ANALOG DEVICES

AD9164 API Specification Rev 1.0

TABLE OF CONTENTS

INTRODUCTION

PURPOSE

This document serves as a programmer's reference for using and utilizing various aspects of the ADI High Speed Converters DAC Application Program Interface (API) library for the AD9164 DAC. It describes the general structure of the AD9164 API library, provides a detail list of the API functions and its associated data structures, macros, and definitions.

SCOPE

Currently the AD9164 API libraries targets the AD9164 16-bit 12 GSPS, RF Digital to Analog Converter

Table 1 AD9164

Target Device Name	Device Description	Device Release Status	API Release Status
AD9164	16-bit 12 GSPS, RF Digital to Analog Converter	Released	Rev 1.0.0

DISCLAIMER

The software and any related information and/or advice is provided on and "AS IS" basis, without representations, guarantees or warranties of any kind, express or implied, oral or written, including without limitation warranties of merchantability fitness for a particular purpose, title and non-infringement. Please refer to the Software License Agreement applied to the source code for full details.

SOFTWARE ARCHITECTURE

The AD9164 API library is a collection of APIs that provide a consistent interface to a variety of ADI High Speed Converter DAC devices. The APIs are designed so that there is a consistent interface to the devices.

The library is a software layer that sits between the application and the DAC hardware. The library is intended to serve two purposes:

1- Provide the application with a set of APIs that can be used to configure RX hardware without the need for low-level register access. This makes the application portable across different revisions of the hardware and even across different hardware modules.

2- Provide basic services to aid the application in controlling the DAC module, such as interrupt service routine, DAC high-level control and status information.

The driver does not, in any shape or form, alter the configuration or state of DAC module on its own. It is the responsibility of the application to configure the part according to the required mode of operation, poll for status, etc… The library acts only as an abstraction layer between the application and the hardware.

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

- Configuring the JESD Interface
- Configuring the NCO

The application should access the DAC device only through the DAC libaries exported APIS. It is not recommended for the application to access the DAC hardware device directly using direct SPI access. If the application chooses to directly access the DAC hardware this should be done in a very limited scope, such as for debug purposes and it should be understood that this practice may affect the reliability of the API functions.

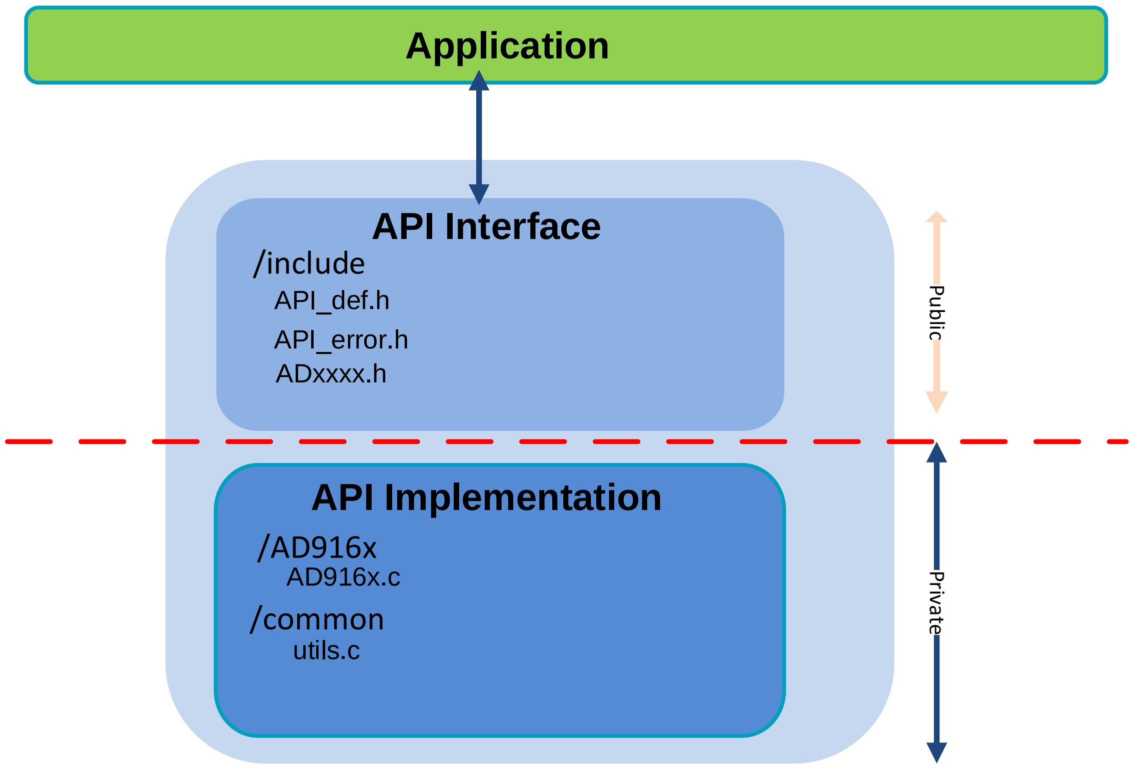

Figure 1 Simple Overview of the DAC API Architecture

FOLDER STRUCTURE

The collective files of the AD9164 API library are structure as depicted in Figure 2. Each branch is explained in the following sections. The library is supplied in source format. All source files are in standard ANSI C to simply porting to any platform.

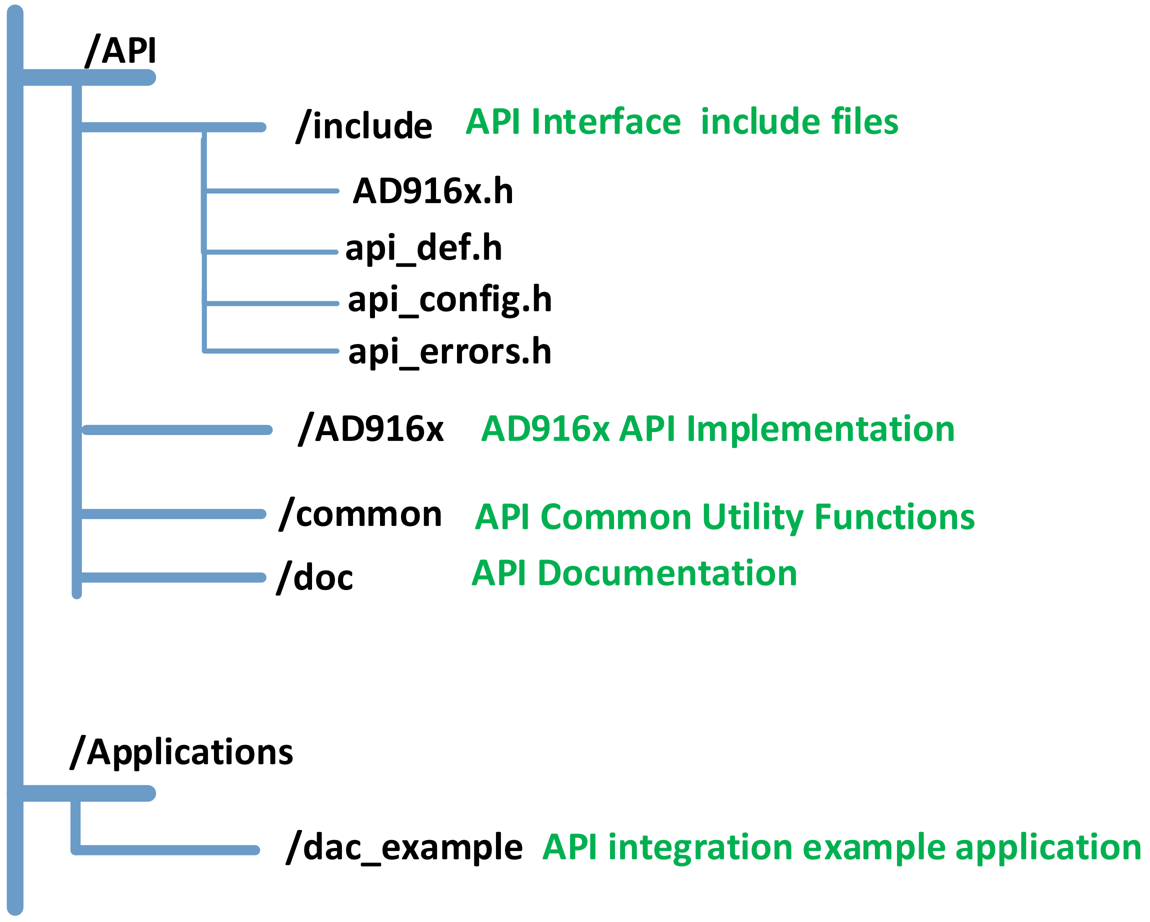
/API
/include
API Interface include files
AD916x.h
api_def.h
api_config.h
api_errors.h
/AD916x
AD916x API Implementation
/common
API Common Utility Functions
/doc
API Documentation
/Applications
/dac_example
API integration example application

Figure 2 AD9164 Source Code Folder Structure

/API

The AD9164 API root folder contain all the source code and documentation for the AD9164 API.

/API/include

This folder contains all the API public interface files. These are the header files required by the client application.

/API/AD9164

This folder includes the main API implementation code for the AD9164 DAC APIs and any private header files uses by the API. ADI maintains this code as intellectual property and all changes are at their sole discretion.

/API/common

This folder contains ADI helper functions common to all APIs, these functions are internal private functions not designed for use by client application.

/API/doc

This folder contains the doxygen documentation for the AD9164 APIs.

/Application/

This folder contains simple source code examples of how to use the DAC API. The application targets the AD9164 evaluation board platform. Customers can use this example code as a guide to develop their own application based on their requirements.

API INTERFACE

OVERVIEW

The header files listed in include folder, ***/API/include***, describe public interface of the DAC API the client application. It consists of a number of header files listed in Table 2. Each API library will have a header file that lists its supported APIs that the client application may use to interface with the ADI device. For example, the AD9164.h header file lists all the APIs that are available to control and configure the AD9164 DAC device. The other header files are used for definitions and configurations that may be used by the client application. The features of which will be described in subsequent sections.

Table 2 DAC API Interface

Device Name	Description	To be included in Client Application
AD9164.h	Lists AD9164 DAC API Library exposed to client application	Yes
api_config.h	Defines the various configuration options for the DAC Module	No
api_def.h	Defines any macros/enumerations or structures or definitions common to and used by all DAC API Libraries	No
api_error.h	Defines the DAC API interface errors and error handlers common to and used by all DAC API Libraries	Yes

AD9164.H

The AD9164 API library has a main interface header file AD9164.h header file that defines the software interface to the AD9164 DAC. It consists of a list of API functions and a number of structures and enumerations to describe the configurations and settings that are configurable on that particular device. In addition, the DAC device handle ***ad9164_handle_t*** this is a data structure that acts a software reference to a particular instance to of the DAC device. This handle maintains a reference to HAL functions and the configuration status of the chip.
This reference shall be instantiated by the client application, initialized by the application with client specific data.

