USB Communication Device Class (CDC) Abstract Control Model Library for Analog Devices ADSP-BF70x User's Guide Revision 2. 0

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) CDC library creates a simplified interface for developing a Communication Device Class (CDC) Abstract Control Model (ACM) Serial Emulation device using the Analog Devices ADSP-BF707 EZ-Board. The CLD BF70x CDC Library also includes support for a serial console and timer functions which facilitates creating timed events quickly and easily. The library's BF707 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 CDC Library API section of this document.

USB Background

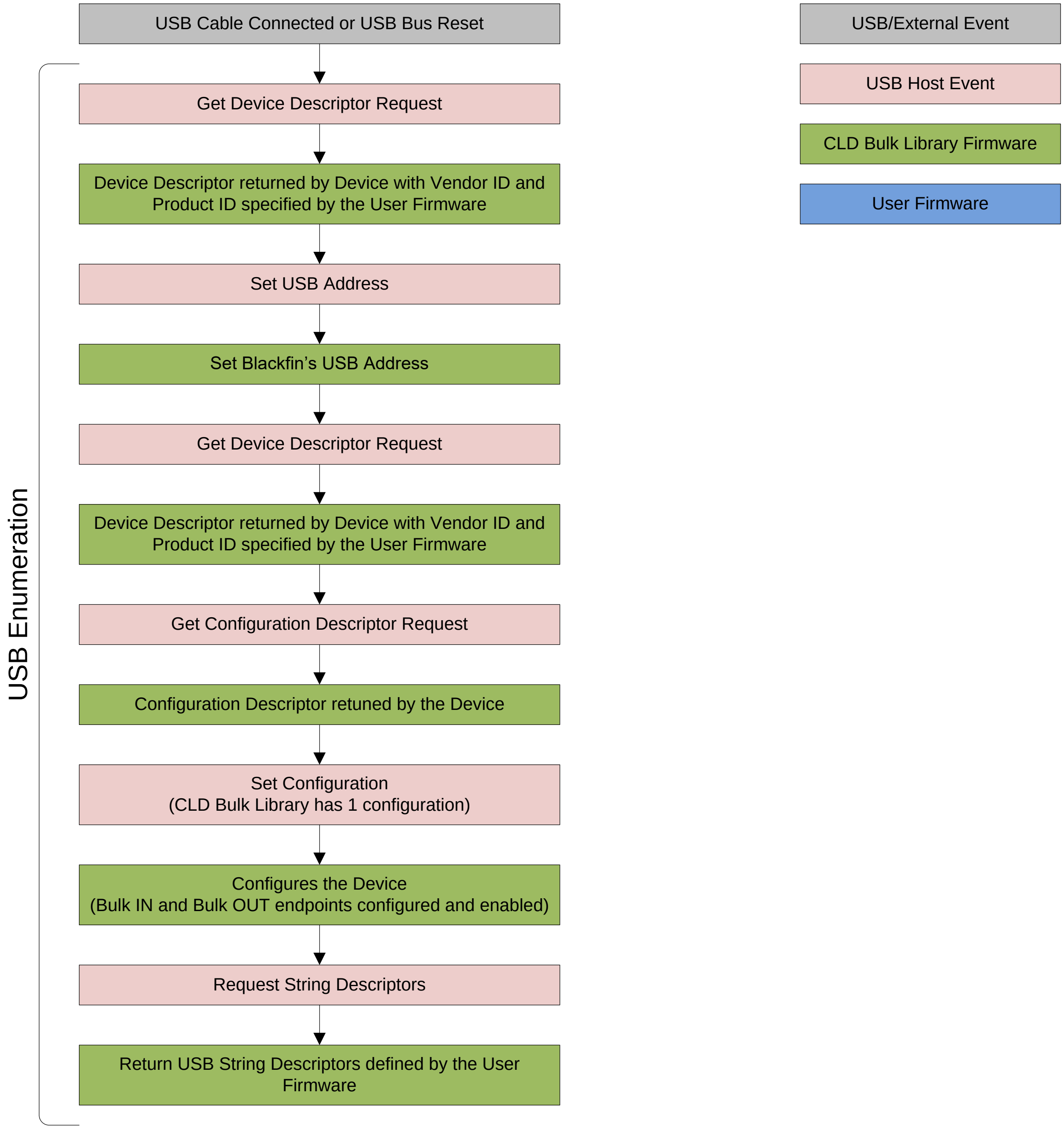
The following is a very basic overview of some of the USB concepts which are necessary to use the CLD BF70x CDC Library. However, it is still recommended that developers have at least a basic understanding of the USB 2.0 protocol as well as the CDC 1.2 Protocol. The following are some resources to refer to when working with USB:

- The USB 2.0 Specification: http://www.usb.org/developers/docs/usb20_docs/
- The USB CDC Class specification v1.2:http://www.usb.org/developers/docs/devclass_docs/
- 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, so all USB terminology is from the Host's perspective. For example a '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 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 used by a USB Host 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 the USB Enumeration. The CLD BF70x CDC 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_cdc_lib_init section of this document. The CLD BF70x CDC Library facilitates USB enumeration and is Chapter 9 compliant without User Application intervention as shown in the flow chart below. If you'd like additional information on USB Chapter 9 functionality or USB Enumeration please refer to one of the USB resources listed above.

CLD BF70x CDC 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 Bulk Library has 1 configuration)
Configures the Device
(Bulk IN and Bulk OUT endpoints configured and enabled)
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 which 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 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 CDC Library uses Control, Interrupt and Bulk endpoints, so these endpoint types will be discussed in more detail below.

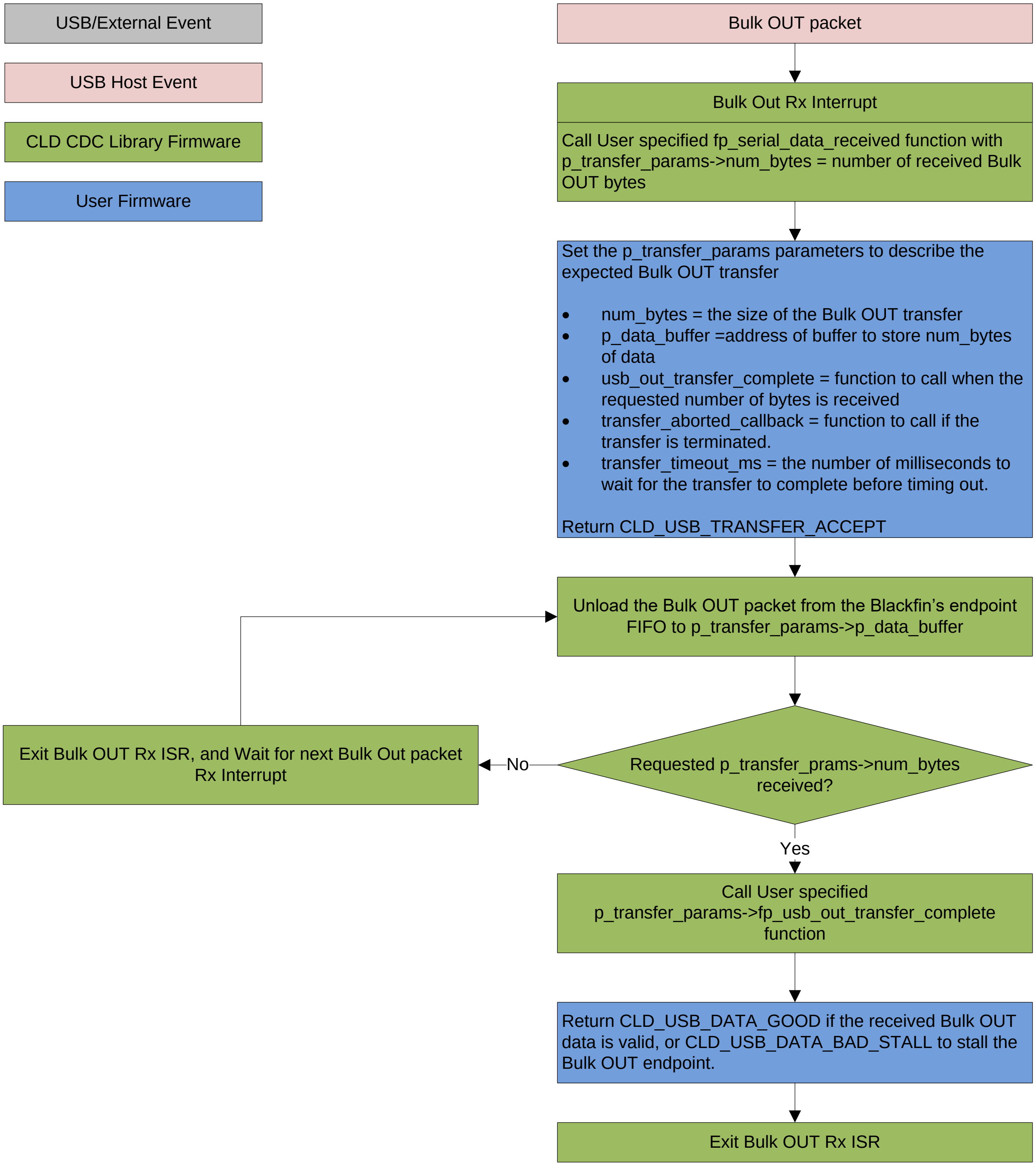
A Control Endpoint is the only bi-directional endpoint type, and is typically used for command and status transfers. A Control Endpoint transfer is made up of three stages (Setup Stage, Data Stage and Status Stage). The Setup Stage sets the direction and size of the optional Data Stage. The Data Stage 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 CDC Library Endpoint 0 is used for USB Chapter 9 requests, as well as CDC requests. These CDC requests are discussed in more detail in the CDC Abstract Control Model Background section 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. An example of how this is used with CDC is when a parity error is detect. When a CDC device detects a parity error the device reports the error condition to the Host in a Serial State Notification using the CDC Interrupt IN endpoint. This is more efficient then requiring the host to repeatedly send Control Endpoint requests asking an error has occurred.

Bulk Endpoints are used to transfer large amounts of data where data integrity is critical, but does not require deterministic timing. A characteristic of Bulk Endpoints is that they can fill USB bandwidth that isn't used by the other endpoint types. This makes Bulk the lowest priority endpoint type, but it can also be the fastest as long as the other endpoints don't saturate the USB Bus. An example of a devices that uses Bulk endpoints is a Mass Storage Device (thumb drives). The CLD BF70x CDC Library includes a Bulk IN and Bulk OUT endpoint, which are used to send and receive serial data with the USB Host, respectively.

The flow charts below give an overview of how the CLD BF70x CDC Library and the User firmware interact to process Bulk OUT and Bulk IN transfers. For the Interrupt IN endpoint the CLD BF70x CDC Library uses individual functions to send CDC Notifications, which abstracts the User from the Interrupt IN endpoint. Additionally, the User firmware code snippets included at the end of this document provide a basic framework for implementing the CDC firmware using the CLD BF70x CDC Library.

CLD BF70x CDC Library Bulk OUT Flow Chart



Bulk OUT packet

Bulk Out Rx Interrupt

Call User specified fp_serial_data_received function with p_transfer_params->num_bytes = number of received Bulk OUT bytes

Set the p_transfer_params parameters to describe the expected Bulk OUT transfer

- num_bytes = the size of the Bulk OUT transfer
- p_data_buffer =address of buffer to store num_bytes of data
- usb_out_transfer_complete = function to call when the requested number of bytes is received
- transfer_aborted_callback = function to call if the transfer is terminated.
- transfer_timeout_ms = the number of milliseconds to wait for the transfer to complete before timing out.

Return CLD_USB_TRANSFER_ACCEPT

Unload the Bulk OUT packet from the Blackfin's endpoint FIFO to p_transfer_params->p_data_buffer

Requested p_transfer_prams->num_bytes received?

