USB Audio Version 1.0
for Analog Devices ADSP-BF70x
User's Guide Revision 2.00

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 reproduce and distribute executable files created using this software for use on Analog Devices Blackfin family processors only. Nothing else gives you the right to use this software.

Introduction

The Closed Loop Design (CLD) Audio 1.0 library creates a simplified interface for developing a USB Audio v1.0 device using the Analog Devices ADSP-BF707 EZ-Board and Analog Devices Audio EI3 Extender Board. The CLD BF70x Audio 1.0 library also includes support for a serial console and 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 BF70x Audio 1.0 Library API section of this document.

USB Background

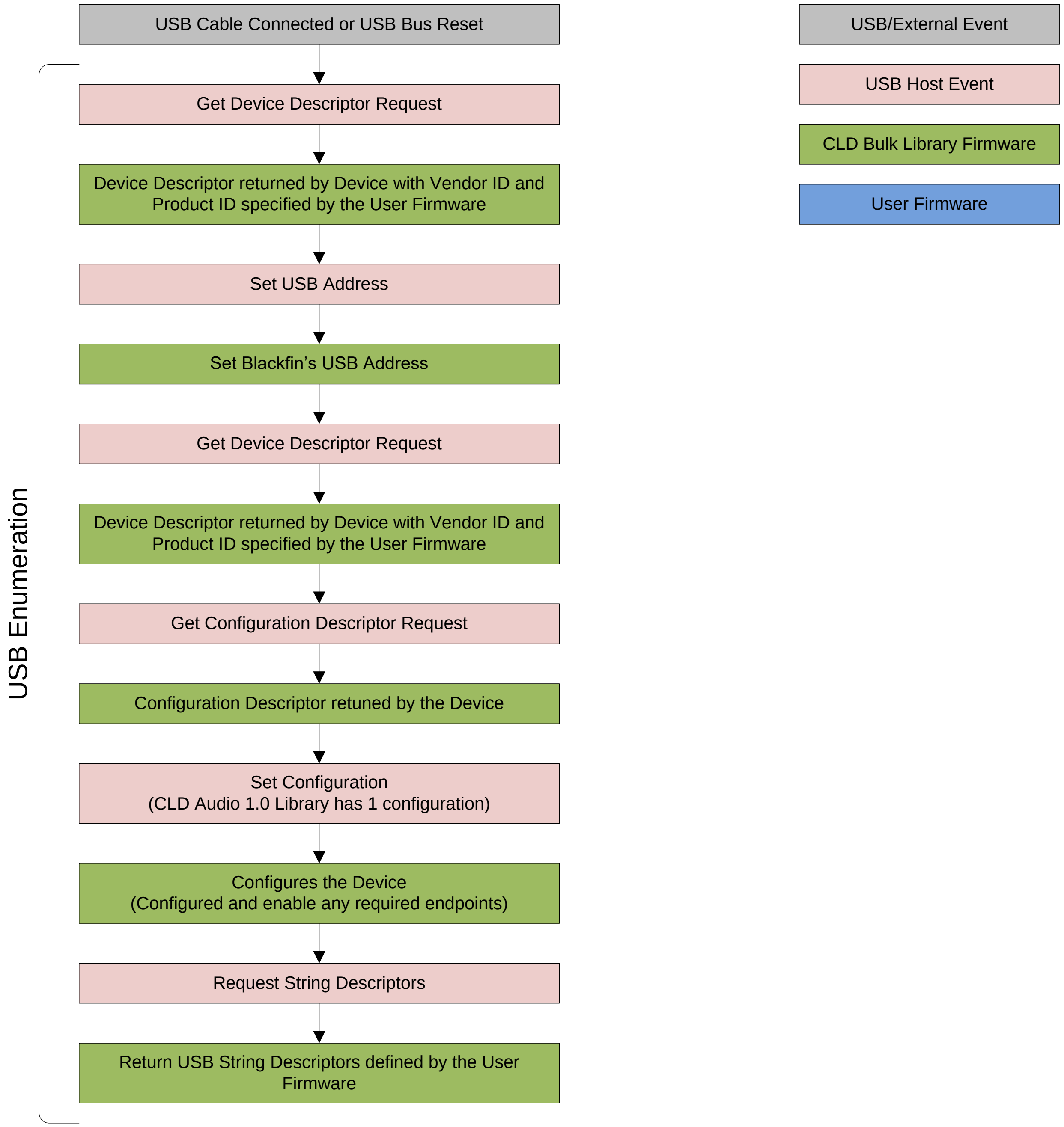
The following is a very basic overview of some of the USB concepts that are necessary to use the CLD BF70x Audio 1.0 Library. However, it is still recommended that developers have at least a basic understanding of the USB 2.0 protocol. The following are some resources to refer to when working with USB and USB Audio v1.0:

- The USB 2.0 Specification: http://www.usb.org/developers/docs/usb20_docs/
- The USB Device Class Definition for Audio Devices: http://www.usb.org/developers/docs/devclass_docs/audio10.pdf
- The USB Device Class Definition for Audio Data Formats: http://www.usb.org/developers/docs/devclass_docs/frmts10.pdf
- 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 BF70x Audio 1.0 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_bf70x_audio_1_0_lib_init section of this document. The CLD BF70x Audio 1.0 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 BF70x Audio 1.0 Library USB Enumeration Flow Chart


USB Cable Connected or USB Bus Reset
Get Device Descriptor Request
Device Descriptor returned by Device with Vendor ID and Product ID specified by the User Firmware
Set USB Address
Set Blackfin’s USB Address
Get Device Descriptor Request
Device Descriptor returned by Device with Vendor ID and Product ID specified by the User Firmware
Get Configuration Descriptor Request
Configuration Descriptor retuned by the Device
Set Configuration (CLD Audio 1.0 Library has 1 configuration)
Configures the Device (Configured and enable any required endpoints)
Request String Descriptors
Return USB String Descriptors defined by the User Firmware
USB Enumeration
USB/External Event
USB Host Event
CLD Bulk Library Firmware
User Firmware

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 BF70x Audio 1.0 Library uses Control 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 BF70x Audio 1.0 Library Endpoint 0 is also used to handle USB Audio Device Class v1.0 defined Set and Get requests. These requests are discussed in more detail in the USB Audio Device Class v1.0 Background section of this document

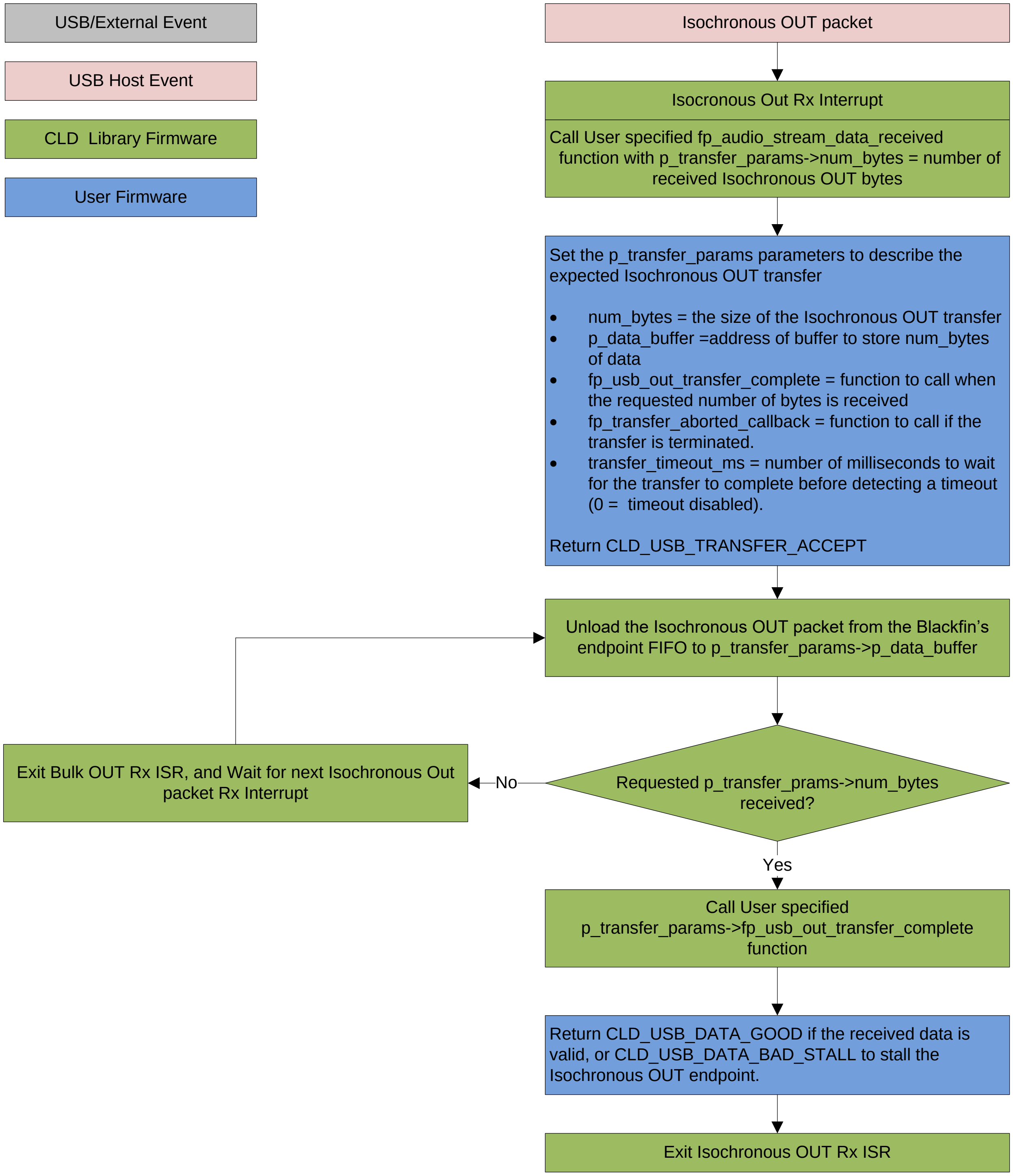
Isochronous Endpoints have the following characteristics which make them well suited for streaming audio data:

- 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.

