USB Audio 2.0 Device Class Library for Analog Devices ADSP-SC594 User's Guide Revision 1.02

Closed Loop Design, LLC

support@cld-llc.com

Table of Contents

Disclaimer

This software is supplied "AS IS" without any warranties, express, implied or statutory, including but not limited to the implied warranties of fitness for purpose, satisfactory quality and non-infringement. Closed Loop Design LLC extends you a royalty-free right to use, reproduce, and distribute executable files created using this software for use with Analog Devices ADSP-SC5xx family processors only. Nothing else gives you the right to use this software.

Introduction

The Closed Loop Design (CLD) Audio 2.0 library creates a simplified interface for developing a USB Audio v2.0 device using the Analog Devices EV-SOMCRR-EZKIT and the EV-SC594-SOM System-on-Module boards. The CLD SC594 Audio 2.0 library also includes support for timer functions that facilitate creating timed events quickly and easily. The library's User application interface is comprised of parameters used to customize the library's functionality as well as callback functions used to notify the User application of events. These parameters and functions are described in greater detail in the CLD SC594 Audio 2.0 Library API section of this document.

USB Background

The following is a very basic overview of some of the USB concepts that are necessary to use the CLD SC594 Audio 2.0 Library. However, it is still recommended that developers have at least a basic understanding of the USB 2.0 protocol. The following are some resources to refer to when working with USB, and USB Audio v2.0:

- The USB 2.0 Specification
- The USB Device Class Definition for Audio Devices v2.0,
  The USB Device Class Definition for Audio Data Formats v.2.0
  The USB Device Class Definition for Terminal Types v.2.0
- USB in a Nutshell: A free online wiki that explains USB concepts.
  http://www.beyondlogic.org/usbnutshell/usb1.shtml
- "USB Complete" by Jan Axelson ISBN: 1931448086

USB is a polling based protocol where the Host initiates all transfers, all USB terminology is from the Host's perspective. For example an 'IN' transfer is when data is sent from a Device to the Host, and an 'OUT' transfer is when the Host sends data to a Device.

The USB 2.0 protocol defines a basic framework that devices must implement in order to work correctly. This framework is defined in the Chapter 9 of the USB 2.0 protocol, and is often referred to as the USB 'Chapter 9' functionality. Part of the Chapter 9 framework is standard USB requests that a USB Host uses to control the Device. Another part of the Chapter 9 framework is the USB Descriptors. These USB Descriptors are used to notify the Host of the Device's capabilities when the Device is attached. The USB Host uses the descriptors and the Chapter 9 standard requests to configure the Device. This process is called USB Enumeration. The CLD library includes support for the USB standard requests and USB Enumeration using some of the parameters specified by the User application when initializing the library. These parameters are discussed in the cld_sc594_audio_2_0_lib_init section of this document. The CLD library facilitates USB enumeration and is Chapter 9 compliant without User Application intervention as shown in the flow chart below. For additional information on USB Chapter 9 functionality or USB Enumeration please refer to one of the USB resources listed above.

CLD Library USB Enumeration Flow Chart

All USB data is transferred using Endpoints that act as a source or sink for data based on the endpoint's direction (IN or OUT). The USB protocol defines four types of Endpoints, each of which has unique characteristics that dictate how they are used. The four Endpoint types are: Control, Interrupt, Bulk, and Isochronous. Data that is transmitted over USB is broken up into blocks of data called packets. For each endpoint type there are restrictions on the allowed max packet size. The allowed max packet sizes also

vary based on the USB connection speed. Please refer to the USB 2.0 protocol for more information about the max packet size supported by the four endpoint types.

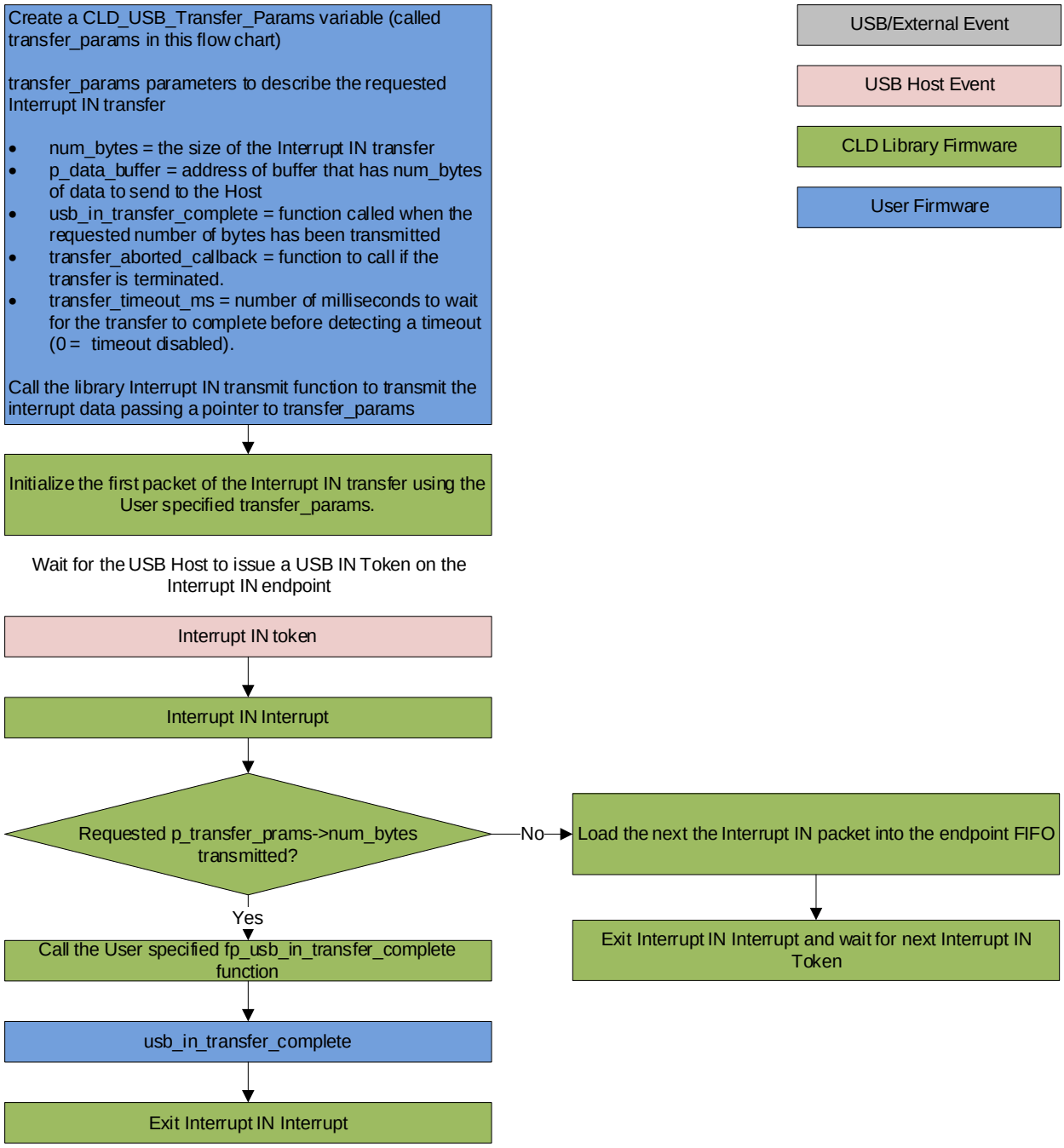
The CLD SC594 Audio 2.0 Library uses Control, Interrupt, and Isochronous endpoints, these endpoint types will be discussed in more detail below.

A Control Endpoint is the only bi-directional endpoint type, and is typically used for command and status transfers. A Control Endpoint transfer is made up of three stages (Setup Stage, Data Stage, and Status Stage). The Setup Stage sets the direction and size of the optional Data Stage. The Data Stage is where any data is transferred between the Host and Device. The Status Stage gives the Device the opportunity to report if an error was detected during the transfer. All USB Devices are required to include a default Control Endpoint at endpoint number 0, referred to as Endpoint 0. Endpoint 0 is used to implement all the USB Protocol defined Chapter 9 framework and USB Enumeration. In the CLD library Endpoint 0 is also used to handle the USB Audio Device Class v2.0 defined Set and Get requests. These requests are discussed in more detail in the USB Audio Device Class v2.0 Background sections of this document

Interrupt Endpoints are used to transfer blocks of data where data integrity and deterministic timing is required. Deterministic timing is achieved by allowing the Device to specify a requested interval used by the Host to initiate USB transfers, which gives the Device a guaranteed maximum time between opportunities to transfer data. Interrupt Endpoints are particularly useful when the Device needs to report to the Host when a change is detected without having to wait for the Host to ask for the information. This is more efficient then requiring the host to repeatedly send Control Endpoint requests asking if anything has changed.

The flow charts below give an overview of how the CLD Library and the User firmware interact to process Interrupt IN transfers.

CLD Library Interrupt IN Flow Chart



Isochronous Endpoints have the following characteristics which make them well suited for streaming audio data:

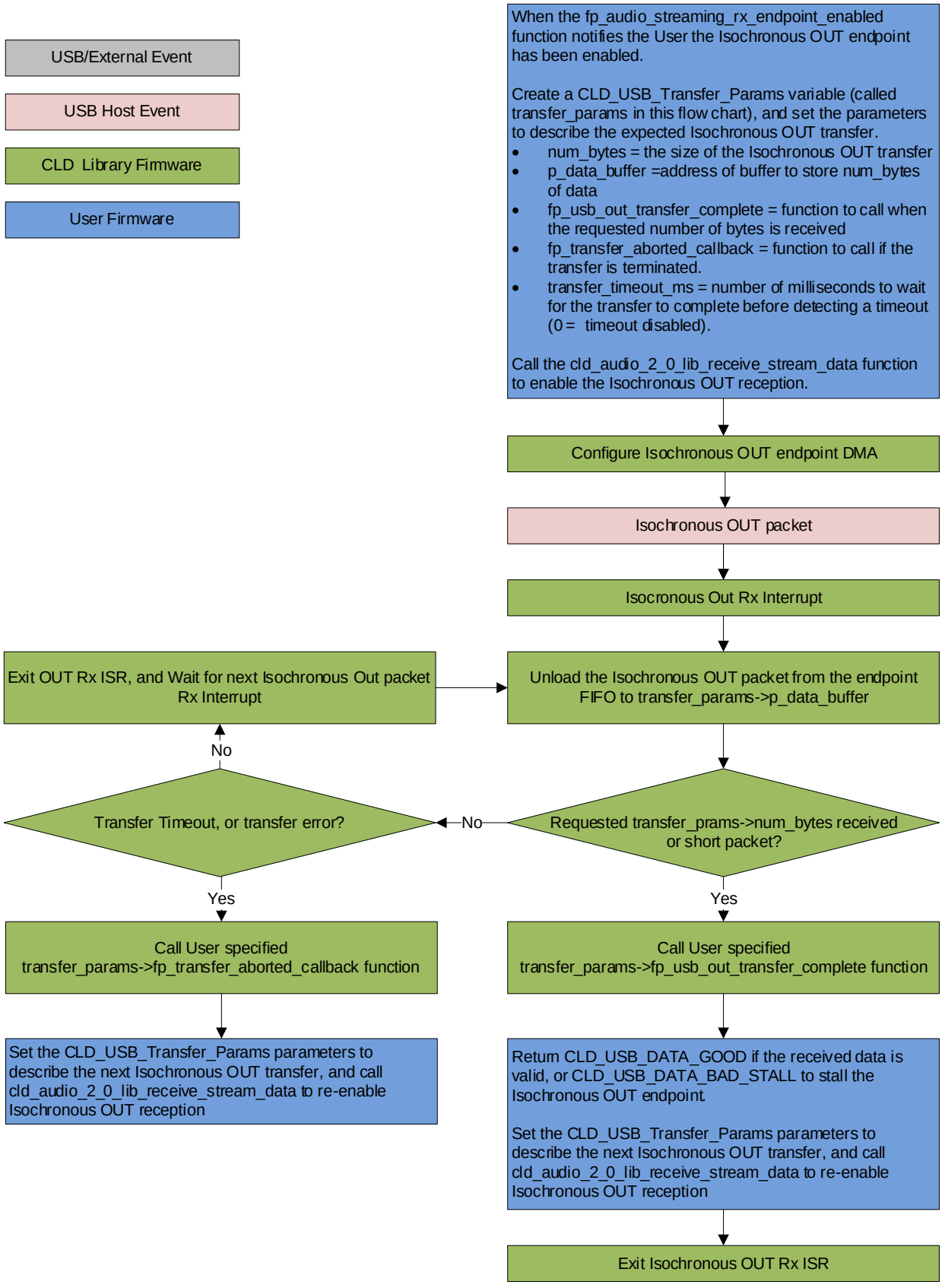
- Guaranteed USB bandwidth with bounded latency
- Constant data rate as long as data is provided to the endpoint.

