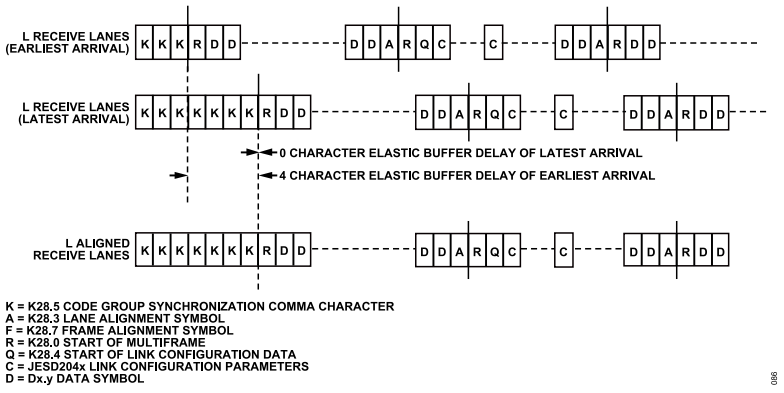

Figure 85. Lane Alignment During ILAS

8-Bit/10-Bit Link (JESD204B) Synchronization Status and Error Monitoring

The **adi_apollo_jrx_j204b_lane_error_get()** function of the Apollo MxFE API will retrieve the per-lane status of the link synchronization process as well as monitor each lane for 8-bit/10-bit encoding errors once the link has been established. Link synchronization status is checked by the function by retrieving the status of the bit fields associated with CGS, FS, CKS, ILS for each instantiated lane. Once the link is synchronized (CGS, FS, CKS, ILS, and ILD all "PASS") and the link is in the user data phase (sample data is being received by the DACs), 8-bit/10-bit errors, such as bad disparity (BD), not in table (NIT), and unexpected K-characters (UEKC) can be detected and counted. The parameters returned by the **adi_apollo_jrx_j204b_lane_error_get()** function are described Table 72.

Error thresholds for BD, NIT, and/or UEKC errors can be set using the TBD function. The error threshold determines the number of BD, NIT, and UEKC errors that will cause their respective parameters to read back as an error (see Table 72). IRQs can also be set to provide an interrupt whenever the threshold is violated. the Apollo MxFE can also be programmed to re-assert the SYNCOUT output pin to initiate re-synchronization when the threshold is violated.

The information in Table 72 and Table 73 is preliminary and subject to change until the supporting API is released.

The error counts can be checked for BD, NIT, and UEK errors. The error counts are on a per lane and per error type basis. A register is dedicated to each error type and lane. Use the TBD function to read the error count registers. The parameters returned by this function are described in Table 73.

APOLLO MXFE TRANSMITTER

Table 72. JESD204B Lane Status Parameters from adi_apollo_jrx_j204b_lane_error_get()

Parameter	Description	Enumerations	Comments
jrx_dl_204b_bd	Bad Disparity (BD) error status for Lanes [L-1:0]	1 = BD error	Per lane BD error flag (Read only). Indicates jrx_dl_204b_bde_cnt>=jrx_dl_204b_eth
jrx_dl_204b_nit	Not-in-table (NIT) error status for Lanes [L-1:0]	1 = NIT error	Per lane NIT error flag (Read only). Indicates jrx_dl_204b_nit_cnt>=jrx_dl_204b_eth
jrx_dl_204b_uek	Unexpected K-char (UEKC) error status for Lanes [L-1:0]	1 = UEKC error	Per lane UEKC error flag (Read only). Indicates jrx_dl_204b_uek_cnt>=jrx_dl_204b_eth
jrx_dl_204b_cks	Checksum (CKS) error status for Lanes [L-1:0]	1 = CKS_PASS 0 = CKS_ FAIL	Per lane CKS status flag (Read only)
jrx_dl_204b_ils	Initial Lane Synchronization (ILS) status for Lanes [L-1:0]	1 = ILS_PASS 0 = ILS_ FAIL	Per lane ILS status flag (Read only)
jrx_dl_204b_ild	Inter-Lane De-skew (ILD) status for Lanes [L-1:0]	1 = ILD_ PASS 0 = ILD_ FAIL	Per lane ILD status flag (Read only)
jrx_dl_204b_fs	Frame Sync (FS) status for Lanes [L-1:0]	1 = FS_PASS 0 = FS_ FAIL	Per lane FS status flag (Read only)
jrx_dl_204b_cgs	Code Group Sync (CGS) status for Lanes [L-1:0]	1 = CGS_PASS 0 = CGS_FAIL	Per lane CGS status flag (Read only)

Table 73. JESD204B Link Status monitoring Parameters

Parameter	Description	Enumerations	Comments
jrx_dl_204b_bd_cnt	BD error counters for Lanes [L-1 : 0]		8-bit error counter per lane
jrx_dl_204b_nit_cnt	NIT error counters for Lanes [L-1 : 0]		8-bit error counter per lane
jrx_dl_204b_uek_cnt	UEKC error counters for Lanes [L-1 : 0]		8-bit error counter per lane
jrx_dl_204b_eth	Error Threshold for BD, NIT and UEK Errors Count.		BD, NIT and UEK errors are counted and compared to the Error Threshold value
sync_assert_mask	SYNCOUT assertion mask	TBD (001) TBD (010) TBD (100) : TBD (111)	Assert SYNCOUT if UEKC error count > threshold Assert SYNCOUT if NIT error count > threshold Assert SYNCOUT if BD error count > threshold : Assert SYNCOUT if any error count > threshold
jrx_dl_204b_irq_vec		TBD (0) TBD (1) TBD (2 TBD (3) TBD (4) TBD (5) TBD (6) TBD (7) TBD (8)	CodeGroupSync State Machine Flag. FrameSync State Machine Flag GoodCheckSum Flag ILS State Machine Flag InterLaneDeskew Flag UEK error Count > jrx_dl_204b_eth in any lane NIT error Count > jrx_dl_204b_eth in any lane BD error Count > jrx_dl_204b_eth in any lane. Configuration Mismatch Flag.

SYNCOUTB_A[0:1]P/N and SYNCOUTB_B[0:1]P/N Outputs

The SYNCOUTB_A[0:1]P/N and SYNCOUTB_B[0:1]P/N signals are necessary for supporting JESD204B modes of operation and are LVDS or CMOS selectable via the syncb_lvds_mode parameter that is stored in the device profile per A/B-side. When LVDS mode is selected with no internal termination, use controlled impedance traces routed as 100 Ω differential impedance and 50 Ω to ground when routing these signals. The SYNCOUTB signal can be routed to one, or both of the SYNCOUTB pins using the syncb_out_sel parameter that is also stored in the device profile per A/B-side. These parameters are described in Table 74.

Table 74. JESD204B/C Receiver SYNCOUTB Configuration Parameters

Parameter	Description	Enumerations	Comments
syncb_out_sel	Selects SYNCOUTB pin	0 = ADI_APOLLO_JESD_DFRM_PWR_ON_SYNC_PAD1	Selects SYNCOUTB_A0/B0 pin

APOLLO MXFE TRANSMITTER

Table 74. JESD204B/C Receiver SYNCOUTB Configuration Parameters (Continued)

Parameter	Description	Enumerations	Comments
		1 = ADI_APOLLO_JESD_DFRM_PWR_ON_SYNC_PAD2, 2 = ADI_APOLLO_JESD_DFRM_PWR_ON_ALL_SYNC_PADS 3 = ADI_APOLLO_JESD_DFRM_PWR_OFF_ALL_SYNC_PADS	Selects SYNCOUTB_A1/B1 pin Selects both A0/B0 and A1/B1 pins No SYNCOUTB
syncb_lvds_mode	Selects the SYNCOUTB mode	0 = ADI_APOLLO_JESD_FRM_SYNCB_CMOS 1 = ADI_APOLLO_JESD_FRM_SYNCB_LVDS_WITH_INTL_TERM 2 =ADI_APOLLO_JESD_FRM_SYNCB_LVDS_NO_INTL_TERM	CMOS output LVDS w/termination LVDS, no termination

Monitoring Errors Via SYNCOUTB_xx

When one or more BD, NIT, or UEKC errors occur, the error is reported on the appropriate SYNCOUTB_xx pins, per the JESD204B specification. The SYNCOUTB_xx signals are asserted for two jrx_conv_clk periods when an error occurs. jrx_conv_clk equals f_{DAC}/(INTERPOLATION × JRX_NS). JRX_NS per JESD204 mode is given in the JESD204B/C Receiver Mode Tables section. Note that jrx_conv_clk is called conv_clk in the API functions. The jrx_dl_204b_sync_err_enable parameter is set to 1 (enabled) by default. For use cases where JRX_NS > S (See JESD204B/C Receiver Mode Tables section), jrx_dl_204b_sync_err_enable should be set to 0 (error reporting disabled). This parameter is set using the TBD function of the Apollo MxFE API.

8-Bit/10-Bit Link Error PA Protection

The Apollo MxFE offers the feature to ramp down the DAC outputs to 0 V dc in the event of any 8-bit/10-bit link error (BD, NIT, UEKC), FIFO error (fifo_full, fifo_empty), or synchronization error (CGS, FS, ILS, ILD, CKS) . In addition, The parameters to use to ramp down the outputs are described in Table 75. The TBD function in the Apollo MxFE API implements this feature based on these parameters.

The information in Table 75 is preliminary and subject to change until the supporting API is released.

Table 75. JESD204B Link Error PA Protection Parameters

Parameter	Description/ Enumerations	Comments
en_204b_uek_jrx_int_gainoff	1 = enable the JESD204B receiver UEK error soft off gain source.	The device ramps the analog output down to 0 V dc if the UEK threshold is reached.
en_204b_nit_jrx_int_gainoff	1 = enable the JESD204B receiver NIT error soft off gain source.	The device ramps the analog output down to 0 V dc if the NIT threshold is reached.
en_204b_bd_jrx_int_gainoff	1 = enable the JESD204B receiver BD error soft off gain source.	The device ramps the analog output down to 0 V dc if the BD threshold is reached.
en_204b_fifo_full_jrx_int_gainoff	1 = enable the JESD204B receiver fifo_full error soft off gain source.	The device ramps the analog output down to 0 V dc if a fifo_full error occurs.
en_204b_ fifo_empty_jrx_int_gainoff	1 = enable the JESD204B receiver fifo_empty error soft off gain source.	The device ramps the analog output down to 0 V dc if a fifo_empty error occurs.
en_204b_cgs_jrx_int_gainoff	1 = enable the JESD204B receiver CGS soft off gain source	The device ramps the analog output down to 0 V dc if a CGS error occurs.
en_204b_fs_jrx_int_gainoff	1 = enable the JESD204B receiver FS soft off gain source	The device ramps the analog output down to 0 V dc if an FS error occurs.
en_204b_ils_jrx_int_gainoff	1 = enable the JESD204B receiver ILS soft off gain source	The device ramps the analog output down to 0 V DC if an ILS error occurs.
en_204b_ild_jrx_int_gainoff	1 = enable the JESD204B receiver ILD soft off gain source	The device ramps the analog output down to 0 V dc an ILD error occurs.
en_204b_cks_jrx_int_gainoff	1 = enable the JESD204B receiver CKS soft off gain source.	The device ramps the analog output down to 0 V dc if a CKS error occurs.

JESD204C Receiver 64-bit/66-bit Link Layer and Synchronization

Synchronization Header Alignment

The synchronization transition bit in the synchronization header ensures that there is a data transition at every block boundary (66 bits). A state machine in the JESD204C receiver detects a data transition and then looks for another transition 66 bits later. If the state machine detects bit

transitions at 66-bit intervals for 64 consecutive blocks, the SH_LOCK state is achieved. The machine is restarted if 64 consecutive transitions are not detected.

Extended Multiblock Sync

When synchronization header alignment is achieved, the receiver looks for the end of the EoEMB sequence (100001) in the transition bits. The structure of the synchronization word ensures that this sequence can only happen at the appropriate time.

When an EoEMB sequence is identified, the state machine examines every 32nd synchronization word to ensure that the end of multiblock pilot signal (00001) is present. If E = 1, the EoEMB bit is also present with the pilot signal.

If E is >1, the pilot signal includes the EoEMB bit for every E×32 transition bit. When four consecutive valid sequences are detected, the EMB_LOCK state is achieved. The monitoring of every E×32 transition bit continues. If a valid sequence is not detected, the EMB_LOCK is lost, and the alignment process resets.

Extended Multiblock (Lane) Alignment

Extended multiblock alignment achieves lane alignment when using the 64-bit/66-bit link layer, which is similar to using the 8-bit/10-bit link layer in that an elastic buffer is employed in the JESD204C receiver on each lane to store incoming data. During extended multiblock alignment, the buffers for each lane start storing data at the incoming data EoEMB boundary (rather than the /K/ to /R/ boundary during ILAS when using the 8-bit/10-bit link layer) and all lanes release the respective buffered data coincident with the last arriving lane EoEMB boundary.

Figure 86 illustrates how extended multiblock lane alignment is achieved. Each lane receive buffer, except for on the last arriving lane, starts buffering data when the last bit of the EoEMB synchronization word bit field (see Table 30) is received. When the last arriving lane EoEMB synchronization word bit field is received, the release of all lane receive buffers is triggered so that all lanes align.

64-Bit/66-Bit Error Monitoring and Resynchronization

Error monitoring during the transmission of sample data is achieved by monitoring the CRC-12 data bits transmitted as part of the synchronization word and comparing the value to the CRC-12 value that is automatically calculated in the JESD204C receiver. See the 64-Bit/66-Bit Link Establishment Overview section for details. The API functions used to enable the JESD204C link error IRQs and retrieve the IRQ status are described in Table 77. Refer to the JESD204B/C Debug Guide for details on debugging CRC errors. Note that SH, MB, and EMB errors will result in the JESD204C link to automatically perform a re-synchronization since these represent a “link down” state. CRC errors will not automatically result in a link reset but could be indicative of poor signal integrity on the JESD data lanes if there are no other associated errors that result in a loss of link synchronization.

If the receiver detects certain errors, synchronization is lost. In this case, the receiver restarts the synchronization state machines automatically. The transmitter continues sending data. The receiver must resynchronize. Errors causing an automatic re-sync are:

- Invalid Sync Header (JRX_204C_SH_IRQ = 1)
- Unexpected EoMB sequence (JRX_204C_MB_IRQ = 1)
- Unexpected EoEMB sequence (JRX_204C_EMB_IRQ = 1)

System software can monitor the status of both the synchronization header alignment and the extended multiblock alignment state machines to allow the system controller to be informed of the respective states. If resynchronization is required, the system controller must power down both sides of the link. If a reconfiguration or clocking change is required, do so while the link is powered down. Resynchronization takes place automatically at link power-up.

If the receiver detects too many CRC-12 errors as determined by the crc_err_threshold parameter, the JRX_204C_CRC_IRQ can be enabled so that the system controller can be notified. In this case (JRX_204C_CRC_IRQ = 1), the controller may choose to force a re-synchronization since this is not done automatically. In this case, the receiver restarts the synchronization state machine and the transmitter continues sending data. The receiver must resynchronize.

Monitor the status of both the synchronization header alignment and extended multiblock alignment state machines so the system controller is informed of their respective states. If resynchronization is required, the system controller powers down both sides of the link. If a reconfiguration or clocking change is required, perform it while the link is powered down. Resynchronization takes place automatically at link power up.

APOLLO MXFE TRANSMITTER

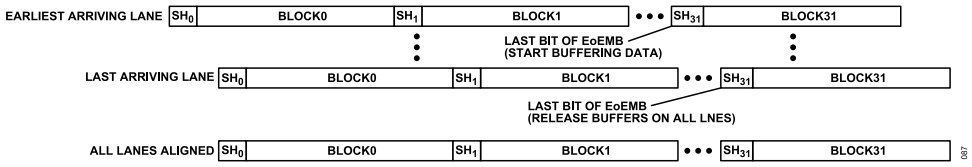

Figure 86. JESD204C Extended Multiblock (Lane) Alignment

Monitoring the 64-Bit/66-Bit Synchronization State

The 64-bit/66-bit link layer starts automatically when the link is powered on. The synchronization process begins with synchronization header alignment. When the JESD204C receiver aligns to the incoming synchronization header, it can be referred to as synchronization header lock (SH_LOCK).

When the SH_LOCK is achieved, the synchronization process progresses to extended multiblock synchronization, and then to extended multiblock alignment. Within the JESD204C receiver, this process is controlled by a state machine.

The state machine operation is shown in Figure 87. State 2, State 3, and State 4 represent the three phases of the synchronization process. When these phases are complete, the state machine locks (state = 6) and the link is up.

The **adi_apollo_jesd_rx_j204c_lane_status_get()** API function will return the status of the JESD204C receiver state machine as described in Table 76. Refer to the JESD204B/C Debug Guide for more details.

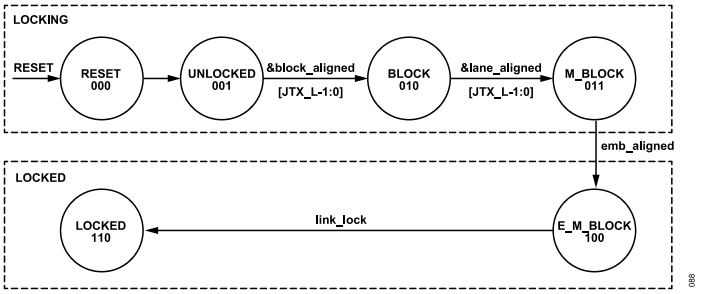

Figure 87. JESD204C 64b/66b Synchronization State Diagram

Table 76. JESD204C Receiver Lane Status Parameters

Parameter Name	Description	Enumeration
JRX_DL_204C_STATE	Current state of the JESD204C link	0: RESET. 1: UNLOCKED. Complete Sync Header (SH) detection and alignment 2: BLOCK. Complete the lane de-skew and lane alignment 3: DESKEW. Complete Multi-Block detection and alignment 4: M_BLOCK. Complete Extended Multi-block detection 5: E_M_BLOCK. Complete Extended Multi-block alignment 6: Link Ready. Link is up and running

IRQ's can be used to inform the system controller of JESD204C errors such as CRC, SH, MB, and EMB errors. The functions, parameters and enumeration for these are described in Table 77.

Table 77. API Functions for JESD204C Error Monitoring IRQs

API Function Name	Applicable Parameter(s)	Enumeration
adi_apollo_jesd_rx_j204c_crc_irq_enable()	JRX_204C_CRC_IRQ_ENABLE	1: Enables the CRC error IRQ
adi_apollo_jesd_rx_j204c_crc_irq_status_get()	JESD204C CRC IRQ	0: No CRC errors or IRQ 1: CRC error and IRQ has occurred
adi_apollo_jesd_rx_j204c_sh_irq_enable()	JRX_204C_SH_IRQ_ENABLE	1: Enables the SH error IRQ
adi_apollo_jesd_rx_j204c_sh_irq_status_get()	JESD204C SH IRQ	0: No SH errors or IRQ 1: SH error and IRQ has occurred
adi_apollo_jesd_rx_j204c_mb_irq_enable()	JRX_204C_MB_IRQ_ENABLE	1: Enables the MB error IRQ

APOLLO MXFE TRANSMITTER

Table 77. API Functions for JESD204C Error Monitoring IRQs (Continued)

API Function Name	Applicable Parameter(s)	Enumeration
adi_apollo_jesd_rx_j204c_mb_irq_status_get()	JESD204C MB IRQ	0: No MB errors or IRQ 1: MB error and IRQ has occurred
adi_apollo_jesd_rx_j204c_emb_irq_enable()	JRX_204C_EMB_IRQ_ENABLE	1: Enables the EMB error IRQ
adi_apollo_jesd_rx_j204c_emb_irq_status_get()	JESD204C EMB IRQ	0: No EMB errors or IRQ 1: EMB error and IRQ has occurred

Common JESD204B/C Link Status Checking

The **adi_apollo_jrx_link_status_get()** API function will return the status of the JESD204B/C receiver related parameters listed in Table 78.

Table 78. JESD204B/C Receiver Link Status Parameters

Parameter Name	Description	Enumeration
DOWN_SCALE_OVERFLOW	Checks If NS Down_Scale_Ratio is Too Large	0 : No overflow 1 : Overflow
JESD_MODES_NOT_IN_TABLE(JRX)	Quick Config Mode Number validity (for link0 and link1 – 2 bits)	0 : VALID : 1 : INVALID (not in mode table)
JRX_CORE_USR_DATA_RDY	Output Conv_sample Contains Valid User Data. This may not exactly align with an LMFC boundary.	0 : User Data Not Ready. The link has not been established yet and JESD received data is not available. 1 : JESD User Data Ready. The link is established and correctly received data is available
JRX_CORE_SYSREF_RCVD	SYSREF Phase Has Been Established	0 : No SYSREF. JRX has not received SYSREF signal and data receiving has not started.
JRX_CORE_CFG_INVALID	S, NS, F, NP and L Have to Be Within Pre-defined Range	0 : JESD is correctly configured. 1 : SYSREF Received. JRX has received SYSREF and can receive data 1 : Invalid Configuration. At least one of the parameters is invalid: S, F, NP and L

The **adi_apollo_jrx_lane_fifo_status()** API function will return the status of the JESD204B/C receiver FIFO as described in Table 79.

Table 79. JESD204B/C Receiver FIFO Status Parameters

Parameter Name	Description	Enumeration
LANE_FIFO_EMPTY	FIFO empty flag per instantiated PHY lane (up to 12), For example, if SERDIN lanes 0,3,4, and 7 are used, 0x099 return values means all 4 lanes have empty FIFOs.	1 : FIFO is empty
LANE_FIFO_FULL	FIFO full flag per instantiated PHY lane (up to 12), For example, if SERDIN lanes 0,3,4, and 7 are used, 0x099 return values means all 4 lanes have full FIFOs.	1 : FIFO is full

The “data ready” (DATA_RDY) signal in the JESD204B/C receiver is another indication that the link layer is synchronized and the sample data being received is valid. An IRQ can be enabled to detect the loss of the DATA_RDY flag. To enable and retrieve the JRX_DATA_RDY_LOST_IRQ, the **adi_apollo_reg_TX_DIGITAL0/1__JRX__jrx_wrapper__JRX_DATA_RDY_LOST_IRQ0/1_ get()** and **adi_apollo_reg_TX_DIGITAL0/1__JRX__jrx_wrapper__JRX_DATA_RDY_LOST_IRQ0/1_set()** functions can be used. A returned value of 1 indicate data ready has been lost.

See the JESD204B/C Debug Guide section for more details on debugging link errors.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

JESD204B/C Receiver Transport Layer

The transport layer handles un-packing the data that consists of samples and optional control bits from the 8-bit octets coming out of the link layer. The transport layer packing and un-packing is controlled by rules derived from the link parameters.

For more information on the transport layer, refer to the Analog Devices webcast, the JESD204B Transport Layer here.

APOLLO MXFE TRANSMITTER

JESD204B/C Receiver Transport Layer Configuration Parameters

The parameters for the transport layer in the device profile are determined by the Quick Configuration Value as described in the JESD204B/C Receiver Mode Tables. Alternately, the JESD204B/C mode parameters can be set individually if the quick_cfg_en parameter is not set. The most relevant parameters related to the JESD transport layer are described in Table 80. These parameters are found in the **adi_apollo_jesd_rx_link_cfg_t** structure of the device profile.

Table 80. JESD204B/C Receiver Transport Layer Parameters

Parameter	Description	Enumerations	Comments
quick_cfg_en	Enable Quick configuration mode	Boolean (0 or 1)	FALSE = configure JESD using parameter values TRUE= configure JESD using quick_mode_id
quick_mode_id	Quick configuration mode ID	uint8_t	See Table 81 (JESD204 Mode Number)
l_minus1	number of lanes per link, values supported are 1, 2, 3, 4, 6, 8, or 12	ADI_APOLLO_LANES_PER_LINK_[1,2,3,4,6,8,12]	1 = 1 lane, etc.
m_minus1	number of virtual converters per link, values supported are 1, 2, 3, 4, 6, 8, 12, or 16	ADI_APOLLO_CONV_PER_LINK_[1,2,3,4,6,8,12,16]	1 = 1 virtual converter, etc.
f_minus1	octets per lane per frame, values supported are 1, 2, 3, 4, 6, 8, 12, 16, 24, or 32	ADI_APOLLO_OCT_PER_FRM_[1,2,3,4,6,8,12,16,24,32]	1 = 1 octet per frame, etc.
s_minus1	samples per virtual converter per frame cycle.(S = (8*L*F)/(M*N'))	ADI_APOLLO_SAMP_CONV_FRM_[1,2,3,4,6,8,12,16]	1 = 1 sample per converter, etc. Valid values are 1, 2, 3, 4, 6, 8, 12, or 16
np_minus1	number of bits per JESD word, values supported are 8, 12, or 16.	ADI_APOLLO_BITS_PER_SAMP_[8,12,16]	8 = JESD word size is 8, etc.
k_minus1	number of frames per multiframe, (must be ≤ 256)	uint8_t	= 32 for 204B and K=E*(256/F) for 204C
e_minus1	Number of blocks in a multiblock (204C only)	uint8_t	E =LCM(F,256)/F
Cs	number of control bits per sample, values supported are 0, 1, 2, or 3	ADI_APOLLO_CONT_BITS_PER_SAMP_[01,2,3]	0 = no control bits, etc.

Transmit Mux3 (JESD204B/C Receiver Sample Crossbar)

The JESD204B/C Receiver Sample Crossbar is provided for flexibility in routing sample data through the data path when needed. At this time, there have been no use cases where this has been found to be necessary. Since this is the case, transmit mux3 configured automatically based on the JESD204B/C receiver's M value and the device configuration (4T4R or 8T8R).

Transmit Path Deterministic Latency

The JESD204B/C systems contain various clock domains distributed throughout. Data traversing from one clock domain to a different clock domain can lead to ambiguous delays in the JESD204B/C link. These ambiguities lead to nonrepeatable latencies across the link from power cycle to power cycle with each new link establishment. The Apollo MxFE's JESD204B/C receiver supports subclass 0 operation (deterministic latency across power cycles not required) and subclass 1 operation (using SYSREF to ensure deterministic latency across power cycles).

Subclass 0

Even though not required, the Apollo MxFE's subclass 0 mode provides deterministic latency to be within several jrx_conv_clk cycles as long as all lanes arrive within the lane de-skewing capability of the device. The Apollo MxFE can de-skew up to 3 PCLK cycles of inter-lane skew. When using a JESD204C mode, this amounts to 198 UI. For JESD204B modes, this amounts to 120 UI. This mode does not require any signal on the SYSREF_x_P/N pins, which can be left disconnected. The jrx_conv_clk (conv_clk in the API) value is the rate at which digital samples are processed through the JESD204B/C receiver and is equal to F_{DAC}/(interpolation × NS). F_{DAC} is the rate at which the DAC samples data. NS is the number of samples processed per F_{DAC} in the data path and its value depends on the JESD204B/C mode and the device configuration (4T4R vs 8T8R). The NS value is given in the Table 81 table.

Subclass 1

Subclass 1 mode provides deterministic latency and allows the link to be synchronized to within ±1 MCLK period (or less) when configured appropriately. This latency requires an external, low jitter SYSREF_x_P/N signal that is accurately phase aligned to the MCLK.

APOLLO MXFE TRANSMITTER

Link Delay

The link delay of a JESD204B/C system is the sum of the fixed and variable delays from the transmitter, channel, and receiver, as shown in Figure 88.

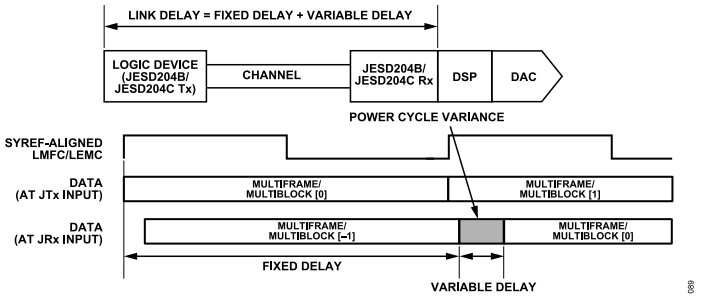

Figure 88. JESD204B Link Delay = Fixed Delay + Variable Delay

Deterministic Latency Requirements

The following key factors are required for achieving deterministic latency in a JESD204B/C Subclass 1 system:

- The SYSREF_x_P/N signal distribution skew within the system must be less than the desired uncertainty.
- SYSREF setup and hold time requirements must be met for each device in the system when hardware-based synchronization is used. When the firmware and TDC based synchronization approach is used there are no setup or hold time requirements.
- The receive buffer of the JESD204B/C receiver is a finite resource. As stated previously, the latency variation across all lanes within a single link must be less than 3 PCLK cycles. When using a JESD204C mode, this amounts to 198 UI. For JESD204B modes, this amounts to 120 UI. The total latency variation across all links, and devices in an aligned system must be less than MIN(32, 1/2 *K*S/NS) jrx_conv_clk cycles. These values include both variable delays and the variation in fixed delays from lane to lane, link to link, and device to device in the system.

To achieve the deterministic latency, the JESD204B/C receiver has a receive buffer implemented using a FIFO with a read timing referenced to the LMFC/LEMC which aligns to the SYSREF. The LMFC/LEMC boundary of the received data is possibly located significantly far away from the next local LMFC/LEMC. In this case, the receive buffer may not be deep enough to hold all the data. For this reason, the **adi_apollo_jrx_phase_adjust_set()** API function can be called to adjust the read pointer of the buffer by using the **phase_adjust** parameter (16-bit value in jrx_conv_clk units) so that the buffer releases data sufficiently close to, but after, the boundary of LMFC/LEMC of the received data. The boundary of the LMFC/LEMC on the received data could be very close to the local LMFC/LEMC which means the total link latency in the system is nearly an integer multiple of the LMFC/LEMC period. For such case, from power cycle to power cycle, the data arrival time of the incoming LMFC/LEMC boundary at the receive buffer could straddle exactly on the boundary of the local LMFC/LEMC. For such case, it's recommended to use the **phase_adjust** parameter to move the buffer read pointer so that the received LMFC/LEMC of all lanes and all links, across all power cycles, arrives at least two jrx_conv_clk cycles prior to the buffer release time (or the local LMFC/LEMC boundary).

Tips and tools for ensuring deterministic latency can be achieved:

- Calling the **adi_apollo_jrx_phase_diff_get()** function will return the phase_diff parameter to determine the timing difference between the LMFC/LEMC boundary on the received data relative to the internal SYSREF (or the local LMFC/LEMC) on Apollo MxFE to see if the above scenarios are in play.
  - The values of phase_diff are close to zero (e.g. 0~2) or close to the value of (K*S)/NS (e.g. 29~31 when (K*S)/NS=32) which suggests to adjust the LMFC/LEMC phase.
  - The values larger than 32 (buffer depth) is recommended to adjust the LMFC/LEMC phase.
- If the hardware-based SYSREF alignment method is used, check if the setup and hold time requirements for the SYSREF are met. Refer to the SYSREF Input and Setup/Hold Detector section.
  - If the SYSREF setup and hold (SH) detector indicates a potential timing problem, the phase of the SYSREF signal supplied to the Apollo MxFE must be adjusted until the sh_before and sh_after parameters indicate that there is no potential timing problem.

APOLLO MXFE TRANSMITTER

- Regardless of alignment method, sysref_phase can be read to determine if SYSREF alignment has been achieved. This can be done by calling the **adi_apollo_clk_mcs_sysref_phase_get()** function.
  - This read indicates the phase relationship between the incoming SYSREF signal relative to the internal SYSREF and the LMFC/LEMC boundaries.
  - A readback of 0 indicates that internal SYSREF (and LMFC/LEMC) are aligned to the external SYSREF. A nonzero value such as 10, for example, indicates that the SYSREF rising edge is 10 MCLK cycles later than the internal SYSREF.

In Figure 89 and Figure 90 example, the link delay is approximately an integer multiple of an LMFC/LEMC period, and the power-cycle variation occurs across an LMFC/LEMC boundary in the JESD204B/C receiver. Use the **phase_adjust** parameter to move the read pointer beyond the last arriving data. The step size for this adjustment is in jrx_conv_clk cycles, where jrx_conv_clk = f_{DAC}/(INTERPOLATION × NS). Note that jrx_conv_clk is called conv_clk in the API functions.

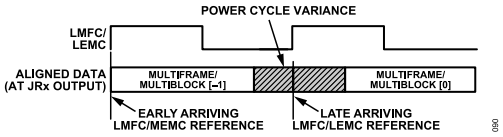

Figure 89. Link Delay > LMFC/LEMC Period Example

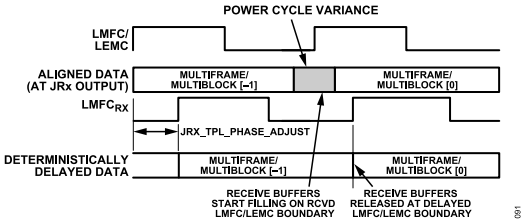

Figure 90. Using phase_adjust parameter to Ensure Deterministic Latency

The method to select the appropriate value for **phase_adjust** parameter is described in the LMFC/LEMC Delay Adjustment Example section.

Setting the **phase_adjust** parameter appropriately ensures that the receive buffer absorbs all link delay variation. This ensures that all data arrives before reading to achieve the deterministic latency.

LMFC/LEMC Delay Adjustment Example

The JESD204B/C receivers' **phase_adjust** parameter is stored in the **adi_apollo_jesd_rx_link_cfg_t** structure in the device profile and has a default value of zero. Each system design will have its own optimal value for **phase_adjust**. The **phase_adjust** parameter must be set during configuration prior to enabling the JESD204B/C link. To determine the optimal value for a given system, use the method as follows. This method is implemented in the Apollo MxFE example code at examples\ads10_apollo_ex_main\fullchip_sc1_dl.c. The **adi_apollo_jrx_phase_diff_get()** API function returns the **phase_diff** parameter which is the measured phase difference between the local LMFC/LEMC boundary and the LMFC/LEMC boundary of the arriving data using in jrx_conv_clk cycles. This information is used to calculate the appropriate **phase_adjust** parameter setting. The jrx_phase_diff value will range between 0 to ((K*S)/NS)-1 .

To determine the **phase_adjust** parameter setting, take the following steps:

1. Cycle the link power at least 20 times using the process described in the Configuring the JESD204B/C Receiver section to determine the latest possible arriving time of the incoming data LMFC/LEMC boundary across all power cycles.
2. After each power-on cycle, read the **phase_diff** parameter and record the values.
3. The latest possible arriving time is equal to the largest value recorded in Step 2. The count values range from 0 to ((K × S)/NS) – 1 (the number of jrx_conv_clk cycles). If the range of values recorded span the terminal count value of the jrx_conv_clk counter, the latest possible arriving time value could be from 0 to 3 (see the example in Figure 92).
4. To ensure appropriate margin, add 2 to the latest possible arriving time and program this value into the **phase_adjust** parameter. This parameter value is stored in the **adi_apollo_jesd_rx_link_cfg_t** structure in the device profile.

Figure 91 shows an example using JESD204B Mode 63 (1 × 1 interpolation). Step 1 through Step 3 in this example reveal that there are only two possible values recorded in Step 2, that is, 7 and 0. Accounting for the counter roll-over from 7 to 0, 0 is the latest arriving time (from Step

APOLLO MXFE TRANSMITTER

2). Therefore, a value of 2 is programmed into the **phase_adjust** parameter (0 + 2, as described in Step 3). The data read out of the read buffer is deterministically delayed and output on the delayed LMFC/LEMC boundary (read pointer).

Figure 92 shows an example using JESD204C Mode 12 (2 × 4 interpolation). The receive buffer depth is always 32 jrx_conv_clk cycles deep. The LEMC contains 128 jrx_conv_clk cycles worth of data. Therefore, the receive buffer cannot store an entire LEMC worth of data. To avoid data errors, perform Step 1 through Step 3 for this example, which results in four possible values recorded in Step 2 (63, 64, 65, and 66). If no adjustment is made to the **phase_adjust** parameter, the buffer must have a minimum depth of 64 (127-63) to avoid data errors. However, because the latest arrival time of the received data was determined to be at a count of 66 (as described in Step 2), a value of 68 is programmed into the **phase_adjust** parameter (66 + 2, as described in Step 3). The data read out of the read buffer is deterministically delayed and output on the delayed LMFC/LEMC boundary (read pointer). Note that this method ensures the minimum latency through the DAC.

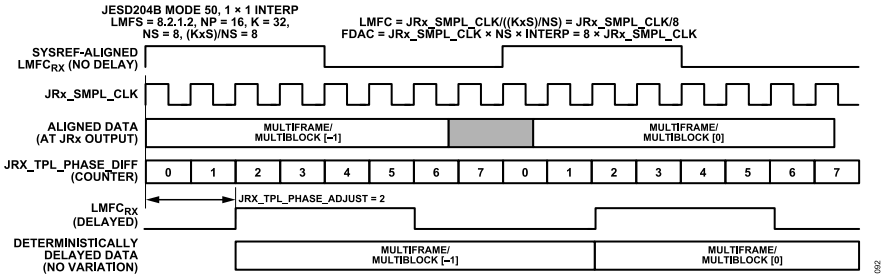

Figure 91. Example 1, Setting LMFC/LEMC Delay for Deterministic Latency

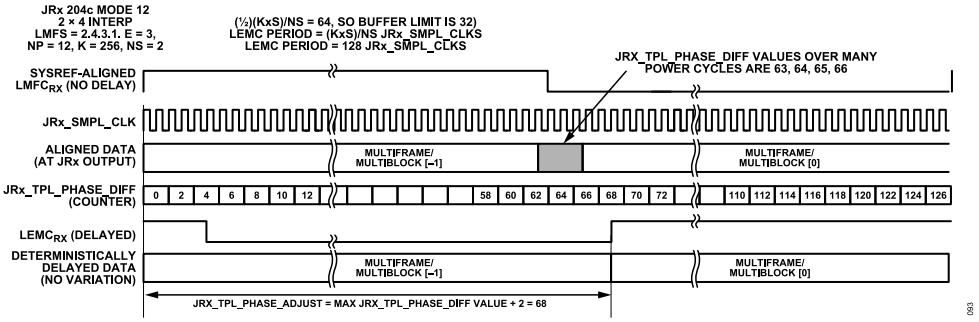

Figure 92. Example 2, Setting LMFC/LEMC Delay for Deterministic Latency

Phase Adjustment for Multi-chip Synchronization

To achieve multi-chip synchronization across the Tx data path of more than one Apollo MxFE device, each of the devices need to achieve subclass 1 synchronization using the phases described in the Device Synchronization section. After this, Adjustments to determine the **phase_adjust** parameter value for each device in a multilink or multichip system (similar to determining the deterministic latency for a single link), take the following steps:

1. Cycle the link power for all links and devices at least 20 times using the process described in the Apollo MxFE Bring-up Procedure section to determine the latest possible arriving time of the incoming data LMFC/LEMC boundary across all power cycles, links, and devices.
2. When each power-on cycle is complete, read the **phase_diff** parameter for all links and devices in the system and record the values. The count values range from 0 to ((K × S)/NS) – 1 (the number of jrx_conv_clk cycles). The latest possible arriving time across all links and devices is equal to the largest value recorded in Step 1. If the range of values recorded span the terminal count value of the jrx_conv_clkcounter, the latest possible arriving time value could be from 0 to 3.
   - To ensure appropriate margin, add 2 to the latest possible arriving time and set that value as the **phase_adjust** parameter for all the devices in the system. This parameter value is stored in the **adi_apollo_jesd_rx_link_cfg_t** structure in the device profile.

Figure 93 shows an example using JESD204C Mode 47. Step 1 through Step 3 for this example reveal that there are eight possible values across all power cycles, links, and devices recorded in Step 1. These values range from 47 to 54. 54 is the latest arriving time (as described in Step 2). Therefore, a value of 56 (54+2) is plugged in as the **phase_adjust** parameter for each of the devices in the system. The data coming out of the read buffer is deterministically delayed and output on the delayed LMFC/LEMC boundary.

APOLLO MXFE TRANSMITTER

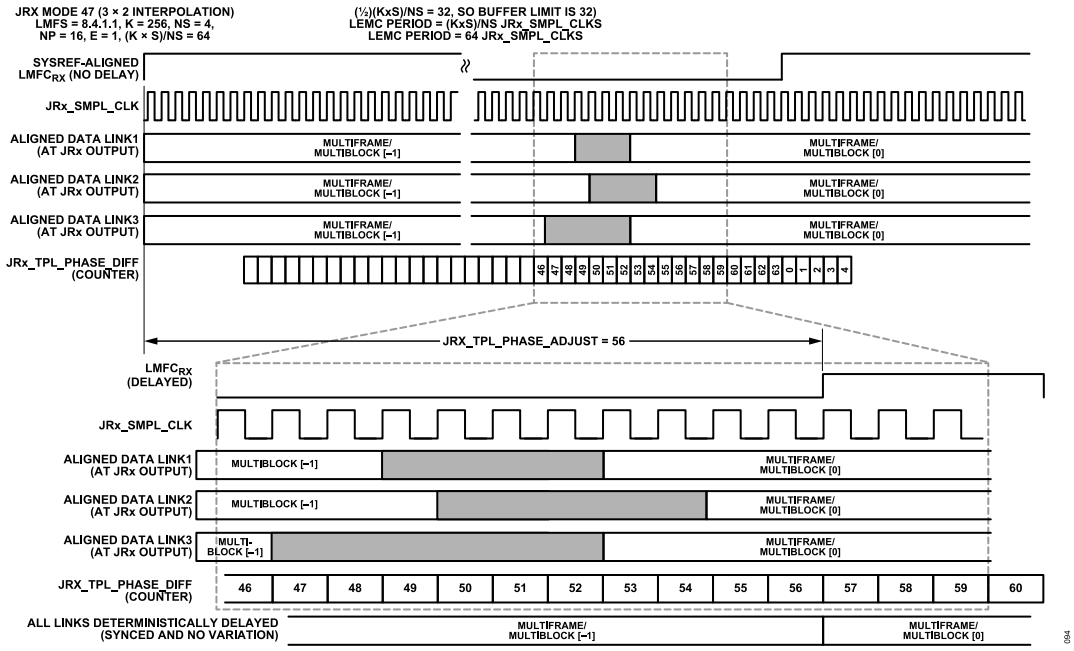

Figure 93. Setting LMFC/LEMC Delay for Multilink or Multichip Synchronization

JESD204B/C Receiver Mode Tables

The device receiver (DAC path) supports many JESD204B and JESD204C modes, as described in Table 81. The Fine x Coarse interpolation modes supported per JESD204B/C mode number depends on whether the device is operating in 204B or 204C mode as indicated in Table 82. Interpolation modes supported by JESD204B for a given JESD204 receiver mode are marked with a "B". Interpolation modes supported by JESD204C for a given JESD204 receiver mode are marked with a "C". If the interpolation mode is supported by both 204B and 204C, it is marked with "B, C".

