AD917x API Specification
Rev 1.1
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>5</td>
</tr>
<tr>
<td>Purpose</td>
<td>5</td>
</tr>
<tr>
<td>Scope</td>
<td>5</td>
</tr>
<tr>
<td>Disclaimer</td>
<td>5</td>
</tr>
<tr>
<td>Software Architecture</td>
<td>6</td>
</tr>
<tr>
<td>Folder Structure</td>
<td>7</td>
</tr>
<tr>
<td>API Interface</td>
<td>9</td>
</tr>
<tr>
<td>Overview</td>
<td>9</td>
</tr>
<tr>
<td>ad917x.h</td>
<td>9</td>
</tr>
<tr>
<td>api_config.h</td>
<td>10</td>
</tr>
<tr>
<td>adi_def.h</td>
<td>10</td>
</tr>
<tr>
<td>HAL Function Pointer DataTypes</td>
<td>12</td>
</tr>
<tr>
<td>*hw_open_t</td>
<td>12</td>
</tr>
<tr>
<td>*hw_close_t</td>
<td>13</td>
</tr>
<tr>
<td>*spi_xfer_t</td>
<td>14</td>
</tr>
<tr>
<td>*tx_en_pin_ctrl_t</td>
<td>15</td>
</tr>
<tr>
<td>*reset_pin_ctrl_t</td>
<td>16</td>
</tr>
<tr>
<td>*delay_us_t</td>
<td>17</td>
</tr>
<tr>
<td>ADI API Enumerations DataTypes</td>
<td>18</td>
</tr>
<tr>
<td>adi_chip_id_t</td>
<td>18</td>
</tr>
<tr>
<td>signal_type_t</td>
<td>19</td>
</tr>
<tr>
<td>signal_coupling_t</td>
<td>20</td>
</tr>
<tr>
<td>jesd_link_t</td>
<td>21</td>
</tr>
<tr>
<td>jesd_syncoutb_t</td>
<td>22</td>
</tr>
<tr>
<td>jesd_param_t</td>
<td>24</td>
</tr>
<tr>
<td>Error Handling</td>
<td>25</td>
</tr>
<tr>
<td>Error Codes</td>
<td>25</td>
</tr>
<tr>
<td>AD917x API Library</td>
<td>26</td>
</tr>
<tr>
<td>AD917x API Reference Handle</td>
<td>27</td>
</tr>
<tr>
<td>ad917x_handle_t</td>
<td>27</td>
</tr>
<tr>
<td>AD917x API Definitions, Data Structures and Enumerations</td>
<td>29</td>
</tr>
<tr>
<td>ad917x.dds_select_t</td>
<td>29</td>
</tr>
<tr>
<td>ad917x_dac_select_t</td>
<td>30</td>
</tr>
<tr>
<td>ad917x_channel_select_t</td>
<td>31</td>
</tr>
<tr>
<td>ad917x_jesd_link_stat_t</td>
<td>32</td>
</tr>
<tr>
<td>ad917x_jesd_serdes_pll_flg_t</td>
<td>33</td>
</tr>
<tr>
<td>AD917x APIs</td>
<td>34</td>
</tr>
<tr>
<td>ad917x_init</td>
<td>34</td>
</tr>
<tr>
<td>ad917x_deinit</td>
<td>35</td>
</tr>
<tr>
<td>ad917x_reset</td>
<td>36</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ad917x_get_chip_id</td>
<td>37</td>
</tr>
<tr>
<td>ad917x_set_dac_clk_frequency</td>
<td>38</td>
</tr>
<tr>
<td>ad917x_get_dac_clk_frequency</td>
<td>39</td>
</tr>
<tr>
<td>ad917x_set_dac_pll_config</td>
<td>40</td>
</tr>
<tr>
<td>ad917x_get_dac_clk_status</td>
<td>42</td>
</tr>
<tr>
<td>ad917x_set_dac_clk</td>
<td>43</td>
</tr>
<tr>
<td>ad917x_set_clkout_config</td>
<td>44</td>
</tr>
<tr>
<td>ad917x_set_page_idx</td>
<td>45</td>
</tr>
<tr>
<td>ad917x_get_page_idx</td>
<td>46</td>
</tr>
<tr>
<td>ad917x_set_channel_gain</td>
<td>47</td>
</tr>
<tr>
<td>ad917x_get_channel_gain</td>
<td>48</td>
</tr>
<tr>
<td>ad917x_set_dc_cal_tone_amp</td>
<td>49</td>
</tr>
<tr>
<td>ad917x_ddsm_cal_dc_input_set</td>
<td>50</td>
</tr>
<tr>
<td>ad917x_ddsm_cal_dc_input_get</td>
<td>51</td>
</tr>
<tr>
<td>ad917x_dc_test_tone_set</td>
<td>52</td>
</tr>
<tr>
<td>ad917x_dc_test_tone_get</td>
<td>53</td>
</tr>
<tr>
<td>ad917x_nco_channel_freq_get</td>
<td>54</td>
</tr>
<tr>
<td>ad917x_nco_main_freq_get</td>
<td>55</td>
</tr>
<tr>
<td>ad917x_jesd_config_datapath</td>
<td>56</td>
</tr>
<tr>
<td>ad917x_jesd_get_cfg_param</td>
<td>57</td>
</tr>
<tr>
<td>ad917x_jesd_set_sysref_enable</td>
<td>58</td>
</tr>
<tr>
<td>ad917x_jesd_get_sysref_enable</td>
<td>59</td>
</tr>
<tr>
<td>ad917x_jesd_set_syncouth_enable</td>
<td>60</td>
</tr>
<tr>
<td>ad917x_jesd_get_cfg_status</td>
<td>61</td>
</tr>
<tr>
<td>ad917x_jesd_set_scrambler_enable</td>
<td>62</td>
</tr>
<tr>
<td>ad917x_jesd_set_lane_xbar</td>
<td>63</td>
</tr>
<tr>
<td>ad917x_jesd_get_lane_xbar</td>
<td>64</td>
</tr>
<tr>
<td>ad917x_jesd_invert_lane</td>
<td>65</td>
</tr>
<tr>
<td>ad917x_jesd_enable_datapath</td>
<td>66</td>
</tr>
<tr>
<td>ad917x_jesd_get_pll_status</td>
<td>67</td>
</tr>
<tr>
<td>ad917x_jesd_enable_link</td>
<td>68</td>
</tr>
<tr>
<td>ad917x_nco_set_ftw</td>
<td>70</td>
</tr>
<tr>
<td>ad917x_nco_get_ftw</td>
<td>71</td>
</tr>
<tr>
<td>ad917x_nco_set_phase_offset</td>
<td>72</td>
</tr>
<tr>
<td>ad917x_nco_get_phase_offset</td>
<td>73</td>
</tr>
<tr>
<td>ad917x_nco_enable</td>
<td>74</td>
</tr>
<tr>
<td>ad917x_register_write</td>
<td>76</td>
</tr>
<tr>
<td>ad917x_register_read</td>
<td>77</td>
</tr>
<tr>
<td>ad917x_get_revision</td>
<td>78</td>
</tr>
</tbody>
</table>

Build and Integration Guide

Building the AD917x API Library...

