USB Communication Device Class (CDC) Abstract Control Model Library for Analog Devices ADSP-SC598 User's Guide Revision 1.00

Closed Loop Design, LLC

support@cld-llc.com

Table of Contents

Disclaimer 3

Introduction 3

USB Background 3

CLD Library USB Enumeration Flow Chart 4

CLD Library Interrupt IN Flow Chart 6

CLD CDC Library Bulk OUT Flow Chart 8

CLD CDC Library Bulk IN Flow Chart 10

CDC Abstract Control Model Background 11

CDC Notifications Interrupt IN Endpoint 11

CDC Abstract Control Model Control Endpoint Requests 12

Dependencies 21

CLD SC598 CDC Library Scope and Intended Use 21

CLD CDC Example v1.00 Description 21

Running the Example Project 21

CLD SC598 CDC Library API 23

cld_sc598_cdc_lib_init 23

cld_sc598_cdc_lib_main 34

cld_cdc_lib_receive_serial_data 35

cld_cdc_lib_transmit_serial_data 37

cld_cdc_lib_send_network_connection_state 38

cld_cdc_lib_send_response_available 39

cld_cdc_lib_send_serial_state 40

cld_cdc_lib_resume_paused_control_transfer 41

cld_lib_usb_connect 42

cld_ lib_usb_disconnect 42

cld_time_125us_tick 42

cld_usb0_isr_callback 43

cld_time_get 43

cld_time_passed_ms 44

cld_time_get_125us 44

cld_time_passed_125us 45

cld_lib_status_decode 45

cld_lib_access_usb_phy_reg 46
Adding the CLD SC598 CDC Library to an Existing CrossCore Embedded Studio Project 47
User Firmware Code Snippets 49
main.c 49
user.c 50

Disclaimer

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

Introduction

The Closed Loop Design (CLD) CDC/ACM library creates a simplified interface for developing a USB Communication Device Class (CDC) Abstract Control Model (ACM) Serial Emulation device using the Analog Devices EV-SOMCRR-EZKIT and the EV-SC598-SOM System-on-Module boards. The CLD SC598 CDC library also includes support for timer functions that facilitate creating timed events quickly and easily. The library's User application interface is comprised of parameters used to customize the library's functionality as well as callback functions used to notify the User application of events. These parameters and functions are described in greater detail in the CLD SC598 CDC Library API section of this document.

USB Background

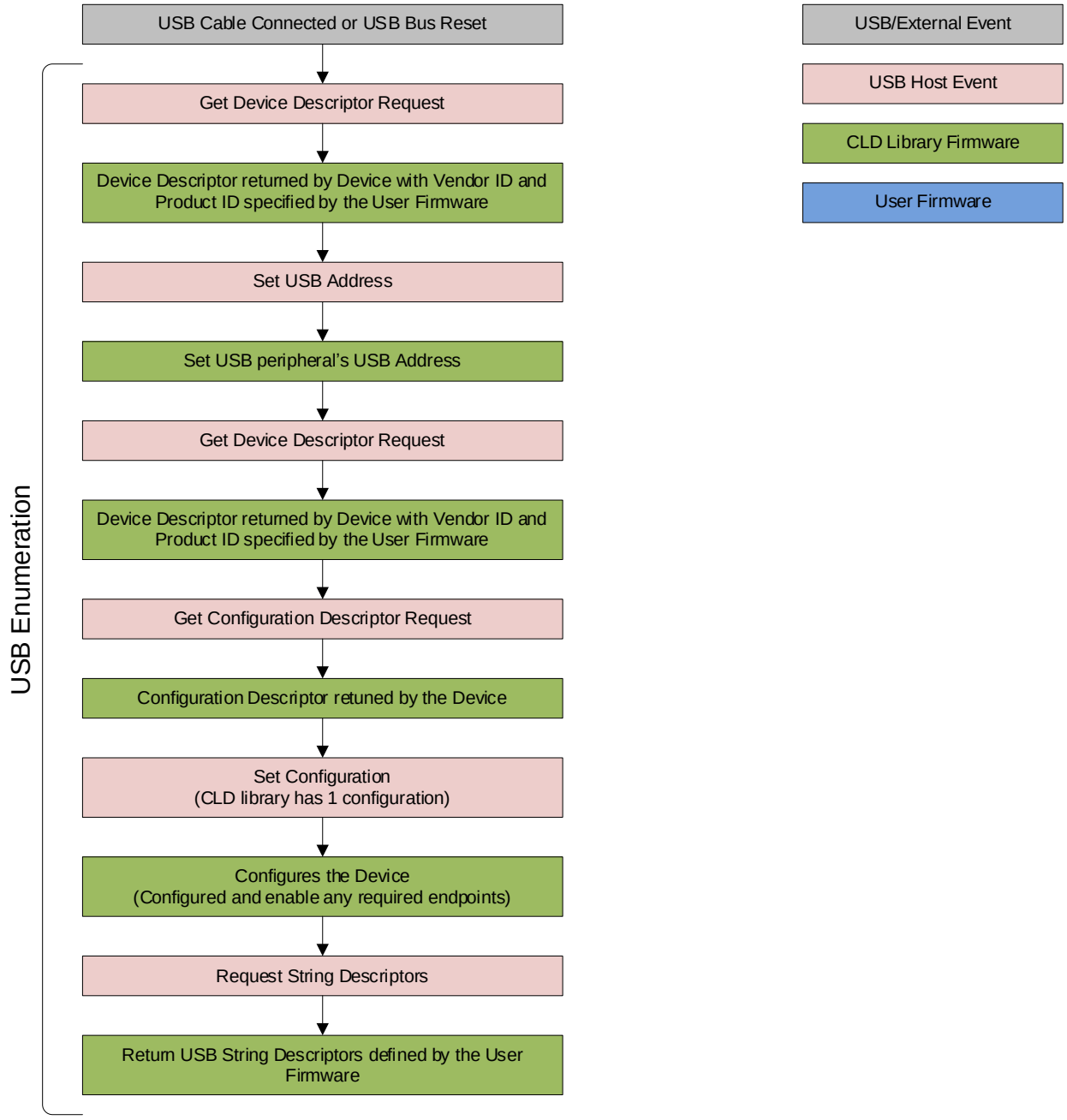
The following is a very basic overview of some of the USB concepts that are necessary to use the CLD SC598 CDC Library. However, it is still recommended that developers have at least a basic understanding of the USB 2.0 protocol. The following are some resources to refer to when working with USB, and CDC 1.2 protocols:

- The USB 2.0 Specification
- The USB CDC Class specification v1.2
- USB in a Nutshell: A free online wiki that explains USB concepts. http://www.beyondlogic.org/usbnutshell/usb1.shtml
- "USB Complete" by Jan Axelson ISBN: 1931448086

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

The USB 2.0 protocol defines a basic framework that devices must implement in order to work correctly. This framework is defined in the Chapter 9 of the USB 2.0 protocol, and is often referred to as the USB 'Chapter 9' functionality. Part of the Chapter 9 framework is standard USB requests that a USB Host uses to control the Device. Another part of the Chapter 9 framework is the USB Descriptors. These USB Descriptors are used to notify the Host of the Device's capabilities when the Device is attached. The USB Host uses the descriptors and the Chapter 9 standard requests to configure the Device. This process is called USB Enumeration. The CLD library includes support for the USB standard requests and USB Enumeration using some of the parameters specified by the User application when initializing the library. These parameters are discussed in the cld_sc598_cdc_lib_init section of this document. The CLD library facilitates USB enumeration and is Chapter 9 compliant without User Application intervention as shown in the flow chart below. For additional information on USB Chapter 9 functionality or USB Enumeration please refer to one of the USB resources listed above.

CLD Library USB Enumeration Flow Chart



All USB data is transferred using Endpoints that act as a source or sink for data based on the endpoint's direction (IN or OUT). The USB protocol defines four types of Endpoints, each of which has unique characteristics that dictate how they are used. The four Endpoint types are: Control, Interrupt, Bulk and Isochronous. Data that is transmitted over USB is broken up into blocks of data called packets. For each endpoint type there are restrictions on the allowed max packet size. The allowed max packet sizes also vary based on the USB connection speed. Please refer to the USB 2.0 protocol for more information about the max packet size supported by the four endpoint types.