- In the event of a transport error there is no retrying.

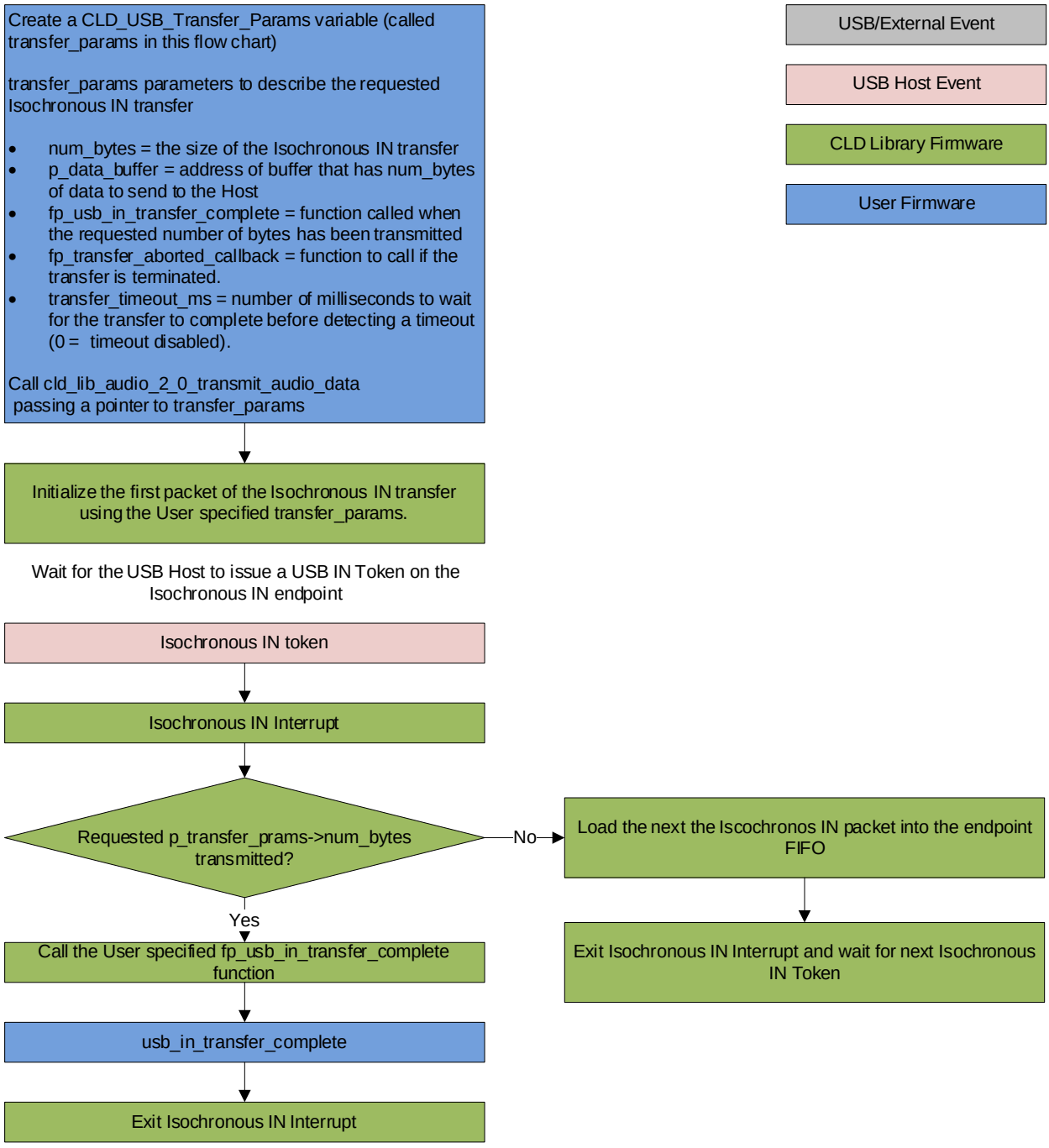
These characteristics allow for streaming audio data to be transmitted with deterministic timing. In the event of a USB transport error the audio data is dropped instead of being retried like a Bulk or Interrupt endpoint. This allows the streaming audio data to remain in sync. The CLD library supports an Isochronous IN and Isochronous OUT endpoint, which are used to send and receive streaming audio data with the USB Host, respectively.

The flow charts below give an overview of how the CLD library and the User firmware interact to process Isochronous OUT and Isochronous IN transfers. Additionally, the User firmware code snippets included at the end of this document provide a basic framework for implementing a USB Audio v2.0 device using the CLD SC594 Audio 2.0 Library.

CLD Audio 2.0 Library Isochronous OUT Flow Chart



CLD Audio 2.0 Library Isochronous IN Flow Chart

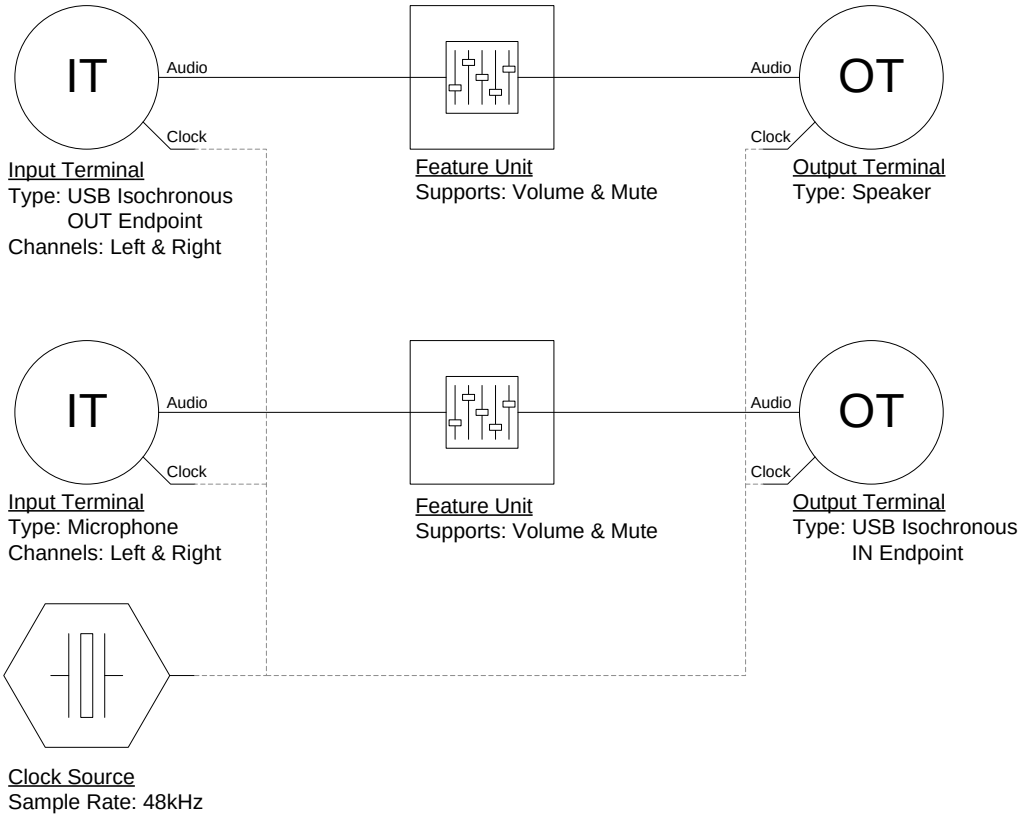


USB Audio Device Class v2.0 Background

The following is a basic overview of some USB Audio Device v2.0 concepts that are necessary to use the CLD SC594 Audio 2.0 Library. However, it is recommended that developers have at least a basic understanding of the USB Audio Device Class v2.0 protocol.

The USB Audio Device Class v2.0 protocol is a USB Standard Class released by the USB IF committee, and it provides a standardized way for a device that is capable of audio input/output to communicate with a USB Host. The USB Audio Device Class v2.0 USB descriptors provide a detailed description of the Device's capabilities. This information includes the Device's supported audio sample rate(s), audio data format, input and output terminals and how the various audio processing components are connected and controlled.

The Device's audio processing capabilities are described using a series of USB Audio Class Terminal and Unit Descriptors. The Terminal Descriptors define how audio data is input and output (speakers, microphones, USB Isochronous endpoints, etc.). The Unit Descriptors describe the Device's audio processing capabilities and how they connect to the input/output Terminals. The diagram below shows how the audio Terminal and Unit entities are connected in the CLD example project to implement a basic device with a stereo speaker output, and stereo input.



More complex audio devices are created by connecting multiple Unit entities together to describe the Device's capabilities. For more information about the available Unit and Terminal entities, and how they are used please refer to the USB Audio Class Device v2.0 specification.

In order to successfully communicate with a USB Audio device the USB Host needs to know how the audio data is formatted. This is done using an audio stream format descriptor, which is part of the Streaming Audio Interface configuration. The USB Audio Device Class v2.0 specification supports multiple audio data formats which are described in the USB Device Class Definition for Audio Data Formats v2.0 specification.

Isochronous Endpoint Bandwidth Allocation

As mentioned previously, one of the advantages of Isochronous endpoints is that they provide guaranteed USB bandwidth. However, this can also be a disadvantage when the bandwidth isn't being used as it is wasted.

To avoid this disadvantage the USB Audio Device Class v2.0 protocol requires that audio data streaming interfaces include two settings. The default setting does not include any Isochronous endpoints so its bandwidth requirement is zero. An alternate interface includes the required Isochronous endpoint(s). This allows the USB Host to enable the Isochronous endpoints when it needs to send or receive audio data, and disable them when the audio device is idle. This switch is done using the USB Chapter 9 Set Interface standard request.

When the CLD SC594 Audio 2.0 Library receives a Set Interface request the appropriate User callback function is called. Please refer to the fp_audio_streaming_rx_endpoint_enabled and fp_audio_streaming_tx_endpoint_enabled function pointer descriptions in the cld_sc594_audio_2_0_lib_init section of this document for more information.

USB Audio Device Class v2.0 Control Endpoint Requests

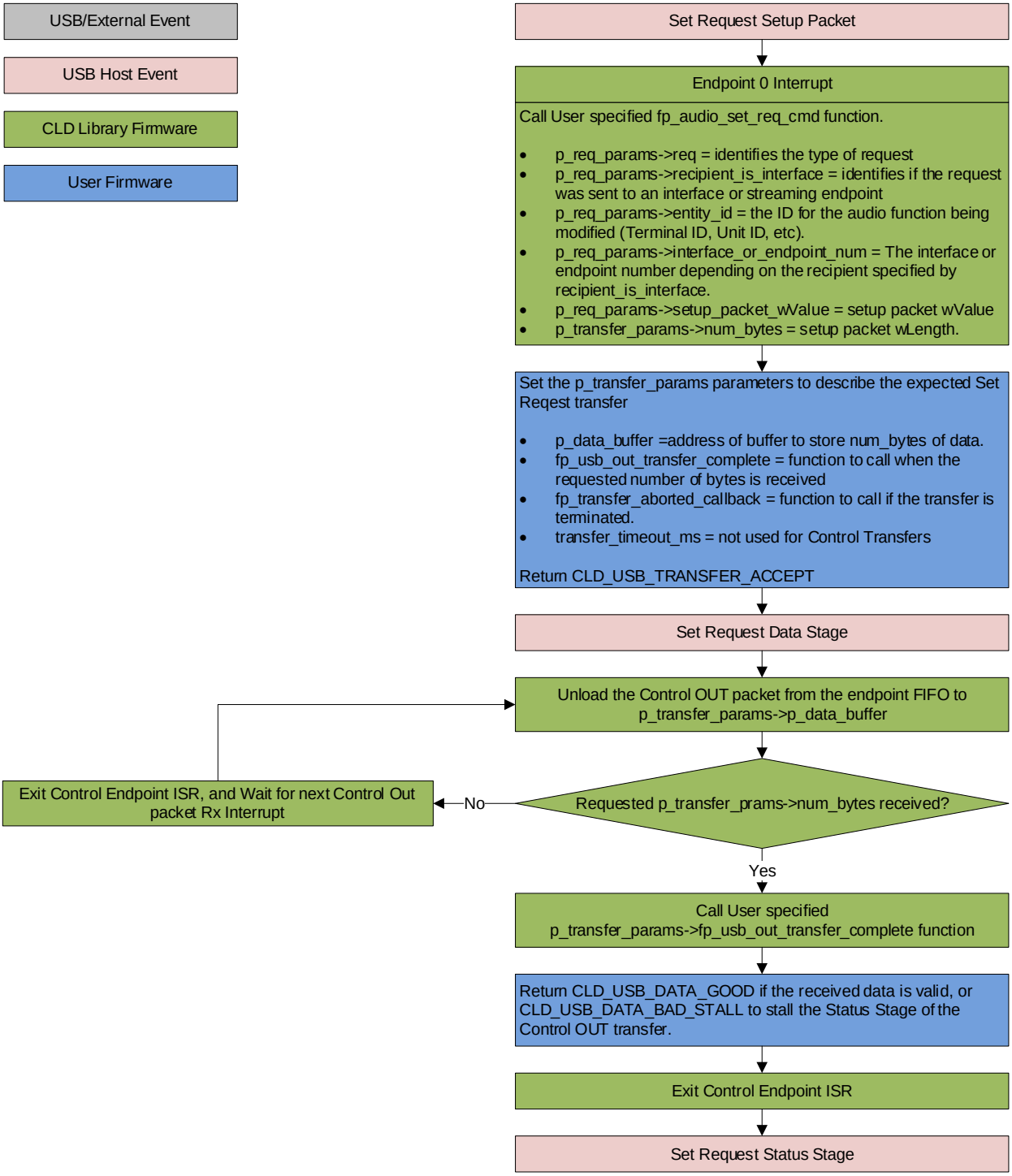
The USB Audio Device Class v2.0 control endpoint requests are broken down into Set and Get requests. These requests are used to control the various Terminal and Unit entities defined in the Configuration Descriptor. The CLD library support for these requests is explained in the following sections.