Integrating the AD917x API Library into an Application...
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>
<thead>
<tr>
<th>Target Device Name</th>
<th>Device Description</th>
<th>Device Release Status</th>
<th>API Release Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD9172</td>
<td>16-bit 12.6 GSPS, RF Digital to Analog Converter with Channelizers</td>
<td>Released</td>
<td>Rev 1.1.1</td>
</tr>
<tr>
<td>AD9171</td>
<td>16-bit 6GSPS, RF Digital to Analog Converter.</td>
<td>Released</td>
<td>Rev 1.1.1</td>
</tr>
<tr>
<td>AD9173</td>
<td>16-bit 12.6 GSPS, RF Digital to Analog Converter with Channelizers</td>
<td>Released</td>
<td>Rev 1.1.1</td>
</tr>
</tbody>
</table>

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 libraries 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.
FOLDER STRUCTURE
The collective files of the AD917x API library are structured 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
   - AD917x.h
   - api_def.h
   - api_config.h
   - api_errors.h
/AD917x AD916x API Implementation
/common API Common Utility Functions
/doc API Documentation

/Applications
/dac_example API integration example application

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.
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

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Description</th>
<th>To be included in Client Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD917x.h</td>
<td>Lists AD917x DAC API Library exposed to client application</td>
<td>Yes</td>
</tr>
<tr>
<td>api_config.h</td>
<td>Defines the various configuration options for the DAC Module</td>
<td>No</td>
</tr>
<tr>
<td>api_def.h</td>
<td>Defines any macros/enumerations or structures or definitions common to and used by all DAC API Libraries</td>
<td>No</td>
</tr>
<tr>
<td>api_error.h</td>
<td>Defines the DAC API interface errors and error handlers common to and used by all DAC API Libraries</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Structure Member</th>
<th>Description</th>
<th>User Read/Write Access</th>
<th>Required by API</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_data</td>
<td>Void Pointer to a user defined data structure. Shall be passed to all HAL functions.</td>
<td>Read/Write</td>
<td>Optional</td>
</tr>
<tr>
<td>sdo</td>
<td>Device SPI Interface configuration for DAC hardware</td>
<td>Read/Write</td>
<td>Yes</td>
</tr>
<tr>
<td>syncoutb</td>
<td>Desired Signal type for SYNCOUTB signal</td>
<td>Read/Write</td>
<td></td>
</tr>
<tr>
<td>sysref</td>
<td>Desired Input coupling for sysref signal</td>
<td>Read/Write</td>
<td></td>
</tr>
<tr>
<td>dac_freq_hz</td>
<td>DAC Clock Frequency in Hz. Valid range 2.9GHz to 12GHz</td>
<td>Read/Write</td>
<td>Yes</td>
</tr>
<tr>
<td>dev_xfer</td>
<td>Pointer to SPI data transfer function for DAC hardware</td>
<td>Read/Write</td>
<td>Yes</td>
</tr>
<tr>
<td>delay_us</td>
<td>Pointer to delay function for DAC hardware</td>
<td>Read/Write</td>
<td>Yes</td>
</tr>
<tr>
<td>hw_open</td>
<td>Pointer to platform initialization function for DAC hardware</td>
<td>Read/Write</td>
<td>Optional</td>
</tr>
<tr>
<td>hw_close</td>
<td>Pointer to the platform shutdown function for DAC hardware</td>
<td>Read/Write</td>
<td>Optional</td>
</tr>
<tr>
<td>event_handler</td>
<td>Pointer to a client event handler function for DAC device.</td>
<td>Read/Write</td>
<td>Optional</td>
</tr>
</tbody>
</table>
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

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

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.
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**

```c
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`. 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**

```c
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**

```c
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 uint8_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

```c
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**

```c
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**

```c
#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
```c
#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**

```c
#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.**

<table>
<thead>
<tr>
<th>ERROR CODE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API_ERROR_OK</td>
<td>API completed successfully</td>
</tr>
<tr>
<td>API_ERROR_SPI_SDO</td>
<td>API could not complete successfully due to SPI_SDO configuration in API Handle</td>
</tr>
<tr>
<td>API_ERROR_INVALID_HANDLE_PTR</td>
<td>API could not complete successfully due to invalid pointer to API Handle</td>
</tr>
<tr>
<td>API_ERROR_INVALID_XFER_PTR</td>
<td>API could not complete successfully due to invalid pointer to SPI transfer function</td>
</tr>
<tr>
<td>API_ERROR_INVALID_DELAYUS_PTR</td>
<td>API could not complete successfully due to invalid pointer to Delay function</td>
</tr>
<tr>
<td>API_ERROR_INVALID_PARAM</td>
<td>API could not complete successfully due to invalid API parameter</td>
</tr>
<tr>
<td>API_ERROR_FTW_LOAD_ACK</td>
<td>API could not complete successfully due to Frequency Turning Word No-ACK</td>
</tr>
<tr>
<td>API_ERROR_NCO_NOT_ENABLED</td>
<td>API could not complete successfully due to NCO not currently Enabled</td>
</tr>
<tr>
<td>API_ERROR_INIT_SEQ_FAIL</td>
<td>API could not complete successfully due to NVRAM load error</td>
</tr>
</tbody>
</table>

**Table 6 API HAL function Error Code Macro definitions.**

<table>
<thead>
<tr>
<th>ERROR CODE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API_ERROR_SPI_XFER</td>
<td>SPI HAL function return an error during the implementation of this API</td>
</tr>
<tr>
<td>API_ERROR_US_DELAY</td>
<td>DELAY HAL function return an error during the implementation of this API</td>
</tr>
<tr>
<td>API_ERROR_TX_EN_PIN_CTRL</td>
<td>TX_ENABLE pin ctrl HAL function return an error during the implementation of this API</td>
</tr>
<tr>
<td>API_ERROR_RESET_PIN_CTRL</td>
<td>RESET pin ctrl HAL function return an error during the implementation of this API</td>
</tr>
<tr>
<td>API_ERROR_EVENT_HNDL</td>
<td>EVENT Handle HAL function return an error during the implementation of this API</td>
</tr>
<tr>
<td>API_ERROR_HW_OPEN</td>
<td>HW Open HAL function returned an error during the implementation of this API</td>
</tr>
<tr>
<td>API_ERROR_HW_CLOSE</td>
<td>HW Close HAL function returned an error during the implementation of this API</td>
</tr>
</tbody>
</table>

---
AD917X API REFERENCE HANDLE

AD917X_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 AD917x.h

typedef struct {
  void *user_data;
  spi_sdo_config_t sdo;
  signal_type_t syncoutb;
  signal_coupling_t sysref;
  uint64_t dac_freq_hz;
  spi_xfer_t dev_xfer;
  delay_us_t delay_us;
  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;
}ad917x_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 AD917x 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 `ad917x_init` in order to ensure SPI access to the AD917x.

signal_type_t syncoutb;
This member holds the desired SYNCOOUTB signal type, CMOS or LVDS for the AD917x DAC in the client system. `signal_type_t` enumerates the possible signal types and is defined in `api_def.h`. This member should be configured prior to calling `ad917x_init` and `ad917x_reset` in order to sure correct operation for the target hardware.

signal_coupling_t sysref;
This member holds the desired `sysref` signal coupling mode AC or DC coupling for the AD917x DAC in the client system.

signal_coupling_t
Enumerates the possible signal types and is defined in `api_def.h`. 
This member should be configured prior to calling `ad917x_init` and `ad917x_reset` in order to ensure correct operation for the target hardware.

```c
uint64_t dac_freq_hz;
```

This member holds the frequency of the DAC CLK provided to the AD917x DAC in the target system. It is important to configure this variable correctly prior to configuring the AD917x with operational modes such as NCO, JESD and data path as this value is used as reference for internal. This value should be accessed via the ad917x _set_dac_clk_frequency API and the `ad917x_get_dac_clk_frequency`.

```c
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.

```c
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.

```c
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.

```c
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.

```c
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.

```c
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.
AD917X API DEFINITIONS, DATA STRUCTURES AND ENUMERATIONS

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

**AD917X_DDS_SELECT_T**

Description

An enumeration of the Direct Digital Synthesis Blocks within the AD917x Device.

Synopsis

```c
#include AD917x.h

typedef enum {
    AD917X_DDSM = 0,
    AD917X_DDSC = 1
} ad917x_dds_select_t;
```

Fields

- AD917X_DDSM: Main DAC Datapath DDS
- AD917X_DDSC: Channel Datapath DDS

Notes
AD917X_DAC_SELECT_T

Description
An enumeration of the DACs within the AD917x Device, used to select or target a DAC.