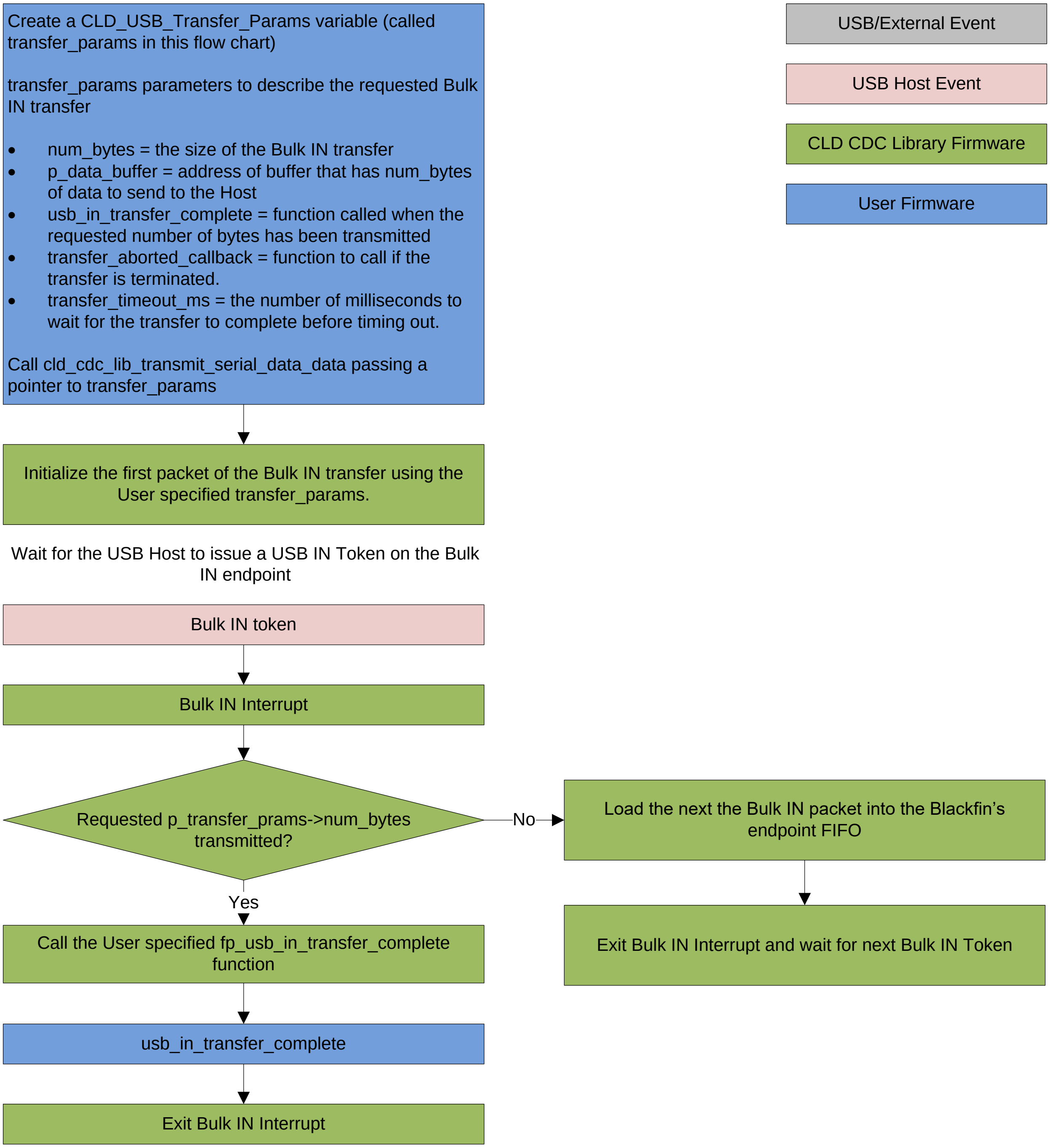
Exit Bulk OUT Rx ISR, and Wait for next Bulk Out packet Rx Interrupt

Call User specified p_transfer_params->fp_usb_out_transfer_complete function

Return CLD_USB_DATA_GOOD if the received Bulk OUT data is valid, or CLD_USB_DATA_BAD_STALL to stall the Bulk OUT endpoint.

Exit Bulk OUT Rx ISR

CLD BF70x CDC Library Bulk IN Flow Chart



USB/External Event

USB Host Event

CLD CDC 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 Bulk IN transfer

- num_bytes = the size of the Bulk IN transfer
- p_data_buffer = address of buffer that has num_bytes of data to send to the Host
- usb_in_transfer_complete = function called when the requested number of bytes has been transmitted
- transfer_aborted_callback = function to call if the transfer is terminated.
- transfer_timeout_ms = the number of milliseconds to wait for the transfer to complete before timing out.

Call cld_cdc_lib_transmit_serial_data_data passing a pointer to transfer_params

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

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

Bulk IN token

Bulk IN Interrupt

Requested p_transfer_prams->num_bytes transmitted?

Load the next the Bulk IN packet into the Blackfin’s endpoint FIFO

Exit Bulk IN Interrupt and wait for next Bulk IN Token

Call the User specified fp_usb_in_transfer_complete function

usb_in_transfer_complete

Exit Bulk IN Interrupt

CDC Abstract Control Model Background

The USB Communication Device Class (CDC) Abstract Control Model (ACM) protocol is a USB Standard Class protocol released by the USB IF committee. The Communication Device Class was created to provide a standardized way for USB communication devices to interface with a computer, and covers a wide range of communication devices. The CLD BF70x CDC Library implements a Abstract Control Model Serial Emulation device, so the scope of this document is limited to the CDC ACM Serial Emulation functionality.

A CDC device is comprised of two USB interfaces. The first interface uses the Communication Device Class and includes a single Interrupt IN endpoint used to send Notifications to the host. The second interface uses the Data Interface Class and includes a Bulk IN and Bulk OUT endpoint, which are used to transfer the serial emulation data with the USB Host.

CDC Notifications Interrupt IN Endpoint

The CDC protocol requires all devices to include a Interrupt IN endpoint which is used to send CDC Notifications to the Host. For the CDC Abstract Control Model these Notifications include the Network Connection, Response Available and Serial State Notifications. These Notifications are discussed below:

Network Connection Notification

The Network Connection Notification is used to report if the network is connected or disconnected to the Host.

Response Available Notification

The Response Available Notification is used to notify the Host that a protocol specific response is available, which is retrieved by the Host using the Get Encapsulated Response control endpoint request described in the CDC Abstract Control Model Control Endpoint Requests section of this document.

Serial State Notification

The Serial State Notification is similar to the interrupt status register of a UART, and is used to report the serial link status to the Host. The table below shows the data fields of the Serial State Notification. All of the Serial State fields are active high, so a field is set to a '1' when it is active.

Field	Description
bOverRun	Received serial data was received while processing the previously received data.
bParity	A parity error has occurred.
bFraming	A framing error has occurred
bRingSignal	The current state of the ring signal detection
bBreak	The current state of the break detection.
bTxCarrier	State of the transmission carrier. This corresponds to the RS-232 DSR signal.
bRxCarrier	State of the receive carrier detection. This signal corresponds to the RS-232 DCD signal.

Once the Serial State Notification has been sent the device will re-evaluate the above fields. For the bTxCarrier and bRxCarrier the Serial State Notification is sent when these signals change. For the remaining fields once the Serial State Notification has been sent their value is reset to zero, and will be sent again when the field is set to a '1'.

CDC Abstract Control Model Control Endpoint Requests

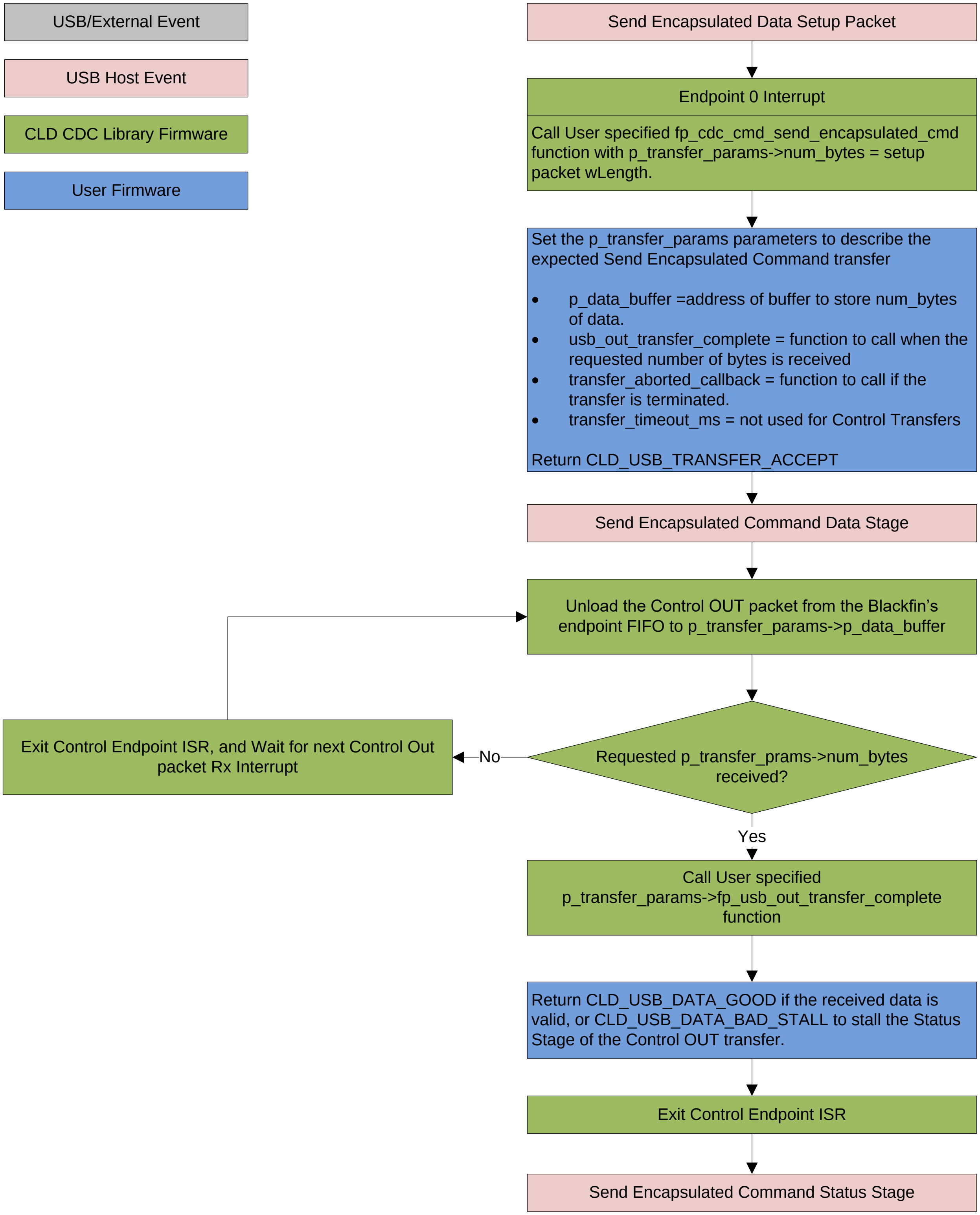
The CDC Abstract Control Model defines a couple Control Endpoint requests that a CDC peripheral is required to support as well as some optional Control Endpoint requests. The Control Endpoint requests used by the CLD BF70x CDC Library are explained in the following sections, and include flow charts showing how the CLD BF70x CDC Library and the User firmware interact to the Control Endpoint requests.

Additionally, the User firmware code snippets included at the end of this document provide a basic framework for implementing the CDC control requests using the CLD BF70x CDC Library.

Send Encapsulated Command (required)

Send Encapsulated Command is a Control OUT request and is used by the Host to send protocol specific data to the device.

CLD CDC Send Encapsulated Command Flow Chart



Send Encapsulated Data Setup Packet

Endpoint 0 Interrupt

Call User specified fp_cdc_cmd_send_encapsulated_cmd function with p_transfer_params->num_bytes = setup packet wLength.

Set the p_transfer_params parameters to describe the expected Send Encapsulated Command transfer

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

Return CLD_USB_TRANSFER_ACCEPT

Send Encapsulated Command 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?

Exit Control Endpoint ISR, and Wait for next Control Out packet Rx Interrupt

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.