Table 81. DAC Path Supported JESD204B/C Modes per A and B-sides

JESD204 Mode Number	L	M	F	S	N'	K (204B)	K (204C)	E (204C)	JRX_NS (204B)	JRX_NS (204C)	Dual link support per A/B	Mode Support Conditions	
												8T8R:AD9088	4T4R:AD9084
0	1	16	32	1	16	32	8	1	1	1	NO	when interpolation ratio>2x	Not Supported
1	1	16	24	1	12	32	32	3	1	1	NO	when interpolation ratio>2x	Not Supported
2	1	8	16	1	16	32	16	1	1	1	YES	when interpolation ratio>1x	when interpolation ratio>2x
3	1	8	12	1	12	32	64	3	1	1	YES	when interpolation ratio>1x	when interpolation ratio>2x
4	1	4	8	1	16	32	32	1	1	1	YES	No restrictions	when interpolation ratio>1x
5	1	4	6	1	12	32	128	3	2	2	YES	No restrictions	when interpolation ratio>1x
6	1	2	4	1	16	32	64	1	if 4t4r & CxF=12, 2; else, 1	2	YES	No restrictions	No restrictions
7	2	16	16	1	16	32	16	1	1	1	NO	when interpolation ratio>2x	Not Supported
8	2	16	12	1	12	32	64	3	1	1	NO	when interpolation ratio>2x	Not Supported
9	2	8	8	1	16	32	32	1	1	1	YES	when interpolation ratio>1x	when interpolation ratio>2x
10	2	8	6	1	12	32	128	3	1	2	YES	when interpolation ratio>1x	when interpolation ratio>2x
11	2	4	4	1	16	32	64	1	if 4t4r & CxF=12, 2; else, 1	2	YES	No restrictions	when interpolation ratio>1x
12	2	4	3	1	12	32	256	3	2	2	YES	No restrictions	when interpolation ratio>1x
13	2	2	2	1	16	32	128	1	if 4t4r & CxF=6,25	4	YES	No restrictions	No restrictions

APOLLO MXFE TRANSMITTER

Table 81. DAC Path Supported JESD204B/C Modes per A and B-sides (Continued)

JESD204 Mode Number	L	M	F	S	N'	K (204B)	K (204C)	E (204C)	JRX_NS (204B)	JRX_NS (204C)	Dual link support per A/B	Mode Support Conditions	
												8T8R:AD9088	4T4R:AD9084
									0, or 350, 4; else, 2				
14	2	1	1	1	16	32	256	1	N/A	8	YES	when interpolation ratio=1x	when interpolation ratio=1x
15	3	12	8	1	16	32	32	1	1	1	NO	when interpolation ratio>2x	Not Supported
16	3	8	4	1	12	32	64	1	1	2	YES	when interpolation ratio>1x	when interpolation ratio>2x
17	3	6	4	1	16	32	64	1	if 4t4r & CxF=12, 2; else, 1	2	YES	when interpolation ratio>1x	when interpolation ratio>2x
18	3	4	2	1	12	32	128	1	if 4t4r & CxF=6, 4; else, 2	4	YES	No restrictions	when interpolation ratio>1x
19	3	3	2	1	16	32	128	1	N/A	4	YES	when interpolation ratio=1x	Not Supported
20	3	2	1	1	12	32	256	1	N/A	8	YES	No restrictions	No restrictions
21	3	1	1	2	12	32	256	1	N/A	16	YES	when interpolation ratio=1x	when interpolation ratio=1x
22	4	16	8	1	16	32	32	1	1	1	NO	when interpolation ratio>2x	Not Supported
23	4	16	6	1	12	32	128	3	1	2	NO	when interpolation ratio>2x	Not Supported
24	4	8	4	1	16	32	64	1	if 4t4r & CxF=12, 2; else, 1	2	YES	when interpolation ratio>1x	when interpolation ratio>2x
25	4	8	3	1	12	32	256	3	2	2	YES	when interpolation ratio>1x	when interpolation ratio>2x
26	4	4	2	1	16	32	128	1	if 4t4r & CxF=6, 4; else, 2	4	YES	No restrictions	when interpolation ratio>1x
27	4	4	3	2	12	32	256	3	if 4t4r & CxF=6, 4; else, 2	4	YES	No restrictions	when interpolation ratio>1x
28	4	4	1	1	8	32	256	1	4	8	YES	No restrictions	when interpolation ratio>1x
29	4	2	1	1	16	32	256	1	4	8	YES	No restrictions	No restrictions
30	4	2	3	4	12	32	256	3	4	8	YES	No restrictions	No restrictions
31	4	2	1	2	8	32	256	1	8	16	YES	No restrictions	No restrictions
32	4	1	1	2	16	32	256	1	N/A	16	YES	when interpolation ratio=1x	when interpolation ratio=1x
33	4	1	3	8	12	32	256	3	8	16	YES	when interpolation ratio=1x	when interpolation ratio=1x
34	4	1	1	4	8	32	256	1	16	32	YES	when interpolation ratio=1x	when interpolation ratio=1x
35	6	12	4	1	16	32	64	1	1	2	NO	when interpolation ratio>2x	Not Supported
36	6	8	2	1	12	32	128	1	2	4	YES	when interpolation ratio>1x	when interpolation ratio>2x
37	6	6	2	1	16	32	128	1	4	4	YES	when interpolation ratio>1x	when interpolation ratio>2x
38	6	4	1	1	12	32	256	1	8	8	YES	No restrictions	when interpolation ratio>1x
39	6	3	3	4	12	32	256	3	8	8	YES	when interpolation ratio=1x	Not Supported
40	6	3	1	1	16	32	256	1	N/A	8	YES	when interpolation ratio=1x	Not Supported
41	6	2	1	2	12	32	256	1	8	16	YES	No restrictions	No restrictions
42	6	1	1	4	12	32	256	1	16	32	YES	when interpolation ratio=1x	when interpolation ratio=1x
43	8	16	4	1	16	32	64	1	1	2	NO	when interpolation ratio>2x	Not Supported
44	8	16	3	1	12	32	256	3	2	2	NO	when interpolation ratio>2x	Not Supported
45	8	8	2	1	16	32	128	1	if 4t4r & CxF=6, 4; else, 2	4	NO	when interpolation ratio>1x	when interpolation ratio>2x

APOLLO MXFE TRANSMITTER

Table 81. DAC Path Supported JESD204B/C Modes per A and B-sides (Continued)

JESD204 Mode Number	L	M	F	S	N'	K (204B)	K (204C)	E (204C)	JRX_NS (204B)	JRX_NS (204C)	Dual link support per A/B	Mode Support Conditions	
												8T8R:AD9088	4T4R:AD9084
46	8	8	3	2	12	32	256	3	if 4t4r & CxF=6, 4; else, 2	4	NO	when interpolation ratio>1x	when interpolation ratio>2x
47	8	4	1	1	16	32	256	1	4	8	NO	No restrictions	when interpolation ratio>1x
48	8	4	3	4	12	32	256	3	if 4t4r & CxF=3, 8; else, 4	8	NO	No restrictions	when interpolation ratio>1x
49	8	4	1	2	8	32	256	1	8	16	NO	No restrictions	when interpolation ratio>1x
50	8	2	1	2	16	32	256	1	8	16	NO	No restrictions	No restrictions
51	8	2	3	8	12	32	256	3	8	16	NO	when interpolation ratio = 1x	No restrictions
52	8	2	1	4	8	32	256	1	16	32	NO	No restrictions	No restrictions
53	8	1	1	4	16	32	256	1	16	32	NO	when interpolation ratio=1x	when interpolation ratio=1x
54	8	1	3	16	12	32	256	3	16	32	NO	when interpolation ratio=1x	when interpolation ratio=1x
55	8	1	1	8	8	32	256	1	32	32	NO	when interpolation ratio=1x	when interpolation ratio=1x
56	12	16	2	1	12	32	128	1	2	4	NO	when interpolation ratio>2x	Not Supported
57	12	12	2	1	16	32	128	1	2	4	NO	when interpolation ratio>2x	Not Supported
58	12	12	3	2	12	32	256	3	2	4	NO	when interpolation ratio>2x	Not Supported
59	12	8	1	1	12	32	256	1	4	8	NO	when interpolation ratio>1x	when interpolation ratio>2x
60	12	6	1	1	16	32	256	1	4	8	NO	when interpolation ratio>1x	when interpolation ratio>2x
61	12	4	1	2	12	32	256	1	8	16	NO	No restrictions	when interpolation ratio>1x
62	12	4	1	3	8	32	256	1	12	24	NO	when interpolation ratio=1x	Not Supported
63	12	3	1	2	16	32	256	1	16	16	NO	when interpolation ratio=1x	Not Supported
64	12	3	3	8	12	32	256	3	8	16	NO	when interpolation ratio=1x	Not Supported
65	12	3	1	4	8	32	256	1	16	32	NO	when interpolation ratio=1x	Not Supported
66	12	2	1	4	12	32	256	1	16	32	NO	No restrictions	No restrictions
67	12	2	1	6	8	32	256	1	24	24	NO	when interpolation ratio = 1x	when interpolation ratio = 1x
68	12	1	1	8	12	32	256	1	32	32	NO	when interpolation ratio=1x	when interpolation ratio=1x
69	12	1	1	12	8	32	256	1	24	24	NO	when interpolation ratio=1x	when interpolation ratio=1x
70	4	4	2	2	8	32	128	1	4t4r -> 8, 8t8r -> 4	8	YES	only 204B and 3x	only 204B and 3x
71	4	2	2	2	16	32	128	1	4t4r -> 8,8t8r -> 4	8	YES	only 204B and 3x	only 204B and 3x
72	6	4	2	2	12	32	128	1	4t4r -> 8, 8t8r -> 4	8	YES	only 204B and 3x	only 204B and 3x
73	8	4	2	2	16	32	128	1	4t4r -> 8, 8t8r -> 4	8	NO	only 204B and 3x	only 204B and 3x
74	12	8	2	2	12	32	128	1	4	8	NO	only 204B and 3x	only 204B and 3x
75	12	6	2	2	16	32	128	1	4	8	NO	only 204B and 3x	only 204B and 3x
76	12	1	1	6	16	32	256	1	N/A	24	NO	only 204C and 1x	only 204C and 1x
77	12	2	1	3	16	32	256	1	N/A	24	NO	only 204C and 1x/2x	only 204C and 1x/2x

APOLLO MXFE TRANSMITTER

Table 82. Supported Interpolation Modes per JESD204B/C Modes

Total Interp	1	2	3	4	6	8	12	16	24	32	48	64	96	128	192	256	384	512	768
	Fine x Coarse Interpolation Modes																		
JESD204 Mode Number	1x1	1x2	1x3	1x4 2x2	1x6 2x3	2x4 4x2 1x8	1x12 2x6 4x3	4x4 8x2 2x8	2x12 4x6 8x3	4x8 8x4 16x2	4x12 8x6 16x3	8x8 16x4 32x2	8x12 16x6 32x3	16x8 32x4 64x2	16x12 32x6 64x3	32x8 64x4	32x12 64x6	64x8	64x12
0									C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C
1									C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C
2								C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C
3							C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B
4					C	C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B
5					C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B
6			C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B		
7								C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C
8							C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B
9					C	C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B
10					C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B
11			C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B		
12			C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B		
13		C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B			
14	C																		
15					C	C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B
16			C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B		
17			C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B		
18		C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B			
19	C																		
20	C																		
21	C																		
22					C	C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B
23					C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B
24			C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B		
25			C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B		
26		C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B			
27		C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B			
28	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B					
29	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B					
30	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B						
31	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B							
32	C																		
33	B,C																		
34	B,C																		
35			C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B		
36		C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B			
37		C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B			
38	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B					
39	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B						
40	C																		
41	C	B,C	B,C	B,C	B,C		B,C	B,C	B,C	B	B	B							
42	B,C																		

APOLLO MXFE TRANSMITTER

Table 82. Supported Interpolation Modes per JESD204B/C Modes (Continued)

Total Interp	1	2	3	4	6	8	12	16	24	32	48	64	96	128	192	256	384	512	768
	Fine x Coarse Interpolation Modes																		
JESD204 Mode Number	1x1	1x2	1x3	1x4 2x2	1x6 2x3	2x4 4x2 1x8	1x12 2x6 4x3	4x4 8x2 2x8	2x12 4x6 8x3	4x8 8x4 16x2	4x12 8x6 16x3	8x8 16x4 32x2	8x12 16x6 32x3	16x8 32x4 64x2	16x12 32x6 64x3	32x8 64x4	32x12 64x6	64x8	64x12
43			C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B		
44			C	C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B		
45		C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B			
46		C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B			
47	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B					
48	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B						
49	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B							
50	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B							
51	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B							
52	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B									
53	B,C																		
54	B,C																		
55	B,C																		
56		C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B			
57		C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B			
58		C	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B			
59	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B					
60	C	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B					
61	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B							
62	B,C																		
63	C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B							
64	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B	B							
65	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B									
66	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B,C	B	B									
67	B,C																		
68	B,C																		
69	B,C																		
70			B																
71			B																
72			B																
73			B																
74			B																
75			B																
76	C																		
77	C	C																	

TRANSMIT DIGITAL DATA PATH OVERVIEW

The architectures of the transmit digital data path of the AD9084 (Figure 94) and AD9088 (Figure 95) share many similar design elements. The key differences are highlighted in Table 1 in the beginning of this document.

The AD9088 has a total of eight transmit data paths, each routed to a single DAC core through a crossbar mux. The AD9084 has a total of four transmit data paths, which can run at a higher data rate compared to the AD9088. On the AD9084, the default DACs on each side are DAC_A0 and DAC_A3 on one side or DAC_B0 and DAC_B3 on the other. A variant of the AD9084 is available that makes all four DACs accessible on

APOLLO MXFE TRANSMITTER

each side, which could simplify PCB routing to achieve a target design specification to improve isolation between channels. While the AD9084 supports FSRC, the AD9088 does not.

Figure 94 and Figure 95 show a single side (A or B) of the Transmit Digital Data path located between the JESD204 receiver IP and the DAC cores. The AD9084 has a total of four transmit channels. The AD9088 has a total of eight transmit paths, each routed to a single DAC core.

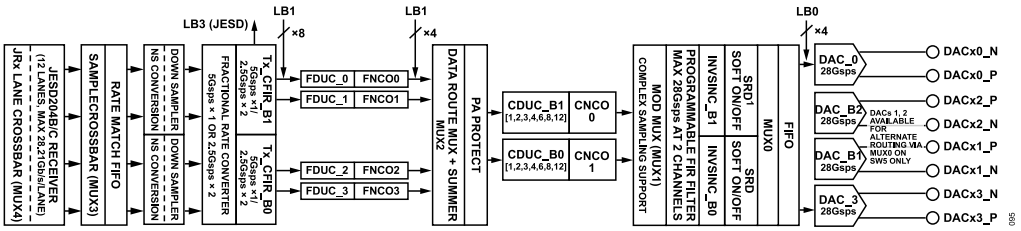

Figure 94. AD9084 transmit digital data path block diagram (per A/B-side)

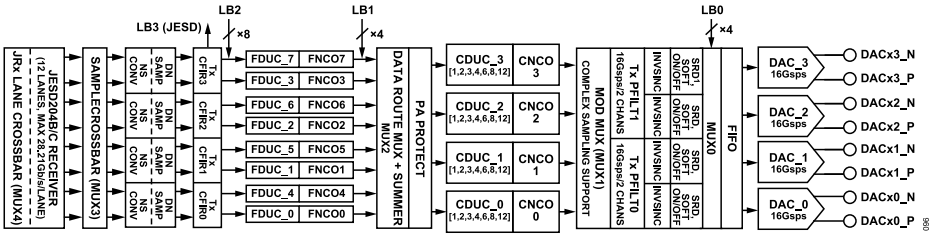

Figure 95. AD9088 transmit digital data path block diagram (per A/B-side).

The AD9084 and AD9088 include various Digital Signal Processing (DSP) blocks, such as the CFIR, FDUC filters, FNCO, CDUC filters, CNCO, MUX1, PFILT, INV SINC, and MUX0. The AD9084 additionally includes the FSRC block and the AUX DACs.

The AD9084 and AD9088 support a single-stage or two-stage up-conversion using the FDUC and CDUC blocks. Each digital up-converter (DUC) block, whether fine DUC (FDUC) or coarse DUC (CDUC), consists of a set of selectable cascaded interpolation filters followed by an optional, digital quadrature (I/Q) up-converter with a Modulus NCO for carrier modulation. The FDUC block supports interpolation factors of 2, 4, 8, 16, 32, and 64, while the CDUC block upstream supports interpolation factors of 2, 3, 4, 6, 8, and 12. The maximum output data rate of the AD9084 FDUC is 5Gsps and 2.5Gsps for the AD9088 FDUC. For higher input data rates, the FDUC must be bypassed.

The use of only the FDUC and CDUC chain provides access to integer interpolation rate. When combined with the rate conversion FIFO (RATECONV), and the Fractional Sample Rate Converter (FSRC) blocks, fractional interpolation rates are supported. With some restrictions due to clocking architecture, FDUCs and CDUCs may be also assigned different interpolation rates to support channels of different baseband data rates (bandwidths) across the JESD204B/C links: Referring to the AD9084 in Figure 94, the channel formed by FDUC0/1 and CDUC0 can have different data rates compared to the channel formed by FDUC2/3 and CDUC1. For example, FDUC0 and FDUC1 can be assigned an interpolation rate that is twice the rate of FDUC2 and FDUC3, when the following conditions are met:

1. The input data into FDUC0/1 and FDUC2/3 is supplied by separate JESD204B/C links (“dual-link”)
2. The output from FDCU0/1 is routed only to CDUC0 and the output from FDUC2/3 is routed to only CDUC1. If this is not true, then the data rate at the outputs of the FDUCs must be the same.
3. The output from CDUC0/1 must match the DAC core sample rate

This same feature exists in the AD9088, where FDUC7 and FDUC3, FDUC6 and FDUC2, together with CDUC3 and CDUC1, respectively, form a Tx channel grouping that can have different interpolation rates (and data rates) from the channel grouping formed by FDUC5 and FDUC1, FDUC4 and FDUC0, together with CDUC2 and CDUC0, when the above conditions are met.

The two-stage up-conversion allows carrier aggregation from multiple baseband channels that are transmitted across a single JESD204B/C link, where the carriers are to be up-converted as separate sub-bands in the FDUCs, summed, and up-converted again in one or more CDUCs to the target RF bands. Aggregated carriers must be located within the integrated bandwidth supported at the input of the CDUC. Alternatively, for wideband signals exceeding the FDUC data rate limitations, or when up-converting using a single FDUC, the unused FDUCs may be bypassed, so that the samples can be routed directly to the CDUC for processing.

APOLLO MXFE TRANSMITTER

When using the FSRC, up to two I/Q data paths can be supported by the FSRC block, and signals with a larger number of carriers must be aggregated before sending the data across the JESD204B/C link.

To facilitate flexible routing of data samples to the DAC cores, the transmit data path includes multiple multiplexer (MUX) stages, namely MUX2, MUX1, and MUX0. An addition MUX is located as part of the JESD204B/C receiver block, which allows re-mapping samples from the JESD204B/C core to the appropriate transmit data path channel, FSRC and/or CFIR blocks.

MUX2 is a crossbar that allows routing one or multiple FDUC outputs to one or multiple CDUC paths, which has a maximum complex data rate of 5000 MSPS in the AD9084 and 2500MSPS in the AD9088. This maximum data rate in effect sets the maximum allowed interpolation ratio in the FDUC block, for a given input data rate from JESD204B/C link.

MUX1, in combination with MUX0, allows separating the channels for further filtering and equalization, or summing multiple I/Q pairs into individual I and Q streams that are routed to individual DAC cores for signal reconstruction into an external mmWave I/Q Modulator. Various other configurations are also possible as described in subsection Tx Path Mux1 and Transmit Mux0.

Transmit Digital Data Path: Programming With the API

Example code for configuring the device, including the Tx data path, may be found in the \pkg\src\examples folder of the Apollo MxFE API.

Static configuration of the Transmit digital data path is performed by the device API using the appropriate parameters from the device profile. Specifically, the **adi_apollo_cfg_data_path()** function performs this task as shown in the Figure 12. The parameters associated with configuring the transmit data path are automatically generated and inserted into the device profile by the device profile generator tool, a component of the ACE evaluation software package for the Apollo MxFE. The parameters associated with configuring the transmit data path are described in each of the transmit path functional block subsections that follow.

Initial configuration changes of the transmit data path is accomplished by creation of a new profile. If the data path configuration is to be changed dynamically during operation, then Dynamic reconfiguration must be used by the end application. The Dynamic Reconfiguration section explains which functional blocks and their parameters can be dynamically reconfigured without bringing the JESD204B/C link down. The Configuration Updates After Device Bring-Up section describes the features that can and cannot be updated after the initial device bring-up.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

TRANSMIT PATH DSP FUNCTIONS (@ INTERFACE DATA RATE)

DSP functions operating at the interface rate include the functions within the Rate Match FIFO. This is where transmit data rate conversion takes place using the FSRC sub-block as well as the down-sampler sub-block. The transmit path CFIR function also operates at the interface data rate.

TRANSMIT DATA RATE CONVERSION

The AD9084 and AD9088 include DSP blocks that allow performing a rate conversion (interpolation) and/or a frequency shift (frequency mixing), to allow maximizing the clock rate of the DAC core while also minimizing the data rate across the JESD204B/C. Both integer and fractional rate conversions are supported. The fractional rate conversion (AD9084 only), combined with the ability to dynamically reconfigure the DSP blocks, allows the device to quickly switch between device configurations to accommodate multiple communication standards, channel bandwidths, and/or frequency plans.

Down-sampler, NS Converter, and Fractional Rate Conversion (FSRC) Blocks

The Down-sampler (DOWNSAMP), and NS Converter (NSCONV) blocks allow for different dynamically controlled interpolation rates without the need for changing the JESD204B/C data rate. The Rate Match FIFO (RATECONV) used together with the FSRC block, allow a fractional rate conversion between the JESD204B/C receiver block and the Transmit Data path downstream. Once data samples are received by the JESD204B/C receiver block and are released by the FIFO, an initial fractional data rate conversion, with a non-integer factor between 1.0 and 2.0, may be performed prior to the integer rate conversions performed in downstream DSP blocks, namely the FDUCs and CDUCs. Thus, the FSRC helps matching the running DAC rate to the JESD204B/C link data rate when the two rates have a non-integer relation.

When there is an integer relation between the required DSP data rate to the JESD204B/C link data rate, the FSRC block is not needed and may be bypassed.

APOLLO MXFE TRANSMITTER

When not using the FSRC block, the link data rate must be an integer-multiple of the DAC clock rate. This may limit the number of available clock rate options and topologies in a given application or require adjusting the SERDES lane rate if multiple frequency plans that do not share the same baseband rate are to be supported.

When using the FSRC block, the link data rate can have a fractional relation to the DAC clock rate, The FSRC block is shared between two Tx data paths with a maximum I/Q data rate of 5000 MSPS, to support up-to 5000 MSPS I and 5000MSPS Q with one data path or 2500 MSPS I and 2500 MSPS Q with two data paths. The block includes a 31-tap filter with a passband ripple <0.05dB, stopband rejection > 90dB, and a complex (I/Q) bandwidth equal to 80% of the input data rate. It is designed to allow input data rate conversion ratios between 1.0 and 2.0, not inclusive, in the form of 1+N/M, where {M and N}>0 and M≠N. in the API, it is programmed using the parameters fsrc_rate_int + fsrc_rate_frac_a / fsrc_rate_frac_b.

Once configured, the FSRC block can be dynamically reconfigured during device run-time without having to reinitialize the device or require reinitializing the JESD204B/C link.

Please see the Fractional Sample Rate Converter (FSRC) for Transmit and Receive (AD9084 Only) section for more details.

An API example for the TX JESD FSRC data path testing can be found at \src\examples\ads10_apollo_ex_main\tx_jesd_fsrc.c as part of the Apollo MxFE API package.

Down-sampler (DOWNSAMP) Block

Since many TX channels are transmitted across a single link, their link data rates must match so that the data may be correctly framed and deframed into octets. Related information on JESD204B/C receiver operation is available in the JESD204B/C Receiver Lane Rate Adaptation section. The TX channels within a single link may need different levels of interpolation and hence will have different data-rates. To match the link data rates for each channel, a common practice is to repeat the signal samples of the lower data-rate signals, thus matching the number of samples transmitted per second across all channels with a link. Once the signals are received and deframed by the JESD204B/C core, the DOWNSAMP block removes (drops) the repeated samples to recover the original signal. This allows the system to use different interpolation rates per channel within the same link as needed. The block is typically used in conjunction with integer rate interpolation when FSRC is disabled. It is also used during Dynamic Reconfiguration to support higher levels of interpolation when the JESD204B/C link rate is a factor of 2^N faster than the reconfigured data-rate.

The interpolation factors must be related by a factor of 2^N, and so the down-sampler drops $2^N - 1$ samples of the lower data rate channels (the one that includes the repeated samples) for each 1 sample of the highest data-rate channel. Thus, the host processor performs the opposite action of inserting these redundant $(2^N - 1)$ repeated samples.

Transmit Path CFIR Filter Block

The CFIR block contains a 16-tap complex FIR filter that may be extended to 128-taps in a sparse configuration in which 16 of the 128 taps are non-zero and to be modified by the user. Table 83 lists some of the key characteristics of the CFIR block. The maximum data throughput rate is 5000MSPS for 16-bit I and 16-bit Q samples and can accommodate 1, 2, or 4 data streams concurrently. Therefore, there can be a single data stream at 5000MSPS, two data streams at 2500MSPS or four data streams at 1250MSPS.

Table 83. CFIR Block Characteristics

Specification	Parameter	Comment
Max input data rate	5.0 GSPS	Can accommodate 1, 2, 3, or 4 data streams
Max output data rate	5.0 GSPS	Matches input data rate
# of filter taps	16	Complex filter taps
Maximum # of taps	128	This is sparse filter mode, and only 16 taps can be non-zero.
Input data sample width	16 bits	16-bit I & 16-bit Q
Coefficient bit width	16 bits	16-bit I & 16-bit Q
Output sample width	16 bits	16-bit I & 16-bit Q

At the filter output, there's a coarse gain block and then a complex scalar multiplier. This arrangement allows the user to filter and gain the baseband samples ahead of the FDUC block. The gain-adjust block that allows controlling the gain between -18dB and 12dB in 6dB increments, This gain stage is followed by a complex, 16-bit complex scalar block for fine gain adjustments, as shown in Figure 96, in the form of (I+jQ), where I and Q are programmable 16-bit constants.

APOLLO MXFE TRANSMITTER

Care must be taken to avoid sample-clipping where the CFIR output could exceed 16 bits, leading to signal distortion. See the Programmable Complex FIR Filter (CFIR) - Receive and Transmit section for more details on the CFIR block.

An API example for the TX CFIR can be found at \src\examples\ads10_apollo_ex_main\tx_jesd_cfir.c as part of the Apollo MxFE API package.

Table 84. 3-bit enumeration for setting the gain inside the CFIR

Gain_adj	Gain
3'b000	-18 dB
3'b001	-12 dB
3'b010	-6 dB
3'b011	0 dB
3'b100	+6 dB
3'b101	+12 dB

Figure 96. Inside the CFIR block: showing a 16-tap complex FIR, Gain adjust, and the complex scalar multiplier

TRANSMIT FINE DIGITAL UP-CONVERTER (FDUC)

Frequency translation is accomplished with a complex NCO and a digital quadrature (complex) mixer, as shown in Figure 97, where the complex input data after the interpolation filter is multiplied by the NCO complex exponential frequency ($e^{-j\omega ct}$) data output, such that the input spectrum (represented by the input data) is shifted to center around the desired carrier frequency (fc). The fc is generated by the NCO according to a frequency tuning word (FTW) specified by the user. The NCO data-rate , f_{NCO}, is the same as the FDUC output data rate, $f_{IQ\_OUT\_CH}$ such that $f_{NCO} = f_{IQ\_OUT\_CH} = N_{TX} \times f_{IQ\_IN}$, where f_{NCO} is the running clock rate of the NCO and N_{TX} is the total interpolation ratio of the FDUC filters. As a result, the tuning range for the NCO is between $-f_{NCO}/2$ and $+f_{NCO}/2$, although from a practical perspective, the usable frequency tuning range of the FDUC is 80% of the NCO range, due to the pass band response of the interpolation filters downstream.

The FDUC is located upstream, before the MUX2 and CDUC blocks, and thus the maximum data rate at the output of the FDUC Filter block (after interpolation) may not exceed the maximum data rate of the MUX2 block. To avoid this constraint, the FDUC and the MUX2 blocks may be bypassed.

The FDUC is initially configured by the **adi_apollo_tx_fduc_configure()** function based on the parameters supplied in the device profile. This function is called by the high-level **adi_apollo_cfg_data_path()** function that is part of the Apollo MxFE Bring-up Procedure. The FDUC blocks (FDUC Filters, FNCO) may be dynamically reconfigured during device run-time without having to reinitialize the device or interrupt the JESD204B/C link. The NCO FTW may be adjusted dynamically, in response to either an internal or an external trigger signal, to maintain phase coherency and a synchronized phase relation between carriers or across multiple devices. The new parameters may be loaded at any time prior to the update trigger. See the CDUC/FDUC Dynamic Reconfiguration section for more details on dynamic reconfiguration of the FDUC block.

The NCO Phase Coherence (NPC) counter and various triggers simplify implementing phase coherence.

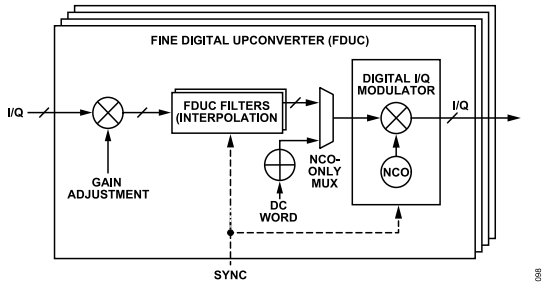

Figure 97. Top-level diagram for the FDUC block

APOLLO MXFE TRANSMITTER

Fine Digital Up-converter Filters (FDUC Filter) and Gain-Adjust Block

The cascaded interpolation filter lineup shown in Figure 98 may be combined to form an interpolation factor of 2, 4, 8, 16, 32 or 64. The filter chain can also be bypassed (1x interpolation). The maximum $f_{IQ\_OUT\_CH}$ out of the interpolation cascade is limited to 5000 MSPS in the AD9084 and to 2500MSPS in the AD9088, with a maximum instantaneous bandwidth equal to 80% of f_{IQ_IN}.

The usable pass-band bandwidth is defined as the frequency band over which the response maintains a pass-band ripple of less than ±0.001 dB, with a stop-band alias (or image) rejection of >85 dB. The filter characteristics for each stage are listed in Table 85. The filtering stages were optimized to reduce power consumption while maintaining a total pass band bandwidth of at least 80% of the f_{IQ_IN} out of the initial interpolation stage (HB1).

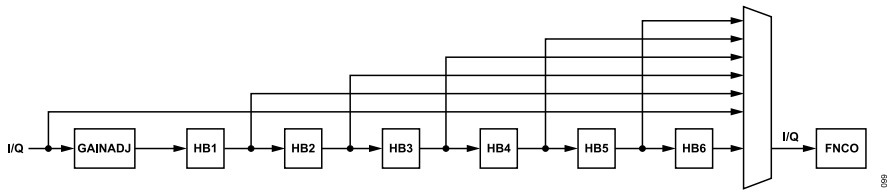

Figure 98. Transmit FDUC Data Path Interpolation Filter Lineup

Table 85. Performance characteristics of the channelizer interpolation Filter stages

Filter Name	Decimation Ratio	Pass-Band Bandwidth (% of f_{IQ_IN})	Pass Band Ripple (dB)	Stop Band Rejection (dB)	Total Pass-Band Bandwidth of the line-up (% of f_{IQ_IN})[1]
HB1	2	80	< ±0.001	>85	80
HB2		40			
HB3		20			
HB4		10			
HB5		5			
HB6		2.5			

[1] The filter BW is quoted relative to the output IQ rate of the filter, as the total bandwidth available to the user. For a non-IQ (real) signal, the filter bandwidth is 40% of its input data rate.

The interpolation settings are configured as part of **adi_apollo_tx_fduc_configure()** based on the **adi_apollo_fduc_ratio_e** enumeration set in tx_fduc.drc_ratio in the device profile. Interpolation can also be triggered to reconfigure dynamically via the method described in the CDUC/FDUC Dynamic Reconfiguration section.

Digital Gain Adjust

The data passing through the FDUC Filter block can be rescaled prior to interpolation and additional processing. This feature is useful in multiband applications to prevent digital clipping when the outputs of two or more 16bit FDUCs are summed into a single 16bit CDUC stage to produce a multiband signal.

To calculate the gain or the gain code, use the following formulas:

$$\text{Gain Code} = 2048 \times \text{Gain} = 2^{11} \times 10^{(\text{dBGain}/20)} \quad (27)$$

$$\text{Gain} = \text{Gain Code} \times (1/2048) \quad (28)$$

$$0 \le \text{Gain} \le (2^{12} - 1)/2^{11} \quad (29)$$

$$\text{dB Gain} = 20 \times \log_{10}(\text{Gain}) \quad (30)$$

$$-\infty \text{ dB} < \text{dBGain} \le +6.018 \text{ dB} \quad (31)$$

Setting the Gain Code = 0x800 corresponds to dBGain = 0dB.

Gain settings are configured as part of **adi_apollo_tx_fduc_configure()** based on the device profile settings for the parameter **tx_fduc.sub_dp_gain_enable** (true/false) and the gain code set in **tx_fduc.sub_dp_gain** parameter.

APOLLO MXFE TRANSMITTER

Fine Digital Up-converter Modulator (FDUC) and Fine NCO (FNCO)

The FDUC has a digital I/Q modulator, that includes a 48-bit complex Fine NCO (FNCO) for carrier generation along with summers and multipliers to result in a frequency translation and sideband cancellation, as may be seen in Figure 99. It performs the same frequency translation as a traditional analog I/Q modulator with a built-in Local Oscillator (LO) source. Being digital, it does not suffer from LO leakage and sideband suppression issues.

Fast Frequency Hopping: This is a special use-case for the FNCO, that allows quickly switching between pre-programmed FTW and Phase pairs. When in this mode, the FTW resolution is 32-bit and the Phase word resolution is 16-bit. More details are available in the FNCO Fast Frequency Hopping (FFH), Hop Profiles (HP), and Phase Coherence section <add link>.

I/Q Modulator vs. a traditional Mixer: The FDUC may be configured to operate as an I/Q Modulator (complex) or as a traditional non-IQ mixer (real-only). Operating the FDUC as a mixer is useful in applications where only real samples are transmitted across the JESD204B/C link. Such operation will result in sidebands, which may need to be filtered externally to the device output – to avoid the need for filtering I/Q samples must be transmitted, at the expense of doubling the link data rate for a given signal bandwidth.

The 48-bit complex FNCO supports the following modes of operation designated by the **adi_apollo_nco_mode_e** enumeration assigned to the nco_mode parameter of the profile tx_fduc object. Currently this parameter is not accessed during device startup and the setting is left at the default value of 0, or ADI_APOLLO_NCO_MOD_INTEGER. A function to implement dual modulus mode is TBD.

NCO Modes

- Integer-N mode: the two's complement, 48-bit frequency tuning word (FTW) and the 48-bit initial phase offset word control the frequency and phase offset, respectively. The frequency resolution is limited to a single LSB of the 48bit binary FTW. For more details see the Integer Mode NCO section.
- Dual modulus (fractional-N) mode: This mode allows a higher frequency resolution that is otherwise not possible using the Integer-N mode. In this mode the FTW sets the initial NCO output frequency, while the remaining fractional frequency step is set using the A/B ratio. For more details see the Dual Modulus NCO section.
- NCO-only operation: In either of the two modes above, the FNCO may receive its input either from the JESD204B/C link, or from an internal dc samples generator, to result in a continuous wave (CW) tone at the NCO output. This operation may be performed during run-time, without resetting the device, and is typically referred to as NCO-only mode or DC-test mode. It may be useful for testing, debug purposes, or to operate the NCO and the DAC core as a DDS.

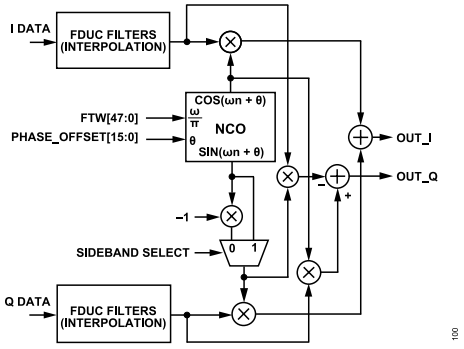

Figure 99. FNCO diagram, also showing the FDUC Filters at the input of the FNCO

The NCO output frequency may be expressed as a ratio M/N, relating the output frequency to the running clock rate of the FNCO:

$$\frac{f_{CARRIER}}{f_{NCO,CLK}} = \frac{M}{N} = \frac{X + \frac{A}{B}}{2^{Bits}} \quad (32)$$

where:
M/N is the normalized ratio of FCARRIR and FNCO,CLK. Note that N and M both must be <= 2^{24}.
X is the FTW, representing the integer-N portion.
A is the delta value that represent the numerator of the fractional-N portion. Set it equal to mod(M*2^{Bits}, N).
B is the modulus value that represent the denominator of the fractional-N portion. Set it equal to N.
Bits is the bit resolution of the NCO (the NCO's FTW). When using FFH, the FTW resolution is 32-bit; without FFH, the resolution is 48-bits.

APOLLO MXFE TRANSMITTER

$f_{NCO,CLK}$ is NCO's running clock frequency, which is equal to the data rate at the output of the FDUC Filters block.
$f_{CARRIER}$ is the desired output frequency of the NCO.

Dual Modulus NCO

This mode is enabled when A≠0 and B≠0. The ratio A/B sets the accuracy of the NCO beyond the 48-bit accuracy possible with integer-N mode. This may be of interest in applications where the NCO remains functional for prolonged periods of time and may accumulate a noticeable phase drift relative to clocks in the system, even given the small initial frequency error of the 48-bit, integer NCO. This error exists because analog clocks are typically designed to operate at frequencies that are base-10 (e.g. 150.0 MHz) while the output frequency of the digital NCO is set by a base-2 (binary) FTW.

To remove this error, the final LSB of the FTW is represented as a ratio A/B. For more details, refer to the AN-953 Application Note and in particular the section titled "CALCULATING THE MODULUS PARAMETERS: A, B, AND X". A summary of the description is provided by the following example:

Consider the case in which $f_{NCO,CLK}$ = 1500 MHz and the desired value of $f_{CARRIER}$ is 150 MHz, which are base-10, non-binary values that results in M/N = $f_{CARRIER}$ / $f_{NCO,CLK}$ = 1/10. Since an exact ratio of 1/10 is not possible with a typical integer-N NCO, a dual-modulus mode must be used.

The following equation apply for Modulus NCO (Note that N should be chosen such that it never exceeds 2^24):

$$Fin/Fs = M/N = (FTW+ (frac\_a/frac\_b))/2^{Bits} \quad (33)$$

where "Bits" is the resolution in bits and:

- For non-hopping case
  - FTW= floor (Fin/Fs * 2^48) : Integer part
  - frac_a = mod(M* 2^48,N)
  - frac_b = N
- For hopping case
  - FTW= floor (Fin/Fs * 2^32) : Integer part
  - frac_a = mod(M* 2^32,N)
  - frac_b = N

Using #unique_81/unique_81_Connect_42_fig_N10059_N10011_N10001 for a 48-bit NCO and plugging in the values $f_{NCO,CLK}$, $f_{carrier}$, M, and N from above, yields:

- X (FTW)=28,147,497,671,065
- A (frac_a)=3
- B (frac_b)=5

Programming these values into the NCO yields an output frequency of exactly 150 MHz, so long that the NCO clock remains 1500 MHz.

Integer Mode NCO

In integer mode, the ratio A/B in #unique_81/unique_81_Connect_42_fig_N10059_N10011_N10001 is set to zero, such that A=0, which is set by nco_phase_inc_frac_a as part of the fduc configuration structure **adi_apollo_fduc_cfg_t**. In this mode the NCO output frequency is controlled solely by the FTW, with a resolution that depends on the bit resolution of the 48-bit phase accumulator.

NCO-Only Mode

In NCO-Only mode, the NCO generates the carrier while the FDUC receives its input from an internal dc word generator, as shown in Figure 97, resulting in a complex, CW output tone at the output of the FDUC. In this mode, the FDUC is also capable of generating waveforms other than sinewave, such as a square wave, sawtooth, or a triangular wave. The NCO is fully functional in this mode, and the phase (nco_phase_offset) and amplitude (dc_testmode_value) are adjustable, in an operation akin to a traditional Direct Digital Synthesizer (DDS).

The parameter dc_testmode_value sets the amplitude of the tone in 2's complement format where:

- Dc_testmode_value = 0 == 0x0 == 0mA
- Dc_testmode_value = 32512 == 0x7F00 corresponds to positive Full-scale AC current
- Dc_testmode_value = -32768 == 0x8000 corresponds to negative Full-scale AC current

This mode is a convenient method to operate the FDUC as a DDS, for applications that require multiple single-tone signals of a varying frequency and amplitude, summed into one or more CDUC paths. Applications that only require a single tone may benefit from using the NCO inside the CDUC block instead, to reduce power consumption and to allow operation at a higher clock rate.

Note that even when the JESD204B/C link is set up and data is properly transferred to the device over the link, this data is not presented to the FDUC until this mode is disabled. However, NCO modes may be toggled during runtime for debug purposes.

FNCO Fast Frequency Hopping (FFH), Hop Profiles (HP), and Phase Coherence

The NCO frequency and phase settings can be stored as a set of up-to 32 profiles, each with a 32-bit FTW and 16-bit phase offset word, to be selectively assigned to the NCO during runtime. This allows to quickly change (hop) the NCO frequency, known as Fast Frequency Hopping (FFH). The range over which the profiles are hopped is also programmable (i.e. it may be programmed to sequentially cycle through any number of profiles, up to 32). Independently hopping the frequency and phase offset is also possible.

Profile hopping may be controlled manually using the **adi_apollo_fnco_active_profile_set()** function. Hopping may be incremented/decremented in response to an external trigger, or in response to an internal trigger event synchronized with the global timestamp counter, which is distributed throughout the chip. The hop may be also performed directly, by assigning the next HP to apply to the FNCO using **adi_apollo_fnco_next_hop_num_set()**, to be triggered by a SPI/HSCI command, a GPIO assertion, a trigger pin, or a trigger issued by the timestamp counter. In order to time the **adi_apollo_fnco_next_hop_num_set()** relative to the internal timestamp counter, the timing of the current hop can be extracted from the timestamp counter to a GPIO to alert the user that its ok to update to next hop profile.

The manner in which the phase is transitioned during a hop event is configurable: The phase may be reset to the initial value programmed for each profile, the phase may smoothly transition between frequencies, or the phase may remain coherent relative to the initial reset event of the NCO (the phase accumulators for all the HP's continuously runs in the background so that the phase information is retained).

Direct Hopping

To use this mode, set the hop_mode_en parameter to 0 in the **adi_apollo_fnco_cfg_t** structure of the device profile at startup, or the mode can be set directly after startup by calling **adi_apollo_fnco_hop_enable()** function and setting the hop_en argument to 0.

The shadow registers to directly assign the FTW and phase offset to the NCO are listed in Table 86. The FTW and phase offset can be set with independent APIs as listed in the table, while the dual modulus parameters are set as part of **adi_apollo_fnco_main_pgm**() using the drc_phase_inc_frac_a/b parameters of the **adi_apollo_fnco_pgm_t** structure.

Table 86. NCO shadow registers to directly set the output frequency and phase

Parameter	Parameter Name	API	Description
FTW	MAIN_PHASE_INC	adi_apollo_fnco_main_phase_inc_set()	48-bit Frequency Tuning Word
Phase	MAIN_PHASE_OFFSET	adi_apollo_fnco_main_phase_offset_set()	48-bit phase offset word. 1 LSB = 360/2^48
A	DRC_PHASE_INC_FRAC_A	adi_apollo_fnco_pgm()	24-bit parameter A, to set frequency in dual-modulus mode
B	DRC_PHASE_INC_FRAC_B	adi_apollo_fnco_pgm()	24-bit parameter B, to set frequency in dual-modulus mode

FFH and HPs

To use HPs and FFH, set.*hop_mode_en*=1'b1 in the device profile fduc NCO section or by use **adi_apollo_fnco_hop_enable()** with the hop_en argument set to 1. The profiles are paged and programmed using the **adi_apollo_fnco_chan_pgm()** function. This function takes an **adi_apollo_fine_nco_chan_pgm_t** structure containing a FTW (.drc_phase_inc) and phase offset (.drc_phase_offset) parameters along with the FNCO's HP number to be configured.

Note that when using FFH, the assignments to the A/B ratio in dual-modulus mode applies to any of the selected HPs. However, the A/B values may be modified ahead of a hop, with the change to occur in response to one of the following events, according to bitfield MOD_PARAM_SCH:

- 0x0: assign A/B to the NCO on the next reset of the timestamp counter
- 0x1: assign A/B to the NCO on the next hop
- 0x2: assign A/B to the NCO in response to an API command to set bitfield FORCE_USE_AB=0x1

APOLLO MXFE TRANSMITTER

The hop process options are selected by running **adi_apollo_fnco_hop_pgm()** which takes an **adi_apollo_fine_nco_hop_t** structure. The following enumerations are used in the structure to choose settings for the hop behavior:

The **adi_apollo_nco_profile_sel_mode_e** parameter in the **adi_apollo_fnco_cfg_t** structure is used to select how a hop is triggered:

- ADI_APOLLO_NCO_CHAN_SEL_TRIG_AUTO: Auto Hop: a hop occurs automatically in response to a synchronized internal trigger.
- ADI_APOLLO_NCO_CHAN_SEL_TRIG_REGMAP: Scheduled Register Map hop: The user arms a hop event via the Register Map (or API) to occur on the next edge of a synchronized internal trigger. The trigger signal may be initially synchronized using SYSREF.
- ADI_APOLLO_NCO_CHAN_SEL_TRIG_GPIO: Scheduled GPIO hop: The user arms a hop event via a GPIO to occur on the next edge of a synchronized internal trigger. The trigger signal may be initially synchronized to an external SYSREF edge.
- ADI_APOLLO_NCO_CHAN_SEL_DIRECT_GPIO: Direct GPIO hop: The user controls a hop event to occur directly in response to a GPIO edge, asynchronously.
- ADI_APOLLO_NCO_CHAN_SEL_DIRECT_REGMAP: Direct Register Map hop: The user requests a hop event to occur asynchronously by calling adi_apollo_fnco_active_profile_set() and supplying the profile index.

The enumeration **adi_apollo_fnco_trig_hop_sel_e** is used to select whether phase and frequency should be hopped together or independently when using an internal trigger for hopping in #2 and #3 above:

- ADI_APOLLO_FNCO_TRIG_HOP_FREQ: hop only the frequency
- ADI_APOLLO_FNCO_TRIG_HOP_PHASE: hop only the phase
- ADI_APOLLO_FNCO_TRIG_HOP_FREQ_PHASE: hop both frequency and phase

item #5 above may be used to debug and configure the profiles the user is interested in.

API examples for FNCO profile hopping can be found in the examples\ ads10_apollo_ex_main folder as part of the Apollo MxFE API package. Among them are

- tx_nco_ffh.c
- fullchip_hop.c
- tx_jesd.c

FDUC Programming in API

Refer to the section NCO Programming and Debug.

TRANSMIT MUX2 + SUMMER

MUX2 is a crossbar multiplexer designed to interface the FDUC blocks with the CDUC blocks upstream, as shown in Figure 100 and Figure 101. MUX2 defines the mapping of complex data transferred between the FDUC outputs and the CDUC inputs of AD9084 or AD9088, and the same signal-chain is repeated in both the B-side and the A-side of the device variants. The AD9084 has four FDUC blocks and two CDUC blocks per side, while AD9088 has eight FDUC blocks and four CDUC blocks per side. Any combination of FDUC outputs can be summed together and mapped to any available CDUC inputs. Some constraints apply when in dual-link mode and the interpolation rates are different across the different FDUCs (the FDUC output data rates do not match). These constraints are as follows:

1. AD9084: FDUC_A0 and FDUC_A1 can only be mapped to CDUC_A0, and FDUC_A2 and FDUC_A3 can only be mapped to CDUC_A1
2. AD9088: FDUC_A0/A4/A1/A5 can only be mapped to CDUC_A0 and CDUC_A2, and FDUC_A2/A6/A3/A7 can only be mapped to CDUC_A1 and CDUC_A3

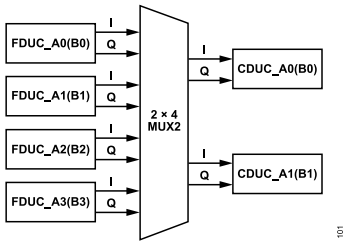

Figure 100. B-side/A-side MUX2 block, showing a 4:2 mux of the AD9084

APOLLO MXFE TRANSMITTER

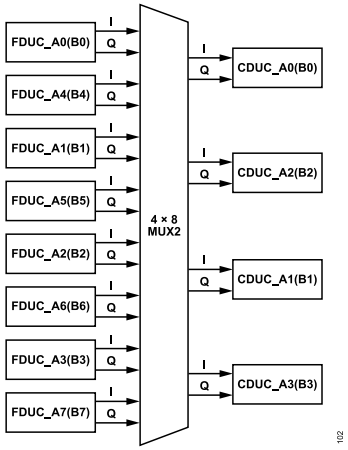

Figure 101. B-side/A-side MUX2 block, showing the 8:4 mux of the AD9088

TRANSMIT COARSE DIGITAL DATA PATH

The CDUC Filters block and the CDUC block share many similarities to the FDUC Filters block and FDUC block, as can be seen in Figure 102. One particular difference is that the coarse data path does not include a gain stage as part of the CDUC Filters block, however a gain stage is available as part of the PA-protect block (evaluation is ongoing, and more information to follow in future revisions of this UG). Other differences, if any, will be outlined in the subsections of this section.

The CDUC is initially configured by the **activate_rx_tx_blocks()** function based on the parameters supplied in the device profile. This function is called by the high-level **adi_apollo_cfg_data_path()** function that is part of the Apollo MxFE Bring-up Procedure. The CDUC blocks (CDUC Filters, CNCO) may be dynamically reconfigured during device run-time without having to reinitialize the device or interrupt the JESD204B/C link. The CNCO FTW may be adjusted dynamically, in response to either an internal or an external trigger signal, to maintain phase coherency and a synchronized phase relation between carriers or across multiple devices. The new parameters may be loaded at any time prior to the update trigger. See the "CDUC/FDUC Dynamic Reconfiguration" section for more details on dynamic reconfiguration of the CDUC block. The CNCO can be configured into several different operational modes as outlined in Table 87.

Table 87. CNCO Mode Enumerations

Enumeration	Value	Mode	Description
ADI_APOLLO_MXR_VAR_IF_MODE	0	Variable IF	Modulate anywhere in a nyquist zone
ADI_APOLLO_MXR_ZERO_IF_MODE	1	Zero IF	Bypass Modulator and keep all CDUC signals centered on zero
ADI_APOLLO_MXR_FS_BY_4_MODE	2	Fs/4 IF	Modulate by Fs/4
ADI_APOLLO_MXR_TEST_MODE	3	Test Mode	Modulate DC input code anywhere in nyquist zone

Example code utilizing this function can be found in \example\ads10_apollo_ex_main\tx_nco.c.

The CDUC Filters output may be passed directly to the CDUC output, fully bypassing the modulation stage and the NCO. This is useful in cases when modulation is not needed, and yet the user desires to operate the DAC at a higher rate compared to the baseband data rate received from the JESD204B/C link.

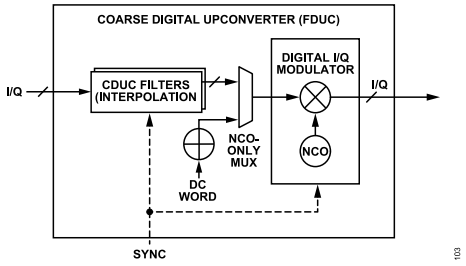

Figure 102. Top-level diagram for the CDUC block

APOLLO MXFE TRANSMITTER

COARSE DIGITAL UP-CONVERTER (CDUC) FILTERS

The CDUC Filters block is conceptually similar to the FDUC Filters block, with the main difference being in how the filters are cascaded to form higher interpolation rates. The supported interpolation rates are 2, 3, 4, 6, 8 and 12, as well as a bypass option (interpolation=1) as shown in Figure 103. Table 88 shows the characteristics of each filter pair. The usable pass band is the frequency band over which the response maintains a pass-band ripple of less than 0.001 dB with an alias (or image) rejection greater than 100 dB.

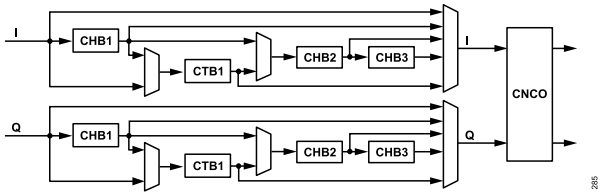

Figure 103. CDUC Interpolation Filter Lineup

Table 88. CDUC Interpolation Filter Characteristics

Filter Name	Interpolation Ratio	Pass-Band Bandwidth (% of f_{IQ_OUT})	Pass Band Ripple (dB)	Stop Band Rejection (dB)	Total Pass-Band Bandwidth of line-up (% of f_{IQ_OUT})[1]
HB1	2	80	< ±0.001	>100	80
TB1	3	80			
HB2	2	40			
HB3	2	20			

[1] The filter BW is quoted relative to the output IQ rate of the filter, as the total bandwidth available to the user. For a non-IQ (real) signal, the filter bandwidth is 40% of its input data rate.