API Handle

A summary of the user configurable components of this handle structure are listed in Table 3. Refer to the ***ad9164_handle_t*** section and the ***HAL Function Pointer DataTypes*** section for full a description and more details on configuration.
The platform specific members of the structure must be configured by the client application prior to calling any API with the handle, refer to the ***DAC Hardware Initialization*** section for more details.

Table 3 Components of the DAC API handle

Structure Member	Description	User Read/Write Access	Required by API
user_data	Void Pointer to a user defined data structure. Shall be passed to all HAL functions.	Read/Write	Optional
sdo	Device SPI Interface configuration for DAC hardware	Read/Write	Yes
dac_freq_hz	DAC Clock Frequency configuration	Read/Write	Yes
dev_xfer	Pointer to SPI data transfer function for DAC hardware	Read/Write	Yes
delay_us	Pointer to delay function for DAC hardware	Read/Write	Yes
hw_open	Pointer to platform initialization function for DAC hardware	Read/Write	Optional
hw_close	Pointer to the platform shutdown function for DAC hardware	Read/Write	Optional
event_handler	Pointer to a client event handler function for DAC device.	Read/Write	Optional
tx_en_pin_ctrl	Pointer to client application control function of DAC device TX_ENABLE pin	Read/Write	Optional
reset_pin_ctrl	Pointer to client application control function of DAC device RESETB pin	Read/Write	Optional

API_CONFIG.H

The API configuration header file, ***api_config.h,*** located in the ***/barium_api/include*** folder defines the compilation build configuration options for the DAC API.

The client application in general is not required to include or modify this file.

ADI_DEF.H

The AD9164 API is designed to be platform agnostic. However, it requires access to some platform functionality, such as SPI read/write and delay functions that the client application must implement and make available to the AD9164 API. These functions are collectively refered to as the platform Hardware Abstraction Layer (HAL).

The HAL functions are defined by the API definition interface header file, ***adi_def.h***. The implementation of these functions is platform dependent and shall be implemented by the client application as per the client application platform specific requirements. The client application will point the AD9164 API to the required platform functions on instantiation of the AD9164 API handle. The following is a description of HAL components.

The AD9164 API handle, ***ad9164_handle_t*** , has a function pointer member for each of the HAL functions and are listed in Table 4. The client application shall assign each pointer the address of the target platform's HAL function implementation prior to calling any DAC API.

Table 4 Short Description of HAL Functions

Function Pointer Name	**Purpose**	**Requirement**
*spi_xfer_t	Implement a SPI transaction	Required
*hw_open_t	Open and initialize ll resources and peripherals required for DAC Device	Optional
*hw_close_t	Shutdown and close any resources opened by hw_open_t	Optional
*delay_us_t	Perform a wait/thread sleep in units of microseconds	Required
*tx_en_pin_ctrl_t	Set DAC device TX_ENABLE pin high or low.	Optional
*reset_pin_ctrl_t	Set DAC device RESETB pin high or low.	Optional
*event_handler_t	Event notification handler	Optional

DAC Hardware Initialization

The client application is responsible for ensuring that all required hardware resources and peripherals required by but external to the DAC are correctly configured. The DAC API handle ***ad9164_handle_t*** defines two pointer function members to which the client application may optionally provide HAL functions to initialize these resources, ****hw_open_t*** and ****hw_close_t.*** If the client application provides valid functions via these function pointers, the DAC initialization APIs ***Ad9164_init*** and ***ad9164_deinit*** shall call hw***_open_t*** and ****hw_close_t*** respectively to handle the initialization and shutdown of required hardware resources. If the client application chooses not use this feature, the AD9164 API assumes that SPI and all the external resources for the AD9164 ReDAC are available.

The DAC API libraries require limited access to hardware interfaces on the target platform. These are depicted in Figure 3.

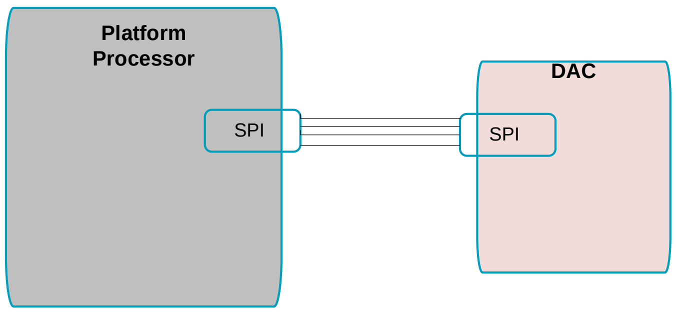

Figure 3 Hardware Controls Required By DAC API HAL

SPI Access

Access to the SPI controller that communicates with the AD9164 DAC devices is required for correct operation of the API. The API requires access to a SPI function that can send SPI commands. This function ***spi_xfer_t*** is defined in detail in the next section. The DAC SPI requires 15 bit addressing with 8-bit data bytes. The AD9164 DAC SPI interface supports 3 wire or 4 wire and the AD9164 DAC must be configured as such to match the platform implementation. This is done during initialization via the ***ad9164_init*** API. Please refer to the target ADI device datasheet for full details of the SPI Interface protocol

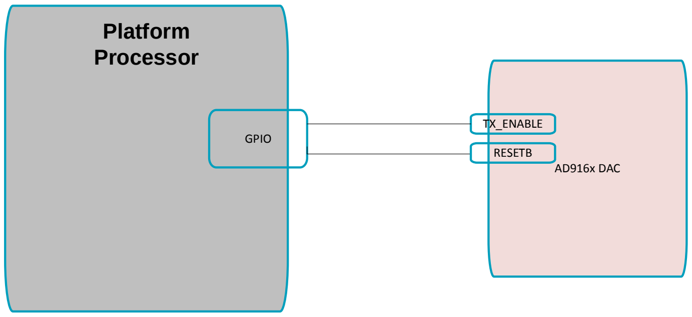

Figure 4 Option Hardware Controls Used by the DAC API HAL

RESETB Pin Access

Optionally access to the function that controls the AD9164 DAC RESETB pin can be included in the HAL layer. This function if provided allows the API to implement a hardware reset rather than a software reset.

The HAL function **reset_pin_ctrl_t** is defined in detail in the next section. Please refer to the target ADI device datasheet for full details on the RESETB pin hardware connections.

TX_ENABLE PIN Access

Optionally access to the function that controls the AD9164 DAC TX_ENABLE pin can be included in the HAL layer. This function if provided allows the client to call control the TX_ENABLE pin via the ***ad9164_transmit_enable*** API. If it is not provided the transmit enable feature in API shall be limited to using the SPI control.

The HAL function **** tx_en_pin_ctrl_t*** is defined in detail in the next section. Please refer to the target ADI device datasheet for full details on the TX_ENABLE pin hardware connections.

System Software Functions

Delay Function

For best performance, it is recommended to provide the API access to the client application's delay function. The delay function can be a wait or sleep function depending on the client application. This function allow the API to wait the recommended microsecond between initialization steps.
The HAL function * ***delay_us_t*** is defined in detail in the next section.

Event Notification

Optionally access to an API event handler function may be included in the HAL layer. This function shall be used to pass interrupt event data to the client application. This function shall be called by the ***ad9164_isr*** API only to notify all interrupt events detected and any additional data available regarding that event. The HAL function ***event_handler_t*** is defined in detail in the next section.

HAL FUNCTION POINTER DATATYPES

*HW_OPEN_T

Description

Function pointer definition to a client application function that implements platform hardware initialization for the AD9164 Device.

This function may initialize external hardware resources required by the AD9164 Device and API for correct functionality as per the target platform. For example initialization of SPI, GPIO resources, clocks etc.

If provided, this function shall be called during the DAC module initialization via API ***ad9164_init***. The API will then assume that all require external hardware resources required by the DAC are configured and it is safe to interact with the DAC device.

Synopsis

typedef void(*hw_open_t)(void *user_data);

Preconditions

Unknown- Platform Implementation.

Post conditions

It is expected that all external hardware and software resources required by the DAC API are now initialized appropriately and accessible by the DAC API.

Dependencies

Unknown- Platform Implementation

Parameters

user_data	A void pointer to a client defined structure containing any parameters/settings that may be required by the function to open the hardware for the ADI Device.

Return value

Zero shall indicate success.

Any other positive integer value may represent an error code to be returned to the application.

Notes

This function is not required to integrate the API. It is an optional feature that may be used by the client application.

*HW_CLOSE_T

Description

Function pointer to function that implements platform hardware de-initialization for the AD9164 Device

This function shall close or shutdown external hardware resources required by the AD9164 Device and API for correct functionality as per the target platform. For example initialization of SPI, GPIO resources, clocks etc.

It should close and free any resources assigned in the ***hw_open_t*** function. This function if provided shall be called during the DAC module de-initialization via API ***ad9164_init*.** The API will then assume that all require external hardware resources required by the DAC are no longer available and it is not-safe to interact with the DAC device.

Synopsis

typedef void(*hw_close_t)(void *user_data);

Preconditions

It is expected that there are no pre conditions to this function. All initialization required shall be performed prior to or during (via ****hw_open_t***) ***ad9164_init***.

Post conditions

It is expected that there are no post conditions to this function.

Dependencies

Unknown- Platform Implementation

Parameters

user_data	A void pointer to a client defined structure containing any parameters/settings that may be required by the function to close the hardware for the ADI Device.

Return value

Zero shall indicate success.

Any other positive integer value may represent an error code to be returned to the application.

Notes

This function is not required to integrate the API. It is an optional feature that may be used by the client application.

*SPI_XFER_T

Description

Function to implement a SPI transaction to the DAC device.

This function shall perform send a read/write SPI command to targeted ADI DAC device. The SPI implementation shall support 15-bit addressing and 8-bit data bytes. This function shall control the SPI interface including the appropriate chip select to correctly send and retrieve data to the targeted ADI DAC device over SPI.

The implementation may support 3-wire or 4-wire mode. The DAC API must be configured to support the platform implementation this is done during the DAC initialization API ad916x_init**.**

Once a DAC device is initialized via the ***ad9164_init***API, it is expected that the API may call this function at any time.

Synopsis

typedef int(*spi_xfer_t)(void *user_data, uint8_t *indata, uint8_t *outdata, int size_bytes);

Preconditions

It is expected that there are no pre conditions to this function. All initialization required shall be performed prior to or during (via ****hw_open_t***) ***Ad9164_init.*** Once a DAC device is initialized via the ***Ad9164_init*** API, it is expected that the API may call this function at any time.

Post conditions

It is expected that there are no post conditions to this function.

Dependencies

Unknown- Platform Implementation

Parameters

user_data	A void pointer to a client defined structure containing any parameters/settings that may be required by the function to implement the SPI for the ADI Device. For example, chip select may be passed to the function via this parameter.
indata	pointer to a uint8_t array with the data to be sent on the SPI
outdata	pointer to a unit8_t array to which the data from the SPI device will be written
size_bytes	an integer value indicating the size in bytes of both indata and outdata arrays.