Synopsis
#include AD917x.h
typedef enum {

    AD917X_DAC_NONE = 0,
    AD917X_DAC0 = 1,
    AD917X_DAC1 = 2
}ad917x_dac_select_t;

Fields

AD917X_DAC_NONE No DAC
AD917X_DAC0 DAC0
AD917X_DAC1 DAC 1

Notes
AD917X_CHANNEL_SELECT_T

Description
An enumeration of the Channelizers within the AD917x Device, used to select or target a Channelizer.

Synopsis
```
#include AD917x.h
typedef enum {
    AD917X_CH_NONE = 0,
    AD917X_CH_0 = BIT(0),
    AD917X_CH_1 = BIT(1),
    AD917X_CH_2 = BIT(2),
    AD917X_CH_3 = BIT(3),
    AD917X_CH_4 = BIT(4),
    AD917X_CH_5 = BIT(5)
} ad917x_channel_select_t;
```

Fields
- **AD917X_CH_NONE**: No Channel
- **AD917X_CH_0**: Channel 0
- **AD917X_CH_1**: Channel 1
- **AD917X_CH_2**: Channel 2
- **AD917X_CH_3**: Channel 3
- **AD917X_CH_4**: Channel 4
- **AD917X_CH_5**: Channel 5

Notes
AD917X_JESD_LINK_STAT_T

Description
A structure of the JESD Interface link Status

Synopsis
#include AD917x.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;
} ad917x_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
**AD917X_JESD_SERDES_PLL_FLG_T**

**Description**

An enumeration of the SERDES PLL Status flags.

**Synopsis**

```c
#include AD917x.h
typedef enum
{
   AD917x_PLL_LOCK_STAT = 0x1,
   AD917x_PLL_REG_RDY= 0x2,
   AD917x_PLL_CAL_STAT=0x4,
   AD917x_PLL_LOSSLOCK=0x8
}ad917x_jesd_serdes_pll_flg_t;
```

**Members**

- **AD917X_PLL_LOCK_STAT**
  
  SERDES PLL lock Status Flag. When set the PLL is locked.

- **AD917X_PLL_REG_RDY**
  
  SERDES PLL Regulator RDY Status Flag.

- **AD917X_PLL_CAL_STAT**
  
  SERDES PLL VCO Calibration Status Flag.

- **AD917X_PLL_LOSSLOCK**
  
  SERDES PLL Upper Calibration Threshold flag.

**Notes**
**AD917X APIS**

**AD917X_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 AD917x.h
ADI_API int ad917x_init(ad917x_handle_t *h);
```

**Parameters**

**ad917x_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 ad917x_reset API immediately after a call to Ad9172_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* for full details.

**h->hw_open**

DAC API handle optionally may be set to valid hardware initialization function for client application. Refer to *hw_open_t* for full details.

**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_HANDLE_PTR

API_ERROR_INVALID_XFER_PTR

API_ERROR_INVALID_PARAM

API_ERROR_HW_OPEN_FAILED

API_ERROR_INIT_SEQ_FAIL

API_ERROR_SPI_SDO

**Notes**
AD917X_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 AD917x Device, for example GPIO SPI etc.

Synopsis
#include AD917x.h

ADI_API int ad917x_deinit(ad917x_handle_t *h);

Parameters
ad917x_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 successfully initialized via a call to the ad917x_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 ad917x_init by client application 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. Refer 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
AD917X_RESET

Description
Performs a full reset of AD917x 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 AD917x.h
ADI_API int ad917x_reset(ad917x_handle_t *h, uint8_t hw_reset);

Parameters
ad917x_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 indicate 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 successfully initialized via a call to the ad917x_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 indicated 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
AD917X_GET_CHIP_ID

Description

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

Synopsis

```
#include AD917x.h
ADI_API int ad917x_get_chip_id(ad917x_handle_t *h, adi_chip_id_t *chip_id);
```

Parameters

- `ad917x_handle_t *h` Pointer to the client application DAC API handle for the target DAC device. Refer to API Handle section for more details.
- `adi_chip_id_t *chip_id` Pointer to a variable of type `adi_chip_id_t` to which the Device Identification data shall be stored.

Preconditions

The DAC device shall be successfully initialized via a call to the `ad917x_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
AD917X _SET_DAC_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 AD917x.h
ADI_API int ad917x_set_dac_clk_frequency(ad917x_handle_t *h, uint64_t dac_clk_freq_hz);

Parameters
ad917x_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. DAC Clock Frequency range is 2.9 GHz to 12GHz
for AD9172 and AD9173.
DAC CLK Frequency range is 2.9 GHz to 6GHz for AD9171.

Preconditions
The DAC device shall be success fully initialized via a call to the ad917x_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
AD917X_GET_DAC_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 AD917x.h
ADI_API int ad917x_get_dac_clk_frequency(ad917x_handle_t *h, uint64_t *dac_clk_freq_hz);

Parameters
ad917x_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 shall be stored.
The frequency value shall be provided in Hz. DAC clock frequency in Hz.
DAC Clock Frequency range is 2.9 GHz to 12GHz for AD9172 and AD9173.
DAC CLK Frequency range is 2.9 GHz to 6GHz for AD9171.

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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
AD917x API Specification  Rev 1.1

AD917X_SET_DAC_PLL_CONFIG

Description
The AD917x may be configured to use a clock directly applied to the device as the DAC clock or may generate a DAC clock using the on chip PLL.
This API allows enabling of on-chip PLL direct configuration of the on-chip PLL parameters.
This API should be used in conjunction with ad917x_set_dac_clk_frequency

Synopsis
#include AD917x.h

ADI_API int ad917x_set_dac_pll_config(ad917x_handle_t *h, uint8_t dac_pll_en,
uint8_t m_div, uint8_t n_div, uint8_t vco_div);

Parameters
ad917x_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 dac_pll_en  Enable for internal DAC Clock generation.
If set, ref_clk_freq_khz must be set with value of reference clock applied by the system.
0 - Do not generate DAC CLK internally.
1 - Generate DAC CLK internally

uint8_t m_div  Reference Clock Pre-divider. Where
M_DIVIDER = Ceiling (Fref_clk_mhz/500 MHz)
Valid Range 1 to 4

uint8_t n_div  VCO Feedback Divider Ratio. Where
N_DIVIDER = Fvco * M_DIVIDER/(8 * Fref_clk)
Valid Range 2 -50

uint8_t vco_div  Required VCO Divider for the Desired DAC CLK, where
Fdac = Fvco/VCO_DIVIDER
Valid range 1-3

Preconditions
The DAC device shall be success fully initialized via a call to the ad917x_init API. Prior to using this function.

Post conditions
This API will configure the on chip PLL parameters for a particular dac clk frequency. The user must call the ad917x_set_dac_clk_frequency API to update API with the desired dac frequency.

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

The on-chip PLL may not be able to generate all supported DAC Clocks. A directly applied DAC clock may be required. The AD917x datasheet should be consulted for full details of the capabilities. API_ERROR_INVALID_PARAM shall be returned if one of the parameters are outside the range of the PLL.
AD917X_GET_DAC_CLK_STATUS

Description
Get DAC CLK Status.

Synopsis
#include AD917x.h

ADI_API int ad917x_get_dac_clk_status(ad917x_handle_t *h, uint8_t *pll_lock_stat, uint8_t *dll_lock_stat);

Parameters
ad917x_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_lock_stat
DAC PLL Lock Status
0 - DAC PLL Not Locked.
1 - DAC PLL Locked.

uint8_t*dll_lock_stat
DAC PLL Lock Status
0 - DAC PLL Not Locked.
1 - DAC PLL Locked.

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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
**AD917X_SET_DAC_CLK**

**Description**
Configure the DAC Clock Input path based on a desired DAC clock frequency, the applied reference clock and the on-chip PLL.

The AD917x may be configured to use a clock directly applied to the device as the DAC clock or may generate a DAC Clock using the clock applied by the system as a reference.

This function shall calculate and apply the required on-chip PLL configuration based on the desired DAC clock frequency and the applied reference clock frequency.

This function may be used instead of the following two APIs *ad917x_set_dac_pll_config* and *ad917x_set_dac_clk_frequency*.

**Synopsis**
```c
#include AD917x.h