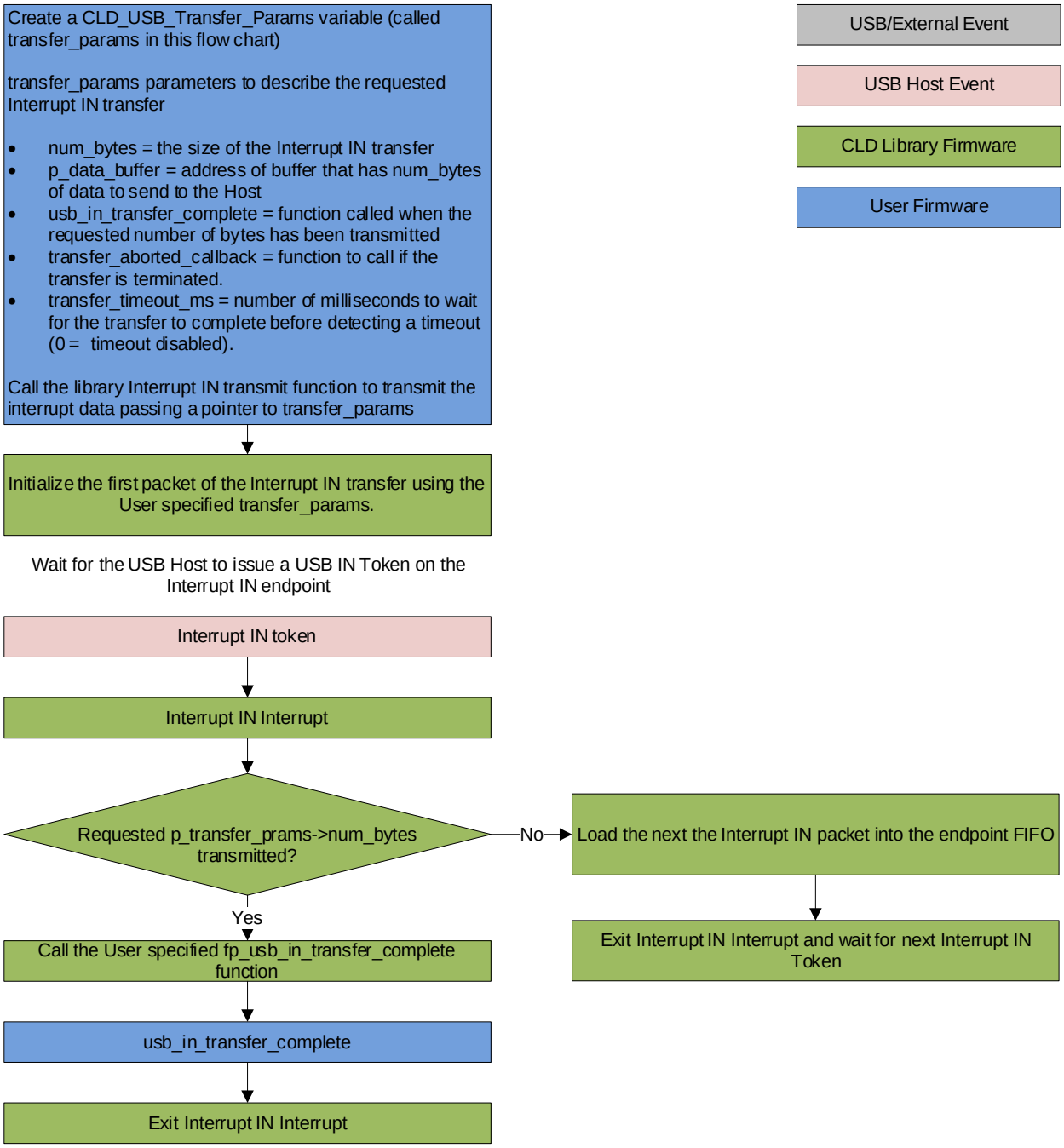
The CLD SC598 CDC Library uses Control, Interrupt, and Bulk endpoints, these endpoint types will be discussed in more detail below.