Additionally, the User firmware code snippets included at the end of this document provide a basic framework for implementing the USB audio Control Endpoint requests using the CLD library.

USB Audio Device Class v2.0 Set Request

The USB Audio Device Class v2.0 Set Request is used to control the audio functions supported by the Device. This includes modifying the attributes if the Unit and Terminal entities as well as controlling features of the streaming audio endpoints.

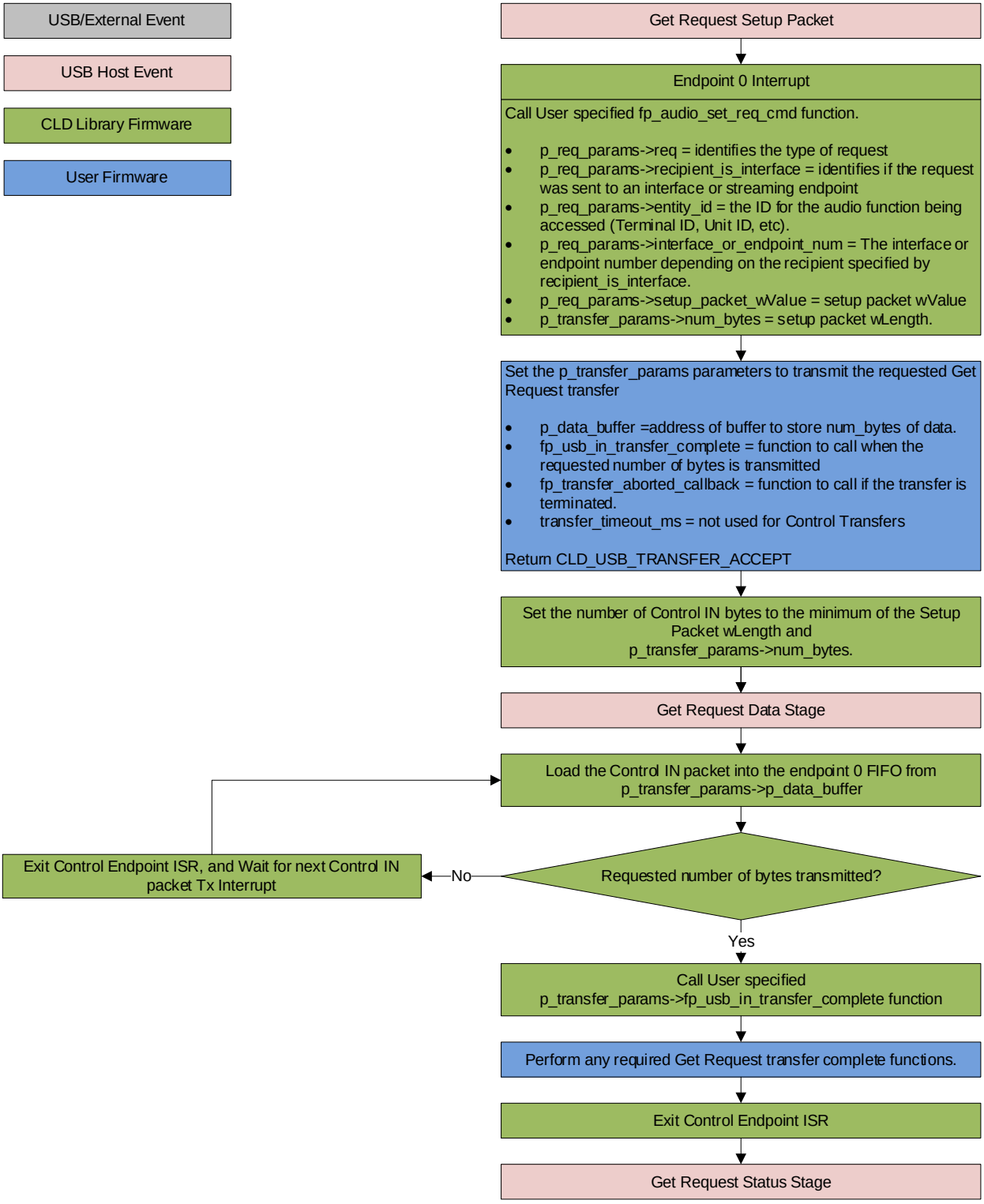
CLD SC594 Audio Device Class v2.0 Set Request Flow Chart



USB Audio Device Class v2.0 Get Request

The Get Request is a Control IN request used by the Host to request data from the audio functions supported by the Device. This includes requesting the attributes of the Unit and Terminal entities as well as features of the audio stream endpoints.

CLD SC594 Audio Device Class v2.0 Get Request Flow Chart



Dependencies

In order to function properly, the CLD SC594 Audio 2.0 Library requires the following resources:

- ULPI (8-PIN interface) compliant USB PHY which outputs a USB clock to the processor.
- The CLD library uses DMA for all USB transfers. Requiring all data transferred over USB to be located in un-cached memory, and be 32-bit aligned. Including buffers used by the CLD library which are located in an ".usb_lib_uncached " memory section. In order for the library to work properly, the User must define the usb_lib_uncached section in their loader file and configure the cache accordingly.
- The User firmware is responsible for enabling the USBC I/O pins in the CCES project Pin Multiplexing project settings.
- The User firmware is responsible for configuring all other non-USB specific peripherals, including clocks, power modes, etc.

CLD SC594 Audio 2.0 Library Scope and Intended Use

The CLD SC594 Audio 2.0 Library implements the USB Audio Device Class v2.0 required functionality to implement a USB Audio device, as well as providing time measurements functionality. The CLD library is designed to be added to an existing User project, and as such only includes the functionality needed to implement the above mentioned USB, and timer keeping features. All other aspects of SC594 processor configuration must be implemented by the User code.

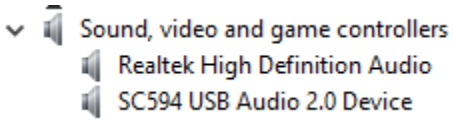
CLD Audio 2.0 Example v1.02 Description

The CLD example project provided with the CLD SC594 Audio 2.0 Library implements a basic USB audio device that supports a single stereo input and stereo output loopback.

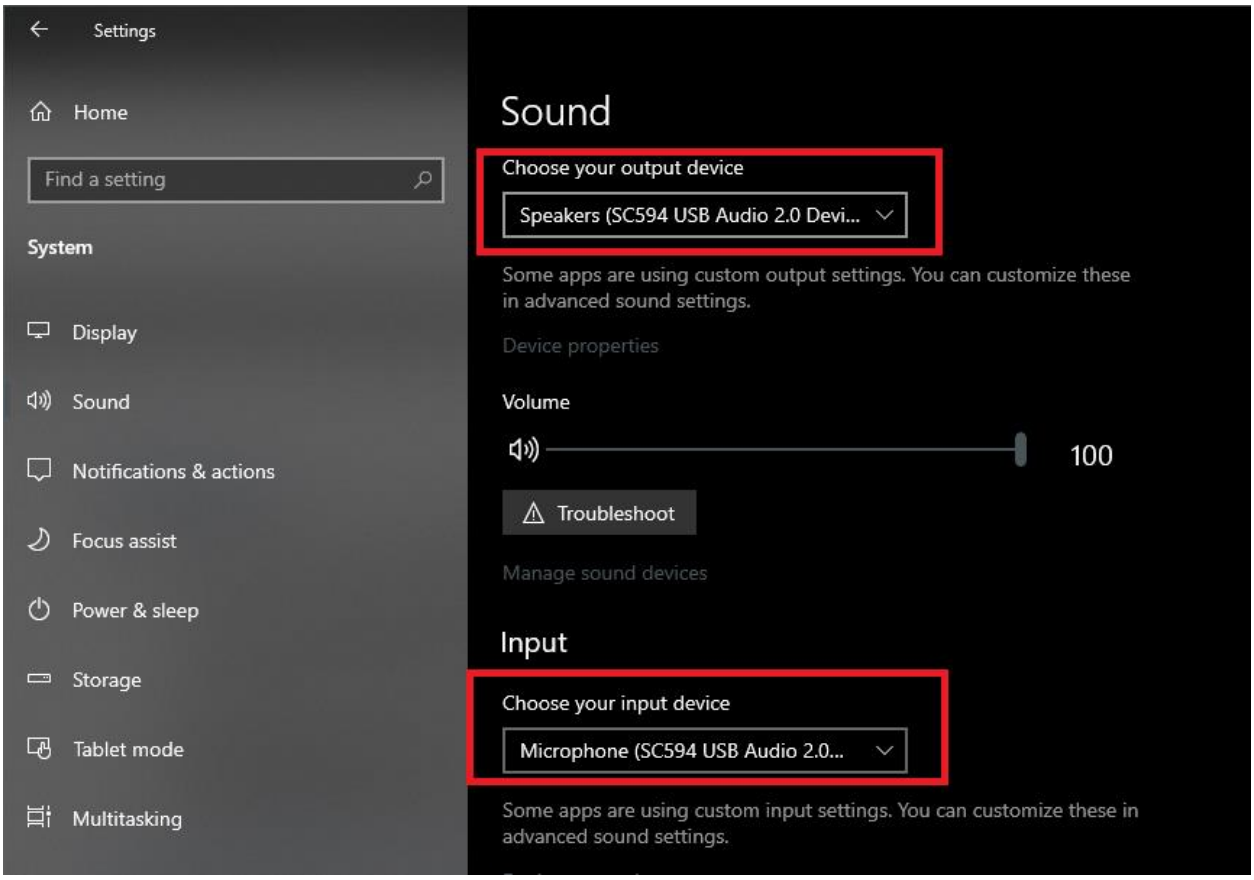
Running the Example Project

1. With the example project was developed using the ADSP SC594 SOM and carrier board, and toggles the LED connected to GPIO port C pin 3 every 250 milliseconds to provide a visual indicator the project is running.

2. Once the example project is running on the EZ Board connect a USB mini-b cable from a PC to the “USB Phy” connector of the carrier board. Windows 10 will install its built-in USB Audio 2.0 driver, and the device will be listed as a USB Audio Device in the Device Manager as shown below. If the SC594 device is not listed in Device Manager, verify the installed version of Windows 10 supports USB Audio 2.0 devices.



3. Under the Sound setting for Windows 10, select the SC594 USB Audio device as the output and input device as shown below:



4. Play an audio file, movie, or other means of outputting audio.

5. The example project will echo the received audio data using its microphone input, which can be seen using Audacity or other audio recording software.

CLD SC594 Audio 2.0 Library API

The following CLD library API descriptions include callback functions that are called by the library based on USB events. The following color code is used to identify if the callback function is called from the USB interrupt service routine, or from mainline. The callback functions called from the USB interrupt service routine are also italicized so they can be identified when printed in black and white.

Callback called from the mainline context
Callback called from the USB interrupt service routine

cld_sc594_audio_2_0_lib_init

```
CLD_RV cld_sc594_audio_2_0_lib_init (CLD_SC594_Audio_2_0_Lib_Init_Params *
p_lib_params)
```

Initializes the CLD SC594 Audio 2.0 Library.

Arguments

p_lib_params	Pointer to a CLD_SC594_Audio_2_0 _Lib_Init_Params structure that has been initialized with the User Application specific data.