COARSE DUC (CDUC) AND NUMERICALLY-CONTROLLED OSCILLATOR (CNCO)

The CNCO is similar in concept to the FNCO, and the descriptions in section Fine Digital Up-converter Modulator (FDUC) and Fine NCO (FNCO) generally applies to the CDUC and the CNCO as well, aside from a few differences as outlined below.

A particular difference is that the CNCO uses 32-bit wide FTW, along with 32-bit A/B values for modulus operation, unlike the 48-bit FTW and 24-bit A/B of the main FNCO phase accumulator.

Integer-mode, dual-modulus mode, and NCO-only mode are supported by the CDUC, just as described in the FDUC section.

Similar to the FDUC, the CDUC operates on complex (I/Q) data, and it is capable to process real-only data just like a traditional analog RF mixer would. To configure the CDUC for real-only operation set the **adi_apollo_drc_mixer_sel_e** parameter to ADI_APOLLO_DRC_MIXER_REAL (0x0) using the **adi_apollo_cnco_mixer_set()** API function.

A conceptual block diagram of the CNCO and associated synchronization and trigger blocks is shown in Figure 104.

APOLLO MXFE TRANSMITTER

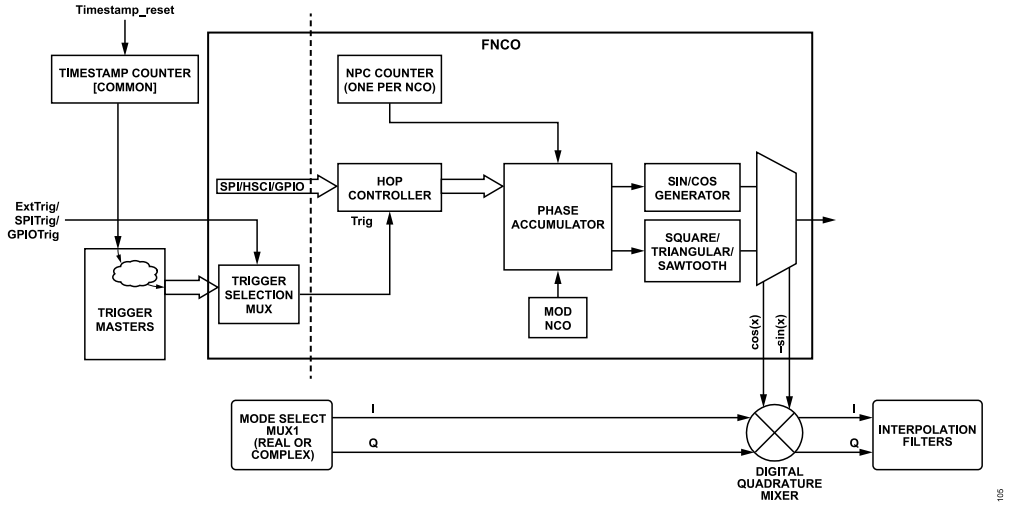

Figure 104. CNCO Operational Diagram with Phase Accumulator Block, Hop Control, Triggers, and Shared Phase Synchronization Blocks

NCO Only Mode

The NCO only mode implementation is identical in the CDUC to the FDUC, with one exception. A particular use-case arises when complex (I/Q) operation is combined with NCO-only mode, for a CDUC that may also operate with real-only input. In NCO only mode, the user has direct control over the amplitude of the CDUC output by setting the DC word at the input of CDUC. As part of the up-conversion process, the I and Q streams are summed which may result in sample overrange when operating at full-scale. Hence the CDUC has a scaling block that attenuates in samples by 3dB and maintains the CDUC output at full-scale without over-ranging.

The scaling block is enabled in the **adi_apollo_cnco_pgm()** function by the setting of the tx_cduc.nco.cmplx_mxr_mult_scale_en parameter in the device profile. ADI_APOLLO_CMPLX_MULT_SCALE_1 allows full-scale input and ADI_APOLLO_CMPLX_MULT_SCALE_0p7 attenuates the input samples, thus avoiding over-ranging the samples at the output.

CNCO Fast Frequency Hopping (FFH), Hop Profiles (HP), and Phase Coherence

While the FNCO can be assigned frequency-phase pairs from different HPs (for example the FTW from HP0 and the phase from HP1), the CNCO can only receive a specific HP pair. However, it is possible to configure the HP's frequency-phase pairs such that only the frequency or only the phase will be adjusted.

The phase to amplitude converter at the output of the phase accumulator, uses a LUT-based phase-to-amplitude converter, which has a data bit width of 16-bits and an address bit width of 7-bits. Additionally, each NCO uses a timestamp synchronization LUT as shown in Figure 104. The two LUTs improves the user's control over synchronization and phase coherence.

Similar to the FNCO, the NCO Phase Coherence (NPC) counter serves as a timing / phase reference to all the CNCOs inside the device. The NPC runs at the rate of $f_{NCO,CLK}/32$.

Refer to the fullchip_hop.c programming example in the ads10_apollo_ex_main folder of the Apollo MxFE Api package.

CNCO Programming in API

Refer to the section NCO Programming and Debug.

TX PATH MUX1

MUX1 is located at the input of the PFILT block and allows routing one or multiple I/Q pairs to the PFILT and INVSINC blocks for equalization, ahead of MUX0 which routes the equalized I and/or Q pairs to the individual DAC cores.

The inputs to MUX1 can be complex (I/Q) or real and receive the outputs of one or two CDUCs. The outputs of the multiple CDUCs may also be summed into a single DAC core, in which case they are rescaled by a factor of ½ to prevent digital sample overflow.

APOLLO MXFE TRANSMITTER

The configuration of MUX1 is defined by the end-application. Consider using a real-only output from MUX1 if the signal is routed to a single DAC core and independent I and Q signals are not needed at the output of the device. Alternatively, consider using a complex (I/Q) output if the reconstructed I and Q pairs are to be routed to individual DAC cores ahead of an additional (external) mmWave IQ modulator for direct, wideband zero-IF or complex-IF topologies.

The MUX1 input may receive real, non-I/Q samples, for applications where real-only samples from the JESD204B/C transport layer are to be interpolated to ease analog filtering requirements downstream from the DAC output.

MUX1 may be also configured to sum two CDUCs, for applications that require multiple wideband bands, wider than the bandwidth the FDUC channels can support.

TRANSMIT PATH DSP FUNCTIONS (@ DAC SAMPLE RATE)

As illustrated in Figure 105, the outputs of the CDUC blocks into MUX1 and other downstream blocks are running at the DAC sample rate. The outputs of these blocks are multiplexed using MUX0 for output into individual DAC cores.

Note that only MUX1 can sum individual CDUC outputs.

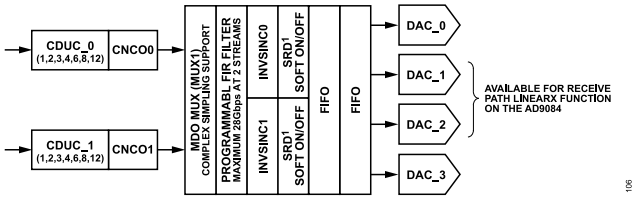

Figure 105. Showing one Apollo MxFE side, Tx path, between two CDUC outputs and four DAC inputs

Tx Path PFILT

Both the transmit and receive path share the same PFILT design. See the Transmit/Receive Programmable Filter (PFILT) section for details on this functional block.

Tx Path Soft On/Off and Slew Rate Detection

The Apollo MxFE incorporates a slew rate detection (SRD) block to detect the difference between two successive DAC input samples and prevents overrange signals from being passed through the output of the DAC thus protecting power sensitive devices, such as PAs. The protection function provides a signal that can ramp down the DAC inputs. This block supplements the PA Protection function and more details can be found in the Transmit Downstream Power Amplifier Protection section.

Inverse Sinc (INVSINC) Block

The INVSINC block allows improving the spectral flatness of the digital signal before it is sent to the DAC core for reconstruction. It receives its input from the PFILT blocks.

To improve the DAC high frequency gain flatness, the AD9084 and AD9088 have an INVSINC block that can equalize the DAC sinc roll-off. The sinc (or sin(x)/x) roll-off is fundamental to the sampling process, and results in a gain roll-off as the signal frequency approaches the DAC Nyquist boundary. The INVSINC block is a digital equalization filter that attenuates lower frequencies to equalize their level relative to higher frequencies. In applications where better equalization is necessary (also known as “spectral flatness”), the roll-off is undesired and may be reduced at the expense of signal power at lower frequencies, as shown in Figure 106 with a 3dB loss at DC and a 2dB loss at about 50% bandwidth. The loss decreases to 0 at about 90% bandwidth. The filter maintains passband ripple of <0.1dB within a real (non-I/Q) bandwidth of up to 40% of the filter's sample rate. The filter may be operated up to 45% bandwidth if a passband ripple if >0.1dB is acceptable.

The filter is non-I/Q, which implies that correction is possible only in the 1st Nyquist zone.

APOLLO MXFE TRANSMITTER

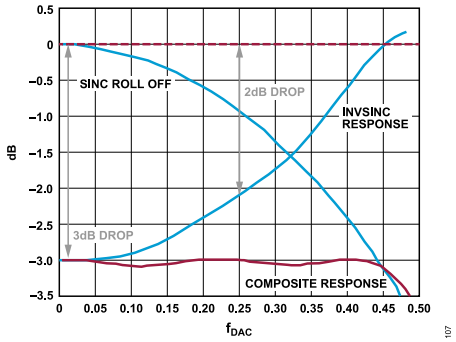

Figure 106. INVSINC Response of B-side and A-side

INVSINC Enable in API

At device startup INVSINC is enabled or disabled using the **adi_apollo_tx_inv_sinc_configure()** function based on the value of inv_sinc_en parameter in the **adi_apollo_txpath_t** structure of the device profile. INVSINC configuration must be done during the data path configuration portion of the Apollo MxFE Bring-up Procedure section, before the Apollo MxFE is issued a one-shot sync request,. INVSINC can also be updated after startup using the same **adi_apollo_tx_inv_sinc_configure()** function with the enumerations **adi_apollo_side_e** and **adi_apollo_cduc_idx_e** to specify the desired CDUC(s).

TRANSMIT MUX0

Mux0 is a crossbar multiplexer that is located within the HSDOUT block and receives its input from the INVSINC block, which in turn receives its inputs from the CDUC blocks across MUX1. MUX1 can sum multiple CDUC outputs into a single I and/or Q stream. MUX0 then {receives the streams in the form of either a 2×4 input to output crossbar for the AD9084 (with two DACs and two Auxiliary DACs per A/B-side) or a 4×4 input to output crossbar for the AD9088 (with four DACs per A/B-side). Mux0 allows mapping the outputs from INVSINC to the digital inputs of the DAC cores. These multiplexer outputs are referred to as Logical DAC Inputs, marked with the suffix _x, as shown in Figure 107, to distinguish these inputs from the physical data inputs that enter the DAC core directly.

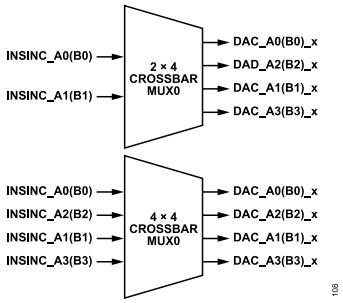

Figure 107. 2×4 and 4×4 Crossbar Mux0 for B-side and A-side

The API library fully supports configuration of Mux0. Mux0 is configured using the **adi_apollo_txmux_hsdout_set()** API function based on the settings in the device profile. The Mux0 specific parameters and the possible values set by this API function are listed in Table 89.

Table 89. Mux 0 Configuration parameters

Parameter	Description	Enumerations
device	Pointer to the APOLLO device data structure	
side_sel	Device side(s) to set HSDOUT	ADI_APOLLO_SIDE_NONE = 0x0, (No side) ADI_APOLLO_SIDE_A = 0x1, (A-side) ADI_APOLLO_SIDE_B = 0x2, (B-side) ADI_APOLLO_SIDE_ALL = 0x3, (All sides)
dac	Array indices are DAC index for A-side (0-3) and values are corresponding HSDOUT crossbar input channels (0-3) to be connected	
length	Length of the DAC array, max value ADI_APOLLO_NUM_DAC	

APOLLO MXFE TRANSMITTER

NCO PROGRAMMING AND DEBUG

During initial system bring-up, a typical debug step may include sending a CW (sine wave) signal to the DAC outputs from the NCO inside the FDUC or the CDUC, what is typically labeled NCO-only mode or DDS mode. This mode does not require an active JESD204B/C link, but an active link may be running in the background. See the section NCO-Only Mode for more details. The input to the DUC can be then selectively connected to either the internal dc-word generator (NCO-only mode) or to the data path upstream that would presumably receive samples from an active JESD204B/C link.

In addition to operating the device as a DDS, NCO-only mode also helps isolate any issues that appear related to the JESD204B/C link from issues that appear to be related to the RF front-end or the remaining signal-chain downstream from the DAC outputs.

The DAC output could be also looped to the ADC input, to help debug the ADC core (while the NCO-only mode in inside the CDDC and FDDC blocks can be used to confirm the Rx data path and the JESD204B/C link of the receiver).

Generating a CW within the FDUC tests a longer section of the data path including the CDUC, since the FDUC output can be routed to the input of the CDUC, through MUX2.

The API currently supports loading two methods for updating the NCO frequency:

- Direct manipulation of the NCO frequency. The process is similar to loading a single frequency profile using the FFH engine.
- Loading multiple frequency profiles that can be later hopped to using FFH.

In both cases the input can be the internal dc word generator or the baseband data sent across a JESD204B/C link.

A typical procedure for manipulating the NCO registers may be:

1. Configure the device into a known state by loading the correct device profile. The simplest approach would be to execute one of the examples, such as in .\apollo_api_dev2\examples\ads10_apollo\main.c, to a breakpoint located after the initial configuration was complete, generally after calls to set up the NCOs and to **adi_apollo_clk_mcs_oneshot_sync()** are executed (or adi_apollo_clk_mcs_trig_sync_enable() is set and a trigger sent for phase 2 alignment)
2. Enable the NCO (typically already enabled by the device profile)
3. Enable NCO hopping (typically already enabled by the device profile)
4. Load an array of frequency values (FTWs). Unused profiles can be assigned “0”.
5. Load an array of phase values (phase offset). Unused profiles can be assigned “0”.
6. Select an active NCO profile to hop to
7. (optional) add a settling delay (should settled in a few 100nS or less)
8. If in NCO-only mode, adjust the NCO amplitude using its dc word

Steps #2-5 may be performed of initial bring up, to be programmed using the device profile. Or the user may elect to modify these parameters after. An outline of the NCO methods that are currently supported is shown in Table 90.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 90. FNCO Configuration Methods

Method	Parameter	Description	Enumerations
adi_apollo_cnco_enable_set()adi_apollo_fnco_enable_set()	enable	Enables the NCO	0: disable1: enable
adi_apollo_cnco_hop_enable()adi_apollo_fnco_hop_enable()	Hop_en	Enables NCO hopping (FFH)	0: disable1: enable
adi_apollo_cnco_profile_load()adi_apollo_fnco_profile_load()		Method to load a frequency profile pairs, both frequency and phase. If *length*>1, then the words' array *words* will be loaded sequentially, starting with the profile index indicated in *first*. The profiles may be then manually hopped to using another method.	
	word_sel	Selects the word to update, either the phase increment (FTW) or offset	0: phase increment (FTW)1: phase offset
	first	Int, 0 to 15. The first profile to load	

APOLLO MXFE TRANSMITTER

Table 90. FNCO Configuration Methods (Continued)

Method	Parameter	Description	Enumerations
	words	Array, of length *length*, of phase increment or offset words (32-bit). Length must be equal to the input argument *length*.	
	length	Int, where [first + length must be < 16]. Number of profiles to load	
adi_apollo_cnco_active_profile_set()adi_apollo_fnco_active_profile_set()		Selects the active frequency profile	
	profile_num	Int. the index of the FFH frequency profile number to apply the new setting to.	
adi_apollo_cnco_ftw_set()adi_apollo_fnco_ftw_set()		Quick method to assigns a new FTW to a frequency profile entry and make that profile active. Note that this method currently does not update the NCO phase.	
	profile_num	Int. the index of the FFH frequency profile number to apply the new setting to.	
	active_en	Bool. Sets the frequency profile in profile_num as the active profile. NOTE: If in direct SPI/HSCI mode, output will be immediate. If the update is triggered, the change will be delayed until the next trigger edge received.	0: keeps the current profile as active1: set profile_num as the active profile
	ftw	Frequency tuning word value. This sets the phase accumulator phase counter, which defines the output frequency.	
adi_apollo_cnco_test_mode_val_set()adi_apollo_fnco_test_mode_val_set()		Sets the dc word level, and thus the NCO output swing (amplitude) in NCO-only mode.	
	Val	Int. 16-bit word to set the dc word level	
adi_apollo_hal_delay_us()	us	Float. Pauses program execution for [n] uS.	
adi_apollo_clk_mcs_oneshot_sync()		Requests Oneshot sync to synchronize all the clocks, either to an internal SYSREF or an external SYSREF. This also aligns the NCO clocks across the data paths.	
adi_apollo_cnco_chan_pgm()		General method to update CNCO parameters of specific CDUC channel	
	profile_num	Int. the index of the FFH frequency profile number to apply the new setting to.	
	config	Structure. NCO channel config, based on **adi_apollo_**coarse_nco_chan_pgm_t. The syntax is the same as the one used in the device profile file.	
adi_apollo_fnco_mode_set()	mode	NCO mixer mode	0: Variable IF Mode1: Zero IF Mode2: Fs/4 Hz IF Mode3: Test Mode
adi_apollo_fnco_mixer_set()	mixer	Real or Complex Mixer select	0: Real Mixing Mode1: Complex Mixing Mode

DAC OUTPUT

DAC Core Architecture

The 16-bit, 28 GSPS DAC core uses a quad-switch architecture to achieve 28 GSPS operation with reduced power consumption compared to a conventional switch architecture. Both novel static and dynamic on-chip calibration techniques are used to enhance its dc and ac linearity performance. All DACs are pre-calibrated and the calibration coefficients are loaded during device boot-up. Dynamic element shuffling of the DAC switches is available to further reduce any residual gain or timing mismatch errors within the switch arrays.

The DAC includes an input data shuffler to randomize data from the digital and improves two-tone Inter-Modulation Distortion (IMD). This shuffler must be enabled in the main digital data path through the parameter "shuffle_en" in data structure of "dac_config" in the device profile, and the DAC data must be enabled in order to pass uncorrupted data from the digital to the DAC.

Table 91. API Functions for DAC control

API Function Name	Applicable Parameter(s)	Description
adi_apollo_dac_scrambler_enable_set()	enable/disable	1 = Enable scrambling of data to the DAC.

APOLLO MXFE TRANSMITTER

Table 91. API Functions for DAC control (Continued)

API Function Name	Applicable Parameter(s)	Description
adi_apollo_dac_data_enable()	enable/disable	1 = Enable DAC data

The quad-switch architecture is shown in Figure 108 along with a weighted NMOS current source whose current is switched between I_{OUTP} and I_{OUTN}. Only one pair of switches is enabled for a half-clock cycle with each pair being clocked on alternative clock edges such that a different data sample is reconstructed on the positive and negative clock edges. This mode of operation is commonly referred to as double data rate (DDR) since the DAC's output clock update rate is doubled relative to the input clock frequency. To maintain a low image spurs, the DAC core includes an on-chip duty-cycle correction circuit to maintain near 180º offsets between the positive and negative clock edges.

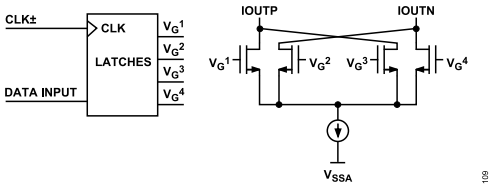

Figure 108. Quad-Switch Architecture

The 4 (or 8 based on the selected device model) DACs each provide complementary current outputs, DACxP and DACxN, where x = 0 thru 3 (or 7 based on the selected device model). Figure 109 shows a simplified equivalent model for the DAC core, referenced to an external -1 V regulated supply along with a pair of internal, 25 Ω pull-up termination resistors referenced to AVDD1P8_DAC provide a 50 Ω differential termination directly on the die. Note that the signal reference plane is referenced to this 1.8V supply domain hence this supply must be kept "clean" to avoid any common-mode noise from coupling onto the signal when the differential signal is translated to a single-ended signal referenced to AGND.

The NMOS DAC core output can be modeled as a pair of dc current sources that source half of the I_{OUTFS} current and a differential ac current source that has a peak level of $I_{OUTFS}/2$. Note for this NMOS DAC, I_{OUTFS} is fixed by design to a nominal setting of 20 mA . The value of this ac current source (I_{AC}) depends on the DAC code data, DAC_{CODE}, which represents the signal sample that is latched into the DAC core following each DAC clock cycle.

This relationship is shown in the following equation:

$$I_{AC} = (DAC_{CODE}-32768)/65535 \times I_{OUTFS} \quad (34)$$

where:
DAC_{CODE} = 0 to 65535 (decimal) and I_{OUTFS} =20mA.
I_{AC} assumes positive and negative values.

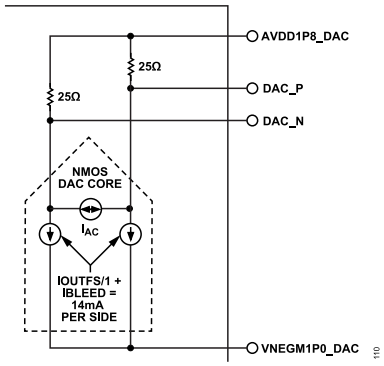

Figure 109. Simplified model of NMOS DAC

The optimal common mode voltage for the DAC output is 1.8V. Using the DC bias of a RF balun to provide the AVDD1P8_DAC supply domain for each leg is shown in Figure 110. Alternatively, a pair of RF choke connected to AVDD1P8_DAC along with DC blocking capacitors can be used as shown in Figure 111.

APOLLO MXFE TRANSMITTER

The signal *dependent* current, I_{AC}, with +/-10 mA full-scale swing generates a voltage across the DAC_P and DAC_N outputs. The resulting full-scale voltage swing depends on the external load appearing in parallel with the on-chip differential 50-ohm load. Maximum power transfer occurs when the external load is matched to the source impedance (or are complex conjugates of each other). In the simplest case using an ideal 50 Ω external load (and 1:1 ideal balun), the full-scale differential voltage swing is +/-500 mVp.p which corresponds to – 2 dBm for a reconstructed sine wave across the 50-ohm external load. In practice, the maximum signal power over frequency will be lower due to the balun's insertion loss, impedance mismatch loss, and the DAC's sinc roll-off.

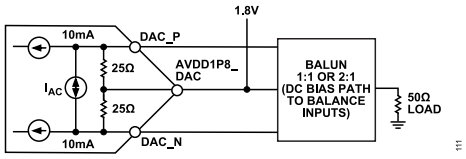

Figure 110. Equivalent DAC Output Circuit and Recommended DAC Output Network.

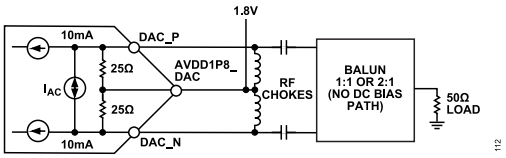

Figure 111. Equivalent DAC Output Circuit and Optional DAC Output Network with Balun w/out DC Bias.

DAC Output Impedance Characteristics

Figure 112 shows the typical differential S11 characteristics of the DAC outputs at the BGA Balls. Slight differences exist between the DAC outputs due to small variations in the trace length of the different channels in the AD908X package. These small differences have minimal impact on the DAC output frequency response. Nevertheless, ADI suggests simulating the DAC output response using the available s-parameter models together with an EM sim of the PCB layout and RF front-end components, to estimate the response of each DAC output channel in the target design.

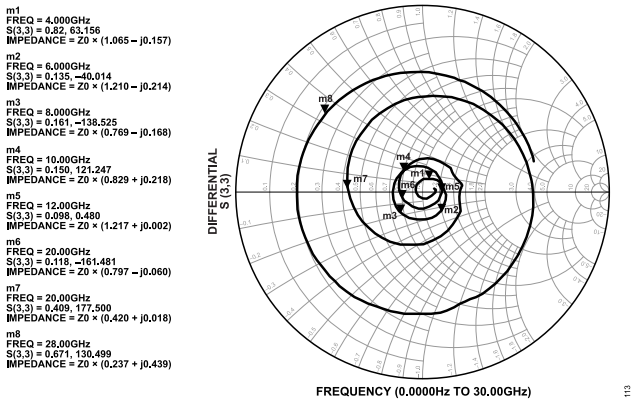

Figure 112. Smith Chart with Differential S11 of DAC0 Output Normalized to 50 Ω (Simulated)

Figure 113 shows the equivalent parallel resistance of the DAC, as derived from S11. The resistance does vary over frequency. When matching the DAC differential output to a single-ended 50 Ω load, use the following guidelines:

- Use a 1:1 or 2:1 balun depending on the DAC impedance over the frequency band of interest.
- Place the balun as close as possible to the DAC output pins and optimize the trace impedance to ensure lowest pass-band ripple and highest output bandwidth.
- Consider the amplitude and phase balance of the balun over the frequency region of interest, especially when a frequency region can be limited by an aliased even order harmonic such as the 2nd harmonic.
- For best RF performance, use an ac analysis model when optimizing the frequency response to an external component (such as a balun), along with an extracted PCB layout model. Keysight ADS models are available from ADI, which include an ac analysis model and the S-parameters of the DAC output.

APOLLO MXFE TRANSMITTER

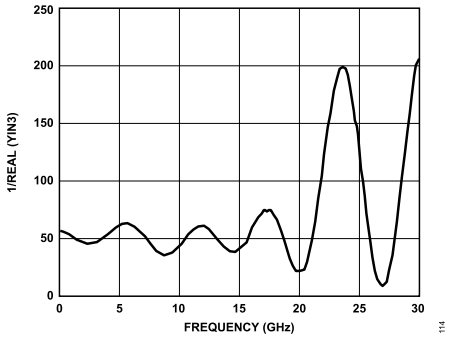

Figure 113. DAC0 Simulated Equivalent Parallel DAC Differential Output Resistance vs. Frequency

Because the output impedance of the DAC is complex, comprised of a resistive and a reactive component, the actual rms power delivered across a balun to a 50 Ω nominal load depends on the operating frequency. The rms power delivered to the load is influenced by the following factors:

- Proper PCB layout, balun selection, and balun placement on the PCB, which together impact the quality of the match between the 50 Ω load and DAC output impedance.
- The DAC's inherent sinc frequency response.
- The digital gain setting relative to the full-scale digital output of 0 dBFS. Note the optional INVSINC digital filter (prior to the DAC digital input) can be used to equalize the passband response albeit signal power at the lower frequency region will be reduced by 3 dB to account for the filter peaking in the high frequency region.
- The characteristics of the reconstructed waveform is defined by its crest factor or peak-to-average power ratio (PAPR). A signal with a high PAPR results in a lower rms power.

Figure 114 shows the simulated DAC output frequency response with an ideal 1:1 balun with sinx/x. Figure 115 is the frequency response with INVSINC enabled.

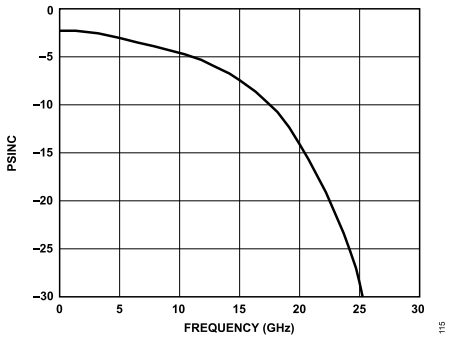

Figure 114. Simulated DAC Frequency Response with fDAC=28 GHz with sinx/x into an ideal 1:1 Balun

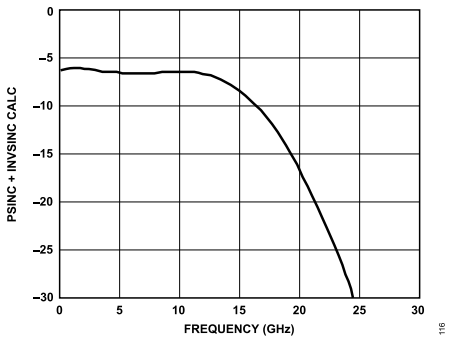

Figure 115. Simulated DAC Frequency Response with fDAC=28 GHz with sinx/x and INVSINC into an ideal 1:1 Balun

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

DYNAMIC RECONFIGURATION

The dynamic reconfiguration feature of the AD9084 and AD9088 refers to the devices ability to fast-switch the data path decimation/interpolation without disrupting the serdes link. This feature applies to the configuration of the following blocks and settings:

- FDDC decimation ratio /FDUC Interpolation ratio: Value selection via SPI/HSCI/GPIO
- CDDC decimation ratio /CDUC Interpolation ratio: Value selection via SPI/HSCI/GPIO
- FSRC decimation ratio /Interpolation ratio: FSRC Configuration via SPI/HSCI

The decimation/interpolation values of the blocks are programmed via API prior to a trigger, with the change taking effect after the trigger. Please reference below section, Hopping and Dynamic Reconfiguration Terminology for a list and description of common terms.

Dynamic Reconfiguration Methods and Procedure

To set up dynamic reconfiguration, the user must specify the trigger source. SPI/HSCI, external trigger pin, or internal trigger master. Each trigger is retimed by SYSREF. Then, specify how the next ratio will be set, by SPI/HSCI or GPIO. The procedure and options are shown below in Figure 116. Program the next ratio, and it will be reconfigured on the next trigger. Optionally, program one of the GPIO pins to show status of a reconfiguration done signal, more information in GPIO Quick Config. Note : SPI based reconfiguration trigger will yield non-deterministic timing and is only recommended for debug purposes.

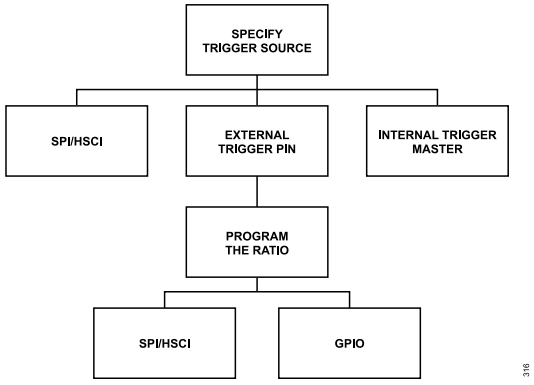

Figure 116. Dynamic Reconfiguration Trigger and Ratio Programming Options

CDDC/FDDC Dynamic Reconfiguration

The CDDC and FDDC values can be dynamically reconfigured to change the decimation ratio while maintaining the JESD link. The total decimation post reconfiguration must be greater than the decimation that the JESD link was initialized for to maintain the JESD lane rate. To maintain the correct lane rate, the Apollo MxFE rate match FIFO uses a feature called sample repeat. It will repeat valid samples to maintain the JESD lane rate which will then be removed once the data has reached the FPGA.

CDUC/FDUC Dynamic Reconfiguration

The CDUC and FDUC values can be dynamically reconfigured to change the interpolation ratio while maintaining the JESD link. To maintain the correct lane rate, the FPGA must repeat valid samples to be removed by Apollo's rate match FIFO.

FSRC Dynamic Reconfiguration

The FSRC block can be dynamically reconfigured to change the FSRC rate. During transmit, the FPGA will insert invalid samples (-FS) to match the JESD lane rate. These invalid samples are removed by the Apollo MxFE rate match FIFO so only valid data is present in the Apollo MxFE data path. On the receive side of Apollo, invalid samples are inserted by the rate match FIFO and removed by FPGA. The initial FSRC rate is programed from the values stored in the device profile. When both FSRC and DDC/DUC dynamic reconfiguration are desired, only **invalid_en** should be used. The FPGA or rate match FIFO will insert/remove invalid samples to accommodate the data rate change.

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

Dynamic Reconfiguration API

Table 92 outlines the API functions that support dynamic reconfiguration. Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder. API example code fullchip_sr_dr.c goes through a full example of dynamic reconfiguration and fullchip_fsrc_dr.c goes through a full example of dynamic reconfiguration with FSRC.

Table 92. API functions for dynamic reconfiguration

Function	Description
adi_apollo_trigts_reconfig_trig_sel_mux_set()	Sets the trigger selection MUX output for reconfiguration. Specify Tx or Rx and side of apollo to configure the trigger source for.
adi_apollo_trigts_cdrc_trig_sel_mux_set()	Selects the trigger source for CDDC or CDUC.
adi_apollo_trigts_fdrc_trig_sel_mux_set()	Selects the trigger source for FDDC or FDUC.
adi_apollo_reconfig_ctrl_pgm()	Program the dynamic reconfiguration block. The controller modes of coherence, trigger, reconfiguration interval.
adi_apollo_cddc_dcm_set()	Set the CDDC decimation that will be applied on the trigger for reconfiguration.
adi_apollo_fddc_dcm_set()	Set the FDDC decimation that will be applied on the trigger for reconfiguration.
adi_apollo_cduc_interp_set()	Set the CDUC interpolation that will be applied on the trigger for reconfiguration.
adi_apollo_fduc_interp_set()	Set the FDUC interpolation that will be applied on the trigger for reconfiguration.
adi_apollo_fsrc_ratio_set()	Set the FSRC ratio that will be applied on the trigger for reconfiguration.
adi_apollo_fsrc_mode_1x_enable_set()	Enable or disable 1x mode for FSRC. Disable if setting to a value >1.
adi_apollo_clk_mcs_trig_reset_dsp_enable()	After reconfiguration, resynchronizes Rx and Tx digital blocks when the trigger occurs.
adi_apollo_clk_mcs_man_reconfig_sync()	Manual reconfiguration sync. Clears RMFIFO status.

HOPPING

The hopping feature of the AD9084 and AD9088 refers to the devices ability to hop active profiles for various blocks in the datapath without disrupting the serdes link. The profile mentioned in this case differs from the device profile used during initialization. Hopping refers to changing the active profile of the following blocks and settings:

- PFILT: The Programmable Filter (PFILT) coefficients can be dynamically switched across four settings via SPI/HSCI/GPIO.
- CFIR: The Complex FIR(CFIR) coefficients can be dynamically switched across two settings via SPI/HSCI/GPIO.
- CNCO: The AD9084 and AD9088 CNCO supports frequency and/or phase hopping across 16 sets which can be initiated by SPI/HSCI/GPIO.
- FNCO: The AD9084 and AD9088 FNCO supports hopping of frequency, phase across 32 sets which can be initiated by SPI/HSCI/GPIO.
- BMEM: The AD9084 and AD9088 BMEM blocks supports hopping of programmable delay across 4 sets which can be initiated by SPI/HSCI/GPIO.

These hopping profiles store frequency tuning words, phase offsets, filter coefficients, and delay values. Please reference below section, Hopping and Dynamic Reconfiguration Terminology for a list and description of common terms.

Table 93. Dynamic Reconfiguration Support per Functional Block

Block Name	Profile hop supported	Number of DSP profiles	DSP profiles Selected Via	Synchronized Hop support	Profile Hop Initiated by
CNCO	Frequency, Phase Offset	16	GPIO / SPI / HSCI	Yes	GPIO direct SPI direct Internal Trigger External Trigger
FNCO	Frequency, Phase Offset	32	GPIO / SPI / HSCI	Yes	GPIO direct SPI direct Internal Trigger External Trigger
PFILT	Coefficient Profiles	4	GPIO / SPI / HSCI	Yes	GPIO direct SPI direct

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

Table 93. Dynamic Reconfiguration Support per Functional Block (Continued)

Block Name	Profile hop supported	Number of DSP profiles	DSP profiles Selected Via	Synchronized Hop support	Profile Hop Initiated by
					Internal Trigger External Trigger
CFIR	Coefficient Profiles	2	GPIO / SPI / HSCI	Yes	GPIO direct SPI direct
BMEM	Programmable Delay	4	GPIO / SPI / HSCI	Yes	GPIO Direct SPI Direct Internal Trigger External Trigger

Hopping Methods and Procedure

Hopping uses pre-loaded profiles to change the respective block active profile. First, the user would load the desired hopping profiles. The user can choose any combination of channels to hop together or individually. Then the hopping method must be specified. There are two methods of hopping, direct and scheduled. Direct hopping happens immediately, with no trigger required and asynchronous to SYSREF. Direct hopping can be done by SPI/HSCI or GPIO. Once direct hopping is scheduled, specify the profile to hop to and it will be hopped immediately.

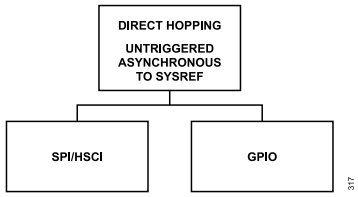

Figure 117. Direct Hopping Options

Scheduled hopping happens on a trigger and is synchronized to SYSREF. The user needs to specify how the next profile is selected and the trigger source. Specify next profile by SPI/HSCI or GPIO. Auto-select is another option to specify the next profile but it can only be used for the CNCO/FNCO hopping. The trigger source can be SPI/HSCI, external trigger pin, or internal trigger master. The sources are retimed to SYSREF so the hop is aligned throughout the chip. In a multi-apollo application, the external trigger pin can be used to achieve system hopping alignment when the devices are synchronized.

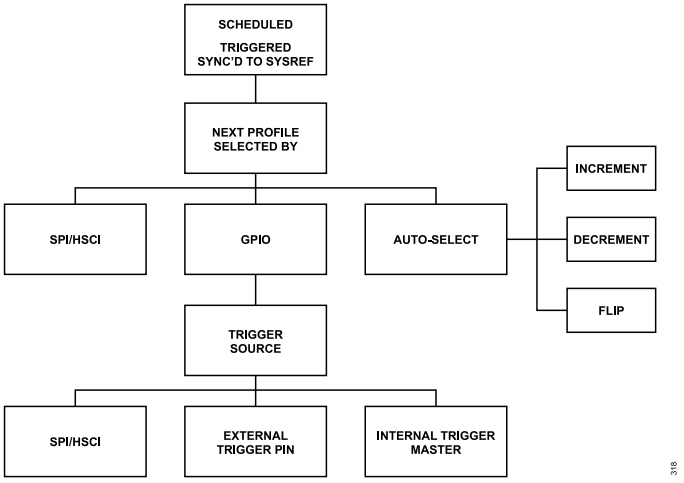

Figure 118. Scheduled Hopping Options

CNCO/FNCO Hopping

The CNCO block has 16 frequency tuning words and 16 phase offset profiles that can be set in the device profile with the fields nco_phase_inc_words and nco_phase_offset_words respectively or programmed with the API. The FNCO block has 32 frequency tuning words

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

and 32 phase offset words that can be set in the device profile or programmed with the API. Once loaded, both FNCO and CNCO profiles can be hopped with the method mentioned in the above section, Hopping Methods and Procedure. Auto mode is available to select profiles for the CNCO and FNCO. Auto mode has 3 subcategories: increment, decrement, and flip.

Auto Hopping

Auto hopping has three available modes, increment, decrement, and auto flip. In increment mode, the NCO will hop to the next profile, incrementing on each trigger. If the current profile is the last profile (15 or 31), it will wrap back around to profile 0. In decrement mode, the NCO will hop to the next profile, decrementing on each trigger. If the current profile is profile 0, it will wrap to the lastprofile (15 or 31). In auto flip mode, a high and low limit for the profile number must be specified. The NCO will hop to the next profile, incrementing on each trigger until the high limit is reached. It will then decrement back to the low limit, flip, and then start incrementing.

PFILT/CFIR Hopping

Each PFILT block has 4 coefficient banks. These banks must be programmed through SPI/HSCI prior to profile hopping. They can be loaded through the API or included in the device profile with the key coefficients. The CFIR block has 2 coefficient banks. These banks must be programmed through SPI/HSCI prior to profile hopping. They can be loaded through the API or included in the device profile with the keys coeffs_i and coeffs_q. Once loaded, the PFILT/CFIR profiles can be hopped with the methods mentioned in the above section, Hopping Methods and Procedure.

BMEM Hopping

Please refer to the hopping delay mode in the BMEM Delay Mode section for a summary of how to hop BMEM.

Hopping API

Table 94 outlines the APIs functions that support hopping. Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder. API example function fullchip_hop.c goes through a full example of hopping all blocks using the functions shown below.

Table 94. API Functions for Hopping

Function	Description
adi_ads10_ex_pfilt_coeff_file_load()	Reads PFILT coeffs from a file and loads into selected PFILT bank(s)
adi_apollo_pfilt_coeff_transfer()	Select a filter bank to transfer to active bank
adi_apollo_gpio_hop_pfilt_enable_set()	Enable hopping for selected PFILTs. Slice select must be set to SPI.
adi_apollo_pfilt_profile_sel_mode_set()	Set the PFILT profile selection and hopping mode
adi_apollo_trigts_pfilt_trig_sel_mux_set()	Select the trigger selection MUX output for PFILT. Utilize if doing scheduled trigger hopping.
adi_apollo_pfilt_next_hop_num_set()	Sets the next PFILT profile selection for the hop. Utilize if doing scheduled trigger hopping.
adi_apollo_trigts_pfilt_trig_mst_config()	Configures PFILT trigger master offset and period.
adi_apollo_cfir_coeff_pgm()	Load CFIR coefficients into profile.
adi_apollo_cfir_scalar_pgm()	Load CFIR complex scalar values into profile.
adi_apollo_cfir_gain_pgm()	Load CFIR gain adjustment values into profile.
adi_apollo_cfir_pgm()	Configures the CFIR parameters
adi_apollo_cfir_profile_sel()	Configures the CFIR for a specific profile
adi_apollo_gpio_hop_cfir_enable_set()	Enable hopping for select CFIRs. Slice select must be set to SPI.
adi_apollo_cfir_profile_sel_mode_set()	Set the CFIR profile selection and hopping mode.
adi_apollo_trigts_cfir_trig_sel_mux_set()	Select the trigger selection MUX output for CFIR. Utilize if doing scheduled trigger hopping.
adi_apollo_cfir_next_hop_num_set()	Set the next CFIR profile selection for the hop. Utilize if doing scheduled trigger hopping.
adi_apollo_trigts_cfir_trig_mst_config()	Configures CFIR trigger master offset and period.
adi_apollo_cnco_profile_load()	Loads coarse NCO phase increment or offset profiles.
adi_apollo_gpio_hop_cnco_enable_set()	Enable hopping for selected CNCOs. Slice select must be set to SPI.
adi_apollo_cnco_profile_sel_mode_set()	Sets the profile selection mode for the CNCO.
adi_apollo_cnco_active_profile_set()	Sets the active profile for CNCO when the profile select mode is direct register.

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

Table 94. API Functions for Hopping (Continued)

Function	Description
adi_apollo_cnco_hop_enable()	Enables CNCO hopping
adi_apollo_cnco_next_hop_num_set()	Sets the CNCO next hop number for trigger based hopping.
adi_apollo_trigts_cnco_trig_mst_config()	Configures the CNCO trigger master offset and period.
adi_apollo_fnco_hop_enable()	Enables FNCO hopping.
adi_apollo_fnco_profile_sel_mode_set()	Sets the FNCO profile selection mode.
adi_apollo_fnco_profile_load()	Loads fine NCO phase increment or offset profiles.
adi_apollo_gpio_hop_fnco_enable_set()	Enables hopping for the selected FNCOs. Slice select must be set to SPI.
adi_apollo_fnco_active_profile_set()	Sets the active profile for FNCO when the profile select mode is direct register.
adi_apollo_fnco_hop_pgm()	Enables FNCO hopping and configures the hop settings.
adi_apollo_fnco_next_hop_num_set()	Sets the FNCO next hop number for trigger based hopping.
adi_apollo_trigts_fnco_trig_mst_config()	Configures the FNCO trigger master offset and period.
adi_apollo_gpio_hop_block_select_set()	Set the block profile hopping as GPIO or SPI.
adi_apollo_gpio_hop_slice_select_set()	Set the slice profile hopping as GPIO or SPI. If SPI, overrides GPIO select for slice, side, and terminal.
adi_apollo_trigts_trig_now()	Creates a trigger pulse via SPI/HSCI
adi_apollo_trigts_ts_reset_mode_set()	Sets the timestamp counter reset mode via SPI or SYSREF.
adi_apollo_trigts_ts_reset()	Resets the timestamp counter.
adi_apollo_trigts_mst_mute_mask_set()	Selects which mute mask to use for muting after a specified count number.

Hopping and Dynamic Reconfiguration Terminology

- **Internal_sysref**: This is the global internal reference generated by sync logic and will be aligned to external SYSREF from the pin. The alignment occurs at a SYNC_EVENT mentioned below.
- **SYNC_EVENT (SE)**: The SYNC_EVENT refers to a full chip event that causes a re-alignment of the “internal_sysref” and all root clocks within AD9084/AD9088. This could be a change in the external SYSREF phase monitored by the sync logic or initiated by the customer FPGA/ASIC.
- **Oneshot-Sync**: This is the process for aligning internal clocking to an external SYSREF signal. The receiver and transmitter digital resynchronize their clocks based on this Oneshot-Sync.
- **External trig event (ETE)**: A trigger coming into any of the AD9084 or AD9088’s four trigger inputs can be used to qualify the next rising edge of internal_sysref for deterministically triggering certain events such as timestamp counter reset, NCO reset, data path reconfiguration, divided clocks etc. The specific edge of internal_sysref which has been qualified by an external trig input to act as a deterministic trigger is called as an External Trig Event. (ETE). The ETE could be a reconfiguration trigger event (ERTE) or a command trigger event (ECTE). While ERTE signals a reconfiguration of decimation/interpolation, ECTE could trigger NCO reset, frequency hops etc. Only ETE can be used to reset the timestamp counter.
- **Internal Trig Event (ITE)**: A trigger generated by on-chip trigger module(s), that run based on the timestamp counter, is called an Internal trigger event. This ITE can also be used for deterministically triggering events such as reconfiguration, NCO reset etc. The ITE could be a reconfiguration trigger event (IRTE) or a command trigger event (ICTE). While IRTE signals a reconfiguration of decimation/interpolation, ICTE could trigger NCO reset, frequency hops etc. ITE cannot be used to reset the timestamp counter. ITE can be used as reconfiguration trigger (IRTE) only if the spacing between IRTEs is a multiple of jtx_conv_clk or jrx_conv_clk period.
- **Timestamp (TS) counter**: Timestamp counter is a 64-bit incrementing counter running in digital which can be reset by an ETE. Timestamp counter would be used by a “Trigger module” block to generate periodic internal trigger events at programmable periodicity and offsets with respect to each other. Physically, there would be one timestamp counter each in receiver_A, transmitter_A, receiver_B and transmitter_B. The timestamp counters in receiver_A and receiver_B would function identically (should be reset by the same ETE). Similarly, the timestamp counters in transmitter_A and transmitter_B would function identically. In chip configuration cases where ADC and DAC operate in the same frequencies, all four timestamp counters would function identically.
- **NCO Phase coherence (NPC) counters**: The NPC counters would be 64-bit incrementing counters associated with each NCO, which can be reset by an ETE or ITE. The NCOs on the system would depend on this counter to maintain coherence across frequency hops.
- An example diagram showing the terms discussed above is given in Figure 119 :

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

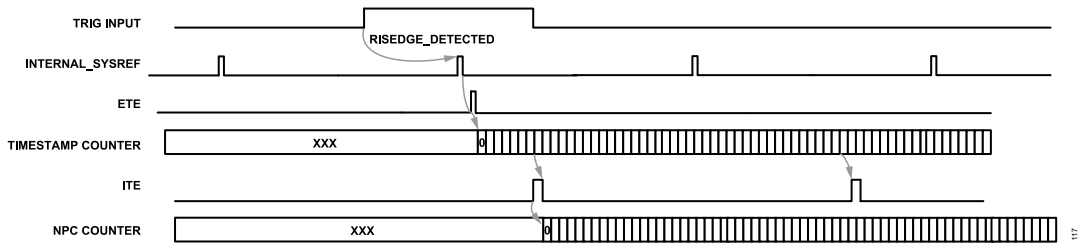

Figure 119. Example Timing Diagram of Dynamic Reconfiguration

- **System Tick (ST)**: This is the time resolution that the user can change the parameters of the system. This would be decided based on the system parameters. For example, if this is 1us then the user could change frequency/phase etc. on any 1us boundary. Reconfiguration of data path to different decimation ratios/interpolation ratios would also be needed at System Tick boundaries.

