Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

AD917x API Specification
Rev 1.1

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 AD917x family of DACs. It describes the general structure of the AD917x API library, provides a detail list of the API functions and its associated data structures, macros, and definitions.

SCOPE

Currently the AD917x API libraries targets the AD917x 16-bit 12 GSPS, RF Digital to Analog Converter with Channelizers

Table 1 AD917x

Target Device Name	Device Description	Device Release Status	API Release Status
AD9172	16-bit 12.6 GSPS, RF Digital to Analog Converter with Channelizers	Released	Rev 1.1.1
AD9171	16-bit 6GSPS, RF Digital to Analog Converter.	Released	Rev 1.1.1
AD9173	16-bit 12.6 GSPS, RF Digital to Analog Converter with Channelizers	Released	Rev 1.1.1

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 AD917x 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 NCOs

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.

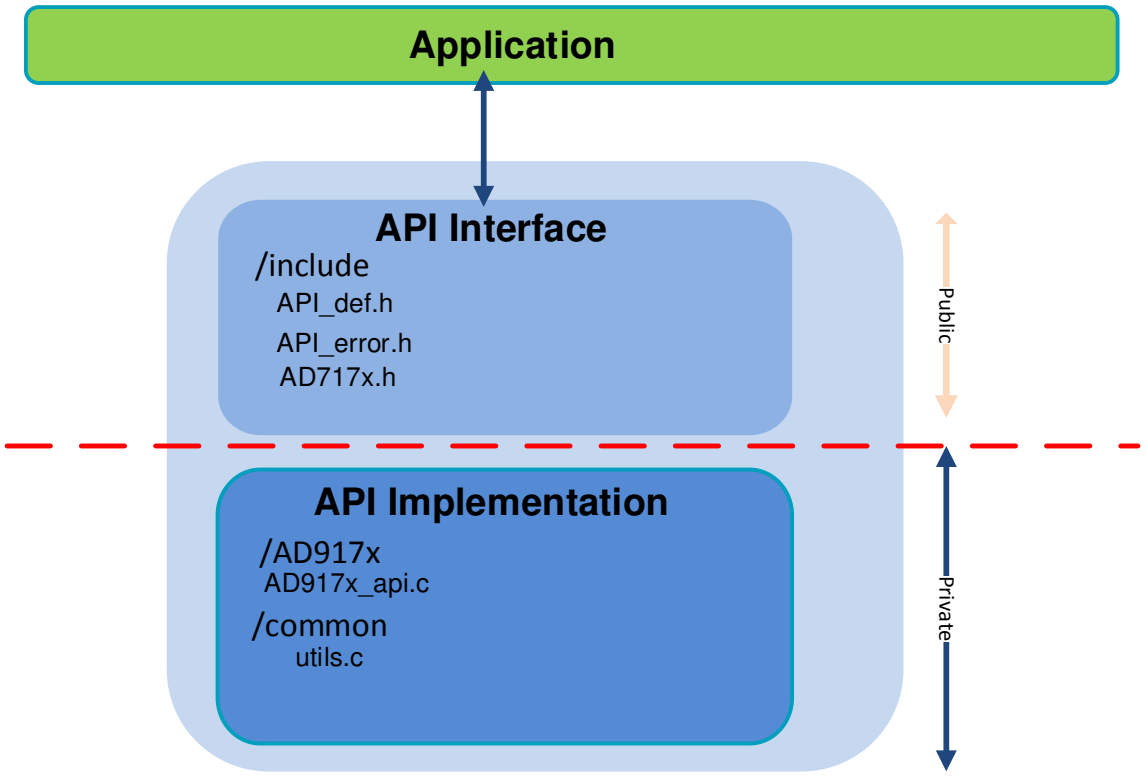

Figure 1 Simple Overview of the DAC API Architecture

FOLDER STRUCTURE

The collective files of the AD917x 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.