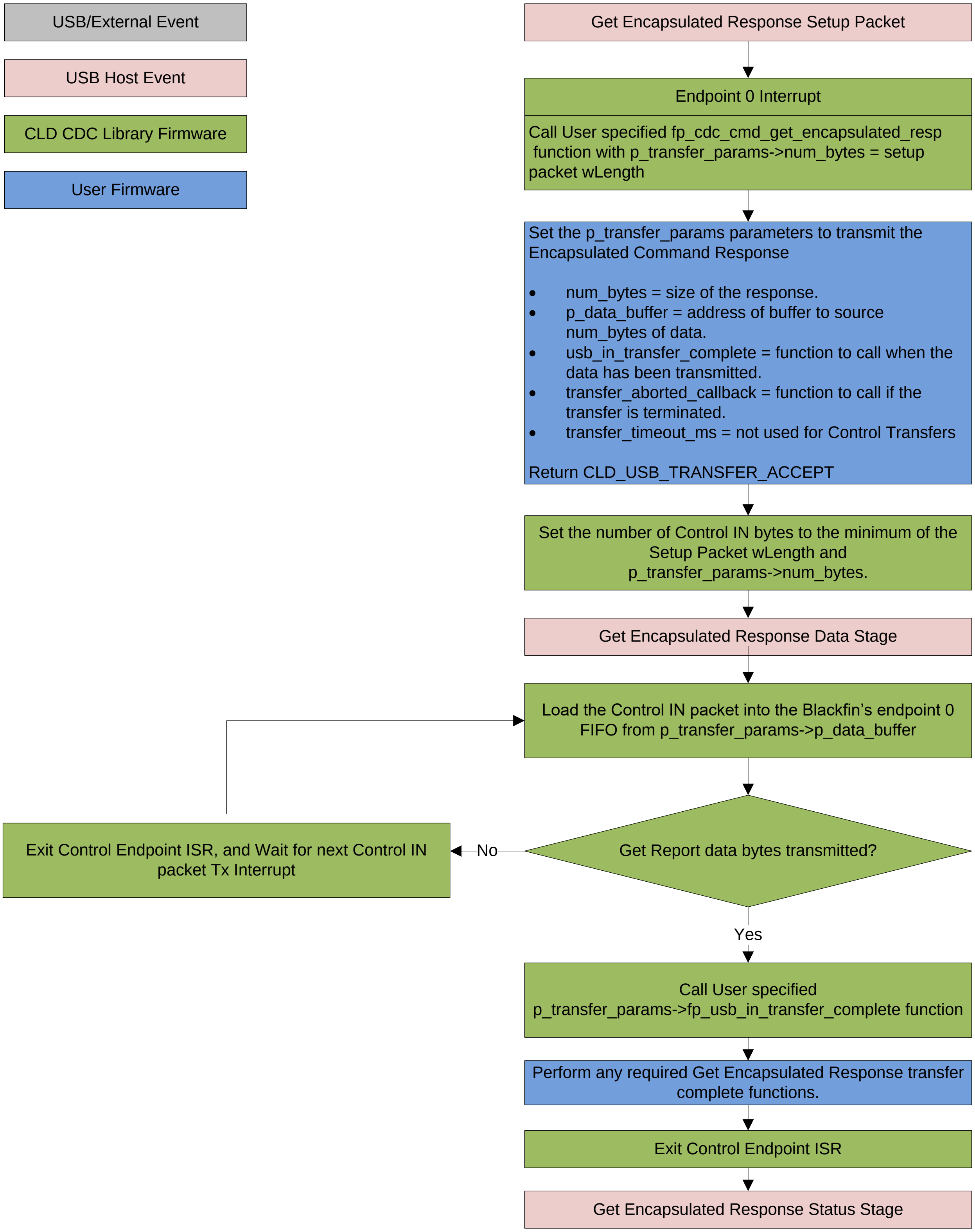
Exit Control Endpoint ISR

Send Encapsulated Command Status Stage

Get Encapsulated Command (required)

Get Encapsulated Command is a Control IN request used by the Host to request protocol specified data.

CLD BF70x CDC Library Get Encapsulated Command Flow Chart



USB/External Event
USB Host Event
CLD CDC Library Firmware
User Firmware
Get Encapsulated Response Setup Packet
Endpoint 0 Interrupt
Call User specified fp_cdc_cmd_get_encapsulated_resp function with p_transfer_params->num_bytes = setup packet wLength
Set the p_transfer_params parameters to transmit the Encapsulated Command Response
• num_bytes = size of the response.
• p_data_buffer = address of buffer to source num_bytes of data.
• usb_in_transfer_complete = function to call when the data has been transmitted.
• 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 Encapsulated Response Data Stage
Load the Control IN packet into the Blackfin's endpoint 0 FIFO from p_transfer_params->p_data_buffer
Get Report data bytes transmitted?
No
Exit Control Endpoint ISR, and Wait for next Control IN packet Tx Interrupt
Yes
Call User specified p_transfer_params->fp_usb_in_transfer_complete function
Perform any required Get Encapsulated Response transfer complete functions.
Exit Control Endpoint ISR
Get Encapsulated Response Status Stage

Set Line Coding (optional)

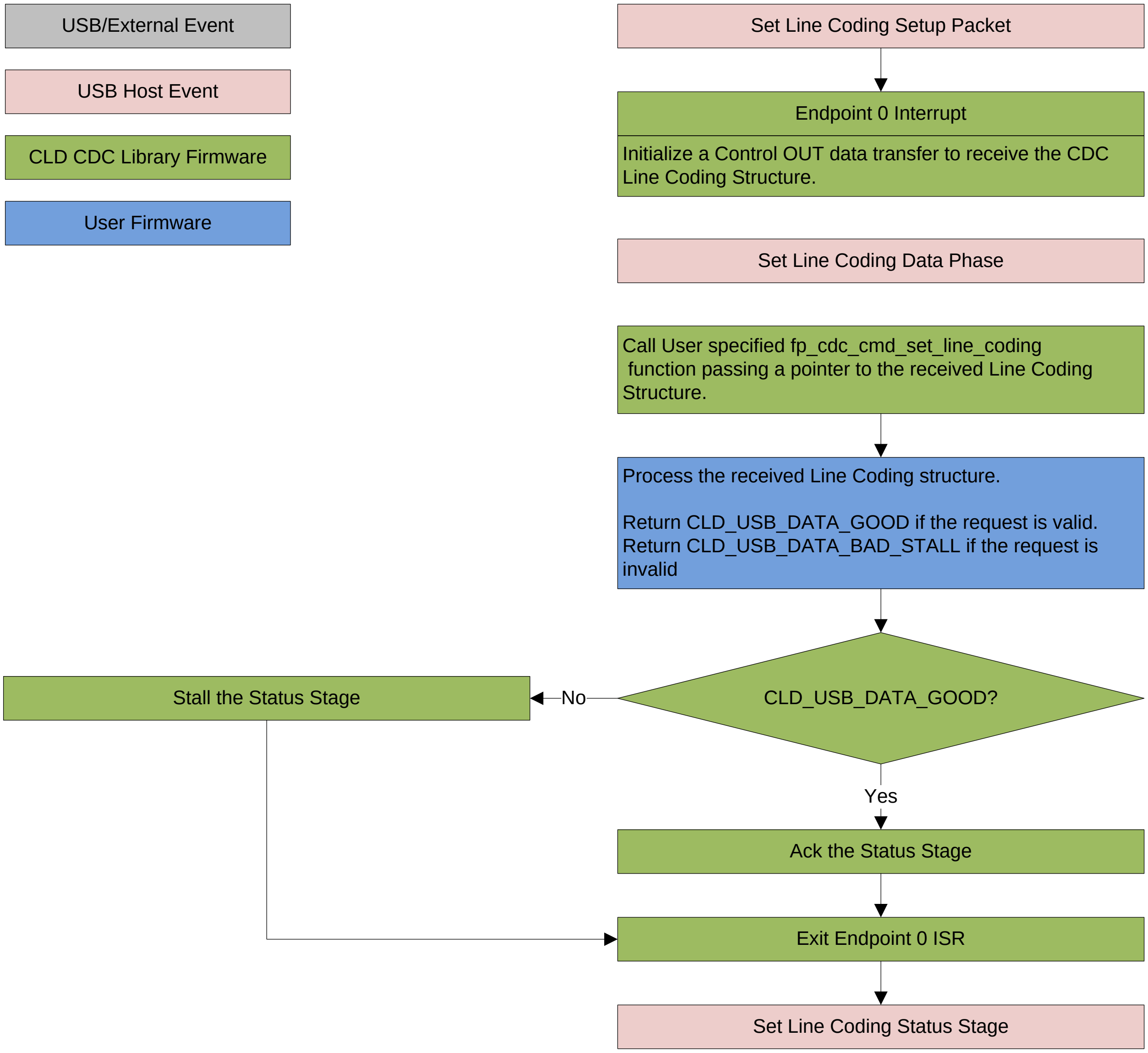
The Set Line Coding Control OUT request is used by the Host configure the UART parameters of emulated serial port. The Set Line Coding request includes the following line coding structure in the Control OUT Data Phase.

```
typedef struct
{
    unsigned long data_terminal_rate;                   /* CDC Data Terminal Rate in
                                                            bits per second. */
    unsigned char num_stop_bits;                        /* CDC Number of stop bits
                                                            0 = 1 stop bit
                                                            1 = 1.5 stop bits
                                                            2 = 2 stop bits */
    unsigned char parity;                               /* CDC Parity setting
                                                            0 = None
                                                            1 = Odd
                                                            2 = Even
                                                            3 = Mark
                                                            4 = Space */
    unsigned char num_data_bits;                        /* CDC number of data bits
                                                            (Only 5, 6, 7, 8 and 16
                                                             allowed) */

} CLD_CDC_Line_Coding;
```

In response to a Set Line Coding command the CDC device should implement the requested configuration, or stall the endpoint if the request is invalid.