ADI_API int ad917x_set_dac_clk(ad917x_handle_t *h, uint64_t dac_clk_freq_hz, uint8_t dac_pll_en, uint64_t ref_clk_freq_hz);
```

**Parameters**
- `ad917x_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`: Desired DAC Clk Frequency in Hz.
  - DAC Clock Frequency range is 2.9 GHz to 12GHz for AD9172 and AD9173. DAC CLK Frequency range is 2.9 GHz to 6GHz for AD9171.
- `uint8_t dac_pll_en`: Enable for internal DAC PLL
- `uint64_t ref_clk_freq_hz`: Value of reference clock frequency applied to AD917x. Set to 0 if DAC CLK is applied to the pin.

**Preconditions**
The DAC device shall be successfully initialized via a call to the *ad917x_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**
The on-chip PLL may not be able to generate all supported DAC Clocks. A directly applied DAC clock may be required.
The AD917x datasheet should be consulted for full details of the capabilities. API_ERROR_INVALID_PARAM shall be returned if one of the parameters are outside the range of the PLL.
**AD917X_SET_CLKOUT_CONFIG**

**Description**

The AD917x DAC provides an output clock signal generated from the DAC clk via the CLKOUT pin. This API sets CLKOUT signal configuration.

**Synopsis**

```c
#include AD917x.h
ADI_API int ad917x_set_clkout_config(ad917x_handle_t *h, uint8_t l_div);
```

**Parameters**

- `ad917x_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 l_div` Output clock divider setting. Valid range 1 to 4.

**Preconditions**

The DAC device shall be successfully initialized via a call to the *ad917x_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**
AD917X_SET_PAGE_IDX

Description
Select Page

Synopsis
#include AD917x.h
ADI_API int ad917x_set_page_idx(ad917x_handle_t *h, const unsigned int dac, const unsigned int channel);

Parameters
ad917x_handle_t *h Pointer to the client application DAC API handle for the target DAC device. Refer to API Handle section for more details.
unsigned int dac DAC number.
unsigned int channel Channel number.

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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
**AD917X_GET_PAGE_IDX**

**Description**
Get Page index

**Synopsis**
```
#include AD917x.h

ADI_API int ad917x_get_page_idx(ad917x_handle_t *h, int *dac, int *channel);
```

**Parameters**
- `ad917x_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 *dac`  
  Pointer to the DAC number.
- `int *channel`  
  Pointer to the Channel number.

**Preconditions**
The DAC device shall be success fully initialized via a call to the *ad917x_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**
AD917X_SET_CHANNEL_GAIN

Description
Sets the scalar channel gain value. It is paged by CHANNEL_PAGE in Reg08

Synopsis
#include AD917x.h
ADI_API int ad917x_set_channel_gain(ad917x_handle_t *h, const uint16_t gain);

Parameters
  ad917x_handle_t *h  Pointer to the client application DAC API handle for the target DAC device. Refer to API Handle section for more details.
  uint16_t gain  Gain value.

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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
AD917x API Specification  Rev 1.1

AD917X_GET_CHANNEL_GAIN

Description
Sets the scalar channel gain value. It is paged by CHANNEL_PAGE in Reg08

Synopsis
```
#include AD917x.h
ADI_API int ad917x_get_channel_gain (ad917x_handle_t *h, const uint16_t *gain);
```

Parameters
- `ad917x_handle_t *h`  Pointer to the client application DAC API handle for the target DAC device.
  Refer to API Handle section for more details.
- `uint16_t *gain`  Pointer to the gain value.

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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
**AD917X_SET_DC_CAL_TONE_AMP**

**Description**
Sets the DC tone amplitude. This amplitude goes to both I and Q paths. It is paged by CHANNEL_PAGE in Reg08

**Synopsis**
```
#include AD917x.h
ADI_API int ad917x_set_dc_cal_tone_amp(ad917x_handle_t *h, const uint16_t amp);
```

**Parameters**
- `ad917x_handle_t *h` Pointer to the client application DAC API handle for the target DAC device. Refer to *API Handle* section for more details.
- `uint16_t amp` Calibration tone amplitude.

**Preconditions**
The DAC device shall be successfully initialized via a call to `ad917x_init` 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**
AD917X_DDSM_CAL_DC_INPUT_SET

Description
Set Main DAC Cal DC Input

Synopsis
#include AD917x.h
ADI_API int ad917x_ddsm_cal_dc_input_set(ad917x_handle_t *h, int ddsm_cal_dc_input_en);

Parameters
ad917x_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 ddsm_cal_dc_input_en Enable flag:
0 - Disabled
1 - Enabled

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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
AD917x API Specification

AD917X_DDSM_CAL_DC_INPUT_GET

Description
Get Main DAC Cal DC Input

Synopsis
#include AD917x.h
ADI_API int ad917x_ddsm_cal_dc_input_get(ad917x_handle_t *h, int *ddsm_cal_dc_input_en);

Parameters
ad917x_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 *ddsm_cal_dc_input_en Pointer to integer, where the result will be stored
0 - Disabled
1 - Enabled

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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_f.

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
AD917X_DC_TEST_TONE_SET

Description
Set DC Test Tone enable status

Synopsis
#include AD917x.h
ADI_API int ad917x_dc_test_tone_set(ad917x_handle_t *h, int dc_test_tone_en);

Parameters
ad917x_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 dc_test_tone_en  Enable flag
0 - Disabled
1 - Enabled

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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
AD917X_DC_TEST_TONE_GET

Description
Set DC Test Tone enable status

Synopsis
#include AD917x.h
ADI_API int ad917x_dc_test_tone_get(ad917x_handle_t *h, int *dc_test_tone_en);

Parameters
ad917x_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 *dc_test_tone_en
Pointer to integer, where the result will be stored
0 - Disabled
1 - Enabled

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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
AD917X_NCO_CHANNEL_FREQ_GET

Description
Get a Channel NCO frequency in Hz

Synopsis
```
#include AD917x.h