Return value

Zero shall indicate success.

Any other positive integer value may represent an error code to be returned to the application.

Notes

indata and outdata arrays shall be the same size.

* TX_EN_PIN_CTRL_T

Description

Function to implement set the TX_ENABLE pin of the DAC device high or low.

Once a DAC device is initialized via the ***ad9164_init*** API, it is expected that the API may call this function at any time.

Synopsis

typedef int(*tx_en_pin_ctrl_t)(void *user_data, uint8_t enable);

Preconditions

It is expected that there are no pre conditions to this function. All initialization required shall be performed prior to or during (via ****hw_open_t***) ***ad9164_init.*** Once a DAC device is initialized via the ***ad9164_init*** API, it is expected that the API may call this function at any time.

Post conditions

It is expected that there are no post conditions to this function.

Dependencies

Unknown- Platform Implementation

Parameters

user_data	A void pointer to a client defined structure containing any parameters/settings that may be required by the function to implement the SPI for the ADI Device. For example, chip select may be passed to the function via this parameter.
enable	A uint8_t value indicating the desired enable/disable setting for the tx_enable pin.
	A value of 1 indicates TX_ENABLE pin is set HIGH.
	A value of 0 indicates TX_ENABLE pin is set LOW.

Return value

Zero shall indicate success.

Any other positive integer value may represent an error code to be returned to the application.

Notes

This function is not required to integrate the API. It is an optional feature that may be used by the client application should the user which to control the pin via the API. The relevant API function is ***ad9164_transmit_enable .***

*RESET_PIN_CTRL_T

Description

Function to implement set the RESETB pin of the DAC device high or low.

Synopsis

typedef int(*reset_pin_ctrl_t)(void *user_data, uint8_t enable);

Preconditions

It is expected that there are no pre conditions to this function. All initialization required shall be performed prior to or during (via ***hw_open_t**) **ad9164_init.*** Once a DAC device is initialized via the ***ad9164_init*** API, it is expected that the API may call this function at any time.

Post conditions

It is expected that there are no post conditions to this function.

Dependencies

Unknown- Platform Implementation

Parameters

user_data	A void pointer to a client defined structure containing any parameters/settings that may be required by the function to implement the SPI for the ADI Device. For example, chip select may be passed to the function via this parameter.
enable	A uint8_t value indicating the desired enable/disable setting for the tx_enable pin.
	A value of 1 indicates RESETB pin is set HIGH.
	A value of 0 indicates RESETB pin is set LOW.

Return value

Zero shall indicate success.

Any other positive integer value may represent an error code to be returned to the application.

Notes

This function is not required to integrate the API. It is an optional feature that may be used by the client application should the user which to control the pin via the API. The relevant API function is ***ad9164_reset.***

* DELAY_US_T

Description

Function to implement a delay for specified number of microseconds.

Any timer hardware initialization required for the platform dependent implementation of this function must be performed prior to providing to calling any DAC APIs.

Once a DAC device is initialized via the ***ad9164_init*** API, it is expected that the API may call this function at any time.

Synopsis

typedef int(*delay_us_t)(void *user_data, int us);

Preconditions

It is expected that there are no pre conditions to this function. All initialization required shall be performed prior to or during (via ****hw_open_t***) ***ad9164_init.*** Once a DAC device is initialized via the ***ad9164_init*** API, it is expected that the API may call this function at any time.

Post conditions

It is expected that there are no post conditions to this function.

Dependencies

Unknown- Platform Implementation

Parameters

user_data A void pointer to a client defined structure containing any parameters/settings that may be required by the function to implement the delay for the ADI Device.

int us time to delay/sleep in microseconds

Return value

Zero shall indicate success.

Any other positive integer value may represent an error code to be returned to the application.

Notes

This is required for optimal performance.

Performs a blocking or sleep delay for the specified time in microseconds.

*EVENT_HANDLER_T

Description

A Client application implementation of the API event notification handler. This function is called by the ***ad9164_isr*** API to send notification of events to the application.

Event notification shall depend on which interrupts have been enable via the ***ad9164_set_events*** API. Full details of the available events are listed in the Interrupt Handling section and the target DAC device datasheet.

Synopsis

typedef int(*event_handler_t)(uint16_t event, uint8_t ref, void* data);

Preconditions

It is expected that there are no pre conditions to this function. All initialization required shall be performed prior to or during (via ****hw_open_t***) ***ad9164_init.*** Once a DAC device is initialized via the ***ad9164_init*** API, it is expected that the API may call this function at any time.

Post conditions

It is expected that there are no post conditions to this function.

Dependencies

Unknown- Platform Implementation

Parameters

uint16_t event	A uint16_t value representing the event that has been detected. Event types are enumerated by ***ad9164_event_t***. Addition information is detailed in the ***Interrupt Events*** section.
uint8_t ref	A uint8_t value that represents the event reference data if required. For example for Link events, the reference data shall be the Lane number on which the event occurred. Addition information is detailed in the ***Interrupt Events*** section.
void* data	A pointer to any additional event data that may be relevant. Currently none of the AD9164 API Events provides any event data. Addition information is detailed in the ***Interrupt Events*** section.

Return value

Zero shall indicate success.

Any other positive integer value may represent a error code to be returned to the application.

Notes

This is required for optimal performance.

Performs a blocking or sleep delay for the specified time in microseconds.

ERROR HANDLING

ERROR CODES

Each API return value represents a DAC API error code. The possible error codes for ADI device APIs are defined by a number of macros listed in ***api_errors.h.*** Table 5 lists the possible error codes and their meanings returned by the AD9164 APIs.

If a HAL function, called by during the execution of an API, returns a non-zero value the API shall return an error code indicating that there was an error returned from particular HAL function. Table 6 lists the possible errors returned by API due to a HAL function.

Table 5 API Error Code Macro definitions.

ERROR CODE	**Description**
API_ERROR_OK	API completed successfully
API_ERROR_SPI_SDO	API could not complete success fully due to SPI_SDO configuration in API Handle
API_ERROR_INVALID_HANDLE_PTR	API could not complete success fully due to invalid pointer to API Handle
API_ERROR_INVALID_XFER_PTR	API could not complete success fully due to invalid pointer to SPI transfer function
API_ERROR_INVALID_DELAYUS_PTR	API could not complete success fully due to invalid pointer to Delay function
API_ERROR_INVALID_PARAM	API could not complete successfully due to invalid API parameter
API_ERROR_FTW_LOAD_ACK	API could not complete successfully due to Frequency Turning Word No-ACK
API_ERROR_NCO_NOT_ENABLED	API could not complete successfully due to NCO not currently Enabled
API_ERROR_INIT_SEQ_FAIL	API could not complete successfully due to NVRAM load error.

Table 6 API HAL function Error Code Macro definitions.

ERROR CODE	**Description**
API_ERROR_SPI_XFER	SPI HAL function return an error during the implementation of this API
API_ERROR_US_DELAY	DELAY HAL function return an error during the implementation of this API
API_ERROR_TX_EN_PIN_CTRL	TX_ENABLE pin ctrl HAL function return an error during the implementation of this API
API_ERROR_RESET_PIN_CTRL	RESET pin ctrl HAL function return an error during the implementation of this API
API_ERROR_EVENT_HNDL	EVENT Handle HAL function return an error during the implementation of this API
API_ERROR_HW_OPEN	HW Open HAL function returned an error during the implementation of this API
API_ERROR_HW_CLOSE	HW Close HAL function returned an error during the implementation of this API

INTERRUPT HANDLING

The AD9164 API provides basic support for the AD9164 Interrupt feature. It provides an API, ***ad9164_set_events***, to enable and disable the desired interrupt to trigger and an API to check if interrupt has been triggered and the interrupt source ***ad9164_get_interrupt_status.***

Additionally, it provides a simple ISR API, ***ad9164_isr***, that will check for and clear any interrupts trigger and provide event notification to the client application. In order to use the ISR api the client application must provide the user with a notification handler function,****event_handler_t.*** The prototype of this function is defined in ***adi_def.h***

INTERRUPT EVENTS

Table 6 summarized the list of events that the API interrupt feature supports. Events are categorized as device events that can only occur per device or JESD link events that can occur at per link per lane level. For JESD link the ISR, ad9164_isr*,* shall report via the *ref* parameter the lane number on which the event.

Table 7 API Event Summary.

EVENT	TYPE	Reference	DATA	DESCRIPTION
EVENT_SYSREF_JITTER	Device Event	None	None	SYSREF JITTER Threshold Event
EVENT_DATA_RDY	Device Event	None	None	DATA Ready Event
EVENT_JESD_LANE_FIFO_ERR	Device Event	None	None	JESD LANE FIFO Underflow/overflow Event
EVENT_JESD_PRBS_IMG_ERR	Device Event	None	None	PRBS Q Error Event
EVENT_JESD_PRBS_REAL_ERR	Device Event	None	None	PRBS I Error Event
EVENT_JESD_ILAS_ERR	Device Event	None	None	ILAS Configuration Mismatch Event
EVENT_JESD_BAD_DISPARITY_ERR	JESD Link Event	Lane No	None	Bad Disparity Error Event
EVENT_JESD_NOT_IN_TBL_ERR	JESD Link Event	Lane No	None	Not-In-Table Error Event
EVENT_JESD_K_ERR	JESD Link Event	Lane No	None	Unexpected K Error Event
EVENT_JESD_ILD_ERR	JESD Link Event	Lane No	None	Inter-lane De-Skew Event
EVENT_JESD_ILS_ERR	JESD Link Event	Lane No	None	Good Checksum Event
EVENT_JESD_CKSUM_ERR	JESD Link Event	Lane No	None	Frame Synchronization Event
EVENT_JESD_CGS_ERR	JESD Link Event	Lane No	None	Code Group Synchronization Event

AD9164 API LIBRARY

AD9164 API REFERENCE HANDLE

AD9164_HANDLE_T

Description

DAC Device reference handle data structure that acts a software reference to a particular instance to of the DAC device. This reference maintains a reference to HAL functions and the configuration status of the chip.

Synopsis

#include AD9164.h

typedef struct {

void *user_data;

spi_sdo_config_t sdo;

uint64_t dac_freq_hz;

spi_xfer_t dev_xfer;

delay_us_t delay_us;

event_handler_t event_handler;

tx_en_pin_ctrl_t tx_en_pin_ctrl;

reset_pin_ctrl_t reset_pin_ctrl;

hw_open_t hw_open;

hw_close_t hw_close;

}ad9164_handle_t;

Members

void *user_data;

A void pointer that acts a container structure for client application data to be provided to the HAL functions. The client application must define this structure as per its platform requirements. This pointer shall be passed as the parameter to the ***hw_open_**t and ***hw_close_t** function calls to pass any client application specific data. The DAC API shall not access this data directly. The client application must initialize this member appropriately prior to calling any DAC API function.

spi_sdo_config_t sdo;

This member hold the desired SPI interface configuration, 3-wire or 4-wire, for the AD9164 DAC in the client system.

spi_sdo_config_t enumerates this configuration and is defined in ***api_def.h.*** This member should be correctly configured prior to calling ***ad9164_init*** in order to ensure SPI access to the AD9164.

uint64_t dac_freq_hz ;

This member holds the frequency of the DAC CLK provided to the AD9164 DAC in the target system.