Reconfiguration can be initiated by an External Reconfiguration Trigger Event (ERTE) or an Internal Reconfiguration Trigger Event (IRTE). The SPI write (or GPIO based profile selection for FDDC/FDUC) regarding the configuration change (change of decimation/interpolation ratio) can happen any time before the corresponding Reconfiguration Trigger Event (RTE). The new configuration comes into effect only with the RTE. The JESD parameters do not change during a reconfiguration and the link doesn't go down. All measurements of determinism/coherence while in a given configuration are with respect to the time of the RTE which initiated that configuration, OR with respect to T0 (first configuration) which is selectable by the coherence mode setting in the **adi_apollo_ reconfig_ctrl_pgm_t** API structure.

For maintaining determinism/coherence of the NCO with respect to a given RTE, it is important to reset the NPC counter with this RTE.

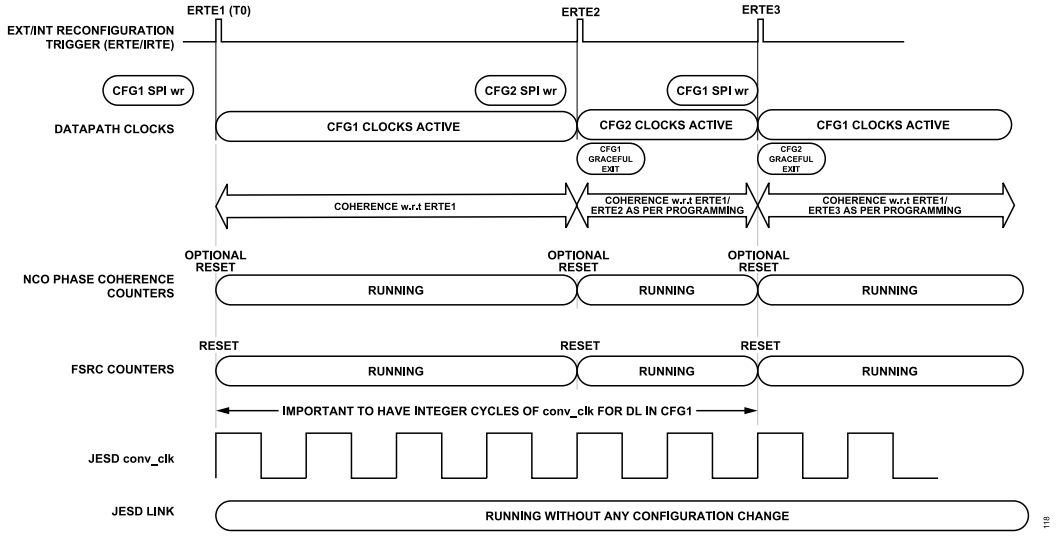

Figure 120. Example timing of dynamic reconfiguration between two configs (CFG1 -> CFG2 -> CFG1)

T0 Coherence with FSRC disabled: Implies that Coherence of NCOs and DL is with regards to the RTE which initiated the first configuration, as shown in Figure 121. T0 is defined as the reference time chosen (by SPI programming) to reset the clocking, timestamp counter and NCOs.

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

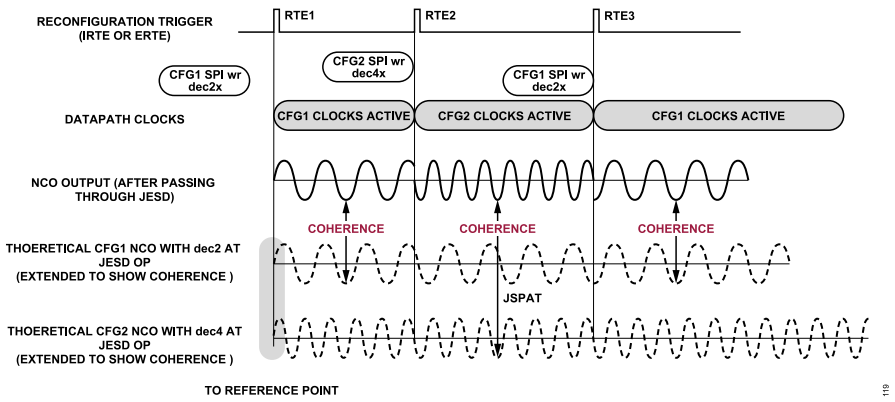

Figure 121. Example of dynamic reconfiguration T0 Coherence with FSRC disabled

T0 Coherence with FSRC enabled: If FSRC is enabled. T0 Coherence mode is not supported.

Tn Coherence: Coherence and Deterministic Latency for a given configuration are measured relative to the time of the reconfiguration trigger which initiated the configuration. Tn is defined as the reconfiguration trigger corresponding to a given nth configuration. The NCOs are reset with every reconfiguration trigger in this mode.

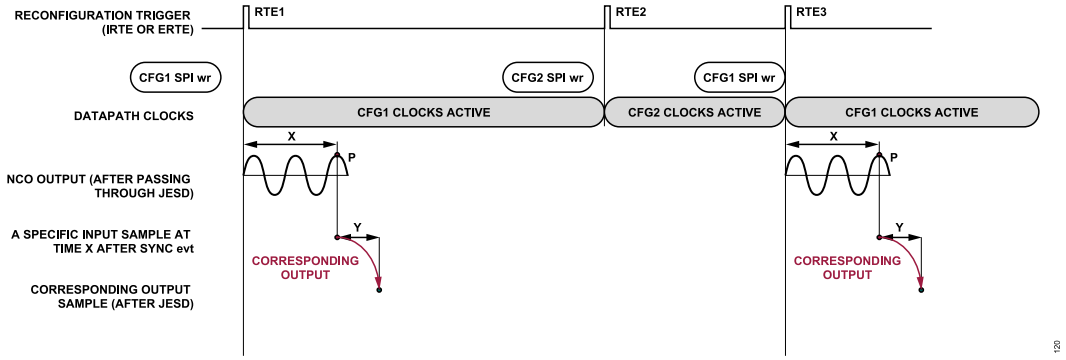

Figure 122. Example of dynamic reconfiguration Tn Coherence

SPECTRUM SNIFFER

The Spectrum Sniffer provides users information with fast indications of where the strong and weak signals are within the digitized signal bandwidth. This can be used for detecting strong signals and taking action for surveillance applications or for managing frequency hops to a clean spectrum for communications or radar systems.

The spectrum sniffer generates a 512-point FFT, where resolution bandwidth is a function of the ADC sample rate and can be calculated by Fs/512, where Fs is the sample frequency. Table 95 gives the resolution bandwidth (FFT Bin width) for 20 GHz and 14 GHz sample rates, for a 512-point FFT. The spectrum sniffer has a dynamic range of -1dBFS to -25dBFS for real data and -1dBFS to -31dBFS for complex data.

Table 95. Resolution Bandwidth (FFT Bin Width) for 512-point FFT

Sampling Frequency	Resolution Bandwidth(FFT Bin Width)
20 GHz	39.0625 MHz
14 GHz	27.34375 MHz

Sniffer Modes

There are four modes that the Spectrum Sniffer operates in:

- Normal Magnitude
- Instant Magnitude
- Normal IQ

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

- Instant IQ

In "Normal" operation, the sniffer stores new FFT bin data only when the input spectral content crosses the specified programmable threshold. In "Instant" operation, the sniffer automatically and unconditionally stores new FFT bin data, ignoring the thresholds. In "Magnitude" operation, the sniffer automatically calculates and stores the magnitude of the input signal. The sniffer can be programmed to store real magnitude data only, or the complex magnitude data. In "IQ" operation, the sniffer stores bin data as in-phase (I) and quadrature (Q). Sorting, exponential averaging, and continuous mode are not supported in IQ mode.

The four modes above are programmed in the API function **adi_apollo_sniffer_pgm()** and are set to “Normal Magnitude” by default.

Sniffer Input

There are two instances (one per side) of the Spectrum Sniffer on the device. One pair of complex inputs from two ADCs or single real data from one ADC per side (A or B) is supported for both 4T4R and 8T8R. The Receive MUX0 crossbar module can be used to select the inputs to the Spectrum Sniffer and thus any of the ADCs can be fed to the Spectrum Sniffer.

Each sniffer can select its data input source from one or two ADCs on its respective side through the API function **adi_apollo_sniffer_mux_set()**. In the real magnitude setting, one ADC must be selected as the real input. In the complex magnitude setting, one ADC is selected as the real input and another ADC is selected as the imaginary input. Likewise, in IQ mode, one ADC is selected as the in-phase input and another ADC is selected as the quadrature input.

Sniffer User Control Modes

During operation, the sniffer can be controlled by SPI interface (via API functions) or GPIO to request data from the ADC and start FFT computation. In SPI control mode, the sniffer data request sequence is automatically handled by the API functions. In GPIO control mode, a trigger sequence on GPIOs (fft_enable_0, fft_hold_0, fft_done_0) is issued by the user to request data. See the CMOS GPIOs (Functional Stage) section for more information on GPIO programming.

This control mode is programmed during sniffer initialization by the parameters of "fft_hold_sel" and "fft_enable_sel" in the **adi_apollo_sniffer_init_t** data structure.

Programmable Thresholds

The sniffer has two programmable thresholds, max_threshold and min_threshold, which is used in normal mode. The sniffer compares the magnitude of each bin data and checks if any data crosses above the max_threshold or below the min_threshold. If any bin data meets either condition, an interrupt is generated that triggers the sniffer to sort and store data. Additionally, each bin that meets either condition is marked by a value "1" in their corresponding bin location in the max_threshold_bin or min_threshold_bin data fields in the **adi_apollo_sniffer_t** data structure. In instant mode, these thresholds are ignored, and data is unconditionally stored.

Functionality and Data Processing Options

The sniffer has a variety of functionality and hardware-based data processing options that can be applied before data is stored. These options are configured by API function **adi_apollo_sniffer_pgm()** with the data structure **adi_apollo_sniffer_pgm_t**.

- **Sorting**: When sorting is enabled (sort_enable = 1), the magnitude if bin data is sorted from least to greatest before it is stored. When sorting is disabled, magnitude data is left unsorted and is ordered by its bin number. Sorting is enabled by default and is only available in magnitude mode.
- **Top/Bottom FFT Averaging**: Each sniffer instance has one top FFT engine and one bottom FFT engine that is half of the FFT point delay than the top engine. Both engines compute in parallel as shown in Figure 123. With this feature enabled, the outputs of the top and bottom FFT engine are averaged which assists in avoiding certain critical time domain events. Top/Bottom FFT averaging is enabled by default and can be disabled by setting bottom_fft_enable to be value "0".

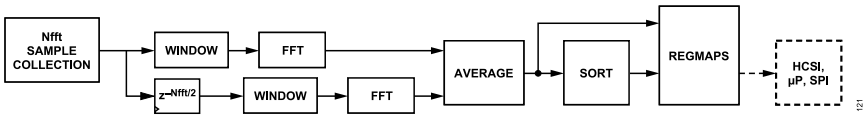

Figure 123. Averaging of top and bottom FFT

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

- **Continuous Mode**: In continuous mode, the sniffer continuously monitors the bin data before the users read the data, if any new data crosses the thresholds, it overwrites the previous data so that the latest data is stored and read by the users afterward. When the continuous mode is disabled, the sniffer works in single mode, the data is stored only once during one sniffer event. Hence, it may miss some critical data during this period in single mode. Continuous mode is disabled by default and can be enabled by continuous_mode = 1, only supported in magnitude mode and the exponential averaging is always disabled when continuous mode is enabled.
- **Exponential averaging**: the sniffer's FFT engine provides a feature to perform exponential averaging with the equation $y[n] = 2^{-\alpha}x[n] + (1 - 2^{-\alpha}y[n-1])$. This feature is enabled by setting the alpha_factor between 1 and 8 which directly controls the weight of the previous sample, and disabled by setting to 0. The average is reset to zero when the averaged data crosses the threshold and is stored, or when FFT computation is disabled. Exponential averaging is disabled by default and is only available in single mode and magnitude mode.
- **Dither**: Dither is added to input samples to reduce spurs across the spectrum. Dither is enabled by default and can be disabled by setting parameter dither_enable to be value "0".
- **Windowing**: A hard-coded frequency domain Hamming window can be applied to optimize the resolution bandwidth, especially beneficial to capture the narrow spectral peaks. However, in I/Q output mode, windowing may de-emphasize the data towards the edges of the 512-bin spectrum, and there is chance that the threshold may not be reached resulting in the desired data not being captured. Windowing is enabled by default and can be disabled by setting parameter window_enable to be value "0".
- **Low power mode**: In single mode, when exponential averaging is disabled, spectrum sniffer enters in low power mode by stopping FFT engine after a valid data is found. FFT Engine is started again when user requests a new data. This saves power while sorting is happening, or user is reading data. The expected power savings for this mode is 600mW. Low power mode is enabled by default and controlled by parameter low_power_enable.

Sniffer Data Output

The data structure **adi_apollo_sniffer_fft_data_t** contains the sniffer output bin data.

In magnitude mode, the array **mag_i_data** contains the magnitude of bin data and the array **bin_q_data** contains the bin number. The length of these arrays is 256 in real mode, and 512 in complex mode. Bin numbers (1 - 256) represent the real part of the spectrum, and bin numbers (257 - 512) represent the imaginary part of the spectrum. Thus, in real mode, only the lower 256 bins contain valid data. The bin number and its corresponding magnitude will always share the same array index. When sorting is disabled, the bin numbers are ordered numerically, and the magnitude data is ordered accordingly. When sorting is enabled, the magnitude data is sorted in ascending order and the bins are reordered accordingly. Bin numbers and magnitude data are sorted and still stored in their original frequency zones.

In I/Q mode, the array **mag_i_data** contains the in-phase data and **bin_q_data** contains the quadrature data. Both arrays contain magnitude data for each component across 512 bins. The bin number associated with the I or Q data is its array index + 1.

To convert magnitude data values to power, the following equation is used:

$$P_{dB} = 20\log_{10}(M)dB - 35dB \quad (35)$$

where M is the resulting magnitude data, and the 35dB term arises from the sniffer's 35dB dynamic range.

Spectrum Sniffer Latency (Preliminary)

Figure 124 illustrates the latency-inducing blocks of the sniffer function along with latency values for the case where the ADC sampling rate is 20Gsps.

- Each clock period is 1.6ns in 4T4R, 3.2ns in 8T8R.
- Sorting is at 1.6ns for both 4T4R and 8T8R.
- Setting fft_enable low and then high increases the required time by around 66 cycles to get first data.
- Disabling sorting saves around 516 cycles to get data.
- In real mode, sorting takes 260 cycles instead of 516 cycles. Only need to sort half of the bins

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

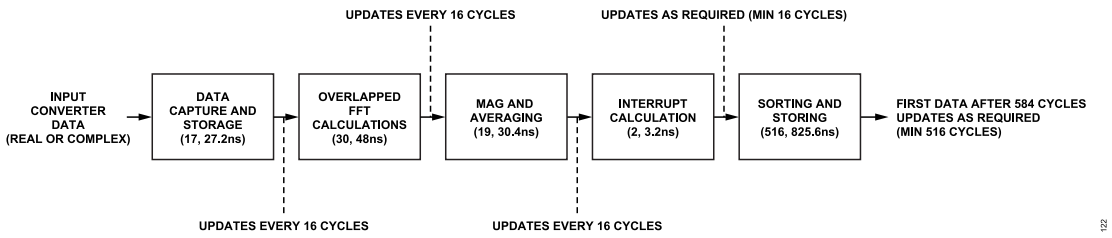

Figure 124. Spectrum Sniffer Latency at 20Gbps

Sniffer API Functions and Example

An example is provided in **examples\ads10_apollo_ex_main\rx_sniffer.c** which demonstrates the procedure of the sniffer configuration, data capture and data access. The high level sequence is shown as below,

1. Call API function **adi_apollo_sniffer_init()** to configure the control mode (SPI vs GPIO), real or complex mode, and the thresholds.
2. Run clock conditioning calibration and ADC foreground/background calibrations.
3. Call API function **adi_apollo_sniffer_pgm()** which calls **adi_apollo_sniffer_adc_mux_set()** to configure MUX0 to select the ADCs as sniffer inputs. Then configure the sniffer functionality as discussed in Functionality and Data Processing Options.
4. Until now, the sniffer hardware is configured. The next step is to toggle the FFT engine to capture the data which can be done by either calling API function **adi_apollo_sniffer_data_get()** (SPI control mode) or toggling GPIO. In this example, GPIO control mode is selected so it calls the GPIO functions to,
   - Set GPIO fft_enable to HIGH,
   - Set GPIO fft_hold to LOW,
   - Wait for GPIO fft_done to be HIGH,
   - Set GPIO fft_hold to HIGH.
5. Read back data by calling API function **adi_apollo_sniffer_fft_data_get()**.

The sniffer relevant API functions are listed in Table 96. Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 96. Spectrum Sniffer API Functions

Functions	Description
adi_apollo_sniffer_enable_set(adi_apollo_device_t *device, adi_apollo_blk_sel_t sniffers, uint8_t enable)	Enable sniffer function.
adi_apollo_sniffer_fft_enable_set(adi_apollo_device_t *device, adi_apollo_blk_sel_t sniffers, uint8_t enable)	Set sniffer fft_enable value in data request sequence. Can be mapped to GPIO.
adi_apollo_sniffer_fft_hold_set(adi_apollo_device_t *device, adi_apollo_blk_sel_t sniffers, uint8_t hold)	Set sniffer fft_hold value in data request sequence. Can be mapped to GPIO
adi_apollo_sniffer_init(adi_apollo_device_t *device, adi_apollo_blk_sel_t sniffers, adi_apollo_sniffer_init_t *config)	Configure the control mode (SPI vs GPIO), real or complex mode, and the thresholds.
adi_apollo_sniffer_pgm(adi_apollo_device_t *device, adi_apollo_blk_sel_t sniffers, adi_apollo_sniffer_pgm_t *config)	Program the sniffer functionality and data processing options.
adi_apollo_sniffer_fft_data_get(adi_apollo_device_t *device, adi_apollo_blk_sel_t sniffers, adi_apollo_sniffer_param_t *config, adi_apollo_sniffer_fft_data_t *fft_data_params)	Read back sniffer FFT data.
adi_apollo_sniffer_data_get(adi_apollo_device_t *device, adi_apollo_blk_sel_t sniffers, adi_apollo_sniffer_param_t *config, adi_apollo_sniffer_fft_data_t *fft_data_params)	Enable sniffer capture and get the data with SPI control mode.
adi_apollo_sniffer_adc_mux_set(adi_apollo_device_t *device, adi_apollo_blk_sel_t sniffers, uint16_t adc)	Set ADC mux0 for sniffer inputs.

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

Table 96. Spectrum Sniffer API Functions (Continued)

Functions	Description
adi_apollo_sniffer_fft_done_get(adi_apollo_device_t *device, adi_apollo_blk_sel_t sniffers, uint8_t *done)	Read back the FFT done status.

Table 97. Spectrum Sniffer API Parameters (in adi_apollo_sniffer_init_t structure)

Parameter name	Default	Description
sniffer_enable	0	Enable Spectrum Sniffer. 0: Spectrum Sniffer Disabled 1: Spectrum Sniffer Enabled
fft_enable_sel	1	Select between registers and GPIO for FFT_ENABLE Bit. 0: Select GPIO control 1: Select registers control
fft_hold_sel	1	Select between registers and GPIO for FFT_HOLD Bit. 0: Select GPIO control 1: Select registers control
real_mode	0	Select between real and complex FFT operation. 0: Select Complex FFT. Pair of converters used for I and Q sampling 1: Select Real FFT
max_threshold[7:0]	0xFF	Maximum threshold for comparison with magnitude Output from FFT Engine.
min_threshold[7:0]	00	Minimum threshold for comparison with magnitude Output from FFT Engine.

Table 98. Spectrum Sniffer API Parameters (in adi_apollo_sniffer_pgm_t structure)

Parameter name	Default	Description
bottom_fft_enable	1	0: Bottom FFT Disabled. 1: Bottom FFT Enabled
sort_enable	1	Enable Sorting of FFT Output Data in Magnitude Mode on Interrupt 0: Bypass Sorting and Store Directly to registers 1: Sorting Enabled
window_enable	1	0: Windowing Disabled. 1: Windowing Enabled
low_power_enable	1	0: Normal mode 1: Low power Enabled
dither_enable	1	0: Dithering Disabled. 1: Dithering Enabled
continuous_mode	0	0: Single mode 1: Continuous mode
sniffer_mode (of type adi_apollo_sniffer_mode_e)	0	Spectrum Output Mode 0: ADI_APOLLO_SNIFFER_NORMAL_MAGNITUDE 1: ADI_APOLLO_SNIFFER_INSTANT_MAGNITUDE 2: ADI_APOLLO_SNIFFER_NORMAL_IQ 3: ADI_APOLLO_SNIFFER_INSTANT_IQ
alpha_factor[3:0]	0	Alpha factor, Value of 0 Indicates exponential averaging is Disabled Values of 1-8 Indicates exponential averaging is enabled Values of 9-F are invalid

Table 99. Spectrum Sniffer API Parameters (in adi_apollo_sniffer_fft_data_t structure)

Parameter name	Description
mag_i_data [ADI_APOLLO_SNIFFER_FFT_LENGTH]	Contents of Magnitude / I data registers For magnitude mode, Sorted / Unsorted Magnitude Output For IQ mode, Unsorted I output

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

Table 99. Spectrum Sniffer API Parameters (in adi_apollo_sniffer_fft_data_t structure) (Continued)

Parameter name	Description
bin_q_data [ADI_APOLLO_SNIFFER_FFT_LENGTH]	Contents of Bin # / Q data registers For magnitude mode, Bin Number of Respective magnitude For IQ mode, Unsorted Q Output
bin_above_threshold[ADI_APOLLO_SNIFFER_FFT_LENGTH]	Contents of bin above threshold data registers (Magnitude mode only)
bin_below_threshold[ADI_APOLLO_SNIFFER_FFT_LENGTH]	Contents of bin below threshold data registers (Magnitude mode only)
max_threshold	Max threshold value Each bit Indicates if corresponding bin is above maximum threshold
min_threshold	Min threshold value Each bit Indicates if corresponding bin is below minimum threshold
valid_data_length	Length of valid data in arrays (512 complex, 256 real)
fft_is_complex	Whether or not the FFT is complex or real
data_mode (of type adi_apollo_sniffer_mode_e)	Sniffer mode during data capture \ref adi_apollo_sniffer_mode_e

Table 100. Sniffer GPIOs

GPIO name	Direction	Description
fft_enable	Input	Enable FFT Engine. This is Used Only If FFT_ENABLE_SEL Bit is Set Low. 0: FFT Engine is Disabled 1: FFT Engine is Enabled
fft_hold	Input	Hold Overwriting of Data to registers. This is Used Only If FFT_HOLD_SEL Bit is Set Low.. 0: Write New Data to registers After Reading Previous Data 1: Indicates to hold registers Data to Read It, This is set when FFT_DONE is 1
fft_done	Output	High Indicates Writing to registers is Complete.

FRACTIONAL SAMPLE RATE CONVERTER (FSRC) FOR TRANSMIT AND RECEIVE (AD9084 ONLY)

Fractional Sample Rate Conversion (FSRC) is a capability designed into the Apollo MxFE which allows a non-integer decimation ratio or interpolation factor to be enabled, programmed, and applied to the digital data stream.

A major benefit of FSRC is to configure the ADC/DAC sample rates (digital data rates) to non-integer multiples of the data rates, to support the different data rates of multiple standards (i.e. 4G, 5G, Wi-Fi, BT etc.). An additional benefit is that FSRC can be used to support integer decimation/interpolation factors when the desired integer value cannot be attained by the standard DDCs/DUCs.

FSRC supports operation on two streams of independent data. AD9084 has one FSRC per the A and B-side.

- Max data rate :
  - Max . data rate = 5GSPS
  - When using two streams of data Max data rate = 2.5GSPS
- FSRC is not supported in the AD9088 (8T8R)

The key design targets:

- Passband ripple: 0.05dB
- Stop band requirement: 90 dB
- SNR: 80dB
- SFDR: 95dB
- Rate change supported: 1.0 to 2.0 (both not inclusive)
- Pass BW = 80% BW (complex BW)
- Frac_delay_offset step size. 16-bit programmable fractional word.

FSRC provides any sampling ratio N_{FSRC}/M_{FSRC}, where N_{FSRC} and M_{FSRC} are integers and N_{FSRC}/M_{FSRC} must be between 1 and 2, i.e. $1 < N_{FSRC}/M_{FSRC} < 2$.

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

$f_{OUT\_FSRC} = f_{IN\_FSRC}/(N_{FSRC}/M_{FSRC}) = f_{IN\_FSRC} *(M_{FSRC}/N_{FSRC})$ for the ADC path

$f_{OUT\_FSRC} = f_{IN\_FSRC} * (N_{FSRC}/M_{FSRC})$ for the DAC path

FSRC is programmed in the API via **adi_apollo_rx_fsrc_configure()** and **adi_apollo_tx_fsrc_configure()**, both of which are sub functions of the top-level example bring up function **adi_ads10_apollo_ex_startup()**. The FSRC configuration functions take information from the rx_fsrc and tx_fsrc portion of the device profile. The FSRC value can also be updated dynamically using **apollo_rx_fsrc_configure()**. See the Dynamic Reconfiguration section for more details.

FSRC programming uses an expression of the form fsrc_rate_int + fsrc_rate_frac_a / fsrc_rate_frac_b, the components of which represent the integer and fractional portions of the expression used to calculate and program MFSRC/NFSRC, and its reciprocal NFSRC/MFSRC. Each of these parameters comes from the device profile. In a user application the values needed are calculated by the AD9084 Profile Generator, based on the N and M values given to the generator by the user.

Invalid Samples

When FSRC is enabled, fractional rates in the data stream might be present. The data rate being passed from each virtual converter to JESD cannot be an arbitrary fractional data rate because the dp_conv_clock is generated from fs/8 for 4T4R (FSRC is not supported for 8T8R).

In the receiver path, the Apollo MxFE inserts invalid samples into the data stream to support fractional sample rate conversion and user needs to remove the invalid samples at the logic device before further data processing by the logic device.

For the transmitter path, the user inserts the invalid samples at the logic device before transmitting to the Apollo MxFE and the invalid samples are removed at the Apollo MxFE before further data processing at the Apollo MxFE transmitter data path.

Definition of Invalid Sample

An invalid sample is defined to be a negative full scale (-FS) value sample. The valid and invalid samples are combined to ensure the data rate is converted from what is in the data path to the rate that the JESD is expecting. Before doing the invalid sample insertion, all actual -FS valued samples are adjusted to the value of (-FS+1).

Receiver path Invalid Sample Insertion/transmitter path Invalid Sample Removal is performed by the respective receiver/transmitter rate match FIFO.

Invalid Sample Insertion Guidelines

- Invalid data and valid data are clocked in the JESD link based on the JTx, JRx and DP “conv_clock”.
- The “conv_clock” processes samples in groups of NS samples at a time, at the “conv clock” rate.
  - See step 7 in the Guidelines for Setting the Internal SYSREF Period section for a definition for conv_clk
- NS is the Number-of-Samples parameter in one “conv_clk” period
- NS=1, 2, 4, 8 for Apollo
- The rate match FIFO expects samples in one “conv_clk” period (NS number of samples) to either be all-valid or all-invalid. If the rate match FIFO does not see that this is the case, it will insert invalid samples so that the samples in one “conv_clk” period are either all-valid or all-invalid.
- As such Invalids/Valids must be inserted by the FPGA (transmitter path) or RM FIFO (receiver path) in NS-sized chunks.

Figure 125 shows a conceptual example of how valid data (green) and invalid data (red) are inserted in NS sized groups. Valid and invalid data are combined so the data rate matches what is needed by the JESD204 configuration. This is managed by the rate match FIFO.

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

SCENARIOS																																	
NS = 4																																	
					LEMC/LMFC BOUNDARY																													
SAMPLES	–4	–3	–2	–1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
NS = 1	INVALID	VALID	VALID	INVALID	VALID	INVALID	VALID	VALID	INVALID	VALID	INVALID	INVALID	INVALID	VALID	VALID	INVALID	VALID	INVALID	VALID	INVALID	VALID	INVALID	VALID	INVALID	VALID	INVALID	VALID	INVALID	INVALID	VALID	INVALID	VALID	INVALID	VALID	INVALID	VALID
NS = 2	INVALID	INVALID	VALID	VALID	INVALID	INVALID	VALID	VALID	INVALID	INVALID	INVALID	INVALID	VALID	VALID	INVALID	INVALID	VALID	VALID	INVALID	INVALID	VALID	VALID	INVALID	INVALID	INVALID	INVALID	VALID	VALID	INVALID	INVALID	VALID	VALID	INVALID	INVALID	VALID	VALID
NS = 4	INVALID	INVALID	INVALID	INVALID	VALID	VALID	VALID	VALID	INVALID	INVALID	INVALID	INVALID	INVALID	INVALID	INVALID	INVALID	VALID	VALID	VALID	VALID	INVALID	INVALID	INVALID	INVALID	VALID	VALID	VALID	VALID	INVALID	INVALID	INVALID	INVALID	VALID	VALID	VALID	VALID
NS = 8	VALID	VALID	VALID	VALID	INVALID	INVALID	INVALID	INVALID	INVALID	INVALID	INVALID	INVALID	VALID	VALID	VALID	VALID	VALID	VALID	VALID	VALID	INVALID	INVALID	INVALID	INVALID	INVALID	INVALID	INVALID	INVALID	VALID	VALID	VALID	VALID	VALID	VALID	VALID	VALID
etc.																																				

125

Figure 125. Conceptual Example of Invalid Sample Insertion

As previously stated, with the FSRC block enabled, we could have fractional decimation values possible on the data stream . However, the jtx_conv_clk and dp_conv_clk need to be generated based on an fs/8 root clock. Hence, the actual data rate being passed on each virtual_converter input to JESD cannot be an arbitrary fractional data rate. This data, which potentially includes Invalid Samples, and associated parameters are programmed into a parameter called link_total_ratio. This value is pulled from the tx_link_cfg section of the device profile. The device profile values are set by the profile generator tool, based on the N, M, and coarse and fine decimation values given to the profile generator. link_total_ratio is programmed as part of the **adi_apollo_rx_dformat_ configure()** subfunction of **adi_ads10_apollo_ex_startup()**. The following shows how the values are calculated:

The actual decimated data rate available after the FSRC and up-sampler (note the up-sampler is disabled when FSRC is enabled since same data rate is expected in the data paths) is:

Fs/ (LINK_DDC_DEC*FSRC_DEC), where

- Fs is the undecimated core sample rate
- LINK_DDC_DEC is the decimation factor provided by the FDDC and CDDC
- FSRC_DEC is the fractional decimation factor (N_{FSRC}/M_{FSRC}, see FSRC section)

Definition: let link_dp_ratio = LINK_DDC_DEC*FSRC_DEC.

Note: link_dp_ratio is not a programmed field. It represents the actual total decimation rate for valid samples.

link_total_ratio: Indicates the final JESD data rate including invalid samples.

The link_total_ratio is programmed such that:

link_total_ratio*NS/8 = floor(link_dp_ratio*NS/8);

where NS is the JTX's NS (NUM_SAMPLES, referring to no. of parallel samples taken in by JTX per channel) parameter corresponding to each mode (documented in mode table).

NS can take values 1,2,4,8.

For cases where link_total_ratio is not equal to link_dp_ratio, as illustrated in Figure 126, invalid sample insertion is required. To perform this, the parameter INVALID_EN in the rx_dformat section of the device profile must be enabled, which indicates to the rate match FIFO that invalid data insertion needs to be done in order to create a non-fractional data rate compatible with the JESD204 requirements. This parameter is set or cleared by the profile generator. Note that INVALID_EN=1 can be used for all FSRC cases even if invalid samples are not needed for rate matching. The rate match FIFO will not add invalid samples if invalid samples are not needed. INVALID_EN=1 will still instruct the rate match FIFO to convert actual -FS values to -FS+1. A post processing algorithm for removing invalid samples (-FS) will still be valid even if there aren't any invalids inserted. In cases where link_total_ratio is equal to link_dp_ratio, invalid sample insertion is not needed.

Figure 126 shows the location of rate match FIFO relative to the FSRC and Sample-Repeat up-sampler.

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

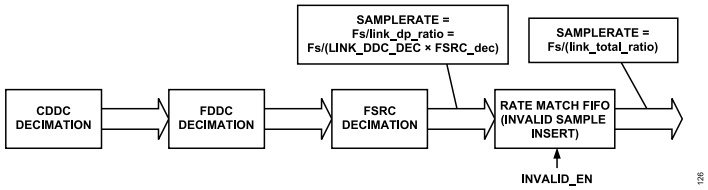

Figure 126. Invalid Samples Insertion

These functions are performed at the receiver rate match FIFO block:

1. Invalid samples are inserted for rate-matching on receiver side. An invalid sample is a negative full-scale (-FS) value sample.
   - Before doing the invalid sample insertion, all actual -FS valued samples would be saturated to (-FS+1) in the receiver.
   - On inserting the invalid samples, the data rate is converted from the actual data path data rate to the rate at which JESD data would be transmitted at the receiver.
2. The read and write pointers are incremented by the number of active samples required to be consistent with the link_total_ratio programmed value.
   - A valid data counter keeps track of number of valid data present in the FIFO locations. Every write clock edge i.e. the clock generated out of the gated Fs/8 clock will increment the valid count in terms of the number of active samples defined whereas the read clock based on the ungated fs/8 will decrement the valid counter by the number of active samples.
   - If the number of valid samples is less than the number of active samples, as many invalid samples as the number of active samples are sent out. Example: if the Rate match FIFO had 2 valid samples and the number of active samples is 4, the rate match block will send out 4 invalid samples on the lower parallel paths out of the 8 parallel paths until the next write clock which increments the number of valid samples.
3. When invalid samples are being inserted, the read pointer and valid count hold their value.
   - Since the FSRC can be programmed to 2.0 (not inclusive), there can be a maximum only two edges of write clock of rate match FIFO before the next edge of read clock is possible

An FSRC example using the rate match FIFO is illustrated in Figure 127. The dp_clk is generated based on the gated Fs/8 clocks (gated based on FSRC decimation). The dp_conv_clk is generated based on an ungated Fs/8 clock. Since the frequencies between the two are not matched, their alignments would bring in a worst-case path of Fs/8 (2.5GHz for the receiver) between them.

The functionality of the block is explained with the help of the timing diagram in Figure 127.

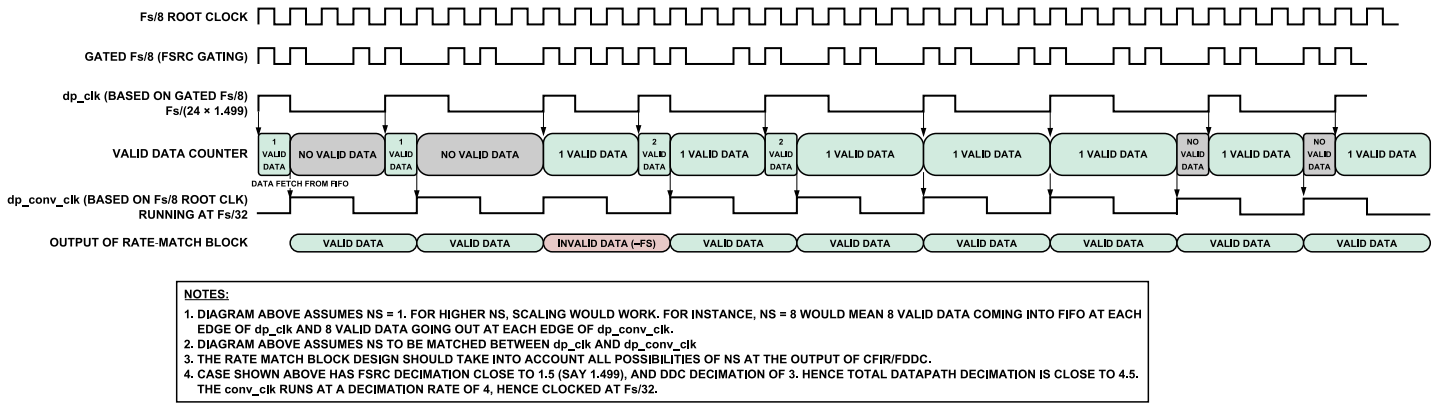

Figure 127. FSRC and Rate Match FIFO Example

FSRC in 1x Mode

For applications that do not need fractional decimation or interpolation but need to add a fractional delay, or cases where the sample rate is changed between a fractional rate and N_{FSRC}/M_{FSRC} = 1, FSRC 1x mode must be used for the N_{FSRC}/M_{FSRC} = 1 cases. 1x Mode is available in both receiver and transmitter.

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

1x mode in FSRC has the capability to provide N_{FSRC}/M_{FSRC} of exactly 1.0 for applications which use FSRC ratios ($N_{FSRC}/M_{FSRC} \neq 1$) at times and FSRC ratio (N_{FSRC}/M_{FSRC}) =1 at other times. For cases needing only $N_{FSRC}/M_{FSRC} = 1$ and fractional delay is not needed, FSRC can be disabled.

FSRC 1x Mode API

The API parameter fsrc_1x_mode puts the FSRC into 1x mode using **adi_apollo_fsrc_mode_1x_enable_set()**

See Table 102 for information on this.

FSRC Fractional Delay

The FSRC has the capability of adding a fractional delay to the digital data stream. The FSRC output will be delayed by a fraction of a pre-interpolation or post-decimation sample period.

FSRC Fractional Delay API

Sample_frac_delay is a member of the **adi_apollo_fsrc_pgm_t()** structure that can be used to produce a delay of a fraction of one sample period of data at the FSRC input (transmitter path) or FSRC output (receiver path). This value is programmed **by adi_apollo_fsrc_pgm()**.

See Table 102 for information on this.

Receiver Rate Match FIFO API

The rate match FIFO on the receiver side is programmed with the following API functions:

- adi_apollo_fsrc_rx_rm_fifo_pgm()
- adi_apollo_rx_rm_fifo_pgm_t()

See Table 103 and Table 104 for information on these functions.

FSRC API

The AD9084 API contains several FSRC examples in the ads10_apollo_ex_main example folder, including:

- ads10_apollo_ex_main/fullchip_fsrc_dr.c,
- ads10_apollo_ex_main/fullchip_ fsrc_sc1_ext_trig.c
- ads10_apollo_ex_main/ tx_jesd_sr_dr.c and more.

Examples of profiles with FSRC settings include profile id00_uc08a.h, id00_uc06_F.h, and more. In all FSRC enabled profiles the below settings are preset in the device profile by the profile generator and programmed during startup:

- N_{FSRC}/M_{FSRC} programming parameters fsrc_rate_int, fsrc_rate_frac_a, fsrc_rate_frac_b, and fsrc_delay
- The rate match FIFO rm_fifo member of rx_dformat has been set up with invalid_en = true (Enables incrementing of actual negative full-scale data to be (-FS + 1)
- The sample_repeat_en member of rm_fifo is set to false

Full FSRC configuration is done by **adi_apollo_tx_fsrc_configure()**, which takes parameters from the deivce profile and is loaded during startup. The fullchip_fsrc_dr.c example uses the locally defined **apollo_tx_fsrc_configure()** function to illustrate updating the FSRC ratio, 1x mode, and any interpolation changes required for dynamic reconfiguration of the FSRC block.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

See Table 101, Table 102, Table 103, Table 104, Table 105, and Table 106 for information on these functions and associated parameters.

Table 101. adi_apollo_fsrc_pgm - Configure FSRC parameters

Parameter	Description	Enumerations or Settings	Comment
fsrcs	Select FSRC	adi_apollo_fsrc_sel_e	

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

Table 101. adi_apollo_fsrc_pgm - Configure FSRC parameters (Continued)

Parameter	Description	Enumerations or Settings	Comment
		ADI_APOLLO_FSRC_NONE	No FSRC
		ADI_APOLLO_FSRC_A0	FSRC 0 of SIDE A
		ADI_APOLLO_FSRC_A1	FSRC 1 of SIDE A
		ADI_APOLLO_FSRC_B0	FSRC 0 of SIDE B
		ADI_APOLLO_FSRC_B1	FSRC 1 of SIDE B
		ADI_APOLLO_FSRC_ALL	All FSRC
config	Programming parameters	adi_apollo_fsrc_pgm_t	See Table 102

Table 102. adi_apollo_fsrc_pgm_t - FSRC programming parameters

Parameter	Description and Enumerations or Settings
fsrc_rate_int	A variable of FSRC rate = (FSRC rate_int + rate_frac_a/rate_frac_b)/2^48
fsrc_rate_frac_a	A variable of FSRC rate = (FSRC rate_int + rate_frac_a/rate_frac_b)/2^48
fsrc_rate_frac_b	A variable of FSRC rate = (FSRC rate_int + rate_frac_a/rate_frac_b)/2^48
gain_reduction	Gain Reduction for Decimation. This is not a user setting; it is a property of the FSRC decimation implementation. gain_reduction = round(2^12 * M_{FSRC}/N_{FSRC})
sample_frac_delay	Used to delay FSRC output sample by a fraction of a pre-interpolation or post-decimation sample period. For interpolation: Range=FSRC input sample period (1/f_pre_int) f_pre_int = fDAC_CLK/(CDUC_Factor*FDUC_Factor*FSRC_ratio) For decimation: Range=FSRC output sample period (1/f_post_dec). f_post_dec = fADC_CLK/(CDDC_Factor*FDDC_Factor*FSRC_ratio) The sample_frac_delay bitfield is 16 bits. . step size = Range/2^16 delay = step size*sample_frac_delay
ptr_syncrstval	This is used for the FIFO pointer value when the internal FIFO-sync is generated
ptr_overwrite	FSRC programming parameters 0: FIFO reset pointer value computed in design ptr_overwrite = 0 is the only valid user setting
fsrc_data_mult_dither_en	If this bit is 1, dither is added into FSRC in the data and delta multiplication calculation
fsrc_dither_en	If this bit is 1, dither is added into FSRC in the delta delay calculation
fsrc_4t4r_split	If this bit is set, the second instance of an FSRC on A-side (FSRC_A1 and/or FSRC_B1) is enabled. This bit is only to be used for 4T4R two stream use case When fsrc_4t4r_split =1 each FSRC per side is split into two FSRCs
fsrc_bypass	If this bit is 1, the FSRC is bypassed
fsrc_en	If this bit is 1, FSRC is enabled
fsrc_1x_mode	If this bit is 1, FSRC will be in 1x mode

Table 103. adi_apollo_fsrc_rx_rm_fifo_pgm - Configure Receiver Rate Match FIFO parameters

Parameter	Description	Enumerations or Settings	Comment
links	Target link select	adi_apollo_jesd_link_select_e	
		ADI_APOLLO_LINK_NONE	No Link
		ADI_APOLLO_LINK_A0	A-side Link 0
		ADI_APOLLO_LINK_A1	A-side Link 1
		ADI_APOLLO_LINK_B0	B-side Link 0
		ADI_APOLLO_LINK_B1	B-side Link 1
		ADI_APOLLO_LINK_ALL	All Links
config		adi_apollo_rx_rm_fifo_pgm_t	See Table 104

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

Table 104. adi_apollo_rx_rm_fifo_pgm_t - Receiver Rate Match FIFO programming parameters

Parameter	Description	Enumerations or Settings
dcm_ratio		Chip Decimation Ratio corresponding to link. The least overall decimation among DDCs present in link
total_dcm		Link Total Decimation
invalid_en		When link_total_ratio is not equal to Chip Decimation , Invalid_en should be 1 to indicate that invalid data can be inserted by the rate match FIFO.
dfor_ddc_dither_en		Dformat dither enable for DDC mode for link. 0: dformat dither disable, 1: dformat dither enable
sample_repeat_en		Sample Repeat Enable. 1:Enable, 0:Disable
startup_force_inv_en		If this bit is high and invalid_en bit is also high for that link, then after the sync (clkgen_sync) and until the reconfig_done signal (inside reconfiguration control), invalid samples will be transmitted

Table 105. adi_apollo_rx_fsrc_configure Configures Receiver FSRC (similar to adi_apollo_tx_fsrc_configure - see device profile)

Parameter	Description	Enumerations or Settings	Comment
side	Select device side, A or B	adi_apollo_sides_e	
		ADI_APOLLO_SIDE_IDX_A	Select side A
		ADI_APOLLO_SIDE_IDX_B	Select side B
		ADI_APOLLO_NUM_SIDES	Indicates how many sides are selected
idx	Select FSRC	adi_apollo_fsrc_idx_e	When FSRC_4t4r_split =1 there are two FSRC blocks per device side. After the device side is determined, this determines how many FSRC blocks are enabled per side.
		ADI_APOLLO_FSRC_IDX_0,	1st FSRC on the selected side
		ADI_APOLLO_FSRC_IDX_1,	2nd FSRC on the selected side
		ADI_APOLLO_FSRCS_PER_SIDE	Indicates how many FSRCs are selected per side
config	Configuration parameters.	adi_apollo_fsrc_cfg_t	See Table 106
jrx_link_config	JRx link configuration parameters	**adi_apollo_jesd_rx_link_cfg_t**	Additional parameter for adi_apollo_tx_fsrc_configure. See structure definition in the device profile.

Table 106. adi_apollo_fsrc_cfg_t - Fractional sampling rate converter configuration

Parameter	Description	Enumerations or Settings	Comment
fsrc_rate_int	Rate, integer part (48 bits)	Set from Device Profile	A variable of FSRC rate = (FSRC rate_int + rate_frac_a/ rate_frac_b)/2^48
fsrc_rate_frac_a	Rate, numerator of fractional part (48 bits)	Set from Device Profile	A variable of FSRC rate = (FSRC rate_int + rate_frac_a/ rate_frac_b)/2^48
fsrc_rate_frac_b	Rate, denominator of fractional part (48 bits)	Set from Device Profile	A variable of FSRC rate = (FSRC rate_int + rate_frac_a/ rate_frac_b)/2^48
fsrc_delay	Sample fractional delay	Set from Device Profile	This is used to delay the output sample by this fractional delay
enable	FSRC enable	Set from Device Profile	If this bit is 1, FSRC is enabled
gain_reduction	Gain Reduction for Decimation	Set from Device Profile	12 bit value
ptr_syncrstval	Pointer Sync reset Value	Set from Device Profile	6 bit value
ptr_overwrite	Pointer overwrite	Set from Device Profile	
data_mult_dither_en	Enable Dither Addition for FSRC in Data Mult	Set from Device Profile	
dither_en	Enable Dither Generation and Addition	Set from Device Profile	
split_4t4r		Set from Device Profile	
mode_1x		Set from Device Profile	

TRANSMIT/RECEIVE PROGRAMMABLE FILTER (PFILT)

The transmitter/receiver programmable PFILT is an optional signal processing block that enables the user to provide customized FIR digital filtering directly to the wideband signal content represented at the ADC output(s)/DAC input(s). For the receive path (Rx), each PFILT takes two outputs from a pair of ADCs through MUX0 and provides the outputs to CDDCs through MUX1. For the transmit path (Tx), each PFILT takes two outputs from CDUC through MUX1 and provides the outputs to DACs through MUX0. Two PFILT inputs could represent two independent real channels or a pair of I and Q channels. This block avoids the need to perform the same function in an ASIC or FPGA, which allows users to

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

take full advantage of the digital filtering capability in the data path. The net result of using this filter is that considerable power and cost savings (reduced MOPS and FPGA processing overhead) can be realized on an ASIC or FPGA.

Supported PFILT Modes

Both the AD9084 and AD9088 support the PFILT block with the same number of coefficients (maximum 32 taps). The filter mode supported by the PFILT is listed in Table 107.