CLD BF70x CDC Library Set Line Coding Flow Chart

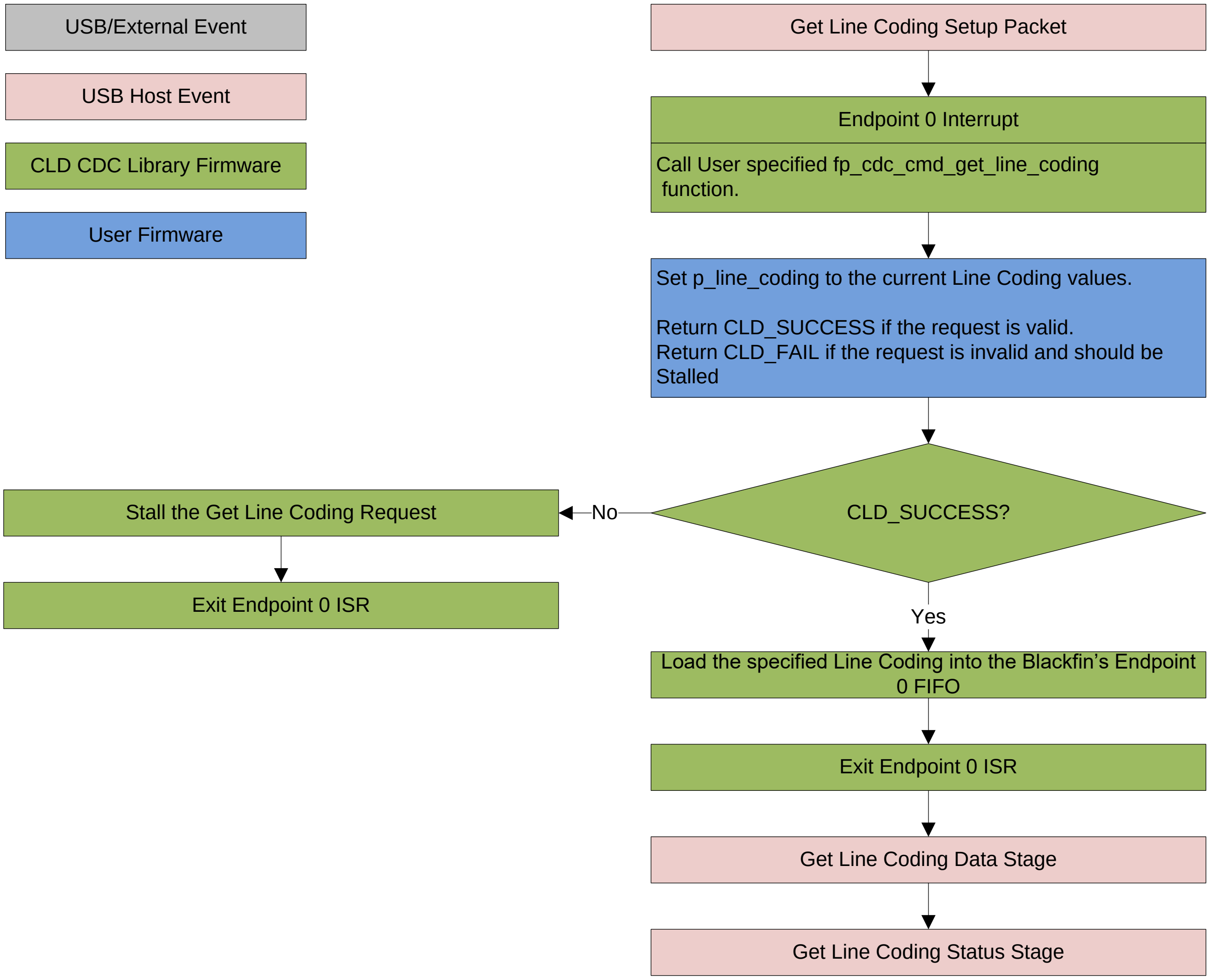


USB/External Event
USB Host Event
CLD CDC Library Firmware
User Firmware
Set Line Coding Setup Packet
Endpoint 0 Interrupt
Initialize a Control OUT data transfer to receive the CDC Line Coding Structure.
Set Line Coding Data Phase
Call User specified fp_cdc_cmd_set_line_coding function passing a pointer to the received Line Coding Structure.
Process the received Line Coding structure.
Return CLD_USB_DATA_GOOD if the request is valid.
Return CLD_USB_DATA_BAD_STALL if the request is invalid
CLD_USB_DATA_GOOD?
No
Stall the Status Stage
Yes
Ack the Status Stage
Exit Endpoint 0 ISR
Set Line Coding Status Stage

Get Line Coding (optional)

The Get Line Coding Control IN request is used by the Host request current UART parameters of emulated serial port. The Get Line Coding request includes line coding structure described in the Set Line Coding section in the Control IN Data Phase.

CLD BF70x CDC Library Get Line Coding Flow Chart

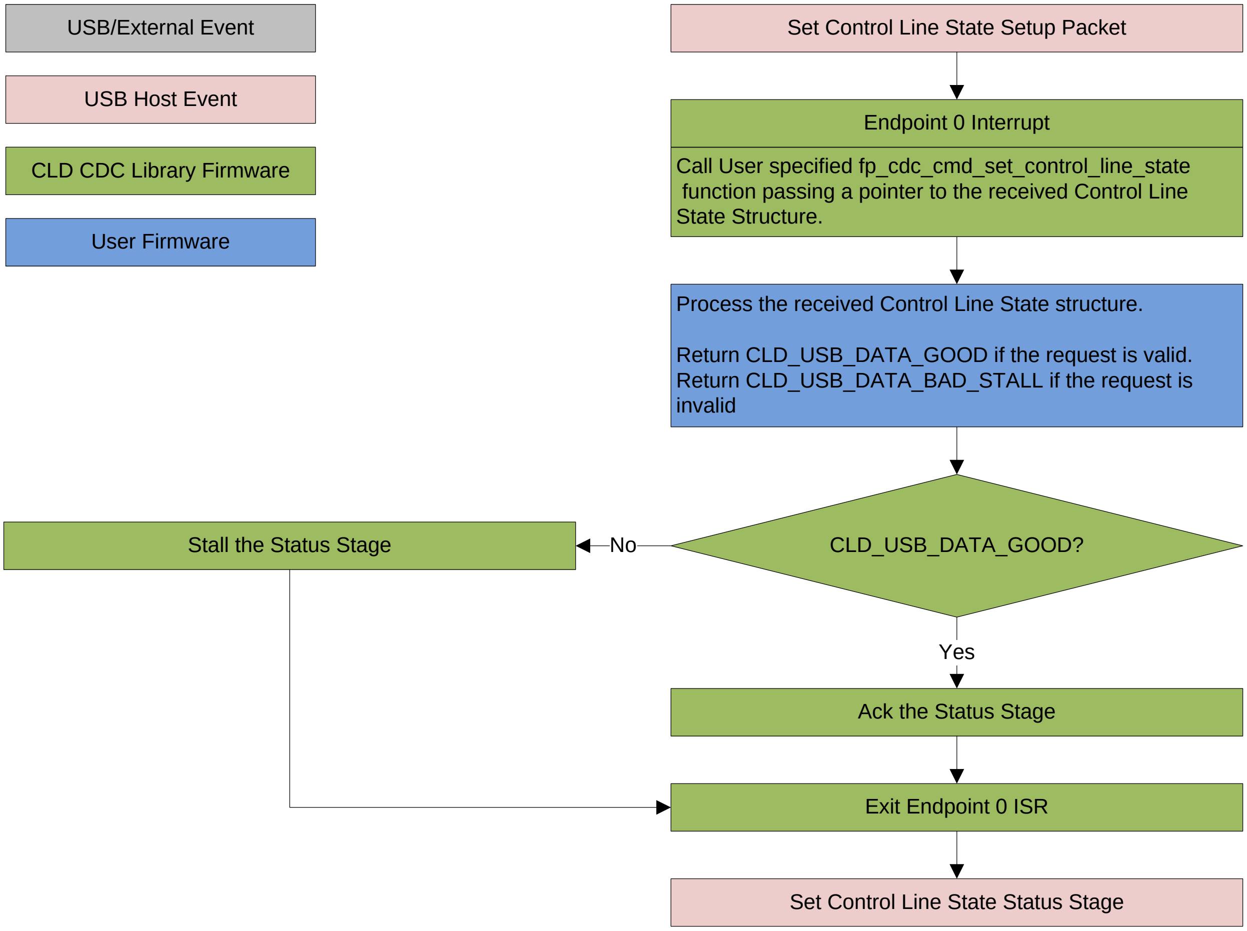

USB/External Event
USB Host Event
CLD CDC Library Firmware
User Firmware
Get Line Coding Setup Packet
Endpoint 0 Interrupt
Call User specified fp_cdc_cmd_get_line_coding function.
Set p_line_coding to the current Line Coding values.
Return CLD_SUCCESS if the request is valid.
Return CLD_FAIL if the request is invalid and should be Stalled
CLD_SUCCESS?
No
Stall the Get Line Coding Request
Exit Endpoint 0 ISR
Yes
Load the specified Line Coding into the Blackfin's Endpoint 0 FIFO
Exit Endpoint 0 ISR
Get Line Coding Data Stage
Get Line Coding Status Stage

Set Control Line State (optional)

The Set Control Line State Control OUT request is used by the Host to set the value of the emulated serial port RS-232 RTS and DTR control signals. The Set Control Line State request includes the following control signal structure in the Control OUT Data Phase.

```
typedef struct
{
    union
    {
        struct
        {
            unsigned short dte_present  : 1;            /* Indicates to DCE if DTE is
                                                           present or not.
                                                           This signal corresponds to
                                                           V.24 signal 108/2
                                                           and RS-232 signal DTR.
                                                              0 - Not Present
                                                              1 - Present   */
            unsigned short activate_carrier : 1;        /* Carrier control for half
                                                           duplex modems.
                                                           This signal corresponds to
                                                           V.24 signal 105 and RS-232
                                                           signal RTS.
                                                              0 - Deactivate carrier
                                                              1 - Activate carrier
                                                           The device ignores the
                                                           value of this bit when
                                                           operating in full duplex
                                                           mode.  */
            unsigned short reserved         : 14;
        } bits;
        unsigned short state;
    } u;
} CLD_CDC_Control_Line_State;
```

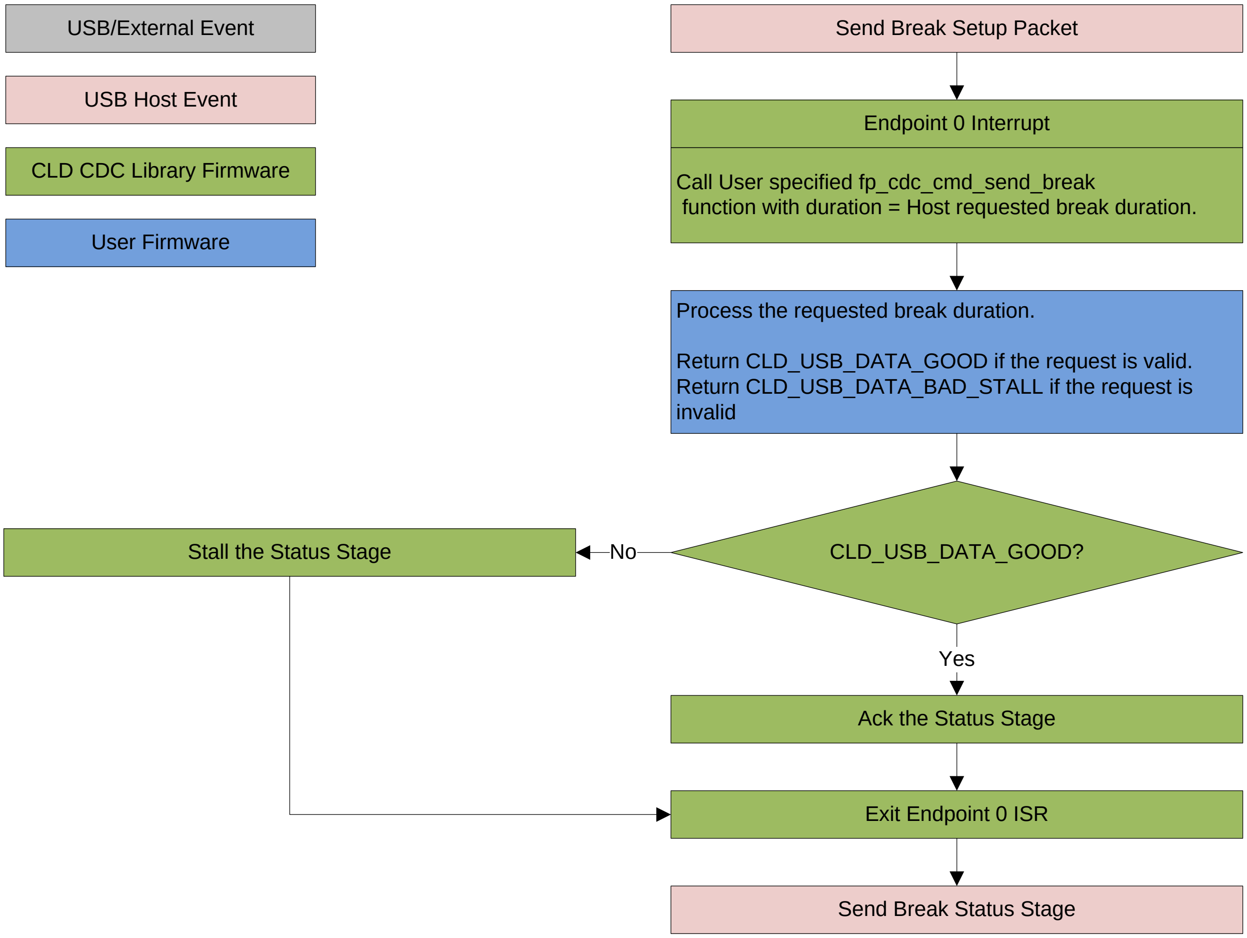
CLD BF70x CDC Library Set Control Line State Flow Chart


USB/External Event
USB Host Event
CLD CDC Library Firmware
User Firmware
Set Control Line State Setup Packet
Endpoint 0 Interrupt
Call User specified fp_cdc_cmd_set_control_line_state function passing a pointer to the received Control Line State Structure.
Process the received Control Line State structure.
Return CLD_USB_DATA_GOOD if the request is valid.
Return CLD_USB_DATA_BAD_STALL if the request is invalid
CLD_USB_DATA_GOOD?
No
Stall the Status Stage
Yes
Ack the Status Stage
Exit Endpoint 0 ISR
Set Control Line State Status Stage

Send Break (optional)

The Send Break Control OUT request is used by the Host request the device to generate a RS-232 style break for the specified duration (in milliseconds). If the duration is set to 0xFFFF the device should generate a break until another Send Break command is received with a duration of 0.