It is important to configure this variable correctly prior to configuring the AD9164 with operational modes such as NCO, JESD and data path as this value is used as reference for internal

This value should be access via the ad9164***_dac_set_clk_frequency*** API and the ***ad9164_dac_get_clk_frequency.***

spi_xfer_t dev_xfer;

A function pointer to the client application defined SPI HAL function refer to ***spi_xfer_t** section for a detailed definition of this function. The client application must initialize this member appropriately prior to calling any DAC API function.

delay_us_t delay_us;

A function pointer to the client application defined microsecond delay. HAL function refer to *** delay_us_t** section for a detailed definition of this function. The client application must initialize this member appropriately prior to calling any DAC API function.

event_handler_t event_handler;

A function pointer to the client application's implementation of the event notification handler.

HAL function refer to ***event_handler_t** section for a detailed definition of this function.

tx_en_pin_ctrl_t tx_en_pin_ctrl;

A function pointer to the client application's implementation of TX_ENABLE pin control function, refer to ***** ***tx_en_pin_ctrl_t*** section for a detailed definition of this function.

reset_pin_ctrl_t reset_pin_ctrl;

A function pointer to the client application's implementation of RESETB pin control function, refer to ****reset_pin_ctrl_t*** section for a detailed definition of this function.

hw_open_t hw_open;

A function pointer to the client application HAL resources initialization function refer to ****hw_open_t*** section for a detailed definition of this function. The client application must initialize this member appropriately prior to calling any DAC API function.

hw_close_t hw_close;

A function pointer to the client application HAL resources de-initialization function refer to ****hw_close_t*** section for a detailed definition of this function. The client application must initialize this member appropriately prior to calling any DAC API function.

AD9164 API DEFINITIONS, DATA STRUCTURES AND ENUMERATIONS

This section describes all the structures and enumerations defined by the DAC API interface.

AD9164_CHIP_ID_T

Description

A structure detailing the AD9164 Device Identification Data. Please refer to the specific DAC Data sheet for expected values.

Synopsis

```
#include AD9164.h
typedef struct {
        uint8_t chip_type;
        uint16_t prod_id;
        uint8_t prod_grade;
        uint8_t dev_revision;
}ad9164_chip_id_t;
```

Fields

uint8_t chip_type	Chip Type
uint16_t prod_id	Product ID code
uint8_t prod_grade	Product Grade
uint8_t dev_revision	Silicon Revision.

Notes

Product ID and product grade are only available after initialization.

AD9164_TX_ENABLE_PIN_MODE_T

Description

An enumeration of the configuration options for the TX_ENABLE pin functionality.

Synopsis

```
#include AD9164.h
typedef enum {
        NONE,
        RESET_NCO,
        ZERO_DATA_DAC,
        ZERO_DATA_IN_PATH,
        CURRENT_CONTROLL
}ad9164_tx_enable_pin_mode_t;
```

Fields

NONE	No functionality is assigned to the TX_ENABLE pin
RESET_NCO	NCO is reset based on status of TX_ENABLE pin
ZERO_DATA_DAC	Data to DAC is zeroed based on status of TX_ENABLE pin
ZERO_DATA_IN_PATH	Data in data path is zeroed based on status of TX_ENABLE pin.
CURRENT_CONTROL	Full scale current control is influenced by TX_ENABLE pin.

Notes

AD9164_HOPF_MODE_T

Description

An enumeration of the configuration options for the Frequency Hopping

Synopsis

```
#include AD9164.h
typedef enum {
        ad9164_PHASE_CONTINOUS = 0x00,
        ad9164_PHASE_NONCONTINOUS = (1u << 6),
        ad9164_PHASE_COHERENT = (2u << 6),
        ad9164_FFH_MODE_MASK = (3u << 6)
}ad9164_hopf_mode_t;
```

Fields

ad9164_PHASE_CONTINOUS

ad9164_PHASE_NONCONTINOUS

ad9164_PHASE_COHERENT

ad9164_FFH_MODE_MASK

Notes

AD9164_EVENT_T

Description

An enumeration of the configuration options for the Frequency Hopping

Synopsis

```
#include AD9164.h
typedef enum {
        EVENT_SYSREF_JITTER,
        EVENT_DATA_RDY,
        EVENT_JESD_LANE_FIFO_ERR,
        EVENT_JESD_PRBS_IMG_ERR,
        EVENT_JESD_PRBS_REAL_ERR,
        EVENT_JESD_BAD_DISPARITY_ERR,
        EVENT_JESD_NOT_IN_TBL_ERR,
        EVENT_JESD_K_ERR,
        EVENT_JESD_ILD_ERR,
        EVENT_JESD_ILS_ERR,
        EVENT_JESD_CKSUM_ERR,
        EVENT_JESD_FS_ERR,
        EVENT_JESD_CGS_ERR,
        EVENT_JESD_ILAS_ERR,
        NOF_EVENTS

} ad9164 event_t;
```

Fields

EVENT_SYSREF_JITTER

EVENT_DATA_RDY

EVENT_JESD_LANE_FIFO_ERR,

EVENT_JESD_PRBS_IMG_ERR,

EVENT_JESD_PRBS_REAL_ERR,

EVENT_JESD_BAD_DISPARITY_ERR,

EVENT_JESD_NOT_IN_TBL_ERR,

EVENT_JESD_K_ERR,

EVENT_JESD_ILD_ERR,

EVENT_JESD_ILS_ERR,

EVENT_JESD_CKSUM_ERR,

EVENT_JESD_FS_ERR,

EVENT_JESD_CGS_ERR,

EVENT_JESD_ILAS_ERR,

NOF_EVENTS

Notes

AD9164 _JESD_LINK_STAT_T

Description

A structure of the JESD Interface link Status

Synopsis

#include AD9164.h

typedef struct {

uint8_t code_grp_sync_stat;

uint8_t frame_sync_stat;

uint8_t good_checksum_stat;

uint8_t init_lane_sync_stat;

}ad9164_jesd_link_stat_t;

Members

uint8_t code_grp_sync_stat;

A uint8_t bit wise representation of Code Group Sync Status for all JESD Lanes. Where bit 0 represents CGS status for Lane 0 and bit 1 represents CGS status for Lane 1 etc. A value of 1 indicates CGS status is complete, a value of 0 indicates CGS failed.

uint8_t frame_sync_stat;

A uint8_t bit wise representation of Frame Sync Status for all JESD Lanes. Where bit 0 represents Frame Sync status for Lane 0 and bit 1 represents Frame Sync status for Lane 1 etc. A value of 1 indicates Frame Synchronization status is complete, a value of 0 indicates Frame Synchronization failed.

uint8_t good_checksum_stat;

A uint8_t bit wise representation of Good Checksum Status for all JESD Lanes. Where bit 0 represents Checksum status for Lane 0 and bit 1 represents checksum status for Lane 1 etc. A value of 1 indicates valid checksum status, a value of 0 indicates checksumfailed.

uint8_t init_lane_sync_stat;

A uint8_t bit wise representation of Initial Lane Synchronization Status for all JESD Lanes. Where bit 0 represents Lane Synchronization status for Lane 0 and bit 1 represents Lane Synchronization status for Lane 1 etc. A value of 1 indicates Lane Synchronization completed, a value of 0 Lane Synchronization failed.

Notes

AD9164 _JESD_SYSREF_MODE_T

Description

A enumeration of the JESD SYSREF modes.

Synopsis

```
#include AD9164.h

typedef enum {
        SYSREF_NONE,
        SYSREF_ONESHOT,
        SYSREF_CONT
        SYSREF_MON,
        SYSREF_MODE_INVLD
}ad9164_jesd_sysref_mode_t;
```

Members

SYSREF_NONE	No SYSREF synchronization.
SYSREF_ONESHOT	One shot SYSREF Mode
SYSREF_CONT	Continous SYSREF Synchronization mode.
SYSREF_MON	SYSREF Monitor Mode
SYSREF_MODE_INVLD	Invalid or unknown SYSREF Mode.

Notes

AD9164_JESD_CFG_ERR_FLG_T

Description

An enumeration of JESD configuration error flags.

Synopsis

```
#include AD9164.h
typedef enum
{
        INTPL_FACTOR_INVLD = 0x1,
        SUBCLASS_V_INVLD = 0x2,
        K_INVLD = 0x4,
        LMFS_INPOL_INVLD = 0x8,
        LMFC_DELAY_INVLD = 0x10,
        SYSREF_JTTR_WIN_INVLD =0x20
}ad9164_jesd_cfg_err_flg_t;
```

Members

INTPL_FACTOR_INVLD	Invalid interpolation factor set.
SUBCLASS_V_INVLD	Illegal Subclass V set
K_INVLD	Illegal K set.
LMFS_INPOL_INVLD	Unsupported LMFS combination set for selected Interpolation factor
LMFC_DELAY_INVLD	Illegal LMFC delay set
SYSREF_JTTR_WIN_INVLD	Illegal Sysref Jitter window set.

Notes

AD9164_JESD_SERDES_PLL_FLG_T

Description

An enumeration of the SERDES PLL Status flags.

Synopsis

```
#include AD9164.h
typedef enum
{
        PLL_LOCK_STAT = 0x1,
        PLL_CAL_STAT = 0x8,
        PLL_UPP_CAL_THRES =0x10,
        PLL_LWR_CAL_THRES =0x20
}ad9164_jesd_serdes_pll_flg_t;
```

Members

PLL_LOCK_STAT	SERDES PLL lock Status Flag. When set the PLL is locked.
PLL_CAL_STAT	SERDES PLL Calibration Status Flag. When set the PLL is calibrated.
PLL_UPP_CAL_THRES	SERDES PLL Upper Calibration Threshold flag.When set the PLL calibration upper threshold was crossed. Re-calibration required.
PLL_LWR_CAL_THRES	SERDES PLL lower Calibration Threshold flag. When set the PLL calibration lower threshold was crossed. Re-calibration required.

Notes

JESD_PRBS_PATTERN_T

Description

An enumeration of all available PRBS patterns.

Synopsis

#include AD9164.h

typedef enum

{

PRBS_NONE,

PRBS7,

PRBS15,

PRBS31,

PRBS_MAX

}jesd_prbs_pattern_t;

Members

PRBS_NONE	No PRBS patterned enabled. PRBS off.
PRBS7	PRBS7 pattern
PRBS15	PRBS15 pattern
PRBS31	PRBS31 pattern
PRBS_MAX	Number of member in this enum. Number of PRBS pattern options.

Notes

AD9164_PRBS_TEST_T

Description

A structure of the parameters that describe the results of a PRBS test on the JESD interface.