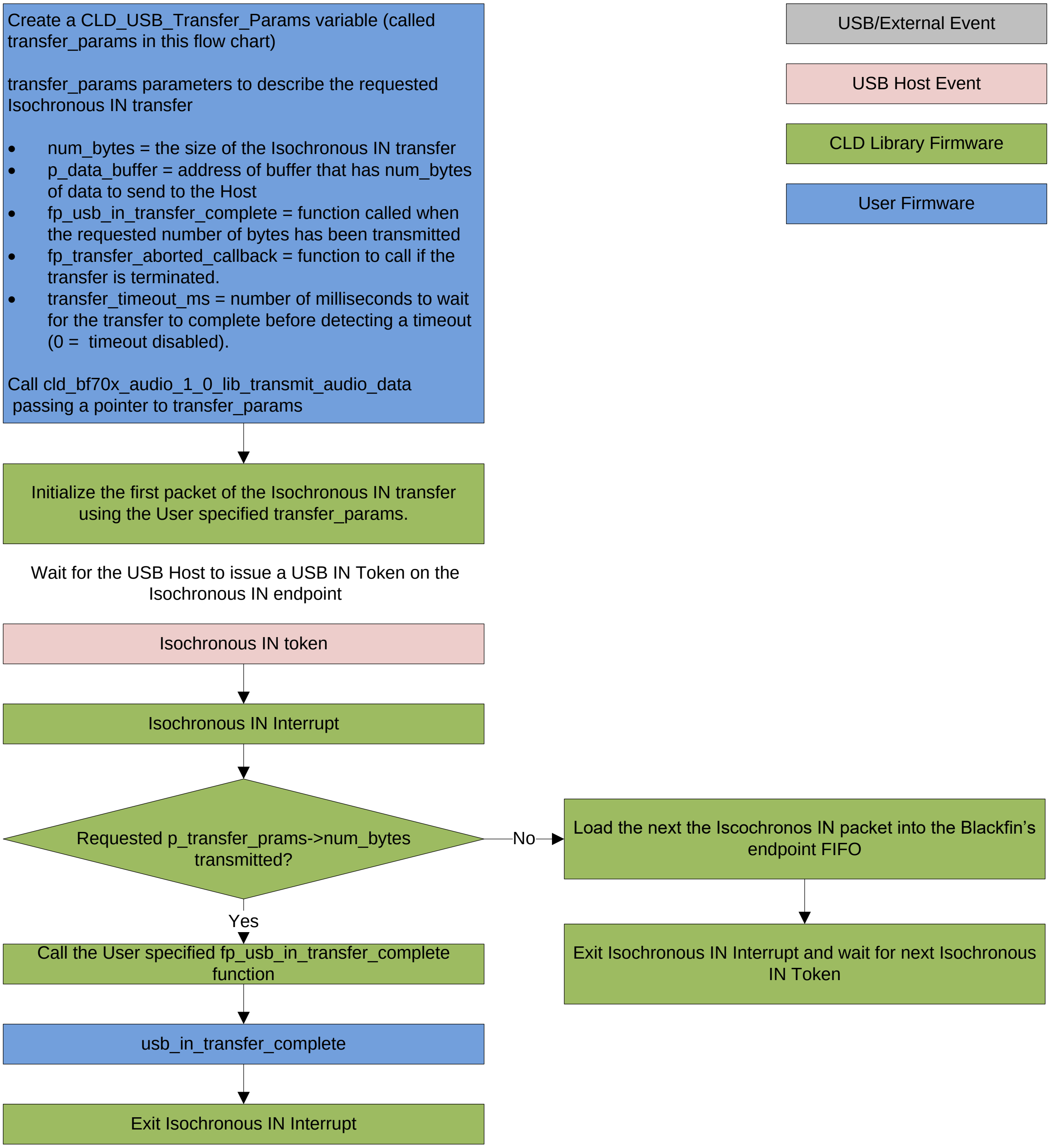
These characteristics allow for streaming audio data to be transmitted with deterministic timing. In the event of a USB transport error the audio data is dropped instead of being retried like a Bulk or Interrupt endpoint. This allows the streaming audio data to remain in sync. The CLD BF70x Audio 1.0 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 BF70x Audio 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 v1.0 device using the CLD BF70x Audio 1.0 Library.

CLD BF70x Audio 1.0 Library Isochronous OUT Flow Chart


USB/External Event
USB Host Event
CLD Library Firmware
User Firmware
Isochronous OUT packet
Isocronous Out Rx Interrupt
Call User specified fp_audio_stream_data_received function with p_transfer_params->num_bytes = number of received Isochronous OUT bytes
Set the p_transfer_params parameters to describe the expected Isochronous OUT transfer
• num_bytes = the size of the Isochronous OUT transfer
• p_data_buffer =address of buffer to store num_bytes of data
• fp_usb_out_transfer_complete = function to call when the requested number of bytes is received
• fp_transfer_aborted_callback = function to call if the transfer is terminated.
• transfer_timeout_ms = number of milliseconds to wait for the transfer to complete before detecting a timeout (0 = timeout disabled).
Return CLD_USB_TRANSFER_ACCEPT
Unload the Isochronous OUT packet from the Blackfin's endpoint FIFO to p_transfer_params->p_data_buffer
Requested p_transfer_prams->num_bytes received?
No
Exit Bulk OUT Rx ISR, and Wait for next Isochronous Out packet Rx Interrupt
Yes
Call User specified p_transfer_params->fp_usb_out_transfer_complete function
Return CLD_USB_DATA_GOOD if the received data is valid, or CLD_USB_DATA_BAD_STALL to stall the Isochronous OUT endpoint.
Exit Isochronous OUT Rx ISR

CLD BF70x Audio 1.0 Library Isochronous IN Flow Chart



USB/External Event

USB Host Event

CLD Library Firmware

User Firmware

Create a CLD_USB_Transfer_Params variable (called transfer_params in this flow chart)

transfer_params parameters to describe the requested Isochronous IN transfer

- num_bytes = the size of the Isochronous IN transfer
- p_data_buffer = address of buffer that has num_bytes of data to send to the Host
- fp_usb_in_transfer_complete = function called when the requested number of bytes has been transmitted
- fp_transfer_aborted_callback = function to call if the transfer is terminated.
- transfer_timeout_ms = number of milliseconds to wait for the transfer to complete before detecting a timeout (0 = timeout disabled).

Call cld_bf70x_audio_1_0_lib_transmit_audio_data passing a pointer to transfer_params

Initialize the first packet of the Isochronous IN transfer using the User specified transfer_params.

Wait for the USB Host to issue a USB IN Token on the Isochronous IN endpoint

Isochronous IN token

Isochronous IN Interrupt

Requested p_transfer_prams->num_bytes transmitted?

No

Load the next the Iscochronos IN packet into the Blackfin's endpoint FIFO

Exit Isochronous IN Interrupt and wait for next Isochronous IN Token

Yes

Call the User specified fp_usb_in_transfer_complete function

usb_in_transfer_complete

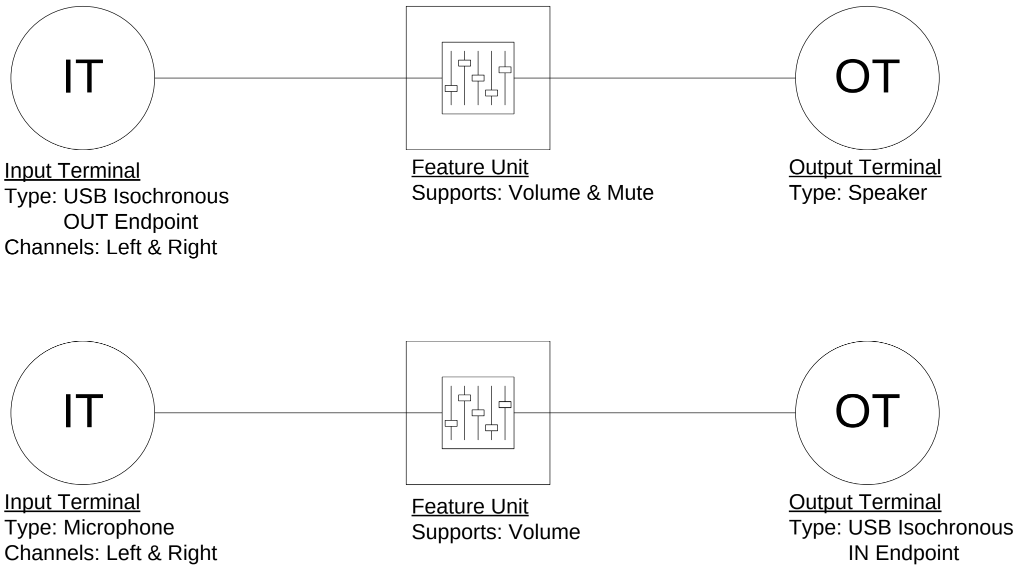
Exit Isochronous IN Interrupt

USB Audio Device Class v1.0 Background

The following is a basic overview of some USB Audio Device v1.0 concepts that are necessary to use the CLD BF70x Audio 1.0 Library. However, it is recommended that developers have at least a basic understanding of the USB Audio Device Class v1.0 protocol.

The USB Audio Device Class v1.0 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 v1.0 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 Audio 1.0 example project to implement a basic device with a stereo speaker output, and stereo microphone 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 v1.0 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 a audio stream format descriptor, which is part of the Streaming Audio Interface configuration. The USB Audio Device Class v1.0 specification supports multiple audio data formats which are described in the USB Device Class Definition for Audio Data Formats v1.0 specification. (www.usb.org/developers/docs/devclass_docs/frmts10.pdf)

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 v1.0 protocol requires that audio data streaming interfaces include two settings. The default setting does not have any Isochronous endpoints so its bandwidth requirement is zero. The alternate interface setting includes the required Isochronous endpoint. 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 BF70x Audio 1.0 Library receives a Set Interface request a 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_bf70x_audio_1_0_lib_init section of this document for more information.