Return Value

This function returns the CLD_RV type which represents the status of the CLD library initialization process. The CLD_RV type has the following values:

CLD_SUCCESS	The library was initialized successfully
CLD_FAIL	There was a problem initializing the library
CLD_ONGOING	The library initialization is being processed

Details

The cld_sc594_audio_2_0_lib_init function is called as part of the device initialization and must be repeatedly called until the function returns CLD_SUCCESS or CLD_FAIL. If CLD_FAIL is returned the library will output an error message identifying the cause of the failure using the fp_cld_lib_status function if defined by the User application. Once the library has been initialized successfully the main program loop can start.

The CLD_SC594_Audio_2_0_Lib_Init_Params structure is described below:

```
typedef struct
{
    unsigned short vendor_id;
    unsigned short product_id;
    unsigned char  usb_bus_max_power
    unsigned short device_descriptor_bcdDevice
    unsigned char  phy_hs_timeout_calibration;
    unsigned char  phy_fs_timeout_calibration;
    CLD_Boolean    phy_delay_req_after_ulip_chirp_cmd;

    CLD_RV (*fp_init_usb_phy) (void);
    unsigned char  audio_control_category_code;
```

```
    CLD_Audio_2_0_Control_Interrupt_Params *
            p_audio_control_interrupt_params;

    unsigned char * p_unit_and_terminal_descriptors;
    unsigned short  unit_and_terminal_descriptors_length;

    CLD_Audio_2_0_Stream_Interface_Params *
                  p_audio_streaming_rx_interface_params;

    CLD_Audio_2_0_Rate_Feedback_Params * p_audio_rate_feedback_rx_params;

    CLD_Audio_2_0_Stream_Interface_Params *
                  p_audio_streaming_tx_interface_params;

    CLD_USB_Transfer_Request_Return_Type (*fp_audio_set_req_cmd)
                  (CLD_Audio_2_0_Cmd_Req_Parameters * p_req_params,
                   CLD_USB_Transfer_Params * p_transfer_data);

    CLD_USB_Transfer_Request_Return_Type (*fp_audio_get_req_cmd)
                  (CLD_Audio_2_0_Cmd_Req_Parameters * p_req_params,
                   CLD_USB_Transfer_Params * p_transfer_data);

    void (*fp_audio_streaming_rx_endpoint_enabled) (CLD_Boolean enabled);
    void (*fp_audio_streaming_tx_endpoint_enabled) (CLD_Boolean enabled);

    const char * p_usb_string_manufacturer;
    const char * p_usb_string_product;
    const char * p_usb_string_serial_number;
    const char * p_usb_string_configuration;
    const char * p_usb_string_communication_class_interface;
    const char * p_usb_string_data_class_interface;

    unsigned char user_string_descriptor_table_num_entries;
    CLD_Audio_2_0_Lib_User_String_Descriptors *
                  p_user_string_descriptor_table;

    unsigned short usb_string_language_id;

    void (*fp_cld_usb_event_callback) (CLD_USB_Event event);


    void (*fp_cld_lib_status) (unsigned short status_code,
                               void * p_additional_data,
                               unsigned short additional_data_size);


} CLD_SC594_Audio_2_0_Lib_Init_Params;
```

A description of the CLD_SC594_Audio_2_0_Lib_Init_Params structure elements is included below:

Structure Element	Description
vendor_id	The 16-bit USB vendor ID that is returned to the USB Host in the USB Device Descriptor. USB Vendor ID's are assigned by the USB-IF and can be purchased through their website (www.usb.org).

<table>
<tr><td>product_id</td><td>The 16-bit product ID that is returned to the USB Host in the USB Device Descriptor.</td></tr>
<tr><td>usb_bus_max_power</td><td>USB Configuration Descriptor bMaxPower value (0 = self-powered).
Refer to the USB 2.0 protocol section 9.6.3.</td></tr>
<tr><td>device_descriptor_bcd_device</td><td>USB Device Descriptor bcdDevice value.
Refer to the USB 2.0 protocol section 9.6.1.</td></tr>
<tr><td>phy_hs_timeout_calibration</td><td>High Speed USB timeout PHY calibration value See ADSP-SC59x Hw Reference Manual bits 2:0 of the USBC_CFG register</td></tr>
<tr><td>phy_fs_timeout_calibration</td><td>High Speed USB timeout PHY calibration value See ADSP-SC59x Hw Reference Manual bits 2:0 of the USBC_CFG register</td></tr>
<tr><td>fp_init_usb_phy</td><td>User defined function used to initialize and reset the USB Phy

The fp_init_usb_phy function returns the CLD_RV type, which has the following values:
<table>
<tr><td>Return Value</td><td>Description</td></tr>
<tr><td>CLD_ONGOING</td><td>Results in this function getting additional runtime.</td></tr>
<tr><td>CLD_SUCCESS</td><td>USB Phy initialized successfully.</td></tr>
<tr><td>CLD_FAIL</td><td>Phy initialization failed, causes USB library initialization failure.</td></tr>
</table></td></tr>
<tr><td>audio_control_category_code</td><td>Audio Control Interface Header Descriptor bCategory code
(refer to: USB Device Class Definition of Audio Devices v 2.0 section 4.7.2)</td></tr>
<tr><td>p_audio_control_interrupt_params</td><td>Pointer to the CLD_SC594_Audio_2_0_Control_Interrupt_Params structure that describes the optional Interrupt IN endpoint.

Set to CLD_NULL if not required

The CLD_ Audio_2_0_Control_Interrupt_Params structure contains the following elements:
<table>
<tr><td>Structure Element</td><td>Description</td></tr>
<tr><td>endpoint_number</td><td>Sets the USB endpoint number of the Interrupt IN endpoint.

The endpoint number must be within the following range: 1 ≤ endpoint number ≤ 12. Any other endpoint number will result in the cld_sc594_audio_2_0_lib_init function returning CLD_FAIL</td></tr>
<tr><td>b_interval_full_speed</td><td>Full-Speed polling interval in the USB Endpoint Descriptor. (See USB 2.0 section 9.6.6)</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td><table>
<tr><td>b_interval_high_speed</td><td>High-Speed polling interval in the USB Endpoint Descriptor. (See USB 2.0 section 9.6.6)</td></tr>
</table></td></tr>
<tr><td>p_unit_and_terminal_descriptors</td><td>Pointer to the Unit and Terminal Descriptors which are part of the Audio Control interface in the USB Configuration Descriptor.</td></tr>
<tr><td>unit_and_terminal_descriptors_length</td><td>The length of the Unit and Terminal Descriptors addressed by p_unit_and_terminal_descriptors.</td></tr>
<tr><td>p_audio_streaming_rx_interface_params</td><td>Pointer to a CLD_Audio_2_0_Stream_Interface_Params structure that describes how the Isochronous OUT endpoint and related USB Audio Streaming interface should be configured. The a CLD_Audio_2_0_Stream_Interface_Params structure contains the following elements:

<table>
<tr><td>Structure Element</td><td>Description</td></tr>
<tr><td>endpoint_num</td><td>Sets the USB endpoint number of the Isochronous endpoint. The endpoint number must be within the following range:

$1 \leq$ endpoint num ≤ 12. Any other endpoint number will result in the cld_sc594_audio_2_0_lib_init function returning CLD_FAIL</td></tr>
<tr><td>max_packet_size_full_speed</td><td>Sets the Isochronous endpoint's max packet size when operating at Full Speed. The maximum max packet size is 1023 bytes.</td></tr>
<tr><td>max_packet_size_high_speed</td><td>Sets the Isochronous endpoint's max packet size when operating at High Speed. The maximum max packet size is 1024 bytes.</td></tr>
<tr><td>b_interval_full_speed</td><td>Full-Speed polling interval in the USB Endpoint Descriptor. (See USB 2.0 section 9.6.6)</td></tr>
<tr><td>b_interval_high_speed</td><td>High-Speed polling interval in the USB Endpoint Descriptor. (See USB 2.0 section 9.6.6)</td></tr>
<tr><td>b_terminal_link</td><td>The Terminal ID of the Terminal connected to this endpoint.</td></tr>
<tr><td>b_format_type</td><td>Format type of the streaming interface</td></tr>
<tr><td>bm_formats</td><td>Supported audio format bitmap.</td></tr>
<tr><td>b_nr_channels</td><td>Number of audio channels supported by the streaming interface.</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>i_channel_config</td><td>Index of the string descriptor describing the first physical channel. These strings should be defined in the user_string_descriptor_table.</td></tr>
<tr><td>p_encoder_descriptor</td><td>Pointer to an optional USB Audio 2.0 Encoder descriptor.</td></tr>
<tr><td>p_decoder_descriptor</td><td>Pointer to an optional USB Audio 2.0 Decoder descriptor.</td></tr>
<tr><td>p_format_descriptor</td><td>Pointer to the format descriptor defined in the USB Device Class Definition for Audio Data Formats v2.0 specification.</td></tr>
<tr><td>p_audio_stream_endpoint_data_descriptor</td><td>Pointer to the Audio Streaming endpoint data descriptor (See USB Device Class Definition for Audio Devices v2.0 section 4.10.1.2).</td></tr>
</table>
</td></tr>
<tr><td>p_audio_rate_feedback_rx_params</td><td>Pointer to a CLD_Audio_2_0_Rate_Feedback_Params structure that describes how the Isochronous IN feedback endpoint. The a CLD_Audio_2_0_Rate_Feedback_Params structure contains the following elements:
<table>
<tr><td>Structure Element</td><td>Description</td></tr>
<tr><td>max_packet_size_full_speed</td><td>Sets the Isochronous endpoint's max packet size when operating at Full Speed. The maximum max packet size is 1023 bytes.</td></tr>
<tr><td>max_packet_size_high_speed</td><td>Sets the Isochronous endpoint's max packet size when operating at High Speed. The maximum max packet size is 1024 bytes.</td></tr>
<tr><td>b_interval_full_speed</td><td>Full-Speed polling interval in the USB Endpoint Descriptor. (See USB 2.0 section 9.6.6)</td></tr>
<tr><td>b_interval_high_speed</td><td>High-Speed polling interval in the USB Endpoint Descriptor. (See USB 2.0 section 9.6.6)</td></tr>
</table>
</td></tr>
<tr><td>p_audio_streaming_tx_interface_params</td><td>Pointer to a CLD_Audio_2_0_Stream_Interface_Params structure that describes how the Isochronous IN endpoint and related USB Audio Streaming interface should be configured.
Refer to the p_audio_streaming_rx_interface_params description (above) for information about the CLD_SC594_Audio_2_0_Stream_Interface_Params structure.</td></tr>
<tr><td>fp_audio_set_req_cmd</td><td>Pointer to the function that is called when a USB Audio Device Class v2.0 Set Request is received. This function has a pointer to the CLD_USB_Transfer_Params structure ('p_transfer_data'), and</td></tr>
</table>

a pointer to the CLD_ Audio_2_0_Cmd_Req_Parameters (p_req_params) as its parameters.

The following CLD_Audio_2_0_Cmd_Req_Parameters structure elements are used to processed a Set Request:

Structure Element	Description

req	Identifies the type of request. The valid types if requests are listed below:
	`CLD_REQ_CURRENT`
	`CLD_REQ_RANGE`
	`CLD_REQ_MEMORY`

recipient_is_interface	Identifies if the request was sent to an interface or Audio streaming endpoint
entity_id	The ID for the audio function being modified (Terminal ID, Unit ID, etc)
interface_or_endpoint_num	The interface or endpoint number for the request depending on the recipient specified by the recipient_is_interface parameter.
setup_packet_wValue	wValue field from the USB Setup Packet.

The following CLD_USB_Transfer_Params structure elements are used to processed a Set Request:

Structure Element	Description
num_bytes	The number of bytes from the Setup Packet wLength field, which is the number of bytes that will be transferred to p_data_buffer before calling the fp_usb_out_transfer_complete callback function.
p_data_buffer	Pointer to the data buffer to store the Set Reqeust data. The size of the buffer should be greater than or equal to the value in num_bytes.
fp_usb_out_transfer_complete	Function called when num_bytes of data has been written to the p_data_buffer memory.
fp_transfer_aborted_callback	Function called if there is a problem receiving the data, or

	if the transfer is interrupted.
transfer_timeout_ms	Not used for Control Requests since the Host has the ability to interrupt any Control transfer.

The fp_audio_set_req_cmd function returns the CLD_USB_Transfer_Request_Return_Type, which has the following values:

Return Value	Description
CLD_USB_TRANSFER_ACCEPT	Notifies the CLD Library that the Set Request data should be accepted using the p_transfer_data values.
CLD_USB_TRANSFER_PAUSE	Requests that the CLD Library pause the Set Request transfer. This causes the Control Endpoint to be nak'ed until the transfer is resumed by calling cld_audio_2_0_lib_ resume_paused_control_ transfer.
CLD_USB_TRANSFER_DISCARD	Requests that the CLD Library discard the number of bytes specified in p_transfer_params-> num_bytes. In this case the library accepts the Set Request from the USB Host but discards the data.
CLD_USB_TRANSFER_STALL	This notifies the CLD Library that there is an error and the request should be stalled.

fp_audio_get_req_cmd

Pointer to the function that is called when a USB Audio Device Class v2.0 Get Request is received. This function has a pointer to the CLD_USB_Transfer_Params structure ('p_transfer_data'), and a pointer to the CLD_Audio_2_0_Cmd_Req_Parameters (p_req_params) as its parameters.