ADI_API int ad917x_nco_channel_freq_get(ad917x_handle_t *h, ad917x_channel_select_t channel,
                                         int64_t *carrier_freq_hz);
```

Parameters
- `ad917x_handle_t *h` Pointer to the client application DAC API handle for the target DAC device.
  Refer to API Handle section for more details.
- `ad917x_channel_select_t channel` Channel number
  - `AD917x_CH_0` - Channel 0 NCO
  - `AD917x_CH_1` - Channel 1 NCO
  - `AD917x_CH_2` - Channel 2 NCO
  - `AD917x_CH_3` - Channel 3 NCO
  - `AD917x_CH_4` - Channel 4 NCO
  - `AD917x_CH_5` - Channel 5 NCO
- `int64_t *carrier_freq_hz` Pointer to 64 bit integer, where the result frequency will be stored

Preconditions
The DAC device shall be successfully initialized via a call to the `ad917x_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
AD917X_NCO_MAIN_FREQ_GET

Description
Get a Main DAC NCO frequency in Hz

Synopsis
#include AD917x.h

ADI_API int ad917x_nco_main_freq_get(ad917x_handle_t *h, ad917x_dac_select_t dac,
int64_t *carrier_freq_hz);

Parameters
ad917x_handle_t *h Pointer to the client application DAC API handle for the target DAC device.
Refer to API Handle section for more details.
ad917x_dac_select_t dac Main data path DAC NCO select. Can be only one of:
AD917x_DAC0 - DAC0 NCO select
AD917x_DAC1 - DAC1 NCO select
int64_t *carrier_freq_hz Pointer to 64 bit integer, where the result frequency will be stored

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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
AD917X_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 AD917x.h
ADI_API int ad917x_jesd_config_datapath(ad917x_handle_t *h, uint8_t dual_en,
uint8_t jesd_mode, uint8_t ch_intpl, uint8_t main_intpl);

Parameters
ad917x_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 dual_en Dual Link enable setting
0 - Single Link Mode
1 - Dual Link Mode
uint8_t jesd_mode The desired value of the pre-defined JESD link modes supported by the AD917x.
Valid range 0 to 21. Based on this value the AD917x JESD interface shall be configured as per one of the supported JESD parameter configurations. Refer to the user guide for full details on the modes and the corresponding JESD settings.
uint8_t ch_intpl The desired channel interpolation.
uint8_t main_intpl The desired main dac data path interpolation.

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_init API prior to using this function.

Post conditions
The DAC device will set a bit to indicate if this configuration is valid/supported mode. Use ad917x_jesd_get_cfg_status API to verify the configuration validity.

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 ad917x_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

AD917X_JESD_GET_CFG_PARAM

Description
API to return all the current JESD Parameters.

Synopsis
#include AD917x.h
ADI_API int ad917x_jesd_get_cfg_param(ad917x_handle_t *h, jesd_param_t *jesd_param);

Parameters
ad917x_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 all the JESD parameters currently configured will be stored.

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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
**AD917X_JESD_SET_SYSREF_ENABLE**

**Description**
Enable AD917x SYSREF +- Pin Input Interface for the target system SYSREF signal

**Synopsis**
```
#include AD917x.h
ADI_API int ad917x_jesd_set_sysref_enable(ad917x_handle_t *h, uint8_t en)
```

**Parameters**
- `ad917x_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 SYSREF Input Interface
  - 1 - Power Up SYSREF Input
  - 0 - Power Down SYSREF Input

**Preconditions**
The DAC device shall be successfully initialized via a call to the `ad917x_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**
AD917X_JESD_GET_SYSREF_ENABLE

Description
Configure AD917x SYSREF +- Pin Input Interface for the target system SYSREF signal

Synopsis
#include AD917x.h
ADI_API int ad917x_jesd_get_sysref_enable(ad917x_handle_t *h, uint8_t *en)

Parameters
ad917x_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        Pointer to variable to which SYSREF Input Interface Enable status shall be stored
                   1 - Power Up SYSREF Input
                   0 - Power Down SYSREF Input

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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


**AD917x API Specification  Rev 1.1**

**AD917X_JESD_SET_SYNCOUTB_ENABLE**

**Description**
Configure and enable/disable the SYNCOUT_B Output Signal.

**Synopsis**
```
#include AD917x.h
ADI_API int ad917x_jesd_set_syncoutb_enable(ad917x_handle_t *h, jesd_syncoutb_t syncoutb, uint8_t en);
```

**Parameters**
- **ad917x_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_syncoutb_t syncoutb**
  Target SYNCOUTB Signal.
  Valid values defined by ad917x_syncoutb_t
  - SYNCOUTB_0
  - SYNCOUTB_1
  - SYCNOUTB_ALL
- **uint8_t en**
  Enable/Disable SYNCOUTB for target SYNCOUTB signal. Range 0 to 1
  - 0 - Disable
  - 1 - Enable

**Preconditions**
The DAC device shall be successfully initialized via a call to the **ad917x_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**
AD917X_JESD_GET_CFG_STATUS

Description

Returns JESD Configuration Valid Mode Status.

Synopsis

#include AD917x.h

ADI_API int ad917x_jesd_get_cfg_status(ad917x_handle_t *h, uint8_t *cfg_valid);

Parameters

ad917x_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 *cfg_valid Pointer to a variable in which the Valid JESD Configuration status shall be stored

0 - Invalid JESD and Interpolation Mode Configured
1 - Valid JESD and Interpolation Mode Configured.

Preconditions

The DAC device shall be successfully initialized via a call to ad917x_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
AD917X_JESD_SET_SCRAMBLER_ENABLE

Description
Enable or Disable the descrambler for the JESD Interface.

Synopsis
#include AD917x.h

ADI_API int ad917x_jesd_set_scrambler_enable(ad917x_handle_t *h, uint8_t en);

Parameters

ad917x_handle_t *h Pointer to the client application DAC API handle for the target DAC device.

uint8_t en Enable control for JESD Scrambler.

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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
**AD917X_JESD_SET_LANE_XBAR**

**Description**

Configure AD917x Lane Cross Bar to route the physical JESD lanes to the desired logical lanes.

**Synopsis**

```c
#include AD917x.h
ADI_API int ad917x_jesd_set_lane_xbar(ad917x_handle_t *h, uint8_t physical_lane, uint8_t logical_lane);
```

**Parameters**

- `ad917x_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 indicating the Physical Lanes to be routed to the serdes logical indicated by the logical_lane parameter.

- `uint8_t logical_lane`  
  `uint8_t` value indicating the corresponding logical lane for the physical lane listed in parameter physical_lane.

**Preconditions**

The DAC device shall be successfully initialized via a call to the `ad917x_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**
AD917X_JESD_GET_LANE_XBAR

Description
Return the physical to logical lane mapping set by the configured by the current Lane Cross Bar configuration.

Synopsis
```
#include AD917x.h
ADI_API int ad917x_jesd_get_lane_xbar(ad917x_handle_t *h, uint8_t *phy_log_map);
```

Parameters
```
ad917x_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 a 8 deep uint8_t array. Each element of the array represents the physical lane 0 – 7 and the value represents the logical lane assigned to that physical lane.
```

Preconditions
The DAC device shall be successfully initialized via a call to the `ad917x_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
AD917X_JESD_INVERT_LANE

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

Synopsis
```
#include AD917x.h

ADI_API int ad917x_jesd_invert_lane(ad917x_handle_t *h, uint8_t logical_lane, uint8_t invert)
```

Parameters
- `ad917x_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`
  Desired invert status for the logical lane represented in `logical_lane` parameter.
  - Set to 1 to invert.
  - Set to 0 to de-invert.

Preconditions
The DAC device shall be successfully initialized via a call to the `ad917x_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
AD917x API Specification   Rev 1.1

AD917X_JESD_ENABLE_DATAPATH

Description
Configure power up and enable the AD917x the JESD Interface.

Synopsis
#include AD917x.h
ADI_API int ad917x_jesd_enable_datapath(ad917x_handle_t *h, uint8_t lanes_msk, uint8_t run_cal, uint8_t en);

Parameters
ad917x_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 disable the respective JESD Lane.

uint8_t run_cal Parameter to indicate if JESD equalization routine 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 successfully initialized via a call to the ad917x_init API prior to using this function.
JESD interface should be successfully configured via ad917x_jesd_config_datapath API.

Post conditions
JESD Datapath is now enable. The status of the JESD PLL can now be verified via

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
**AD917x_JESD_GET_PLL_STATUS**

**Description**

Configure power up and enable the AD917x the JESD Interface.

**Synopsis**

```c
#include AD917x.h
ADI_API int ad917x_jesd_get_pll_status(ad917x_handle_t *h, uint8_t *pll_status);
```

**Parameters**

- `ad917x_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 the variable that will be set with the PLL status.  
  - `bit[0] => SERDES PLL Lock Status`
  - `bit[1] => Regulator Status`
  - `bit[2] => Calibration Status`
  - `bit[3] => LOSS_LOCK Status`

**Preconditions**

The DAC device shall be successfully initialized via a call to the `ad917x_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**
AD917X_JESD_ENABLE_LINK

Description

Configure power up and enable the AD917x the JESD Interface.

Synopsis