Synopsis

#include AD9164.h

typedef struct

{

uint32_t phy_prbs_err_cnt;

uint8_t phy_prbs_pass;

uint8_t phy_src_err_cnt;

}ad9164_prbs_test_t;

Members

uint32_t phy_prbs_err_cnt

A uint32 value describing the number of PRBS errors detected on the link.

uint8_t phy_prbs_pass

A uint8 value inidicating the sucess or failure of the test. A 0 indicates a failure 1 indicates a pass.Any other value is invalid.

uint8_t phy_src_err_cnt

A uint8 value describing the source error count.

Notes

AD9164 APIS

AD9164_INIT

Description

API to initialize the DAC Module

This API must be called first before any other API calls. It performs internal API initialization of the memory and API states.

If DAC API handle member hw_open is not NULL the function to which it points shall be called to initialize DAC external resources. For example GPIO, SPI etc.

This function shall complete any universal initialization configuration of the DAC such as SPI.

Synopsis

#include AD9164.h

ADI_API int ad9164_init(ad9164_handle_t *h);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.

Preconditions

If ***hw_open*** function pointer is set to NULL. DAC external hardware resources must be initialized.

Post conditions

On successful completion, DAC Module shall be in initialized state, ready for configuration.

Note although the API is now initialized, it is recommended to call the Ad9164_init API immediately after call int Ad9164_init API to ensure all SPI register setting are restore to default and ADI recommendations.

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

h->hw_open DAC API handle optionally may be set to valid hardware initialization function for client application. Refere to ****hw_open_t.***

Return value

Any other positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

API_ERROR_HW_OPEN_FAILED

Notes

AD9164_DEINIT

Description

Shutdown the DAC Module

This API must be called last. No other API should be called after a call to this API.

It performs internal API initialization of the memory and API states and ensure targeted DAC is in good state for power down. If DAC API handle member ***hw_close_t*** is not NULL the function to which it points shall be called to de-initialize DAC external resources. This function may be used to de-initialize and release and hardware resources required by the API and AD9164 Device, for example GPIO SPI etc.

Synopsis

#include AD9164.h

ADI_API int ad9164_deinit(ad9164_handle_t *h);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

On successful completion, DAC Module shall be in shutdown state. DAC module shall need to be re-initialized via ***ad9164_init*** by client before any further use.

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

h->hw_close DAC API handle optionally may be set to valid hardware initialization function for client application. Refere to ***hw_close_t***.

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_HANDLE_PTR

Notes

AD9164_RESET

Description

Performs a full reset of AD9164 DAC, resetting all SPI registers to their default values, restoring the desired SPI configuration and ADI recommended initialization sequence.

This API can trigger a software reset via a SPI control or trigger a hardware reset by toggling the RESETB hardware pin. In order to trigger a hardware reset the API must be provided with a client defined HAL function, ***reset_pin_ctrl_t,*** that provides the API with control over the RESETB pin on the client application.

The type of reset triggered by the API is determined by the hw_reset parameter.

Synopsis

#include AD9164.h

ADI_API int ad9164_reset(ad9164_handle_t *h, uint8_t hw_reset);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t hw_reset	a uint8_t value to indictate the type of reset to be triggered. A value of 1 indicates a hardware reset is to be triggered. A value of 0 indicates a soft reset is to be triggered.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init***API. Prior to using this function.

Post conditions

The DAC shall be fully reset followed with reconfiguration of SPI interface and ADI recommended initialization sequence.

Dependencies

h->dev_xfer.

DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

h->reset_pin_ctrl_t.

If hw reset is desired as indictated by the API parameter hw_reset the DAC API handle must be initialized to valid function that controls the RESETB for the client application. Refer to ***reset_pin_ctrl_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_SPI_SDO

API_ERROR_INVALID_XFER_PTR

Notes

AD9164_GET_CHIP_ID

Description

API to retriev ADI chip identification, product type and revision data.

Synopsis

#include AD9164.h

ADI_API int ad9164_get_chip_id(ad9164_handle_t *h, ad9164_chip_id_t *chip_id);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
ad9164_chip_id_t *chip_id	Pointer to a variable of type ***ad9164_chip_id_t*** to which the Device Identication data shall be stored.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_SET_EVENTS

Description

An API to set the events that trigger the interrupt signal on the AD9164 Device. Available interrupt events are lited by the **ad9164_event_t** enumeration. Interrupt events may enabled or disabled using this API.

Synopsis

#include AD9164.h

ADI_API int ad9164_set_events(ad9164_handle_t *h,
ad9164_event_t *event_list, uint8_t list_size, uint8_t en)

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
event_list	Pointer to an array of type **ad9164_event_t** that represents a list of events to be enable or disabled based or the **en** parameter.
list_size	An 8-bit value representing the size of the array to which **event_list** parameter points, the number or events to be enabled/disabled.
en	A enable/disable variable to indicate whether the events listed by event_list array shall be enabled or disabled. A value of 0 disables the interrupt events. A value of 1 enables the interrupt events.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API. Prior to using this function.

Post conditions

Following a call to this API the configuration of the interrupt signal shall trigger when any of the enabled events are detected by the AD9164 device.

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_GET_INTERRUPT_STATUS

Description

An API to get the interrupt controller status to indicate if interrupts have triggered and are pending based on SPI register status.

Synopsis

#include AD9164.h

ADI_API int ad9164_get_interrupt_status(ad9164_handle_t *h, uint8_t *int_pending, uint8_t *int_status);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
int_pending	Pointer to an uint8_t variable where the interrupt status shall be stored. A value of 1 indicates at least one interrupt event has occurred. A value of 0 indicates not interrupt events have occurred.
int_status	Pointer to a 3-deep uint8_t array to which the current status of the interrupt status registers s shall be stored. May be set to NULL if this data is not required.

Preconditions

The AD9164 device shall be success fully initialized via a call to the **ad9164_init** API. Prior to using this function.

Post conditions

None

Dependencies

h->**dev_xfer.** AD9164 API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

Note this API does not clear any interrupts detected.

AD9164_ISR

Description

An API that checks for and handle triggered interrupts, checks the status and can notify the API user of the events and any relevant status. In order to for the API user to be notified of events by this API the AD9164 Handle must be initialized with a valid an event notification. Refer to ***event_handler_t*** for more details. If an event handler function is not provided the API will simply clear any interrupts detected without notifying the application.

Synopsis

#include AD9164.h

ADI_API int ad9164_isr(ad9164_handle_t *h)

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.

Preconditions

The DAC device shall be success fully initialized via a call to the **ad9164_init** API. Prior to using this function.

Post conditions

All detected interrupts shall be cleared.

Dependencies

h->dev_xfer	DAC API handle shall be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***
h->event_handler	DAC API handle may be initialized to a valid event notification function for the client application handler. Refer to ***event_handler_t** section for more details.

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_DAC_SET_CLK_FREQUENCY

Description

Set the API software reference for the value of the hardware DAC clock supplied to the DAC device.

The correct value must be supplied for correct operation of the DAC features.

Synopsis

#include AD9164.h

ADI_API int ad9164_dac_set_clk_frequency(ad9164_handle_t *h, uint64_t dac_clk_freq_hz);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint64_t dac_clk_freq_hz	DAC clock frequency in Hz. Valid range is 850MHz to 6GHz.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.**

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_DAC_GET_CLK_FREQUENCY

Description

Set the API software reference for the value of the hardware DAC clock supplied to the DAC device.

The correct value must be supplied for correct operation of the DAC features.

Synopsis

#include AD9164.h

ADI_API int ad9164_dac_get_clk_frequency(ad9164_handle_t *h, uint64_t *dac_clk_freq_hz);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint64_t *dac_clk_freq_hz	Pointer to a uint64_t variable where the current DAC clock frequency value setting whall be stored. The frequency value shall be provided in Hx. The valid range is 850MHz to 6GHz

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.**

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_DAC_SET_FULL_SCALE_CURRENT

Description

Adjust the DAC's full-scale current output value; this can be adjusted over a range 8mA to 40mA.

Synopsis

#include AD9164.h

ADI_API int ad9164_dac_set_full_scale_current(ad9164_handle_t *h, uint8_t current);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t current	Desired full scale output current in mA. Valid range 8 to 40.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->**dev_xfer.** DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_CONFIG_DATAPATH

Description

Configure the JESD DAC mode as per the desired JESD interface parameters and datapath, the DAC clk frequency and the interpolation rate.

The JESD lane rate for the configuration is calculated and returned via the lane_rate_MBPS parameter.

The API shall check the parameter values and return an error if the desired JESD interface is not supported for the desired DAC mode. Refer to the DAC datasheet for full details on the JESD configurations and DAC modes supported.

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_config_datapath(ad9164_handle_t *h, jesd_param_t jesd_param,

uint8_t interpolation, uint64_t *lane_rate_mbps);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
jesd_param_t jesd_param,	A variable of jesd_param_t describing the desire JESD link parameters. Refer to the ***jesd_param_t*** description for full details on the JESD Parameters and the DAC datasheet for supported JESD modes.
uint8_t interpolation	A uint8_t value representing the desired DAC interpolation mode. Valid interpolation modes are 1,2,3,4,6,8,12,16
uint64_t *lane_rate_mbps	A uint64_t pointer to which the calculated JESD lane rate based on the desired DAC and JESD interface parameters shall be returned. Set to NULL if lane_rate not required.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer.	DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***
h->dac_freq_hz	DAC Clock value must be initialized to the correct value as per the hardware setting for correct operation of this API. Refer to ***ad9164_handle_t*** for more details on dac_freq_hz

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_GET_CFG_PARAM

Description

API to return the all the current JESD Parameters.

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_get_cfg_param(ad9164_handle_t *h, jesd_param_t *jesd_param);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
jesd_param_t *jesd_param	Pointer to a structure of type ***jesd_param_t*** to which the all the JESD parameters currently configured will be stored.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_SET_SYSREF_MODE

Description

Configure the SYSREF synchronization mode for the JESD Interface

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_set_sysref_mode(ad9164_handle_t *h, jesd_sysref_mode_t mode, uint8_t jitter_window)

Parameters

ad9164_handle_t *h Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.

jesd_sysref_mode_t mode The format for the SYSREF synchronization signal for the JESD Interface to the AD9164 Device. Valid modes are enumerated by ***jesd_sysref_mode_t***

SYSREF_NONE	No SYSREF synchronization.
SYSREF_ONESHOT	One shot SYSREF Mode
SYSREF_CONT	Continous SYSREF Synchronization mode.
SYSREF_MON	SYSREF Monitor Mode

uint8_t jitter_window a uint8_t variable to which SYSREF's jitter window tolerance value shall be stored.