CLD BF70x CDC Library Send Break Flow Chart


USB/External Event
USB Host Event
CLD CDC Library Firmware
User Firmware
Send Break Setup Packet
Endpoint 0 Interrupt
Call User specified fp_cdc_cmd_send_break function with duration = Host requested break duration.
Process the requested break duration.
Return CLD_USB_DATA_GOOD if the request is valid.
Return CLD_USB_DATA_BAD_STALL if the request is invalid
CLD_USB_DATA_GOOD?
No
Stall the Status Stage
Yes
Ack the Status Stage
Exit Endpoint 0 ISR
Send Break Status Stage

Dependencies

In order to function properly the CLD BF70x CDC Library requires the following Blackfin resources:

- 24Mhz clock input connected to the Blackfin USB0_CLKIN pin.
- Optionally the CLD BF70x CDC 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 CDC Library approximate memory footprint is as follows:

Code memory:	29480 bytes
Data memory:	4884 bytes
Total:	34364 bytes or 33.56k
Heap memory:	1152 bytes (only malloc'ed if optional cld_console is enabled)

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

CLD BF70x CDC Library Scope and Intended Use

The CLD BF70x CDC Library implements a USB Communication Class Abstract Control Model Serial Emulation device, as well as providing time measurements and optional bi-directional UART console functionality. The CLD BF70x CDC 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 CDC Uart Example v2.0 Description

The CLD_CDC_Uart_example_v2_0 project provided with the CLD BF70x CDC Library implements a basic USB to Serial device using the ADSP-BF707 EZ-Board and one of the BF707's UARTs. The firmware included in this example to interface with the BF707 UART uses the Analog Devices System Services driver. This was done to show how the CLD BF70x CDC Library co-exists with the ADI System Services. This example is not indented to be a used as a complete stand alone project. Instead, this project only includes the User functionality required to create a basic USB to Serial device, and it is up to the User to include their own custom system initialization and any extra required functionality.

CLD BF70x CDC 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_cdc_lib_init

```
CLD_RV cld_bf70x_cdc_lib_init (CLD_BF70x_CDC_Lib_Init_Params *
cld_bf70x_cdc_lib_params)
```

Initialize the CLD BF70x CDC Library.

Arguments

`cld_bf70x_cdc_lib_params`	Pointer to a CLD_BF70x_CDC_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 CDC 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_cdc_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_CDC_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;

    CLD_Serial_Data_Bulk_Endpoint_Params * p_serial_data_rx_endpoint_params;
    CLD_Serial_Data_Bulk_Endpoint_Params * p_serial_data_tx_endpoint_params;
    CLD_CDC_Notification_Endpoint_Params * p_notification_endpoint_params;

    CLD_USB_Transfer_Request_Return_Type (*fp_serial_data_received)
                    (CLD_USB_Transfer_Params * p_transfer_data);

    CLD_USB_Transfer_Request_Return_Type (*fp_cdc_cmd_send_encapsulated_cmd)
                    (CLD_USB_Transfer_Params * p_transfer_data);

    CLD_USB_Transfer_Request_Return_Type (*fp_cdc_cmd_get_encapsulated_resp)
                    (CLD_USB_Transfer_Params * p_transfer_data);

    CLD_USB_Data_Received_Return_Type (*fp_cdc_cmd_set_line_coding)
                    (CLD_CDC_Line_Coding * p_line_coding);

    CLD_RV (*fp_cdc_cmd_get_line_coding) (CLD_CDC_Line_Coding * p_line_coding);

    CLD_USB_Data_Received_Return_Type (*fp_cdc_cmd_set_control_line_state)
                    (CLD_CDC_Control_Line_State * p_control_line_state);

    CLD_USB_Data_Received_Return_Type (*fp_cdc_cmd_send_break)
                    (unsigned short duration);

    unsigned char usb_bus_max_power;
    unsigned char support_cdc_network_notification;
    unsigned short cdc_class_bcd_version;
    unsigned char  cdc_class_control_protocol_code;
    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_communication_class_interface;
    const char * p_usb_string_data_class_interface;