The following CLD_Audio_2_0_Cmd_Req_Parameters structure elements are used to processed a Get Request:

Structure Element	Description

req	Identifies the type of request. The valid types if requests are listed below:
	`CLD_REQ_CURRENT`
	`CLD_REQ_RANGE`
	`CLD_REQ_MEMORY`

recipient_is_interface	Identifies if the request was sent to an interface or Audio

	streaming endpoint
entity_id	The ID for the audio function being accessed (Terminal ID, Unit ID, etc)
interface_or_endpoint_num	The interface or endpoint number for the request depending on the recipient specified by the recipient_is_interface parameter.
setup_packet_wValue	wValue field from the USB Setup Packet.

The following CLD_USB_Transfer_Params structure elements are used to processed a Set Request:

Structure Element	Description
num_bytes	The number of bytes from the Setup Packet wLength field, which is the number of bytes that the device can send from p_data_buffer before calling the fp_usb_out_transfer_ complete callback function.
p_data_buffer	Pointer to the data buffer used to source the Get Request data. The size of the buffer should be greater than or equal to the value in num_bytes.
fp_usb_in_transfer_complete	Function called when num_bytes of data has been transmitted to the USB Host.
fp_transfer_aborted_callback	Function called if there is a problem transmitting the data, or if the transfer is interrupted.
transfer_timeout_ms	Not used for Control Requests since the Host has the ability to interrupt any Control transfer.

The fp_audio_get_req_cmd function returns the CLD_USB_Transfer_Request_Return_Type, which has the following values:

Return Value	Description
CLD_USB_TRANSFER_ACCEPT	Notifies the CLD library that the Get Request data should be transmitted using the p_transfer_data values.
CLD_USB_TRANSFER_PAUSE	Requests that the CLD library

	<table><tr><td></td><td>pause the Get Request transfer. This causes the Control Endpoint to be nak'ed until the transfer is resumed by calling cld_audio_2_0_lib_ resume_paused_control_ transfer.</td></tr><tr><td>CLD_USB_TRANSFER_DISCARD</td><td>Requests that the CLD library to return a zero length packet in response to the Get Request.</td></tr><tr><td>CLD_USB_TRANSFER_STALL</td><td>This notifies the CLD library that there is an error and the request should be stalled.</td></tr></table>
fp_audio_streaming_rx_endpoint_ enabled	Function called when the Isochronous OUT streaming interface is enabled/disabled by the USB Host using the Set Interface command.
fp_audio_streaming_tx_endpoint_ enabled	Function called when the Isochronous IN streaming interface is enabled/disabled by the USB Host using the Set Interface command.
p_usb_string_manufacturer	Pointer to the null-terminated string. This string is used by the library to generate the Manufacturer USB String Descriptor. If the Manufacturer String Descriptor is not used set p_usb_string_manufacturer to CLD_NULL.
p_usb_string_product	Pointer to the null-terminated string. This string is used by the CLD library to generate the Product USB String Descriptor. If the Product String Descriptor is not used set p_usb_string_product to CLD_NULL.
p_usb_string_serial_number	Pointer to the null-terminated string. This string is used by the CLD library to generate the Serial Number USB String Descriptor. If the Serial Number String Descriptor is not used set p_usb_string_serial_number to CLD_NULL.
p_usb_string_configuration	Pointer to the null-terminated string. This string is used by the CLD library to generate the Configuration USB String Descriptor. If the Configuration String Descriptor is not used set p_usb_string_configuration to CLD_NULL.
p_usb_string_audio_control_interface	Pointer to the null-terminated string. This string is used by the CLD library to generate the Audio Control Interface USB String Descriptor. If this interface String Descriptor is not used set it to CLD_NULL.
p_usb_string_audio_streaming_ out_interface	Pointer to the null-terminated string. This string is used by the CLD library to generate the Audio OUT Streaming Interface USB String Descriptor. If this interface String Descriptor is not used set it to CLD_NULL.
p_usb_string_audio_streaming_in _interface	Pointer to the null-terminated string. This string is used by the CLD library to generate the Audio IN Streaming Interface USB String Descriptor. If this interface String Descriptor is not used set it to CLD_NULL.
user_string_descriptor_table_num _entries	The number of entries in the array of CLD_Audio_2_0_Lib_User_String_Descriptors structures addressed by p_user_string_descriptor_table. Set to 0 if

<table>
<tr><td></td><td>p_user_string_descriptor_table is set to CLD_NULL.</td></tr>
<tr><td>p_user_string_descriptor_table</td><td>Pointer to an array of CLD_Audio_2_0_Lib_User_ String_Descriptors structures used to define any custom User defined USB string descriptors. This table is used to define any USB String descriptors for any string descriptor indexes that are used in the Terminal or Unit Descriptors.

Set to CLD_NULL is not used.

The CLD_Audio_2_0_Lib_User_String_Descriptors structure elements are explained below:
<table>
<tr><th>Structure Element</th><th>Description</th></tr>
<tr><td>string_index</td><td>The USB String Descriptor index for the string. The string_index value is set to the index specified in the Terminal or Unit Descriptor associated with this string.</td></tr>
<tr><td>p_string</td><td>Pointer to a null terminated string.</td></tr>
</table></td></tr>
<tr><td>usb_string_language_id</td><td>16-bit USB String Descriptor Language ID Code as defined in the USB Language Identifiers (LANGIDs) document (www.usb.org/developers/docs/USB_LANGIDs.pdf).
0x0409 = English (United States)</td></tr>
<tr><td>fp_cld_usb_event_callback</td><td>Function that is called when one of the following USB events occurs. This function has a single CLD_USB_Event parameter.

Note: This callback can be called from the USB interrupt or mainline context depending on which USB event was detected. The CLD_USB_Event values in the table below are highlighted to show the context the callback is called for each event.

The CLD_USB_Event has the following values:
<table>
<tr><th>Return Value</th><th>Description</th></tr>
<tr><td>CLD_USB_CABLE_CONNECTED</td><td>USB Cable Connected.</td></tr>
<tr><td>CLD_USB_CABLE_DISCONNECTED</td><td>USB Cable Disconnected</td></tr>
<tr><td>CLD_USB_ENUMERATED_CONFIGURED_FS</td><td>USB device enumerated (USB Configuration set to a non-zero value) at Full-Speed</td></tr>
<tr><td>CLD_USB_ENUMERATED_CONFIGURED_HS</td><td>USB device enumerated (USB Configuration set to a non-zero value) at High-Speed</td></tr>
<tr><td>CLD_USB_UN_CONFIGURED</td><td>USB Configuration set to 0</td></tr>
<tr><td>CLD_USB_BUS_RESET</td><td>USB Bus reset received</td></tr>
</table>
Note: Set to CLD_NULL if not required by application</td></tr>
</table>

<table>
<tr>
<td>fp_cld_lib_status</td>
<td>Pointer to the function that is called when the CLD library has a status to report. This function has the following parameters:

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>status_code</td><td>16-bit status code. If the most significant bit is a '1' the status being reported is an Error.</td></tr>
<tr><td>p_additional_data</td><td>Pointer to additional data included with the status.</td></tr>
<tr><td>additional_data_size</td><td>The number of bytes in the specified additional data.</td></tr>
</table>

If the User plans on processing outside of the fp_cld_lib_status function they will need to copy the additional data to a User buffer.</td>
</tr>
</table>

cld_sc594_audio_2_0_lib_main

```
void cld_sc594_audio_2_0_lib_main (void)
```

CLD SC594 Audio 2.0 Library mainline function

Arguments

None

Return Value

None.

Details

The cld_sc594_audio_2_0_lib_main function is the CLD library mainline function that must be called in every iteration of the main program loop in order for the library to function properly.

cld_audio_2_0_lib_receive_stream_data

```
CLD_USB_Data_Receive_Return_Type cld_audio_2_0_lib_receive_stream_data
(CLD_USB_Transfer_Params * p_transfer_data)
```

CLD Audio 2.0 Library function used to receive data over the Isochronous OUT endpoint.

Arguments

p_transfer_data	Pointer to a CLD_USB_Transfer_Params structure used to describe the data being received.

Return Value

This function returns the CLD_USB_Data_Receive_Return_Type type which reports if the Isochronous OUT transmission has been configured. CLD_USB_Data_Receive_Return_Type has the following values:

CLD_USB_TRANSMIT_SUCCESSFUL	The library has configured the requested Isochronous IN transfer.
CLD_USB_TRANSMIT_FAILED	The library failed to configure the requested Isochronous OUT transfer. This will happen if the Isochronous OUT endpoint is busy, or if the p_transfer_data-> data_buffer is set to CLD_NULL
CLD_USB_RECEIVE_FAILED_MISALIGNED	The requested USB transfer failed because the specified memory location isn't 32-bit aligned.
CLD_USB_RECEIVE_FAILED_NUM_BYTES	The transfer failed because the num_bytes field of the passed CLD_USB_Transfer_Params structure was not a multiple of the endpoint max packet size. Note: the max packet size is determined based on the values specified by the User, and the enumerated USB speed.

Details

The cld_audio_2_0_lib_receive_stream_data enables the Isochronous OUT endpoint to receive the data specified by the p_transfer_data parameter from the USB Host. This function should be called when the streaming RX endpoint is enabled, in fp_usb_out_transfer_complete, and in fp_transfer_aborted_callback.

The CLD_USB_Transfer_Params structure is described below.

Structure Element	Description
num_bytes	The number of bytes to transfer to the USB Host. Once the specified number of bytes has been transmitted the fp_usb_in_transfer_complete callback function will be called.
p_data_buffer	Pointer to the data to be sent to the USB Host. This buffer must include the number of bytes specified by num_bytes.
fp_usb_out_*transfer*_complete	Function called when the specified data has been received, or the Host send a short packet (less than the max packet size) signaling the end of a transfer. This function is passed the number of received bytes.
fp_usb_in_transfer_complete	Not used for OUT transfers.
fp_transfer_aborted_callback	Function called if there is a problem receiving the data to the USB

	Host. This function can be set to CLD_NULL if the User application doesn't want to be notified if a problem occurs.
transfer_timeout_ms	Isochronous OUT transfer timeout in milliseconds. If the Isochronous OUT transfer takes longer then this timeout the transfer is aborted and the fp_transfer_aborted_callback is called. Setting the timeout to 0 disables the timeout

cld_audio_2_0_lib_transmit_audio_data

```
CLD_USB_Data_Transmit_Return_Type cld_audio_2_0_lib_transmit_audio_data
      (CLD_USB_Transfer_Params * p_transfer_data)
```

CLD Audio 2.0 Library function used to send data over the Isochronous IN endpoint.

Arguments

p_transfer_data	Pointer to a CLD_USB_Transfer_Params structure used to describe the data being transmitted.

Return Value

This function returns the CLD_USB_Data_Transmit_Return_Type type which reports if the Isochronous IN transmission request was started. The CLD_USB_Data_Transmit_Return_Type type has the following values:

CLD_USB_TRANSMIT_SUCCESSFUL	The library has started the requested Isochronous IN transfer.
CLD_USB_TRANSMIT_FAILED	The library failed to start the requested Isochronous IN transfer. This will happen if the Isochronous IN endpoint is busy, or if the p_transfer_data-> data_buffer is set to CLD_NULL
CLD_USB_TRANSMIT_FAILED_MISALIGNED	The requested USB transfer failed because the specified memory location isn't 32-bit aligned.

Details

The cld_audio_2_0_lib_transmit_audio_data function transmits the data specified by the p_transfer_data parameter to the USB Host using the Device's Isochronous IN endpoint.