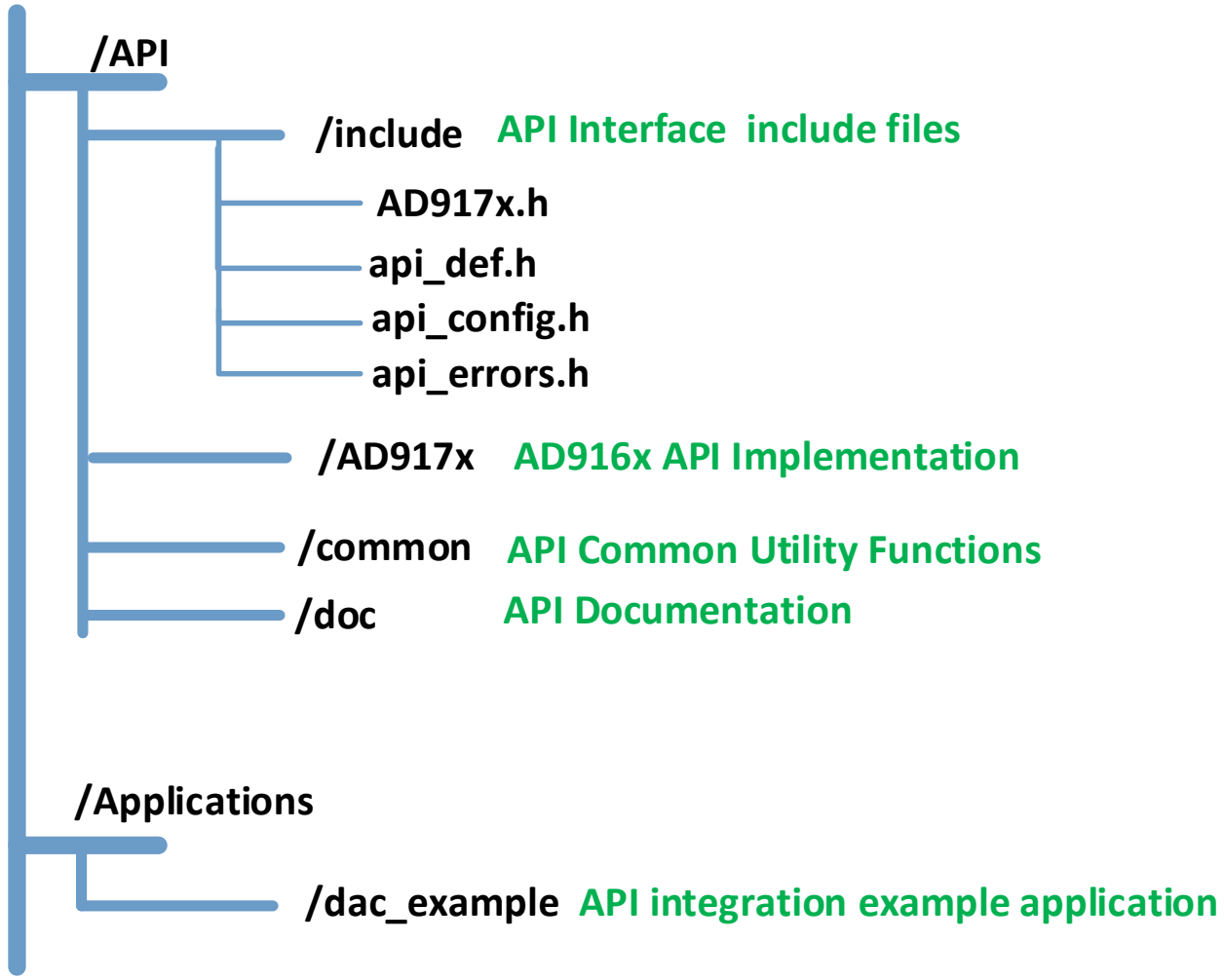

Figure 2 AD917x Source Code Folder Structure

/API

The AD917x API root folder contain all the source code and documentation for the AD917x API.

/API/include

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

/API/AD917x

This folder includes the main API implementation code for the AD917x 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/AD917x/doc

This folder contains the doxygen documentation for the AD917x APIs.