Valid values and their respective tolerance in

DAC clock_cycles are as follows:

0 => +/- 0.5 DAC Clock Cycles

4 => +/- 4 DAC Clock Cycles

8 => +/- 8 DAC Clock Cycles

12 => +/- 16 DAC Clock Cycles

12 => +/- 16 DAC Clock Cycles

20 => +/- 20 DAC Clock Cycles

24 => +/- 24 DAC Clock Cycles

28 => +/- 28 DAC Clock Cycles

Valid only in Monitor Mode. This parameter ignored in all other modes.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_GET_SYSREF_MODE

Description

Configure the SYSREF synchronization mode for the JESD Interface

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_get_sysref_mode(ad9164_handle_t *h, jesd_sysref_mode_t *mode, uint8_t *jitter_window)

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
jesd_sysref_mode_t *mode	Pointer to which the current SYSREF synchronization signal configuration for the JESD Interface shall be stored. Valid modes are enumerated by ***jesd_sysref_mode_t***

SYSREF_NONE	No SYSREF synchronization.
SYSREF_ONESHOT	One shot SYSREF Mode
SYSREF_CONT	Continous SYSREF Synchronization mode.
SYSREF_MON	SYSREF Monitor Mode (Not fully supported yet)

uint8_t jitter_window Pointer to a uint8_t variable to shich the the Sysref Window Tolerance value shall be stored.

Valid values and their respective tolerance in

DAC clk cycles are as follows:

0 => +/- 0.5 DAC Clock Cycles

4 => +/- 4 DAC Clock Cycles

8 => +/- 8 DAC Clock Cycles

12 => +/- 16 DAC Clock Cycles

12 => +/- 16 DAC Clock Cycles

20 => +/- 20 DAC Clock Cycles

24 => +/- 24 DAC Clock Cycles

28 => +/- 28 DAC Clock Cycles

Valid only in Monitor Mode. This parameter ignored in all other modes.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_GET_SYSREF_STATUS

Description

Read back SYSREF and LMFC synchronization status data.

API to return the current synchronization data for the SYSREF synchronization signal and LMFC synchronization.

This data can be used to determine dynamic link latency settings in SYSREF Monitor Mode.

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_get_sysref_status(ad9164_handle_t *h,

uint8_t *sysref_count, uint16_t *sysref_phase, uint16_t *sync_lmfc_stat);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t *sysref_count	Pointer to the variable where the detected SYSREF signal count shall be stored. Set to NULL if read back data is not required.
uint16_t *sysref_phase	Pointer to the variable where the phase used to sample the SYSREF signals shall be stored. Set to NULL if read back data is not required.
uint16_t *sync_lmfc_stat 4	Pointer to the variable where the LMFC status shall be stored. This value may be used to determine synchronization status. This value may be used to determine synchronization status. A value of 0 to indicates valid synchronization. Set to NULL if this read back data is not required.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_GET_DYNAMIC_LINK_LATENCY

Description

API to read back the Dynamic Link Latency.

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_get_dynamic_link_latency(ad9164_handle_t *h, uint8_t *latency);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t *latency	Pointer to the variable where the dynamic link latency in units of pclks shall be stored.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API prior to using this function.

Post conditions

None

Dependencies

h->**dev_xfer.** DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_SET_DYNAMIC_LINK_LATENCY

Description

API to adjust the LMFC Delay and LMFC variance.

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_set_lmfc_delay(ad9164_handle_t *h, uint8_t delay, uint8_t var);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t delay	A 5-bit value to set the desired global LMFC delay in units of Frame Clock Cycles.
uint8_t delay	A 5-bit value to set the desired variable LMFC delay.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.**

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_SET_LANE_XBAR

Description

Configure the JESD DAC mode as per the desired JESD interface parameters and data path, the DAC clock frequency and the interpolation rate.

The JESD lane rate for the configuration is calculated and returned via the lane_rate_mbps parameter.

The API shall check the parameter values and return an error if the desired JESD interface is not supported for the desired DAC mode. Refer to the DAC datasheet for full details on the JESD configurations and DAC modes supported.

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_set_lane_xbar(ad9164_handle_t *h, uint8_t physical_lane, uint8_t logical_lane)

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t physical_lane	uint8_t value representing the Physical Lane to be routed to the SERDES logical lane indicated by the *logical_lane* parameter. Valid values are 0 to 7.
uint8_t logical_lane	uint8_t value representing the SERDES logical lane for the physical lane indicated by the parameter *physical_lane*. Valid values are 0 to 7.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API. Prior to using this function. This API shall be called to configure the Lane mapping prior to enabling the JESD link.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_GET_LANE_XBAR

Description

API to get the current JESD lane crossbar configuration for the data-link layer. Returns an 8 deep array of values representing the list logical lanes assigned to each physical lane.

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_get_lane_xbar(ad9164_handle_t *h, uint8_t *phy_log_map);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t *phy_log_map	Pointer to an 8-deep array of values representing the list logical lanes assigned to each physical lane. The index of the array represents the physical lane (0 to 7) and the value at that index the logical lane (0 to 7) assigned to that physical lane.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_INVERT_LANE

Description

Invert or un-invert logical lanes.

Each logical lane can be inverted which can be used to ease routing of SERDIN signals.

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_invert_lane(ad9164_handle_t *h, uint8_t logical_lane, uint8_t invert)

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t logical_lane	uint8_t value representing the SERDES logical lane for the physical lane indicated by the parameter *physical_lane*. Valid values are 0 to 7.
uint8_t invert	A uint8_t value to indicate the desired invert status for the logical lane listed by logical_lane parameter. A value of 1 inverts the logical lane. A value of 0 inverts the logical lane.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API. Prior to using this function. This API shall be called to configure the Lane mapping prior to enabling the JESD link.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.**

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_ENABLE_SCRAMBLER

Description

Enable or disable the descrambler for the JESD Receiver.

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_enable_scrambler(ad9164_handle_t *h, uint8_t en);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t en	Enable control for the JESD Scrambler. Set to 1 to enable the de-scrambler on the JESD Receiver; set to 0 to disable.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API. Prior to using this function. The JESD link should be not be enabled when calling this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_GET_CFG_STATUS

Description

API to return the JESD Configuration Error Flag Status. There are six error flag to indicate errors in the current JESD configuration. ***jesd_cfg_err_flg_t*** enumerates the error flags.

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_get_cfg_status(ad9164_handle_t *h, uint8_t *jesd_cfg_stat);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
Uint8_t jesd_cfg_stat	JESD configuration error status. Each bit in status byte represents an error flag as listed below and are enumerated by ***jesd_cfg_err_flg_t.***

INTPL_FACTOR_INVLD	Invalid interpolation factor set.
SUBCLASS_V_INVLD	Illegal Subclass V set
K_INVLD	Illegal K set.
LMFS_INPOL_INVLD	Unsupported LMFS combination set for selected Interpolation factor
LMFC_DELAY_INVLD	Illegal LMFC delay set
SYSREF_JTTR_WIN_INVLD	Illegal SYSREF Jitter window set.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_ENABLE_DATAPATH

Description

Enable the JESD Interface

Power up and enable the DAC's JESD Interface.

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_enable_datapath(ad9164_handle_t *h, uint8_t lanes_msk, uint8_t run_cal, uint8_t en);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t lanes_msk,	a uint8 bit wise value representing the lanes to be enabled on the JESD Interface. Where bit 0 represents lane 0, bit 1 represents lane 1 etc. Set to one to enable the respective JESD Lane, set to 0 to disable the respective JESD Lane.
uint8_t run_cal	Parameter to indicate if JESD physical lane calibration should be run prior to enabling interface. Set to 1 to run calibration, set to 0 to disable calibration.
uint8_t en	Enable control for the JESD interface. Set to 1 to powerup and enable JESD interface. Set to 0 to disable JESD interface.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API prior to using this function.

JESD interface should be successfully configured via ***ad9164_jesd_config_datapath*** prior to calling this API.

Post conditions

Following a call to this API the SERDES PLL should be lock. The PLL lock status can be checked by calling the

ad9164_jesd_get_pll_status API.

If the SERDES PLL is locked the AD9164 JESD Rx is ready for JESD Link bring up with a JESD Tx. Refer to

ad9164_jesd_enable_link.

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_ENABLE_LINK

Description

Enable the JESD Interface

Power up and enable the DAC's JESD Interface.

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_enable_link(ad9164_handle_t *h, uint8_t en);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t en	Enable control for the JESD Link Bringup. Set to 1 to enable SERDES Link bringup. Set to 0 to disable SERDES link.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API. Prior to using this function.

JESD interface should be successfully enabled via ***ad9164_jesd_enable_datapath*** prior to calling this API.

For successful link bring up the SERDES PLL shall be in a locked state prior to calling this API. This can be verified usint the ***ad9164_jesd_get_pll_status*** API.

In addition, the system JESD Tx shall be configured appropriately for successful link bring up.

Post conditions

Following a call to this API the JESD interface bring up shall be instigated. Link Status may be verified by a call to

ad9164_get_jesd_link_status.

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_GET_PLL_STATUS

Description

Get SERDES PLL Status.

Read and return the status data for the SERDES PLL

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_get_pll_status(ad9164_handle_t *h, uint8_t *pll_status);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t *pll_status	Pointer to a unit8 variable to which the PLL status shall be written.
	Bit [0] represents the PLL lock status.
	Bit [3] represents the PLL calibration status.
	Bit [4] represents the PLL upper calibration Threshold error status.
	Bit [5] represents the PLL lower calibration Threshold error status.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_JESD_GET_LINK_STATUS

Description

Get JESD Link Status.

Read back JESD Status for all lanes. Status reported includes Code Group Synchronization (CGS) status, Frame Synchronization status, checksum status and Initial Lane Synchronization (ILAS) status for the active JESD link. Refer to ***jesd_link_stat_t*** for more details.

Synopsis

#include AD9164.h

ADI_API int ad9164_jesd_get_link_status(ad9164_handle_t *h, jesd_link_stat_t *link_status);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
jesd_link_stat_t *link_status	Pointer to a variable of type ***jesd_link_stat_t*** that shall be set with the current JESD link status data.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API. Prior to using this function.

Post conditions

Follow a call to this API the SERDES PLL should be lock. The PLL lock status can be checked by calling the

Dependencies

h->**dev_xfer.** DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_TRANSMIT_ENABLE_PIN

Description

Configure TX_ENABLE pin functionality

TX_ENABLE pin can be configured so influence the data for tx-disable mode different points in the data path. Such as zero-ing data at the input to the data path, instigate NCO reset or zero-data an input to the DAC.

Synopsis

#include AD9164.h

ADI_API int ad9164_transmit_enable_pin(ad9164_handle_t *h, tx_enable_pin_mode_t tx_en_func);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
tx_en_func	Desired TX_ENABLE pin functionality. Defined by the enumeration

NONE	No functionality is assigned to the TX_ENABLE pin
RESET_NCO	NCO is reset based on status of TX_ENABLE pin
ZERO_DATA_DAC	Data to DAC is zeroed based on status of TX_ENABLE pin
ZERO_DATA_IN_PATH	Data in data path is zeroed based on status of TX_ENABLE pin.
CURRENT_CONTROL	Full scale current control is influenced by TX_ENABLE pin.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_TRANSMIT_ENABLE

Description

An API to control the transmission out from the DAC device. The transmission out from the device can be control by software via a SPI register control or the TX_ENABLE pin on the DAC device. How the output transmission is controlled is determined by the TX_ENABLE configuration. The user can configures this using ***ad9164_transmit_enable_pin*** API.