```c
#include AD917x.h
ADI_API int ad917x_jesd_enable_link(ad917x_handle_t *h, jesd_link_t link, uint8_t en);
```

Parameters

- `ad917x_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_t link`: Target Link on which to start JESD Link Bring up Procedure
- `uint8_t en`: Enable control for the JESD Link
  - 0 - Enable Link
  - 1 - Disable Link

Preconditions

The DAC device shall be successfully initialized via a call to the `ad917x_init` API prior to using this function.

Post conditions

Link Should now be enabled. Status of the link may verified via `ad917x_jesd_get_link_status` API.

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
AD917X_JESD_GET_LINK_STATUS

Description
Configure power up and enable the AD917x the JESD Interface.

Synopsis
#include AD917x.h
ADI_API int ad917x_jesd_get_link_status(ad917x_handle_t *h,
                                           jesd_link_t link, ad917x_jesd_link_status_t *link_status);

Parameters
ad917x_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_t link Target Link on which to start JESD Link Bring up Procedure

ad917x_jesd_link_stat_t *link_status
                       Pointer to the variable of type jesd_link_status that will be set with current
                       jesd link reaback data.

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_init API prior to using this function.

Post conditions
Link Should now be enabled

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
AD917X_NCO_SET_FTW

Description

The AD917x 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

```c
#include AD917x.h
ADI_API int ad917x_nco_set_ftw(ad917x_handle_t *h, const ad917x_dds_select_t nco, const uint64_t ftw,
                                  const uint64_t acc_modulus, const uint64_t acc_delta);
```

Parameters

- `ad917x_handle_t *h` Pointer to the client application DAC API handle for the target DAC device. Refer to API Handle section for more details.
- `ad917x_dds_select_t nco` Channel or Main data path select
  - `AD917x_DDSM` - Main data path select
  - `AD917x_DDSC` - Channel data path select
- `uint64_t ftw` Frequency tuning word value.
- `uint64_t acc_modulus` Modulus value.
- `Uint64_t acc_delta` Delta value.

Preconditions

The DAC device shall be successfully initialized via a call to the `ad917x_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
AD917X_NCO_GET_FTW

Description
Get FTW, ACC and MOD values for the paged NCO. The page should be selected in advance.

Synopsis
#include AD917x.h

ADI_API int ad917x_nco_get_ftw(ad917x_handle_t *h, const ad917x_dds_select_t nco, uint64_t *ftw,
                                  uint64_t *acc_modulus, uint64_t *acc_delta);

Parameters
ad917x_handle_t *h  Pointer to the client application DAC API handle for the target DAC device.
                       Refer to API Handle section for more details.
ad917x_dds_select_t nco  Channel or Main data path select
                       AD917x_DDRM - Main data path select
                       AD917x_DDSC - Channel data path select
uint64_t *ftw  Pointer to a variable where the frequency tuning word value will be stored.
uint64_t *acc_modulus  Pointer to a variable where the modulus value will be stored
uint64_t *acc_delta  Pointer to a variable where the delta value will be stored.

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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
**AD917X_NCO_SET_PHASE_OFFSET**

**Description**

Sets main datapath and/or channel datapath NCO phase offset.

**Synopsis**

```c
#include AD917x.h
ADI_API int ad917x_nco_set_phase_offset(ad917x_handle_t *h, const ad917x_dac_select_t dacs, const uint16_t dacs_po, const ad917x_channel_select_t channels, const uint16_t ch_po);
```

**Parameters**

- **ad917x_handle_t *h**: Pointer to the client application DAC API handle for the target DAC device. Refer to **API Handle** section for more details.

- **ad917x_dac_select_t dacs**: DAC number
  - `AD917x_DAC0` - DAC0 NCO
  - `AD917x_DAC1` - DAC1 NCO

- **uint16_t dacs_po**: The phase offset value for the selected DAC(s).

- **ad917x_channel_select_t channels**: Channel number
  - `AD917x_CH_0` - Channel 0 NCO
  - `AD917x_CH_1` - Channel 1 NCO
  - `AD917x_CH_2` - Channel 2 NCO
  - `AD917x_CH_3` - Channel 3 NCO
  - `AD917x_CH_4` - Channel 4 NCO
  - `AD917x_CH_5` - Channel 5 NCO.

- **uint16_t ch_po**: The phase offset value for the selected channel(s).

**Preconditions**

The DAC device shall be successfully initialized via a call to the `ad917x_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**
AD917X_NCO_GET_PHASE_OFFSET

Description

Gets main datapath and/or channel datapath NCO phase offset.

Synopsis

```
#include AD917x.h
ADI_API int ad917x_nco_get_phase_offset (ad917x_handle_t *h, const ad917x_dac_select_t dacs, const uint16_t *dacs_po,
const ad917x_channel_select_t channels, const uint16_t *ch_po);
```

Parameters

- `ad917x_handle_t *h` Pointer to the client application DAC API handle for the target DAC device. Refer to
  `API Handle` section for more details.
- `ad917x_dac_select_t dacs` DAC number
  - `AD917x_DAC0` - DAC0 NCO
  - `AD917x_DAC1` - DAC1 NCO
- `uint16_t *dacs_po` The phase offset value for the selected DAC(s).
- `ad917x_channel_select_t channels` Channel number
  - `AD917x_CH_0` - Channel 0 NCO
  - `AD917x_CH_1` - Channel 1 NCO
  - `AD917x_CH_2` - Channel 2 NCO
  - `AD917x_CH_3` - Channel 3 NCO
  - `AD917x_CH_4` - Channel 4 NCO
  - `AD917x_CH_5` - Channel 5 NCO
- `uint16_t *ch_po` The phase offset value for the selected channel(s).

Preconditions

The DAC device shall be successfully initialized via a call to the `ad917x_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
AD917X_NCO_ENABLE

Description
Enable/Disable NCOs. Enables only the DACs and Channel NCOs provided as parameters. All other DACs and Channel NCOs are disabled.

Synopsis
#include AD917x.h
ADI_API int ad917x_nco_enable(ad917x_handle_t *h, const ad917x_dac_select_t dacs, const ad917x_channel_select_t channels);

Parameters
ad917x_dac_select_t dacs  DAC(s) to enable
AD917x_DAC0 - DAC0 NCO
AD917x_DAC1 - DAC1 NCO

ad917x_channel_select_t channels  Channel(s) to enable
AD917x_CH_0  - Channel 0 NCO
AD917x_CH_1  - Channel 1 NCO
AD917x_CH_2  - Channel 2 NCO
AD917x_CH_3  - Channel 3 NCO
AD917x_CH_4  - Channel 4 NCO
AD917x_CH_5  - Channel 5 NCO

Preconditions
The DAC device shall be successfully initialized via a call to the ad917x_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
**AD917X_NCO_SET**

**Description**
Set NCO to produce a desired frequency with a desired amplitude.

**Synopsis**
```c
#include AD917x.h
ADI_API int ad917x_nco_set(ad917x_handle_t *h, const ad917x_dac_select_t dacs,
                            const ad917x_channel_select_t channels, int64_t carrier_freq_hz,
                            const uint16_t amplitude, int dc_test_tone_en,
                            int ddsm_cal_dc_input_en);
```

**Parameters**
- **ad917x_handle_t *h**  
  Pointer to the client application DAC API handle for the target DAC device. Refer to `API Handle` section for more details.
- **ad917x_dac_select_t dacs**  
  DAC(s) to select
  - **AD917x_DAC0** - DAC0 NCO
  - **AD917x_DAC1** - DAC1 NCO
- **ad917x_channel_select_t channels**  
  Channel(s) to select
  - **AD917x_CH_0** - Channel 0 NCO
  - **AD917x_CH_1** - Channel 1 NCO
  - **AD917x_CH_2** - Channel 2 NCO
  - **AD917x_CH_3** - Channel 3 NCO
  - **AD917x_CH_4** - Channel 4 NCO
  - **AD917x_CH_5** - Channel 5 NCO
- **int64_t carrier_freq_hz**  
  Desired carrier frequency
- **uint16_t amplitude**  
  Desire amplitude.
- **int dc_test_tone_en**  
  Enable Test tone.
- **int ddsm_cal_dc_input_en**  
  Enable DDSM DC input.

**Preconditions**
The DAC device shall be successfully initialized via a call to the `ad917x_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 `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

AD917X_REGISTER_WRITE

Description
Performs SPI register write access to DAC

Synopsis
#include AD917x.h

ADI_API int ad917x_register_write (ad917x_handle_t *h, const uint16_t address, const uint8_t data);

Parameters
ad917x_handle_t *h
指针到客户端应用程序DAC API句柄用于目标DAC设备。参见API句柄部分以获取更多详细信息。

const uint16_t address,
15位值表示SPI寄存器位置在目标DAC设备上。
参见AD917x数据表中有效值。

uint8_t *data
指针到一个8位变量，其中读取的寄存器值通过SPI将被存储。

Preconditions
DAC设备应当成功初始化通过调用ad917x_init API。在使用此函数之前。

Postconditions
None

Dependencies
h->dev_xfer。DAC API句柄必须初始化到有效的SPI传输函数用于客户端应用。
参见*spi_xfer_t。

Return value
任何正整数值可能代表一个错误代码要返回给应用程序。
参见错误代码部分以获取详细信息。此API错误代码可能的返回值如下。

API_ERROR_OK
API_ERROR_INVALID_XFER_PTR
API_ERROR_INVALID_PARAM

Notes
**AD917X_REGISTER_READ**

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

**Synopsis**
```c
#include AD917x.h