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

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

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

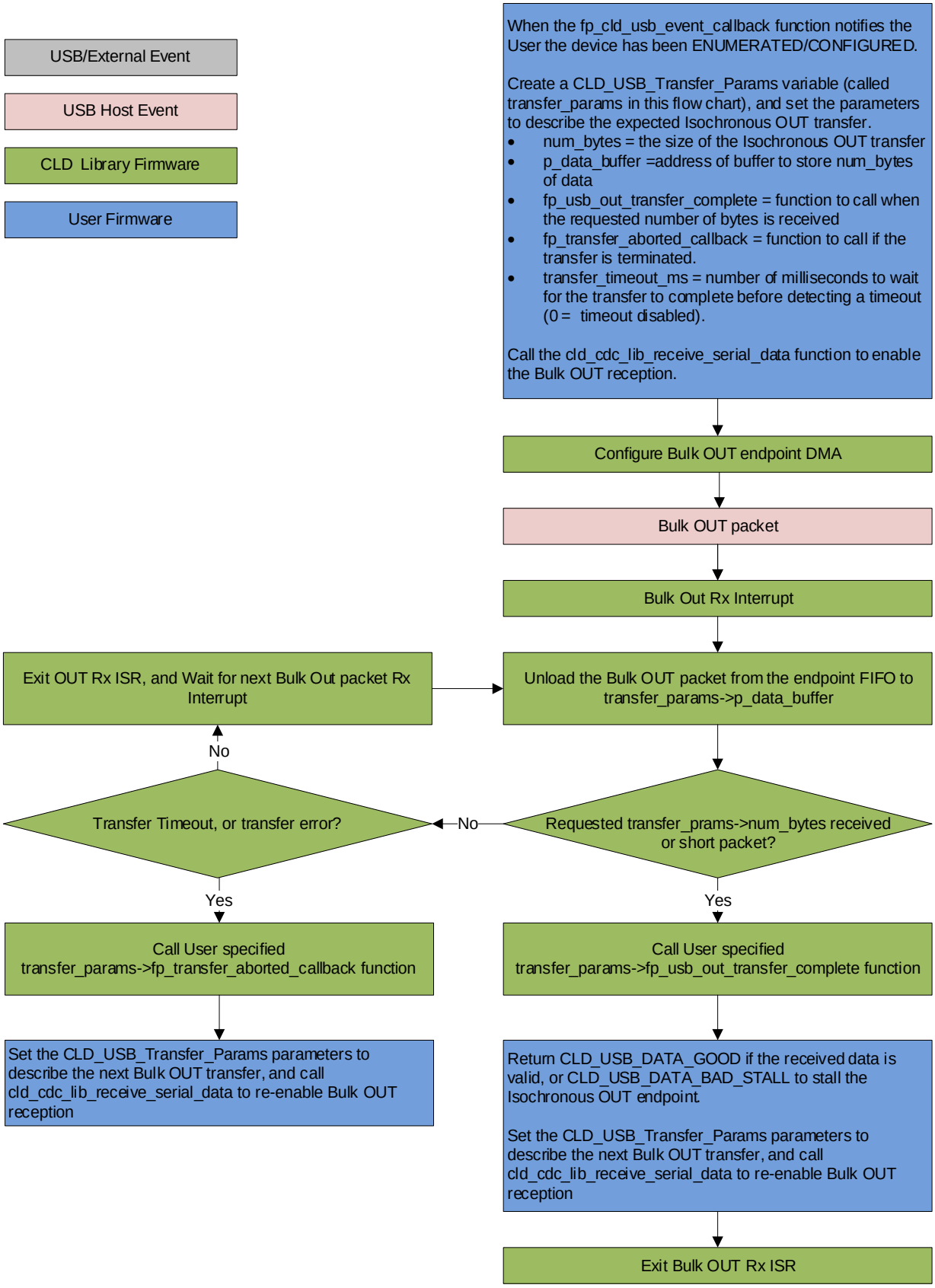
CLD Library Interrupt IN Flow Chart



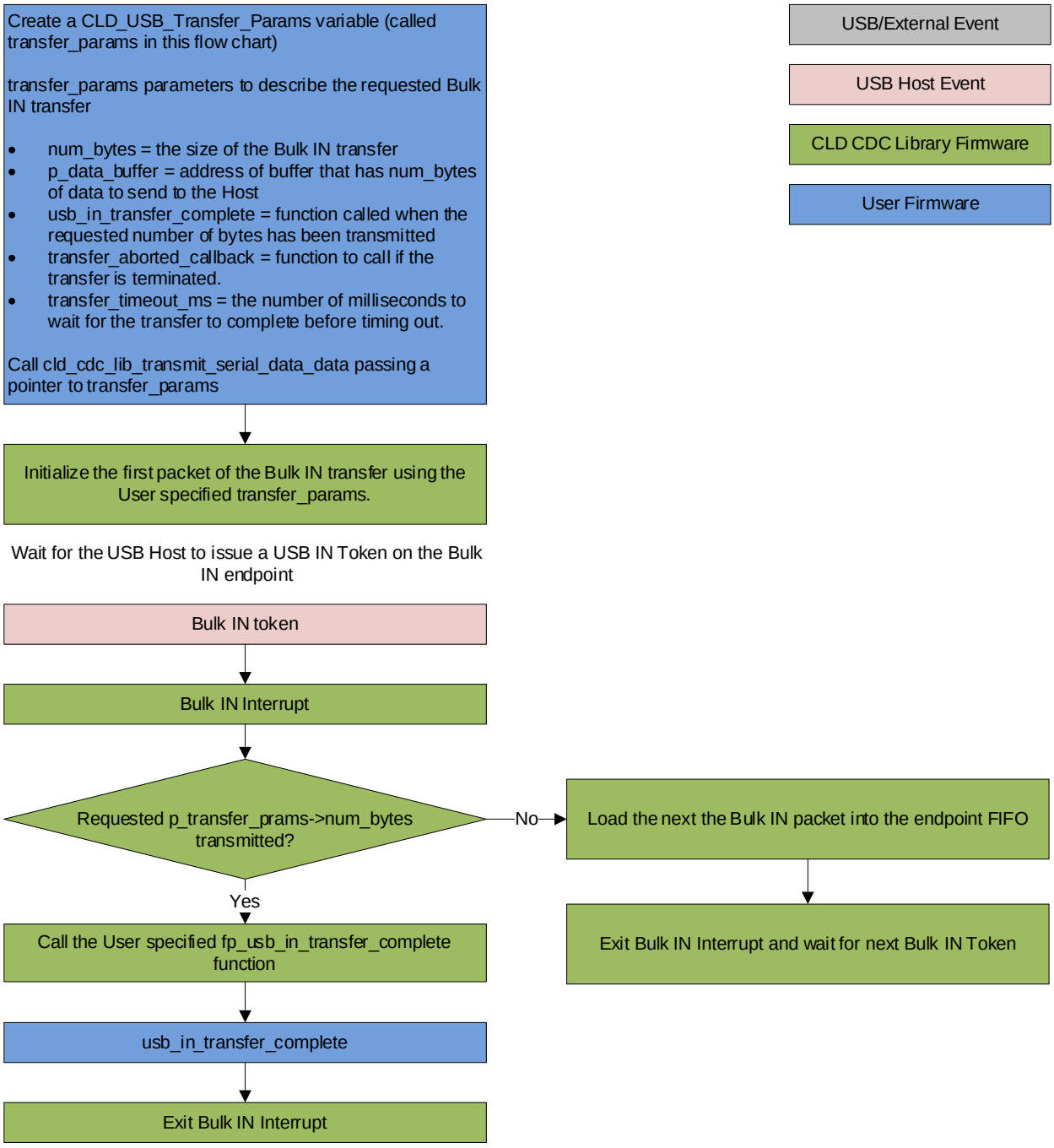
Bulk Endpoints are used to transfer large amounts of data where data integrity is critical, but does not require deterministic timing. A characteristic of Bulk Endpoints is that they can fill USB bandwidth that isn't used by the other endpoint types. This makes Bulk the lowest priority endpoint type, 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 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 CLD SC598 CDC Library and the User firmware interact to process Bulk OUT and Bulk IN transfers.

CLD CDC Library Bulk OUT Flow Chart


USB/External Event
USB Host Event
CLD Library Firmware
User Firmware
When the fp_cld_usb_event_callback function notifies the User the device has been ENUMERATED/CONFIGURED.
Create a CLD_USB_Transfer_Params variable (called transfer_params in this flow chart), and set the 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).
Call the cld_cdc_lib_receive_serial_data function to enable the Bulk OUT reception.
Configure Bulk OUT endpoint DMA
Bulk OUT packet
Bulk Out Rx Interrupt
Unload the Bulk OUT packet from the endpoint FIFO to transfer_params->p_data_buffer
Requested transfer_prams->num_bytes received or short packet?
No
Yes
Transfer Timeout, or transfer error?
No
Yes
Exit OUT Rx ISR, and Wait for next Bulk Out packet Rx Interrupt
Call User specified transfer_params->fp_usb_out_transfer_complete function
Call User specified transfer_params->fp_transfer_aborted_callback function
Return CLD_USB_DATA_GOOD if the received data is valid, or CLD_USB_DATA_BAD_STALL to stall the Isochronous OUT endpoint.
Set the CLD_USB_Transfer_Params parameters to describe the next Bulk OUT transfer, and call cld_cdc_lib_receive_serial_data to re-enable Bulk OUT reception
Set the CLD_USB_Transfer_Params parameters to describe the next Bulk OUT transfer, and call cld_cdc_lib_receive_serial_data to re-enable Bulk OUT reception
Exit Bulk OUT Rx ISR

CLD CDC Library Bulk IN Flow Chart



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 library implements an 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 an 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

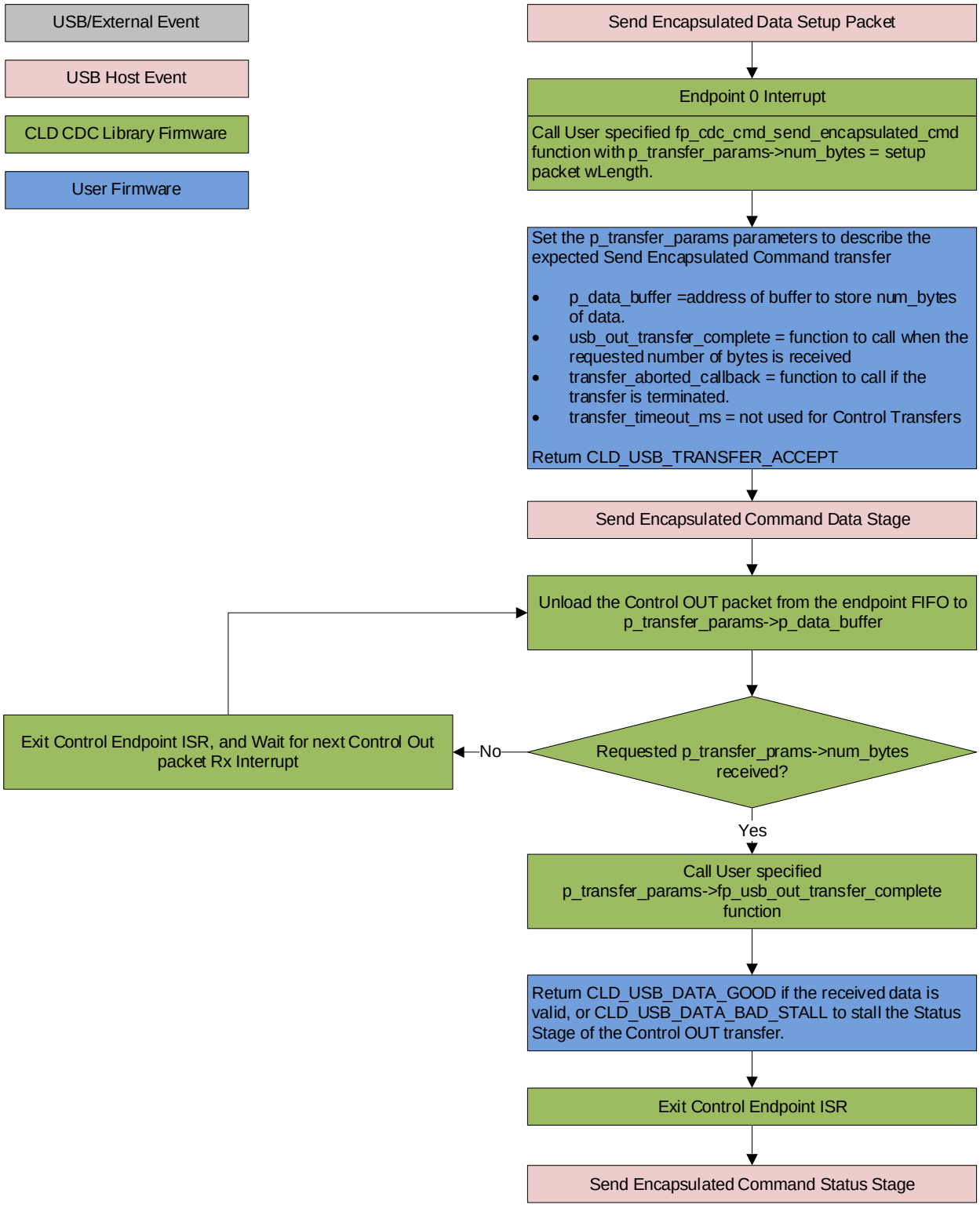
The CDC Abstract Control Model defines a couple Control Endpoint requests that a CDC peripheral is required to support as well as some optional Control Endpoint requests. The Control Endpoint requests used by the CLD library are explained in the following sections, and include flow charts showing how the CLD SC598 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 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.