Table 107. Programmable FIR Filter Mode

Filter Mode	Explanation
16-Tap Real Filter Mode	Two 16-tap real sub-filters, one for each ADC/DAC (See Figure 128)
32-Tap Real Filter Mode	One 32-tap real filter, only on one ADC/DAC (See Figure 129)
8-Tap Real Filter Mode	Two 8-tap real sub-filters, one for each ADC/DAC (See Figure 130)
Half-Complex Mode	Two real sub-filters, 16-taps for cross term and direct term (See Figure 131)
Matrix Mode	Four real sub-filters, 8 taps for each of A, B, C, D (See Figure 132)

Each mode can be mathematically described by the following equations, where DIN1 and DIN2 denote two inputs, representing either two independent real channels or a pair of I and Q channels, DOUT1 and DOUT2 denote the corresponding outputs, and the asterisk symbol (*) denotes convolution.

Real 16-Tap filter for each channel (see Figure 128):

DOUT1[n] = DIN1[n] * A[n]

DOUT2[n] = DIN2[n] * D[n]

Real 32-Tap filter for either channel (see Figure 129):

DOUT1[n] = DIN1[n] * A[n]

or

DOUT2[n] = DIN2[n] * D[n]

Real 8-Tap filter for each channel (See Figure 130):

DOUT1[n] = DIN1[n] * A[n]

DOUT2[n] = DIN2[n] * D[n]

Half-Complex filter using two real 16-Tap sub-filters for the channels (see Figure 131):

DOUT1[n] = DIN1[n-p]

DOUT2[n] = DIN1[n] * B[n] + DIN2[n] * D[n]

or

DOUT1[n] = DIN1[n] * A[n] + DIN2[n] * C[n]

DOUT2[n] = DIN2[n-p]

Matrix filter using four real 8-Tap sub-filters for the channels (see Figure 132).

DOUT1[n] = DIN1[n] * A[n] + DIN2[n] * C[n]

DOUT2[n] = DIN1[n] * B[n] + DIN2[n] * D[n]

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

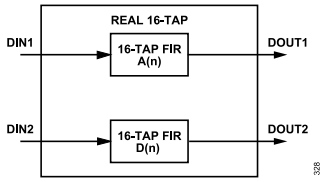

Figure 128. Real 16-Tap PFILT Mode

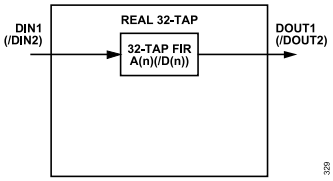

Figure 129. Real 32-Tap PFILT Mode

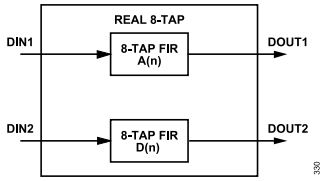

Figure 130. Real 8-Tap PFILT Mode

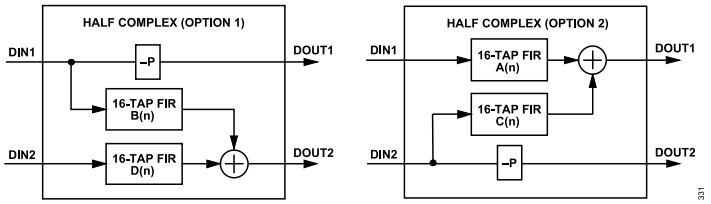

Figure 131. Half Complex with N/2-Taps PFILT Mode

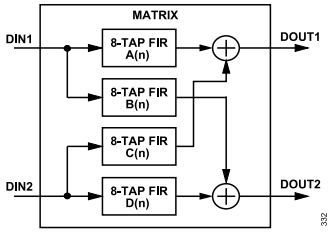

Figure 132. Full Matrix with N/4-Taps PFILT Mode

PFILT Data Path Connectivity

Figure 133 and Figure 134 illustrate the PFILT connectivity to the adjacent blocks in the Rx data path and Tx data path in AD9084, respectively.

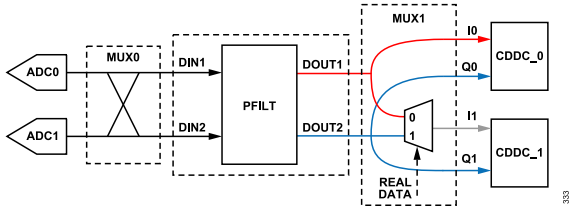

Figure 133. PFILT Rx Data Path Connectivity for AD9084

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

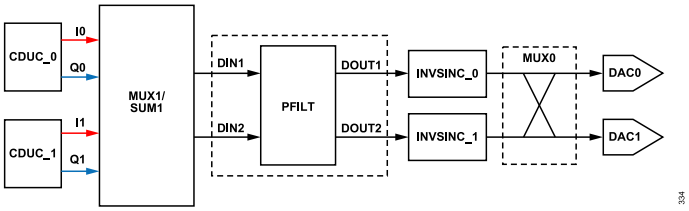

Figure 134. PFILT Tx Data Path Connectivity for AD9084

In the Rx data path as shown in Figure 133, MUX0 can be configured to multiplex the two ADC outputs in any combinations to generate DIN1 and DIN2 (i.e., DIN1=ADC0, DIN2=ADC1 or DIN1=ADC1, DIN2=ADC0 or DIN1=ADC0, DIN2=ADC0 or DIN1=ADC1, DIN2=ADC1). MUX1 takes DOUT1 and DOUT2 to generate two pairs of complex signals including both I and Q components, one for each CDDC. For CDDC_0, it takes I0 and Q0, where I0 = DOUT1 and Q0 =DOUT2. For CDDC_1, it takes I1 and Q1, where Q1 =DOUT2 and I1 could be either DOUT1 or DOUT2. When the PFILT has real input data (i.e., DIN1 and DIN2 are real), I1 =DOUT2, otherwise, I1 =DOUT1. Note CDDC has a Real Mixing and Complex Mixing mode. For real data, the Real Mixing is used, in which the Q input to the CDDC is ignored. For complex data, the Complex Mixing mode is used where both I and Q components are used.

In the Tx data path as shown in Figure 134, MUX1/SUM1 can be configured to take two pairs of complex output signals from CDUC_0 and CDUC_1 to generate two real input signals DIN1 and DIN2 for the PFILT. Four different modes can be configured: DIN1 = I0, DIN2 = I1 or DIN1 = I0 + I1, DIN2 = Q0 + Q1 or DIN1 = I0, DIN2 = Q0 or DIN1 = I1, DIN2 = Q1. After filtering, DOUT1 becomes the input of INVSINC_0 and DOUT2 becomes the input of INVSINC_1. MUX0 in the Tx data path operates similarly as the MUX0 in the Rx data path to generate input signals for DAC0 and DAC1. Note user can configure MUX1/SUM1 properly based on their application to provide a pair of real inputs or a pair of complex inputs to PFILT. Different from the Rx data path, Tx PFILT only generates one pair of output signals.

For AD9088, each east/west side has two PFILTs. The data path connectivity is similar to AD9084. The PFILT Rx Data Path Connectivity with real input data and complex input data are shown in Figure 135 and Figure 136 respectively. The PFILT Tx Data Path Connectivity with real input data and complex input data are shown in Figure 137 and Figure 138 respectively. Note Mux0 is 4x4 for both Tx and Rx. Each one of the four Mux0 output can take any one of the four inputs. Similarly, the for real data, the Q input to each CDDC is ignored with Real Mixing mode.

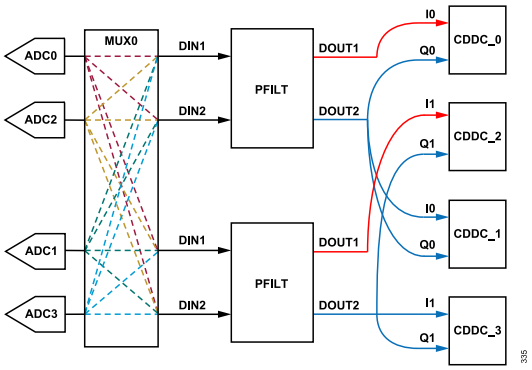

Figure 135. PFILT Rx Data Path Connectivity with Real Data for AD9088

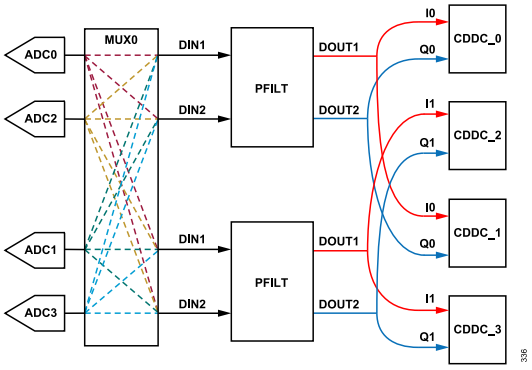

Figure 136. PFILT Rx Data Path Connectivity with Complex Data for AD9088

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

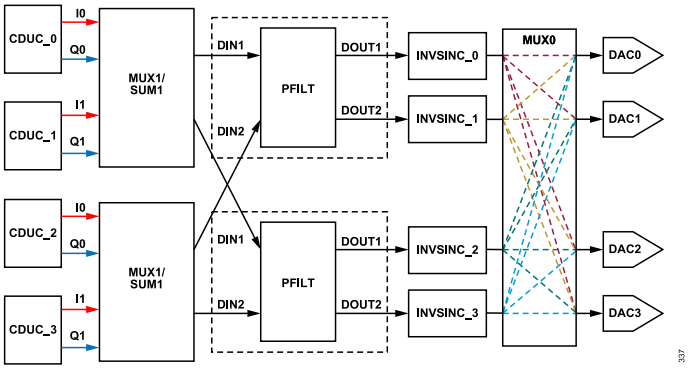

Figure 137. PFILT Tx Data Path Connectivity with Real Data for AD9088

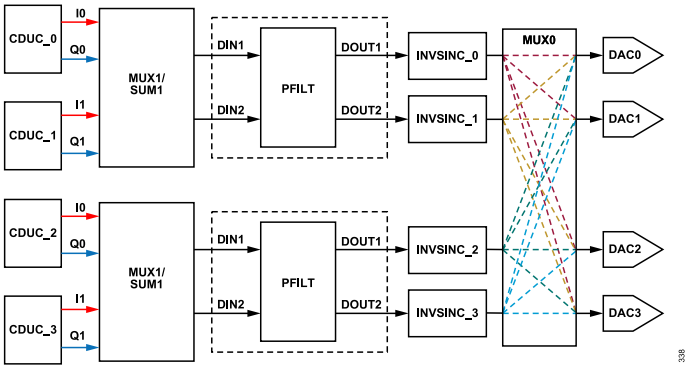

Figure 138. PFILT Tx Data Path Connectivity with Complex Data for AD9088

Programmable Gain

A programmable gain block at the output of each sub-filter can be used to apply a gain for adjustment, for example, to account for the gain of the filter coefficients. The gain block supports both the scalar gain and shift gain. The shift gain ranges from 0dB to 24dB, in 6dB steps. The scalar gain is an unsigned fractional number ranging from 1/64 (b000000) to 1 (b111111). The maximum scalar gain and the maximum shift gain cannot be used together. Therefore, the maximum gain that can be achieved is (63/64) × (24dB).

The programmable gain block is shown in Figure 139 (Note although it only shows the Matrix mode, all other modes can be considered as a simplification from this mode.).

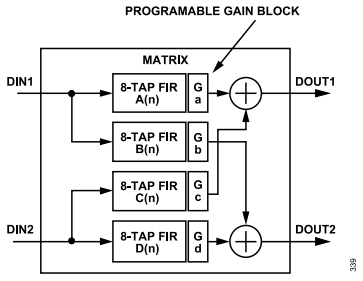

Figure 139. Optional Gain Block Following Each Filter

Users can configure both the shift gain and scaler gain in the API structure **adi_apollo_pfilt_gain_dly_pgm_t**, where **pfir_xx_gain** stands for the shift gain and **pfir_xx_scalar_gain** stands for the scalar gain. ix stands for the straight filter gain for DIN1 stream (i.e., Ga), iy stands for the cross-filter gain for DIN1 stream (i.e., Gb), qx stands for the straight filter gain for DIN2 stream (i.e., Gd) and qy stands for the cross filter gain for DIN2 stream (i.e., Gc). Therefore,

Ga = pfir_ix_gain × pfir_ix_scalar_gain

Gb = pfir_iy_gain × pfir_iy_scalar_gain

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

Gc = pfir_qy_gain × pfir_qy_scalar_gain

Gd = pfir_qx_gain × pfir_qx_scalar_gain

After configuring the gain values in the structure, API function **adi_apollo_pfilt_gain_dly_pgm()** is used to program the PFILT gains. Besides PFILT gains, user can also configure and program the delay in Half Complex mode through the same structure and API function.

Overflow Control

Internal saturation logic is present at carefully chosen points where overflow is likely to occur.

The DC gain of this structure is

Gain_DOUT1 = sum(A) × Ga + sum(C) × Gc

Gain_DOUT2 = sum(B) × Gb + sum(D) × Gd

Where sum( ) stands for the summation of the corresponding filter coefficients.

The worst-case gain is:

Gain_DOUT1_worst = sum(abs(A)) × Ga + sum(abs(C)) × Gc

Gain_DOUT2_worst = sum(abs(B)) × Gb + sum(abs(D)) × Gd

Saturation occurs if the output of each sub-filter (A/B/C/D) exceeds 4 or if the final output is >1.0.

ADC Averaging for Receiver PFILT

ADC averaging is used to average the outputs of two ADCs. When enabled, the two input data streams are added or subtracted from each other. ADC averaging is intended for use where the two input data streams carry the same data with different uncorrelated noise. ADC averaging should not be used with complex input data. To enable ADC averaging, user can configure API structure **adi_apollo_pfilt_mode_pgm_t** by setting **mode_switch** to be 0 and then set **add_sub_sel** for addition or subtraction. Example code for this feature can be found in the rx_adc_pave.c file that is in the ads10_apollo_ex_main folder of the API. The ADC averaging operation is illustrated in Figure 140. Note the DIN2 can be ignored in this case.

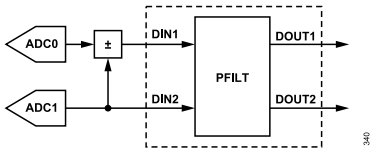

Figure 140. ADC Averaging for Rx PFILT

PFILT Coefficient Bank Description and Fast Updating Between Coefficient Banks

Each PFILT uses multiple coefficient banks as shown in Figure 141 for the AD9084 and AD9088. This implementation enables a user to dynamically switch between different coefficient sets that may have been configured or optimized for different applications or physical channel impairments.

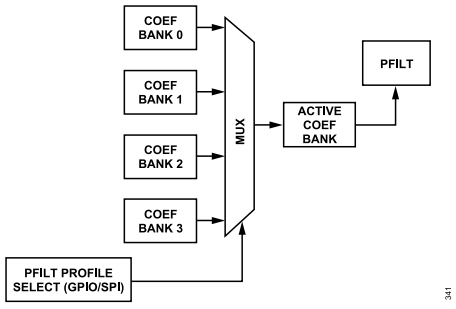

Figure 141. PFILT Coefficient Bank Load

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

One of the four sets of coefficients is used by PFILT at a time, and the other three sets are offline and can be updated through the SPI port. User can select the active bank through GPIO or SPI using APIs. The PFILT profile select mode is defined by **adi_apollo_pfilt_profile_sel_mode_e**. The selection could be direct or trigger based.

If using the GPIOx pins for selection, refer to the GPIO section for more information on the pin assignment. The specified coefficient bank is loaded whenever the GPIO pins are toggled. The supported GPIO configurations are listed in Table 108 and Table 109 .

When a coefficient set is selected, a transfer pulse transfers all coefficients together to the active coefficient bank, which is the working copy used by the PFILT engine. Having a separate transfer bit ensures that all coefficients used by the filters in the PFILT are changed simultaneously when changing from one set of coefficients to another.

Expect a normal filter transient during the coefficient switch. In addition to the normal transient from switching coefficients, invalid data can propagate for a few cycles after the switch.

Table 108. Supported GPIO Configurations for AD9084 (4T4R)

		N/2-Tap Mode	Other Modes
Config 0	GPIO[1:0]	Switch between all four coefficient banks for both Stream DIN1 and DIN2.	Switch between all four coefficient banks for both Stream DIN1 and DIN2.
Config 1	GPIO[0]	Switch between coefficient banks 0 and 1 for Stream DIN1(0 for bank 0 and 1 for bank 1).	Switch between coefficient sets 0 and 1 for both Stream DIN1 and DIN2.
	GPIO[1]	Switch between coefficient banks 2 and 3 for Stream DIN2 (0 for bank 2 and 1 for bank 3).	Unused

Table 109. Supported GPIO Configurations for AD9088 (8T8R)

		All Modes
Config 0	GPIO[1:0]	Switch between all four coefficient sets (for both PFILTs at each side).
Config 1	GPIO[0]	Switch between coefficient sets 0 and 1 for the first PFILT.
	GPIO[1]	Switch between coefficient sets 2 and 3 for the second PFILT.

PFILT Configuration Device Profile

PFILT is configured using the **adi_apollo_pfilt_cfg_t** data structure pulled from the **rx_pfilt** and **tx_pfilt** members of the device profile. The parameters defined in the **adi_apollo_pfilt_cfg_t** structure are listed in Table 110.

Table 110. adi_apollo_pfilt_cfg_t Parameters

Parameter	Description	Type
enable	Filter enable	bool (1: enable, 0: disable)
coeffs	PFILT coefficients	Unsigned 16-bit matrix. See coefficient table (16 bit with Q1.15 format)
i_mode	Filter mode for I stream	**adi_apollo_pfilt_mode_e**
q_mode	Filter mode for Q stream	**adi_apollo_pfilt_mode_e**
real_data	Real or complex data stream selection	**adi_apollo_pfilt_data_e**
dq_mode	Dual or quad mode	**adi_apollo_pfilt_dq_mode_e**
add_sub_sel	Addition or subtraction operation	**adi_apollo_pfilt_mode_sw_add_sub_e**
mode_switch	3dB averaging selection (only for receiver)	**adi_apollo_pfilt_mode_sw_ave_en_e**
pfir_ix_gain_db	I stream shift gain	**adi_apollo_pfilt_gain_e**
pfir_iy_gain_db	I stream shift gain (only in matrix mode)	**adi_apollo_pfilt_gain_e**
pfir_qx_gain_db	Q stream shift gain	**adi_apollo_pfilt_gain_e**
pfir_qy_gain_db	Q stream shift gain (only in matrix mode)	**adi_apollo_pfilt_gain_e**
pfir_ix_scalar_gain_db	I stream scalar gain	Unsigned 8-bit
pfir_iy_scalar_gain_db	I stream scaler gain (only in matrix mode)	Unsigned 8-bit
pfir_qx_scalar_gain_db	Q stream scalar gain	Unsigned 8-bit
pfir_qy_scalar_gain_db	Q stream scalar gain (only in matrix mode)	Unsigned 8-bit
hc_prog_delay	Programmable delay line	Unsigned 8-bit

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

The enumerations used by the data structures in the device profile to configure the PFILT are listed in Table 111.

Table 111. Enumerations associated with PFILT

Enumerations	Parameters	Description
adi_apollo_pfilt_idx_e	ADI_APOLLO_PFILT_IDX_0	0: PFILT index 0
	ADI_APOLLO_PFILTS_PER_SIDE	1: PFILT per side
adi_apollo_pfilt_mode_e	ADI_APOLLO_PFILT_MODE_DISABLED	0: Disabled (filters bypassed)
	ADI_APOLLO_PFILT_MODE_N_DIV_BY_4_REAL	1: Real N/4 Tap (8-Tap) Filter for the Channel
	ADI_APOLLO_PFILT_MODE_N_DIV_BY_2_REAL	2: Real N/2 Tap (16-Tap) Filter for the Channel
	ADI_APOLLO_PFILT_MODE_MATRIX	4: Real N/4 tap (8-Tap) Matrix mode of operation
	ADI_APOLLO_PFILT_MODE_HALF_COMPLEX	6: Half Complex Filter using N/2-Tap (16-Tap) Filters for the Q/I channel + N/2 (16-Tap) Tap Programmable Delay Line for the Channel
	ADI_APOLLO_PFILT_MODE_N_REAL	7: Real N Tap (32-Tap) Filter for the I/Q
adi_apollo_pfilt_gain_e	ADI_APOLLO_PFILT_GAIN_ZERO_DB	0: 0dB
	ADI_APOLLO_PFILT_GAIN_POS_6_DB	1: 6dB
	ADI_APOLLO_PFILT_GAIN_POS_12_DB	2: 12dB
	ADI_APOLLO_PFILT_GAIN_POS_18_DB	3: 18dB
	ADI_APOLLO_PFILT_GAIN_POS_24_DB	4: 24dB
adi_apollo_pfilt_coeffs_sets_per_prfl_e	ADI_APOLLO_PFILT_COEFF_SET_0	0: select PFILT coefficient set 0
	ADI_APOLLO_PFILT_COEFF_SET_1	1: Select PFILT coefficient set 1
	ADI_APOLLO_PFILT_COEFF_SET_2	2: Select PFILT coefficient set 2
	ADI_APOLLO_PFILT_COEFF_SET_3	3: Select PFILT coefficient set 3
	ADI_APOLLO_PFILT_COEFF_SETS	4: Select all PFILT coefficient sets
adi_apollo_pfilt_dq_mode_e	ADI_APOLLO_PFILT_DUAL_MODE	0x0: PFILT DUAL MODE
	ADI_APOLLO_PFILT_QUAD_MODE	0x1: PFILT QUAD MODE
adi_apollo_pfilt_data_e	ADI_APOLLO_PFILT_COMPLEX_DATA	0x0: PFILT COMPLEX DATA
	ADI_APOLLO_PFILT_REAL_DATA	0x1: PFILT REAL DATA
adi_apollo_pfilt_mode_sw_add_sub_e	ADI_APOLLO_PFILT_SUB_FOR_MOD_SW	0x0: Select subtraction operation
	ADI_APOLLO_PFILT_ADD_FOR_MOD_SW	0x1: Select addition operation
adi_apollo_pfilt_mode_sw_ave_en_e	ADI_APOLLO_PFILT_DISABLE_3DB_AVG_MOD_SW	0: Disable Mode Switch (3dB Average). Only for receiver PFILT.
	ADI_APOLLO_PFILT_ENABLE_3DB_AVG_MOD_SW	1: Enable Mode Switch (3dB Average). Only for receiver PFILT

PFILT Configuration API

The API library fully supports configuration of the PFILT blocks. PFILT is configured using the API functions listed in Table 112 based on the settings in the device profile. Refer to apollo_api.chm for more details on the device API functions, structures, and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 112. API functions for PFILT configuration

Function	Description
adi_apollo_pfilt_mode_pgm()	Configure PFILT mode parameters
adi_apollo_pfilt_gain_dly_pgm()	Configure gain and delay parameters
adi_apollo_pfilt_coeff_pgm()	Configure PFILT coefficients parameters
adi_apollo_pfilt_coeff_ntap_set()	Set PFILT coefficients for Real 32-tap and Real 16-tap modes
adi_apollo_pfilt_coeff_half_complex_set()	Set PFILT coefficients for half-complex mode
adi_apollo_pfilt_coeff_full_matrix_set()	Set PFILT coefficients for full-matrix mode
adi_apollo_pfilt_half_complex_delay_set()	Set PFILT delay in half-complex mode

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

Table 112. API functions for PFILT configuration (Continued)

Function	Description
adi_apollo_pfilt_coeff_transfer()	Select filter bank to transfer to active
adi_apollo_pfilt_inspect()	Inspect PFILT parameters
adi_apollo_pfilt_mode_enable_set()	set the PFILT mode only, can be used to enable or bypass the filter
adi_apollo_pfilt_profile_sel_mode_set()	Set the PFILT profile selection and hopping mode
adi_apollo_pfilt_next_hop_num_set()	Set the next PFILT profile selection or hop
adi_apollo_pfilt_ave_mode_set()	Set the Rx PFILT ADC averaging mode
adi_apollo_pfilt_data_type_set()	Set the PFILT data type for real or complex processing

PFILT Mode Hopping

Apollo's dynamic reconfiguration feature enables hopping form one PFILT configuration to another without bringing down the JESD204B/C link when certain conditions are met. Refer to the PFILT/CFIR Hopping subsection of the Dynamic Reconfiguration section for details.

Use Case Scenarios to Filter Modes Mapping

The optimal PFILT configuration depends on the application. Besides general purpose filtering, possible use cases for the different PFILT configurations include the following:

- Spectrum Equalization: the PFILT can be used to compensate for gain and/or phase impairments vs. frequency. This use case can either be for real signals or for complex signals. For real signals, one filter is used for each converter. When only one converter is used, the PFILT one real filter mode can be used such as the one 32-Tap Real Filter Mode. When two converters are used, the PFILT dual real filter mode can be used such as the dual 16-Tap Real Filter Mode. For complex signals, the dual real filter mode can be used, with one sub-filter for I and one sub-filter for Q signal. In addition, Matrix Mode can be used when configuring the four sub-filters properly.
- Crosstalk Correction: the PFILT views the converters as pairs that can be either two real signals or a complex I/Q pair. If the channels are separate real signals, the Matrix Mode can be used for crosstalk correction. In the Matrix Mode configuration, the filters B(n), and C(n) can be used to model the coupling transfer function between the channels and subtract an estimate of the cross talk from the other channel. The A(n) and D(n) filters are programmed with a single nonzero coefficient at the appropriate coefficient to compensate for group delay of the cross-term filter.
- Spectrum Equalization with Crosstalk Correction: this case is a combination of equalization of spectrum impairments and channel-to-channel crosstalk correction where the Matrix Mode can again be used. For this the single coefficient of A(n) and D(n) is replaced by a filter transfer function to perform the equalization and C(n) and D(n) can perform the crosstalk correction.
- Quadrature Error Correction: when a complex signal is sampled, then imperfections in the amplitude and phase imbalance of the I/Q signal can be corrected. The most common method for error correction is to use the Half-Complex Mode and the Matrix Mode. The Half-Complex mode can be used if there is no frequency asymmetry and the Matrix Mode can be used if there is frequency asymmetry.

In summary:

- The 32-tap Real Filter Mode could be used when only one ADC/DAC is enabled or when two ADC outputs/DAC inputs are averaged/combined. In this mode, PFILT could be used for spectrum equalization (e.g., magnitude compensation, phase error correction, etc.).
- The 16-tap or the 8-tap Real Filter Mode could be used when both ADCs/DACs are enabled for spectrum equalization for two real signals or a complex I/Q signal.
- The Half-Complex Mode could be used for quadrature error correction when one converter samples I data and the other samples Q data.
- The Matrix Mode could be used for crosstalk correction, quadrature error correction or spectrum equalization or a combination of some of them.

PROGRAMMABLE COMPLEX FIR FILTER (CFIR) - RECEIVE AND TRANSMIT

The Apollo MxFE CFIR has two operation modes: normal mode and sparse mode. In the normal mode, the CFIR is a 16-tap complex FIR filter. In the sparse mode, the 16-tap complex FIR filter may be extended up to 128-tap in which only 16 taps are non-zero. In addition, at the filter output, a gain adjustment block can be used to adjust the gain between -18 to +12 dB in 6 dB steps. Subsequently, a complex scalar after the gain adjustment can be used for more accurate gain adjustment or phase rotation.

Table 113 lists some of the key characteristics of the CFIR block. The maximum data rate is 5 GSPS for 16-bit I and 16-bit Q samples. The CFIR can accommodate 1, 2, or 4 data streams concurrently depending on different operating modes as shown in Table 113.

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

Table 113. CFIR Block Characteristics

Specification	Parameter	Comment
Max input data rate	5.0 GSPS	Can accommodate 1, 2, or 4 data streams
Max output data rate	5.0 GSPS	Matches input data rate
Maximum Number of filter taps (normal mode)	16	Maximum complex filter taps in normal mode
Maximum number of taps (sparse mode)	128	Maximum complex filter taps in sparse mode, and only 16 taps can be non-zero
Input data sample width	16 bits	16-bit I & 16-bit Q
Coefficient bit width	16 bits	16-bit I & 16-bit Q
Output sample width	16 bits	16-bit I & 16-bit Q

As shown in Table 114, depending on the Apollo mode and FDDC/FDUC mode, the CFIR can receive 1, 2, 3, or 4 distinct data streams. Each data stream can have their own filter coefficients, gain adjust and complex scalar values.

Table 114. CFIR Operating Modes

Apollo mode	FDDC/FDUC mode	Max data streams for each CFIR
4T4R	Dfine ≠ 1 (or) Ifine ≠ 1	Two data streams maximum (at 2.5GSPS)
8T8R	Dfine ≠ 1 (or) Ifine ≠ 1	Four data streams maximum (at 1.25GSPS)
4T4R	Dfine = 1 (or) Ifine = 1	One data stream maximum (at 5GSPS)
8T8R	Dfine = 1 (or) Ifine = 1	Two data streams maximum (at 2.5GSPS)

Note the number of streams the CFIR receives is not part of the CFIR programmability. It is determined by the FDDC on the Rx path and FDUC on the Tx path.

As aforementioned, there's a coarse gain adjustment block (Gain_adj, 6dB step size) at the filter output and then a complex scalar multiplier (Cmlt). This arrangement allows user to filter and change the gain of the baseband samples ahead of the FDUC block at the transmit channel or after the FDDC block at the receive channel. The gain adjustment block allows controlling the gain between -18dB and 12dB in 6dB increments. This gain stage is followed by a complex, 16-bit complex scalar block for fine gain adjustments, in the form of (I+jQ), where I and Q are programmable 16-bit constants.

Note that the user needs to apply appropriate gains to avoid sample-clipping where the CFIR output exceeds 16 bits, leading to signal distortion.

Normal CFIR Mode

The normal CFIR mode can be expressed by the following equation:

$$y[n] = \sum_{i=0}^{15} x[n-i]*h[i] \tag{36}$$

Where x[n] stands for the CFIR input signal, h[n] stands for the CFIR coefficients, and y[n] stands for the CFIR output signal, which could all be complex values.

The gain adjust operation shown in Figure 142 is just a bit shift operation with overflow detection and saturation logic. The 0 dB gain level is determined by loading the filter coefficients with a negative impulse response (0x8000 for h[0] followed by all 0's). Then, loading the complex scalar with –1 (0x8000) in the complex multiplication (Cmlt) block. Setting the gain_adj value to 3'b011 (0dB) in the Gain_adj block produces output samples that equal the input samples.

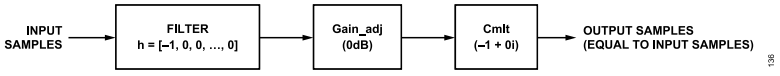

Figure 142. CFIR Data Flow for Gain Adjustment and Configurations to Produce Output Samples Equal to the Input Samples

adi_apollo_cfir_gain_pgm() API can be used to configure the gain adjustment value. Table 115 lists a set of supported gain adjustment values and the corresponding API constants defined by the "**adi_apollo_cfir_gain_e**" enumeration.

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

Table 115. CFIR Configuration Gain Values

adi_apollo_cfir_gain_e	Gain
ADI_APOLLO_CFIR_GAIN_MINUS18_DB	-18 dB
ADI_APOLLO_CFIR_GAIN_MINUS12_DB	-12 dB
ADI_APOLLO_CFIR_GAIN_MINUS6_DB	-6 dB
ADI_APOLLO_CFIR_GAIN_ZERO_DB	0 dB
ADI_APOLLO_CFIR_GAIN_PLUS6_DB	+6 dB
ADI_APOLLO_CFIR_GAIN_PLUS12_DB	+12 dB

The complex scalar operation multiplies the data samples from the "Gain_adj" block ($x_r + i*x_i$) by a constant complex value ($c_r + i*c_i$) and then generate the final CFIR output samples y_r+i*y_i, as shown in the following equation. The default value of the complex scalar is (16'h7fff + 16'h0000*i).

$$(y_r + i \times y_i) = (x_r \times c_r - x_i \times c_i) + I \times (x_r \times c_i + x_i \times c_r) \quad (37)$$

adi_apollo_cfir_scalar_pgm() API can be used to configure the constant complex value.

Note that overflow detection and saturation is checked at the complex scalar output. The CFIR IRQ will be supported in a future revision of the Apollo MxFE API.

Sparse CFIR Mode

In sparse mode, the 16-tap CFIR can be expanded up to 128 taps with the limitation that only up to 16 taps can be non-zero. Note, there are other limitations that are explained below.

Figure 143 illustrates how the 16 non-zero taps are selected corresponding to the 128 samples in the delay line. As shown in this figure, there are a total of eight data stores (Dstore0-7), each representing 16 I/Q samples. The 4 lower store blocks (Dstore0/2/5/7) feed into the data path mux (Dpath_mux) block directly. The upper data store blocks (Dstore1/3/4/6) indicate the relative delay inserted between the blue blocks. The Dpath_mux receives 64 sets of I/Q samples and outputs 16 I/Q samples based on the configuration of hsel. Dstore0 represents the most recent 16 samples (x[n], x[n-1], …., x[n-15]) which are always fed into the Dpath_mux block. For Dstore2, it has two options depending on the configuration of mem_sel0. If mem_sel0 = 0, Dstore2 represents the next 16 I/Q samples (x[n-16], x[n-17], …., x[n-31]) relative to Dstore0 with no extra delay inserted. If mem_sel0 = 1, Dstore2 represents a set of samples which is delayed by 16 samples caused by Dstore1, which is x[n-32], x[n-33], …., x[n-47]. Similarly, for Dstore5, it has three options depending on the configuration of mem_sel1. If mem_sel1 = 0, Dstore5 represents the next 16 I/Q samples relative to Dstore2 with no extra delay inserted. If mem_sel1 = 1, Dstore5 represents a set of samples which is delayed by 16 samples caused by Dstore3. If mem_sel1 = 2, Dstore5 represents a set of samples which is delayed by 32 samples caused by Dstore3 and Dstore4. For Dstore7, it has two options depending on the configuration of mem_sel2. If mem_sel2 = 0, Dstore7 represents the next 16 I/Q samples relative to Dstore5 with no extra delay inserted. If mem_sel2 = 1, Dstore7 represents a set of samples which is delayed by 16 samples caused by Dstore6.

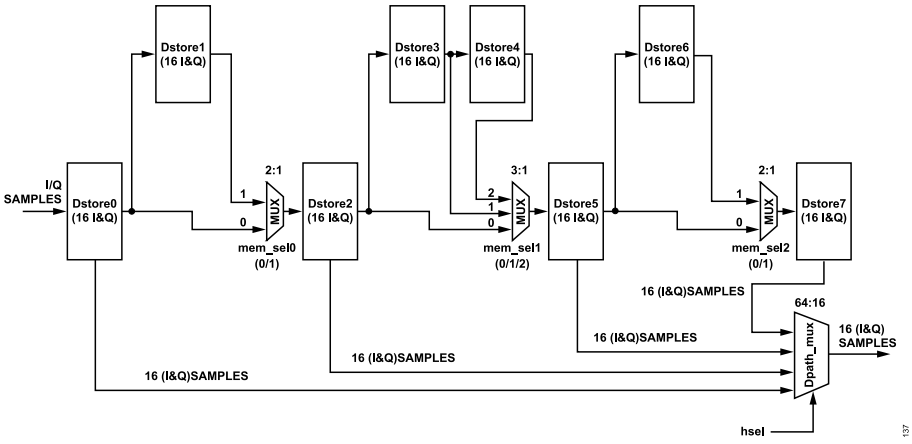

Figure 143. 16 Non-zero Taps Selection in CFIR Sparse Mode

Mathematically, the samples in the blue blocks can be derived as:

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

Dstore0: x[n], x[n-1], …, x[n-15]

Dstore2: x[n-16*(mem_sel0+1)], x[n-16*(mem_sel0+1) -1], …, x[n-16*(mem_sel0+1) -15]

Dstore5: x[n-16*(mem_sel0 +mem_sel1+2)], x[n-16*(mem_sel0+mem_sel1+2) -1], …, x[n-16*(mem_sel0+mem_sel1+2) -15]

Dstore7: x[n-16*(mem_sel0 +mem_sel1+mem_sel2+3)], x[n-16*(mem_sel0+mem_sel1+mem_sel2+3) -1], …, x[n-16*(mem_sel0+mem_sel1+mem_sel2+3) -15]

"hsel" defines up to 16 non-zero values and each value is from 0 to 63, which picks 16 samples from the 64 inputs:

Values 0 to 15 map to samples in Dstore0. Values16 to 31 map to samples in Dstore2. Values 32 to 47 map to samples in Dstore5. Values 48 to 63 map to samples in Dstore7.

Figure 144 shows two examples of selecting the location of 16 non-zero taps relative to the 128 input samples with the proper configurations of mem_sel[0:2] and hsel:

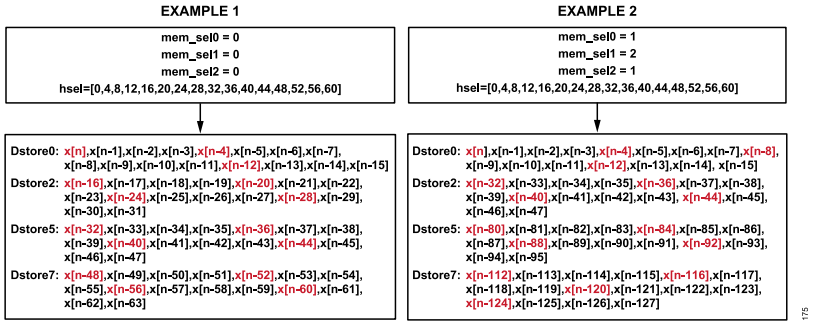

Figure 144. Examples of Selecting 16 non-zero Taps Locations

Sparse CFIR Mode Limitations

As shown in Example 2 in Figure 144, the Dstore0/2/5/7 samples could be non-continuous, which poses a limitation on the hsel values that can be selected. The reason is that the CFIR is implemented as multiple data paths and each data path usually generates 2 consecutive output samples for each clock cycle. In order to do that, if sample x[k] is selected, then x[k-1] has to be available as well. But for non-continuous Dstores, x[k-1] might not be available which means x[k] could not be selected.

In Example 2, due to the above reason, hsel values "15,31,47 and 63" could not be selected. In Example 1, although Dstores0/2/5/7 are continuous, user should note that the last sample which corresponds to hsel value 63 could not be selected.

Figure 145 shows another example of the limitation. In this configuration, sample x[n-31], x[n-63] and x[n-95] are non-continuous, which correspond to hsel value 31, 47 and 63.

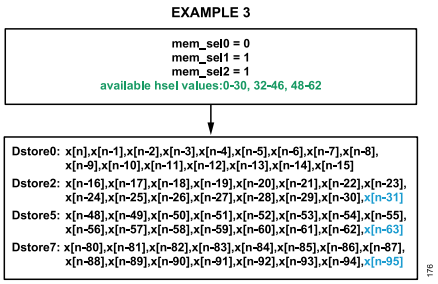

Figure 145. Available hsel Values due to CFIR Sparse Mode Limitation

Receive Path CFIR Operating Modes

For each of the A/B sides, there are two receiver CFIR instances per data path on the AD9084 and four per data path on the AD9088. The receiver CFIR receives data from the FDDCs, then the receiver CFIR output is driven into the receiver FSRC block.

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

The receiver CFIR can be used to post-equalize the roll-off within the interested band that caused by the analog and/or digital components before the CFIR, such as amplifier, mixer, converter, reflection ripples due to the impedance mismatching, digital filters, and etc.

Transmit Path CFIR Operating Modes

Similar to the receive path, for each of the A/B side, there are two transmit CFIR instances per data path on AD9084 and four per data path on the AD9088. The transmit CFIR receives data from the transmitter FSRC, then the transmitter CFIR drives its output to the FDUCs.

The transmit CFIR can be used to pre-equalize the roll-off within the interested band that caused by the analog and/or digital components after the CFIR, such as digital filters, amplifier, mixer, converter, reflection ripples due to the impedance mismatching, and etc.

CFIR Configuration API

The API library fully supports configuration of CFIR. The CFIR is configured using the API functions listed in Table 116 based on the settings in the device profile. Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 116. API Functions for CFIR Configurations

Function	*Description*
adi_apollo_cfir_pgm()	Configure CFIR parameters
adi_apollo_cfir_coeff_pgm()	Load CFIR coefficients for one or more profiles
adi_apollo_cfir_scalar_pgm()	Load CFIR complex scalar values for one or more profiles
adi_apollo_cfir_gain_pgm()	Load CFIR gain adjustment values for one or more profiles
adi_apollo_cfir_sparse_coeff_sel_pgm()	Load CFIR sparse mode coefficients sel (hsel) for one or more profiles
adi_apollo_cfir_sparse_mem_sel_pgm()	Load CFIR sparse mode mem_sel values for one or more profiles
adi_apollo_cfir_profile_sel()	Select CFIR profile
adi_apollo_cfir_mode_enable_set()	Enable or bypass CFIR
adi_apollo_cfir_inspect()	Inspect CFIR parameters
adi_apollo_cfir_profile_sel_mode_set()	Set the CFIR profile selection and hoping mode
adi_apollo_cfir_next_hop_num_set()	Set the next CFIR profile selection or hop

CFIR is configured using the **adi_apollo_cfir_cfg_t** data structure pulled from rx_cfir and tx_cfir sections in the device profile. The parameters defined in the **adi_apollo_cfir_cfg_t** structure are listed in Table 117 .

Table 117. adi_apollo_cfir_cfg_t parameters

Parameter	Description	Types
coeffs_i	Real part of CFIR coefficients	3-D array
coeffs_q	Imaginary part of CFIR coefficients	3-D array
cfir_gain_DB	CFIR gain in dB	Enum: ADI_APOLLO_CFIR_GAIN_MINUS18_DB
		ADI_APOLLO_CFIR_GAIN_MINUS12_DB
		ADI_APOLLO_CFIR_GAIN_MINUS6_DB
		ADI_APOLLO_CFIR_GAIN_ZERO_DB
		ADI_APOLLO_CFIR_GAIN_PLUS6_DB
		ADI_APOLLO_CFIR_GAIN_PLUS12_DB
scalar_i	Real part of scalar gain	2-D array (Cr)
scalar_q	Imaginary part of scalar gain	2-D array (Ci)
Enable	Enable CFIR block	Enable/disable CFIR in boolean
sparse_mode	Enable sparse filter	Enable/disable sparse mode in boolean
cfir_mode	Select single or dual mode	Enum: ADI_APOLLO_CFIR_SINGLE_BAND
		ADI_APOLLO_CFIR_DUAL_BAND

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

Use Case Scenarios

One main use case of CFIR is to equalize the RF carriers independent of PFILT. Comparing with PFILT, CFIR has a lower sample rate, which could achieve better equalization performance than PFILT with fewer number of taps since the taps are not wasted on out of band spectrum. However, the disadvantage of using the CFIR for equalization is that different coefficients are required for different RF carrier frequencies. User could store two sets of coefficients and switch between them. For a system with many different RF frequencies (such as frequency hopping), user may need to reprogram coefficients on the fly when switching RF frequencies. Refer to the PFILT/CFIR Hopping sub-section of the Dynamic Reconfiguration section.

Another main use case of CFIR is the sparse mode which has 16 non-zero taps with a total of 128 taps. The remaining taps are zero. The non-zero taps can be anywhere in the impulse response, therefore, the CFIR can compensate longer time echoes from long cable without adding more non-zero taps.

LOOPBACK MODES

The Apollo MxFE device has four loopback paths in the design and illustrated in Figure 146 and include preliminary estimates of the latency for each segment of the loopback functions.

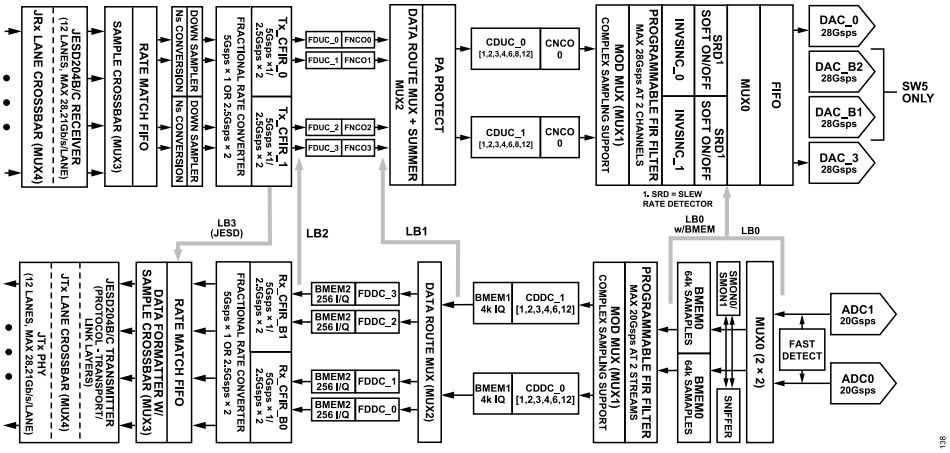

Figure 146. AD9084 Loopback Block Diagram with Latency Estimations

Receiver to Transmitter Analog Loopback (Loopback0)

Loopback0 (LB0), the data out from the ADC data FIFO is looped back to the transmitter DAC data FIFO via minimum latency path. In this loopback mode, latency is deterministic and the receive data path is not interrupted (data still flows through to the JESD204B/C outputs). When using LB0, the ADC and DAC sample rates must be the same.

Users have the option to add programmable delay using the BMEM block that is located in the “HSDIN” block of the receive path.

LB0 Configuration Using API

The API functions for configuring and enabling LB0 are in the loopback0.c example code file which is part of the ads10_apollo_ex_main example folder. The adi_apollo_loopback.h header file contains the device structures for each of the LB0 functions.

The API functions for configuring and enabling LB0 with BMEM delay are in the lb0_bmem_delay_hop.c example code file. In addition to the adi_apollo_loopback.h header file containing the device structures for the LB0 functions, the adi_apollo_bmem.h file structures are also used. Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Receive CDDC to Transmit CDUC (Loopback1)

Loopback1 (LB1) will route the data out of the CDDC in the receive path into the CDUC in the transmit path at the data router mux. Loopback1 characteristics:

- Latency is deterministic for a particular mode

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

- Programmable delay using CDDC BMEMs
- MxNx, 1xNx & 1x1x cases supported.
- Blending support present with 1x, 0.5x, 0.25x scaling options for LB data

Loopback1 constraints:

- ADCs and DACs should run at same Fs
- Data rate should be same in receiver & transmitter
- Same decimation/interpolation configurations
- Blending Not supported for Fine DDC/DUC bypass cases.
- Only CDDC_A0 -> CDUC_A0 , CDDC_A1 -> CDUC_A1 supported . Crossing not supported.
- Max Data rate at CDDC output for MxNx cases ≤ 5GSPS

Note that:

- M is the Fine interpolation/Decimation
- N is the Coarse interpolation/Decimation
- Blending is the weighted summation of LB data with transmitter JESD data. For the LB data, the supported weights are 1x, 0.5x, and 0.25x.

LB1 Configuration Using API

The loopback1 and loopback2 example code is located in the loopback1_2.c file which is part of the ads10_apollo_ex_main example folder. The adi_apollo_loopback.h header file contains the device structures for all of the loopback functions.

Receive FDDC to Transmit FDUC (Loopback2)

Loopback2 (LB2) will route the data out of the FDDC in the receive path into the FDUC in the transmit path at the data router mux. LB2 characteristics:

- Supports Loopback in Fine DDC/DUC enabled & Fine DDC/DUC bypass with FNCO enabled cases
- Latency is deterministic for a particular mode
- Programmable delay using CDDC & FDDC BMEMs

Loopback2 constraints:

- ADCs and DACs should run at same Fs
- Data rate should be same in receiver & transmitter => same decimation/interpolation configurations
- Same channel FDDC to FDUC loopback supported. Crossing not supported.
- With FDDC BMEM enabled, both FDDCs in a channel should be in same decimation

LB2 Configuration Using API

The loopback1 and loopback2 example code is located in the loopback1_2.c file which is part of the ads10_apollo_ex_main example folder. The adi_apollo_loopback.h header file contains the device structures for all of the loopback functions.

JESD204 (JRx to JTx) Loopback (Loopback3)

As indicated in Figure 146, the JESD204 loopback (LB3) allows the user to route sample data from the JESD204B/C receiver's transport layer to the input of the JESD204B/C transmitter's transport layer. Note that the loopback point in the JESD204B/C receiver is after the FSRC and CFIR blocks , so these blocks must be disabled.