    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_CDC_Lib_Init_Params;
```

A description of the CLD_BF70x_CDC_Lib_Init_Params structure elements is included below:

Structure Element	Description
uart_num	Identifies which of the ADSP-BF707 UARTs should be used by the CLD BF70x CDC 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 CLD_UART_DISABLE If uart_num is set to CLD_UART_DISABLE the CLD BF70x CDC Library will not use a UART, and the cld_console functionality is disabled.
uart_baud	Sets the desired UART baud rate used for the cld_console. The remaining cld_console UART parameters are as follows:

<table>
<tr><td></td><td>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 CDC Library the frequency of the ADSP_BF707 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.
Note: Set to CLD_NULL if not required by application</td></tr>
<tr><td>vendor_id</td><td>The 16-bit USB vendor ID 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 returned to the USB Host in the USB Device Descriptor.</td></tr>
<tr><td>p_serial_data_rx_endpoint_params</td><td>Pointer to a CLD_Serial_Data_Bulk_Endpoint_Params structure that describes how the Bulk OUT endpoint should be configured. The CLD_Serial_Data_Bulk_Endpoint_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 Bulk endpoint. The endpoint number must be within the following range:
1 ≤ endpoint_num ≤ 12. Any other endpoint number will result in the cld_bf70x_cdc_lib_init function returning CLD_FAIL</td></tr>
<tr><td>max_packet_size_full_speed</td><td>Sets the Bulk endpoint's max packet size when operating at Full Speed. The valid Bulk endpoint max packet sizes are as follows:
8, 16, 32, and 64 bytes.</td></tr>
<tr><td>max_packet_size_high_speed</td><td>Sets the Bulk endpoint's max packet size when operating at High Speed. The valid Bulk endpoint max packet sizes are as follows:
8, 16, 32, 64 and 512 bytes.</td></tr>
</table></td></tr>
<tr><td>p_serial_data_tx_endpoint_params</td><td>Pointer to a CLD_Serial_Data_Bulk_Endpoint_Params structure that describes how the Bulk IN endpoint should be configured. The CLD_Serial_Data_Bulk_Endpoint_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</td></tr>
</table></td></tr>
</table>

<table>
<tr>
<td></td>
<td>
<table>
<tr><td></td><td>of the Bulk endpoint. The endpoint number must be within the following range:
1 ≤ endpoint_num ≤ 12. Any other endpoint number will result in the cld_bf70x_cdc_lib_init function returning CLD_FAIL</td></tr>
<tr><td>max_packet_size_full_speed</td><td>Sets the Bulk endpoint's max packet size when operating at Full Speed. The valid Bulk endpoint max packet sizes are as follows:
8, 16, 32, and 64 bytes.</td></tr>
<tr><td>max_packet_size_high_speed</td><td>Sets the Bulk endpoint's max packet size when operating at High Speed. The valid Bulk endpoint max packet sizes are as follows:
8, 16, 32, 64 and 512 bytes.</td></tr>
</table>
</td>
</tr>
<tr>
<td>p_notification_endpoint_params</td>
<td>Pointer to a CLD_CDC_Notification_Endpoint_Params structure that describes how the Interrupt IN endpoint should be configured. The CLD_CDC_Notification_Endpoint_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 Interrupt endpoint. The endpoint number must be within the following range:
1 ≤ endpoint_num ≤ 12. Any other endpoint number will result in the cld_bf70x_cdc_lib_init function returning CLD_FAIL</td></tr>
<tr><td>max_packet_size_full_speed</td><td>Sets the Interrupt endpoint's max packet size when operating at Full Speed. The maximum max packet size is 64 bytes.</td></tr>
<tr><td>polling_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>max_packet_size_high_speed</td><td>Sets the Interrupt endpoint's max packet size when operating at High Speed. The maximum max packet</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>
<table>
<tr><td></td><td>size
1024 bytes.</td></tr>
<tr><td>polling_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>*fp_serial_data_received*</td>
<td>
Pointer to the function that is called when the Bulk 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 Bulk OUT transfer:
<table>
<tr><td>Structure Element</td><td>Description</td></tr>
<tr><td>num_bytes</td><td>The number of bytes to transfer to the p_data_buffer before calling the fp_usb_out_transfer_complete callback function.

When the fp_serial_data_received function is called num_bytes is set the number of bytes in the current Bulk OUT packet. If the Bulk OUT total transfer size is known num_bytes can be set to the transfer size, and the CLD BF70x CDC Library will complete the entire bulk transfer without calling fp_serial_data_received again. If num_bytes isn't modified the fp_serial_data_received function will be called for each Bulk OUT packet.</td></tr>
<tr><td>p_data_buffer</td><td>Pointer to the data buffer to store the received Bulk OUT data. The size of the buffer should be greater than or equal to the value in num_bytes.</td></tr>
<tr><td>*fp_usb_out_transfer_complete*</td><td>Function called when num_bytes of data has been transferred to the p_data_buffer memory.</td></tr>
<tr><td>*fp_transfer_aborted_callback*</td><td>Function called if there is a</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>
<table>
<tr><td></td><td>problem transferring the requested Bulk OUT data.</td></tr>
<tr><td>transfer_timeout_ms</td><td>Bulk OUT transfer timeout in milliseconds. If the Bulk 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</td></tr>
</table>
The fp_serial_data_received function returns the CLD_USB_Transfer_Request_Return_Type, which has the following values:
<table>
<tr><td>Return Value</td><td>Description</td></tr>
<tr><td>CLD_USB_TRANSFER_ACCEPT</td><td>Notifies the CLD BF70x CDC Library that the Bulk 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 CDC Library pause the current transfer. This causes the Bulk OUT endpoint to be nak'ed until the transfer is resumed by calling cld_bf70x_cdc_lib_resume_paused_serial_data_transfer.</td></tr>
<tr><td>CLD_USB_TRANSFER_DISCARD</td><td>Requests that the CLD BF70x CDC Library discard the number f bytes specified in p_transfer_params-> num_bytes. In this case the library accepts the Bulk OUT data from the USB Host but discards the data. This is similar to the concepts of frame dropping in audio/video applications.</td></tr>
<tr><td>CLD_USB_TRANSFER_STALL</td><td>This notifies the CLD BF70x CDC Library that there is an error and the Bulk OUT endpoint should be stalled.</td></tr>
</table>
</td>
</tr>
<tr>
<td><i>fp_cdc_cmd_send_encapsulated_cmd</i></td>
<td>Pointer to the function that is called when a CDC Send Encapsulated Command request is received. This function a pointer to the CLD_USB_Transfer_Params structure ('p_transfer_data') as its parameters.

The following CLD_USB_Transfer_Params structure elements are used to processed a Send Encapsulated Command transfer:</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 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 Send Encapsulated Command 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_cdc_cmd_send_encapsulated_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 CDC Library that the Send Encapsulated Command data should be accepted using the p_transfer_data values.
CLD_USB_TRANSFER_PAUSE	Requests that the CLD BF70x CDC Library pause the Set Report transfer. This causes the Control Endpoint to be nak'ed until the transfer is resumed by calling cld_bf70x_cdc_lib_resume_paused_control_transfer.
CLD_USB_TRANSFER_DISCARD	Requests that the CLD BF70x CDC Library discard the number of bytes specified in

<table>
<tr>
<td></td>
<td>
<table>
<tr><td></td><td>p_transfer_params-> num_bytes. In this case the library accepts the Send Encapsulated Command from the USB Host but discards the data. This is similar to the concepts of frame dropping in audio/video applications.</td></tr>
<tr><td>CLD_USB_TRANSFER_STALL</td><td>This notifies the CLD BF70x CDC Library that there is an error and the request should be stalled.</td></tr>
</table>
</td>
</tr>
<tr>
<td>*fp_cdc_cmd_get_encapsulated_resp*</td>
<td>
Pointer to the function that is called when a CDC Get Encapsulated Response request is received. This function takes a pointer to the CLD_USB_Transfer_Params structure ('p_transfer_data') as its parameters.

The following CLD_USB_Transfer_Params structure elements are used to processed a Get Encapsulated Response request:
<table>
<tr><td>Structure Element</td><td>Description</td></tr>
<tr><td>num_bytes</td><td>The number of bytes from the Setup Packet wLength field.</td></tr>
<tr><td>p_data_buffer</td><td>Pointer to the data buffer to source the Get Encapsulated Response data. The size of the buffer should be greater than or equal to the value in num_bytes.</td></tr>
<tr><td>*fp_usb_in_transfer_complete*</td><td>Function called when Get Encapsulated Response data has been transferred to the Host.</td></tr>
<tr><td>*fp_transfer_aborted_callback*</td><td>Function called if there is a problem transferring the data, or if the transfer is interrupted</td></tr>
<tr><td>transfer_timeout_ms</td><td>Not used for Control Requests since the Host has the ability to interrupt any Control transfer.</td></tr>
</table>
The fp_cdc_cmd_get_encapsulated_resp function returns the CLD_USB_Transfer_Request_Return_Type, which has the following values:
<table>
<tr><td>Return Value</td><td>Description</td></tr>
<tr><td>CLD_USB_TRANSFER_ACCEPT</td><td>Notifies the CLD BF70x CDC Library that the Get</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>
<table>
<tr><td></td><td>Encapsulated Response data should be transferred using the p_transfer_data values.</td></tr>
<tr><td>CLD_USB_TRANSFER_PAUSE</td><td>Requests that the CLD BF70x CDC Library pause the Get Encapsulated Response transfer. This causes the Control Endpoint to be nak'ed until the transfer is resumed by calling cld_bf70x_cdc_lib_resume_paused_control_transfer.</td></tr>
<tr><td>CLD_USB_TRANSFER_DISCARD</td><td>Requests that the CLD BF70x CDC Library to return a zero length packet in response to the Get Encapsulated Response request.</td></tr>
<tr><td>CLD_USB_TRANSFER_STALL</td><td>This notifies the CLD BF70x CDC Library that there is an error and the request should be stalled.</td></tr>
</table>
</td>
</tr>
<tr>
<td><i>fp_cdc_cmd_set_line_coding</i></td>
<td>
Pointer to the function that is called when a CDC Set Line Coding request is received. This function takes a pointer to the Host specified CLD_CDC_Line_Coding structure ('p_line_coding') as its parameters.

The following CLD_CDC_Line_Coding structure elements are used to processed a Set Line Coding request:
<table>
<tr><td>Structure Element</td><td>Description</td></tr>
<tr><td>data_terminal_rate</td><td>Serial baud rate in bits per second.</td></tr>
<tr><td>num_stop_bits</td><td>CDC Number of stop bits.
0 = 1 stop bit
1 = 1.5 stop bits
2 = 2 stop bits.</td></tr>
<tr><td>parity</td><td>CDC parity setting
0 = None
1 = Odd
2 = Even
3 = Mark
4 = Space</td></tr>
<tr><td>num_data_bits</td><td>CDC Number of data bits (only 5, 6, 7, 8 and 16 are valid).</td></tr>
</table>

The fp_cdc_cmd_set_line_coding function returns the CLD_USB_Data_Received_Return_Type, which has the following values:
<table>
<tr><td>Return Value</td><td>Description</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>
<table>
<tr><td>CLD_USB_DATA_GOOD</td><td>Notifies the CLD BF70x CDC Library that the request is valid.</td></tr>
<tr><td>CLD_USB_DATA_BAD_STALL</td><td>Notifies the CLD BF70x CDC Library that the request is invalid, and should be stalled.</td></tr>
</table>
</td>
</tr>
<tr>
<td><i>fp_cdc_cmd_get_line_coding</i></td>
<td>
Pointer to the function that is called when a CDC Get Line Coding request is received. This function takes a pointer to CLD_CDC_Line_Coding structure ('p_line_coding') as its parameters. The User firmware should set the p_line_coding structure values based on its active settings.

The following CLD_CDC_Line_Coding structure elements are used to processed a Get Line Coding request:
<table>
<tr><td>Structure Element</td><td>Description</td></tr>
<tr><td>data_terminal_rate</td><td>Serial baud rate in bits per second.</td></tr>
<tr><td>num_stop_bits</td><td>CDC Number of stop bits.
0 = 1 stop bit
1 = 1.5 stop bits
2 = 2 stop bits.</td></tr>
<tr><td>parity</td><td>CDC parity setting
0 = None
1 = Odd
2 = Even
3 = Mark
4 = Space</td></tr>
<tr><td>num_data_bits</td><td>CDC Number of data bits (only 5, 6, 7, 8 and 16 are valid).</td></tr>
</table>

The fp_cdc_cmd_get_line_coding function returns CLD_RV, which has the following values:
<table>
<tr><td>Return Value</td><td>Description</td></tr>
<tr><td>CLD_SUCCESS</td><td>Notifies the CLD BF70x CDC Library that the request is valid and the p_line_coding value should be returned to the Host.</td></tr>
<tr><td>CLD_FAIL</td><td>Notifies the CLD BF70x CDC Library that the request is invalid, and should be stalled.</td></tr>
</table>
</td>
</tr>
<tr>
<td><i>fp_cdc_cmd_set_control_line_state</i></td>
<td>
Pointer to the function that is called when a CDC Set Control Line State request is received. This function takes a pointer to the Host specified CLD_CDC_Control_Line_State structure ('p_control_line_state') as its parameters.

The following CLD_CDC_Control_Line_State structure
</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>elements are used to processed a Set Control Line State request:
<table>
<tr><th>Structure Element</th><th>Description</th></tr>
<tr><td>dte_present</td><td>Controls if the DTE is present or not. This corresponds to the RS-232 DTR signal.
0 = Not Present
1 = Present</td></tr>
<tr><td>activate_carrier</td><td>Carrier control used in half duplex serial links. This signal corresponds to the RS-232 RTS signal.
0 = Disabled
1 = Active</td></tr>
</table>

The fp_cdc_cmd_set_control_line_state function returns the CLD_USB_Data_Received_Return_Type, which has the following values:
<table>
<tr><th>Return Value</th><th>Description</th></tr>
<tr><td>CLD_USB_DATA_GOOD</td><td>Notifies the CLD BF70x CDC Library that the request is valid.</td></tr>
<tr><td>CLD_USB_DATA_BAD_STALL</td><td>Notifies the CLD BF70x CDC Library that the request is invalid, and should be stalled.</td></tr>
</table>
</td>
</tr>
<tr>
<td><i>fp_cdc_cmd_send_break</i></td>
<td>Pointer to the function that is called when a CDC Send Break request is received. This function takes the host specified duration in milliseconds ('duration') as its parameters.

The fp_cdc_cmd_send_break function returns the CLD_USB_Data_Received_Return_Type, which has the following values:
<table>
<tr><th>Return Value</th><th>Description</th></tr>
<tr><td>CLD_USB_DATA_GOOD</td><td>Notifies the CLD BF70x CDC Library that the request is valid.</td></tr>
<tr><td>CLD_USB_DATA_BAD_STALL</td><td>Notifies the CLD BF70x CDC Library that the request is invalid, and should be stalled.</td></tr>
</table>
</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>support_cdc_network_connection</td>
<td>Tells the CLD BF70x CDC Library if the User firmware supports the CDC Network Connection Notification.
0 = Not supported
1 = Supported</td>
</tr>
<tr>
<td>cdc_class_bcd_version</td>
<td>CDC Class Version in BCD. Returned in the CDC Header Functional Descriptor's bcdCDC field. (refer to the CDC specification v1.2 section 5.3.2.1).</td>
</tr>
</table>

<table>
<tr><td>cdc_class_control_protocol_code</td><td>Value used in the CDC interface descriptor's bInterfaceProtocol field. The valid CDC Protocol codes are defined in the CDC v.1.2 specification in Table 5 on page 13.</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>p_usb_string_manufacturer</td><td>Pointer to the null-terminated string. This string is used by the CLD BF70x CDC Library to generate the Manufacturer USB String Descriptor. If the Manufacturer String Descriptor is not used set p_usb_string_manufacturer to CLD_NULL.</td></tr>
<tr><td>p_usb_string_product</td><td>Pointer to the null-terminated string. This string is used by the CLD BF70x CDC Library to generate the Product USB String Descriptor. If the Product String Descriptor is not used set p_usb_string_product to CLD_NULL.</td></tr>
<tr><td>p_usb_string_serial_number</td><td>Pointer to the null-terminated string. This string is used by the CLD BF70x CDC 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.</td></tr>
<tr><td>p_usb_string_configuration</td><td>Pointer to the null-terminated string. This string is used by the CLD BF70x CDC Library to generate the Configuration USB String Descriptor. If the Configuration String Descriptor is not used set p_usb_string_configuration to CLD_NULL.</td></tr>
<tr><td>p_usb_string_communication_class_interface</td><td>Pointer to the null-terminated string. This string is used by the CLD BF70x CDC Library to generate the CDC Interface USB String Descriptor. If the CDC Interface String Descriptor is not used set p_usb_string_communication_class_interface to CLD_NULL.</td></tr>
<tr><td>p_usb_string_data_class_interface</td><td>Pointer to the null-terminated string. This string is used by the CLD BF70x CDC Library to generate the Data Class Interface USB String Descriptor. If the Data Interface String Descriptor is not used set p_usb_string_data_class_interface to CLD_NULL.</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><i>CLD_USB_ENUMERATED_CONFIGURED_HS</i></td><td>USB device enumerated at High-Speed(USB Configuration set to a non-zer</td></tr>
</table>
</td></tr>
</table>

<table>
<tr>
<td></td>
<td>
<table>
<tr><td></td><td>value)</td></tr>
<tr><td><em>CLD_USB_ENUMERATED_CONFIGURED_FS</em></td><td>USB device enumerated at Full-Speed(USB Configuration set to a non-zero value)</td></tr>
<tr><td><em>CLD_USB_UN_CONFIGURED</em></td><td>USB Configuration set to 0</td></tr>
<tr><td><em>CLD_USB_BUS_RESET</em></td><td>USB Bus reset received</td></tr>
<tr><td><em>CLD_USB_BUS_SUSPEND</em></td><td>USB Suspend detected</td></tr>
<tr><td><em>CLD_USB_BUS_RESUME</em></td><td>USB Resume detected</td></tr>
</table>

<b>Note:</b> Set to CLD_NULL if not required by application
</td>
</tr>
<tr>
<td><em>fp_cld_lib_status</em></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>
<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_bf70x_cdc_lib_main

```
void cld_bf70x_cdc_lib_main (void)
```

CLD BF70x CDC Library mainline function

Arguments

None

Return Value

None.

Details

The cld_bf70x_cdc_lib_main function is the CLD BF70x CDC Library mainline function which must be called in every iteration of the main program loop in order for the library to function properly.

cld_bf70x_cdc_lib_transmit_serial_data

```
CLD_USB_Data_Transmit_Return_Type cld_bf70x_cdc_lib_transmit_serial_data
      (CLD_USB_Transfer_Params * p_transfer_data)
```

CLD BF70x CDC Library function used to send serial over the Bulk 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 Bulk 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 Bulk IN transfer.
CLD_USB_TRANSMIT_FAILED	The library failed to start the requested Bulk IN transfer. This will happen if the Bulk IN endpoint is busy, or if the p_transfer_data-> data_buffer is set to NULL

Details

The cld_bf70x_cdc_lib_transmit_serial_data function transmits the data specified by the p_transfer_data parameter to the USB Host using the Device's Bulk 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);
    void 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 have been transmitted the 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 Bulk 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	USB transfer timeout in milliseconds. If the Bulk 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_bf70x_cdc_lib_send_network_connection_state

```
CLD_USB_Data_Transmit_Return_Type cld_bf70x_cdc_lib_send_network_connection_state
      (CLD_BF70x_CDC_Lib_Network_Connection_State state)
```

CLD BF70x CDC Library function used to send the CDC Network Connection Notification using the Interrupt IN endpoint.

Arguments

state	The Network Connection state to send to the Host.

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 busy, or if the p_transfer_data->data_buffer is set to NULL

Details

The cld_bf70x_cdc_lib_send_network_connection_state function transmits the network connection state specified by the state parameter to the USB Host using the Device's Interrupt IN endpoint.

The CLD_BF70x_CDC_Lib_Network_Connection_State enum values are listed below.

Enum Element	Description
CLD_CDC_NETWORK_DISCONNECTED	The CDC Network is disconnected.
CLD_CDC_NETWORK_CONNECTED	The CDC Network is connected.

cld_bf70x_cdc_lib_send_response_available

```
CLD_USB_Data_Transmit_Return_Type cld_bf70x_cdc_lib_send_response_available
       (CLD_BF70x_CDC_Lib_Network_Connection_State state)
```

CLD BF70x CDC Library function used to send the CDC Response Available Notification using the Interrupt IN endpoint.

Arguments

None.

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 busy, or if the p_transfer_data->data_buffer is set to NULL

Details

The cld_bf70x_cdc_lib_send_response_available function transmits the CDC Response Available Notification to the USB Host using the Device's Interrupt IN endpoint. The Host can then request the response data using a Send Encapsulated Response Control endpoint request.

cld_bf70x_cdc_lib_send_serial_state

```
CLD_USB_Data_Transmit_Return_Type cld_bf70x_cdc_lib_send_serial_state
      (CLD_CDC_Serial_State * p_serial_state)
```

CLD BF70x CDC Library function used to send the CDC Serial State Notification using the Interrupt IN endpoint.

Arguments

p_serial_state	Pointer to a CLD_CDC_Serial_State structure used to report the current state of the emulated serial port to the USB Host.

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 busy, or if the p_transfer_data-> data_buffer is set to NULL

Details

The cld_bf70x_cdc_lib_send_serial_data function transmits the current CDC Serial State specified by the p_serial_state parameter to the USB Host using the Device's Interrupt IN endpoint.

The CLD CLD_CDC_Serial_State structure is described below.