ADI_API int ad917x_register_read(ad917x_handle_t *h, const uint16_t address, uint8_t *data);
```

**Parameters**
- **ad917x_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 AD917x 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 successfully initialized via a call to the `ad917x_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**
AD917X_GET_REVISION

Description
Get API Revision Data.

Synopsis
#include AD917x.h
ADI_API int ad917x_get_revision (ad917x_handle_t *h, uint8_t *rev_major, uint8_t *rev_minor, uint8_t *rev_rc)

Parameters
ad917x_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
BUILD AND INTEGRATION GUIDE

This section provides an overview of build and integration steps required when using the AD917x API Source code.

As ADI provides the full source code, the user can integrate and build the libraries as per their application and platform requirements. This section is intended to act as a guide to the provided source code and example build files.

BUILDING THE AD917X API LIBRARY

1. **Port the Source Code**
   In order to build the AD917x API library the source must be ported from the ADI provided source code package to the target platform.
   From the /API folder port the following folders:
   `/AD917x`
   `/common`
   `/include`

2. **Building the AD917x Library**
   Example make-files are provided to build the AD917x API library. It is located in the /API/AD917x folder. The example make-files are configured for the generation of a static library using the GCC compiler on a Linux platform. Make-files may be used as is, or used as a reference to create make-files for target application platform and tool-chain see debug.mak, release.mak for tool chain and flag options.
   `/AD917x/makefile`
   `/AD917x/debug.mak`
   `/AD917x/release.mak`

INTEGRATING THE AD917X API LIBRARY INTO AN APPLICATION

1. **Implement the Hardware Abstraction Functions**
   The API requires access to a small number of platform specific, hardware and system control functions such as a system delay/sleep function or SPI bus controller functions, GPI controller. The end user must provide and implement these functions as per the AD917x API requirements.
   These functions prototypes are defined in API/include/api_def.h and are explained in detail in DAC Hardware Initialization Error! Reference source not found. section of this document.
   The minimum requirement is to provide the following:
   a. A function to implement SPI transaction to the AD917x DAC on the target hardware.
      ```c
      typedef int spi_xfer (void *user_data, uint8_t *indata, uint8_t *outdata, int size_bytes);
      ```
   b. A function to implement a delay function that will implement a sleep or delay in the order of microseconds to suspend the execution of API.
      ```c
      int delay_us (void *user_data, unsigned int us);
      ```

2. **Include the AD917x API Interface headers**
   The following header files define the interface to the AD917x API and should be included in the application.
   a. API/include/AD917x.h
   b. API/include/api_error.h

3. **Instantiate AD917x Handler**
   For each AD917x DAC device, the application must instantiate an AD917x handler reference. Refer to the ad917x_handle_t description for a full description of the AD917x handler.
For each handler instantiated by the application, all the require members of the AD917x handler must be initialized prior to calling any APIs with that handler as parameter. As per the AD917x API specification, the following members are required for correct operation of the AD917x APIs.

- dev_xfer, Pointer to SPI data transfer function for DAC hardware
- delay_us, Pointer to delay function for DAC hardware
- sdo, Device SPI Interface configuration for DAC hardware
- dac_freq_hz, DAC Clock Frequency configuration

**Point handler to SPI and Delay HAL Functions**

The members dev_xfer and delay_us must be set to point to the HAL function implemented by the application. (Refer to step one, **Implement the Hardware Abstraction Functions** in this list).

Appendix A provides a pseudo code example of how the handler can be instantiated with the hardware abstraction functions.

**Set the SPI Interface Configuration**

Another member of the handler that must be instantiated correctly is the SPI interface configuration. The hardware SPI interface to the AD917x device may be 3-wire or 4-wire depending on the target hardware. The handler must be configured appropriated base on the target application hardware. In the AD917x DAC example application, the handler is instantiated with 4-wire mode as per the AD917x evaluation platform. Refer to Appendix A to see the handler being instantiated with SPI_SDO value, meaning 4-wire mode.

4. **Set the DAC CLK Frequency**

The AD917x device requires a hardware DAC clock input to the device. The AD917x may be provided directly with the desired DAC clock or may provide a reference clock and use the on-chip PLL to generated the DAC clock. Refer to AD917x Datasheet for full configuration options of the DAC clock.

This desired configuration will be determined the application depending on the target platform and application.

The following APIs may be used to configure the AD917x for the desired configuration.

- ad917x_set_dac_clk_frequency
- ad917x_set_dac_pll_config

The desired frequency of the hardware clock input to the DAC must be supplied to the via the dac_freq_hz member of the AD917x handler. This is done via the ad917x_set_dac_clk_frequency API.

Alternatively ad917x_set_dac_clk may be used to configure the on chip PLL based on the desired system's clock frequencies rather than PLL parameters.

5. **Create the application**

Using the AD917x API provided in /API/include/AD917x.h write the application code to initialize, configure and monitor the AD917x API as per your target application requirements.

The ADI source code package for the AD917x API provides some code example applications for using the AD917x API to configure AD917x DAC devices. These can be found in the /Applications folder and may be used as a reference during the development of the customer's application.
APPENDIX A

PSEUDO CODE EXAMPLE FOR AD917X HANDLE

In the pseudo code example A, shows an example of the AD917x handle being configure with the minimum client application. In the pseudo code example B, shows an example of the AD917x handle being configured with all the optional configurations.

For further details, refer to the DAC Hardware Initialization section and the ad917x_handle_t section for full a description and more details on configuration.

/*
 * \brief pseudo Code example client code to initialise AD917x
 *
 */