The CLD_USB_Transfer_Params structure is described below.

```
typedef struct
{
    unsigned long num_bytes;
    unsigned char * p_data_buffer;
    union
    {
        CLD_USB_Data_Received_Return_Type (*fp_usb_out_transfer_complete)(void);
        void (*fp_usb_in_transfer_complete) (void);
    }callback;
    void (*fp_transfer_aborted_callback) (void);
    CLD_Time transfer_timeout_ms;
} CLD_USB_Transfer_Params;
```

A description of the CLD_USB_Transfer_Params structure elements is included below:

Structure Element	Description
num_bytes	The number of bytes to transfer to the USB Host. Once the specified number of bytes has been transmitted the fp_usb_in_transfer_complete callback function will be called.
p_data_buffer	Pointer to the data to be sent to the USB Host. This buffer must include the number of bytes specified by num_bytes.
fp_usb_out_transfer_complete	Not Used for Isochronous IN transfers
fp_usb_in_transfer_complete	Function called when the specified data has been transmitted to the USB Host. This function pointer can be set to CLD_NULL if the User application doesn't want to be notified when the data has been transferred.
fp_transfer_aborted_callback	Function called if there is a problem transmitting the data to the USB Host. This function can be set to CLD_NULL if the User application doesn't want to be notified if a problem occurs.
transfer_timeout_ms	Isochronous IN transfer timeout in milliseconds. If the Isochronous IN transfer takes longer then this timeout the transfer is aborted and the fp_transfer_aborted_callback is called. Setting the timeout to 0 disables the timeout

cld_audio_2_0_lib_transmit_interrupt_data

```
CLD_USB_Data_Transmit_Return_Type cld_audio_2_0_lib_transmit_interrupt_data
      (CLD_USB_Transfer_Params * p_transfer_data)
```

CLD Audio 2.0 Library function used to send data over the optional Interrupt IN endpoint.

Arguments

p_transfer_data	Pointer to a CLD_USB_Transfer_Params structure used to describe the data being transmitted.

Return Value

This function returns the CLD_USB_Data_Transmit_Return_Type type which reports if the Interrupt IN transmission request was started. The CLD_USB_Data_Transmit_Return_Type type has the following values:

CLD_USB_TRANSMIT_SUCCESSFUL	The library has started the requested Interrupt IN transfer.
CLD_USB_TRANSMIT_FAILED	The library failed to start the requested Interrupt IN transfer. This will happen if the Interrupt IN endpoint is disabled, is busy, if the number of bytes isn't 6, or if the p_transfer_data-> data_buffer is set to CLD_NULL
CLD_USB_TRANSMIT_FAILED_MISALIGNED	The requested USB transfer failed because the specified memory location isn't 32-bit aligned.

Details

The cld_audio_2_0_lib_transmit_interrupt_data function transmits the data specified by the p_transfer_data parameter to the USB Host using the Device's Interrupt IN endpoint.

According to the USB Device Class Definition for Audio Devices v2.0 the Interrupt IN message is a fixed size (6 bytes), so if the User tries to transfer more, or less, then 6 bytes the cld_audio_2_0_w_lib_transmit_interrupt_data function will return CLD_USB_TRANSMIT_FAILED.

The CLD_USB_Transfer_Params structure is described below.

```
typedef struct
{
    unsigned long num_bytes;
    unsigned char * p_data_buffer;
    union
    {
        CLD_USB_Data_Received_Return_Type (*fp_usb_out_transfer_complete)(void);
        void (*fp_usb_in_transfer_complete) (void);
    }callback;
    void (*fp_transfer_aborted_callback) (void);
    CLD_Time transfer_timeout_ms;
} CLD_USB_Transfer_Params;
```

A description of the CLD_USB_Transfer_Params structure elements is included below:

Structure Element	Description
num_bytes	The number of bytes to transfer to the USB Host. Once the specified number of bytes has been transmitted the fp_usb_in_transfer_complete callback function will be called.
p_data_buffer	Pointer to the data to be sent to the USB Host. This buffer must include the number of bytes specified by num_bytes.
fp_usb_out_transfer_complete	Not Used for Interrupt IN transfers
fp_usb_in_transfer_complete	Function called when the specified data has been transmitted to the USB Host. This function pointer can be set to CLD_NULL if the User application doesn't want to be notified when the data has been transferred.
fp_transfer_aborted_callback	Function called if there is a problem transmitting the data to the USB Host. This function can be set to CLD_NULL if the User application doesn't want to be notified if a problem occurs.
transfer_timeout_ms	Interrupt IN transfer timeout in milliseconds. If the Interrupt IN transfer takes longer then this timeout the transfer is aborted and the fp_transfer_aborted_callback is called. Setting the timeout to 0 disables the timeout

cld_audio_2_0_lib_transmit_audio_rate_feedback_data

```
CLD_USB_Data_Transmit_Return_Type
cld_audio_2_0_lib_transmit_audio_rate_feedback_data
      (CLD_USB_Audio_Feedback_Params * p_transfer_data)
```

CLD Audio 2.0 Library function used to transfer audio OUT rate feedback data over the optional rate feedback Isochronous IN endpoint.

Arguments

CLD_USB_Audio_Feedback_Params	Pointer to a CLD_USB_Audio_Feedback_Params structure used to describe the data being transmitted.

Return Value

This function returns the CLD_USB_Data_Transmit_Return_Type type which reports if the Interrupt IN transmission request was started. The CLD_USB_Data_Transmit_Return_Type type has the following values:

CLD_USB_TRANSMIT_SUCCESSFUL	The library has scheduled the requested Isochronous IN transfer.
CLD_USB_TRANSMIT_FAILED	The library failed to schedule the requested Isochronous IN transfer. This will happen if the Isochronous IN endpoint is disabled, or busy.

Details

The cld_audio_2_0_lib_transmit_audio_rate_feedback_data function transmits the data specified by the p_transfer_data parameter to the USB Host using the Device's Isochronous IN endpoint.

The CLD_USB_Audio_Feedback_Params structure is described below.

```
typedef struct
{
    float desired_data_rate;
    void (*fp_usb_in_transfer_complete) (void);
    void (*fp_transfer_aborted_callback) (void);
    CLD_Time transfer_timeout_ms;
} CLD_USB_Audio_Feedback_Params;
```

A description of the CLD_USB_Audio_Feedback_Params structure elements is included below:

Structure Element	Description
desired_data_rate	Feeback value in kHz (for example use 44.1 for 44.1kHz)
fp_usb_in_transfer_complete	Function called when the specified data has been transmitted to the USB Host. This function pointer can be set to CLD_NULL if the User application doesn't want to be notified when the data has been transferred.
fp_transfer_aborted_callback	Function called if there is a problem transmitting the data to the USB Host. This function can be set to CLD_NULL if the User application doesn't want to be notified if a problem occurs.
transfer_timeout_ms	Interrupt IN transfer timeout in milliseconds. If the Interrupt IN

	transfer takes longer then this timeout the transfer is aborted and the fp_transfer_aborted_callback is called. Setting the timeout to 0 disables the timeout

cld_audio_2_0_lib_resume_paused_control_transfer

```
void cld_audio_2_0_lib_resume_paused_control_transfer (void)
```

CLD library function used to resume a paused Control endpoint transfer.

Arguments

None

Return Value

None.

Details

The cld_audio_2_0_lib_resume_paused_control_transfer function is used to resume a Control transfer which was paused by the `fp_audio_set_req_cmd, or fp_audio_get_req_cmd` function returning CLD_USB_TRANSFER_PAUSE. When called the cld_audio_2_0_lib_resume_paused_control_transfer function will call the User application's `fp_audio_set_req_cmd, or fp_audio_get_req_cmd` function passing the CLD_USB_Transfer_Params of the original paused transfer. The User function can then chose to accept, discard, or stall the Control endpoint request.

cld_lib_usb_connect

```
void cld_lib_usb_connect (void)
```

CLD Library function used to connect to the USB Host.

Return Value
None.

Details
The cld_lib_usb_connect function is called after the CLD library has been initialized to connect the USB device to the Host.

cld_ lib_usb_disconnect

```
void cld_lib_usb_disconnect (void)
```

CLD library function used to disconnect from the USB Host.

Return Value
None.

Details
The cld_lib_usb_disconnect function is called after the CLD library has been initialized to disconnect the USB device to the Host.

cld_time_125us_tick

```
void cld_time_125us_tick (void)
```

CLD library timer function that should be called once per 125 microseconds.

Arguments
None

Return Value
None.

Details
This function should be called once every 125 microseconds in order to the CLD to processed periodic events.

cld_usb0_isr_callback

```
void cld_usb0_isr_callback (void)
```

CLD library USB interrupt service routines

Arguments
None

Return Value
None.

Details
These USB ISR functions should be called from the corresponding USB Port Interrupt Service Routines as shown in the CLD provided example projects.

cld_time_get

```
CLD_Time cld_time_get(void)
```

CLD library function used to get the current CLD time in milliseconds.

Arguments
None

Return Value
The current CLD library time.

Details
The cld_time_get function is used in conjunction with the cld_time_passed_ms function to measure how much time has passed between the cld_time_get and the cld_time_passed_ms function calls in milliseconds.

cld_time_passed_ms

```
CLD_Time cld_time_passed_ms(CLD_Time time)
```

CLD library function used to measure the amount of time that has passed in milliseconds.

Arguments

`time`	A CLD_Time value returned by a cld_time_get function call.

Return Value

The number of milliseconds that have passed since the cld_time_get function call that returned the CLD_Time value passed to the cld_time_passed_ms function.

Details

The cld_time_passed_ms function is used in conjunction with the cld_time_get function to measure how much time has passed between the cld_time_get and the cld_time_passed_ms function calls in milliseconds.

cld_time_get_125us

```
CLD_Time cld_time_get_125us(void)
```

CLD library function used to get the current CLD time in 125 microsecond increments.

Arguments

None

Return Value

The current CLD library time.

Details

The cld_time_get_125us function is used in conjunction with the cld_time_passed_125us function to measure how much time has passed between the cld_time_get_125us and the cld_time_passed_125us function calls in 125 microsecond increments.

cld_time_passed_125us

```
CLD_Time cld_time_passed_125us(CLD_Time time)
```

CLD library function used to measure the amount of time that has passed in 125 microsecond increments.

Arguments

time	A CLD_Time value returned by a cld_time_get_125us function call.

Return Value

The number of 125microsecond increments that have passed since the cld_time_get_125us function call that returned the CLD_Time value passed to the cld_time_passed_125us function.

Details

The cld_time_passed_125us function is used in conjunction with the cld_time_get_125us function to measure how much time has passed between the cld_time_get_125us and the cld_time_passed_125us function calls in 125 microsecond increments.

cld_lib_status_decode

```
char * cld_lib_status_decode (unsigned short status_cod,
                              void * p_additional_data,
                              unsigned short additional_data_size)
```

CLD Library function that returns a NULL terminated string describing the status passed to the function.

Arguments

status_code	16-bit status code returned by the CLD library. Note: If the most significant bit is a '1' the status is an error.
p_additional_data	Pointer to the additional data returned by the CLD library (if any).
additional_data_size	Size of the additional data returned by the CLD library.

Return Value

This function returns a decoded Null terminated ASCII string.

Details

The cld_lib_status_decode function can be used to generate an ASCII string which describes the CLD library status passed to the function. The resulting string can be used by the User to determine the meaning of the status codes returned by the CLD library.

cld_lib_access_usb_phy_reg

```
CLD_RV cld_lib_access_usb_phy_reg (CLD_USB_PHY_Access_Params * p_params)
```

CLD Library function used to read or write the USB phy registers.

Arguments

p_params	Pointer to the CLD_USB_PHY_Access_Params structure describing the phy access.

Return Value

CLD_SUCCESS – USB phy access complete.
CLD_ONGOING – USB phy access in progress, continue calling cld_lib_access_usb_phy_reg until it returns CLD_SUCCESS or CLD_FAIL.
CLD_FAIL – Error occurred while accessing the phy.

Details

The cld_lib_access_usb_phy_reg function performs the USB phy access described by the p_params parameter.

The CLD_USB_PHY_Access_Params structure is described below.

```
typedef struct
{
    CLD_Boolean write;
    unsigned char reg_addr;
    unsigned char v_ctrl;
    unsigned char reg_data;
} CLD_USB_PHY_Access_Params;
```

A description of the CLD_USB_PHY_Access_Params structure elements is included below:

Structure Element	Description
write	TRUE = register write, FALSE = register read
reg_addr	Address of the USB phy register being accessed
v_ctrl	ULPI Vendor Control Register Address
reg_data	Data being written to, or read from, the USB phy register.