USB Audio Device Class v1.0 Control Endpoint Requests

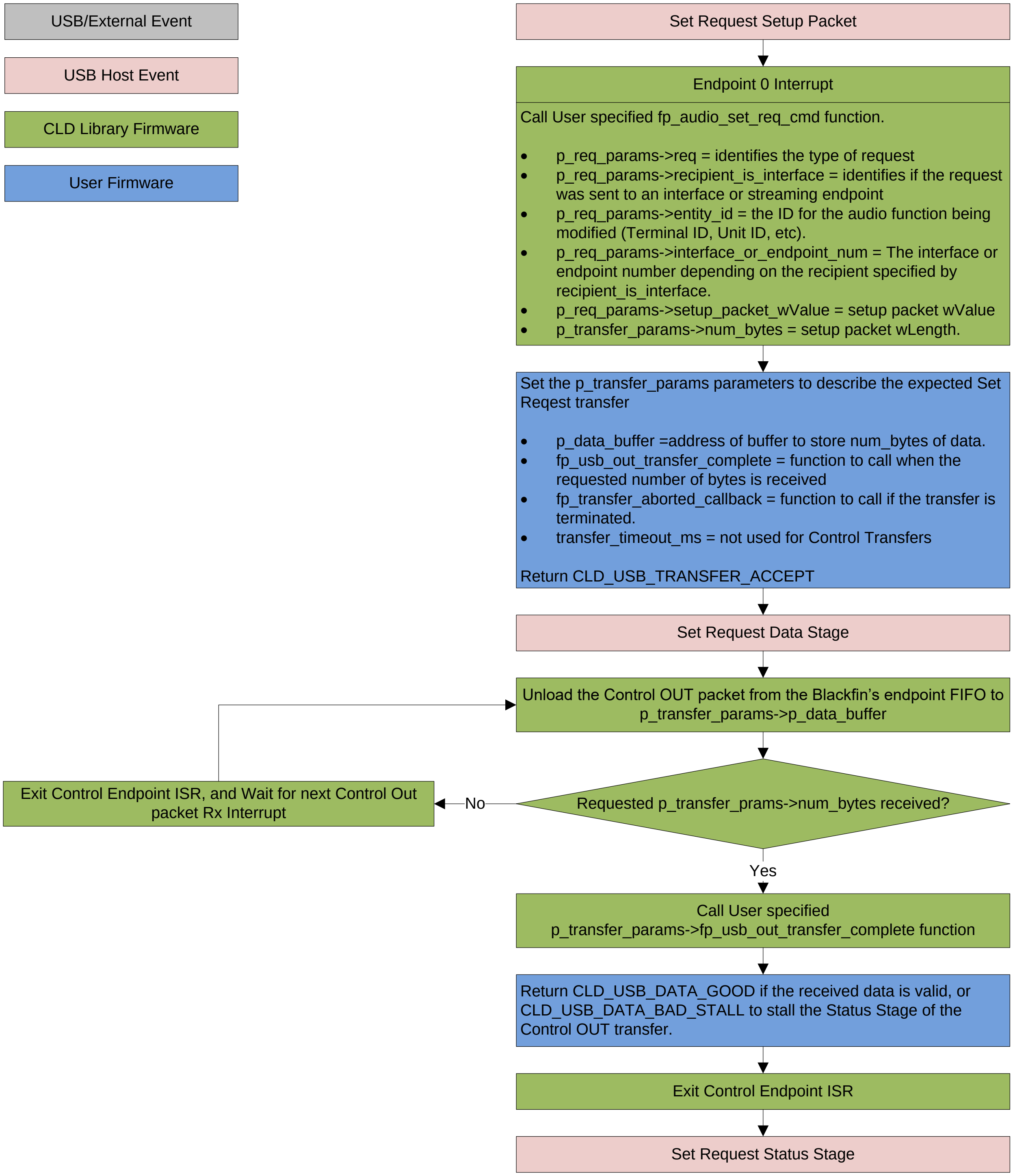
The USB Audio Device Class v1.0 control endpoint requests are broken down into Set and Get requests. These requests are used to control the various Terminal and Unit entities defined in the Configuration Descriptor. The CLD BF70x Audio 1.0 Library support for these requests is explained in the following 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 BF70x Audio 1.0 Library.

USB Audio Device Class v1.0 Set Request

The USB Audio Device Class v1.0 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.

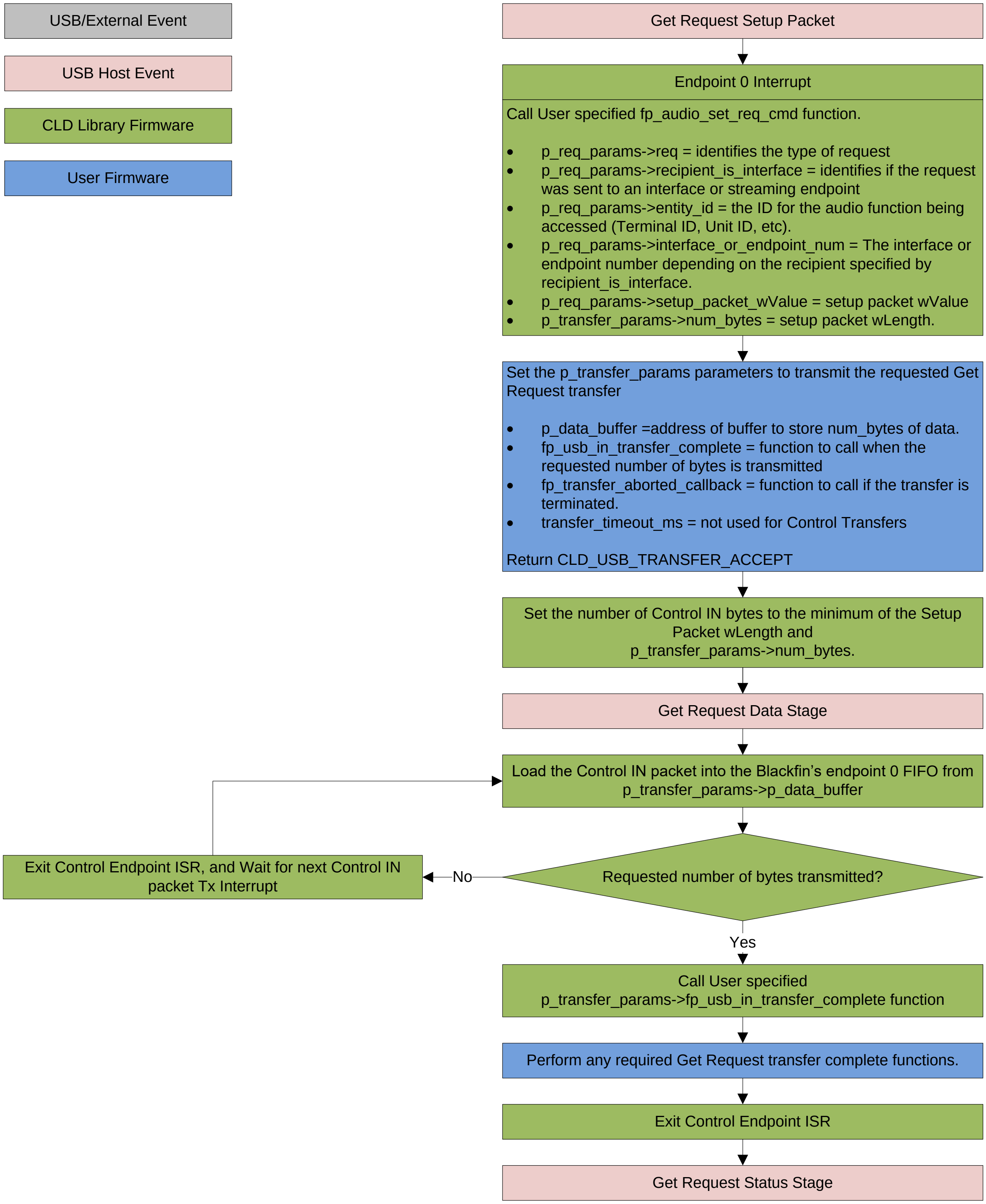
CLD BF70x Audio Device Class v1.0 Set Request Flow Chart


USB/External Event
USB Host Event
CLD Library Firmware
User Firmware
Set Request Setup Packet
Endpoint 0 Interrupt
Call User specified fp_audio_set_req_cmd function.
• p_req_params->req = identifies the type of request
• p_req_params->recipient_is_interface = identifies if the request was sent to an interface or streaming endpoint
• p_req_params->entity_id = the ID for the audio function being modified (Terminal ID, Unit ID, etc).
• p_req_params->interface_or_endpoint_num = The interface or endpoint number depending on the recipient specified by recipient_is_interface.
• p_req_params->setup_packet_wValue = setup packet wValue
• p_transfer_params->num_bytes = setup packet wLength.
Set the p_transfer_params parameters to describe the expected Set Reqest transfer
• p_data_buffer =address of buffer to store num_bytes of data.
• fp_usb_out_transfer_complete = function to call when the requested number of bytes is received
• fp_transfer_aborted_callback = function to call if the transfer is terminated.
• transfer_timeout_ms = not used for Control Transfers
Return CLD_USB_TRANSFER_ACCEPT
Set Request Data Stage
Unload the Control OUT packet from the Blackfin's endpoint FIFO to p_transfer_params->p_data_buffer
Requested p_transfer_prams->num_bytes received?
No
Exit Control Endpoint ISR, and Wait for next Control Out packet Rx Interrupt
Yes
Call User specified p_transfer_params->fp_usb_out_transfer_complete function
Return CLD_USB_DATA_GOOD if the received data is valid, or CLD_USB_DATA_BAD_STALL to stall the Status Stage of the Control OUT transfer.
Exit Control Endpoint ISR
Set Request Status Stage

USB Audio Device Class v1.0 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 BF70x Audio Device Class v1.0 Get Request Flow Chart



USB/External Event

USB Host Event

CLD Library Firmware

User Firmware

Get Request Setup Packet

Endpoint 0 Interrupt

Call User specified fp_audio_set_req_cmd function.

- p_req_params->req = identifies the type of request
- p_req_params->recipient_is_interface = identifies if the request was sent to an interface or streaming endpoint
- p_req_params->entity_id = the ID for the audio function being accessed (Terminal ID, Unit ID, etc).
- p_req_params->interface_or_endpoint_num = The interface or endpoint number depending on the recipient specified by recipient_is_interface.
- p_req_params->setup_packet_wValue = setup packet wValue
- p_transfer_params->num_bytes = setup packet wLength.