```
typedef struct
{
    union
    {
        struct
        {
            unsigned short rx_carrier       : 1;
            unsigned short tx_carrier       : 1;
            unsigned short break_detect     : 1;
            unsigned short ring_signal      : 1;
            unsigned short framing_error    : 1;
            unsigned short parity_error     : 1;
            unsigned short rx_data_overrun  : 1;
            unsigned short reserved         : 9;
        } bits;
        unsigned short state;
    } u;
} CLD_CDC_Serial_State;
```

A description of the CLD_CDC_Serial_State structure elements is included below:

Structure Element	Description
rx_carrier	State of receiver carrier detection mechanism of device. This signal corresponds to V.24 signal 109 and RS-232 signal DCD.
tx_carrier	State of transmission carrier. This signal corresponds to V.24 signal 106 and RS-232 signal DSR.
break_detect	State of break detection mechanism of the device.
ring_signal	State of ring signal detection of the device.
framing_error	A framing error has occurred.
parity_error	A parity error has occurred.
rx_data_overrun	Received data has been discarded due to overrun in the device.

Once the Serial State Notification has been sent the device re-evaluates the above fields. For the tx_carrier and rx_carrier the Serial State Notification is sent when these signals change. For the remaining fields once the Serial State Notification has been sent their value is reset to zero, and will be sent to the Host again when the field is set to a '1'.

cld_bf70x_cdc_lib_resume_paused_serial_data_transfer

```
void cld_bf70x_cdc_lib_paused_resume_serial_data_transfer (void)
```

CLD BF70x CDC Library function used to resume a paused Serial Data Bulk OUT transfer.

Arguments

None

Return Value

None.

Details

The cld_bf70x_cdc_lib_resume_paused_serial_data_transfer function is used to resume a Bulk OUT transfer that was paused by the `fp_serial_data_received` function returning CLD_USB_TRANSFER_PAUSE. When called the cld_bf70x_cdc_lib_resume_paused_serial_data_transfer function will call the User application's `fp_serial_data_received` function passing the CLD_USB_Transfer_Params of the original paused transfer. The `fp_serial_data_received` function can then chose to accept, discard, or stall the Bulk OUT request.

cld_bf70x_cdc_lib_resume_paused_control_transfer

```
void cld_bf70x_cdc_lib_resume_paused_control_transfer (void)
```

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

Arguments

None

Return Value

None.

Details

The cld_bf70x_cdc_lib_resume_paused_control_transfer function is used to resume a Control transfer that was paused by the `fp_cdc_cmd_send_encapsulated_cmd` or `fp_cdc_cmd_get_encapsulated_resp` function returning CLD_USB_TRANSFER_PAUSE. When called the cld_bf70x_cdc_lib_resume_paused_control_transfer function will call the User application's `fp_cdc_cmd_send_encapsulated_cmd` or `fp_cdc_cmd_get_encapsulated_resp` 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 CDC 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 CDC Library has been initialized to connect the USB device to the Host.

cld_lib_usb_disconnect

```
void cld_lib_usb_disconnect (void)
```

CLD BF70x CDC 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 CDC 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 CDC 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 CDC 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.

If a one millisecond resolution is granular enough for your needs, you can have a virtually unlimited number of timed events when using cld_time_get and cld_time_passed_ms.

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.

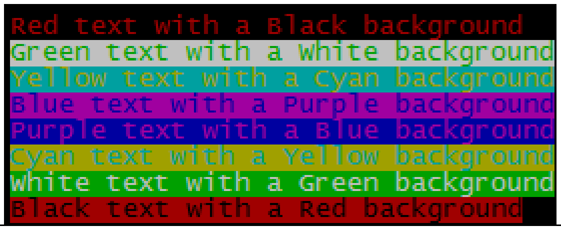
cld_console

```
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_CDC_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. A feature of cld_console is that it is non-blocking, i.e. long messages can be queued and the function call returns prior to the message draining from the buffer. Overly long messages are truncated to 128 bytes, and up to 1024 characters can be in escrow to be transmitted. Received characters can be processed by supplying a console_rx_byte function in the library init structure.

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

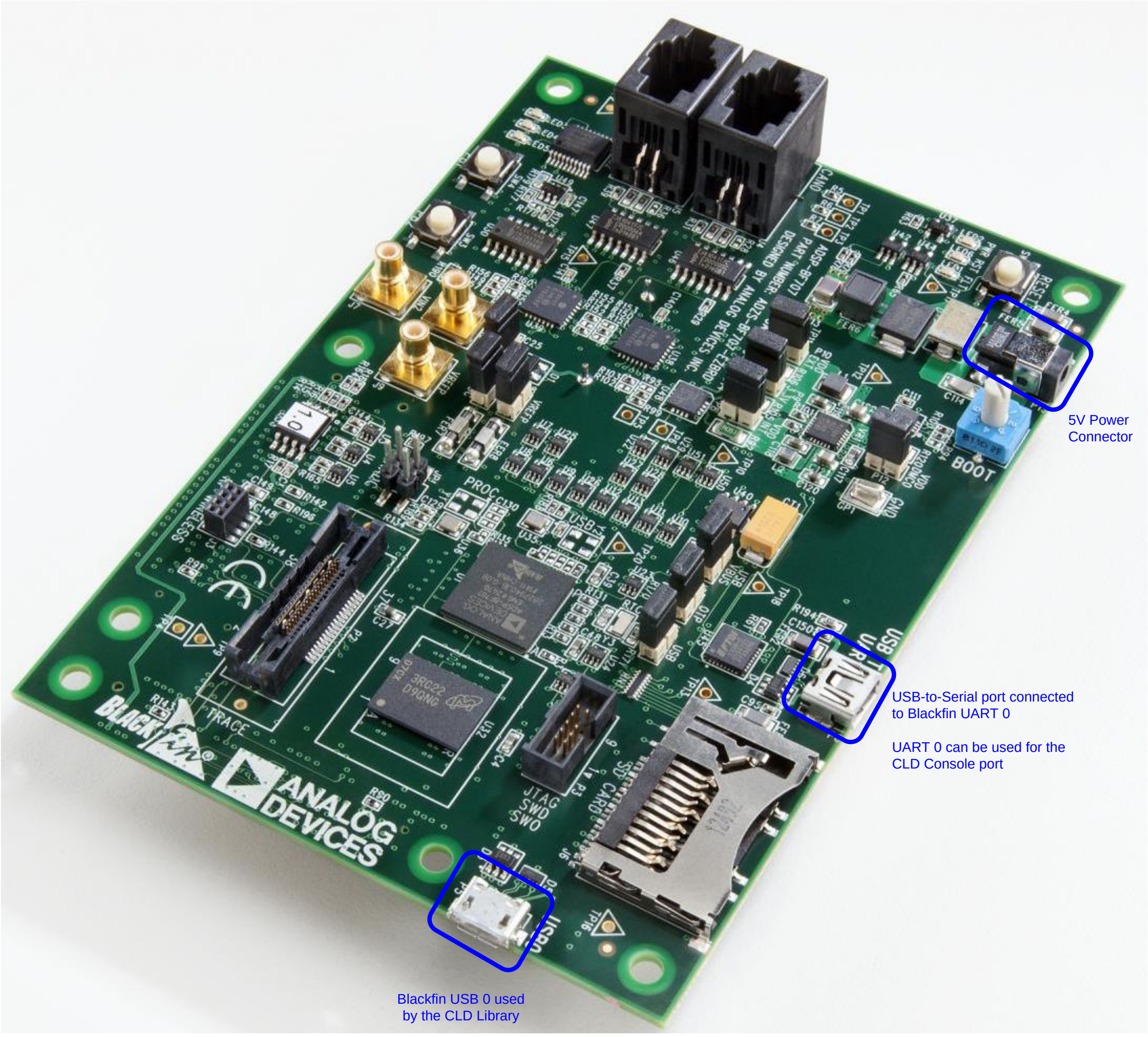
This function returns a decoded Null terminated ASCII string.

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:



5V Power Connector

USB-to-Serial port connected to Blackfin UART 0

UART 0 can be used for the CLD Console port

Blackfin USB 0 used by the CLD Library

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 CDC 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 CDC Library console connected to UART0.

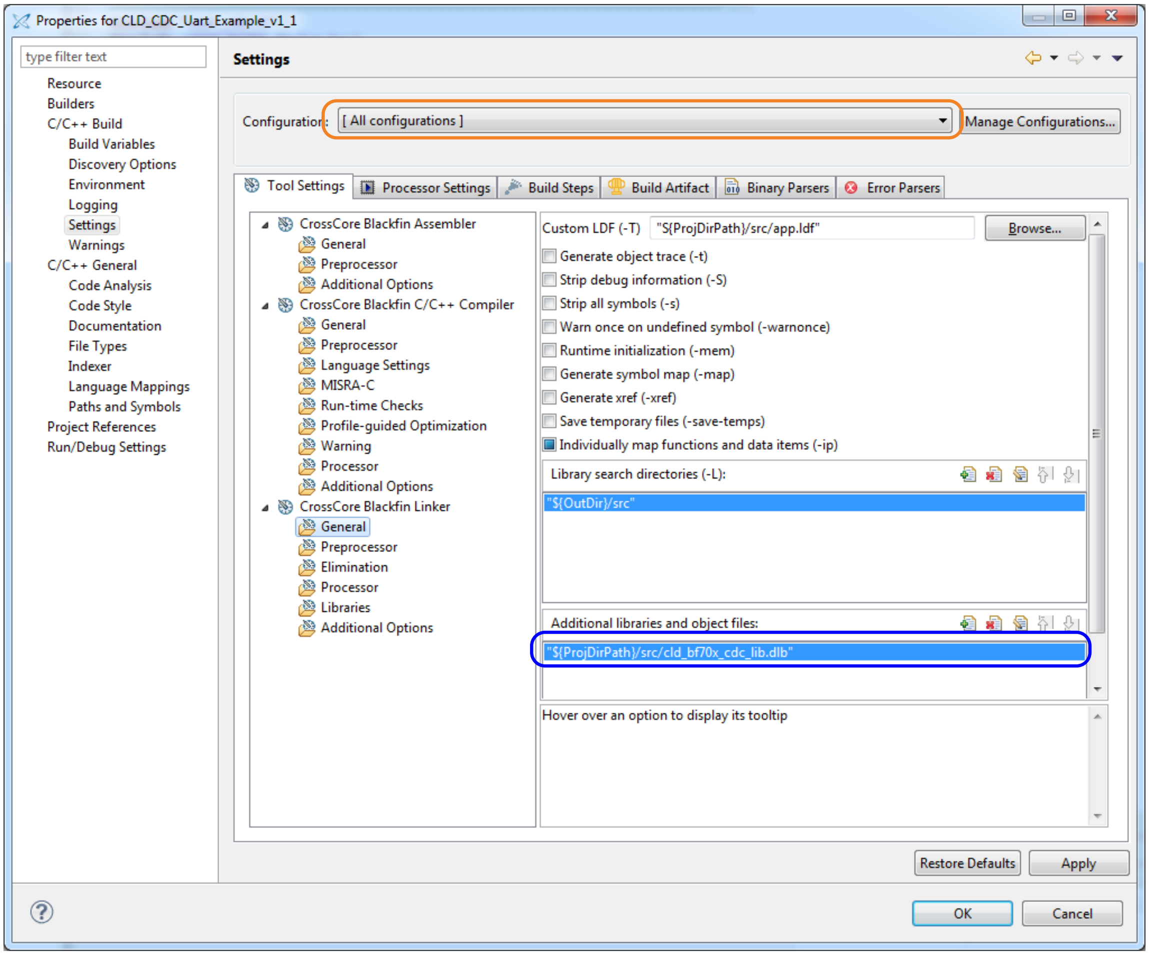
Adding the CLD BF70x CDC Library to an Existing CrossCore Embedded Studio Project

In order to include the CLD BF70x CDC 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_cdc_lib.h and cld_bf70x_cdc_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 CDC Library needs to be included in the project's 'Additional libraries and object files' as shown in the diagram below (circled in blue). This lets the linker know where the cld_bf70x_cdc_lib.dlb file is located.