JESD204 Loopback constraints:

- ADCs and DACs should run at same Fs
- Loopback mode is supported only for link0.
- Only the sample data is looped back (control bits are not). So, CS parameter(number of controls bits should be 0).

ADVANCED DIGITAL FEATURES FOR SYSTEM APPLICATIONS

- The delay formula for JTX depends on JTX_Ns value. So, it may not be the same delay value as the non-loopback mode. But the same formula will still work.
- The transmitter data path doesn't support the interpolation settings in Table 118 . So, they cannot be used when operating the JESD204 loopback.

Table 118. Interpolation Modes not Supported for JESD204 Loopback

Variant	CxF_interp
4T4R	12x1, 6x1, 1x1
4T4R_LS	3x1, 12x1, 2x1, 1x1, 6x1
8T8R	6x1, 12x1
8T8R_LS	3x1, 6x1, 1x1, 2x1, 12x1

*LS = "low sample rate mode"

- In non-loopback mode JTX_NS is calculated differently so some modes are not supported as listed in Table 119.

Table 119. JESD204B/C Transmitter Modes not Supported for JESD204 Loopback

JESD204 Transmit Mode Number	Device Variant	COARSE_DEC	FINE_DEC
6	4T4R_LS	4	1
11	4T4R_LS	4	1
12	4T4R_LS	4	1
13	4T4R_LS	4	1
18	4T4R_LS	4	1
26	4T4R_LS	4	1
27	4T4R	4	1
27	4T4R_LS	4	1

LB3 Configuration Using API

The jesd_loopback example code is located in the jesd_loopback.c file which is part of the ads10_apollo_ex_main example folder. The adi_apollo_loopback.h header file contains the device structures for all of the loopback functions.

MORE AUXILIARY FEATURES

RECEIVE AGC ASSIST FUNCTIONS

There are three functions in the receive path that detect the input signal level to assist RF front-end automatic gain control (AGC): Fast Detect, Over-range Detect, and Signal Monitor. Fast Detect is optimized for speed and protection. Over-range Detect is used to gate calibration updates. Signal Monitor is optimized for linearity and precision in achieving the desired signal level. In the data flow through the signal chain, Fast Detect is first after the ADC. Followed by Over-Range Detect, and then Signal Monitor is placed last. Figure 147 shows the signal chain, and locations of each block. Further details on each section are described in the subsequent sections.

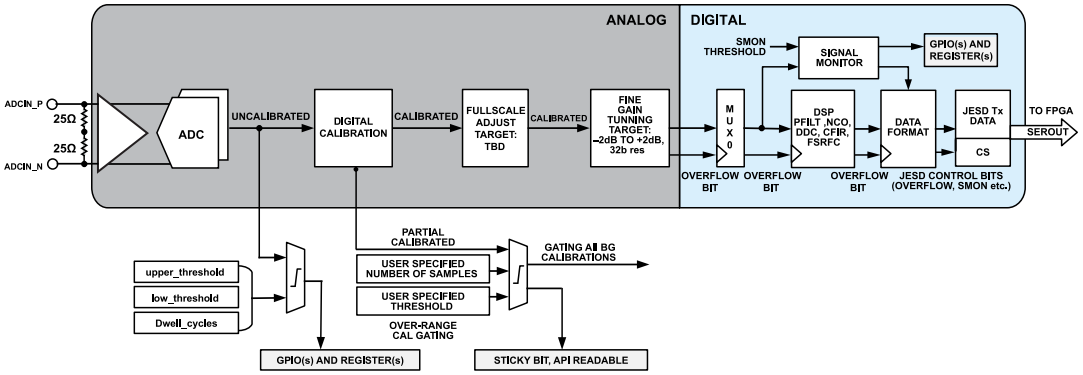

Figure 147. Receive Signal Chain and Block Locations of Input Signal Level Detection

Fast Detect (FD)

Each ADC has its own Fast Detect (FD). FD may refer to either the Fast Detect functional block as a whole or its one binary output bit. The name “fast detect” is inspired by the importance of speed in the critical path of detecting a threshold at which the ADC is nearing saturation and then responding quickly with a reduction in the magnitude of the signal driving the ADC – i.e., via an AGC loop. The objective in using FD is to avoid ADC saturation and the loss of valid information in the Rx path while waiting for digital filters to be cleared before the Rx path stream is valid again. The greatest possible speed is achieved by mapping each FD signal to a GPIO pin, where it can be directed into an attenuator of the ADC’s incoming signal. But there are also other means to convey FD: as SPI/HSCI readback or as a JESD control bit.

To facilitate speed in FD control that isn’t limited by the full pipeline latency of the ADC (from its analog in, to its derived 12-bit output code), FD bypasses most of that latency and uses instead an early approximation of the final code. This results in an error of up to 2dB, which should be counted in as margin when setting the [upper] FD activation threshold. (For greater accuracy at the expense of increased latency, consider using SMON.)

Each FD has three user-programmable detection parameters: Upper Threshold (“UT”), Lower Threshold (“LT”), and Dwell Time. FD is set within one clock cycle of when the absolute value of the approximated ADC code crosses above the UT. FD is reset when the same code drops below the LT for longer than the Dwell Time. Figure 148 illustrates the assertion and de-assertion of FD over time as a function of approximated ADC waveform:

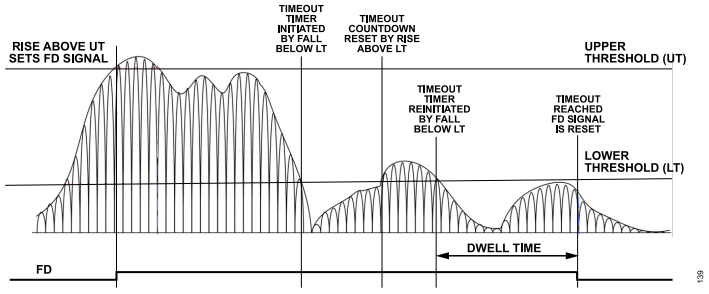

Figure 148. Fast Detect Timing

The FD thresholds are specified with 11-bit resolution, corresponding to the absolute value (bi-directional amplitude) range for the 12-bit ADCs. Given a specified threshold of <*value*>, it translates to dBFS as

$$\text{Threshold (dBFS)} = 20 \times \log_{10}(\langle value \rangle / 2^{11}) \quad (38)$$

MORE AUXILIARY FEATURES

A maximum value of 0x65A is recommended to accommodate a headroom error of up to 2dB mentioned above, meaning the final ADC code magnitude might be just reaching full scale (positive or negative) when the early approximation was only 0x65A.

FD de-assertion dwell time is specified by the User as a 28-bit multiplier of $64/N_{ADC}$ times the ADC sample clock period, where N_{ADC} is the number of ADCs. (Note that Apollo MxFE variants have either 4 or 8 ADCs). As an equation, this is

Dwell Time = <*value*> × $(64/N_{ADC})$ × $1/f_{ADC}$

For example, AD9084 has 4 ADCs, so N_{ADC}=4, and $64/N_{ADC}$=16; the maximum programmed dwell time value of 0xFFFFFFF corresponds to a dwell time of 16 x 2^{28} clock cycles. For f_{ADC} = 20GHz, $1/f_{ADC}$ = 50 ps, and the dwell time would be ~215 ms. There is a required minimum specified <*value*> of 32.

With the asymmetry in the set and reset time for FD (that is more evident when a long dwell time is used) FD is not necessarily intended to stand alone as the only control path in the ADC for implementing automatic gain control (AGC). It is the fastest path available. Another block, SMON, is more suitable for fine tuning the AGC to achieve a targeted input signal level – see the Signal Monitor section for details.

Fast Detect API

The fast detect feature is enabled and configured using the **adi_apollo_adc_fast_detect_pgm()** function. The **adi_apollo_adc_fast_detect_status_get()** function retrieves the fast detect status. Parameters can be found in the **adi_apollo_adc_fast_detect_pgm_t** structure as described in Table 120. Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 120. Fast Detect Parameters

Parameter	Definition
enable	Enable (1) or disable (0) fast detect
upper_threshold	Upper threshold value [threshold = (10^(dbfs/20) * 2^11)]
lower_threshold	Lower threshold value [threshold = (10^(dbfs/20) * 2^11)]
dwell_cycles	Value multiplied by $64/N_{ADC}$ to set FD dwell in ADC sample periods

Over-range (OVR) Calibration Gating

Each ADC has its own Over-range (OVR) block. The OVR block is after FD, and before Signal Monitor. The data the block sees is partially calibrated. The OVR block is used to gate (stop) all background ADC calibrations if a user defined number of samples exceeds a user defined over-range threshold. The output of the OVR block is a sticky bit, which is readable by the API. The sticky bit will indicate that the OVR was triggered, and that the BG calibrations were stopped.

The data is relative to nominal (500 mVpp) full-scale, it does not scale with full-scale adjustment. Because the OVR block uses partially calibrated data, the detection error is < +/- 0.5 dB at nominal full-scale.

The user can define two parameters for the OVR block, number of samples and threshold value. The sample range is 1 to 10,000 samples. This can be stepped in 1 sample increments. The threshold can be scaled by +0.5 dB to -3 dB in 0.5 dB steps (- 0.5 dBFS to -4 dBFS relative to nominal full-scale). By default, the OVR block will use 100 samples and -1 dBFS threshold value.

Over-range API

The functions to set and read the OVR parameters are located in the **adi_apollo_adc.c** file. The user can set the OVR threshold and the number of samples, as described above. The **adi_apollo_adc_input_status_get()** function will read back the ADC's input level threshold that will engage OVR. The **adi_apollo_adc_ovr_threshold_set()** function will configure the ADC input level that will engage OVR. The **adi_apollo_adc_over_samples_get()** function will read back the number of samples that will engage OVR if they are over the threshold. The **adi_apollo_ovr_samples_set()** function configures the number of samples required to be over the threshold for OVR to be engaged. In addition to setting the parameters, the function **adi_apollo_adc_input_status_get()** reads a struct that contains status for ADC input protection monitored events, including OVR.

Signal Monitor (SMON)

Each logical ADC is enumerated based on Rx Mux1 outputs and has its own signal monitor (SMON) block. "SMON" in the text refers to the entire functional block, not to an output signal. SMON captures the peak ADC output code that is generated within a user-specified time

MORE AUXILIARY FEATURES

interval. At the end of each successive time interval, SMON can provide that peak value directly, or post-process it to provide more condensed information, such as how it compares to two thresholds (like the FD block) or how it is changing over adjacent intervals. Within this SMON section, terms such as “signal peak”, “peak amplitude”, and “peak value” are used interchangeably.

The ADC signal and the SMON thresholds are specified with 11-bit resolution. This corresponds to the absolute value (bi-directional amplitude) range for the 12-bit ADCs. Given an ADC signal peak measurement of <value>, it translates to dBFS as

$$\text{signal peak (dBFS)} = 20 \times \log_{10}(\text{<value>}/2^{11}) \quad (39)$$

The most fundamental parameter that the user must program for SMON is the interval (“observation period”) over which the peak amplitude detected will be updated. Thus, it functions as a peak hold detector. The update interval is programmed with 16-bit resolution, supporting up to 65535 ADC clocks.

The peak amplitude can be fed into an AGC loop and used for externally optimizing the ADC drive level. The value can be read via SPI or transported as control bits over the JESD interface.

Alternately or additionally, the peak value can be compared against two user-programmable levels (high and low) and conveyed as two binary values, e.g. via GPIO pins, [1] and [0], respectively. An SMON GPIO signal pair is shown in Figure 149 for an arbitrary ADC waveform. The signals are updated at the end of each SMON interval to reflect the highest region that the ADC absolute value reached during the prior interval.

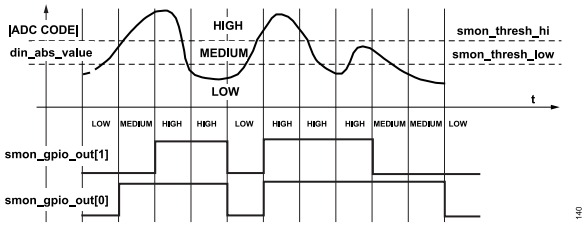

Figure 149. ADC code absolute value crossing SMON thresholds; corresponding SMON GPIO output pair updated at intervals

In multi-Apollo MxFE systems that have established synchronization (which is beyond the scope of this section), SMON updates can be synchronized to a counter based on the SYSREF edge (which is aligned) to facilitate SMON alignment across multiple Apollo MxFE devices.

Signal Monitor API

The signal monitor (SMON) feature is enabled and configured using the **adi_apollo_smon_pgm()** function. The **adi_apollo_smon_read()** and the **adi_apollo_smon_status_update()** functions read and update the signal monitor status. Parameters can be found in the **adi_apollo_smon_pgm_t** structure. Refer to the adi_apollo_smon.c file for more details. Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

TRANSMIT DOWNSTREAM POWER AMPLIFIER PROTECTION

Overview

Several block functions perform monitoring to collectively assist in protection of [external] power amplifiers (PAs). Internally, this is accomplished by control of the DACs via the data streams that drive them. In line with the signal path to each DAC is a multiplier that is controlled by a PA protection block – a.k.a. “PAP” – capable of complete attenuation and “soft” up/down (on/off) ramp control of gain applied to the digital stream. Input monitoring functions capable of triggering a PAP event include:

- excessive output power detection,
- error detection and synchronization discontinuity in the JESD receive path,
- SPI or HSCI errors,
- dynamic reconfiguration,
- clock PLL stability,
- DAC-enable registers or GPIOs,
- excessive slew rate detection (SRD).

Additionally, the PAP block can provide direct flagging of any of these errors to allow the user to take external action to protect the PAs.

MORE AUXILIARY FEATURES

The simplified block diagram in Figure 150, with a right-to-left data flow, represents the functional extent of the PAP block's reach. It shows the flow of data from a baseband processor via JESD out through an external PA via the DAC. The heart of the PAP block is shown as "PA SM" (PA [Protect] State Machine); and includes average power detection (APD), slew rate detection (SRD). JESD errors show inputs to the PA SM prior to data entering the data path. The data flush input from the PA SM is depicted as a mux, suggesting that it can flush – i.e. zero out – data, which is an important aspect of TDD switching.

Ramp down/up control and the ability to flush input data to zeroes are depicted as outputs of the PA SM. The subsequent text offers some elaboration on these functions.

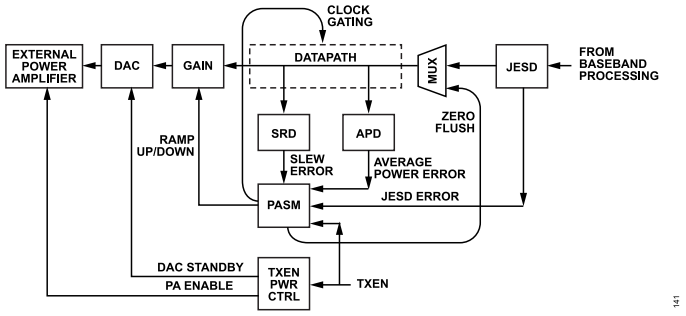

Figure 150. PA Protection Block Diagram

Multiplier – Ramp-Up and Ramp-Down Gain Control

The multiplier block is in the DAC signal paths just prior to the mux that directs incoming data streams to DACs. Each data channel has its own multiplier. Each multiplier is under control of the PAP block, which can attenuate the magnitude of the data stream driving a DAC channel. It is typically desired for attenuation or release of attenuation in a DAC output to take the form of a soft (limited slew-rate) ramp, so a ramp controller is provided, with a ramp rate programmable by the User. During normal operation, when a PAP event is triggered, the ramp controller is activated to begin a controlled ramp down of a given DAC signal. When the event is cleared, and in some cases with an option to add a delay, the attenuation is released, followed by a ramp up to the normal full-scale level. The full-scale level of the maximum gain can exceed 1, if desired, so the multiplier functions also as a DC gain in the digital data path.

Excessive Output Power Detection

The data streams that enter the CDUC block have a monitoring path around the CDUC block and into a power detection block to check for excessive output power. Power that is deemed excessive, per a User-defined threshold, can then trigger a PAP event – e.g., to initiate a protective ramp down of the DAC signal via the multiplier.

CDUC input channels are complex data streams, and average power detection begins by first squaring and then summing the samples in the I and Q channels composing each complex stream. Only the first six MSBs of samples are used for computation.

There are two User–programmed excessive power thresholds per stream corresponding to two moving average filter power computations per data stream. The shorter moving average filter is intended for limiting excessive peaks that might cause clipping or breakdown or a heavily saturated response; the longer moving average filter is intended for preventing thermal breakdown. Excessive power in either computation will trigger a PAP event. The time length of each measurement window over which power is averaged is defined in clock cycles and is specified by the user as an exponential power of 2.

The moving average computation of power and its comparison to a threshold is not instantaneous and thus has limiting conditions for which it can effectively employ the PAP function. (I.e., if a stream can pass through the CDUC and to the multiplier faster than its average power could be calculated and used to trigger a PAP event into the multiplier to attenuate the power, then it would not be effective.) So, it is left to the user to be sure the latency in the signal path is sufficiently long and the averaging window sufficiently short to limit transients. There are also sample rate limitations for the power detection circuit: in the AD9084 the maximum sample rate is 5 GS/s and in the AD9088 it is 4 GS/s. Divide the DAC output clock by the coarse interpolation to determine the sample rate into the power detection circuit.

JESD Error Detection and Protection

In the JESD receiver for the transmit path, errors detected can trigger a PAP event – i.e. a protective ramp-down of DAC outputs. This is the list of JESD errors:

MORE AUXILIARY FEATURES

- JESD204B
  - UEK
  - NIT
  - DIS
- JESD204C
  - CRC_err
  - SH_err
  - MB_err
  - EMB_err
- JESD204B/C
  - Invalid_sample_err
  - Lane_fifo_empty
  - Lane_fifo_full
  - Mfifo_empty
  - Mfifo_full
  - Data_rdy_lost

During synchronization – e.g. when clocking is realigned – the resulting transients can be suppressed using the synchronization notification.

Excessive Slew Rate Detection

As transmit data channels pass into the multiplier, the magnitude difference between adjacent samples can be compared against a user-defined threshold. If the slew rate is found to be excessive, a PAP event can be triggered and the transient suppressed. This crude but simple method of transient control is still effective when the latency in the CDUC is too short for detection in the average power detector to be effective for transient suppression.

Transmit Power Enable/Disable Control

The Apollo MxFE has transmit enable (TxEN) and receive enable (RxEN) functions that are used during Apollo MxFE Bring-up Procedure. The ability to power on the transmitter and receiver can also be controlled by the end application using GPIO pins to provide faster on-off response. Refer to the Using GPIO's to Implement Time Division Duplexing (TDD) <add link> section for more details.

GPIOX PIN OPERATION

AD9084 has 35 CMOS GPIO PADs and 8 LVDS PADs (4 input and 4 output). Each LVDS PAD (SYNCx) can be configured to function as 2 CMOS GPIOs, for a total of 51 CMOS GPIOs. AD9084 has 3 user categories ("stages") of GPIO pin muxing, as described in the GPIO Pin Muxing STAGE Definitions section. Figure 151 is a conceptual functional diagram of how the GPIO pins are muxed.

MORE AUXILIARY FEATURES

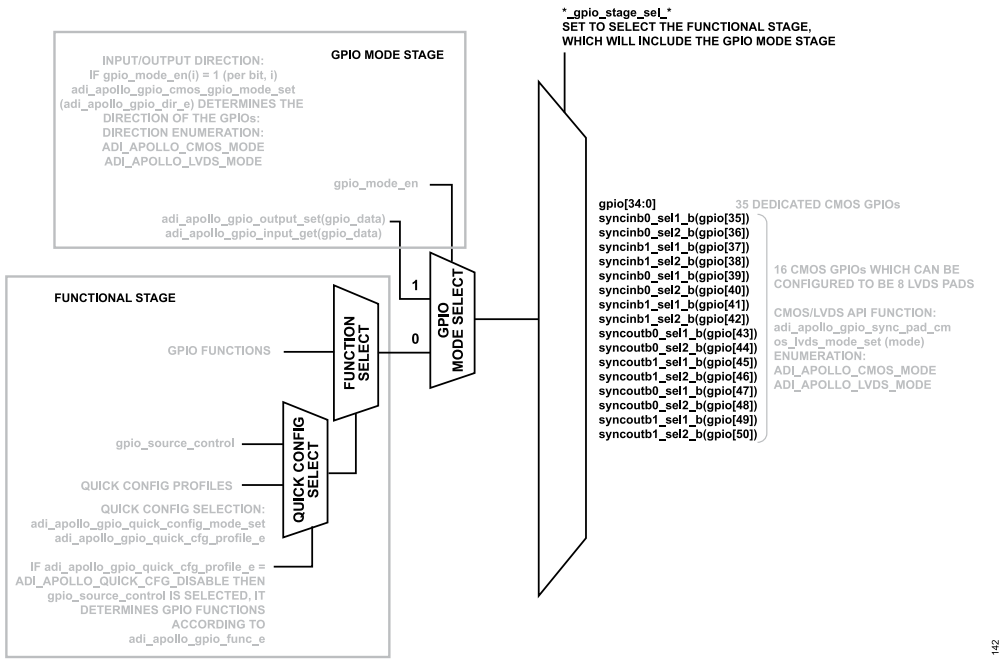

Figure 151. Conceptual GPIO Functional Diagram

GPIO Mapping

The AD9084 GPIO Pin mux has 51 CMOS GPIOs (with the LVDS SYNC pads configured as CMOS). Mapping is as described in the Table 121 below.

Table 121. GPIO Mapping

PAD	Mapping With LVDS Pads Configured as CMOS	GPIO Number
GPIO[34:0]	GPIO[34:0]	34-0
SYNCINB0_B	syncinb0_se1_b	35
	syncinb0_se2_b	36
SYNCINB1_B	syncinb1_se1_b	37
	syncinb1_se2_b	38
SYNCINB0_A	syncinb0_se1_a	39
	syncinb0_se2_a	40
SYNCINB1_A	syncinb1_se1_a	41
	syncinb1_se2_a	42
SYNCOUTB0_B	syncoutb0_se1_b	43
	syncoutb0_se2_b	44
SYNCOUTB1_B	syncoutb1_se1_b	45
	syncoutb1_se2_b	46
SYNCOUTB0_A	syncoutb0_se1_a	47
	syncoutb0_se2_a	48
SYNCOUTB1_A	syncoutb1_se1_a	49
	syncoutb1_se2_a	50

GPIO Pin Muxing STAGE Definitions

AD9084 has 2 user stages of GPIO pin muxing

MORE AUXILIARY FEATURES

- The first stage is the GPIO Mode stage, in which GPIO can be configured to function as an input or output. If it is configured as an output then data driven on that output pin (0 or 1) is sourced from a register bit.
- The second stage is the functional stage, in which user can configure the GPIOs to output selected internal functions; this stage has 1-to-any mapping

Table 122. GPIO Pin Muxing Stages

Signals	Mapping
CMOS Functional Signals	1-to-any

GPIO_MODE (CMOS)

Each CMOS GPIO can be configured to operate in GPIO MODE, meaning:

- the input or output direction can be defined
- which GPIO is enabled and active can be set
- Output Mode: GPIO outputs can be configured to drive 0 or 1
- Input Mode: the values applied to the GPIO inputs are stored in registers, which can be read back

GPIO_MODE API

The API function **adi_apollo_gpio_cmos_gpio_mode_set()** determines which GPIO pins are enabled and the direction of the enabled pins, taking the parameter gpio_dir as shown in Table 66.

The **adi_apollo_gpio_cmos_output_set()** API function determines the logic value (0 or 1) which will appear on the specified GPIO pin for GPIOs configured as OUTPUT using the parameter gpio_data.

The **adi_apollo_gpio_cmos_input_get()** API function allows the user to read back the stored data that has been input on the GPIO pins when the GPIOs are configured as INPUT.

The GPIO mode API parameters are described in Table 123. Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 123. GPIO Mode API Parameters

Parameter	Description	Enumerations or Settings	Comment
gpio_index	Select GPIO pad	Use GPIO number in Table 121	selects which GPIO pin is being configured, GPIO number 0 – 50, 0 – 34 are the CMOS GPIOs, 35 – 50 are for the LVDS SYNC pads configured in CMOS mode, see Table 121
gpio_dir	GPIO Direction (input or Output)	ADI_APOLLO_GPIO_DIR_INPUTADI_APOLLO_GPIO_DIR_OUTPUT	Configure as INPUTConfigure as OUTPUT
gpio_data	If GPIO_dir is OUTPUT: Determines the logic value (0 or 1)	If GPIO_dir is OUTPUT: Set to 0 or 1 depending based on desired logic value	If GPIO_dir is OUTPUT: For GPIOs configured as outputs, this function determines logic value (0 or 1) which will appear on the specified GPIO pin
	If GPIO_dir is INPUT: Points to location of stored logic value	If GPIO_dir is INPUT: Data from GPIO pin	If GPIO_dir is INPUT: pointer to location the data applied to the selected GPIO is stored

CMOS GPIOs (Functional Stage)

There are over 150 GPIO functions available in the design and are defined in the adi_apollo_gpio_types.h file of the device API. Each of these functions can be mapped to any of the 51 CMOS GPIOs. The API is used to select the function to be brought out to a particular GPIO pin. The enumeration name is used for this purpose; the function numbers are for reference and are not needed to select the desired function.

CMOS GPIOs API

The adi_apollo_gpio_cmos_func_mode_set() API function configures the GPIO & SYNC pads to be in CMOS Functional mode. The parameters for these functions are described in Table 124.

MORE AUXILIARY FEATURES

Setting the function takes care of the input/output mode, so in GPIO functional mode the direction (input/output) does not need to be explicitly set.

Table 124. GPIO Parameters of CMOS Mode

Parameter	Description	Enumerations or Settings	Comment
gpio_index	Select GPIO pad	Use GPIO number in Table 121	selects which GPIO pin is being configured, GPIO number 0 – 50, 0 – 34 are the CMOS GPIOs, 35 – 50 are for the LVDS SYNC pads configured in CMOS mode, see Table 121
func_num	CMOS function number	adi_apollo_gpio_func_e	Use enumeration names rather than function numbers

GPIO Quick Config

The AD9084 has 8 GPIO profiles as enumerated in Table 125. The profile is enabled by using the **adi_apollo_gpio_quick_config_mode_set()** function, which takes the input parameter ADI_APOLLO_QUICK_CFG_PROFILE_x, where x is the desired profile number from 1 to 8.The GPIO Quick Config parameters are stored in the **adi_apollo_gpio_cfg_t** structure which is pulled from the device profile. Example use of **adi_apollo_gpio_quick_config_mode_set()** can be found in ads10_apollo_ex_main/fullchip_hop.c. By default, The Apollo GPIO Quick Config Profile is set to profile 1.

Table 125. GPIO Quick Config Profile Enumeration and Description

Enumeration	Profile Description/Use Case	Device Config
ADI_APOLLO_QUICK_CFG_PROFILE_1	204B, Fast Detect, Hopping, Dynamic Reconfiguration & FFT Sniffer	4T4R and 8T8R
ADI_APOLLO_QUICK_CFG_PROFILE_2	204B, Hopping, Dynamic Reconfiguration, and SPI Main	4T4R and 8T8R
ADI_APOLLO_QUICK_CFG_PROFILE_3	204C, , Hopping, Dynamic Reconfiguration, and FFT Sniffer & SPI Main	8T8R
ADI_APOLLO_QUICK_CFG_PROFILE_4	204C, Dual clock, GP_IRQ, Fast Detect, Hopping, Dynamic Reconfiguration, & FFT Sniffer	4T4R and 8T8R
ADI_APOLLO_QUICK_CFG_PROFILE_5	204C, Dual clock, Fast Detect, Hopping, Dynamic Reconfiguration, & FFT Sniffer	4T4R and 8T8R
ADI_APOLLO_QUICK_CFG_PROFILE_6	204B, Hopping, Fast Detect, Reconfiguration,, SPI Main	4T4R and 8T8R
ADI_APOLLO_QUICK_CFG_PROFILE_7	204B, PA protection, Fast Detect, Hopping, Reconfiguration,, SPI Main	4T4R and 8T8R
ADI_APOLLO_QUICK_CFG_PROFILE_8	204B dual link per side, Fast Detect, Hopping, Dynamic Reconfiguration,	4T4R and 8T8R

Table 126. GPIO Quick Config Profiles Use Case Pin Assignments

Ball Name	GPIO Num	Profile 1	Profile 2	Profile 3	Profile 4	Profile 5	Profile 6	Profile 7	Profile 8
GPIO0	0	TxEN0(A)	TxEN0(A)	TxEn0	TxEN0(A)	TxEN0(A)	TxEN0(A)	TxEn0	TxEN0
GPIO1	1	TxEn1(B)	TxEn1(B)	TxEn1	TxEn1(B)	TxEn1(B)	TxEn1(B)	TxEn1	TxEn1
GPIO2	2	RxEN0(A)	RxEN0(A)	TxEN2	RxEN0(A)	RxEN0(A)	RxEN0(A)	TxEN2	RxEN0
GPIO3	3	RxEn1(B)	RxEn1(B)	TxEN3	RxEn1(B)	RxEn1(B)	RxEn1(B)	TxEN3	RxEn1
GPIO4	4	IRQ_JESD_Tx_A	IRQ_JESD_Tx_A	RxEN0	IRQ_JESD_Tx_A	IRQ_JESD_Tx_A	IRQ_JESD_Tx_A	RxEN0	IRQ_JESD_Tx_A
GPIO5	5	IRQ_JESD_Tx_B	IRQ_JESD_Tx_B	RxEn1	IRQ_JESD_Tx_B	IRQ_JESD_Tx_B	IRQ_JESD_Tx_B	RxEn1	IRQ_JESD_Tx_B
GPIO6	6	IRQ_JRx_DataReady_A	IRQ_JRx_DataReady_A	RxEN2	IRQ_JRx_DataReady_A	IRQ_JRx_DataReady_A	IRQ_JRx_DataReady_A	RxEN2	IRQ_JRx_DataReady_A
GPIO7	7	IRQ_JRx_DataReady_B	IRQ_JRx_DataReady_B	RxEN3	IRQ_JRx_DataReady_B	IRQ_JRx_DataReady_B	IRQ_JRx_DataReady_B	RxEN3	IRQ_JRx_DataReady_B
GPIO8	8	JESD_PLL_LOCK	JESD_PLL_LOCK	FREEZE	sync_irq_a	sync_irq_a	JESD_PLL_LOCK	IRQ_JESD_Tx_A	NVM Boot Done
GPIO9	9	PA_protect_Trigger_or_A	PA_protect_Trigger_or_A	Rx_Reconfig_done_a_and_b	sync_irq_b	sync_irq_b	all _IRQ_bad	IRQ_JESD_Tx_B	JESD_PLL_LOCK
GPIO10	10	PA_protect_Trigger_or_B	PA_protect_Trigger_or_B	Tx_Reconfig_done_a_and_b	JESD_PLL_LOCK	JESD_PLL_LOCK	ADC_FD0	IRQ_JRx_DataReady_A	ADC_FD0
GPIO11	11	ADC_FD0	FREEZE	all _IRQ_good	ADC_FD0	ADC_FD0	ADC_FD1	IRQ_JRx_DataReady_B	ADC_FD1

MORE AUXILIARY FEATURES

Table 126. GPIO Quick Config Profiles Use Case Pin Assignments (Continued)

Ball Name	GPIO Num	Profile 1	Profile 2	Profile 3	Profile 4	Profile 5	Profile 6	Profile 7	Profile 8
GPIO12	12	ADC_FD1	all _IRQ_bad	all _IRQ_bad	ADC_FD1	ADC_FD1	ADC_FD2	JESD_PLL_LOCK	ADC_FD2
GPIO13	13	ADC_FD2	Rx_Reconfig_done_a_and_b	ADC_FD0	ADC_FD2	ADC_FD2	ADC_FD3	ADC_FD0	ADC_FD3
GPIO14	14	ADC_FD3	Tx_Reconfig_done_a_and_b	ADC_FD1	ADC_FD3	ADC_FD3	profile_tx_rxn[0]	ADC_FD1	clock_deladj_0
GPIO15	15	clock_deladj_0	clock_deladj_0	ADC_FD2	profile_tx_rxn[0]	profile_tx_rxn[0]	profile_tx_rxn[1]	ADC_FD2	clock_delstr_0
GPIO16	16	clock_delstr_0	clock_delstr_0	ADC_FD3	profile_tx_rxn[1]	profile_tx_rxn[1]	profile[0]	ADC_FD3	all _IRQ_bad
GPIO17	17	profile_tx_rxn[0]	profile_tx_rxn[0]	ADC_FD4	profile_txrx_slice[0]	profile_txrx_slice[0]	profile[1]	I_smon_a[0]	profile_tx_rxn[0]
GPIO18	18	profile_tx_rxn[1]	profile_tx_rxn[1]	ADC_FD5	profile_txrx_slice[1]	profile_txrx_slice[1]	profile[2]	I_smon_a[1]	profile_tx_rxn[1]
GPIO19	19	profile_txrx_slice[0]	profile_txrx_slice[0]	ADC_FD6	profile_txrx_slice[2]	profile_txrx_slice[2]	profile[3]	I_smon_b[0]	profile_txrx_slice[0]
GPIO20	20	profile_txrx_slice[1]	profile_txrx_slice[1]	ADC_FD7	profile_txrx_BA	profile_txrx_BA	profile[4]	I_smon_b[1]	profile_txrx_slice[1]
GPIO21	21	profile_txrx_slice[2]	profile_txrx_slice[2]	profile_tx_rxn[0]	profile_fcn_sel[0]	profile_fcn_sel[0]	N/A	O_pa0 _en	profile_txrx_slice[2]
GPIO22	22	profile_txrx_BA	profile_txrx_BA	profile_tx_rxn[1]	profile_fcn_sel[1]	profile_fcn_sel[1]	N/A	O_pa1 _en	profile_txrx_BA
GPIO23	23	profile_fcn_sel[0]	profile_fcn_sel[0]	profile_fcn_sel[0]	profile_fcn_sel[2]	profile_fcn_sel[2]	N/A	O_pa2 _en	profile_fcn_sel[0]
GPIO24	24	profile_fcn_sel[1]	profile_fcn_sel[1]	profile_fcn_sel[1]	profile[0]	profile[0]	N/A	O_pa3 _en	profile_fcn_sel[1]
GPIO25	25	profile_fcn_sel[2]	profile_fcn_sel[2]	profile_fcn_sel[2]	profile[1]	profile[1]	N/A	O_pa4 _en	profile_fcn_sel[2]
GPIO26	26	profile[0]	profile[0]	profile[0]	profile[2]	profile[2]	N/A	O_pa5 _en	profile[0]
GPIO27	27	profile[1]	profile[1]	profile[1]	profile[3]	profile[3]	N/A	O_pa6 _en	profile[1]
GPIO28	28	profile[2]	profile[2]	profile[2]	profile[4]	profile[4]	N/A	O_pa7 _en	profile[2]
GPIO29	29	profile[3]	profile[3]	profile[3]	clock_deladj_0	clock_deladj_0	clock_deladj_0	clock_deladj_0	profile[3]
GPIO30	30	profile[4]	profile[4]	profile[4]	clock_delstr_0	clock_delstr_0	clock_delstr_0	clock_delstr_0	profile[4]
GPIO31	31	profile_fcn_sel[3]	profile_fcn_sel[3]	profile_fcn_sel[3]	profile_fcn_sel[3]	profile_fcn_sel[3]	TMU_Alarm	M_SPI_CLK	TMU_Alarm
GPIO32	32	TMU_Alarm	M_SPI_CLK	M_SPI_CLK	TMU_Alarm	TMU_Alarm	all _IRQ_good	M_SPI_SDI	gp_interrupt
GPIO33	33	sync_irq_c	M_SPI_SDI	M_SPI_SDI	Rx_Reconfig_done_a_and_b	all _IRQ_bad	sync_irq_c	M_SPI_SDO	sync_irq_c
GPIO34*	34	FREEZE	M_SPI_SDO	M_SPI_SDO	Tx_Reconfig_done_a_and_b	FREEZE	FREEZE	M_SPI_CS[0]	FREEZE
SYNCINB0_P_B	35	SYNCINB0_P_B	SYNCINB0_P_B	clock_deladj_0	all _IRQ_bad	fft_enable_0	SYNCINB0_P_B	SYNCINB0_P_B	SYNCINB0_P_B
SYNCINB0_N_B	36	SYNCINB0_N_B	SYNCINB0_N_B	clock_delstr_0	gp_interrupt	fft_hold_0	SYNCINB0_N_B	SYNCINB0_N_B	SYNCINB0_N_B
SYNCINB1_P_B	37	fft_enable_0	TRIGOUT_Rx_A	TRIGOUT_Rx_A	clock_deladj_1	clock_deladj_1	M_SPI_CLK	clock_deladj_1	SYNCINB1_P_B
SYNCINB1_N_B	38	fft_hold_0	TRIGOUT_Rx_B	TRIGOUT_Rx_B	clock_delstr_1	clock_delstr_1	M_SPI_SDI	clock_delstr_1	SYNCINB1_N_B
SYNCINB0_P_A	39	SYNCINB0_P_A	SYNCINB0_P_A	fft_enable_0	TRIGOUT_Rx_A	TRIGOUT_Rx_A	SYNCINB0_P_A	SYNCINB0_P_A	SYNCINB0_P_A
SYNCINB0_N_A	40	SYNCINB0_N_A	SYNCINB0_N_A	fft_hold_0	TRIGOUT_Rx_B	TRIGOUT_Rx_B	SYNCINB0_N_A	SYNCINB0_N_A	SYNCINB0_N_A
SYNCINB1_P_A	41	fft_done_0	M_SPI_CS[0]	M_SPI_CS[0]	fft_enable_0	fft_done_0	M_SPI_SDO	fft_enable_0	SYNCINB1_P_A
SYNCINB1_N_A	42	FW_IRQ_OUT	FW_IRQ_OUT	FW_IRQ_OUT	fft_hold_0	fft_enable_1	gp_interrupt	fft_hold_0	SYNCINB1_N_A

MORE AUXILIARY FEATURES

Table 126. GPIO Quick Config Profiles Use Case Pin Assignments (Continued)

Ball Name	GPIO Num	Profile 1	Profile 2	Profile 3	Profile 4	Profile 5	Profile 6	Profile 7	Profile 8
SYNCOUTB0_P_B	43	SYNCOUTB0_P_B	SYNCOUTB0_P_B	fft_done_0	fft_done_0	fft_hold_1	SYNCOUTB0_P_B	SYNCOUTB0_P_B	SYNCOUTB0_P_B
SYNCOUTB0_N_B	44	SYNCOUTB0_N_B	SYNCOUTB0_N_B	fft_done_1	fft_enable_1	fft_done_1	SYNCOUTB0_N_B	SYNCOUTB0_N_B	SYNCOUTB0_N_B
SYNCOUTB1_P_B	45	Rx_Reconfig_done_a_and_b	TRIGOUT_Tx_A	TRIGOUT_Tx_A	fft_hold_1	Rx_Reconfig_done_a_and_b	M_SPI_CS[0]	fft_done_0	SYNCOUTB1_P_B
SYNCOUTB1_N_B	46	Tx_Reconfig_done_a_and_b	TRIGOUT_Tx_B	TRIGOUT_Tx_B	fft_done_1	Tx_Reconfig_done_a_and_b	M_SPI_CS[1]	all _IRQ_bad	SYNCOUTB1_N_B
SYNCOUTB0_P_A	47	SYNCOUTB0_P_A	SYNCOUTB0_P_A	fft_enable_1	TRIGOUT_Tx_A	TRIGOUT_Tx_A	SYNCOUTB0_P_A	SYNCOUTB0_P_A	SYNCOUTB0_P_A
SYNCOUTB0_N_A	48	SYNCOUTB0_N_A	SYNCOUTB0_N_A	fft_hold_1	TRIGOUT_Tx_B	TRIGOUT_Tx_B	SYNCOUTB0_N_A	SYNCOUTB0_N_A	SYNCOUTB0_N_A
SYNCOUTB1_P_A	49	all _IRQ_bad	gp_interrupt	gp_interrupt	FW_IRQ_OUT	FW_IRQ_OUT	M_SPI_CS[2]	FW_IRQ_OUT	SYNCOUTB1_P_A
SYNCOUTB1_N_A	50	NVM Boot Done	NVM Boot Done	NVM Boot Done	NVM Boot Done	NVM Boot Done	M_SPI_CS[3]	NVM Boot Done	SYNCOUTB1_N_A

Table 126 Notes:

- The AD9084 GPIOs are all in CMOS mode by default. The user needs to configure LVDS-capable PADs to function in differential mode to operate in LVDS mode. Quick config does not configure the LVDS capable PAD to be in LVDS mode.)
- The fm_tmu_alarm bit needs to be set for GPIO34 to generate outputs as per above sheet. If the bit is not set, the te_fault signal is sent onto the GPIO34 irrespective of the quick config setting.
- For GPIO34, the fw_tmu_alarm bit needs to be set to obtain proper functioning. Otherwise, in all modes, te_fault is sent out.

Using GPIO's to Implement Time Division Duplexing (TDD)

Duplex communication links can be supported using the Apollo MxFE by implementing transmit enable (TxEN) and receive enable (RxEN) functions using GPIO pins. API functions including example code will be supported in a future revision of the Apollo MxFE API package. The Apollo MxFE user guide will fully document this feature once the API is available.

LVDS PADS (SYNC PADS)

LVDS PADs have 1-to-any mapping in functional mode and are configured to function as CMOS GPIOs by default. LVDS PADs can also be configured to function in LVDS mode. Once LVDS PADs are configured to function in differential mode, the corresponding 2 CMOS GPIOs won't be available. Each of the ~140 functions in the adi_apollo_gpio_types.h file can be routed to LVDS PADs

LVDS PADs API

The **adi_apollo_gpio_sync_pad_lvds_func_mode_set()** API function configures Functional Mode for the LVDS SYNC pads. This function takes one of 8 possible sync_pad input enumerations to select syncinb0/b1 and syncoutb0/b1 for each side (A/B) The enumerations are listed in Table 64 with ADI_APOLLO_ prepended. For example, ADI_APOLLO_SYNCINB0_B to use the syncinb0 pad on side B, or ADI_APOLLO_SYNCOUTB1_A to use the syncoutb1 pad on side A. The function argument is selected via the **adi_apollo_gpio_ func_e** enumeration as detailed in adi_apollo_gpio_types.h. For example: **adi_apollo_gpio_sync_pad_lvds_func_mode_set**(&device, ADI_APOLLO_SYNCINB0_B, ADI_APOLLO_FUNC_IRQ_CLOCK_WARNING) sets up syncinb0 pad on side B to output the clock drift warning IRQ.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

This was the CMOS GPIO debug section, which has been removed. I'm keeping this text for now to keep the comment history.

TEMPERATURE MONITORING UNIT (TMU)

The device contains a TMU that functions as a digital thermometer. The TMU is comprised of eight sensors placed at different chip locations. The on-die temperature value is measured and digitized through an ADC. The **adi_apollo_device_tmu_enable()** API function is used to

MORE AUXILIARY FEATURES

enable the TMU; while the **adi_apollo_device_tmu_get()** API function is used to readback the temperature at each sensor, as well as the minimum and maximum die temperatures across the chip at these sensor locations. API code that demonstrates an implementation example for retrieving the on-die temperature can be found in several of the examples in the example sub-directory of the API. These include the **fullchip_*.c** and others in the examples\ ads10_apollo_ex_main folder.

Refer to the Thermal Management Considerations section for information on thermal management on the PCB.

BUFFER MEMORY (BMEM)

As illustrated in Figure 152, there are blocks of buffer memory within 3 different functional blocks in the receiver data path. The HSDIN block as shown in the block diagram encapsulates the sample data buffers (BMEM0), sample crossbar (Mux1), and the functions performed on the sample data – fast detect (FD), signal monitor (SMON) and the spectrum sniffer which locates at the output of the ADCs. On the data path, there are two BMEMs in the CDDC and FDDC block respectively.

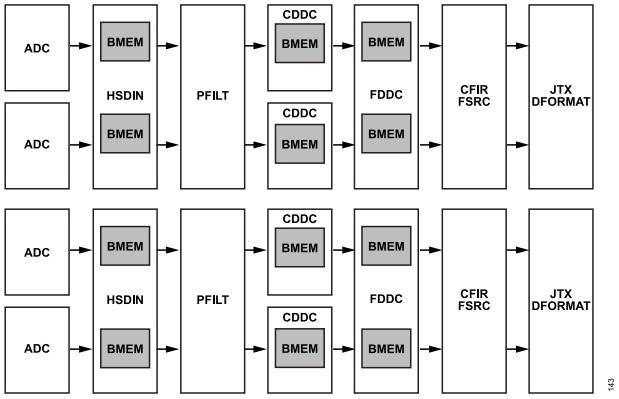

Figure 152. BMEM Locations in the Receive Data Path

The Figure 53 and Figure 54 shows more information about where these 3 BMEM locates at the data path relative to other blocks and the size of each BMEM block. Basically,

- The memory size on AD9084 is twice of the size of AD9088 at the same location,
- The BMEM shares the same configuration on the paired blocks (or channels) which requires the same decimation ratio, and provide the same path delay. For example,
  - In Figure 53, AD9084's BMEM2 at FDDC_0 and FDDC_1 (or FDDC_2 and FDDC_3) shares the same configuration so the decimation ratio on these two FDDC blocks is required to be the same, and the BMEM delay on these two paths are same.
  - Likewise, on AD9088's block diagram (Figure 54), ADC0 and ADC2 are the paired channels, so the BMEM0 on ADC0 and ADC2 has same configuration and provides the same delay. At the downstream of the data path, CDDC_0 and CDDC_2 are required to be configured with the same decimation ratio for BMEM1. The same restriction applies to FDDC_0, FDDC_4, FDDC_1 and FDDC_5 blocks (for BMEM2) as well.

Capture Mode

BMEM0 in HSDIN block supports the data capture which can be used to validate the data (or signals) sampled by ADC or for the debugging purposes. When configuring BMEM0 for this mode, once the capture starts, the data is continuously written to the memory until it is full or reaches to the end address which is configured through API. Then, the data can be read by calling API **adi_apollo_bmem_hsdin_capture_get()**. An example is provided in **examples\ads10_apollo_ex_main\rx_adc_bmem.c**.

Table 127 contains descriptions of API functions that are used to configure and interact with the BMEM capture mode.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in theapollo-sdk\pkg\production\doc folder.

MORE AUXILIARY FEATURES

Table 127. API Functions for BMEM0 Capture Mode

API function	Description
adi_apollo_bmem_hsdin_capture_config(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, adi_apollo_bmem_capture_t *config)	Configure BMEM0 in capture mode.
adi_apollo_bmem_hsdin_capture_run(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems)	Start the data capture.
adi_apollo_bmem_hsdin_capture_get(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, uint32_t data[], uint32_t length)	Retrieve the data from the BMEM buffer.

Delay Mode

The BMEM can be used as a delay block on the data path which basically supports two scenarios,

- **Sample Delay**: BMEM works as delay block to introduce a delay in the data path. **\examples\ads10_apollo_ex_main\rx_bmem_delay.c** provides an example to show how to use BMEM in this scenario.
- **Hopping Delay**: In this delay mode, the BMEM can be configured to quickly switch between four programmable delays. The delay switching happens on a selectable event. The available events are a trigger signal controlled by the trigger module, or a GPIO toggle event. An example is provided in **\examples\ads10_apollo_ex_main\lb0_bmem_delay_hop.c**.

Table 128 and Table 129 contain the API functions that are used to configure and interact with the BMEM delay mode. For more details refer to "apollo_api.chm". This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 128. API Functions for BMEM Sample Delay Mode

API function	Description
adi_apollo_bmem_hsdin_delay_sample_config(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, adi_apollo_bmem_delay_sample_t *config)	Configure BMEM sample delay mode at HSDIN block.
adi_apollo_bmem_cddc_delay_sample_config(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, adi_apollo_bmem_delay_sample_t *config)	Configure BMEM sample delay mode at CDDC block.
adi_apollo_bmem_fddc_delay_sample_config(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, adi_apollo_bmem_delay_sample_t *config)	Configure BMEM sample delay mode at FDDC block.
adi_apollo_bmem_fddc_delay_sample_set(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, uint16_t sample_delay)	Sets BMEM sample delay at FDDC block.
adi_apollo_bmem_cddc_delay_sample_set(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, uint16_t sample_delay)	Sets BMEM sample delay at CDDC block.
adi_apollo_bmem_hsdin_delay_sample_set(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, uint16_t sample_delay)	Sets BMEM sample delay at HSDIN block.
adi_apollo_bmem_hsdin_delay_start(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems)	Starts BMEM sample delay at HSDIN block.
adi_apollo_bmem_cddc_delay_start(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems)	Starts BMEM sample delay at CDDC block.
adi_apollo_bmem_fddc_delay_start(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems)	Starts BMEM sample delay at FDDC block.