Set the p_transfer_params parameters to transmit the requested Get Request transfer

- p_data_buffer =address of buffer to store num_bytes of data.
- fp_usb_in_transfer_complete = function to call when the requested number of bytes is transmitted
- fp_transfer_aborted_callback = function to call if the transfer is terminated.
- transfer_timeout_ms = not used for Control Transfers

Return CLD_USB_TRANSFER_ACCEPT

Set the number of Control IN bytes to the minimum of the Setup Packet wLength and p_transfer_params->num_bytes.

Get Request Data Stage

Load the Control IN packet into the Blackfin’s endpoint 0 FIFO from p_transfer_params->p_data_buffer

Requested number of bytes transmitted?

Exit Control Endpoint ISR, and Wait for next Control IN packet Tx Interrupt

Call User specified p_transfer_params->fp_usb_in_transfer_complete function

Perform any required Get Request transfer complete functions.

Exit Control Endpoint ISR

Get Request Status Stage

Dependencies

In order to function properly, the CLD BF70x Audio 1.0 Library requires the following Blackfin resources:

- 24Mhz clock input connected to the Blackfin USB0_CLKIN pin.
- Optionally, the CLD BF70x Audio 1.0 Library can use one of the Blackfin UARTs to implement a serial console interface.
- The User firmware is responsible for setting up the Blackfin clocks, as well as enabling the Blackfin's System Event Controller (SEC) and configuring SEC Core Interface (SCI) interrupts to be sent to the Blackfin core.

Memory Footprint

The CLD BF70x Audio 1.0 Library approximate memory footprint is as follows:

Code memory:	29464 bytes
Data memory:	5364 bytes
Total:	34828 bytes or 34.01k
Heap memory:	1152 bytes (only malloc'ed if optional cld_console is enabled)

Note: The CLD BF70x Audio 1.0 Library is currently optimized for speed (not space).

CLD BF70x Audio 1.0 Library Scope and Intended Use

The CLD BF70x Audio 1.0 Library implements the USB Audio Device Class v1.0 required functionality to implement a USB Audio device, as well as providing time measurements and optional bi-directional UART console functionality. The CLD BF70x Audio 1.0 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, timer and UART console features. All other aspects of Blackfin processor configuration must be implemented by the User code.

CLD Audio 1.0 Example v2.0 Description

The CLD_Audio_1_0_Ex_v2_0 project provided with the CLD BF70x Audio 1.0 Library implements a basic USB audio device that supports a single stereo microphone input and stereo headphone output. This example is designed to run on the ADSP-BF707 Ez-Board coupled with the Analog Devices Audio EI3 Extender (http://www.analog.com/en/evaluation/eval-bfext-audei3/eb.html), and requires the Audio EI3 Extender board support package to be installed.

For additional information about connecting and using the Audio EI3 Extender please refer to the "Using the ADI Audio EI3 Extender" section of this Users Guide.

CLD BF70x Audio 1.0 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_bf70x_audio_1_0_lib_init

```
CLD_RV cld_bf70x_audio_1_0_lib_init (CLD_BF70x_Audio_1_0_Lib_Init_Params *
cld_audio_1_0_lib_params)
```

Initialize the CLD BF70x Audio 1.0 Library.

Arguments

`cld_audio_1_0_lib_params`	Pointer to a CLD_BF70x_Audio_1_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 BF70x Audio 1.0 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_bf70x_audio_1_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 cld_console UART if enabled by the User application. Once the library has been initialized successfully the main program loop can start.

The CLD_BF70x_Audio_1_0_Lib_Init_Params structure is described below:

```
typedef struct
{
    CLD_Uart_Num uart_num;
    unsigned long uart_baud;
    unsigned long sclk0;
    void (*fp_console_rx_byte) (unsigned char byte);

    unsigned short vendor_id;
    unsigned short product_id;
    unsigned char * p_unit_and_terminal_descriptors;
    unsigned short  unit_and_terminal_descriptors_length;

    CLD_BF70x_Audio_1_0_Stream_Interface_Params *
                  p_audio_streaming_rx_interface_params;

    CLD_BF70x_Audio_1_0_Stream_Interface_Params *
                  p_audio_streaming_tx_interface_params;

    CLD_USB_Transfer_Request_Return_Type (*fp_audio_stream_data_received)
                  (CLD_USB_Transfer_Params * p_transfer_data);

    CLD_USB_Transfer_Request_Return_Type (*fp_audio_set_req_cmd)
                  (CLD_BF70x_Audio_1_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_BF70x_Audio_1_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);

    unsigned char usb_bus_max_power
    unsigned short device_descriptor_bcdDevice

    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_audio_control_interface;
    const char * p_usb_string_audio_streaming_out_interface;
    const char * p_usb_string_audio_streaming_in_interface;

    unsigned char user_string_descriptor_table_num_entries;
    CLD_BF70x_Audio_1_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_BF70x_Audio_1_0_Lib_Init_Params;
```

A description of the CLD_BF70x_Audio_1_0_Lib_Init_Params structure elements is included below:

Structure Element	Description
uart_num	Identifies which of the ADSP-BF70x UARTs should be used by the CLD BF70x Audio 1.0 Library to implement the cld_console (refer to the cld_console API description for additional information). The valid uart_num values are listed below: CLD_UART_0 CLD_UART_1

<table>
<tr><td></td><td>CLD_UART_DISABLE

If uart_num is set to CLD_UART_ DISABLE the CLD BF70x Audio 1.0 Library will not use a UART, and the cld_console functionality is disabled.</td></tr>
<tr><td>uart_baud</td><td>Sets the desired UART baud rate used for the cld_console.
The remaining cld_console UART parameters are as follows:

Number of data bits: 8
Number of stop bits: 1
No Parity
No Hardware Flow Control</td></tr>
<tr><td>sclk0</td><td>Used to tell the CLD BF70x Audio 1.0 Library the frequency of the ADSP_BF70x SCLK0 clock.</td></tr>
<tr><td>fp_console_rx_byte</td><td>Pointer to the function that is called when a byte is received by the cld_console UART. This function has a single parameter ('byte') which is the value received by the UART.
<b>Note:</b> Set to NULL if not required by application</td></tr>
<tr><td>vendor_id</td><td>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).</td></tr>
<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>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_BF70x_Audio_1_0_Stream_Interface_Params structure that describes how the Isochronous IN endpoint and related USB Audio Streaming interface should be configured. The a CLD_BF70x_Audio_1_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 ≤ endpoint_num ≤ 12. Any other endpoint number will result in the cld_bf70x_audio_1_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</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>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>synchronization_type</td><td>Sets the Isochronous endpoint synchronization type.
1 = Asynchronous
2 = Adaptive
3 = Synchronous</td></tr>
<tr><td>b_terminal_link</td><td>The Terminal ID of the Terminal connected to this endpoint.</td></tr>
<tr><td>b_delay</td><td>Delay in frames introduced by this endpoint's data path.</td></tr>
<tr><td>w_format_tag</td><td>Identifies the audio data format use by this interface.</td></tr>
<tr><td>p_format_type_descriptor</td><td>Pointer to the format descriptor defined in the USB Device Class Definition for Audio Data Formats v1.0 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 v1.0 section 4.6.1.2).</td></tr>
</table>
</td></tr>
<tr><td>p_audio_streaming_tx_interface_params</td><td>Pointer to a CLD_BF70x_Audio_1_0_Stream_Interface_Params structure that describes how the Isochronous OUT endpoint and related USB Audio Streaming interface should be configured. Refer to the p_audio_streaming_rx_interface_params description for information about the CLD_BF70x_Audio_1_0_Stream_Interface_Params structure.</td></tr>
<tr><td><i>fp_audio_stream_data_received</i></td><td>Pointer to the function that is called when the Isochronous OUT endpoint receives data. This function takes a pointer to the CLD_USB_Transfer_Params structure ('p_transfer_data') as a parameter.

The following CLD_USB_Transfer_Params structure elements are used to processed a Isochronous OUT transfer:
<table>
<tr><th>Structure Element</th><th>Description</th></tr>
<tr><td>num_bytes</td><td>The number of bytes to transfer to p_data_buffer before calling the</td></tr>
</table>
</td></tr>
</table>

	fp_usb_out_transfer_complete callback function. When the fp_audio_stream_data_received function is called num_bytes is set the number of bytes in the current Isochronous OUT packet. If the Isochronous OUT total transfer size is known num_bytes can be set to the transfer size, and the CLD BF70x Audio 1.0 Library will complete the entire transfer before calling fp_audio_stream_data_received again. If num_bytes isn't modified the fp_audio_stream_data_received function will be called for each Isochronous OUT packet.
p_data_buffer	Pointer to the data buffer to store the received Isochronous OUT data. The size of the buffer should be greater than or equal to the value in num_bytes.
fp_usb_out_transfer_compelete	Function called when num_bytes of data has been transferred to the p_data_buffer memory.
fp_transfer_aborted_callback	Function called if there is a problem transferring the requested Isochronous OUT data.
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

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

Return Value	Description
CLD_USB_TRANSFER_ACCEPT	Notifies the CLD BF70x

<table>
<tr>
<td></td>
<td>
<table>
<tr><td></td><td>Audio 1.0 Library that the Isochronous OUT data should be accepted using the p_transfer_data values.</td></tr>
<tr><td>CLD_USB_TRANSFER_PAUSE</td><td>Requests that the CLD BF70x Audio 1.0 Library pause the current transfer. This causes the Isochronous OUT endpoint to be nak'ed until the transfer is resumed by calling cld_bf70x_audio_1_0_lib_resume_paused_audio_data_ transfer.</td></tr>
<tr><td>CLD_USB_TRANSFER_DISCARD</td><td>Requests that the CLD BF70x Audio 1.0 Library discard the number f bytes specified in p_transfer_params-> num_bytes. In this case the library accepts the Isochronous OUT data from the USB Host but discards the data</td></tr>
<tr><td>CLD_USB_TRANSFER_STALL</td><td>This notifies the CLD BF70x Audio 1.0 Library that there is an error and the Isochronous OUT endpoint should be stalled.</td></tr>
</table>
</td>
</tr>
<tr>
<td><i>fp_audio_set_req_cmd</i></td>
<td>
Pointer to the function that is called when a USB Audio Device Class v1.0 Set Request is received. This function has a pointer to the CLD_USB_Transfer_Params structure ('p_transfer_data') , and a pointer to the CLD_BF70x_Audio_1_0_Cmd_Req_Parameters (p_req_params) as its parameters.