/*Include Client HAL implementation code*/
#include "app_hal.h"

/*Include AD917x API interface Headers*/
#include "AD917x.h"
#include "api_errors.h"
#include "api_def.h"

ad917x_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 Clock Frequency */
  &app_hal_ad916x_spi_xfer,   /* Client App HAL SPI function for AD9164*/
  &app_hal_ad916x_delay_us,               /* Client App HAL Delay Function */
  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 */
};

Figure 4 Example A, Pseudo Code AD917x Handle initialization with Minum Configuration

//AD917x Dac Initialisation by Client Application*/
int app_dac_init(){
  int dacError = API_ERROR_OK;
  ad917x_handle_t *ad917x_h = app_dac_h;
  uint8_t revision[3] = {0,0,0};
  adi_chip_id_t dac_chip_id;

  /*Initialise DAC Module*/
  dacError = ad917x_init(ad917x_h);
  if (dacError != API_ERROR_OK) {
    return -1;
  }

  dacError = ad917x_get_chip_id(ad917x_h, &dac_chip_id);
  if (dacError != API_ERROR_OK) {
    return -1;
  }

  dacError = ad917x_get_revision(ad917x_h,&revision[0],&revision[1],&revision[2]);
  if (dacError !=API_ERROR_OK) {
    return APP_ERR_DAC_FAIL;
  }

  printf("***********\n  AD917x DAC Chip ID: %d %\n", dac_chip_id.chip_type);
  printf("AD917x DAC Product ID: %x %\n", dac_chip_id.prod_id);
  printf("AD917x DAC Product Grade: %d %\n", dac_chip_id.prod_grade);
  printf("AD917x DAC Product Revision: %d %\n", dac_chip_id.dev_revision);
  printf("AD917x Revision: %d,%d,%d %\n", revision[0], revision[1], revision[2]);
  printf("***********");

  return 0;

}
#include "app_spi.h"
#include "app_gpio.h"
#include "app_sleep.h"

/*Client Application function to Implement SPI Transfer for AD9172*/
int app_hal_ad917x_spi_xfer(void *user_data, uint8_t *wbuf, uint8_t *rbuf, int len) {
    /*Pseudo Code example of implementing SPI transfer function*/

    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 address 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_ad917x_delay_us(void *user_data, unsigned int time_us) {
    /*Code to sleep or wait*/
    app_hal_sleep(time_us);
    return 0;
}
AD917x API Specification

/*
 * brief pseudo Code example client code to initialise AD917x
 *
 */
/*Include Client HAL implementation code*/
#include "app_hal.h"

/*Include AD917x API interface Headers*/
#include "AD917x.h"
#include "api_errors.h"
#include "api_def.h"

ad917x_handle_t app_dac_h = {
    &app_hal_user_data, /* Client App HAL does require any specific data*/
    SPI_SDO, /* Client App HAL SPI uses 4 Wire SPI config */
    2000000000, /* Application DAC Clk Frequency */
    &app_hal_ad917x_spi_xfer, /* Client App HAL SPI function for AD9164*/
    &app_hal_ad917x_delay_us, /* Client App HAL Delay Function */
    &app_hal_ad917x_set_tx_enable_pin, /* Client App HAL function to control TX_ENABLE */
    &app_hal_ad917x_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 */
};

/*AD917x Dac Initialisation by Client Application*/
int app_dac_init() {
    int dacError = API_ERROR_OK;
    ad917x_handle_t *ad916x_h = app_dac_h;
    uint8_t revision[3] = {0,0,0};
    adi_chip_id_t dac_chip_id;

    /*Initialise DAC Module*/
    dacError = ad917x_init(ad917x_h);
    if (dacError != API_ERROR_OK) {
        return -1;
    }

    dacError = ad917x_get_chip_id(ad917x_h, &dac_chip_id);
    if (dacError != API_ERROR_OK) {
        return -1;
    }

    dacError = ad917x_get_revision(ad917x_h,&revision[0],&revision[1],&revision[2]);
    if (dacError !=API_ERROR_OK) {
        return APP_ERR_DAC_FAIL;
    }

    printf("*********************************************
    AD917x DAC Chip ID: %d \n", dac_chip_id.chip_type);
    printf("AD917x DAC Product ID: %d \n", dac_chip_id.prod_id);
    printf("AD917x DAC Product Grade: %d \n", dac_chip_id.prod_grade);
    printf("AD917x DAC Product Revision: %d \n", dac_chip_id.dev_revision);
    printf("AD917x Revision: %d. %d.%d \n", revision[0], revision[1], revision[2]);
    printf("*********************************************\n");

    return 0;
}

Figure 6 Example B, Psuedo Code AD917x Handle initialization with Minimum Configuration
APPENDIX B
FLOW CHART FOR EXAMPLE AD917X INITIALISATION

Legend
- AD9172 Configuration

1. Instantiate AD9172 handler:
   ```c
   Ad9172_handler_t dac_h = {...};
   ```

2. Initialize AD9172 API:
   ```c
   Ad9172_init(dac_h);
   ```

3. Initialize AD9172 API:
   ```c
   ad9172_reset(dac_h);
   ```

4. End
FLOW CHART FOR EXAMPLE CLK CONFIGURATION

1. Instantiate AD917x handler
   
   ```c
   ad916x_handler_t dac_h = {…};
   ```

2. Initialize AD917x API
   
   ```c
   ad917x_init(dac_h)
   ```

3. Initialize AD917x API
   
   ```c
   ad917x_reset(dac_h)
   ```

4. Configure system to provide desired DAC CLK or Reference to AD917x
   
   - Review Clock Scheme Parameters
   - Configure system to provide desired DAC CLK or Reference to AD917x

5. Check DAC CLK Status
   
   ```c
   ad917x_get_dac_clk_status(dac_h, &pll_lock, &dll_lock)
   ```

6. Is DLL and PLL Locked?
   
   No

   - Turn ON On Chip PLL, Set reference clock. Set the DAC CLK frequency.
     ```c
     ad917x_set_dac_clk(dac_h, 9830e6, 1, 30720000)
     ```
   
   Yes

   - Turn off On Chip PLL, No reference clock. Set the DAC CLK frequency.
     ```c
     ad917x_set_dac_clk(dac_h, 9830e6, 0, 0)
     ```

7. Is DLL and PLL Locked?
   
   Yes

   - AD917x DAC CLK Configured

Legend

- AD917x Configuration
- System Configuration
FLOW CHART FOR EXAMPLE JESD CONFIGURATION

1. Instantiate AD917x handler
   `ad916x_handler_t dac_h = {...};`
2. Initialize AD917x API
   `ad917x_init(dac_h)`
3. Initialize AD917x API
   `ad917x_reset(dac_h)`
4. Set the DAC Clock frequency for the desired DAC mode
   `ad917x_set_dac_clk(dac_h, 9830e6, 0, 0)`
5. Configure the JESD Receiver datapath
   DUAL Link
   JESD Mode 9
   Channelizer Interpolation: 1
   DAC Interpolation: 8
   `ad917x_jesd_config_datapath(dac_h, 1, 0x9, 1, 8);`
6. Check Serdes PLL status
   No Locked
7. Configure and Enable JESD Transmitter
8. Enable the JESD Receiver datapath, Lane 8
   Equaliser Calibration Mode: 1
   Enable JESD Datapath
   `ad917x_jesd_enable_datapath(dac_h, 0xFF, 0x0, 0x1);`
9. Check Serdes PLL status
10. Enable the JESD Receiver data path
    `ad917x_jesd_enable_link(dac_h, JESD_LINK_ALL, 0x1)`
11. Check Link Status
    Sync
    `ad9156_jesd_get_link_status(dac_h, JESD_LINK0, &link_status)`
    `ad9156_jesd_get_link_status(dac_h, JESD_LINK1, &link_status)`
12. Is JESD Config Error on JESD Transmitter?
    No
    Yes
13. Debug JESD Link Configuration
14. End
REVISION HISTORY

08/06/2017—Rev 1.0

Initial Release with Rev 1.0.2 API................................. Universal
Minor Updated with Rev 1.1.1 API................................. Universal