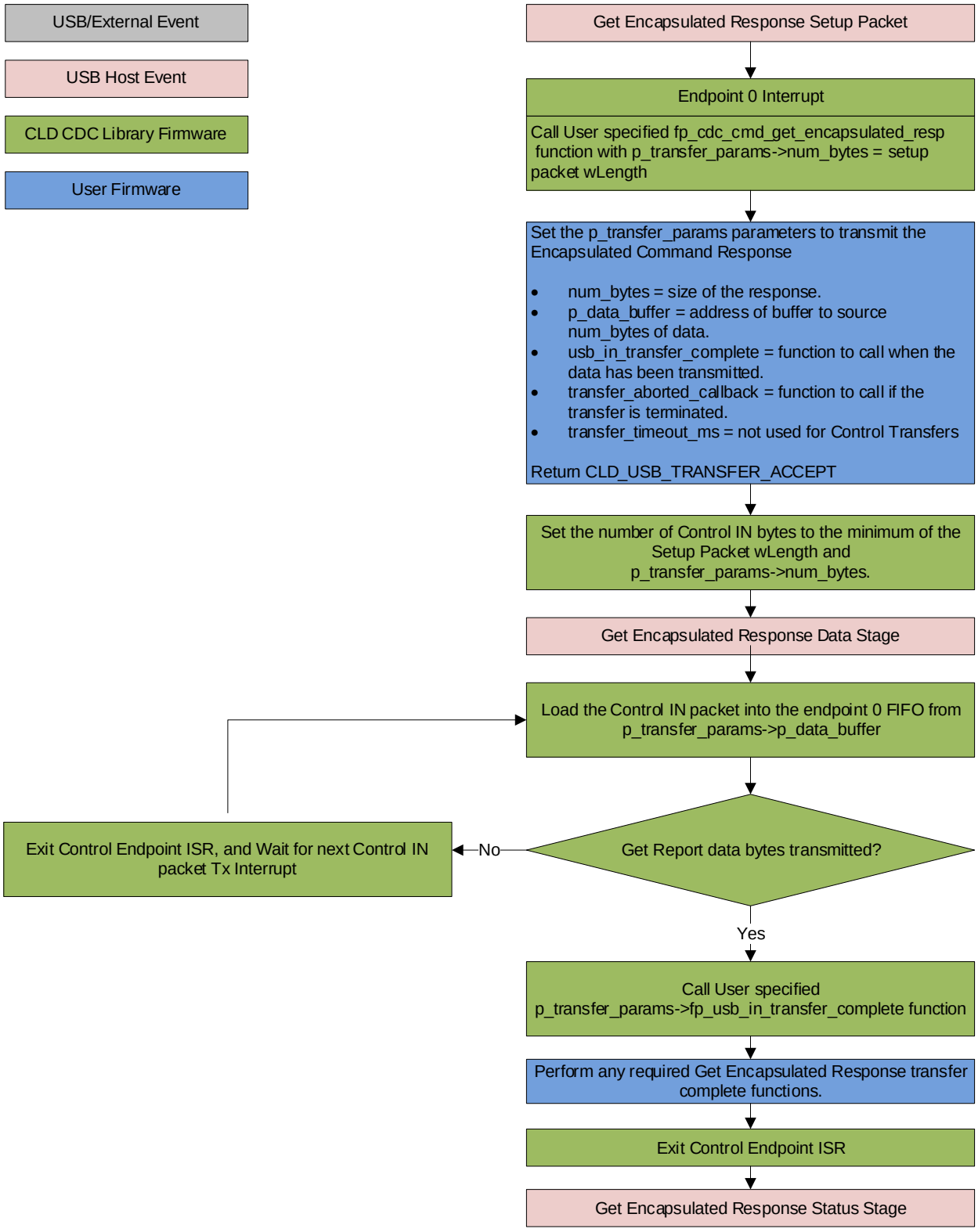
CLD CDC Library Send Encapsulated Command Flow Chart



Get Encapsulated Command (required)

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

CLD CDC Library Get Encapsulated Command Flow Chart



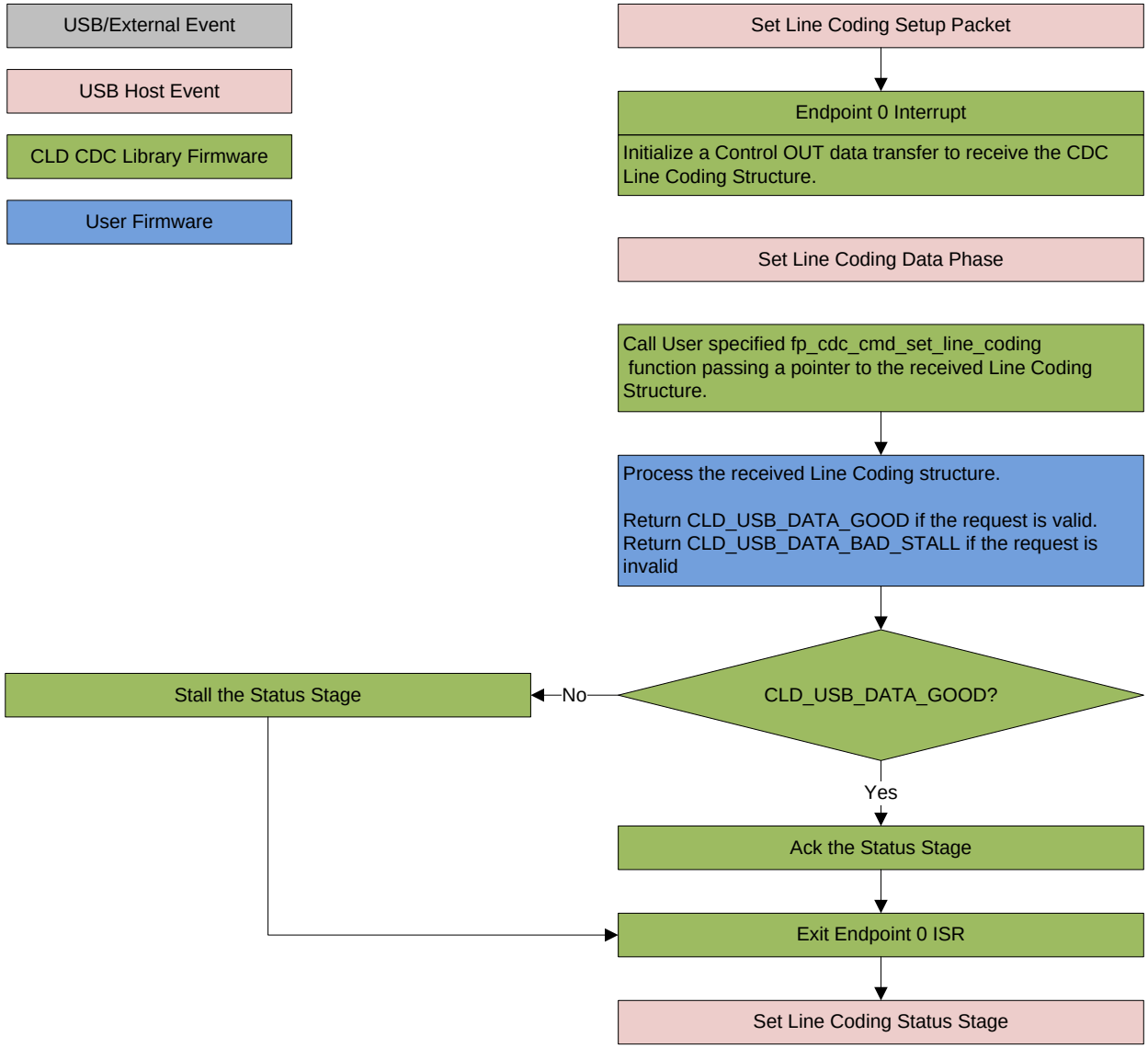
Set Line Coding (optional)

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.