If TX_ENABLE pin is configured to ZERO_DATA_DAC or ZERO_DATA_IN_PATH modes and the API is provided with HAL function **** tx_en_pin_ctrl_t,*** the API shall use the hardware pin control to control the transmit output stream. Otherwise the API will control the transmit output stream via software controls.

Synopsis

#include AD9164.h

ADI_API int ad9164_transmit_enable(ad9164_handle_t *h, const ad9164_transmit_mode_t mode);

Parameters

ad9164_handle_t *h

Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.

ad9164_transmit_mode_t mode

Configure the DAC output transmission as per the desired functionality.

TX_ENABLE	Transmit functionality. Transmit all DAC data.
TX_ZERO_DATA_DAC	Zero data at DAC from transmission stream.
TX_ZERO_DATA_IN_PATH	Zero data at input to datapath.

Preconditions

The DAC device shall be success fully initialized via a call to the ***Ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer.	DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***
h->tx_en_pin_ctrl_t	DAC API handle must be initialized to valid HAL function to control the TX_ENABLE pin for the client application for Hardware control of the DAC

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_CLK_CFG_DUTYCYCLE

Description

Enable and configure Clock Duty Cycle Adjustment for the DAC clock applied to the CLK inputs.

Synopsis

#include AD9164.h

ADI_API int ad9164_clk_cfg_dutycycle(ad9164_handle_t *h, uint8_t en, uint8_t offset);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t en	Enable offset adjustment where the offset is determined by the parameter offset. Set to 1 to apply offset to the clock. Set to 0 disable offset adjustment.
uint8_t offset	Duty cycle offset parameter, should be set to a 5-bit two's complement value to indicate a positive or negative skew. A larger valuer will result in a larger clock duty skew.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_CLK_ADJUST_PHASE

Description

API to apply a phase adjustment of DAC Clock at the CLK+ input or at the CLK- input.

Synopsis

#include AD9164.h

ADI_API int ad9164_clk_adjust_phase(ad9164_handle_t *h, uint8_t pol, uint8_t phase);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t pol	Clock input polarity to which the phase adjustment is to be applied.
	Set to 1 to apply phase adjustment to CLK- input.
	Set to 0 to apply phase adjustment to CLK+ input.
uint8_t phase	Desired number of phase adjustment steps. Valid range for number of phase adjustments steps is 0 to 31. Each step adds a capacitance of 20ff.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.**

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_CLK_SET_CROSS_CTRL

Description

API to configure and enable or disable the clock cross control feature..

Synopsis

#include AD9164.h

ADI_API int ad9164_clk_set_cross_ctrl(ad9164_handle_t *h, uint8_t en, uint8_t crosspoint);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t en	Cross Control Enable. Set to 1 to enable Clock Cross Control; set to 0 to disable Clock Cross. Control.
uint8_t crosspoint	A 3 bit value representing the Clock cross control cross point.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_FIR85_SET_ENABLE

Description

Enable FIR-85 filter for 2-NRZ mode operation.

Synopsis

#include AD9164.h

ADI_API int ad9164_fir85_set_enable(ad9164_handle_t *h, const int en);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
int en	Enable FIR-85 filter 1 => Enable FIR-85 filter 0 => Disable FIR-85 filter

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_FIR85_GET_ENABLE

Description

Get FIR-85 filter for 2-NRZ mode operation enable status.

Synopsis

#include AD9164.h

ADI_API int ad9164_fir85_get_enable(ad9164_handle_t *h, int *en);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
int *en	Pointer to a variable to which the FIR-85 filter enable status shall be stored. 1 => Enable FIR-85 filter 0 => Disable FIR-85 filter

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_FILT_BW90_ENABLE

Description

Set Interpolation Filter Bw Mode to 90% BW.

By default the interpolation filter in the datapath are in a low power 80% bandwith mode. By enabling the high bandwith filter mode, the interpolation filter bandwidth will be set to 90% BW.

Synopsis

#include AD9164.h

ADI_API int ad9164_fir85_bw90_enable(ad9164_handle_t *h, cons tint en);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
int en	Enable high bandwidth mode, 90%, for interpolation filters. 1 => Set interpolation filters to 90% Bandwidth. 0 => Set interpolation filters to 80% Bandwidth.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_INVSINC_ENABLE

Description

Enable inverse sin c (sinc^{-1}) filter.

Synopsis

#include AD9164.h

ADI_API int ad9164_invsinc_enable(ad9164_handle_t *h, const int en);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
int en	Enable the inverse sin c filter. Set to 1 to enable filter; set to 0 to disable filter.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_NCO_SET_FTW

Description

The AD9164 family of DAC support a number of NCOs. This API is used to configure a particular NCO with Frequency Tuning Word, Modulus and Delta parameters.

Synopsis

#include AD9164.h

ADI_API int ad9164_nco_set_ftw(ad9164_handle_t *h, const unsigned int nco_nr, const uint64_t ftw,

const uint64_t acc_modulus, const uint64_t acc_delta);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
int nco_nr	Desired NCO to be configured. Valid values 0 to 31.
int64_t ftw	Frequency tuning word value. For NCO 0 this shall be a 48Bit value. For all other NCOs (1-31) this shall be a 32-bit value.
int64_t acc_modulus	Desired modulus.
uint16_t acc_delta	Desired delta value.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_FTW_LOAD_ACK

API_ERROR_INVALID_PARAM

Notes

AD9164_NCO_GET_FTW

Description

The AD9164 family of DAC support a number of NCOs. This API is used to get the NCO parameter Frequency Tuning Word, Modulus and Delta from a particular NCO.

Synopsis

#include AD9164.h

ADI_API int ad9164_nco_get_ftw(ad9164_handle_t *h, const unsigned int nco_nr,   uint64_t *ftw,

uint64_t *acc_modulus, uint64_t *acc_delta);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
int nco_nr	Desired NCO to from which to retrieve parameters. Valid values 0 to 31.
int64_t *ftw	Pointer to a variable where the frequency tuning word value will be stored For NCO 0 this shall be a 48-Bit value. For all other NCOs (1-31) this shall be a 32-bit value.
int64_t *acc_modulus	Pointer to a variable where the modulus value will be stored
uint16_t *acc_delta	Pointer to a variable where the delta value will be stored.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_NCO_SET_PHASE_OFFSET

Description

Set the NCO Phase offset.

Synopsis

#include AD9164.h

ADI_API int ad9164_nco_set_phase_offset(ad9164_handle_t *h, const uint16_t po);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
const uint16_t po	The phase offset value.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.**

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

Notes

AD9164_NCO_GET_PHASE_OFFSET

Description

Get the NCO Phase offset.

Synopsis

#include AD9164.h

ADI_API int ad9164_nco_get_phase_offset(ad9164_handle_t *h, const uint16_t *po);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
const uint16_t *po	Pointer to uint16_t variable where the current phase offset will be stored.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Postconditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_NCO_SET_HOPF_MODE

Description

The AD9164 family of DAC supports Fast Frequency Hopping. This sets and configures the device for Frequency Hopping Mode.

Synopsis

#include AD9164.h

ADI_API int ad9164_nco_set_hopf_mode(ad9164_handle_t *h, ad9164_hofp_mode_t mode);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
ad9164_hopf_mode_t mode	Desired Fast frequency Hopping Mode. Refer to ***ad9164_hopf_mode_t*** for a full description on the various modes.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent a error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_NCO_GET_HOPF_MODE

Description

The AD9164 family of DAC supports Fast Frequency Hopping. This sets and configures the device for Frequency Hopping

Synopsis

#include AD9164.h

ADI_API int ad9164_nco_get_hopf_mode(ad9164_handle_t *h, ad9164_hopf_mode_t *mode);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
ad9164_hopf_mode_t *mode	A pointer to a variable to which the currently configured Fast frequency Hopping mode shall be stored. Refer to ***ad9164_hopf_mode_t*** for a full description on the various modes.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent a error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_NCO_SET_ENABLE

Description

Set the enable status (Enable or Disable) of the main NCO and modulus.

Synopsis

#include AD9164.h

ADI_API int ad9164_nco_set_enable(ad9164_handle_t *h, const int modulus_en, const int nco_en);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
int modulus_en	Enable dual modulus NCO. 1 => Enable dual Modulus mode. 0 => Disable dual Modulus mode. NCO Integer Mode
int nco_en	Enable NCO. 1 = Enable NCO 0 = Disable NCO

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

For correct NCO operation it is expected that the NCO be correctly configured using

AD9164_NCO_GET_ENABLE

Description

Get the enable status (Enable or Disable) of the main NCO and modulus.

Synopsis

#include AD9164.h

ADI_API int ad9164_noc_get_enable(ad9164_handle_t *h, int *modulus_en, int *nco_en);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
int *modulus_en	Pointer to int variable where the enable dual modulus mode status will be stored. Possible returned values are 1 => Enable dual Modulus mode. 0 => Disable dual Modulus mode. NCO Integer Mode
int *nco_en	Pointer to int variable where the enable NCO status will be stored. Possible returned values are 1 => Enable NCO 0 = >Disable NCO

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_DC_TEST_SET_MODE

Description

Set the DC test mode test-data and enable test mode

Synopsis

#include AD9164.h

ADI_API int ad9164_dc_test_set_mode(ad9164_handle_t *h, const uint16_t test_data, const int en);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
const uint16_t test_data	DC test data value.
const int en	DC test mode enable. A value of zero disables DC Test mode. Any other value enables DC Test mode. en ==0 => Disable DC Test mode en >0 => Enable DC Test mode

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.**

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_DC_TEST_GET_MODE

Description

Set the DC test mode test-data and enable test mode

Synopsis

#include AD9164.h

ADI_API int ad9164_dc_test_get_mode(ad9164_handle_t *h, uint16_t *test_data, int *en);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
const uint16_t *test_data	Pointer to uint16_t variable where the current DC test data value will be stored.
const int *en	Pointer to an integer variable where the current DC test mode enable status will be stored. en ==0 => Disable DC Test mode en >0 => Enable DC Test mode

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Postconditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_NCO_SET_SELECT

Description

Select the desired NCO to be active.

Synopsis

#include AD9164.h

ADI_API int ad9164_nco_set_select(ad9164_handle_t *h, const unsigned int nco_nr);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
const unsigned int nco_nr	Desired NCO to be active. Valid values are from 0 to 31.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_NCO_GET_SELECTED

Description

Get the number of the currently selected NCO

Synopsis

#include AD9164.h

ADI_API int ad9164_nco_get_selected(ad9164_handle_t *h, unsigned int *nco_nr);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
const unsigned int *nco_nr	Pointer to a variable to which the NCO number that is currently selected to be active.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.**

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_NCO_SET

Description

Configure NCO based on desired DAC frequency and carrier frequency.