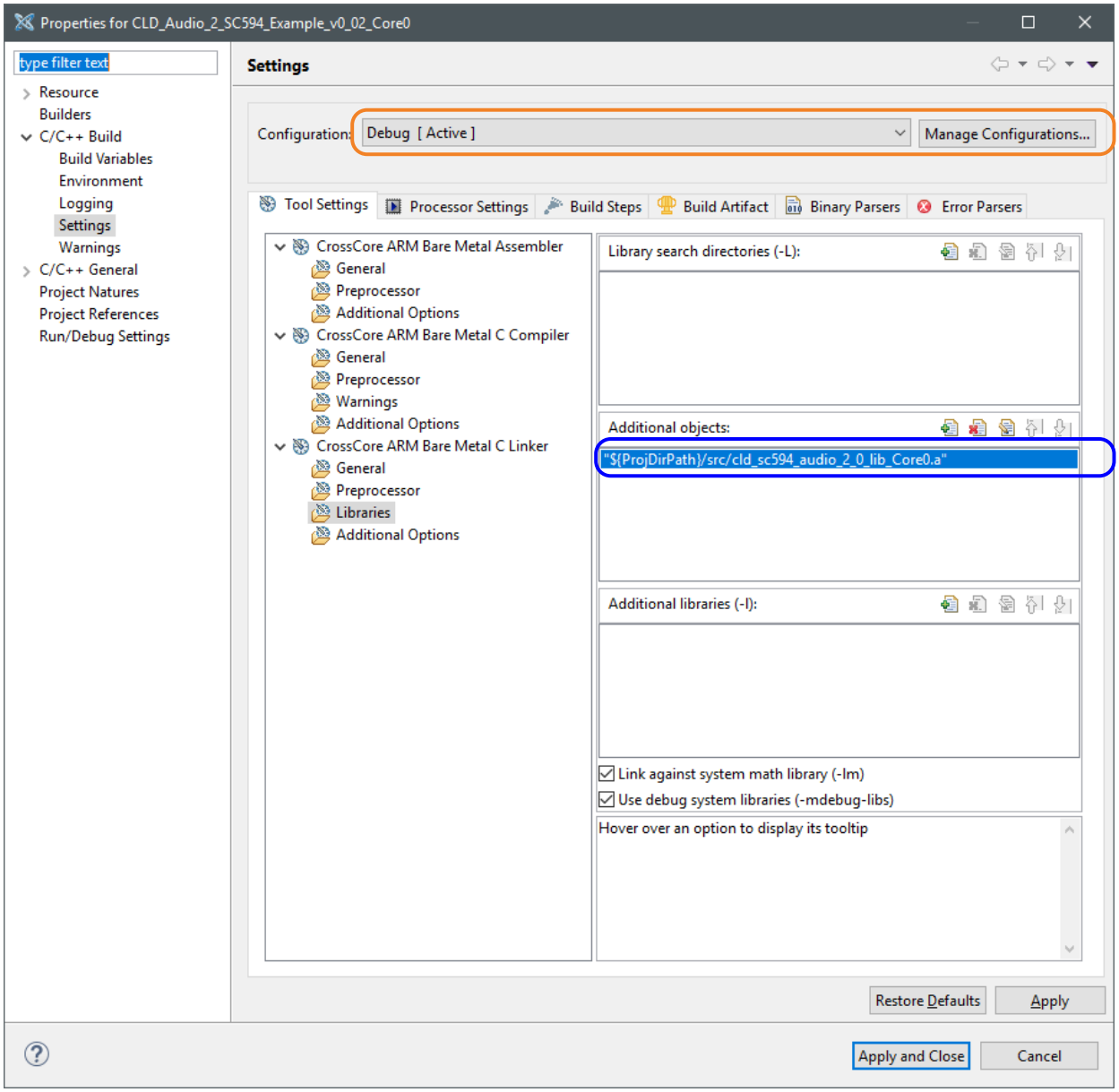
Adding the CLD SC594 Audio 2.0 Library to an Existing CrossCore Embedded Studio Project

In order to include the CLD SC594 Audio 2.0 Library in a CrossCore Embedded Studio (CCES) project you must configure the project linker settings so it can locate the library. The following steps outline how this is done.

1. Copy the cld_sc594_audio_2_0_lib.h and cld_sc594_audio_2_0_lib_Core0.a files to the project's src directory.
2. Open the project in CrossCore Embedded Studio.
3. Right click the project in the 'C/C++ Projects' window and select Properties.

   If you cannot find the 'C/C++ Projects" window, make sure C/C++ Perspective is active. If the C/C++ Perspective is active and you still cannot locate the 'C/C++ Projects' window select Window → Show View → C/C++ Projects.
4. You should now see a project properties window similar to the one shown below.

   Navigate to the C/C++ Build → Settings page and select the CrossCore ARM Bare Metal C Linker's Libraries page. The CLD SC594 Audio 2.0 Library needs to be included in the projects 'Additional objects' as shown in the diagram below (circled in blue). This lets the linker know where the cld_sc594_audio_2_0_lib_Core0.a file is located.



5. The 'Additional objects' setting needs to be set for all configurations (Debug, Release, etc). This can be done individually for each configuration, or all at once by selecting the [All Configurations] option as shown in the previous figure (circled in orange).

User Firmware Code Snippets

The following code snippets are not complete, and are meant to be a starting point for the User firmware. For a functional User firmware example that uses the CLD SC594 Audio 2.0 Library please refer to the CLD example projects included available with the CLD SC594 Audio 2.0 Library.

main.c

```
void main(void)
{
    Main_States main_state = MAIN_STATE_SYSTEM_INIT;

    while (1)
    {
        switch (main_state)
        {
            case MAIN_STATE_SYSTEM_INIT:
                /* Initialize the clock, and power systems.*/

                main_state = MAIN_STATE_USER_INIT;
            break;
            case MAIN_STATE_USER_INIT:
                rv = user_init();
                if (rv == USER_INIT_SUCCESS)
                {
                    main_state = MAIN_STATE_RUN;
                }
                else if (rv == USER_INIT_FAILED)
                {
                    main_state = MAIN_STATE_ERROR;
                }
            break;

            case MAIN_STATE_RUN:
                 user_main();
            break;

            case MAIN_STATE_ERROR:

            break;
        }
    }
}
```