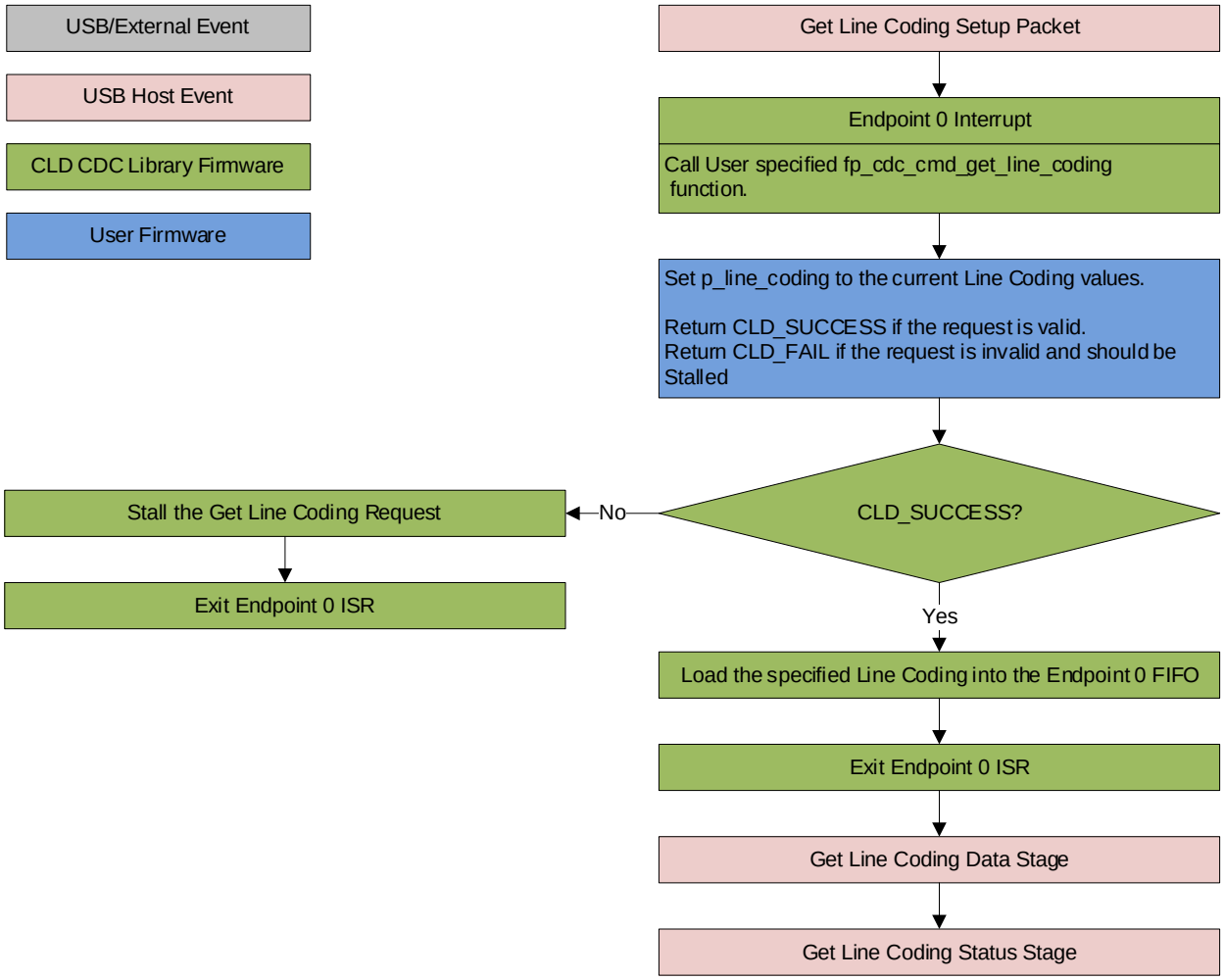
CLD CDC Library Set Line Coding Flow Chart



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 CDC Library Get Line Coding Flow Chart

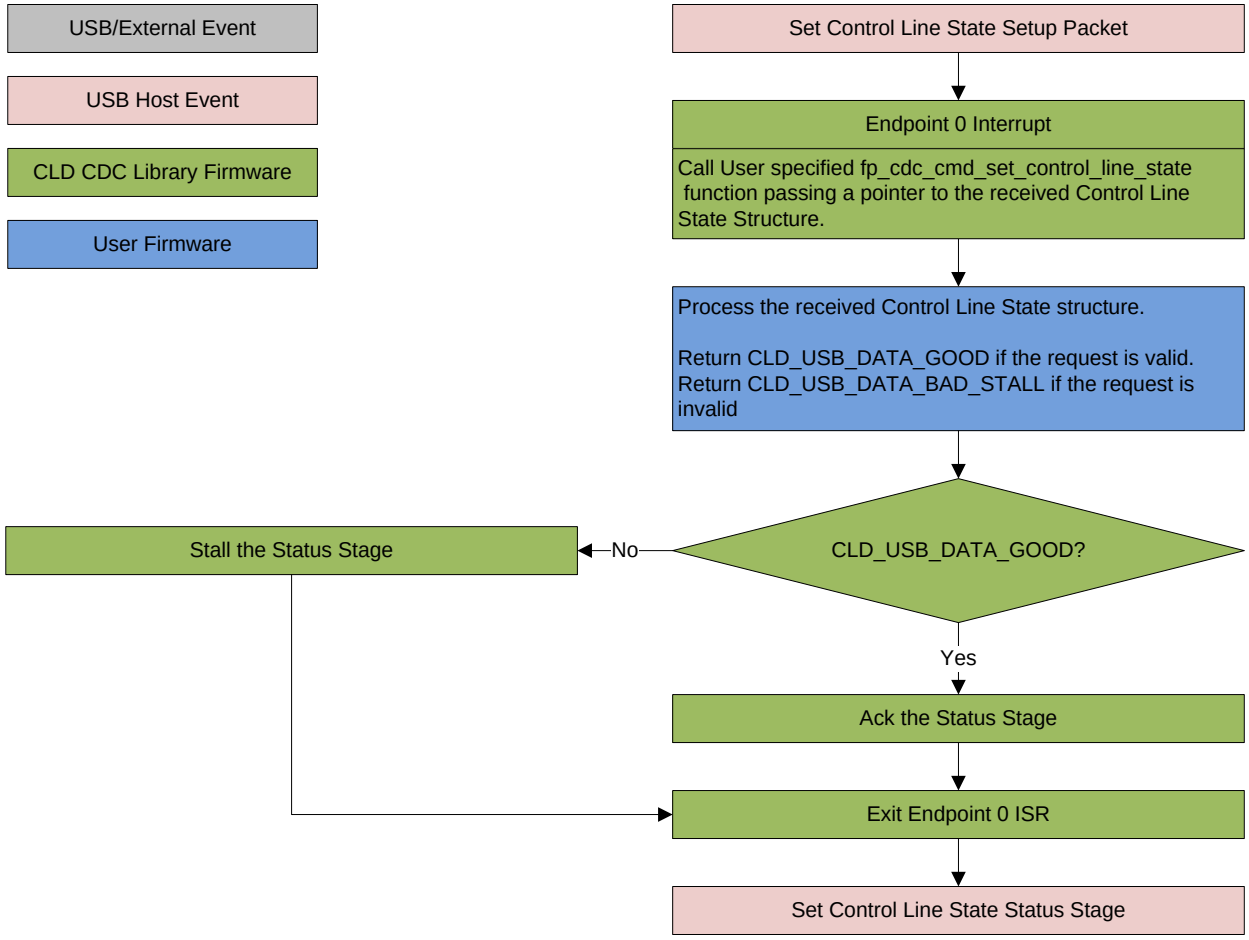


Set Control Line State (optional)