Synopsis

#include AD9164.h

ADI_API int ad9164_nco_set(ad9164_handle_t *h, const unsigned int nco_nr,
const int64_t carrier_freq_hz, const uint16_t amplitude, int dc_test_en);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
const unsigned int nco_nr,	A value to indicate desire NCO. Valid range 0 to 31.
const int64_t carrier_freq,	Desired carrier frequency
const uint16_t amplitude,	Desire amplitude.
int dc_test_en	Enable Test mode.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.**

h->dac_freq_hz DAC Clock value must be initialized to the correct value as per the hardware setting for correct operation of this API.

Refere to **dac_freq_hz.**

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_NCO_GET

Description

Get NCO frequency parameters currently configure on a particular NCO

Synopsis

#include AD9164.h

ADI_API int ad9164_nco_get(ad9164_handle_t *h, const unsigned int nco_nr, const int64_t *carrier_freq,
const uint16_t amplitude, int *dc_test_en);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
const unsigned int nco_nr,	A value to indicate desire NCO. Valid range 0 to 31.
const int64_t carrier_freq,	Pointer to a variable where the currently configured carrier frequency for the target NCO is stored.
const uint16_t amplitude,	Pointer to a variable where the currently configured DAC amplitude.
int dc_test_en	Pointer to a variable where the DC-Test mode status is stored.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ****spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_HANDLE_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_NCO_RESET

Description

Instigates reset of NCO via a SPI register control.

Synopsis

#include AD9164.h

ADI_API int ad9164_nco_reset(ad9164_handle_t *h);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_REGISTER_WRITE

Description

Performs SPI register write access to DAC

Synopsis

#include AD9164.h

ADI_API int ad9164_register_write (ad9164_handle_t *h, const uint16_t address, const uint8_t data);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
const uint16_t address,	A 15-bit value representing a SPI register location on the target DAC device. Refer to AD9164 data sheet for valid values.
uint8_t *data	A pointer to a 8-bit variable to which the register value read over SPI shall be stored.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Postconditions

None

Dependencies

h->dev_xfer. DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.**

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_REGISTER_READ

Description

DAC Device reference handle data structure that acts a sw reference to a particular instance to of the DAC device. This reference maintains a reference to HAL functions and the status of the chip.

Synopsis

#include AD9164.h

ADI_API int ad9164_register_read(ad9164_handle_t *h, const uint16_t address, uint8_t *data);

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
const uint16_t address,	A 15-bit value representing a SPI register location on the target DAC device. Refer to AD9164 data sheet for valid values.
uint8_t *data	A pointer to an 8-bit variable to which the register value read over SPI shall be stored.

Preconditions

The DAC device shall be success fully initialized via a call to the ***ad9164_init*** API. Prior to using this function.

Post conditions

None

Dependencies

h->**dev_xfer.** DAC API handle must be initialized to valid SPI transfer function for the client application. Refer to ***spi_xfer_t.***

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

Notes

AD9164_GET_REVISION

Description

Issue a soft reset via SPI Register control.

Performs a full reset of AD9164, resetting all SPI registers to their default values.

Synopsis

#include AD9164.h

ADI_API int ad9164_get_revision(ad9164_handle_t *h, uint8_t *rev_major, uint8_t *rev_minor, uint8_t *rev_rc)

Parameters

ad9164_handle_t *h	Pointer to the client application DAC API handle for the target DAC device. Refer to ***API Handle*** section for more details.
uint8_t *rev_major	Pointer to a 8 bit variable to which the Major Revision number shall be stored
uint8_t *rev_minor,	Pointer to a 8 bit variable to which the Minor Revision number shall be stored
uint8_t *rev_rc	Pointer to a 8 bit variable to which the Release Candidate Id shall be stored

Preconditions

None

Post conditions

None

Dependencies

None

Return value

Any positive integer value may represent an error code to be returned to the application, refer to ***Error Codes*** section for full details. . Possible return values for this API error codes are as follows.

API_ERROR_OK

API_ERROR_INVALID_PARAM

Notes

APPENDIX A

PSEUDO CODE EXAMPLE FOR AD9164 HANDLE

In the pseudo code example A, shows an example of the AD9164 handle being configure with the minimum client application. In the pseudo code example B, shows an example of the AD9164 handle being configured with all the optional configurations.

For further details, refer to the **DAC Hardware Initialization** section and the ***ad9164_handle_t*** section for full a description and more details on configuration.

```
/*
 * \brief pseudo Code example client code to initialiseAD916x
 *
 */
/*Include Client HAL implementation code*/
#include "app_hal.h"

/*Include AD916x API interface Headers*/
#include "AD9164.h"
#include "api_errors.h"
#include "api_def.h"

ad9164_handle_t app_dac_h = {
                    NULL,                          /* Client App HAL does not require any specific data*/
                    SPI_SDO,                       /* Client App HAL SPI  uses 4 Wire SPI config */
                    2000000000,                    /* Application DAC Clk Frequency */
                    &app_hal_spi_xfer_ad9164,      /* Client App HAL SPI function  for AD9164*/
                    &app_hal_delay_us,             /* Client App HAL Delay Function */
                    NULL,                          /* Client App does not use API Event Handler */
                    0,                            /*  Client App does need API to control TX_ENABLE */
                    0,                             /*  Client App does need API to control RESETB */
                    0,                             /* Client App does want API to initialize external HW required by DAC */
                    0                              /* Client App does want API to initialize external HW required by DAC */
};

/*AD916x Dac Initialisation by Client Application*/
int app_dac_init()
{
          int dacError = API_ERROR_OK;
          ad9164_handle_t *ad9164_h = app_dac_h;

          /*Initialise DAC Module*/
          dacError = ad9164_init(ad9164_h);
          if (dacError != API_ERROR_OK) {
                    return -1;
          }

          dacError = ad9164_get_chip_id(ad9164_h, &dac_chip_id);
          if (dacError != API_ERROR_OK) {
                    return -1;
          }

          dacError = ad9164_get_revision(ad9164_h,&revision[0],&revision[1],&revision[2]);
          if (dacError !=API_ERROR_OK) {
                    return APP_ERR_DAC_FAIL;
          }

          printf("*********************************************\r\n");
          printf("AD9164 DAC Chip ID: %d \r\n", dac_chip_id.chip_type);
          printf("AD9164 DAC Product ID: %d \r\n", dac_chip_id.prod_id);
          printf("AD9164 DAC Product Grade: %d \r\n", dac_chip_id.prod_grade);
          printf("AD9164 DAC Product Revision: %d \r\n", dac_chip_id.dev_revision);
          printf("AD9164 Revision: %d. %d.%d \r\n", revision[0], revision[1], revision[2]);
          printf("*********************************************\r\n");

  return 0;
}
```

Figure 5 Example A, Psuedo Code AD9164 Handle initialization with Minum Configuration

```
#include “app_spi.h”
#include “app_gpio.h”
#include "app_sleep.h"

/*Client Application function to Implement SPI Transfer for AD9164*/
int app_hal_ad9164_spi_xfer(void *user_data, uint8_t *wbuf, uint8_t *rbuf, int len)
{
      /*Pseudo Code example of implementing SPi transfer funtion*/


      uint16_t address;
      uint8_t value;
      uint8_t dac_chip_select;
      struct app_hal_data_t *dac_user_data;

      /*Optional: get any client defined data from user_data*/
      /*For example could be used to retrieve chip select*/
      dac_user_data = (struct app_hal_data_t *) user_data;
      dac_chip_select = dac_user_data->dac_chip_select_ref;

      /*AD916x DAC SPI transactions are always 3 bytes*/
      if (size_bytes != 3)
      {
            /* 2 bytes for adderss and 1 byte data */
            return 2;
      }

      address = wbuf[0];
      address <<= 8;
      address |= wbuf[1];
      value = wbuf[2];

      if ((address & 0x8000) == 0)
      {
            /* Write */
            app_hal_spi_write(dac_chip_select, address, value);
            rbuf[2] = 0xFF;
      }
      else
      {
            /* Read */
            /* Clear the read bit as we read from local array */
            address &= ~0x8000;

            app_hal_spi_write(dac_chip_select, address, &value);
            rbuf[2] = value;
      }
      return 0;
}

int app_hal_ad9164_delay_us(unsigned int time_us)
{
      /*Code to sleep or wait*/
      app_hal_sleep(time_us);
      return 0;

}
```

Figure 6 Psuedo Code example for Required HAL functions

```
/*
 * \brief pseudo Code example client code to initialiseAD916x
 *
 */
/*Include Client HAL implementation code*/
#include "app_hal.h"

/*Include AD916x API interface Headers*/
#include "AD9164.h"
#include "api_errors.h"
#include "api_def.h"

ad9164_handle_t app_dac_h = {
                    &app_hal_user_data,                /* Client App HAL does not require any specific data*/
                    SPI_SDO,                           /* Client App HAL SPI  uses 4 Wire SPI config */
                    2000000000,                        /* Application DAC Clk Frequency */
                    &app_hal_ad9164_spi_xfer,          /* Client App HAL SPI function  for AD9164*/
                    &app_hal_ad9164_delay_us,          /* Client App HAL Delay Function */
                    &app_ad9164_event_handler,         /* Client App does not use API Event Handler */
                    &app_hal_ad9164_set_tx_enable_pin, /*  Client App HAL function to control TX_ENABLE */
                    &app_hal_ad9164_set_resetb_pin,    /*  Client App does need API to control RESETB */
                    &app_init_dac_hw,                  /* Client App does want API to initialize external HW required by DAC */
                    &app_shutdown_dac_hw               /* Client App does want API to initialize external HW required by DAC */
};

/*AD916x Dac Initialisation by Client Application*/
int app_dac_init()
{
          int dacError = API_ERROR_OK;
          ad9164_handle_t *ad9164_h = app_dac_h;

          /*Initialise DAC Module*/
          dacError = ad9164_init(ad9164_h);
          if (dacError != API_ERROR_OK) {
                    return -1;
          }

          dacError = ad9164_get_chip_id(ad9164_h, &dac_chip_id);
          if (dacError != API_ERROR_OK) {
                    return -1;
          }

          dacError = ad9164_get_revision(ad9164_h,&revision[0],&revision[1],&revision[2]);
          if (dacError !=API_ERROR_OK) {
                    return APP_ERR_DAC_FAIL;
          }

          printf("*********************************************\r\n");
          printf("AD9164 DAC Chip ID: %d \r\n", dac_chip_id.chip_type);
          printf("AD9164 DAC Product ID: %d \r\n", dac_chip_id.prod_id);
          printf("AD9164 DAC Product Grade: %d \r\n", dac_chip_id.prod_grade);
          printf("AD9164 DAC Product Revision: %d \r\n", dac_chip_id.dev_revision);
          printf("AD9164 Revision: %d. %d.%d \r\n", revision[0], revision[1], revision[2]);
          printf("*********************************************\r\n");

   return 0;

}
```

Figure 7 Example B, Psuedo Code AD9164 Handle initialization with Minum Configuration

REVISION HISTORY

11/07/2016—Rev 1.0

Initial Release with Rev 1.0.0 API .... Universal