The following CLD_BF70x_Audio_1_0_Cmd_Req_Parameters structure elements are used to processed a Set Request:
<table>
<tr><td>Structure Element</td><td>Description</td></tr>
<tr><td>req</td><td>Identifies the type of request. The valid types if requests are listed below:
CLD_SET_CURRENT
CLD_SET_MIN
CLD_SET_MAX
CLD_SET_RESOLUTION
CLD_SET_MEMORY</td></tr>
<tr><td>recipient_is_interface</td><td>Identifies if the request was sent to an interface or Audio streaming endpoint</td></tr>
<tr><td>entity_id</td><td>The ID for the audio function being modified (Terminal ID, Unit ID, etc)</td></tr>
<tr><td>interface_or_endpoint_num</td><td>The interface or endpoint number for the request</td></tr>
</table>
</td>
</tr>
</table>

	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 BF70x Audio 1.0 Library that the Set Request data should be accepted using the p_transfer_data values.
CLD_USB_TRANSFER_PAUSE	Requests that the CLD BF70x Audio 1.0 Library pause the Set Request transfer. This causes the Control Endpoint to be nak'ed until the transfer is

<table>
<tr>
<td></td>
<td>
<table>
<tr><td></td><td>resumed by calling cld_bf70x_audio_1_0_lib_resume_paused_control_transfer.</td></tr>
<tr><td>CLD_USB_TRANSFER_DISCARD</td><td>Requests that the CLD BF70x Audio 1.0 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.</td></tr>
<tr><td>CLD_USB_TRANSFER_STALL</td><td>This notifies the CLD BF70x Audio 1.0 Library that there is an error and the request should be stalled.</td></tr>
</table>
</td>
</tr>
<tr>
<td><i>fp_audio_get_req_cmd</i></td>
<td>
Pointer to the function that is called when a USB Audio Device Class v1.0 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_BF70x_Audio_1_0_Cmd_Req_Parameters (p_req_params) as its parameters.

The following CLD_BF70x_Audio_1_0_Cmd_Req_Parameters structure elements are used to processed a Set Request:
<table>
<tr><th>Structure Element</th><th>Description</th></tr>
<tr><td>req</td><td>Identifies the type of request. The valid types if requests are listed below:
CLD_GET_CURRENT
CLD_GET_MIN
CLD_GET_MAX
CLD_GET_RESOLUTION
CLD_GET_MEMORY
CLD_GET_STATUS</td></tr>
<tr><td>recipient_is_interface</td><td>Identifies if the request was sent to an interface or Audio streaming endpoint</td></tr>
<tr><td>entity_id</td><td>The ID for the audio function being accessed (Terminal ID, Unit ID, etc)</td></tr>
<tr><td>interface_or_endpoint_num</td><td>The interface or endpoint number for the request depending on the recipient specified by the recipient_is_interface parameter.</td></tr>
<tr><td>setup_packet_wValue</td><td>wValue field from the USB Setup Packet.</td></tr>
</table>

The following CLD_USB_Transfer_Params structure elements are used to processed a Set Request:
</td>
</tr>
</table>

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 BF70x Audio 1.0 Library that the Get Request data should be transmitted using the p_transfer_data values.
CLD_USB_TRANSFER_PAUSE	Requests that the CLD BF70x Audio 1.0 Library pause the Get Request transfer. This causes the Control Endpoint to be nak'ed until the transfer is resumed by calling cld_bf70x_audio_1_0_lib_resume_paused_control_transfer.
CLD_USB_TRANSFER_DISCARD	Requests that the CLD BF70x Audio 1.0 Library to return a zero length packet in response to the Get Request.
CLD_USB_TRANSFER_STALL	This notifies the CLD BF70x Audio 1.0 Library that there is

	an error and the request should be stalled.
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.
usb_bus_max_power	USB Configuration Descriptor bMaxPower value (0 = self powered). Refer to the USB 2.0 protocol section 9.6.3.
device_descriptor_bcd_device	USB Device Descriptor bcdDevice value. Refer to the USB 2.0 protocol section 9.6.1.
p_usb_string_manufacturer	Pointer to the null-terminated string. This string is used by the CLD BF70x Audio 1.0 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 BF70x Audio 1.0 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 BF70x Audio 1.0 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 BF70x Audio 1.0 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 BF70x Audio 1.0 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 BF70x Audio 1.0 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 BF70x Audio 1.0 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_BF70x_Audio_1_0_Lib_User_String_Descriptors structures addressed by p_user_string_descriptor_table. Set to 0 if p_user_string_descriptor_table is set to CLD_NULL.
p_user_string_descriptor_table	Pointer to an array of CLD_BF70x_Audio_1_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.

<table>
<tr><td></td><td>Set to CLD_NULL is not used.

The CLD_BF70x_Audio_1_0_Lib_User_String_Descriptors structure elements are explained below:
<table>
<tr><td>Structure Element</td><td>Description</td></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><td>Return Value</td><td>Description</td></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><i>CLD_USB_ENUMERATED_CONFIGURED_HS</i></td><td>USB device enumerated at High-Speed (USB Configuration set to a non-zero value)</td></tr>
<tr><td><i>CLD_USB_ENUMERATED_CONFIGURED_FS</i></td><td>USB device enumerated at Full-Speed (USB Configuration set to a non-zero value)</td></tr>
<tr><td><i>CLD_USB_UN_CONFIGURED</i></td><td>USB Configuration set to 0</td></tr>
<tr><td><i>CLD_USB_BUS_RESET</i></td><td>USB Bus reset received</td></tr>
<tr><td><i>CLD_USB_BUS_SUSPEND</i></td><td>USB Suspend detected</td></tr>
<tr><td><i>CLD_USB_BUS_RESUME</i></td><td>USB Resume detected</td></tr>
</table>
<b>Note:</b> Set to CLD_NULL if not required by application</td></tr>
<tr><td><i>fp_cld_lib_status</i></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><td>Parameter</td><td>Description</td></tr>
</table></td></tr>
</table>

status_code	16-bit status code. If the most significant bit is a '1' the status being reported is an Error.
p_additional_data	Pointer to additional data included with the status.
additional_data_size	The number of bytes in the specified additional data.

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.

cld_bf70x_audio_1_0_lib_main

```
void cld_bf70x_audio_1_0_lib_main (void)
```

CLD BF70x Audio 1.0 Library mainline function

Arguments

None

Return Value

None.

Details

The cld_bf70x_audio_1_0_lib_main function is the CLD BF70x Audio 1.0 Library mainline function that must be called in every iteration of the main program loop in order for the library to function properly.

cld_bf70x_audio_1_0_lib_transmit_audio_data

```
CLD_USB_Data_Transmit_Return_Type cld_bf70x_audio_1_0_lib_transmit_audio_data
       (CLD_USB_Transfer_Params * p_transfer_data)
```

CLD BF70x Audio 1.0 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

Details

The cld_bf70x_audio_1_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 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_bf70x_audio_1_0_lib_resume_paused_audio_data_transfer

```
void cld_bf70x_audio_1_0_lib_resume_paused_audio_data_transfer (void)
```

CLD BF70x Audio 1.0 Library function used to resume a paused Isochronous OUT transfer.

Arguments

None

Return Value

None.

Details

The cld_bf70x_audio_1_0_lib_resume_paused_audio_data_transfer function is used to resume an Isochronous OUT transfer that was paused by the `fp_audio_stream_data_received` function returning CLD_USB_TRANSFER_PAUSE. When called the cld_bf70x_audio_1_0_lib_resume_paused_audio_data_transfer function will call the User application's fp`_audio_stream_data_received` function passing the CLD_USB_Transfer_Params of the original paused transfer. The `fp_audio_stream_data_received` function can then choose to accept, discard, or stall the Isochronous OUT request.

cld_bf70x_audio_1_0_lib_resume_paused_control_transfer

```
void cld_bf70x_audio_1_0_lib_resume_paused_control_transfer (void)
```

CLD BF70x Audio 1.0 Library function used to resume a paused Control endpoint transfer.

Arguments

None

Return Value

None.

Details

The cld_bf70x_audio_1_0_lib_resume_paused_control_transfer function is used to resume a Control transfer that 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_bf70x_audio_1_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 BF70x Audio 1.0 Library function used to connect to the USB Host.

Arguments

None

Return Value

None.

Details

The cld_lib_usb_connect function is called after the CLD BF70x Audio 1.0 Library has been initialized to connect the USB device to the Host.

cld_ lib_usb_disconnect

```
void cld_lib_usb_disconnect (void)
```

CLD BF70x Audio 1.0 Library function used to disconnect from the USB Host.

Arguments

None

Return Value

None.