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;
```

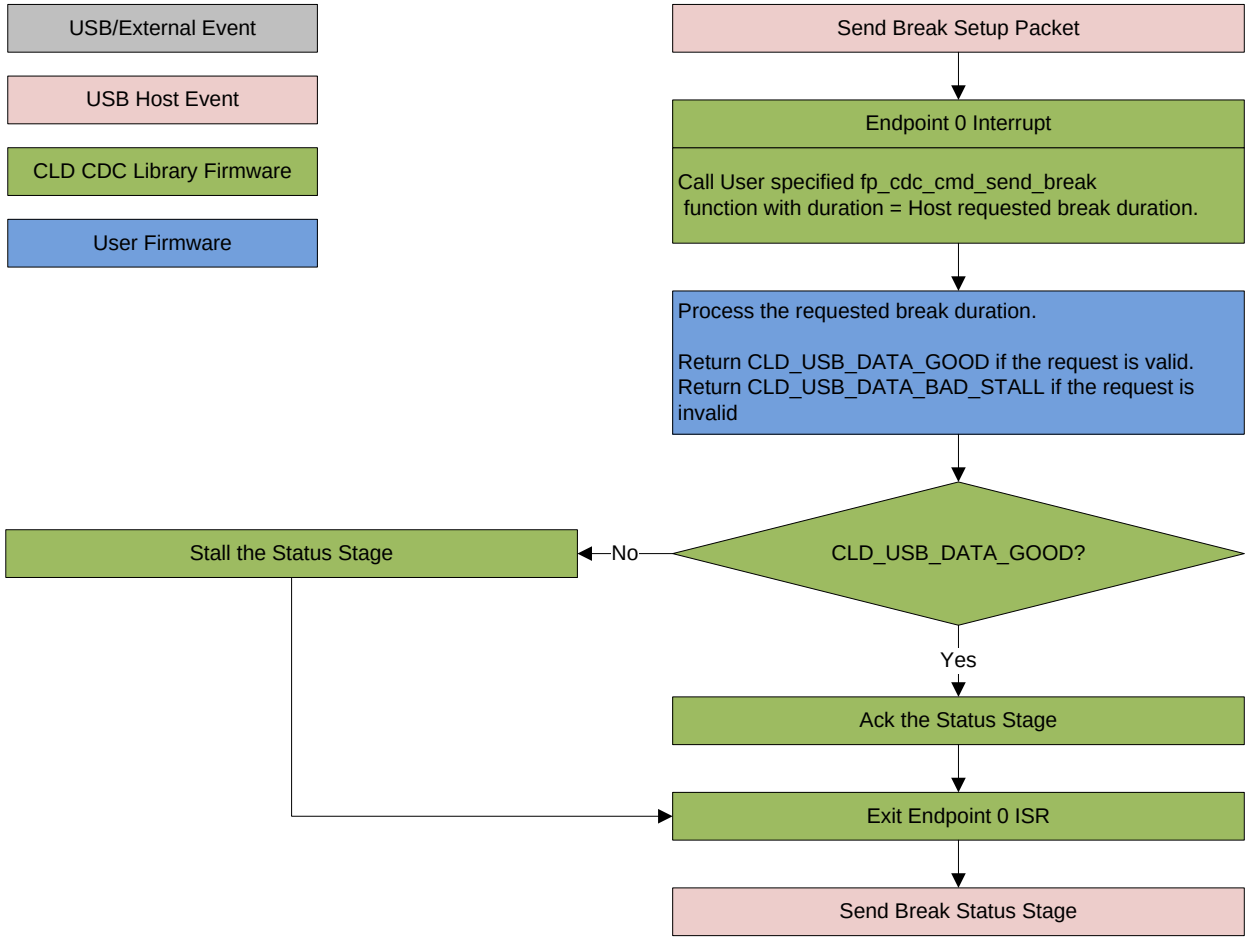
CLD CDC Library Set Control Line State Flow Chart



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 a another Send Break command is received with a duration of 0.

CLD CDC Library Send Break Flow Chart



Dependencies

In order to function properly, the CLD SC598 CDC Library requires the following resources:

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

CLD SC598 CDC Library Scope and Intended Use

The CLD SC598 CDC Library implements the CDC/ACM required functionality to implement a USB CDC device, as well as providing time measurements functionality. The CLD library is designed to be added to an existing User project, and as such only includes the functionality needed to implement the above mentioned USB, and timer keeping features. All other aspects of SC598 processor configuration must be implemented by the User code.

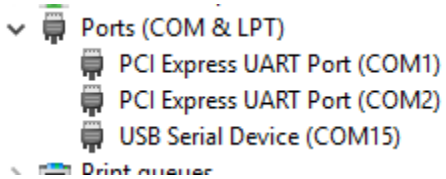
CLD CDC Example v1.00 Description

The CLD example project provided with the CLD SC598 CDC Library implements a CDC Abstract Control model USB serial port echo of data received over USB.

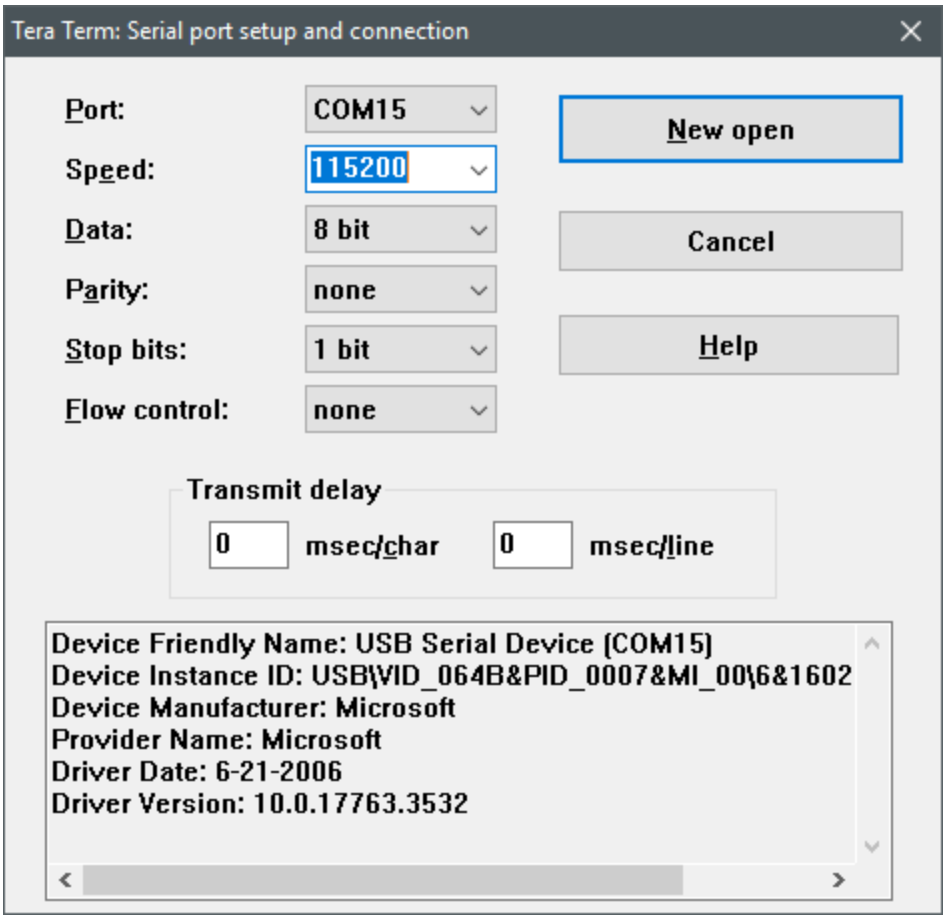
Running the Example Project

1. With the example project was developed using the ADSP SC598 SOM and carrier board, and toggles the LED connected to GPIO port C pin 3 every 250 milliseconds to provide a visual indicator the project is running.

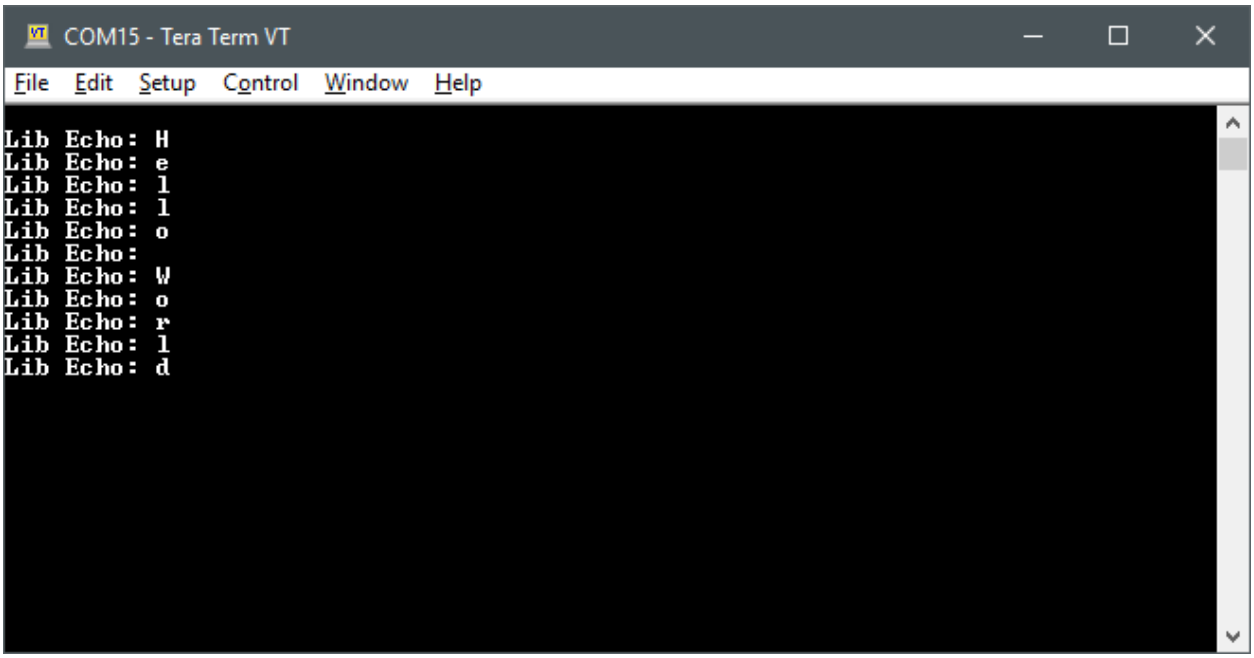
2. Once the example project is running on the EZ Board connect a USB mini-b cable from a PC to the “USB Phy” connector of the carrier board. Windows 10 will install its built-in CDC/ACM driver, and the device will be listed as a USB Serial Device in the Device Manager as shown below:



3. Using TeraTerm, or another serial terminal program, connect to the new serial port as shown below and click New Open:



4. The example project will echo the data it received over USB prepended with “Lib Echo:” as shown below:



CLD SC598 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_sc598_cdc_lib_init

```
CLD_RV cld_sc598_cdc_lib_init (CLD_SC598_CDC_Lib_Init_Params * p_lib_params)
```

Initializes the CLD SC598 CDC Library.

Arguments

p_lib_params	Pointer to a CLD_SC598_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 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_sc598_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 fp_cld_lib_status function if defined by the User application. Once the library has been initialized successfully the main program loop can start.

The CLD_SC598_CDC_Lib_Init_Params structure is described below:

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

    CLD_RV (*fp_init_usb_phy) (void);

    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_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 support_cdc_network_connection;
    unsigned short cdc_class_bcd_version;
    unsigned char  cdc_class_control_protocol_code;

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

    unsigned char user_string_descriptor_table_num_entries;
    CLD_CDC_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_SC598_CDC_Lib_Init_Params;
```

A description of the CLD_SC598_CDC_Lib_Init_Params structure elements is included below:

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

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

The fp_init_usb_phy function returns the CLD_RV type, which has the following values:
<table>
<tr><td>Return Value</td><td>Description</td></tr>
<tr><td>CLD_ONGOING</td><td>Results in this function getting additional runtime.</td></tr>
<tr><td>CLD_SUCCESS</td><td>USB Phy initialized successfully.</td></tr>
<tr><td>CLD_FAIL</td><td>Phy initialization failed, causes USB library initialization failure.</td></tr>
</table></td></tr>
<tr><td>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_sc598_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>
</table>

<table>
<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><th>Structure Element</th><th>Description</th></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_sc598_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><th>Structure Element</th><th>Description</th></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_sc598_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</td></tr></table></td></tr>
</table>

<table>
<tr><td></td><td><table>
<tr><td></td><td>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 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_cdc_cmd_send_encapsulated_cmd</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:

<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, which is the number of bytes that will be transferred to p_data_buffer before calling the fp_usb_out_transfer_ complete callback function.</td></tr>
<tr><td>p_data_buffer</td><td>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.</td></tr>
<tr><td>fp_usb_out_transfer_complete</td><td>Function called when num_bytes of data has been written to the p_data_buffer memory.</td></tr>
<tr><td>fp_transfer_aborted_callback</td><td>Function called if there is a problem receiving 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_send_encapsulated_cmd function returns the</td></tr>
</table>

<table>
<tr><td></td><td>CLD_USB_Transfer_Request_Return_Type, which has the following values:
<table>
<tr><th>Return Value</th><th>Description</th></tr>
<tr><td>CLD_USB_TRANSFER_ACCEPT</td><td>Notifies the CLD library that the Send Encapsulated Command data should be accepted using the p_transfer_data values.</td></tr>
<tr><td>CLD_USB_TRANSFER_PAUSE</td><td>Requests that the CLD library pause the Set Report transfer. This causes the Control Endpoint to be nak'ed until the transfer is resumed by calling cld_cdc_lib_resume_ paused_control_transfer.</td></tr>
<tr><td>CLD_USB_TRANSFER_DISCARD</td><td>Requests that the CLD library discard the number of bytes specified in 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 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><th>Structure Element</th><th>Description</th></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</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>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 library that the Get 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 library pause the Get Encapsulated Response transfer. This causes the Control Endpoint to be nak'ed until the transfer is resumed by calling cld_cdc_lib_resume_ paused_control_transfer.</td></tr>
<tr><td>CLD_USB_TRANSFER_DISCARD</td><td>Requests that the CLD library to return a zero length packet in response to the Get Encapsulated Response request.</td></tr>
<tr><td>CLD_USB_TRANSFER_STALL</td><td>This notifies the CLD library that there is an error and the request should be stalled.</td></tr>
</table>
</td></tr>
<tr><td>fp_cdc_cmd_set_line_coding</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</td></tr>
</table>
</td></tr>
</table>

	1 = 1.5 stop bits 2 = 2 stop bits.
parity	CDC parity setting 0 = None 1 = Odd 2 = Even 3 = Mark 4 = Space
num_data_bits	CDC Number of data bits (only 5, 6, 7, 8 and 16 are valid).

The fp_cdc_cmd_set_line_coding function returns the CLD_USB_Data_Received_Return_Type, which has the following values:

Return Value	Description
CLD_USB_DATA_GOOD	Notifies the CLD library that the request is valid.
CLD_USB_DATA_BAD_STALL	Notifies the CLD library that the request is invalid, and should be stalled.

fp_cdc_cmd_get_line_coding

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:

Structure Element	Description
data_terminal_rate	Serial baud rate in bits per second.
num_stop_bits	CDC Number of stop bits. 0 = 1 stop bit 1 = 1.5 stop bits 2 = 2 stop bits.
parity	CDC parity setting 0 = None 1 = Odd 2 = Even 3 = Mark 4 = Space
num_data_bits	CDC Number of data bits (only 5, 6, 7, 8 and 16 are valid).

The fp_cdc_cmd_get_line_coding function returns CLD_RV, which has the following values:

Return Value	Description
CLD_SUCCESS	Notifies the CLD library that the request is valid and the p_line_coding value should be returned to the Host.
CLD_FAIL	Notifies the CLD library that the request is invalid, and

<table>
<tr><td></td><td><table><tr><td></td><td>should be stalled.</td></tr></table></td></tr>
<tr><td>fp_cdc_cmd_set_control_line_state</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 elements are used to processed a Set Control Line State request:<table><tr><td>Structure Element</td><td>Description</td></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><td>Return Value</td><td>Description</td></tr><tr><td>CLD_USB_DATA_GOOD</td><td>Notifies the CLD library that the request is valid.</td></tr><tr><td>CLD_USB_DATA_BAD_STALL</td><td>Notifies the CLD library that the request is invalid, and should be stalled.</td></tr></table></td></tr>
<tr><td>fp_cdc_cmd_send_break</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><td>Return Value</td><td>Description</td></tr><tr><td>CLD_USB_DATA_GOOD</td><td>Notifies the CLD library that the request is valid.</td></tr><tr><td>CLD_USB_DATA_BAD_STALL</td><td>Notifies the CLD library that the request is invalid, and should be stalled.</td></tr></table></td></tr>
<tr><td>support_cdc_network_connection</td><td>Tells the CLD 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>
<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>
</table>

<table>
<tr><td>p_usb_string_manufacturer</td><td>Pointer to the null-terminated string. This string is used by the library to generate the Manufacturer USB String Descriptor. If the Manufacturer String Descriptor is not used set p_usb_string_manufacturer to CLD_NULL.</td></tr>
<tr><td>p_usb_string_product</td><td>Pointer to the null-terminated string. This string is used by the CLD library to generate the Product USB String Descriptor. If the Product String Descriptor is not used set p_usb_string_product to CLD_NULL.</td></tr>
<tr><td>p_usb_string_serial_number</td><td>Pointer to the null-terminated string. This string is used by the CLD library to generate the Serial Number USB String Descriptor. If the Serial Number String Descriptor is not used set p_usb_string_serial_number to CLD_NULL.</td></tr>
<tr><td>p_usb_string_configuration</td><td>Pointer to the null-terminated string. This string is used by the CLD library to generate the Configuration USB String Descriptor. If the Configuration String Descriptor is not used set p_usb_string_configuration to CLD_NULL.</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 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 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>user_string_descriptor_table_num_entries</td><td>The number of entries in the array of CLD_CDC_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.</td></tr>
<tr><td>p_user_string_descriptor_table</td><td>Pointer to an array of CLD_CDC_Lib_User_String_Descriptors structures used to define any custom User defined USB string descriptors. This table is used to define any USB String descriptors for any string descriptor indexes that are used in the Terminal or Unit Descriptors.

Set to CLD_NULL is not used.

The CLD_CDC_Lib_User_String_Descriptors structure elements are explained below:
<table>
<tr><th>Structure Element</th><th>Description</th></tr>
<tr><td>string_index</td><td>The USB String Descriptor index for the string. The string_index value is set to the index specified in the Terminal or Unit Descriptor associated with this string.</td></tr>
<tr><td>p_string</td><td>Pointer to a null terminated string.</td></tr>
</table>
</td></tr>
<tr><td>usb_string_language_id</td><td>16-bit USB String Descriptor Language ID Code as defined in the USB Language Identifiers (LANGIDs) document (www.usb.org/developers/docs/USB_LANGIDs.pdf).</td></tr>
</table>

<table>
<tr><td></td><td>0x0409 = English (United States)</td></tr>
<tr><td>fp_cld_usb_event_callback</td><td>Function that is called when one of the following USB events occurs. This function has a single CLD_USB_Event parameter.

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

The CLD_USB_Event has the following values:
<table>
<tr><th>Return Value</th><th>Description</th></tr>
<tr><td>CLD_USB_CABLE_CONNECTED</td><td>USB Cable Connected.</td></tr>
<tr><td>CLD_USB_CABLE_DISCONNECTED</td><td>USB Cable Disconnected</td></tr>
<tr><td>CLD_USB_ENUMERATED_CONFIGURED_FS</td><td>USB device enumerated (USB Configuration set to a non-zero value) at Full-Speed</td></tr>
<tr><td>CLD_USB_ENUMERATED_CONFIGURED_HS</td><td>USB device enumerated (USB Configuration set to a non-zero value) at High-Speed</td></tr>
<tr><td>CLD_USB_UN_CONFIGURED</td><td>USB Configuration set to 0</td></tr>
<tr><td>CLD_USB_BUS_RESET</td><td>USB Bus reset received</td></tr>
</table>

Note: Set to CLD_NULL if not required by application</td></tr>
<tr><td>fp_cld_lib_status</td><td>Pointer to the function that is called when the CLD library has a status to report. This function has the following parameters:

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

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

cld_sc598_cdc_lib_main

```
void cld_cdc_lib_main (void)
```

CLD SC598 CDC Library mainline function

Arguments
None

Return Value
None.