Table 129. API Functions for BMEM Hopping Delay

API function	Description
adi_apollo_bmem_hsdin_delay_hop_config(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, adi_apollo_bmem_delay_hop_t *config)	Configure BMEM hopping delay mode at HSDIN block.
adi_apollo_bmem_hsdin_delay_hop_set(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, uint16_t sample_delay[], uint32_t sample_delay_length)	Sets BMEM delay at HSDIN block for the 4 hopping profiles.
adi_apollo_bmem_cddc_delay_hop_config(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, adi_apollo_bmem_delay_hop_t *config)	Configure BMEM hopping delay mode at CDDC block.

MORE AUXILIARY FEATURES

Table 129. API Functions for BMEM Hopping Delay (Continued)

API function	Description
adi_apollo_bmem_cddc_delay_hop_set(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, uint16_t sample_delay[], uint32_t sample_delay_length)	Sets BMEM delay at CDDC block for the 4 hopping profiles.
adi_apollo_bmem_fddc_delay_hop_config(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, adi_apollo_bmem_delay_hop_t *config)	Configure BMEM hopping delay mode at FDDC block.
adi_apollo_bmem_fddc_delay_hop_set(adi_apollo_device_t *device, adi_apollo_blk_sel_t bmems, uint16_t sample_delay[], uint32_t sample_delay_length)	Sets BMEM delay at FDDC block for the 4 hopping profiles.

AWG Mode

In AWG mode, the user can load a known data set (or waveform) into the BMEM which then injects this data into the data path at the downstream instead of the data at the input of the BMEM. This AWG mode is configured in several examples in the example folder **examples\ads10_apollo_ex_main**, such as **rx_bmem_cfir.c**, **rx_bmem_ddc.c**, and **rx_bmem_pfilt.c**. A similar procedure shown below are used in these examples to configure the AWG mode.

1. Configure the BMEM in AWG mode by calling **adi_ads10_apollo_ex_bmem_awg_config()**.
2. Write a sinusoid into the BMEM's Buffer by calling **adi_ads10_apollo_ex_bmem_awg_tone_write()**, this function calls **adi_apollo_bmem_hsdin_awg_stop()** to stop any current AWG playing before to load a new waveform to the memory.
3. Then, it calls **adi_apollo_bmem_hsdin_awg_sample_write()** to load the sinusoid into the BMEM.
4. At the end, it calls **adi_apollo_bmem_hsdin_awg_start()** to start a new AWG play.

MORE APPLICATIONS INFORMATION

SYSTEM CLOCKING SOLUTION

ADF4382 - Clock to Apollo MxFE

The ADF4382 can provide up to 20GHz clock to the Apollo MxFE to support direct clocking of the Apollo MxFE converters. The ADF4382 also has a 2-wire “clock alignment” interface that can be connected to 2 of the Apollo MxFE GPIO pins for clock alignment commands from the Apollo MxFE that are based on SYSREF alignment information gathered by Apollo’s MCS FW Tracking Calibration. This allows sub-clock-cycle adjustments of Apollo’s internal SYSREF. The API example code located at **examples\ads10_apollo_ex_main\mcs_cal.c** configures both the ADF4382 and Apollo’s internal MCS firmware to perform a foreground and optional background clock tracking algorithm. The algorithm runs within the Apollo MxFE and compares the phase of Apollo’s internal SYSREF to Apollo’s external SYSREF. Appropriate adjustments to the ADF4382 are made through the 2-wire interface until Apollo’s internal and external SYSREF are aligned. A background routine can be called to maintain this alignment as temperature changes in the system. The MCS programming flow is described in detail in the TDC and Firmware-based SYSREF Alignment sub-section of the Device Synchronization section.

The AD9084/88-FMCx-EBZ boards provide an example clocking solution with the ADF4382 directly clocking Apollo. The clock tree uses an LTC6955 for reference clock distribution as well as HMC7044 to provide an FPGA reference clock for the JESD204B/C PHY, and phase adjustable SYSREF to the Apollo MxFE and the carrier board FPGA. More information is available in the Apollo MxFE® Evaluation Board User Guide.

A block diagram of this clocking solution, scaled to a system with four Apollo MxFE’s is illustrated in Figure 153. Note that additional HMC7044’s may be needed to provide reference clocks to the FPGA processing the JESD204C data.

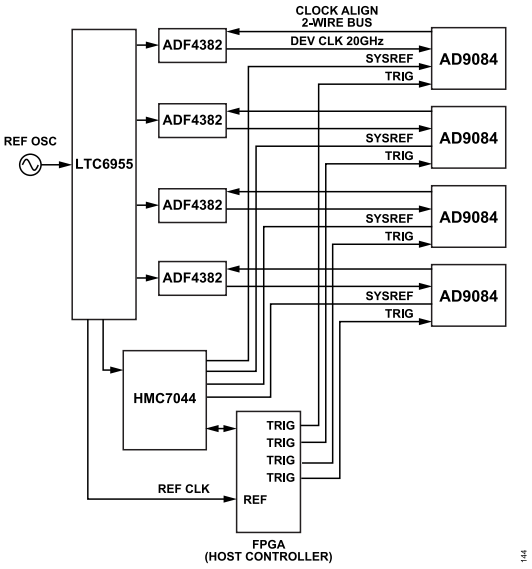

Figure 153. Scalable Apollo MxFE clocking Solution with ADF4382, LTC6955, and HMC7044

ADF4030 - Bi-directional SYSREF to Apollo MxFE

The ADF4030 provides for 10 bidirectional synchronized clock (BSYNC) channels and accepts a reference clock input (REFIN) signal as a frequency reference for generating an output clock on any BSYNC channels that are configured as an output. The hallmark feature of the ADF4030 is the ability to time align the clock edges of any one or more BSYNC channels with respect to the BSYNC channel selected as the reference BSYNC channel. These BSYNC channels can be used for SYSREF generation in JESD204 subclass 1 operation. The ADF4030 is well adapted for multiple connections with other ADF4030 devices for synchronizing clock signals in a system. Each BSYNC is bidirectional, allowing for reversing the direction of the clock signal to measure the propagation delay of the transmission medium. Round trip constructions that use replica paths are also supported. The bidirectional nature of the round trip delay measurement greatly reduces the error in determining the propagation delay through the BSYNC transmission medium as compared to using a replica path. This feature makes the ADF4030 capable to time align the clock edges of BSYNC channels across multiple ADF4030 devices, independent of the tree or cascade architecture in which the ADF4030 system is designed.

MORE APPLICATIONS INFORMATION

The Apollo MxFE includes a bi-directional SYSREF that can be paired with the ADF4030 clock chip to facilitate measurements of SYSREF trace delays. Figure 154 is a high-level block diagram of a scalable system using multiple AD9084 devices with direct clocking from ADF4382 devices. The bi-directional BSYNC (or SYSREF) from each ADF4030 is connected to one or more AD9084 devices, depending on how many AD9084 devices need to be synchronized in the system.

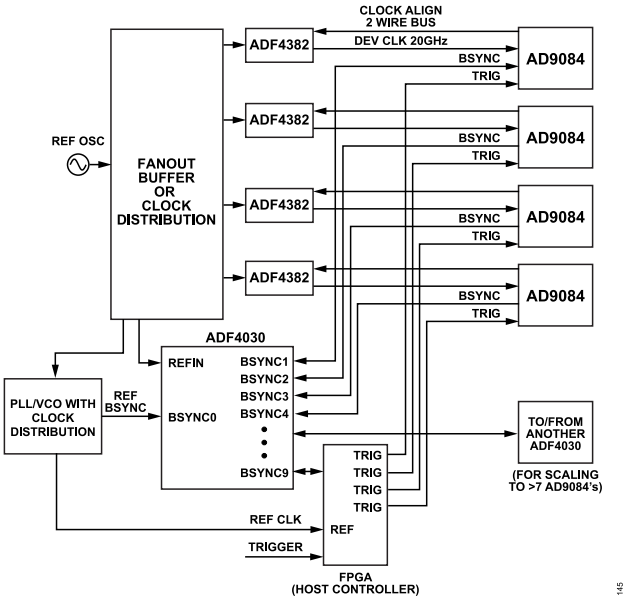

Figure 154. Scalable Apollo MxFE clocking Solution with ADF4030 and ADF4382

An API example in **examples\ads10_apollo_ex_main\bsync_tof.c** can be used to take these measurements, apply proper delays to the ADF4030, and align the SYSREF signals of multiple Apollo MxFE devices to within ± 10ps. The procedure is shown as below,

- Configure AD9084 devices as single clock mode which uses BSYNC channel 5 from ADF4030 as SYSREF input.
- Boot up ADF4030 and configure the reference frequency by calling **adi_ads10_apollo_ex_adf4030_startup()**.
- BSYNC setup for SYSREF generation,
  - Call **adi_ads10_apollo_ex_adf4030_bsync_input_set()** to configure BSYNC channel 0 as SYSREF input,
  - Call **adi_ads10_apollo_ex_adf4030_bsync_output_set()** to configure BSYNC channel 5 to output SYSREF to Apollo, and BSYNC channel 8 to FPGA,
  - Align all BSYNC output channels with BSYNC input channel 0 by calling **adi_ads10_apollo_ex_adf4030_align_bsync_out()**.
- Wait for system temperature to stabilize.
- Call **adi_ads10_apollo_ex_mcs_init_cal_setup()** to configure each ADF4382/AD9084 pair.
- Call **adi_ads10_apollo_ex_mcs_adf4030_apollo_path_delay_measurement()** to perform BSYNC Time-of-Flight (TOF) with ADF4030 to measure the path delays between ADF4030 and AD9084 SYSREF input pin.
- TOF calibration applies offset to correct the path delay by calling **adi_ads10_apollo_ex_mcs_adf4030_apollo_path_delay_offset()**.
- Call **adi_ads10_apollo_ex_mcs_adf4030_fpga_path_delay_measurement()** to perform BSYNC Time-of-Flight (TOF) with ADF4030 to measure the path delays between ADF4030 and FPGA SYSREF input pin and then call **adi_ads10_apollo_ex_mcs_adf4030_fpga_path_delay_offset()** to correct the delay.
- Call **adi_ads10_apollo_ex_adf4030_align_bsync_out()** to apply the delay corrections measured above to the output channels.
- Until now, the SYSREFs to AD9084 and FPGA input pins are well aligned, the next step is to run the MCS Init Calibration to align the internal and external SYSREF for AD9084 by calling **adi_apollo_mcs_cal_init_run()**.

This alignment procedure can be extended to synchronize multiple AD9084 devices by repeating the TOF measurement and delay offset correction as well as the alignment between ADF4030 if a system needs more channels and multiple ADF4030 devices.

MORE APPLICATIONS INFORMATION

TIME DIVISION DUPLEXING (TDD)

Duplex communication links can be supported using the Apollo MxFE by implementing transmit enable (TxEN) and receive enable (RxEN) functions using GPIO pins. API functions including example code will be supported in a future revision of the Apollo MxFE API package.

PCB LAYOUT AND DESIGN CONSIDERATIONS

CAD Symbol, Package Pinout and Unused Balls

The CAD symbols and package pinout are available for each device on their respective product page in the design resources section. For reference, The Ball out for the AD0984DF and AD9088 packages are provided in Figure 155 and Figure 156 respectively. The package pinout is also included in their respective datasheets. Figure 157 provides the package outline drawing for both products. Note, leave any ball that is labeled as DNC open and leave any ball labeled as NC open or connected to a surrounding GND pin

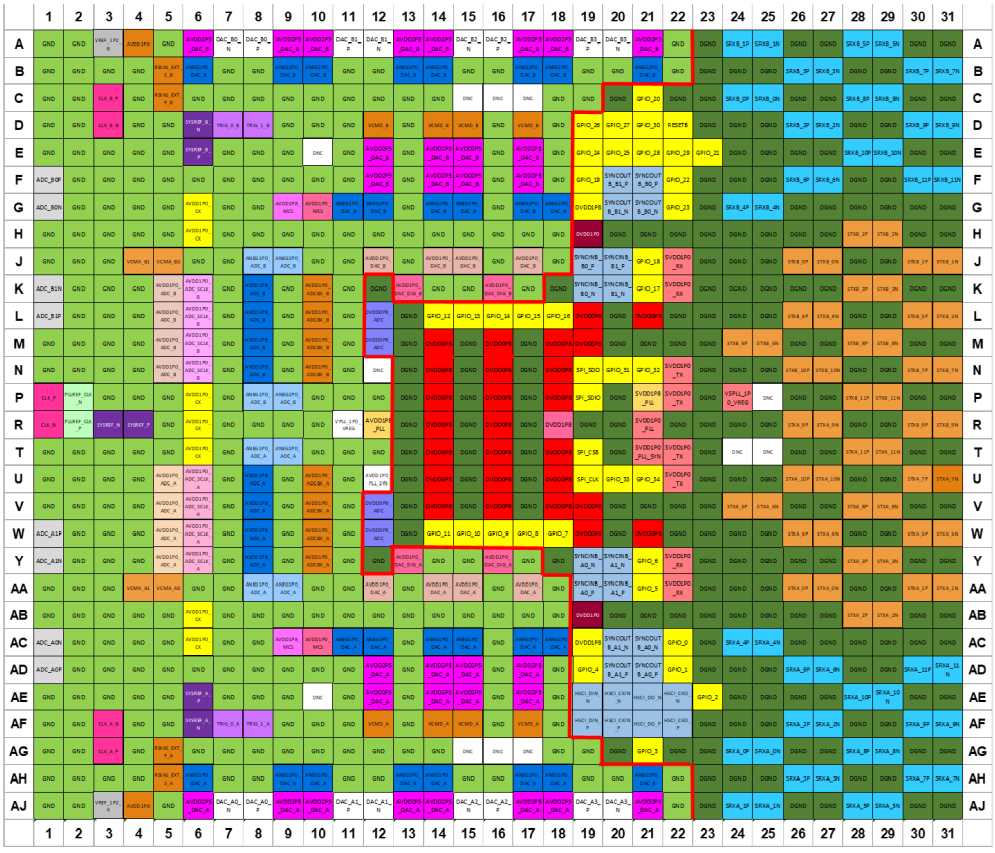

Figure 155. AD9084DF Package Pinout (4T4R Differential ADC)

MORE APPLICATIONS INFORMATION

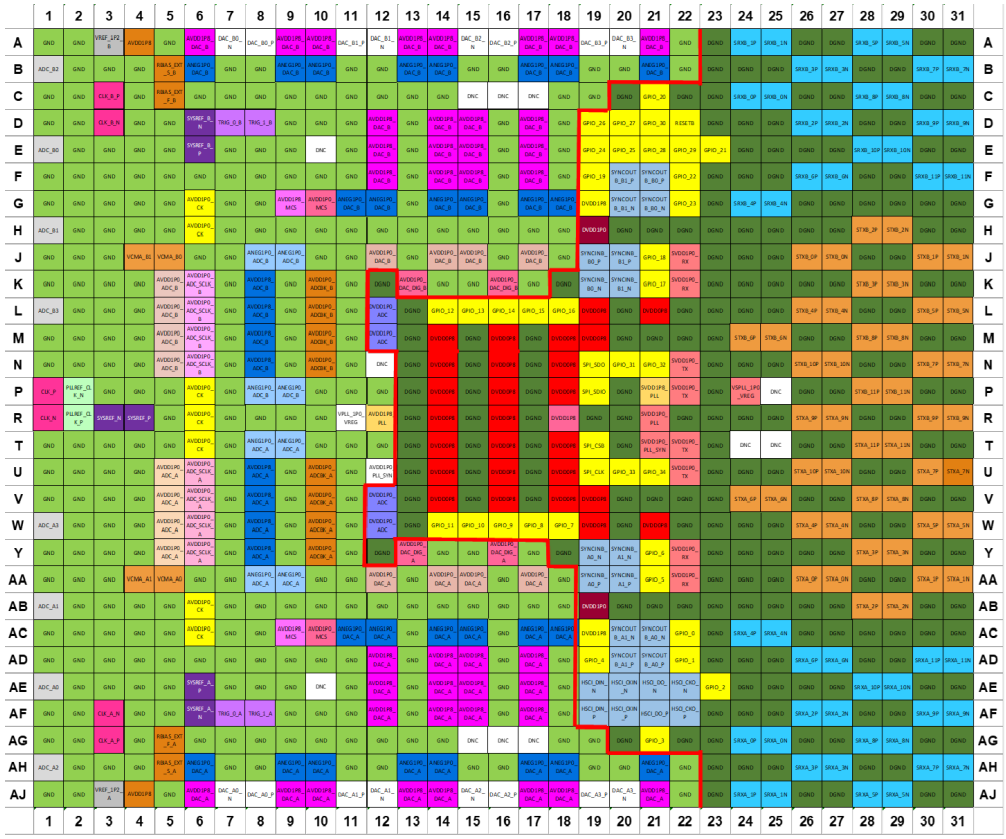

Figure 156. AD9088 Package Pinout (8T8R SE)

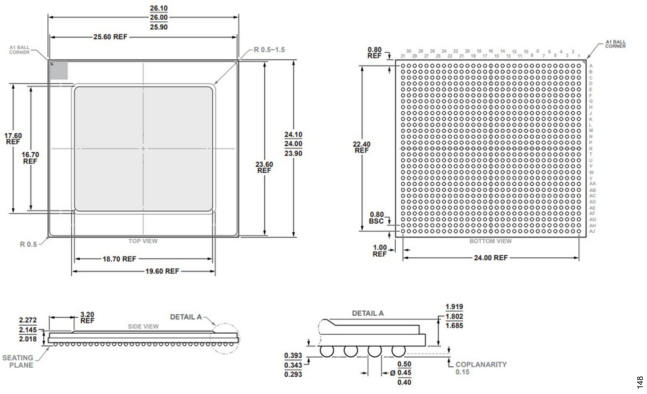

Figure 157. 899-Ball (29x31) Ball Grid Array, Thermally Enhanced [BGA_ED] (BP-899-1)
Dimensions shown in millimeters

Grounding (GND) Strategy

In a mixed-signal design where both digital and analog circuitry is present, care must be taken to avoid digital noise from coupling onto sensitive analog circuitry. Apollo MxFE's ground planes are split into an analog ground (GND) and a digital ground (DGND), which directs the "noisy" digital supply return currents away from the "clean" analog ground domain.

The split between GND and DGND domains extends from the IC to the BGA balls, and is indicated with the thick red line in Figure 141 and Figure 142 . Depending on the system design and its component placement, it may be beneficial to also extend this split on the PCB, joined into a single ground plane at a specific location some distance away from the Apollo MxFE. An example of this approach was implemented on

MORE APPLICATIONS INFORMATION

the AD9084-FMCA-EBZ reference design, where the split is extended to the edges of the PCB and joined together in inner layers to form a single ground plane on Layers 12 and 13 as shown in Figure 158. For more information, please reference the AD9084-FMCA-EBZ layout.

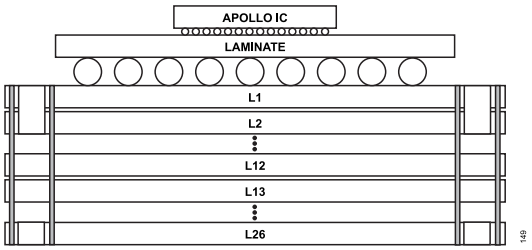

Figure 158. Cross-sectional view of the AD9084-FMCA-EBZ PCB, showing the Ground split on all layers except Layers 12 and 13, joined by stitching vias

The location of the split is design-specific, and primarily depends on the location of the main power supply that provides the input to the power modules that supply the Apollo MxFE digital domains, relative to the locations of analog domains and other sensitive analog circuitry: the return currents to the input and output bulk capacitors should be guided away from sensitive analog.

ADC Input Layout

To simplify system PCB design and provide the flattest possible frequency response, the input impedance of the Apollo ADC was designed to be as close as possible to 50ohms differential. However, due to parasitics on the silicon die and the package traces, the ADC input impedance does vary significantly over frequency. As illustrated in Figure 159, this impedance variation creates a complex matching problem between the differential output impedance of the balun or RxVGA that is driving the Apollo ADC input, the PCB Trace traces, and the load impedance of the Apollo ADC..

The Apollo MxFE ADI evaluation boards use a wideband 0.1-20GHz balun that is nominally a 2:1 transformer. As shown in Figure 3, ADI found that the best option for achieving a good wideband match between the balun output and the Apollo ADC input, is to use a resistive matching network that transforms the 50ohm differential input impedance of the ADC, into a 100ohm differential load impedance for the balun.

For other applications where a 50ohm differential balun or RxVGA is available, the resistive matching pad is not required. The recommendation would be to use a 50ohm differential PCB trace to connect the balun or RxVGA to the Apollo ADC inputs, and to keep the PCB traces as short as possible.

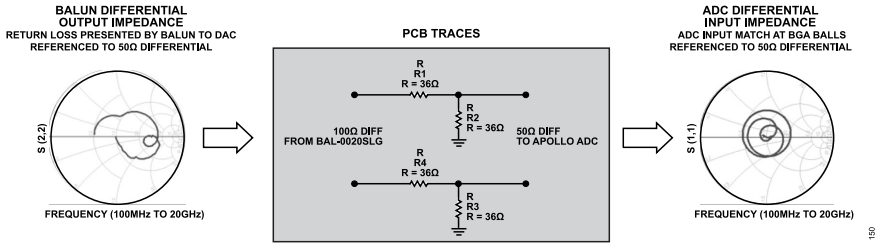

Figure 159. Complex Impedance Matching Problem between ADC Input, PCB Traces, and output Impedance of Balun or RxVGA

The above are good general guidelines for the design of the ADC input network, but in order to allow users to optimize their system board design, ADI has created RFIO models that can simulate the Apollo ADC response vs frequency with a specific PCB design and Balun/RxVGA.

The models are a set of Touchstone .s4p files that that can be connected to models for the PCB traces and the balun/RxVGA. By sensing the voltage delivered to the track and hold portion of the ADC, it is possible to accurately simulate the frequency response of the signal chain vs frequency. Figure 160 provides the simulated and measured Pout vs frequency on the ADI Evaluation board.

MORE APPLICATIONS INFORMATION

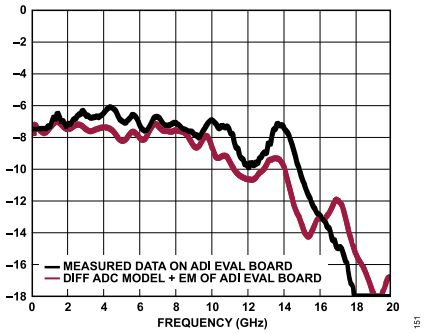

Figure 160. Simulated and Measured ADC response vs frequency on ADI Evaluation Board

The Apollo ADC RFIO models can be used to optimize the PCB design and Balun selection over a particular operating frequency range. The models are available by contacting the Apollo MxFE sales or FAE teams.

DAC Output Layout

To simplify system PCB design and provide flattest possible frequency response, the output impedance of the Apollo DAC was also designed to be as close as possible to 50ohms differential. However, due to parasitics on the silicon die and the package traces, the DAC output impedance does vary significantly over frequency. As illustrated in Figure 161, this impedance variation creates a complex matching problem between the DAC output impedance, PCB Trace impedance, and the load impedance of the balun or transmitter variable gain amplifier (TxVGA) that the DAC is driving.

In general, the Apollo DAC will provide the maximum Pout and flattest frequency response when 50ohm differential PCB traces are used to connect it to a balun or TxVGA that presents a 50ohm differential load. To minimize the period of any VSWR interactions between Apollo MxFE and the balun/TxVGA, it is recommended to keep the PCB traces as short as possible.

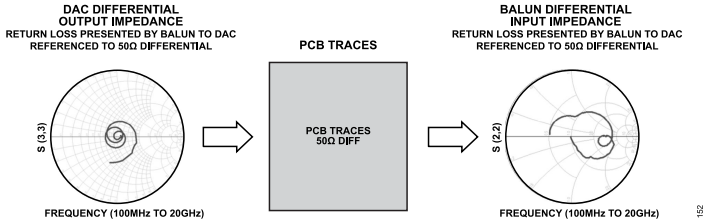

Figure 161. Complex Impedance Matching Problem between DAC Output, PCB Traces, and Input Impedance of Balun or TxVGA

The above are good general guidelines for the design of the DAC output network, but in order to allow users to optimize their system board design, ADI has created RFIO models that can simulate DAC Output power vs frequency with a specific PCB design and Balun/TxVGA.

The models are a set of Touchstone .s4p files that that are driven with ideal current sources and connected to models for PCB traces and the balun/TxVGA. By sensing the voltage and currents at the balun/TxVGA output, it is possible to accurately simulate the output power of the signal chain vs frequency. Figure 162 provides the simulated and measured Pout vs frequency on the ADI Evaluation board, with and without the INVSINC enabled.

MORE APPLICATIONS INFORMATION

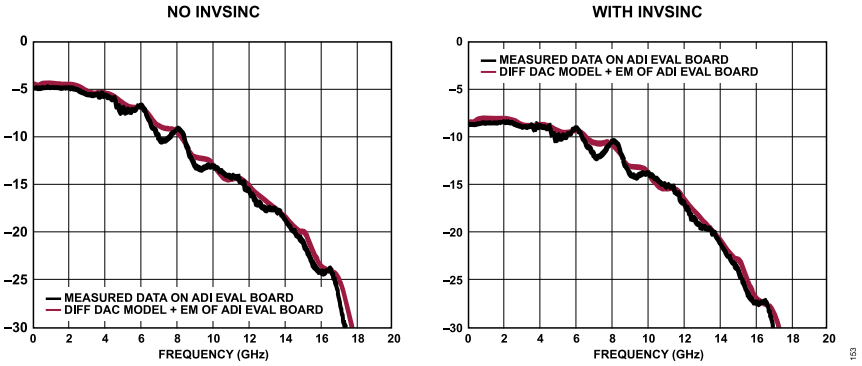

Figure 162. Simulated and Measured DAC Output Power vs Frequency on ADI Evaluation Board

The Apollo DAC RFIO models can be used to optimize the PCB design and Balun selection over a particular operating frequency range. The models are available by contacting the Apollo MxFE sales or FAE teams.

The Apollo DAC outputs require a connection to a +1.8V supply. The 1.8V supply can be provided by a good quality wire wound inductor,. For some baluns, it is also possible to “float” the balun by connecting it’s GND connections to 1.8V, rather than 0V. The balun then provides the required 1.8V bias to the DAC Outputs. Whether a balun or inductor is used to provide the 1.8V to the DAC outputs, care must be taken to transition the RF signal from the 1.8V reference in the package to the 0V reference in the PCB.

Clock Input Layout

Like the Apollo ADC and DAC, the impedance of the Apollo clock input (CLK_X_N/P) was designed to be as close as possible to 50ohms differential. However, again like the ADC and DAC, the parasitics on the silicon die and the package traces cause the CLK_X_N/P input impedance to vary significantly over frequency. As illustrated in Figure 163, this impedance variation creates a complex matching problem between the Balun or clock Driver output impedance, the PCB Trace impedance, and the load impedance of the Apollo clock.

In general, the maximum voltage will be delivered to the Apollo clock when it is driven by a balun or clock driver with a 50ohm output impedance. 50ohm differential PCB traces can then be used to connect to the 50ohm differential input of the Apollo clock. To minimize the period of any VSWR interactions between Apollo MxFE and the balun/TxVGA, it is recommended to keep the PCB traces as short as possible.

For baluns or clock drivers that require a 100ohm differential load impedance(like the ADF4832), it is possible to use a simple quarter wave transformer to convert the 50ohm differential impedance of the Apollo clock into the required 100ohm differential load. Because only a single clock frequency is generally required for a particular application, the quarter wave transformer can be designed to operate at a particular clock frequency, for example 14GHz or 20GHz.

The introduction of the quarter wave transformer does make the balun/clock Driver-to-Apollo clock interface more sensitive to PCB trace impedances and any required SMT components. To maximize the voltage delivered to the Apollo clock at a particular frequency, it is necessary to create a model that includes the PCB traces and SMT components and run simulations to ensure that the quarter wave transformer is behaving as expected at the desired clock frequency.

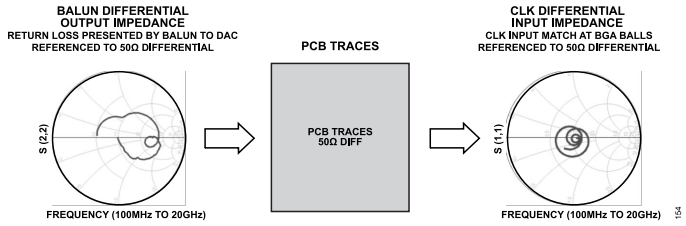

Figure 163. Complex Impedance Matching Problem between Clock Input, PCB Traces, and Output Impedance of Balun or TxVGA

The above are good general guidelines for the design of the CLK_X_N/P input network, but in order to allow users to optimize their system board design, ADI has created RFIO models that can simulate the voltage delivered to the Apollo clock vs frequency with a specific PCB design and balun/clock driver.

MORE APPLICATIONS INFORMATION

The models are a set of Touchstone .s4p files that that can be connected to models for the balun/clock Driver and the PCB traces. By sensing the voltage delivered to the clock input buffer on the silicon die, it is possible to optimize the system board design to provide the required voltage to the Apollo clock at the required frequency. The models are available by contacting the Apollo sales or FAE teams.

HSCI Layout

HSCI is a high-speed digital interface that can run up to 1.6Gbps with a maximum sample clock rate of 800MHz, using LVDS logic and a nominal 100ohm impedance. It is a substitute for the SPI interface which allows faster boot and configuration times. All HSCI signals should be routed as 100ohm differential pairs, DC-coupled. The Apollo MxFE provides on-chip termination of 100ohm.

To minimize clock-to-data skew and meet timing requirements, the clock and data lines should be preferably routed next to each other and pass through the same number of layers and vias on the PCB.

To minimize P-N skew between the two legs of the differential pair, a good target is a skew of less than 100pS, which equates to a single step of the clock-to-data delay adjust circuit. On a typical PCB with a dielectric constant of 4, this translate to a skew of less 590 milli-inches between the P and N branches.

Minimize any RF stubs in the layout (either due to vias or traces) to a critical length of less than 1/10th of the wavelength at the maximum operating frequency. For an 800MHz clock and typical PCB with a dielectric constant of 4 the critical length is around 740 milli-inches.

PCB Material and Stack Up Selection

The evaluation board PCB stack up shown in Figure 164 consists of 26-layers. The actual stackup of an Apollo MxFE system board might be different from the evaluation board, but the underlying principles still apply the same. For the RF analog (ADC inputs, DAC outputs, CLK inputs) and JESD204B/C SERDES traces, a mix of microstrips and striplines are employed. The RF traces use Isola's I-Tera MT-40 series of dielectric with a low Dissipation Factor near 0.0026 to ensure minimal insertion loss. The inner cores use Isola's 370HR material to reduce overall cost. Microvias are employed for signal transitions between the top three and bottom three layers. Back-drills are employed wherever possible to minimize the impact of via stubs. The stackup also uses three cores that use 2oz. thick copper. This is done to aid power handling of the supply rails (like DVDD0P8) that can potentially carry very high current. A via-in-pad approach is used to simplify access to inner layers on the evaluation board and enable decoupling capacitors on the back side of the PCB (under the device). A via-outside-pad approach can also be considered in instances where many of the digital and SERDES pins remain unused.

MORE APPLICATIONS INFORMATION

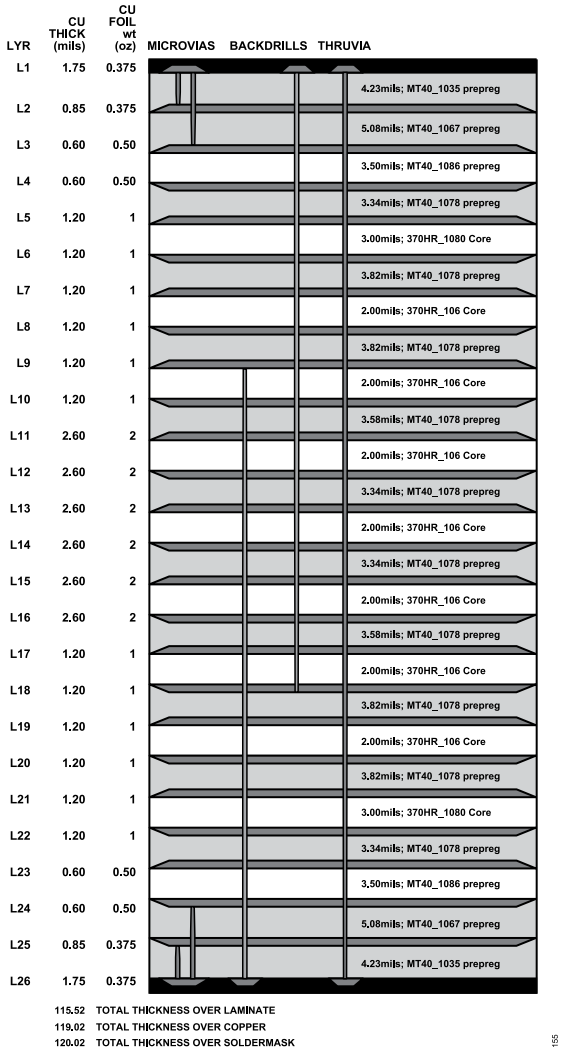

Figure 164. Evaluation Board PCB Stack Up

Component Placement and Routing Priorities

Signals With Highest Routing Priority

Figure 165 shows the top and bottom side of the PCB layout used on the AD9084-FMCA-EBZ, AD9084-FMCB-EBZ and AD9084-FMCC-EBZ evaluation boards, where RF and high speed digital signals have the highest priority. The DAC and ADC networks are routed on the top of the PCB. To minimize coupling between the CLK and DAC/ADC traces, the CLK signals are routed on the bottom of the PCB. ADC and DAC interfaces also makes it difficult to support all RF analog traces on the top side if isolation is of a lower priority. The ADI evaluation boards use the Marki BAL-0020SLG 0.01-20GHz wide band baluns for both the DAC and ADC networks. The BAL-0020SLG allow evaluation of the DAC and ADC over a wide frequency range and also provide excellent amplitude and phase balance. The user may select a narrower band balun design for their particular frequency.

MORE APPLICATIONS INFORMATION

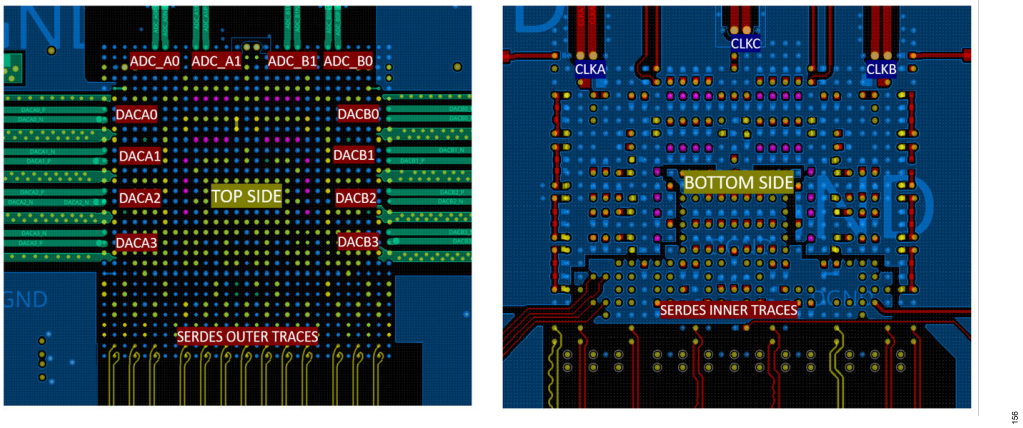

Figure 165. Routing Direction of Critical RF Analog and SERDES Signals

If possible, it is recommended to route the PCB SERDES traces on metal layers that are surrounded by lower loss dielectric materials. This will improve SERDES performance.

As shown in Figure 156, the SERDES Balls are located in 4 separate groups: rows 31 and 30, rows 29 and 28, rows 27 and 26, and rows 25 and 24. It is recommended to route PCB traces for rows 31 and 30 on PCB layer 1, as this provides an easy escape under the package. On the ADI evaluation boards, the PCB traces for rows 29 and 28 were routed on PCB layer 3, which GND on layer 2 and 4. Again, this provides a relatively easy escape for the SERDES traces from under the package.

In order to route the remaining SERDES traces between low loss dielectric materials, it is recommended to take advantage of the symmetry of most PCB stackups. The high performance dielectric materials on the top layers of the board are usually also used on the bottom layers. Consequently, the ADI eval board routes PCB traces for rows 25 and 24 on the bottom layer of the PCB. The SERDES traces for rows 27 and 26 are routed two layers from the bottom metal layer, with GNDS above and below it.

Signals With Lowest Routing Priority

This section provides information on the signals that are of the lowest signal routing priority. These signals can only be routed when all critical signal routes are complete to ensure that these signals do not interfere with the critical component placement and routing.

The SYSREF signal is an optional input control signal used to configure the device for JESD204 subclass 1 operation enabling deterministic latency and/or multichip phase alignment. This signal typically originates from a clock distribution IC that provides an LVDS or PECL signal type that interfaces to the devise SYSREF_x_P/N pins, which provides a 100 Ω on-chip termination. Use loosely coupled, 100 Ω strip line traces for this signal that can be routed on the inner layer.

The SYNCOUTB_xx and SYNCINB_xx control signals are used for JESD204B/C link establishment. Like the SYSREF signal, these signals can be configured for a 100 Ω differential interface or a single-ended CMOS interface. When configured for a differential 100 Ω interface, use loosely coupled 100 Ω strip line traces to the host processor. When configuring for a single-ended interface, use the positive polarity pin and leave the negative polarity pin floating. Route all CMOS input and output signals such as those supporting the SPI interface, as well as optional GPIO and control and status flag signals on the inner PCB layers. Route these signals away from the analog section.

RF AND JESD204B/C SERDES TRANSMISSION LINE LAYOUT

The layout recommendations in this section are intended for high frequency signals. Acceptable PCB designs include the following:

- Match the system board design as close as possible to the evaluation board design files available on the product page.
- Be attentive to power distribution and power ground return methodology to avoid coupling onto signal traces or the ground reference plane. In general, do not run high speed digital lines near dc power distribution routes or RF line routes.
- Use microstrip or coplanar waveguides (CPWG) on the top side of the PCB for transmission lines whenever possible. These structures do not require via structures that cause additional impedance discontinuities that vary across frequency. However, isolation between RF analog inputs/ outputs and space constraints associated with using all JESD204B/C lanes require using the back side of the PCB.

MORE APPLICATIONS INFORMATION

- 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 50 Ω for the DAC outputs, ADC inputs and clock inputs. Differential lines from the balun to the ADC and DAC input/output balls must be as short as possible. The SERDES lanes are microstrip lines designed for a Z_{DIFF} of 100 Ω.
- In most cases, the required board artwork stack up is different than the evaluation board stack up. Optimization of RF transmission lines specific to the desired board environment is essential to the design and layout process.

JESD204B/C SERDES Trace Routing Recommendations

The AD9084 has a total of 48 SERDES pairs, 24 for the JESD204B/C Tx (from the ADCs) and 24 for the JESD204B/C Rx (to the DACs). The SERDES pins are arranged within the section from pins A24 to AJ31 in the package. Refer to the package pinout section in the datasheet for more information. Routing is not symmetric because each differential pair is placed on a separate column. When deciding on the layout of these traces, consider the following guidelines:

- The positive and negative traces of a single trace pair must be of equal length to minimize duty cycle distortion (DCD).
- All SERDES lane traces must be designed for nominally 100 Ω differential impedance.
- Route the differential pairs on a single plane and use a solid ground plane as a reference on the layers above and/or below the traces. Ensure that reference planes for impedance-controlled signals are not segmented or broken for the entire length of a trace.
- Traces can become marginally narrower to escape package balls.
- Due to the large number of SERDES lanes, it may be necessary to route some SERDES traces on buried metal layers or on the bottom layer of the PCB. In those cases, This requires signal vias under the under the AD9084 BGA SERDES BGA balls. If possible, using micro vias or back drilled vias for the SERDES signal nets will minimize the impact of any unwanted open circuit via stubs. In order to provide a good GND connection thru the board for the SERDES traces, through vias should be placed in each of the GND BGA balls on either side of the SERDES BGA Ball.
- Place coupling capacitors close to one end of the SERDES lanes to lessen reflections.
- Minimize the pad area for all connector and passive component choices as much as possible to avoid a capacitive impedance discontinuity that can lead to issues with signal integrity.
- Route the inner transmit lanes to the inner or bottom layers. Use of a laser drilled or micro via may be needed to minimize stub effects when transitioning to an inner layer.
- The on-chip JESD204B/C crossbar multiplexer can be used to alleviate routing when not all SERDES lanes are utilized.
- Ground shielding or fencing is recommended between lanes as space allows.
- To ensure optimal performance of the interface, place the device as close as possible to the baseband processor and route the traces as straight and short as possible.
- Use a PCB material with a low dissipation factor to minimize the insertion loss of the SERDES traces

For distances greater than 6 inches, use a premium PCB material such as Rogers 4003C, Isola I-Tera MT-40 or Tachyon100G

Stripline Vs. Microstrip

When routing the PCB layout for the high-speed DAC, ADC, CLK and SERDES data lines, the user must decide to route the signals using either coupled stripline or coupled microstrip traces. There are positive and negative aspects of both trace types that must be carefully considered before choosing one or the other.

The use of stripline traces has less loss, emits less electromagnetic interference (EMI), and provides better channel-to channel isolation than the use of microstrip lines. However, stripline traces require the use of vias that add line inductance and can limit the BW of ADC/DAC networks.

The use of microstrip traces is simpler to implement if the component placement and density allow routing on the top layer of the PCB, which simplifies impedance control.

If using the top layer of the PCB is problematic or if the advantages of using the stripline traces are needed, consider the following recommendations:

- Minimize the number of vias.
- Use blind vias wherever possible to eliminate via stub effects and use microvias to minimize via inductance.
- If using standard vias, use maximum via length to minimize the stub size. For example, on an 8-layer board, use Layer 7 for the stripline pair.

MORE APPLICATIONS INFORMATION

- Place a pair of ground vias near each via pair to minimize the impedance discontinuity.
- Route SERDES lanes on the top side of the PCB as a differential 100 Ω pair (microstrip) when possible. In cases where this method is not possible, the SERDES signals are routed on the inner layers of the PCB as differential 100 Ω pairs (stripline). To minimize potential coupling, these signals are placed on an inner PCB layer with a via embedded in the component footprint pad where the ball connects to the PCB. AC coupling capacitors (100 nF) on these signals are placed at the connector, away from the chip, to minimize coupling. The JESD204C interface can operate at line rates up to 28.21Gbps. Take care to maintain signal integrity from the Apollo MxFE package to the FPGA or SERDES connector

Isolation Techniques Used on the Evaluation Board

The evaluation board uses a fencing technique to provide isolation between first priority RF analog input and output signals as well as between the SERDES lane pairs (see Figure 166). Ground vias placed around each JESD204B/C pair provide isolation and decrease crosstalk. Space between vias should be less than 1/10 of the shortest wavelength supported.

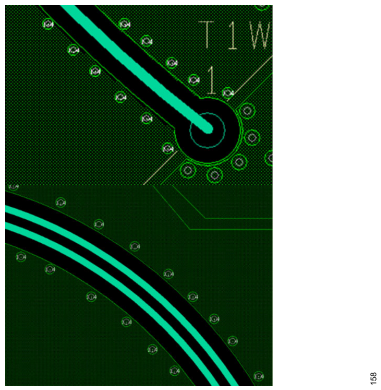

Figure 166. Fencing Technique on RF Analog (Top) and SERDES Lane Pairs (Bottom)

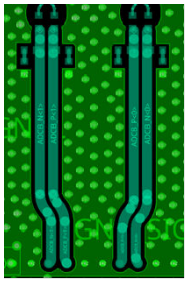

Figure 167. Isolation Optimization Technique on RF Analog

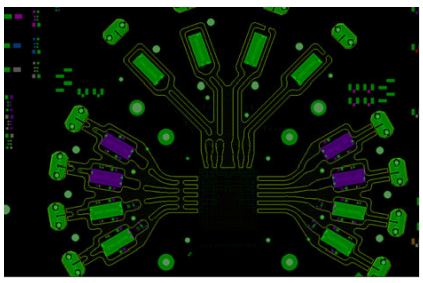

Figure 168. Metal Shield Technique for Enhanced Channel-to-Channel Isolation

MORE APPLICATIONS INFORMATION

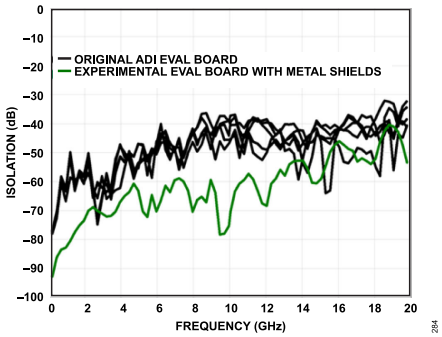

Figure 169. Measured Isolation Between Two Adjacent DAC Channels on the Original and Experimental Analog Devices Evaluation Board Designs

Ensure that spacing between apertures is not more than 1/10 of the shortest wavelength supported. Calculate the wavelength with the following equation:

$$wavelength(m) = \frac{300}{frequency(MHz) \times \sqrt{\varepsilon_r}} \quad (40)$$

Where ε_r is the dielectric constant of the isolator material.

For Rogers 4003C material with a microstrip structure (and taking air as an insulator into account), the εr value is 3.55. For FR4- 370 HR material with a stripline structure, the εr value is 4.6. For example, a maximum RF signal frequency is 6 GHz. For Rogers 4003C material with microstrip structures and εr = 3.55, the minimum wavelength is approximately 26.5 mm. To fulfill the 1/10 of a wavelength rule, square aperture spacing must be at a distance of 2.65 mm or close

The current ADI Eval board has all the DAC, ADC and CLK traces routed on the top of the PCB as microstrip. ADI is building an experimental evaluation board that adds metal shields to the top of the PCB to improve channel-to-channel isolation. The experimental Eval board also has some DAC and ADC traces routed as stripline on layer two of the PCB. It is expected that will also provide improved channel-to-channel isolation. The CLK traces were also moved to the bottom L26 layer of the experimental board. This will maximize the isolation between the ADC inputs and the CLK inputs and eliminate spurs that can be produced by ADC-to-CLK coupling.

Through-hole vias that connect the top layer to the bottom layer and all layers in between are optimal. These vias steer return current to the ground planes near the apertures. Use electromagnetic simulation software to develop accurate slot spacing and square aperture layout when designing the PCB. Figure 170 below shows an example of the differential 100Ω via design used in the evaluation board. Note that this design is specific to the stackup and material choice.

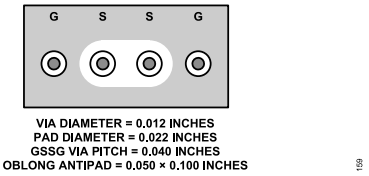

Figure 170. Differential Via Design Example

DEVICE CHANNEL LATENCY

Channel latency is the delay in data transfer in the Apollo MxFE device, and its importance to a system is significant because it directly impacts system overall latency and synchronization. In order to provide the channel latency for Apollo device, a calculator in the spreadsheet is released to the users in the Apollo evaluation software release package and named as "Apollo_Latency_Calculator_B0_v1.0.xlsm". User instructions are included in the spreadsheet to guide the user how to use the calculator to generate the latency for both transmitters and receivers.