Details

The cld_lib_usb_disconnect function is called after the CLD BF70x Audio 1.0 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_usb_isr_callback

```
void cld_usb_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 Routine as shown in the CLD provided example projects.

cld_console_tx_isr_callback

```
void cld_console_tx_isr_callback (void)
```

CLD library console UART transmit interrupt service routines

Arguments

None

Return Value

None.

Details

These transmit ISR functions should be called from the corresponding UART transmit Interrupt Service Routine as shown in the CLD provided example projects.

cld_console_rx_isr_callback

```
void cld_console_rx_isr_callback (void)
```

CLD library console UART receive interrupt service routines

Arguments

None

Return Value

None.

Details

These receive ISR functions should be called from the corresponding UART receive Interrupt Service Routine as shown in the CLD provided example projects.

cld_time_get

```
CLD_Time cld_time_get(void)
```

CLD BF70x Audio 1.0 Library function used to get the current CLD time.

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.

cld_time_passed_ms

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

CLD BF70x Audio 1.0 Library function used to measure the amount of time that has passed.

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.

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.

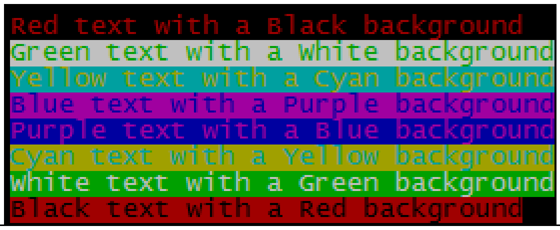
cld_console

```
CLD_RV cld_console(CLD_CONSOLE_COLOR foreground_color, CLD_CONSOLE_COLOR
      background_color, const char *fmt, ...)
```

CLD Library function that outputs a User defined message using the UART specified in the CLD_BF70x_Audio_1_0_Lib_Init_Params structure.

Arguments

`foreground_color`	The CLD_CONSOLE_COLOR used for the console text.
	`CLD_CONSOLE_BLACK`
	`CLD_CONSOLE_RED`
	`CLD_CONSOLE_GREEN`
	`CLD_CONSOLE_YELLOW`
	`CLD_CONSOLE_BLUE`
	`CLD_CONSOLE_PURPLE`
	`CLD_CONSOLE_CYAN`
	`CLD_CONSOLE_WHITE`

`background_color`	The CLD_CONSOLE_COLOR used for the console background.
	`CLD_CONSOLE_BLACK`
	`CLD_CONSOLE_RED`
	`CLD_CONSOLE_GREEN`
	`CLD_CONSOLE_YELLOW`
	`CLD_CONSOLE_BLUE`
	`CLD_CONSOLE_PURPLE`
	`CLD_CONSOLE_CYAN`
	`CLD_CONSOLE_WHITE`
	The foreground and background colors allow the User to generate various color combinations like the ones shown below:
	
	Red text with a Black background
	Green text with a White background
	Yellow text with a Cyan background
	Blue text with a Purple background
	Purple text with a Blue background
	Cyan text with a Yellow background
	White text with a Green background
	Black text with a Red background

`fmt`	The User defined ASCII message that uses the same format specifies as the printf function.
`...`	Optional list of additional arguments

Return Value

This function returns whether or not the specified message has been added to the cld_console transmit buffer.

`CLD_SUCCESS`	The message was added successfully.
`CLD_FAIL`	The message was not added, so the message will not be transmitted. This will occur if the CLD Console is disabled, or if the message will not fit into the transmit buffer.

Details

cld_console is similar in format to printf, and also natively supports setting a foreground and background color.
The following will output 'The quick brown fox' on a black background with green text:

```
cld_console(CLD_CONSOLE_GREEN, CLD_CONSOLE_BLACK, "The quick brown %s\n\r", "fox");
```

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

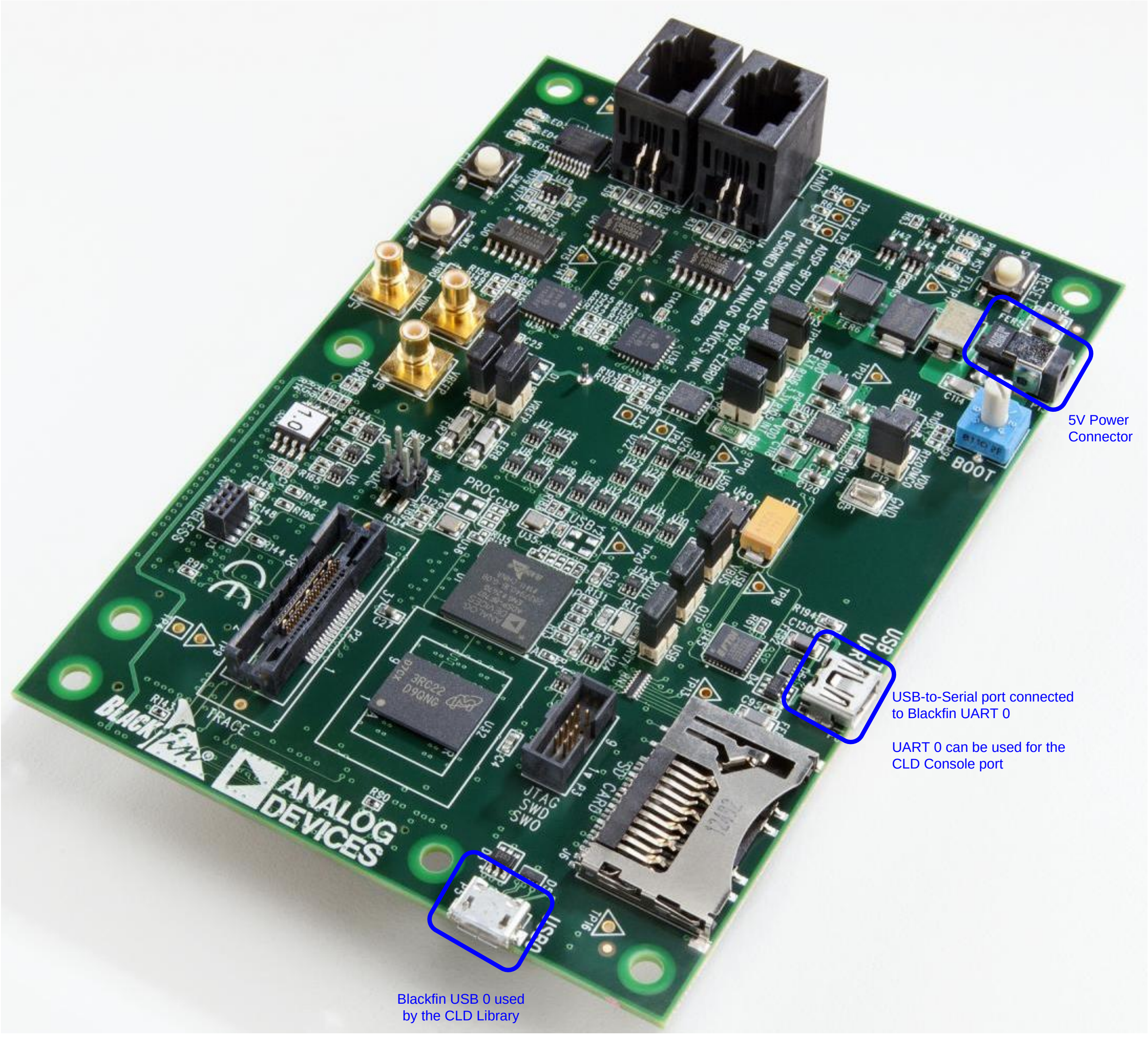
This function returns a decoded Null terminated ASCII string.

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.

Using the ADSP-BF707 Ez-Board

Connections:



Note about using UART0 and the FTDI USB to Serial Converter

On the ADSP-BF707 Ez-Board the Blackfin's UART0 serial port is connected to a FTDI FT232RQ USB-to-Serial converter. By default the UART 0 signals are connected to the FTDI chip. However, the demo program shipped on the Ez-Board disables the UART0 to FTDI connection. If the FTDI converter is used for the CLD BF70x Audio 1.0 Library console change the boot selection switch (located next to the power connector) so the demo program doesn't boot. Once this is done the FTDI USB-to-Serial converter can be used with the CLD BF70x Audio 1.0 Library console connected to UART0.

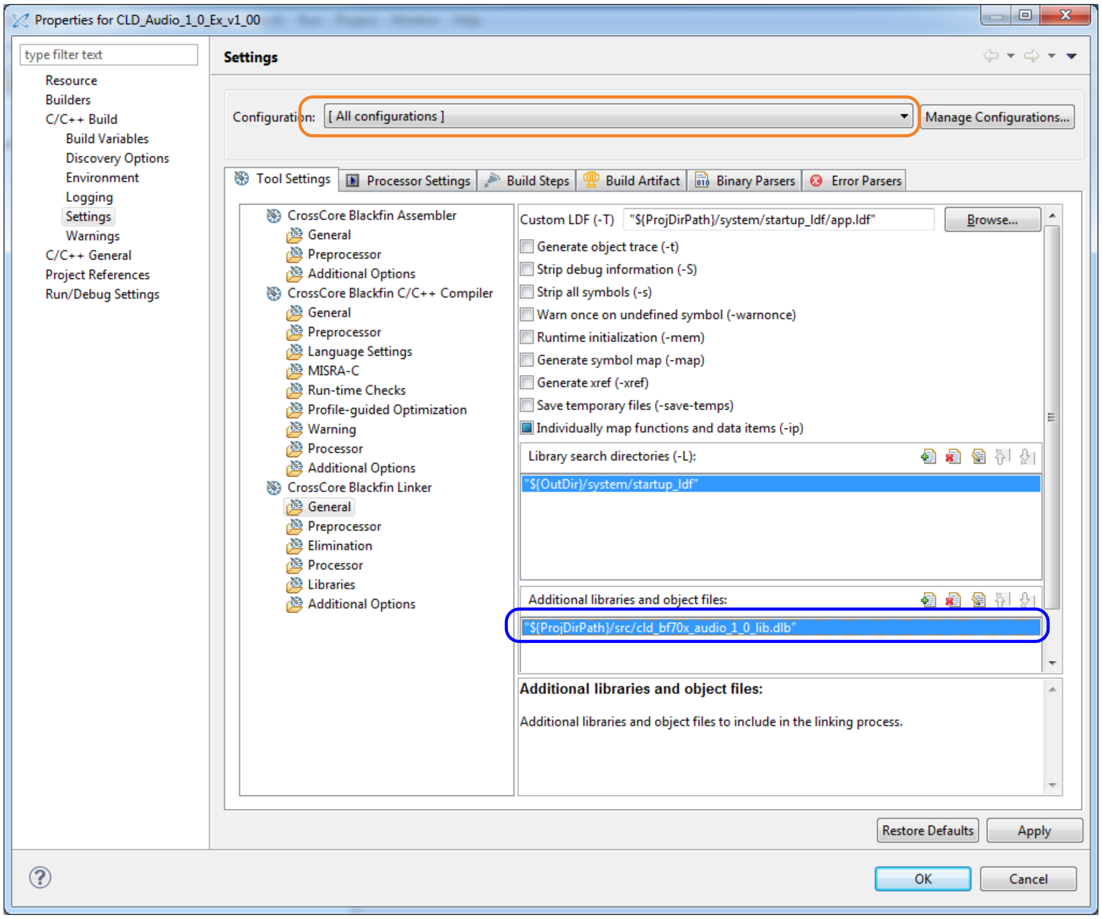