/Application/

This folder contains simple source code examples of how to use the DAC API. The application targets the AD917x 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 several 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 AD917x.h header file lists all the APIs that are available to control and configure the AD917x 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
AD917x.h	Lists AD917x 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

AD917X.H

The AD917x API library has a main interface header file AD917x.h header file that defines the software interface to the AD917x 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 ***ad917x_handle_t*** this is a data structure that acts a software reference to an instance to 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 ***ad917x_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*** 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
syncoutb	Desired Signal type for SYNCOUTB signal	Read/Write	
sysref	Desired Input coupling for sysref signal	Read/Write	
dac_freq_hz	DAC Clock Frequency in Hz. Valid range 2.9GHz to 12GHz	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 ***/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 AD917x 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 AD917x API. These functions are collectively referred 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 AD917x API to the required platform functions on instantiation of the AD917x API handle. The following is a description of HAL components.

The AD917x API handle, ***ad917x_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 ***ad917x_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 *Error! Reference source not found.* and ***ad917x_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 AD917x API assumes that SPI and all the external resources for the AD917x DAC are available.

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

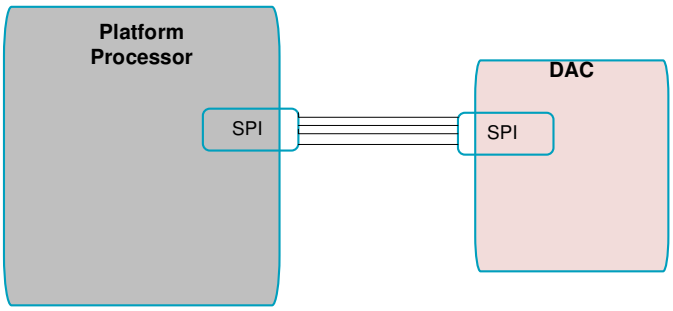

Figure 3 Hardware Controls Required By DAC API HAL

SPI Access

Access to the SPI controller that communicates with the AD917x 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 AD917x DAC SPI interface supports 3 wire or 4 wire and the AD917x DAC must be configured as such to match the platform implementation. This is done during initialization via the *Error! Reference source not found.* API. Please refer to the target ADI device datasheet for full details of the SPI Interface protocol

RESETB Pin Access

Optionally access to the function that controls the AD917x 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.

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 allows the API to wait the recommended microsecond between initialization steps.

The HAL function * ***delay_us_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 AD917x Device.

This function may initialize external hardware resources required by the AD917x 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 ***ad917x_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 AD917x Device

This function shall close or shutdown external hardware resources required by the AD917x 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 ***ad917x_deinit****Error! Reference source not found.* **.** 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***) ***ad917x_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 *Error! Reference source not found.* **.**

Once a DAC device is initialized via the *Error! Reference source not found.* 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***) *Error! Reference source not found.***.** Once a DAC device is initialized via the *Error! Reference source not found.* 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 ***ad917x_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***) ***ad917x_init.*** Once a DAC device is initialized via the ***ad917x_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.

*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***) ***ad917x_init.*** Once a DAC device is initialized via the ***ad917x_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 ***ad917x_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 ***ad917x_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***) ***ad917x_init.*** Once a DAC device is initialized via the ***ad917x_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.

ADI API ENUMERATIONS DATATYPES

The following are a list of enumerations and datatypes that are common to and used by ADI APIs libraries including the AD917x API. They are defined in adi_def.h

ADI_CHIP_ID_T

Description

A structure detailing the AD917x Device Identification Data. Please refer to the specific DAC Data sheet for expected values.

Synopsis

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

SIGNAL_TYPE_T

Description

A enumeration defining various signal types such as CMOS or LVDS.