Properties for CLD_CDC_Uart_Example_v1_1

Settings

Configuration: [All configurations]

Custom LDF (-T) "${ProjDirPath}/src/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}/src"

Additional libraries and object files:
"${ProjDirPath}/src/cld_bf70x_cdc_lib.dlb"

Hover over an option to display its tooltip

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).

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 CDC Library please refer to the CLD CDC UART Example v2.0 project included with the CLD BF70x CDC Library. The CLD CDC Uart Example v2.0 project implements a basic USB to Serial device using the CDC Abstract Model Class Serial Emulation protocol.

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_cdc_init();
                if (rv == USER_CDC_INIT_SUCCESS)
                {
                    main_state = MAIN_STATE_RUN;
                }
                else if (rv == USER_CDC_INIT_FAILED)
                {
                    main_state = MAIN_STATE_ERROR;
                }
            break;

            case MAIN_STATE_RUN:
                 user_cdc_main();
            break;

            case MAIN_STATE_ERROR:

            break;
        }
    }
}
```

user_cdc.c

```
/* CDC Notification Interrupt IN endpoint parameters. */
static CLD_BF70x_CDC_Notification_Endpoint_Params user_cdc_notification_ep_params =
{
    .endpoint_number                = 1,
    .max_packet_size_full_speed     = 64,
    .polling_interval_full_speed    = 1,
    .max_packet_size_high_speed     = 64,
    .polling_interval_high_speed    = 4,   /* 1ms */
};

/* CDC Serial Data Bulk OUT endpoint parameters. */
static CLD_Serial_Data_Bulk_Endpoint_Params user_cdc_serial_data_rx_ep_params =
{
    .endpoint_number                = 2,
    .max_packet_size_full_speed     = 64,
    .max_packet_size_high_speed     = 512,
};

/* CDC Serial Data Bulk IN endpoint parameters. */
static CLD_Serial_Data_Bulk_Endpoint_Params user_cdc_serial_data_tx_ep_params =
{
    .endpoint_number                = 2,
    .max_packet_size_full_speed     = 64,
    .max_packet_size_high_speed     = 512,
};



/* CLD BF70x CDC Library initialization data. */
static CLD_BF70x_CDC_Lib_Init_Params user_cdc_init_params =
{
    .uart_num        = CLD_UART_0,
    .uart_baud       = 115200,
    .sclk0           = 100000000u,
    .fp_console_rx_byte = user_cdc_console_rx_byte,
    .vendor_id       = 0x064b,
    .product_id      = 0x0003,

    /* Pointer to the serial data rx bulk endpoint parameters. */
    .p_serial_data_rx_endpoint_params = &user_cdc_serial_data_rx_ep_params,
    /* Pointer to the serial data tx bulk endpoint parameters.  */
    .p_serial_data_tx_endpoint_params = &user_cdc_serial_data_tx_ep_params,
    /* Pointer to the CDC notification endpoint parameters. */
    .p_notification_endpoint_params = &user_cdc_notification_ep_params,

    /* Function called when serial data is received. */
    .fp_serial_data_received = user_cdc_serial_data_received,
    /* Function called when a CDC Send Encapsulated Command request is received */
    .fp_cdc_cmd_send_encapsulated_cmd   = user_cdc_cmd_send_encapsulated_cmd,
    /* Function called when a CDC Get Encapsulated Command request is received */
    .fp_cdc_cmd_get_encapsulated_resp   = user_cdc_cmd_get_encapsulated_resp,
    /* Function called when a CDC Set Line Coding request is received */
    .fp_cdc_cmd_set_line_coding         = user_cdc_cmd_set_line_coding,
    /* Function called when a CDC Get Line Coding request is received */
    .fp_cdc_cmd_get_line_coding         = user_cdc_cmd_get_line_coding,
    /* Function called when a CDC Set Control Line request is received */
    .fp_cdc_cmd_set_control_line_state  = user_cdc_cmd_set_control_line_state,
    /* Function called when a CDC Send Break request is received */
    .fp_cdc_cmd_send_break              = user_cdc_cmd_send_break,
    .usb_bus_max_power = 0,

    .support_cdc_network_notification   = 1
    .cdc_class_bcd_version              = 0x0120,       /* CDC Version 1.2 */
    .cdc_class_control_protocol_code    = 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       = "Example CDC",
    .p_usb_string_serial_number = CLD_NULL,
    .p_usb_string_configuration = CLD_NULL,
    .p_usb_string_communication_class_interface = "BF707 CDC Interface",
    .p_usb_string_data_class_interface = "BF707 CDC Data",


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

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

```c
typedef enum
{
    USER_CDC_INIT_SUCCESS = 0,
    USER_CDC_INIT_ONGOING,
    USER_CDC_INIT_FAILED,
} User_CDC_Init_Return_Code;

User_CDC_Init_Return_Code user_cdc_init (void)
{
    static unsigned char user_init_state = 0;
    CLD_RV cld_rv = CLD_ONGOING;
    User_CDC_Init_Return_Code init_return_code = USER_CDC_INIT_ONGOING;

    switch (user_init_state)
    {
        case 0:

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

            user_init_state++;
        break;
        case 1:
            /* Initialize the CLD BF70x CDC Library */
            cld_rv = cld_bf70x_cdc_lib_init(&user_cdc_init_params);

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

                init_return_code = USER_CDC_INIT_SUCCESS;
            }
            else if (cld_rv == CLD_FAIL)
            {
                init_return_code = USER_CDC_INIT_FAILED;
            }
            else
            {
                init_return_code = USER_CDC_INIT_ONGOING;
            }
    }
    return init_return_code;
}

void user_cdc_main (void)
{
    cld_bf70x_cdc_lib_main();

}
```

```
/* Function called when a Serial Data Bulk OUT packet is received */
static CLD_USB_Transfer_Request_Return_Type
      user_cdc_serial_data_received (CLD_USB_Transfer_Params * p_transfer_data)
{
    p_transfer_data->num_bytes = /* TODO: Set number of Bulk OUT bytes to
                                          transfer */
    p_transfer_data->p_data_buffer = /* TODO: address to store Bulk OUT data */

    /* User Interrupt transfer complete callback function. */
    p_transfer_data->callback.usb_out_transfer_complete =
                                        user_cdc_serial_data_out_transfer_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 or 0 to
                                                     disable the timeout. */

    /* TODO: Return how the Bulk OUT transfer should be handled (Accept, Pause,
             Discard, or Stall */
}

/* The function below is an example of the Bulk OUT transfer done callback
   specified in the CLD_USB_Transfer_Params structure. */
static CLD_USB_Data_Received_Return_Type user_cdc_serial_data_out_transfer_done (void)
{
    /* TODO: Process the received Bulk 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)*/
}

/* Function called when a Send Encapsulated Command request is received */
static CLD_USB_Transfer_Request_Return_Type user_cdc_cmd_send_encapsulated_cmd
             (CLD_USB_Transfer_Params * p_transfer_data)
{
    p_transfer_data->p_data_buffer = /* TODO: address to store data */
    p_transfer_data->callback.usb_out_transfer_complete =
                                 user_cdc_send_encapsilated_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 Send Encapsulated Command data is received */
static CLD_USB_Data_Received_Return_Type
       user_cdc_send_encapsilated_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 Encapsulated Response request is received */
static CLD_USB_Transfer_Request_Return_Type user_cdc_cmd_get_encapsulated_resp
              (CLD_USB_Transfer_Params * p_transfer_data)
{
    p_transfer_data->num_bytes = /* TODO: Set to size of response */
    p_transfer_data->p_data_buffer = /* TODO: address to source the response data */
    p_transfer_data->callback.usb_in_transfer_complete =
                                  user_cdc_get_encapsulated_resp_transfer_complete;
    p_transfer_data->fp_transfer_aborted_callback = /* TODO: Set to User callback
                                                            function or NULL */
     /* TODO: Return how the Control transfer should be handled (Accept, Pause,
             Discard, or Stall */
}


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

/* Function called when a Set Line Coding Request has been received*/
CLD_USB_Data_Received_Return_Type user_cdc_cmd_set_line_coding
       (CLD_CDC_Line_Coding * p_line_coding)
{
    if ( /* TODO: Check if CDC Line Coding is valid */ )
    {
        /* TODO: Save the requested CDC Line Coding and process it accordingly */
        return CLD_USB_DATA_GOOD;
    }
    else
    {
        return CLD_USB_DATA_BAD_STALL;
    }
}

/* Function called when a Get Line Coding Request has been received*/
CLD_RV user_cdc_cmd_get_line_coding (CLD_CDC_Line_Coding * p_line_coding)
{
    if ( /* TODO: Check if Get CDC Line Coding request is valid */ )
    {
        /* TODO: Copy the current CDC Line Coding into the p_line_coding structure */
        return CLD_SUCCESS;
    }
    else
    {
        return CLD_FAIL;
    }
}
```

```
/* Function called when a CDC Set Control Line State Request has been received*/
CLD_USB_Data_Received_Return_Type user_cdc_cmd_set_control_line_state
       (CLD_CDC_Control_Line_State * p_control_line_state)
{
    if ( /* TODO: Check if CDC Control Line state is valid */ )
    {
        /* TODO: Process the CDC Control Line State */
        return CLD_USB_DATA_GOOD;
    }
    else
    {
        return CLD_USB_DATA_BAD_STALL;
    }
}

/* Function called when a CDC Send Break Request has been received*/
static void user_cdc_cmd_send_break (unsigned short duration)
{
    /* TODO: Process the requested break duration */
}

static void user_cdc_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;
    }
}


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

```
/* The following function will transmit the specified memory using
   the Serial Data Bulk IN endpoint. */
static void user_cdc_transmit_serial_data_in_data (void)
{
    static CLD_USB_Transfer_Params transfer_params;

    transfer_params.num_bytes = /* TODO: Set number of Bulk IN bytes */
    transfer_params.p_data_buffer = /* TODO: address Bulk IN data */
    transfer_params.callback.usb_in_transfer_complete = /* TODO: Set to User callback
                                                                 function or NULL */;
    transfer_params.callback.fp_transfer_aborted_callback = /* TODO: Set to User
callback                                                                 function or
NULL */;
    transfer_params.callback.transfer_timeout_ms = /* TODO: Set to desired timeout in
                                                     milliseconds or 0 to disable the
                                                     timeout*/;

    if (cld_bf70x_cdc_lib_transmit_serial_data(&transfer_params) ==
             CLD_USB_TRANSMIT_SUCCESSFUL)
    {
        /* Bulk IN transfer initiated successfully */
    }
    else
    {
        /* Bulk IN transfer was unsuccessful */
    }
}


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);
}
```