Adding the CLD BF70x Audio 1.0 Library to an Existing CrossCore Embedded Studio Project

In order to include the CLD BF70x Audio 1.0 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_bf70x_audio_1_0_lib.h and cld_bf70x_audio_1_0_lib.dlb 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 Blackfin Linker General page. The CLD BF70x Audio 1.0 Library needs to be included in the projects 'Additional libraries and object files' as shown in the diagram below (circled in blue). This lets the linker know where the cld_bf70x_audio_1_0_lib.dlb file is located.



Configuration: [All configurations]

Custom LDF (-T) "${ProjDirPath}/system/startup_ldf/app.ldf"

☐ Generate object trace (-t)
☐ Strip debug information (-S)
☐ Strip all symbols (-s)
☐ Warn once on undefined symbol (-warnonce)
☐ Runtime initialization (-mem)
☐ Generate symbol map (-map)
☐ Generate xref (-xref)
☐ Save temporary files (-save-temps)
☑ Individually map functions and data items (-ip)

Library search directories (-L):
"${OutDir}/system/startup_ldf"

Additional libraries and object files:
"${ProjDirPath}/src/cld_bf70x_audio_1_0_lib.dlb"

Additional libraries and object files:

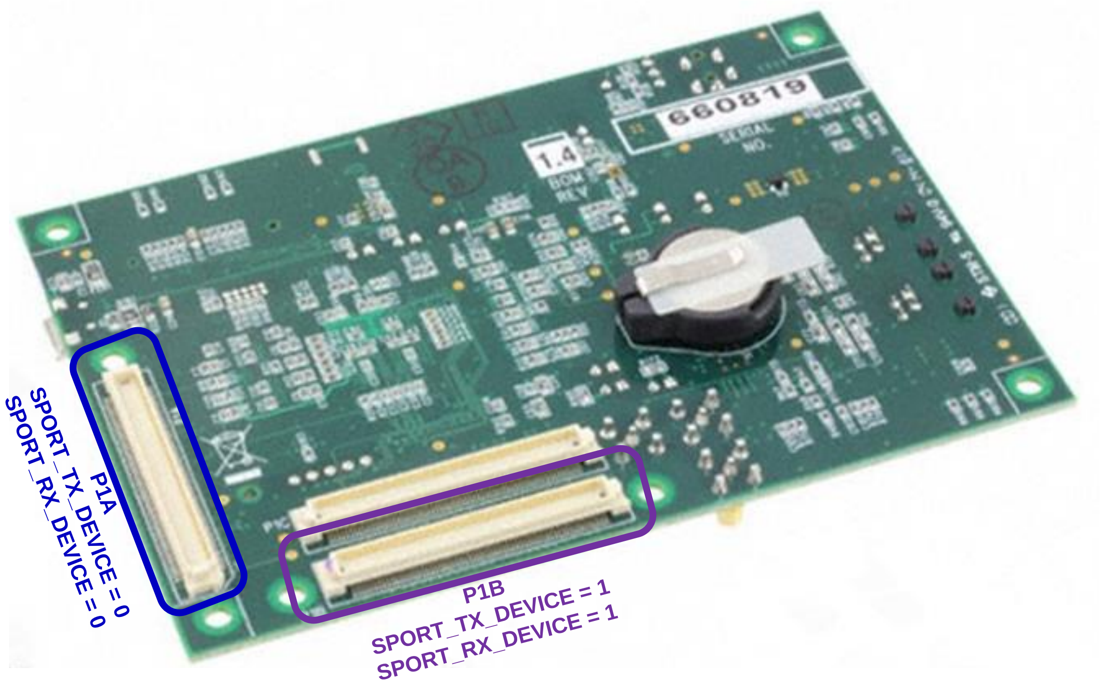
Additional libraries and object files to include in the linking process.

5. The 'Additional libraries and object files' 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).

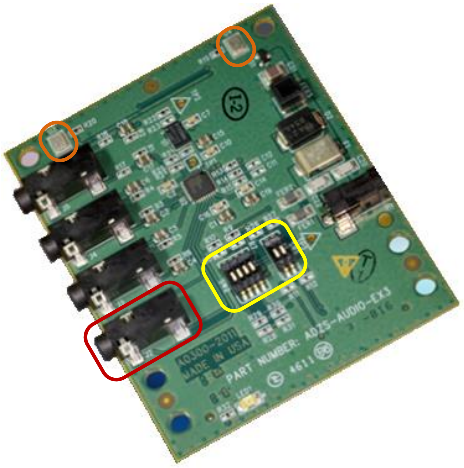
Using the ADI Audio EI3 Extender

Connections:

The Audio EI3 Extender can be connected to ADSP-BF707 Ez-Board using the P1A or P1B connector on the bottom of the Ez-Board (see picture below). By default the CLD Audio 1.0 Example is configured to use the P1A port, but can be modified to use P1B my changing the SPORT_TX_DEVICE and SPORT_RX_DEVICE #define values in user_adau1761.h to 1.



For its audio output the CLD Audio 1.0 example uses the Audio EI3 Extender's headphone jack (circled in red in the picture below). The example's audio input comes from the Audio EI3 Extender's two digital microphones (circled in orange in the picture below). All of the Audio EI3 Extender DIP switches (circled in yellow) should be turned OFF.



Audio EI3 Extender Board Support Package (Required by CLD Audio Example)

The CLD Audio 1.0 Example interfaces to the Audio EI3 Extender's ADAU1761 Codec using the Analog Devices driver provided with the Audio EI3 Extender board support package. This board support package can be downloaded from the Audio EI3 Extender's web page, under the "Product Downloads" section (http://www.analog.com/en/evaluation/eval-bfext-audei3/eb.html).

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 BF70x Audio 1.0 Library please refer to the CLD_Audio_1_0_Ex_v2_0 project included with the CLD BF70x Audio 1.0 Library. The CLD_Audio_1_0_Ex_v2_0 project implements a basic USB Audio device used by the Analog Devices Audio EI3 Extender board.

main.c

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

    while (1)
    {
        switch (main_state)
        {
            case MAIN_STATE_SYSTEM_INIT:
                /* Enable and Configure the SEC. */

                /* sec_gctl - unlock the global lock  */
                pADI_SEC0->GCTL &= ~BITM_SEC_GCTL_LOCK;
                /* sec_gctl - enable the SEC in */
                pADI_SEC0->GCTL |= BITM_SEC_GCTL_EN;
                /* sec_cctl[n] - unlock */
                pADI_SEC0->CB.CCTL &= ~BITM_SEC_CCTL_LOCK;
                /* sec_cctl[n] - reset sci to default */
                pADI_SEC0->CB.CCTL |= BITM_SEC_CCTL_RESET;
                /* sec_cctl[n] - enable interrupt to be sent to core */
                pADI_SEC0->CB.CCTL = BITM_SEC_CCTL_EN;
                pADI_PORTA->DIR_SET = (3 << 0);
                pADI_PORTB->DIR_SET = (1 << 1);

                main_state = MAIN_STATE_USER_INIT;
            break;
            case MAIN_STATE_USER_INIT:
                rv = user_audio_init();
                if (rv == USER_AUDIO_INIT_SUCCESS)
                {
                    main_state = MAIN_STATE_RUN;
                }
                else if (rv == USER_AUDIO_INIT_FAILED)
                {
                    main_state = MAIN_STATE_ERROR;
                }
            break;

            case MAIN_STATE_RUN:
                 user_audio_main();
            break;

            case MAIN_STATE_ERROR:

            break;
        }
    }
}
```