Synopsis

```
#include api_def.h
typedef enum
{
        SIGNAL_CMOS = 0,
        SIGNAL_LVDS,
        SIGNAL_UNKNOWN
}signal_type_t;
```

Fields

SIGNAL_CMOS	CMOS TYPE Signal
SIGNAL_LVDS	**LVDS Type Signal**
SIGNAL_UNKNOWN	Undefined Signal Type

Notes

SIGNAL_COUPLING_T

Description

An enumeration defining AC and DC coupling modes.

Synopsis

```
#include api_def.h
typedef enum
{
        COUPLING_AC = 0,
        COUPLING_DC,
        COUPLING_DC
}signal_coupling_t;
```

Fields

COUPLING_AC	AC Coupled Signal
COUPLING_DC	DC Coupled Signal
COUPLING_DC	Undefined coupling Signal

Notes

JESD_LINK_T

Description

An enumeration of JESD Links available supported on the device.

Synopsis

```
#include api_def.h
typedef enum {
        JESD_LINK_0 = 0x0,
        JESD_LINK_1 = 0x1,
        JESD_LINK_ALL = 0xFF
}jesd_link_t;
```

Fields

JESD_LINK_0	JESD Link 0
JESD_LINK_1	JESD Link 1
JESD_LINK_ALL	ALL JESD LINKS Available

Notes

JESD_SYNCOUTB_T

Description

An enumeration of JESD SYNCOUTB Signals available on the device.

Synopsis

```
#include api_def.h
typedef enum {
    SYNCOUTB_0 = 0x0,    /**< SYNCOUTB 0 */
    SYNCOUTB_1 = 0x1,    /**< SYNCOUTB 1 */
    SYNCOUTB_0 = 0xFF    /**< ALL SYNCOUTB SIGNALS */
}jesd_syncoutb_t;
```

Fields

SYNCOUTB_0	JESD Link 0 SYNCOUTB Signal
SYNCOUTB_1	JESD Link 1 SYNCOUTB Signal
SYNCOUTB_0	ALL JESD LINKS SYNCOUTB Signals Available

Notes

JESD_SYSREF_MODE_T

Description

An enumeration of JESD SYSREF Signal modes of operation

Synopsis

```
#include api_def.h
typedef enum {
    SYSREF_NONE,
    SYSREF_ONESHOT,
    SYSREF_CONT,
    SYSREF_MON,
    SYSREF_MODE_INVLD
}jesd_sysref_mode_t;
```

Fields

SYSREF_NONE,	*No SYSREF Support*
SYSREF_ONESHOT,	*ONE-SHOT SYSREF Mode*
SYSREF_CONT,	*Continuous Sysref Synchronisation*
SYSREF_MON,	*SYSREF monitor Mode*
SYSREF_MODE_INVLD	

Notes

JESD_PARAM_T

Description

A structure defining the parameters of JESD Interface as per the JESD Specification

Synopsis

```
#include api_def.h
typedef struct {
    uint8_t jesd_L;
    uint8_t jesd_F;
    uint8_t jesd_M;
    uint8_t jesd_S;
    uint8_t jesd_HD;
    uint8_t jesd_K;
    uint8_t jesd_N;
    uint8_t jesd_NP;
    uint8_t jesd_CF;
    uint8_t jesd_CS;
    uint8_t jesd_DID;
    uint8_t jesd_BID;
    uint8_t jesd_LID0;
    uint8_t jesd_JESDV;
}jesd_param_t;
```

Members

jesd_L	JESD Lane Param L.
jesd_F	JESD Octet Param F.
jesd_M	JESD Converter Param M.
jesd_S	JESD No of Sample Param S.
jesd_HD	JESD High Density Param HD.
jesd_K	JESD multiframe Param K.
jesd_N	JESD Converter Resolution Param N.
jesd_NP	JESD Bit Packing Sample NP.
jesd_CF	JESD Param CF.
jesd_CS	JESD Param CS.
jesd_DID	JESD Device ID Param DID.
jesd_BID	JESD Bank ID. Param BID
jesd_LID0	JESD Lane ID for Lane 0 Param LIDO
jesd_JESDV	JESD Version

Notes

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 AD917x 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 a 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