Alternatively, a channel latency calculator tool has been integrated into the Profile Generator tool that is part of the ACE Evaluation Software. To enable this feature, just check the “Enable Latency Calculator” box in the Profile Generation Wizard as illustrated in Figure 171 (see red box on left). The estimated latency for both the receive and transmit paths will be displayed in the Profile Generator block diagram, also illustrated in Figure 171 (see red box on right).

MORE APPLICATIONS INFORMATION

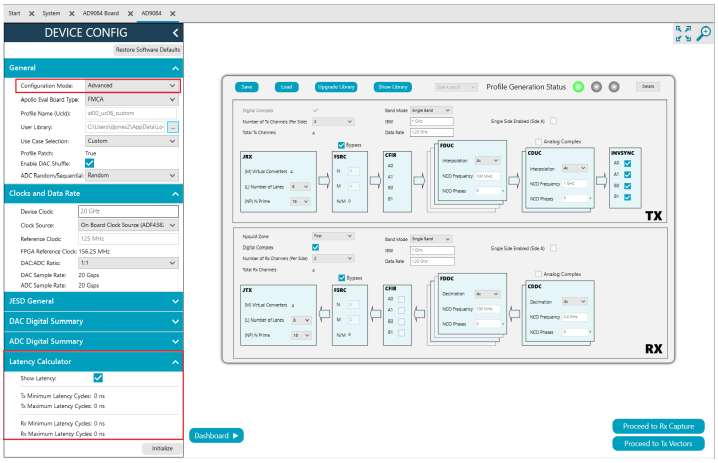

Figure 171. Enabling the Latency Calculator in the Apollo MxFE Profile Generator Tool

POWER CONSUMPTION

Power Consumption Estimation

Power consumption estimates, at the chip level, can be obtained using the Apollo Power Estimator software tool. This Windows-based executable features a simple graphical user interface and integrates a database of measurement data and simulation models. These models and data span a range of device configurations and are based on a statistically representative sample of Apollo MxFE devices from a typical production cycle.

As of this writing, the Apollo Power Estimator is still under development; currently, it provides an estimate of average power consumption, but does not yet support minimum or maximum power estimates.

POWER MANAGEMENT CONSIDERATIONS

The device has multiple digital and analog power domains to power the various mixed-signal blocks and subcircuits. Analog and digital power domains should be kept separate by a filtering network or other means to isolate digital spurious signals from coupling onto sensitive analog RF nodes, and from analog RF nodes coupling to one another. For example, the 1.0V domain is split into multiple 1.0V rails and are typically powered by separate LDOs to improve overall PSRR.

Since device power consumption depends on the device configuration and which DSP blocks are used in a particular application, in some applications it may be possible to reduce the size of the Power Delivery Network (PDN) and make it smaller compared to the PDN proposed in the AD9084-FMCA-EBZ design.

To maintain the best device performance, proper bypassing and isolation techniques must be implemented, as outlined earlier in this section.

Power Delivery Network (PDN)

A suggested power delivery network to accommodate many modes of operation for a single device is shown in Figure 172. The PDN is typically comprised of multiple switch-mode power supplies (colloquially named a “switcher”) that either directly power the device digital domains or supply current to Low-Dropout Regulators (LDOs). The PDN may include filtering stages to isolate unwanted spurious signals from sensitive analog domains. This approach allows the user to take advantage of the various switchers, integrated micro-Modules, LDOs, and power management ICs, to form a PDN that maintains high power efficiency while it also monitors the health of the device supplies and controls power-up or power-down sequencing.

Apollo MxFE’s power rails are turned ON in a specific sequence, typically by a sequencer IC such as the LTC2980. An additional benefit of this approach is the ability of the LTC2980 to adjust LDO voltages, and thus accurately set the supply voltages of the Apollo MxFE analog rails to their target Performance range. See the **AD9084 datasheet** for more details regarding Functional and Performance range of some of the analog rails.

MORE APPLICATIONS INFORMATION

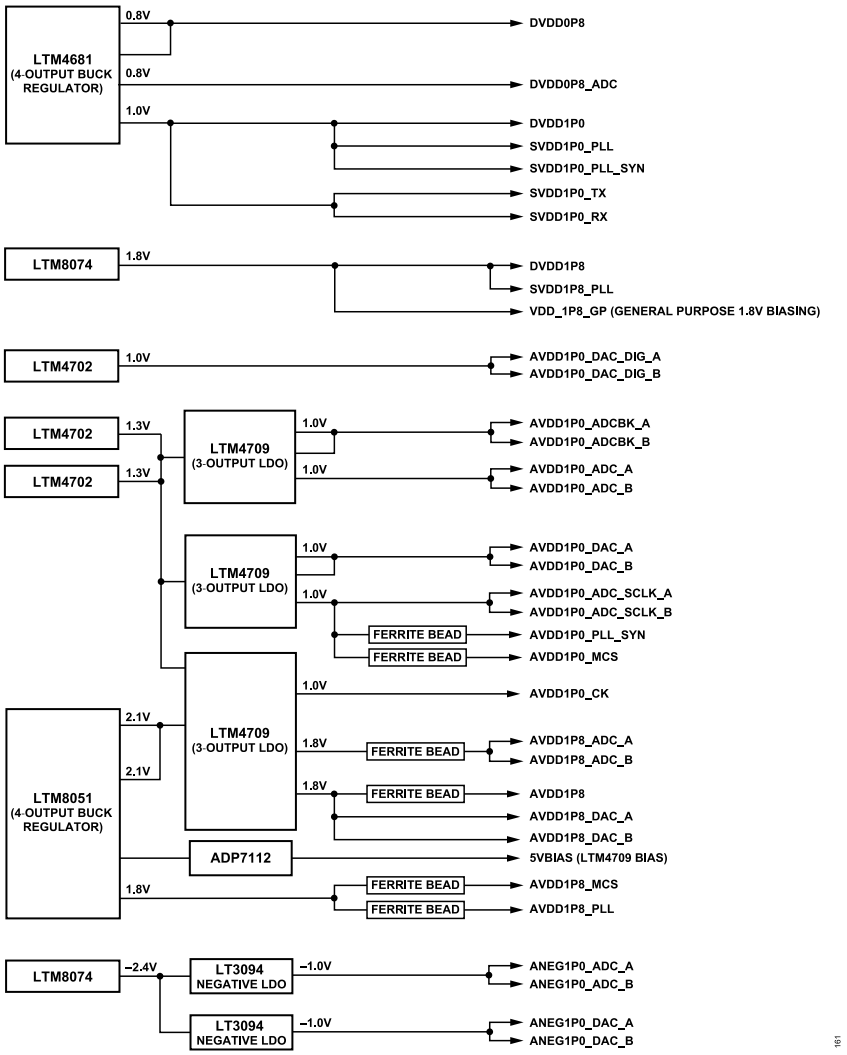

Figure 172. Suggested Power Delivery Network, as implemented in the AD9084-FMCA-EBZ Reference Design

Power Sequencing

The device's power domains must be sequenced in specific order upon device power-up. Although a power-down sequence is not needed, the Apollo MxFE rails should be powered off together and allowed to decay to 0V, whether during normal power-down operation or in response to Fault condition the sequencer IC may have detected.

Power domains may be grouped according to voltage level (while maintaining proper isolation between the digital and analog domains). During power-up, each group is enabled one at a time, starting with the lowest positive voltage (0.8V), continuing to the highest voltage (1.8V), and ending with the negative voltage rails. Each group must reach a voltage range that is within the operating range of the device, before the next group is enabled. Voltage references and internal LDO nodes that are powered externally due to specific system requirements are an exception to this rule and should be sequenced together with their associated power domain, or at the end after all the supply domains were sequenced ON.

The amount of time that it takes to sequence the device up or down, depends on the amount of bulk capacitance at each domain, which in turn depends on the current requirements, intended filtering or isolation performance, and the rate at which current must be delivered to the device during peak demands. While analog domains are generally sensitive to noise but are less dynamic (average current is approximately equal to the peak current during operation), the digital domains may experience large current swings (di/dt) that depend on the type of signal and how it is processed: pulsed signals will require a larger PDN to accommodate the dynamic loading the device presents to the PDN. Such implementation will also take longer to sequence, depending on the required bulk capacitance at the switcher output.

MORE APPLICATIONS INFORMATION

In case of a detected fault condition, such as a sustained over-voltage event, the power domains must be immediately powered down to minimize the risk of damage/reduced-device-life. Generally, remove the supply voltages together, allowing the rails to naturally decay to 0V. Adding a small resistor, possibly 50-100ohm, can force the on-board capacitance to discharge quickly.

An example of the power domain sequence that is implemented on the AD9084-FMCA-EBZ is below:

1. 0.8V digital domains
2. 1.0V digital domains
3. 1.0V analog domains
4. 1.8V digital domains
5. 1.8V analog domains (along with its associated external 1.8V LDO and 1.2V reference)
6. -1.0V analog domains

Please note that during sequencing, while the analog 1.0V domains in step #3 are ON and the analog 1.8V domains in step #5 are OFF, a leakage path exists from the 1.0V domains to the 1.8V domains, that is on the order of 30-40mA if the 1.8V LDOs are at GND potential. The leakage is normal and is to be expected during sequencing with a duration of 100-200mS but leaving the device in this state for longer than 1 second should be avoided. It is preferable to use an LDO that keeps its output at high impedance during the OFF state, such as the LTM4709 – this will guarantee that the unpowered 1.8V rail does not significantly load the 1.0V analog LDO through the Apollo MxFE IC.

The AD9084-FMCA-EBZ is a verified reference design that system integrators are encouraged to use as a starting-point, potentially reducing the size of the solution to fit a design requirement or performance target.

THERMAL MANAGEMENT CONSIDERATIONS

The power consumption of the device depends on its configuration and may exceed 40 W in modes where the ADC and DAC rates are operated at their maximum sample rate with the transmit and receive digital data paths fully utilized (or when operating in bypass mode with maximal SERDES lane rates and input/output lane utilization). When operating the device, ensure that thermal management is considered such that the die temperature of the device does not exceeds the maximum operating temperature of 110°C.

In most applications, active temperature management must be implemented using a heatsink and a cooling fan. To simplify the thermal management implementation and reduce fan noise and fatigue, the device includes an on-chip TMU that can be read back to determine the junction temperature of certain blocks. Refer to the TEMPERATURE MONITORING UNIT (TMU) <add link> for details on using the TMU, including API functions and examples of how to use them.

A typical thermal solution to manage the temperature dissipation is a fan-mounted heatsink, mechanically attached to the PCB, such as shown in Figure 173. The example shows the Device Under Test (DUT) that is to be cooled, a Thermal Interface Material (TIM) to reduce thermal resistivity at the heatsink interface to the DUT, and a heatsink with a top-mounted fan that draws cool air to blow down onto the heatsink fins.

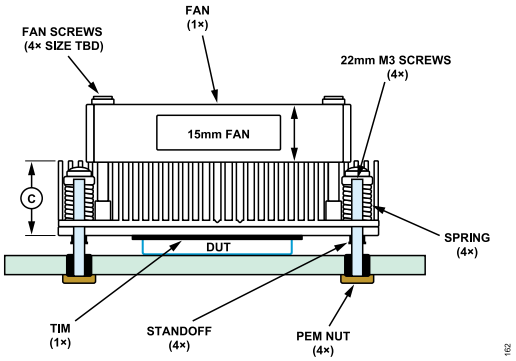

Figure 173. Side view of a thermal solution example implemented on a PCB

DEVICE TEST MODES

The Apollo MxFE device has a variety of test pattern generators for the receive paths and data pattern checkers in the transmit paths. These are described in the following sub-sections.

RECEIVE PATH TEST MODES

There are four different test pattern generators included in the Apollo MxFE receiver design as illustrated in Figure 174.

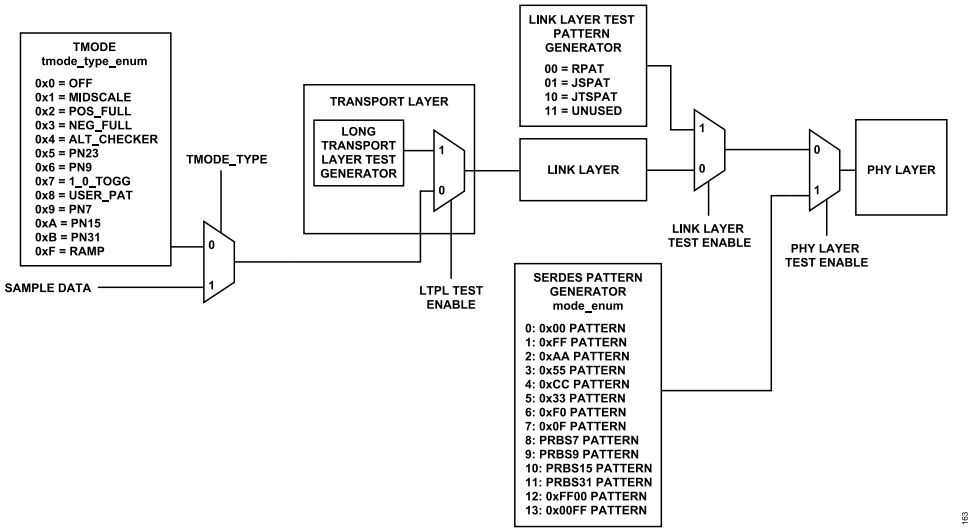

Figure 174. Receive Path Test Mode Equivalent Block Diagram (preliminary)

Receive Path TMODE

The TMODE block resides outside of the JESD204B/C transmitter block and allows the user to inject data patterns directly into any or all the virtual converter paths once the data link has been configured and is up and running. Each of the test pattern generators, including the TMODE block are depicted in Figure 174.

The **adi_apollo_tmode_config_set()** API function is used to configure the TMODE block and route test data to the virtual converter inputs of the JESD204B/C transmitter. The TMODE configuration parameters and their functions are described in Table 21. TMODE is enabled for all virtual converter inputs to the JESD204B/C transmitter whenever the "mode" parameter is a value other than 0x0. The patterns are sent to each virtual converter (M0 – Mn) enabled for the JESD204B/C transmitter mode that the device is operating in. The pattern selected is sent to each of the virtual converters. So, the pattern on M0 will be duplicated for all others. When selecting a PN sequence, this means the sequence is "per converter" rather than "per frame" (across all converters).

If the mode parameter is set to "ADI_APOLLO_TMODE_USER_PAT" the **adi_apollo_tmode_usr_pattern_set()** API function is used to set the patterns. The parameters for this function are described in Table 131. Up to eight patterns can be loaded by passing a unique usr_pattern value to the pat_address index of the function. For more information, refer to the AD9084/AD9088 API specification, integration, and porting guide.

Table 130. "adi_apollo_tmode_config_set" Parameters

Parameter	Description	Enumeration	Comments
mode	Test Mode Type. The enumeration value sets the test data pattern	ADI_APOLLO_TMODE_OFF	0x0:Normal Operation (sample data from data path)
		ADI_APOLLO_TMODE_MIDSCALE	0x1:Midscale Short
		ADI_APOLLO_TMODE_POS_FULL	0x2:Positive Full-Scale, 0x7FFF
		ADI_APOLLO_TMODE_NEG_FULL	0x3:Negative Full-Scale, 0x8000
		ADI_APOLLO_TMODE_ALT_CHECKER	0x4:Alternating, 0x5555-0xAAAA pattern
		ADI_APOLLO_TMODE_PN23	0x5:PN23 Sequence
		ADI_APOLLO_TMODE_PN9	0x6:PN9 Sequence
		ADI_APOLLO_TMODE_1_0_TOGG	0x7:1/0 Word Toggle, 0x0000-0xFFFF
		ADI_APOLLO_TMODE_USER_PAT	0x8:User Pattern Test Mode
		ADI_APOLLO_TMODE_PN7	0x9:PN7 Sequence
		ADI_APOLLO_TMODE_PN15	0xA:PN15 Sequence

DEVICE TEST MODES

Table 130. "adi_apollo_tmode_config_set" Parameters (Continued)

Parameter	Description	Enumeration	Comments
		ADI_APOLLO_TMODE_PN31	0xB:PN31 Sequence
		ADI_APOLLO_TMODE_RAMP	0xF:Ramp Output
res	TMODE output sample resolution	ADI_APOLLO_TMODE_RES_16BIT	0x0:16Bit
		ADI_APOLLO_TMODE_RES_15BIT	0x1:15Bit
		ADI_APOLLO_TMODE_RES_14BIT	0x2:14Bit
		ADI_APOLLO_TMODE_RES_13BIT	0x3:13Bit
		ADI_APOLLO_TMODE_RES_12BIT	0x4:12Bit

Table 131. "adi_apollo_tmode_usr_pattern_set" Parameters

Parameter	Description	Enumeration	Comments
regmap_base_addr	Base address for the user data pattern registers	Set by function	
pat_addr_offset_lsb	Offset to the base address indexed within the function	Set by function	Function "scrolls" through each address and sets the data vale according to usr_pattern
pat_addr_offset_msb			
usr_pattern	Value loaded into user data pattern register	Index of up to 16-bit values	Up to 16-bit value (set by "res" parameter) passed to the function as an index for each virtual converter
tmode_usr_pat_sel	In "User Pattern Test Mode", selects to repeat the pattern or not	boolean	0: Continuous Mode – repeat the 8 usr_pattern values programmed into user data pattern registers1: Single Mode – Output the pattern once, then followed by all 0's

JESD204B/C Transmitter Test Modes

JESD204B/C Transmitter PHY PRBS Testing

Because the JESD204B/C transmitter pattern generator can insert a pattern at the input to the physical layer, it can be used to conduct PHY PRBS testing. This functionality enables bit error rate (BER) testing of each physical lane of the JESD204B link. The test generator sends the same pattern to each lane that is enabled. So, each lane has its own PRBSx pattern running, and the FPGA PHY (JESD204B/C receiver) lanes must each be able self-synchronize to the pattern. Even though this is a PHY test, the JESD204B/C transmitter must be configured for a valid (supported) mode even though the link does not need to be established. Therefore, it is recommended to set up the device for the mode of operation that it is intended to run when bringing up the link. These include settings from the JESD204B/C Transmitter Mode Table, decimation mode (Table 66), and any Transmit Mux4 (JESD204B/C Receiver Lane Crossbar) settings.

The **adi_apollo_serdes_prbs_generator_enable()** API function is used to enable PHY test patterns on each active JESD204B/C physical lane. The parameters for this function are described in Table 132. The following system reference examples are provided as part of the device API package that illustrates the suggested sequence of function calls used to implement a system level PHY PRBS test using the **adi_apollo_serdes_prbs_generator_enable()** and other API functions. They are located in the ads10_apollo_ex_main example folder.

- prbs.c
- jrx_eye_sweep.c
- tx_jesd.c

A high-level flow diagram for this system function is illustrated in Figure 175. The "ads10_apollo_ex_prbs" application example in the Ads10_apollo_ex_main folder for executing the PHY PRBS test is included in the device API package.

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in theapollo-sdk\pkg\production\doc folder.

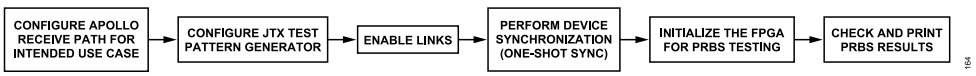

Figure 175. Example JTx PHY PRBS Application code flow diagram

DEVICE TEST MODES

Table 132. "adi_apollo_serdes_prbs_generator_enable" Parameters

Parameter	Description	Enumeration	Comments
mode	Sets the pattern of "JTx Test Pattern Generator"	ADI_APOLLO_SERDES_JTX_DATA_00	0: 0x00 serdes test pattern
		ADI_APOLLO_SERDES_JTX_DATA_FF	1: 0xFF serdes test pattern
		ADI_APOLLO_SERDES_JTX_DATA_AA	2: 0xAA serdes test pattern
		ADI_APOLLO_SERDES_JTX_DATA_55	3: 0x55 serdes test pattern
		ADI_APOLLO_SERDES_JTX_DATA_CC	4: 0xCC serdes test pattern
		ADI_APOLLO_SERDES_JTX_DATA_33	5: 0x33 serdes test pattern
		ADI_APOLLO_SERDES_JTX_DATA_F0	6: 0xF0 serdes test pattern
		ADI_APOLLO_SERDES_JTX_DATA_0F	7: 0x0F serdes test pattern
		ADI_APOLLO_SERDES_JTX_PRBS7	8: PRBS7 serdes test pattern
		ADI_APOLLO_SERDES_JTX_PRBS9	9: PRBS9 serdes test pattern
		ADI_APOLLO_SERDES_JTX_PRBS15	10: PRBS15 serdes test pattern
		ADI_APOLLO_SERDES_JTX_PRBS31	11: PRBS31 serdes test pattern
		ADI_APOLLO_SERDES_JTX_DATA_FF00	12: 0xFF00 serdes test pattern
		ADI_APOLLO_SERDES_JTX_DATA_00FF	13: 0x00FF serdes test pattern
lanes	An array of PHY lane numbers to be tested	array	Valid values are 0-23
lanes_len	The number of lanes to be tested	Decimal value	Valid values are 1-24

JESD204B/C Long Transport Layer Testing

Content to be added in subsequent version of this document.

TRANSMIT PATH TEST MODES

The test pattern checkers included in the Apollo MxFE transmitter design -primarily focused on the JESD204B/C receiver - are illustrated in Figure 176.

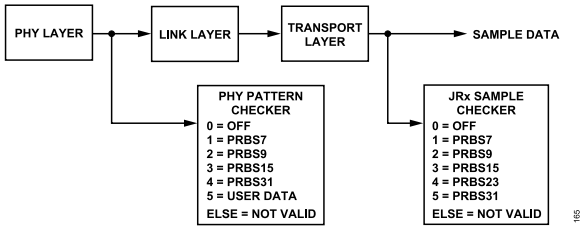

Figure 176. Transmit Path Test Mode Equivalent Block Diagram

NCO Test (NCO-Only) Mode

There are two examples in the API's pkg/src/examples/ads10_apollo_ex_main folder that implement the NCO test mode. These examples are tx_nco and tx_nco_pfilt. Refer to the FDUC NCO-Only Mode, CDUC NCO Only Mode, and NCO Programming and Debug sections for more information.

JESD204B/C Receiver PHY PRBS Testing

The JESD204B/C receiver on the device includes a PRBS pattern checker on the back end of the physical layer. This functionality enables BER testing of each physical lane of the JESD204B link.

The PHY PRBS pattern checker does not require that the JESD204B link is established. The pattern checker can synchronize with a PRBS7, PRBS9, PRBS15, or PRBS31 data pattern. PRBS pattern verification can be performed on multiple lanes at once. The error counts for failing lanes can be read back from registers.

The **adi_apollo_serdes_prbs_checker_enable()** API function is used to enable PHY test patterns on each active JESD204B/C physical lane. The parameters for this function are described in Table 133. A system reference example, **ads10_apollo_ex_prbs()**, is provided as part of the device API package that illustrates the suggested sequence of function calls used to implement a system level PHY PRBS test using

DEVICE TEST MODES

the **adi_apollo_serdes_prbs_checker_enable()** and other API functions. A high-level flow diagram for this system function is illustrated in Figure 177. Parameters associated with the PHY pattern checker and implemented in the system reference example are also included in Table 133. The “ads10_apollo_ex_prbs” application example in the Ads10_apollo_ex_main folder for executing the PHY PRBS test is included in the device API package.

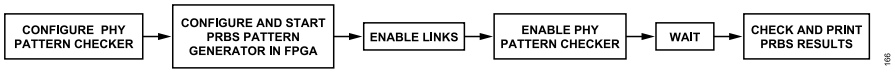

Figure 177. Example JRx PHY PRBS Application code flow diagram

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

Table 133. JESD204B/C PHY PRBS Test related Parameters

Parameter	Description	Enumeration	Comments
prbs_mode	Sets the pattern of “JRx Test Pattern Generator”	ADI_APOLLO_SERDES_JRX_PRBS_7ADI_APOLLO_SERDES_JRX_PRBS_9ADI_APOLLO_SERDES_JRX_PRBS_15ADI_APOLLO_SERDES_JRX_PRBS_31(user data pattern checking support is TBD)	1: PRBS7 serdes test pattern2: PRBS9 serdes test pattern3: PRBS15 serdes test pattern5: PRBS31 serdes test pattern
lanes	An array of PHY lane numbers to be tested	array	Valid values are 0-23
lanes_len	The number of lanes to be tested	Decimal value	Valid values are 1-24
enable	enables the PRBS pattern checker	boolean	1 = Enable
err	PRBS error flag	boolean	1 = PRBS error occurred since last power cycle of digital reset
err_sticky	“sticky” PRBS error flag	boolean	1 = error occurred since last cleared
err_cnt	number of errors since last time cleared	decimal value	range is 0-16,777,215 (24 bits)

JESD204B/C Receiver PHY Eye Scan

The device has built-in comparator circuits that enables the ability to reproduce an eye diagram estimate at the output of the CTLE circuit inside the JESD204B/C receiver core as illustrated in the PHY block diagram in Figure 178. This allows the user to see the same eye that is seen at the CDR input and can be used to verify the signal integrity of the link. Note that the eye monitor can only be run when operating the link in half-rate or quarter-rate modes (8Gbps and above).

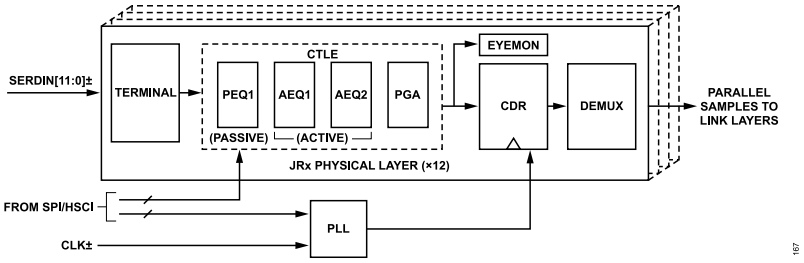

Figure 178. JRx PHY Block Diagram

Executing the JESD204B/C Receiver Eye Scan

The mechanics of the eye scan is largely executed by the device firmware which executes a sweep relative to the center of the eye diagram. There are 33 total steps in this sweep, starting at phase 0 and ending at 32, where step 16 is the center. Each step is 1/32nd of the lane rate UI. Note that the horizontal eye scan range is ±32ps from the center of the eye and the data step size for the returned data is 1/32 of the UI. Therefore, for lane rates less than 16Gbps (half-rate mode), step data outside of the ±32ps scan range will be invalid and values of ±127 will be returned from the eye scan API.

The jrx_eye_sweep example kicks off the eye scan process and its use is demonstrated in the example code found at examples/ads10_apollo_ex_main/jrx_eye_sweep.c The example code follows the flow illustrated in Figure 179.

DEVICE TEST MODES

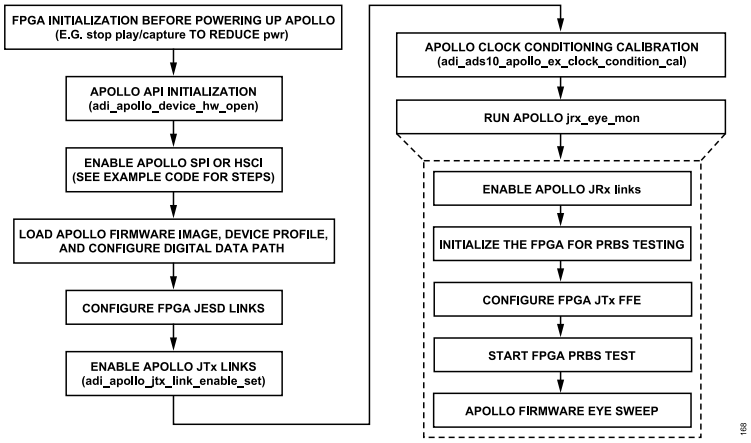

Figure 179. JRx Eye Scan Flow Diagram

When the eye scan is done running on all instantiated lanes (both A and B-sides), the horizontal JRx Eye Sweeps returns Left and Right SPO index values for corresponding lanes and prints them. While the vertical JRx Eye Sweeps return an array of positive and negative SPO values for each SPO step. Each vertical sweep returns an array containing consecutive pair of vertical values (+ and -) starting at horizontal scan step = 0 and ending at horizontal scan step = 32. The vertical values are in units of 4mV. Users can perform multiple back-to-back sweeps, and the number of sweeps can be set using variable `total_sweeps`. Default is 32. At the end of each vertical sweep, for a given lane, all SPO values are saved to a file with the name and location of each file printed. An example print out is shown in Figure 180.

```
Running JRx Vertical EYE Sweeps...
Total Sweeps per lane: 32.
lane_num: 00     Saving File: /home/analog/Apollo/jrx_eye_sweep/id99_uc02/apollo_jrx_eye_data_lane_00.txt
lane_num: 01     Saving File: /home/analog/Apollo/jrx_eye_sweep/id99_uc02/apollo_jrx_eye_data_lane_01.txt
lane_num: 02     Saving File: /home/analog/Apollo/jrx_eye_sweep/id99_uc02/apollo_jrx_eye_data_lane_02.txt
```

276

Figure 180. Example Print Out from Running Example Code

Figure 181 shows a snippet of a .txt file that is saved and the SPO format (first few lines) for a quarter rate example (Lane Rate > 16 Gbps).

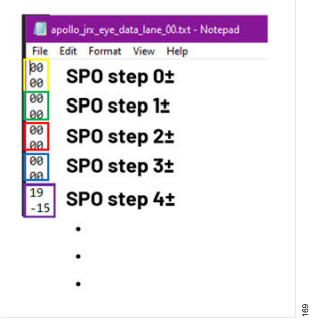

Figure 181. JRx Eye Scan .txt File Format Quarter Rate

The code does not generate a plot of the data, the user is able to choose how to plot. For this example, here's a generated plot with 32 scans overlayed with trend lines is shown in Figure 182.

DEVICE TEST MODES

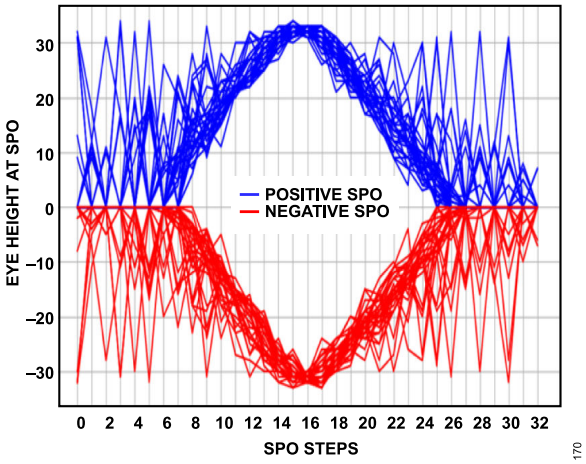

Figure 182. JRx Eye Scan Example Plot Quarter Rate

Figure 183 shows a .txt file that is saved and the SPO format (first few lines) for a half rate example (8 Gbps < Lane Rate < 16 Gbps).

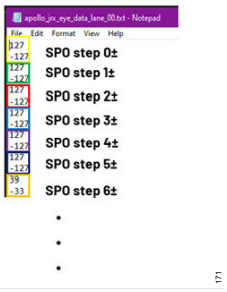

Figure 183. JRx Eye Scan .txt File Format Half Rate

The code does not generate a plot of the data, the user is able to choose how to plot. For this example, here's a generated plot with 32 scans overlayed with trend lines is shown in Figure 184.

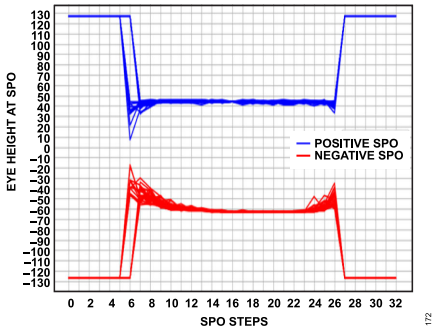

Figure 184. JRx Eye Scan Example Plot Half Rate

JESD204B/C Receiver Data Path PRBS Testing

Content to be added in subsequent version of this document.

SHORT TRANSPORT LAYER (STPL) TEST

Content to be added in subsequent version of this document.

JESD204B/C DEBUG GUIDE

PHYSICAL LAYER PRBS DEBUG

General debugging steps if PHY PRBS testing fails:

- Confirm the appropriate steps and API functions per JESD204B/C Transmitter PHY PRBS Testing or JESD204B/C Receiver PHY PRBS Testing as appropriate
- Confirm matching PRBS patterns on both sides of link
- Confirm appropriate power supply levels
- Confirm appropriate PHY layer settings for the PCB channel insertion loss
  - JESD204B/C Transmitter – Amplitude (swing), pre- and post-tap emphasis, P/N polarity (Table 64)
  - JESD204B/C Receiver – CTLE and/or DFE settings, PHY mode based on lane rate, P/N polarity (Table 69)

Note that if the channel insertion loss, insertion loss deviation, or return loss are too high, this will cause a PHY PRBS failure. It is recommended to perform a full signal integrity simulation of the JESD link using appropriate Algorithmic Modeling Interface (AMI) models and end-to-end channel models. These tests will help determine the appropriate emphasis (JESD204B/C Transmitter PHY) and CTLE (JESD204B/C Receiver PHY) settings.

LANE CROSSBAR MAPPING

It is good practice to confirm both the Receive MUX4 (JESD204B/C Transmitter Crossbar) and the Transmit Mux4 (JESD204B/C Receiver Lane Crossbar) are configured appropriately to match the physical lane assignment needed to accommodate the PCB routing of the JESD204B/C data traces between Apollo and the logic device. Mismatched lane routing typically will not prevent the link from synchronizing but could lead to unexpected data routing. Symptoms of erroneous data routing are data errors and unexpected spectral content. Sample PRBS testing or transport layer testing can reveal errors in the lane crossbar mapping. These tests are not currently supported by the device but will be enabled in future revisions of the device API.

8-BIT/10-BIT LINK ERRORS AND STATUS MONITORING

8-Bit/10-Bit Link (JESD204B) Synchronization Status and Error Monitoring

The **adi_apollo_jrx_j204b_lane_error_get()** function of the Apollo API will retrieve the per-lane status of the link synchronization process as well as monitor each lane for 8-bit/10-bit encoding errors once the link has been established. Link synchronization status is checked by the function by retrieving the status of the bit fields associated with CGS, FS (Frame Sync), CKS (Checksum), ILS (Initial Lane Synchronization)for each instantiated lane. Once the link is synchronized (CGS, FS, CKS, ILS, and ILD all “PASS”) and the link is in the user data phase (sample data is being received by the DACs), 8-bit/10-bit errors, such as Bad Disparity (BD), Not In Table (NIT), and Unexpected K-characters (UEK) can be detected. The parameters returned by the **adi_apollo_jrx_j204b_lane_error_get()** function are described in Table 134. Error fields get set whenever the error counts for BD,NIT, and/or UEK errors are greater than zero. IRQ can also be set to provide an interrupt whenever the threshold is violated.

Table 134. JESD204B Lane Status Parameters from adi_apollo_jrx_j204b_lane_error_get()

Parameter	Description	Enumerations	Comments
jrx_dl_204b_bd	Bad Disparity (BD) error status for Lanes [L-1:0]	1 = BD error	Per lane BD error flag (Read only). Indicates jrx_dl_204b_bde_cnt>=jrx_dl_204b_eth
jrx_dl_204b_nit	Not-in-table (NIT) error status for Lanes [L-1:0]	1 = NIT error	Per lane NIT error flag (Read only). Indicates jrx_dl_204b_nit_cnt>=jrx_dl_204b_eth
jrx_dl_204b_uek	Unexpected K-char (UEK) error status for Lanes [L-1:0]	1 = UEKC error	Per lane UEKC error flag (Read only). Indicates jrx_dl_204b_uek_cnt>=jrx_dl_204b_eth
jrx_dl_204b_cks	Checksum (CKS) error status for Lanes [L-1:0]	1 = CKS_PASS 0 = CKS_ FAIL	Per lane CKS status flag (Read only)
jrx_dl_204b_ils	Initial Lane Synchronization (ILS) status for Lanes [L-1:0]	1 = ILS_PASS 0 = ILS_ FAIL	Per lane ILS status flag (Read only)
jrx_dl_204b_ild	Inter-Lane De-skew (ILD) status for Lanes [L-1:0]	1 = ILD_PASS 0 = ILD_ FAIL	Per lane ILD status flag (Read only)
jrx_dl_204b_fs	Frame Sync (FS) status for Lanes [L-1:0]	1 = FS_PASS 0 = FS_ FAIL	Per lane FS status flag (Read only)

JESD204B/C DEBUG GUIDE

Table 134. JESD204B Lane Status Parameters from adi_apollo_jrx_j204b_lane_error_get() (Continued)

Parameter	Description	Enumerations	Comments
jrx_dl_204b_cgs	Code Group Sync (CGS) status for Lanes [L-1:0]	1 = CGS_PASS 0 = CGS_FAIL	Per lane CGS status flag (Read only)

8-Bit/10-Bit Link Error Debugging Tips

Table 135 lists the causes and actions required in debugging errors reported when errors are returned while monitoring the status and health of the lane using the **adi_apollo_jrx_j204b_lane_error_get()** function .

Table 135. Possible Causes and Actions

Error	Possible Causes/Action
CGS_FAIL	Clock issue (improper clock settings or poor signal quality) JESD204B configuration mismatch (#lanes) power supply noise or droop Check PRBS on each lane first. If PRBS fails, refer to PHY PRBS failure.
FS_ FAIL	JESD204B configuration mismatch or invalid configuration Signal integrity (Run PHY PRBS test)
ILS_ FAIL	JESD204B configuration mismatch. Check JESD204B/C parameters for both ends of the link to ensure they are the same, particularly the K value. If receiver is checking for ramp pattern during ILAS, check pattern or disable checking. Check/verify that the required /R/, /Q/, and /A/ key words are in the ILAS and if they are in the correct position.
ILD_ FAIL	Lane de-skewing failure. If no other errors present, could indicate excessive trace length mismatch.
CKS_ FAIL	JESD204B configuration mismatch. Check JESD204B/C parameters for both ends CKS calculation method mismatch. Check 204B_CKS parameter
BD/NIT/UEKC error	Poor signal integrity – PCB assembly issue, improper PHY settings (emphasis, amplitude, or equalization), or poor PCB design
SYNCOUTB_xx Never Goes Low	Follow Analog Devices recommended initialization procedure. Check that SYNC~ source and board circuitry (both SYNC+ and SYNC− if differential) are properly configured to produce logic levels compliant for the SYNC~ receive device. If logic level is not compliant, then review circuitry for source and receiver configurations to find the problem. Otherwise, consult device manufacturer.

JESD204B/C DEBUG API

Refer to "apollo_api.chm" for more details on the device API functions, structures and parameters. This file is included in the API package in the apollo-sdk\pkg\production\doc folder.

JESD204B/C Transmitter Status

The **adi_apollo_jtx_link_status_get** API function will return the status of the JESD204B/C transmitter related parameters listed in Table 136

Table 136. JESD204B/C Transmitter Link Status Parameters

Parameter Name	Description	Enumeration
JTX_PLL_LOCKED	PLL Locked Status Bit. Applicable to both links.	0 : PLL_NOT_LOCKED 1 : PLL_LOCKED
JTX_PHASE_ESTABLISHED	Validates LEMC phase has been successfully passed from transport layer to link layer	0 : PHASE_NOT_ESTABLISHED 1 : PHASE_ESTABLISHED
JTX_INVALID_MODE	Quick Config Mode Number validity	0 : VALID 1 : INVALID

Table 137 describes debug steps to take for unwanted conditions indicated by the **adi_apollo_jtx_link_status_get()** function.

Table 137. JESD204B/C Transmitter Link Status Debug Steps

Condition	Debug Steps / Possible causes
JTX_PLL is unlocked	Check for presence of CLKIN input, verify frequency and signal quality Validate appropriate PLL settings are used . See SERDES PLL and Configuration section
JTX_PHASE is not established	If single link mode, verify link1 is disabled.

JESD204B/C DEBUG GUIDE

Table 137. JESD204B/C Transmitter Link Status Debug Steps (Continued)

Condition	Debug Steps / Possible causes
	Validate that JESD mode plus DDC settings is supported Validate JTX_NS and fs is supported
JTX_ MODE is invalid	Check the device profile to ensure the JTX_MODE parameter is valid (matches number in the JESD204B/C Transmitter Mode Table

JESD204B/C Receiver Status

The **adi_apollo_jrx_link_status_get()** API function will return the status of the JESD204B/C receiver related parameters listed in Table 138.

Table 138. JESD204B/C Receiver Link Status Parameters

Parameter Name	Description	Enumeration
DOWN_SCALE_OVERFLOW	Checks If NS Down_Scale_Ratio is Too Large	0 : No overflow 1 : Overflow
JESD_MODES_NOT_IN_TABLE(JRX)	Quick Config Mode Number validity (for link0 and link1 – 2 bits)	0 : VALID : 1 : INVALID (not in mode table)
JRX_CORE_USR_DATA_RDY	Output data samples contains valid user data. This may not exactly align with an LMFC boundary.	0 : User Data Not Ready. The link has not been established yet and JESD received data is not available. 1 : JESD User Data Ready. The link is established and correctly received data is available
JRX_CORE_SYSREF_RCVD	SYSREF Phase Has Been Established	0 : No Sysref. JRX has not received SYSREF signal and data receiving has not started. 1 : Sysref Received. JRX has received SYSREF and can receive data
JRX_CORE_CFG_INVALID	S, NS, F, NP and L Have to Be Within Pre-defined Range	0 : JESD is correctly configured. 1 : Invalid Configuration. At least one of the parameters is invalid: S, F, NP, and L

Table 139 describes debug steps to take for unwanted conditions indicated by the **adi_apollo_jrx_link_status_get()** function.

Table 139. JESD204B/C Receiver Link Status Debug Steps

Condition	Debug Steps / Possible causes
DOWN_SCALE_OVERFLOW indicates overflow condition	Contact ADI.
JRX_ MODE is invalid	Check the device profile to ensure the JRX_MODE parameter is valid (matches number in the JESD204B/C Receiver Mode Tables

JESD204B Lane Status

The **adi_apollo_jrx_j204b_lane_status_get()** API will be used to get the status of the JESD204B receiver state machine in Table 140.

Table 140. JESD204B Receiver Lane Status Parameters

Parameter Name	Description	Enumeration
JRX_DL_204B_LANE_STATUS	Current state of the JESD204B link	0: the End of Frame 1: the End of Multi-Frame 2: frame sync lost 3 sync~ output from link layer. 4: data payload available. 0=data not ready, 1=data ready 5: checksum status. 0=bad checksum, 1=good checksum 6: Reserved 7.Reserved

The **adi_apollo_jrx_j204b_lane_error_get()** function of the Apollo API will retrieve the per-lane status of the link synchronization process as well as monitor each lane for 8-bit/10-bit encoding errors once the link has been established. Refer to Table 134 and Table 135 for the debug steps.

JESD204B/C DEBUG GUIDE

JESD204C Lane Status

The **adi_apollo_jrx_j204c_lane_status_get()** API function will return the status of the JESD204C receiver state machine as described in Table 141.

Table 141. JESD204C Receiver Lane Status Parameters

Parameter Name	Description	Enumeration
JRX_DL_204C_STATE	Current state of the JESD204C link	0: RESET. 1: UNLOCKED. Complete Sync Header (SH) detection and alignment 2: BLOCK. Complete the lane de-skew and lane alignment 3: DESKEW. Complete Multi-Block detection and alignment. 4: M_BLOCK. Complete Multi-block alignment. 5: E_M_BLOCK. Complete Extended Multi-block alignment. 6: Link Ready. Link is up and running

Table 142. JESD204C Lane Status Debug Steps

Condition	Debug Steps/Possible Causes
Sync Header alignment	Sync header lock is not achieved. Check for the PHY layer issues, check for the signal integrity of the lanes. Check for the SERDES bit ordering, it should follow MSB first convention.
EMB lock	Clocking issues-Incorrect SYSREF frequency or timing can lead to LEMC alignment failures, hence causing failures in achieving deterministic latency. Link configuration parameters mismatch: Check if the link configuration parameters like (L,M,F, scrambling settings etc) matches on both the JTX and the JRX sides Lane/Polarity inversions: If SH is locked, and EMB not locked, check for the lane mapping and polarity inversions.

JESD204B/C Lane FIFO Status

The receive buffer status of the JESD204B/C Receiver can be monitored as well using the **adi_apollo_jrx_lane_fifo_status()** function. If this function returns FIFO_FULL or FIFO_EMPTY status as shown in , this could be indicative that a skew may be introduced into the clocks on the JESD204B/C Receiver. This does not necessarily mean data is being missed since these flags really mean “almost full” and “almost empty”. The implication of these flags is that the FIFO is close to a timing violation. If the link is not working properly, then there might be an asynchronous relationship between the FIFO read/write clocks. Check the lane rates on the FPGA and Apollo device are consistent and matches with each other.

The **adi_apollo_jrx_lane_fifo_status()** API function will return the status of the JESD204B/C receiver FIFO as described in Table 143

Table 143. JESD204B/C Receiver FIFO Status Parameters

Parameter Name	Description	Enumeration
LANE_FIFO_EMPTY	FIFO empty flag per instantiated PHY lane (up to 12), For example, if SERDIN lanes 0,3,4, and 7 are used, 0x099 return values means all 4 lanes have empty FIFOs.	1 : FIFO is empty
LANE_FIFO_FULL	FIFO full flag per instantiated PHY lane (up to 12), For example, if SERDIN lanes 0,3,4, and 7 are used, 0x099 return values means all 4 lanes have full FIFOs.	1 : FIFO is full

JESD204C IRQ

Table 144. API Functions for JESD IRQ

API function	Description
adi_apollo_jrx_j204c_irq_enable_set(adi_apollo_device_t *device, const uint16_t links, const uint32_t irqs, uint8_t enable)	Enables the IRQ for JESD204C link status alerts.
adi_apollo_jrx_j204c_irq_enable_get(adi_apollo_device_t *device, const uint16_t links, uint32_t *irqs_enabled)	Retrieves the enable status for JESD204C IRQs.

JESD204B/C DEBUG GUIDE

Table 144. API Functions for JESD IRQ (Continued)

API function	Description
adi_apollo_jrx_j204c_irq_get(adi_apollo_device_t *device, const uint16_t links, const uint32_t irqs, uint8_t clear, uint32_t *status)	Retrieves the status of JESD204C IRQs.
adi_apollo_jrx_j204c_irq_clear(adi_apollo_device_t *device, const uint16_t links, const uint32_t irqs)	Clears JESD204C IRQ sticky bits.

The JESD204/C receiver IRQ parameters are described in Table 145. After IRQ is detected, Apollo JRx brings down the link and brings it back up to recover the link.

Table 145. JESD204C Receiver CRC Check/IRQ Parameter

Parameter Name	Description	Enumeration
JRX_J204C_IRQ_CRC	JESD204C CRC IRQ status (per link) – CRC error occurred on at least one lane	0: No CRC errors 1: CRC error has occurred
JRX_J204C_IRQ_SH	JESD204C SH IRQ status (per link) –at least one lane did not achieve sync header lock	0: No SH errors 1: SH error has occurred
JRX_J204C_IRQ_MB	JESD204C MB IRQ status (per link) – at least one lane did not achieve multiblock lock	0: No MB errors 1: MB error has occurred
JRX_J204C_IRQ_EMB	JESD204C EMB IRQ status (per link) – at least one lane did not achieve extended multiblock lock	0: No EMB errors 1: EMB error has occurred
JRX_J204C_IRQ_DATA_RDY_LOST		0: normal operation 1: data_rdy is lost

CRC errors could be indicative of poor signal integrity on the JESD data lanes if there are no other associated errors such as loss of link (JRX_DL_204C_STATE ≠ 6). In this case, the PHY PRBS test can be run. Any of the other IRQs indicate JESD204C synchronization errors (JRX_DL_204C_STATE ≠ 6).

For SH, MB, and EMB errors, users can use the **adi_apollo_jrx_j204c_lane_status_get()** API function to determine which lanes produced the interrupt. If any of these errors occur, they could be indicative of clock or configuration mismatch between the two sides of the link, data errors, or signal integrity issues.

In addition to the above, the receive buffer status of the JESD204B/C Receiver can be monitored as well using the **adi_apollo_jrx_lane_fifo_status()** function. The details refer to JESD204B/C Lane FIFO Status section.

NOTES