user_audio.c

```
#pragma pack (1)
/* USB Audio v1.0 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 */
    0x0C,                   /* bLength */
    0x24,                   /* bDescriptorType = Class Specific Interface */
    0x02,                   /* bDescriptorSubType = Input Terminal */
    0x01,                   /* bTerminalID */
    0x01, 0x01,             /* wTerminalType = USB Streaming */
    0x00,                   /* bAssocTerminal */
    0x02,                   /* bNRChannels */
    0x03, 0x00,             /* wChannelConfig (Left & Right Present) */
    0x00,                   /* iChannelConfig */
    0x00,                   /* iTerminal */
    /* Input Terminal Descriptor - Microphone */
    0x0C,                   /* bLength */
    0x24,                   /* bDescriptorType = Class Specific Interface */
    0x02,                   /* bDescriptorSubType = Input Terminal */
    0x02,                   /* bTerminalID */
    0x01, 0x02,             /* wTerminalType = Microphone */
    0x00,                   /* bAssocTerminal */
    0x02,                   /* bNRChannels */
    0x03, 0x00,             /* wChannelConfig (Left & Right Present) */
    0x00,                   /* iChannelConfig */
    0x00,                   /* iTerminal */
    /* Output Terminal Descriptor - Speaker */
    0x09,                   /* bLength */
    0x24,                   /* bDescriptorType = Class Specific Interface */
    0x03,                   /* bDescriptorSubType = Output Terminal */
    0x06,                   /* bTerminalID */
    0x01, 0x03,             /* wTerminalType - Speaker */
    0x00,                   /* bAssocTerminal */
    0x09,                   /* bSourceID */
    0x00,                   /* iTerminal */
    /* Output Terminal Descriptor - USB Endpoint */
    0x09,                   /* bLength */
    0x24,                   /* bDescriptorType = Class Specific Interface */
    0x03,                   /* bDescriptorSubType = Output Terminal */
    0x07,                   /* bTerminalID */
    0x01, 0x01,             /* wTerminalType - USB Streaming */
    0x00,                   /* bAssocTerminal */
    0x0a,                   /* bSourceID */
    0x00,                   /* iTerminal */
    /* Feature Unit Descriptor */
    0x0A,                   /* bLength */
    0x24,                   /* bDescriptorType = Class Specific Interface */
    0x06,                   /* bDescriptorSubType = Feature Unit */
    0x09,                   /* bUnitID */
    0x01,                   /* bSourceID */
    0x01,                   /* bControlSize */
    0x01,                   /* mbaControls(0) - Mute Supported */
    0x02,                   /* mbaControls(1) - Volume Supported */
    0x02,                   /* mbaControls(2) - Volume Supported */
    0x00,                   /* iFeature */
    /* Feature Unit Descriptor */
    0x0A,                   /* bLength */
    0x24,                   /* bDescriptorType = Class Specific Interface */
    0x06,                   /* bDescriptorSubType = Feature Unit */
    0x0A,                   /* bUnitID */
    0x02,               /* bSourceID */
    0x01,               /* bControlSize */
    0x02,               /* mbaControls(0) - Volume */
    0x00,               /* mbaControls(1) */
    0x00,               /* mbaControls(2) */
    0x00,               /* iFeature */
};

/* Isochronous IN endpoint PCM format descriptor */
static const unsigned char user_audio_in_stream_format_descriptor[] =
{
    0x0b,               /* bLength */
    0x24,               /* bDescriptorType - Class Specific Interface */
    0x02,               /* bDescriptorSubType - Format Type */
    0x01,               /* bFormatType - Format Type 1 */
    0x02,               /* bNrChannels */
    0x04,               /* bSubFrameSize */
    0x20,               /* bBitResolution */
    0x01,               /* bSamFreqType */
    0x80, 0xBB, 0x00,   /* tSamFreq(1) = 48.0Khz */
};

/* Isochronous OUT endpoint PCM format descriptor */
static const unsigned char user_audio_out_stream_format_descriptor[] =
{
    0x0b,               /* bLength */
    0x24,               /* bDescriptorType - Class Specific Interface */
    0x02,               /* bDescriptorSubType - Format Type */
    0x01,               /* bFormatType - Format Type 1 */
    0x02,               /* bNrChannels */
    0x04,               /* bSubFrameSize */
    0x20,               /* bBitResolution */
    0x01,               /* bSamFreqType */
    0x80, 0xBB, 0x00,   /* tSamFreq(1) = 48.0Khz */
};
#pragma pack ()

/* IN Audio Stream Interface Endpoint Data Descriptor */
static const CLD_BF70x_Audio_1_0_Lib_Audio_Stream_Data_Endpoint_Descriptor
             user_audio_in_stream_endpoint_desc =
{
    .b_length = sizeof(CLD_BF70x_Audio_1_0_Lib_Audio_Stream_Data_Endpoint_Descriptor),
    .b_descriptor_type                  = 0x25,         /* Class Specific Endpoint */
    .b_descriptor_subtype               = 0x01,         /* Endpoint - General */
    .bm_attributes                      = 0x01,         /* sampling freq supported */
    .b_lock_delay_units                 = 0x00,         /* Undefined */
    .w_lock_delay                       = 0x00,
};

static const CLD_BF70x_Audio_1_0_Lib_Audio_Stream_Data_Endpoint_Descriptor
             user_audio_out_stream_endpoint_desc =
{
    .b_length = sizeof(CLD_BF70x_Audio_1_0_Lib_Audio_Stream_Data_Endpoint_Descriptor),
    .b_descriptor_type                  = 0x25,         /* Class Specific Endpoint */
    .b_descriptor_subtype               = 0x01,         /* Endpoint - General */
    .bm_attributes                      = 0x01,         /* sampling freq supported */
    .b_lock_delay_units                 = 0x01,         /* Milliseconds */
    .w_lock_delay                       = 0x01,         /* 1 Millisecond */
};

/* Audio Stream IN Interface parameters */
static CLD_BF70x_Audio_1_0_Stream_Interface_Params user_audio_in_endpoint_params =
{
    .endpoint_number            = 1,                /* Isochronous endpoint number */
    .max_packet_size_full_speed = 400,              /* Isochronous endpoint full-speed
                                                       max packet size */
    .max_packet_size_high_speed = 400,              /* Isochronous endpoint high-speed
                                                       max packet size */
    .b_interval_full_speed      = 1,                /* Isochronous endpoint full-speed
                                                       bInterval */
    .b_interval_high_speed      = 4,                /* Isochronous endpoint high-speed
                                                       bInterval - 1 millisecond */
    .synchronization_type       = 0x1,              /* Isochronous endpoint
                                                       synchronization type =
                                                       Asynchronous */
    .b_terminal_link            = 7,                /* Terminal ID of the associated
                                                      Output Terminal */
    .b_delay                    = 1,                /* Delay = 1 Frame */
    .w_format_tag               = 1,                /* PCM */
                                                    /* Pointer to the PCM Format
                                                       Descriptor */
    .p_format_type_descriptor   = (unsigned char*)
                                  user_audio_in_stream_format_descriptor,
    .p_audio_stream_endpoint_data_descriptor =
                    (CLD_BF70x_Audio_1_0_Lib_Audio_Stream_Data_Endpoint_Descriptor*)
                    &user_audio_in_stream_endpoint_desc,
};

/* Audio Stream OUT Interface parameters */
static CLD_BF70x_Audio_1_0_Stream_Interface_Params user_audio_out_endpoint_params =
{
    .endpoint_number            = 1,                /* Isochronous endpoint number */
    .max_packet_size_full_speed = 400,              /* Isochronous endpoint full-speed
                                                       max packet size */
    .max_packet_size_high_speed = 400,              /* Isochronous endpoint high-speed
                                                       max packet size */
    .b_interval_full_speed      = 1,                /* Isochronous endpoint full-speed
                                                       bInterval */
    .b_interval_high_speed      = 4,                /* Isochronous endpoint high-speed
                                                       bInterval - 1 millisecond */
    .synchronization_type       = 0x2,              /* Isochronous endpoint
                                                       synchronization type = Adaptive
                                                       */
    .b_terminal_link            = 1,                /* Terminal ID of the associated
                                                       Output Terminal */
    .b_delay                    = 1,                /* Delay = 1 Frame */
    .w_format_tag               = 1,                /* PCM */
                                                    /* Pointer to the PCM Format
                                                       Descriptor */
    .p_format_type_descriptor   = (unsigned char*)
                                 user_audio_out_stream_format_descriptor,
    .p_audio_stream_endpoint_data_descriptor =
                    (CLD_BF70x_Audio_1_0_Lib_Audio_Stream_Data_Endpoint_Descriptor*)
                    &user_audio_out_stream_endpoint_desc,
};


/* CLD BF70x Audio 1.0 library initialization data. */
static CLD_BF70x_Audio_1_0_Lib_Init_Params user_audio_init_params =
{
    .uart_num   = CLD_UART_0,
    .uart_baud  = 115200,
    .sclk0      = 100000000u,
    .fp_console_rx_byte = user_audio_console_rx_byte,
    .vendor_id = 0x064b,
    .product_id = 0x0005,
    .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),

    .p_audio_streaming_rx_interface_params = &user_audio_out_endpoint_params,
    .p_audio_streaming_tx_interface_params = &user_audio_in_endpoint_params,

    .fp_audio_stream_data_received = user_audio_stream_data_received,

    .fp_audio_set_req_cmd = user_audio_set_req_cmd,
    .fp_audio_get_req_cmd = user_audio_get_req_cmd,

    .fp_audio_streaming_rx_endpoint_enabled =user_audio_streaming_rx_endpoint_enabled,
    .fp_audio_streaming_tx_endpoint_enabled =user_audio_streaming_tx_endpoint_enabled,


    .usb_bus_max_power = 0,
    .device_descriptor_bcdDevice = 0x0100,

    /* USB string descriptors - Set to CLD_NULL if not required */
    .p_usb_string_manufacturer  = "Analog Devices Inc",
    .p_usb_string_product       = "BF707 Audio v1.0 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 = CLD_NULL,
    .p_usb_string_audio_streaming_in_interface = CLD_NULL,

    .user_string_descriptor_table_num_entries = 0,
    .p_user_string_descriptor_table = CLD_NULL,

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

    .fp_cld_usb_event_callback = user_audio_usb_event,
    .fp_cld_lib_status         = user_audio_status,
};
```

```
User_Audio_Init_Return_Code user_audio_init (void)
{
    static unsigned char user_init_state = 0;
    CLD_RV cld_rv = CLD_ONGOING;
    User_Audio_Init_Return_Code init_return_code = USER_AUDIO_INIT_ONGOING;

    switch (user_init_state)
    {
        case 0:

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

            user_init_state++;
        break;
        case 1:
            /* TODO: Configure a timer to generate an interrupt every 125
                     microseconds, and call cld_time_125us_tick from interrupt. */
            /* TODO: Install USB and optionally the Console UART ISRs. */
            /* Initialize the CLD BF70x Audio 1.0 Library */
            cld_rv = cld_bf70x_audio_1_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_AUDIO_INIT_SUCCESS;
            }
            else if (cld_rv == CLD_FAIL)
            {
                init_return_code = USER_AUDIO_INIT_FAILED;
            }
            else
            {
                init_return_code = USER_AUDIO_INIT_ONGOING;
            }
    }
    return init_return_code;
}

void user_audi_main (void)
{
    cld_bf70x_audio_1_0_lib_main();

}

/* Function called when an Isochronous OUT packet is received */
static CLD_USB_Transfer_Request_Return_Type user_audio_stream_data_received
                           (CLD_USB_Transfer_Params * p_transfer_data)
{
    p_transfer_data->num_bytes = /* TODO: Set number of Isochronous OUT bytes to transfer
                                 */
    p_transfer_data->p_data_buffer = /* TODO: address to store Isochronous OUT data */

    /* User Audio transfer complete callback function. */
    p_transfer_data->fp_callback.usb_out_transfer_complete =
                                        user_audio_stream_data_rx_done;
    p_transfer_params->fp_transfer_aborted_callback = /* TODO: Set to User callback
                                                         function or CLD_NULL */;
    p_transfer_params->transfer_timeout_ms = /* TODO: Set to desired timeout */;


    /* TODO: Return how the Isochronous OUT transfer should be handled (Accept, Pause,
             Discard, or Stall */
}
/* The function below is an example if the Isochronous OUT transfer done callback
       specified in the CLD_USB_Transfer_Params structure. */
static CLD_USB_Data_Received_Return_Type user_audio_stream_data_rx_done (void)
{
    /* TODO: Process the received Isochronous OUT transfer and return if the received
             data is good(CLD_USB_DATA_GOOD) or if there is an error
             (CLD_USB_DATA_BAD_STALL)*/

}

static void user_audio_console_rx_byte (unsigned char byte)
{
    /* TODO: Add any User firmware to process data received by the CLD Console UART.*/
}

static void user_audio_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_bf70x_audio_1_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_BF70x_Audio_1_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_BF70x_Audio_1_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);
}
```