user.c

```
#pragma pack (1)
/*
   USB Audio v2.0 Unit and Terminal descriptors that describe a simple
   audio device comprised of the following:

    Input Terminal - USB Streaming Endpoint
        ID = 0x01
        Channels: Left, Right
    Input Terminal - Microphone
        ID = 0x02
        Channels: Left, Right
    Output Terminal - Speaker
        ID = 0x06
        Source ID = 0x09
    Output Terminal - USB Streaming Endpoint
        ID = 0x07
        Source ID = 0x0a
    Feature Unit
        ID = 0x09
        Source ID = 0x01
        Controls:
            Master Channel 0: Mute (Control 1)
            Channel 1 (Left): Volume (Control 2)
            Channel 2 (Right): Volume (Control 2)
    Feature Unit
        ID = 0x0a
        Source ID = 0x02
        Controls:
            Master Channel 0: Volume (Control 2)
 */
/* USB Audio v2.0 Unit and Terminal descriptors that describe a simple audio device.*/
static const unsigned char user_audio_unit_and_terminal_descriptor[] =
{
    /* Input Terminal Descriptor - USB Endpoint */
    0x11,                   /* bLength */
    0x24,                   /* bDescriptorType = Class Specific Interface */
    0x02,                   /* bDescriptorSubType = Input Terminal */
    0x01,                   /* bTerminalID */
    0x01, 0x01,             /* wTerminalType = USB Streaming */
    0x00,                   /* bAssocTerminal */
    0x03,                   /* bCSourceID */
    0x02,                   /* bNRChannels */
    0x03, 0x00, 0x00,0x00,  /* wChannelConfig (Left & Right Present) */
    0x00,                   /* iChannelNames */
    0x00,0x00,              /* bmControls */
    0x00,                   /* iTerminal */
    /* Input Terminal Descriptor - Microphone */
    0x11,                   /* bLength */
    0x24,                   /* bDescriptorType = Class Specific Interface */
    0x02,                   /* bDescriptorSubType = Input Terminal */
    0x02,                   /* bTerminalID */
    0x01, 0x02,             /* wTerminalType = Microphone */
    0x00,                   /* bAssocTerminal */
    0x03,                   /* bCSourceID */
    0x02,                   /* bNRChannels */
    0x03, 0x00, 0x00,0x00,  /* wChannelConfig (Left & Right Present) */
    0x00,                   /* iChannelNames */
    0x00,0x00,              /* bmControls */
    0x00,                   /* iTerminal */
    /* Output Terminal Descriptor - Speaker */
    0x0c,                   /* bLength */
```

```
    0x24,                   /* bDescriptorType = Class Specific Interface */
    0x03,                   /* bDescriptorSubType = Output Terminal */
    0x06,                   /* bTerminalID */
    0x01, 0x03,             /* wTerminalType - Speaker */
    0x00,                   /* bAssocTerminal */
    0x09,                   /* bSourceID */
    0x03,                   /* bCSourceID */
    0x00, 0x00,             /* bmControls */
    0x00,                   /* iTerminal */
    /* Output Terminal Descriptor - USB Endpoint */
    0x0c,                   /* bLength */
    0x24,                   /* bDescriptorType = Class Specific Interface */
    0x03,                   /* bDescriptorSubType = Output Terminal */
    0x07,                   /* bTerminalID */
    0x01, 0x01,             /* wTerminalType - USB Streaming */
    0x00,                   /* bAssocTerminal */
    0x0a,                   /* bSourceID */
    0x03,                   /* bCSourceID */
    0x00, 0x00,             /* bmControls */
    0x00,                   /* iTerminal */
    /* Feature Unit Descriptor */
    0x12,                   /* bLength */
    0x24,                   /* bDescriptorType = Class Specific Interface */
    0x06,                   /* bDescriptorSubType = Feature Unit */
    0x09,                   /* bUnitID */
    0x01,                   /* bSourceID */
    0x0f, 0x00, 0x00, 0x00, /* bmaControls - Master */
    0x0f, 0x00, 0x00, 0x00, /* bmaControls - Left */
    0x0f, 0x00, 0x00, 0x00, /* bmaControls - Right */
    0x00,                   /* iFeature */
    /* Feature Unit Descriptor */
    0x12,                   /* bLength */
    0x24,                   /* bDescriptorType = Class Specific Interface */
    0x06,                   /* bDescriptorSubType = Feature Unit */
    0x0A,                   /* bUnitID */
    0x02,                   /* bSourceID */
    0x0f, 0x00, 0x00, 0x00, /* bmaControls - Master */
    0x0f, 0x00, 0x00, 0x00, /* bmaControls - Left */
    0x0f, 0x00, 0x00, 0x00, /* bmaControls - Right */
    0x00,                   /* iFeature */
    /* Clock Source Descriptor */
    0x08,                   /* bLength */
    0x24,                   /* bDescriptorType = Class Specific Interface */
    0x0a,                   /* bDescriptorSubType = Clock Source */
    0x03,                   /* ClockID */
    0x01,                   /* bmAttributes - Internal Fixed Clock */
    0x00,                   /* bmControls */
    0x00,                   /* bAssocTerminal */
    0x00,                   /* iClockSource */
};

/* Isochronous IN endpoint PCM format descriptor */
static const unsigned char user_audio_in_stream_format_descriptor[] =
{
    0x06,               /* bLength */
    0x24,               /* bDescriptorType - Class Specific Interface */
    0x02,               /* bDescriptorSubType - Format Type */
    0x01,               /* bFormatType - Format Type 1 */
    0x04,               /* bSubSlotSize */
    0x20,               /* bBitResolution */
};
```

```
/* Isochronous OUT endpoint PCM format descriptor */
static const unsigned char user_audio_out_stream_format_descriptor[] =
{
    0x06,               /* bLength */
    0x24,               /* bDescriptorType - Class Specific Interface */
    0x02,               /* bDescriptorSubType - Format Type */
    0x01,               /* bFormatType - Format Type 1 */
    0x04,               /* bSubSlotSize */
    0x20,               /* bBitResolution */
};


#pragma pack ()

/* IN Audio Stream Interface Endpoint Data Descriptor */
static const CLD_Audio_2_0_Lib_Audio_Stream_Data_Endpoint_Descriptor
user_audio_in_stream_endpoint_desc =
{
    .b_length = sizeof(CLD_Audio_2_0_Lib_Audio_Stream_Data_Endpoint_Descriptor),
    .b_descriptor_type                  = 0x25,   /* Class Specific Endpoint */
    .b_descriptor_subtype               = 0x01,   /* Endpoint - General */
    .bm_attributes                      = 0x00,   /* max packet only set to 0 */
    .bm_controls                        = 0x00,
    .b_lock_delay_units                 = 0x00,
    .w_lock_delay                       = 0x00,
};

/* OUT Audio Stream Interface Endpoint Data Descriptor */
static const CLD_Audio_2_0_Lib_Audio_Stream_Data_Endpoint_Descriptor
user_audio_out_stream_endpoint_desc =
{
    .b_length = sizeof(CLD_Audio_2_0_Lib_Audio_Stream_Data_Endpoint_Descriptor),
    .b_descriptor_type                  = 0x25,   /* Class Specific Endpoint */
    .b_descriptor_subtype               = 0x01,   /* Endpoint - General */
    .bm_attributes                      = 0x00,   /* max packet only set to 0 */
    .bm_controls                        = 0x00,
    .b_lock_delay_units                 = 0x02,   /* Milliseconds */
    .w_lock_delay                       = 0x01,   /* 1 Millisecond */
};

/* Audio Stream IN Interface parameters */
static CLD_Audio_2_0_Stream_Interface_Params user_audio_in_endpoint_params =
{
    .endpoint_number            = 2,              /* Isochronous endpoint number */
                                /* Isochronous endpoint full-speed max packet size */
    .max_packet_size_full_speed = USER_AUDIO_MAX_PACKET_SIZE,
                                /* Isochronous endpoint high-speed max packet size */
    .max_packet_size_high_speed = USER_AUDIO_MAX_PACKET_SIZE,
    .b_interval_full_speed      = 1,  /* Isochronous endpoint full-speed bInterval */
                    /* Isochronous endpoint high-speed bInterval - 1 millisecond */
    .b_interval_high_speed      = 4,
                                /* Terminal ID of the associated Output Terminal */
    .b_terminal_link            = 7,
    .b_format_type              = 1,           /* Type 1 Format */
    .bm_formats                 = 0x00000001,  /* Type 1 - PCM format */
    .b_nr_channels              = 2,           /* 2 Channels */
    .bm_channel_config          = 0x00000003,  /* Front Left & Front Right Channels */
    .p_encoder_descriptor       = CLD_NULL,
    .p_decoder_descriptor       = CLD_NULL,
    .p_format_descriptor        = (unsigned
char*)user_audio_in_stream_format_descriptor,
    .p_audio_stream_endpoint_data_descriptor =
(CLD_Audio_2_0_Lib_Audio_Stream_Data_Endpoint_Descriptor*)&user_audio_in_stream_endpoi
```

```
nt_desc,
};

/* Audio Stream OUT Interface parameters */
static CLD_Audio_2_0_Stream_Interface_Params user_audio_out_endpoint_params =
{
    .endpoint_number            = 2,            /* Isochronous endpoint number */
                            /* Isochronous endpoint full-speed max packet size */
    .max_packet_size_full_speed = USER_AUDIO_MAX_PACKET_SIZE,
                            /* Isochronous endpoint high-speed max packet size */
    .max_packet_size_high_speed = USER_AUDIO_MAX_PACKET_SIZE,
                            /* Isochronous endpoint full-speed bInterval */
    .b_interval_full_speed      = 1,
                    /* Isochronous endpoint high-speed bInterval - 1 millisecond */
    .b_interval_high_speed      = 4,
                            /* Terminal ID of the associated Output Terminal */
    .b_terminal_link            = 1,
    .b_format_type              = 1,          /* Type 1 Format */
    .bm_formats                 = 0x00000001, /* Type 1 - PCM format */
    .b_nr_channels              = 2,          /* 2 Channels */
    .bm_channel_config          = 0x00000003, /* Front Left & Front Right Channels */
    .p_encoder_descriptor       = CLD_NULL,
    .p_decoder_descriptor       = CLD_NULL,
    .p_format_descriptor        = (unsigned char*)
                                  user_audio_out_stream_format_descriptor,
    .p_audio_stream_endpoint_data_descriptor =
             (CLD_Audio_2_0_Lib_Audio_Stream_Data_Endpoint_Descriptor*)
      &user_audio_out_stream_endpoint_desc,
};

/* Audio Control Interrupt IN endpoint parameters */
static CLD_Audio_2_0_Control_Interrupt_Params user_audio_interrupt_in_params =
{
    .endpoint_number            = 1, /* Endpoint number */
    .b_interval_full_speed      = 1, /* Interrupt IN endpoint full-speed bInterval */
    .b_interval_high_speed      = 4, /* Interrupt IN endpoint high-speed bInterval */
};


/*!< CLD Library initialization data. */
static CLD_SC594_Audio_2_0_Lib_Init_Params user_audio_init_params =
{
    .vendor_id = 0x064b,        /* Analog Devices Vendor ID */
    .product_id = 0x0008,       /* Product ID. */
    .usb_bus_max_power = 0,

    .device_descriptor_bcdDevice = 0x0100,
    .audio_control_category_code = 0x01,  /* Desktop Speaker */

    .phy_hs_timeout_calibration     = 0, /* TODO: set based on USB Phy. */
    .phy_fs_timeout_calibration     = 0, /* TODO: set based on USB Phy. */
    .phy_delay_req_after_ulip_chirp_cmd = CLD_TRUE, /* TODO: set based on USB Phy. */

    .fp_init_usb_phy                = user_init_usb_phy,

      /* Optional Interrupt endpoint parameters */
    .p_audio_control_interrupt_params = &user_audio_interrupt_in_params,

    /* Unit and Terminal descriptor */
    .p_unit_and_terminal_descriptors = (unsigned char*)
             user_audio_unit_and_terminal_descriptor,
    .unit_and_terminal_descriptors_length =
             sizeof(user_audio_unit_and_terminal_descriptor),
```

```
    /* Pointer to the Interface parameters for the Audio Stream Rx interface. */
    .p_audio_streaming_rx_interface_params = &user_audio_out_endpoint_params,

    /* Pointer to the feedback parameters for the Audio Stream Rx interface. */
    .p_audio_rate_feedback_rx_params       = &user_audio_rate_feedback_params,

    /* Pointer to the Interface parameters for the Audio Stream Tx interface. */
    .p_audio_streaming_tx_interface_params = &user_audio_in_endpoint_params,

    /* Function called when an USB Audio 2.0 Set Request is received.*/
    .fp_audio_set_req_cmd = user_audio_set_req_cmd,

    /* Function called when an USB Audio 2.0 Get Request is received. */
    .fp_audio_get_req_cmd = user_audio_get_req_cmd,

    /* Function called when the Isochronous OUT interface is enabled/disabled */
    .fp_audio_streaming_rx_endpoint_enabled =
             user_audio_streaming_rx_endpoint_enabled,
    /* Function called when the Isochronous IN interface is enabled/disabled */
    .fp_audio_streaming_tx_endpoint_enabled =
             user_audio_streaming_tx_endpoint_enabled,

    /* USB string descriptors - Set to CLD_NULL if not required */
    .p_usb_string_manufacturer  = "Analog Devices Inc",
    .p_usb_string_product       = "SC594 Audio v2.0 Device",
    .p_usb_string_serial_number = CLD_NULL,
    .p_usb_string_configuration = CLD_NULL,
    .p_usb_string_audio_control_interface        = CLD_NULL,
    .p_usb_string_audio_streaming_out_interface  = "USB Audio Output",
    .p_usb_string_audio_streaming_in_interface   = "USB Audio Input",

    .user_string_descriptor_table_num_entries = 0,
    .p_user_string_descriptor_table = CLD_NULL,

    .usb_string_language_id     = 0x0409,              /* English (US) language ID */

    /* Function called when a USB events occurs on USB0. */
    .fp_cld_usb_event_callback = user_usb_event,

    /* Function called when the CLD library reports a status. */
    .fp_cld_lib_status         = user_cld_lib_status,
};
```

```
User_Init_Return_Code user_init (void)
{
    static unsigned char user_init_state = 0;
    CLD_RV cld_rv = CLD_ONGOING;
    User_Init_Return_Code init_return_code = USER_INIT_ONGOING;

    switch (user_init_state)
    {
        case 0:

            /* TODO: add any custom User firmware initialization */

            user_init_state++;
        break;
        case 1:
            /* Initialize the CLD Library */
            cld_rv = cld_sc594_audio_2_0_lib_init(&user_audio_init_params);

            if (cld_rv == CLD_SUCCESS)
            {
                /* Connect to the USB Host */
                cld_lib_usb_connect();

                init_return_code = USER_INIT_SUCCESS;
            }
            else if (cld_rv == CLD_FAIL)
            {
                init_return_code = USER_INIT_FAILED;
            }
            else
            {
                init_return_code = USER_INIT_ONGOING;
            }
    }
    return init_return_code;
}

void user_main (void)
{
    cld_sc594_audio_2_0_lib_main();

}

static CLD_RV user_init_usb_phy (void)
{
    /* TODO: Reset and configure the USB Phy. */
}

static void user_usb_event (CLD_USB_Event event)
{
    switch (event)
    {
        case CLD_USB_CABLE_CONNECTED:
            /* TODO: Add any User firmware processed when a USB cable is connected. */
        break;
        case CLD_USB_CABLE_DISCONNECTED:
            /* TODO: Add any User firmware processed when a USB cable is
               disconnected.*/
        break;
        case CLD_USB_ENUMERATED_CONFIGURED:
            /* TODO: Add any User firmware processed when a Device has been
```

```
                enumerated.*/
            break;
            case CLD_USB_UN_CONFIGURED:
                /* TODO: Add any User firmware processed when a Device USB Configuration
                   is set to 0.*/
            break;
            case CLD_USB_BUS_RESET:
                /* TODO: Add any User firmware processed when a USB Bus Reset occurs. */
            break;
        }
}

/* The following function will transmit the specified memory using
   the Isochronous IN endpoint. */
static user_audio_transmit_isochronous_in_data (void)
{
    static CLD_USB_Transfer_Params transfer_params;

    transfer_params.num_bytes = /* TODO: Set number of IN bytes */
    transfer_params.p_data_buffer = /* TODO: address data */
    transfer_params.callback.fp_usb_in_transfer_complete = /* TODO: Set to User
                                                              callback function or
                                                              CLD_NULL */;
    transfer_params.callback.fp_transfer_aborted_callback = /* TODO: Set to User
                                                             callback function or
                                                             CLD_NULL */;
    transfer_params.transfer_timeout_ms = /* TODO: Set to desired timeout */;

    if (cld_audio_2_0_lib_transmit_audio_data (&transfer_params) ==
            CLD_USB_TRANSMIT_SUCCESSFUL)
    {
        /* Isochronous IN transfer initiated successfully */
    }
    else /* Isochronous IN transfer was unsuccessful */
    {
    }
}

/* Function called when a Set Request is received */
static CLD_USB_Transfer_Request_Return_Type user_audio_set_req_cmd
             (CLD_Audio_2_0_Cmd_Req_Parameters * p_req_params,
              CLD_USB_Transfer_Params * p_transfer_data)
{
    p_transfer_data->p_data_buffer = /* TODO: address to store data */
    p_transfer_data->callback.fp_usb_out_transfer_complete =
                                        user_audio_set_req_cmd_transfer_complete;
    p_transfer_data->fp_transfer_aborted_callback = /* TODO: Set to User callback
                                                       function or CLD_NULL */
     /* TODO: Return how the Control transfer should be handled (Accept, Pause,
              Discard, or Stall */
}

/* Function called when the Set Request data is received */
static CLD_USB_Data_Received_Return_Type user_audio_set_req_cmd_transfer_complete
       (void)
{
    /* TODO: Return if the received data is good (CLD_USB_DATA_GOOD) or bad
       (CLD_USB_DATA_BAD_STALL) */
}

/* Function called when a Get Request is received */
static CLD_USB_Transfer_Request_Return_Type user_audio_get_req_cmd
             (CLD_Audio_2_0_Cmd_Req_Parameters * p_req_params,
```

```
                CLD_USB_Transfer_Params * p_transfer_data)
{
    p_transfer_data->p_data_buffer = /* TODO: address to source data */
    p_transfer_data->callback.fp_usb_in_transfer_complete =
                                        user_audio_get_req_cmd_transfer_complete;
    p_transfer_data->fp_transfer_aborted_callback = /* TODO: Set to User callback
                                                       function or CLD_NULL */
     /* TODO: Return how the Control transfer should be handled (Accept, Pause,
            Discard, or Stall */
}

/* Function called when the Get Request data has been transmitted */
static void user_audio_get_req_cmd_transfer_complete (void)
{
    /* TODO: The Get Request data has been sent to the Host, add any
       User functionality. */
}


static void user_audio_streaming_rx_endpoint_enabled (CLD_Boolean enabled)
{
    if (enabled == CLD_TRUE)
    {
        /* TODO: Add Isochronous OUT endpoint enabled User functionality. */
    }
    else
    {
        /* TODO: Add Isochronous OUT endpoint disabled User functionality. */
    }
}


static void user_audio_streaming_tx_endpoint_enabled (CLD_Boolean enabled)
{
    if (enabled == CLD_TRUE)
    {
        /* TODO: Add Isochronous IN endpoint enabled User functionality. */
    }
    else
    {
        /* TODO: Add Isochronous IN endpoint disabled User functionality. */
    }
}


static void user_cld_lib_status (unsigned short status_code, void * p_additional_data,
                                 unsigned short additional_data_size)
{
    /* TODO: Process the library status if needed.  The status can also be decoded to
            a USB readable string using cld_lib_status_decode as shown below: */

    char * p_str = cld_lib_status_decode(status_code, p_additional_data,
                                         additional_data_size);
}
```