Details
The cld_sc598_cdc_lib_main function is the CLD library mainline function that must be called in every iteration of the main program loop in order for the library to function properly.

cld_cdc_lib_receive_serial_data

```
CLD_USB_Data_Receive_Return_Type cld_cdc_lib_receive_serial_data
(CLD_USB_Transfer_Params * p_transfer_data)
```

CLD CDC Library function used to receive data over the Bulk OUT endpoint.

Arguments

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

Return Value

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

CLD_USB_RECEIVE_SUCCESSFUL	The library has configured the requested Bulk OUT transfer.
CLD_USB_RECEIVE_FAILED	The library failed to configure the requested Bulk OUT transfer. This will happen if the Bulk OUT endpoint is busy, or if the p_transfer_data-> data_buffer is set to CLD_NULL
CLD_USB_RECEIVE_FAILED_MISALIGNED	The requested USB transfer failed because the specified memory location isn't 32-bit aligned.

Details

The cld_cdc_lib_receive_serial_data enables the Bulk OUT endpoint to receive the data specified by the p_transfer_data parameter from the USB Host. This function should be called when the device has been enumerated/configured, in fp_usb_out_transfer_complete, and in fp_transfer_aborted_callback.

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)(unsigned
int num_bytes);
        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 received the fp_usb_in_transfer_complete callback function will be called.

p_data_buffer	Pointer to the data to be sent to the USB Host. This buffer must include the number of bytes specified by num_bytes.
fp_usb_out_*transfer*_complete	Function called when the specified data has been received, or the Host send a short packet (less than the max packet size) signaling the end of a transfer. This function is passed the number of received bytes.
fp_usb_in_transfer_complete	Not used for OUT transfers.
fp_transfer_aborted_callback	Function called if there is a problem receiving the data to the USB Host. This function can be set to CLD_NULL if the User application doesn't want to be notified if a problem occurs.
transfer_timeout_ms	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

cld_cdc_lib_transmit_serial_data

```
CLD_USB_Data_Transmit_Return_Type cld_cdc_lib_transmit_serial_data
      (CLD_USB_Transfer_Params * p_transfer_data)
```

CLD 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
CLD_USB_TRANSMIT_FAILED_MISALIGNED	The requested USB transfer failed because the specified memory location isn't 32-bit aligned.

Details

The cld_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_cdc_lib_send_network_connection_state

```
CLD_USB_Data_Transmit_Return_Type cld_cdc_lib_send_network_connection_state
      (CLD_CDC_Lib_Network_Connection_State state)
```

CLD 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_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_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_cdc_lib_send_response_available

```
CLD_USB_Data_Transmit_Return_Type cld_cdc_lib_send_response_available
      (CLD_CDC_Lib_Network_Connection_State state)
```

CLD 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_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_cdc_lib_send_serial_state

```
CLD_USB_Data_Transmit_Return_Type cld_cdc_lib_send_serial_state
      (CLD_CDC_Serial_State * p_serial_state)
```

CLD 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_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_cdc_lib_resume_paused_control_transfer

```
void cld_cdc_lib_resume_paused_control_transfer (void)
```

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

Arguments

None

Return Value

None.

Details

The cld_cdc_lib_resume_paused_control_transfer function is used to resume a Control transfer which 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_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 Library function used to connect to the USB Host.

Return Value
None.

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

cld_ lib_usb_disconnect

```
void cld_lib_usb_disconnect (void)
```

CLD library function used to disconnect from the USB Host.

Return Value
None.

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

cld_time_125us_tick

```
void cld_time_125us_tick (void)
```

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

Arguments
None

Return Value
None.

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

cld_usb0_isr_callback

```
void cld_usb0_isr_callback (void)
```

CLD library USB interrupt service routines

Arguments
None

Return Value
None.

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

cld_time_get

```
CLD_Time cld_time_get(void)
```

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

Arguments
None

Return Value
The current CLD library time.

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

cld_time_passed_ms

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

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

Arguments

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

Return Value

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

Details

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

cld_time_get_125us

```
CLD_Time cld_time_get_125us(void)
```

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

Arguments

None

Return Value

The current CLD library time.

Details

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

cld_time_passed_125us

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

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

Arguments

time	A CLD_Time value returned by a cld_time_get_125us function call.

Return Value

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

Details

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

cld_lib_status_decode

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

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

Arguments

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

Return Value

This function returns a decoded Null terminated ASCII string.

Details

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

cld_lib_access_usb_phy_reg

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

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

Arguments

p_params	Pointer to the CLD_USB_PHY_Access_Params structure describing the phy access.

Return Value

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

Details

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

The CLD_USB_PHY_Access_Params structure is described below.

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

A description of the CLD_USB_PHY_Access_Params structure elements is included below:

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

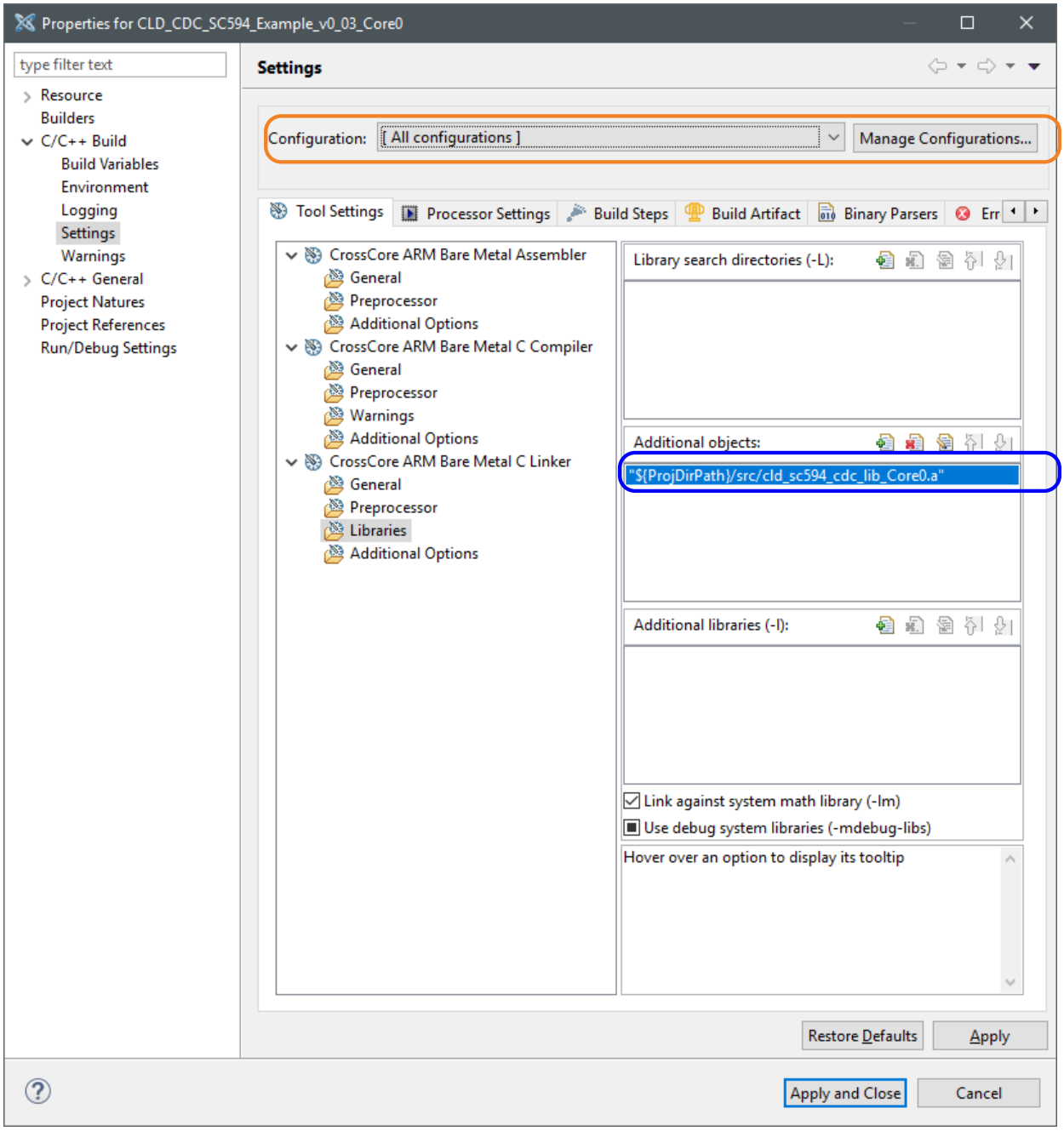
Adding the CLD SC598 CDC Library to an Existing CrossCore Embedded Studio Project

In order to include the CLD SC598 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_sc598_cdc_lib.h and cld_sc598_cdc_lib_Core0.a files to the project's src directory.
2. Open the project in CrossCore Embedded Studio.
3. Right click the project in the 'C/C++ Projects' window and select Properties.

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

   Navigate to the C/C++ Build → Settings page and select the CrossCore ARM Bare Metal C Linker's Libraries page. The CLD SC598 CDC Library needs to be included in the projects 'Additional objects' as shown in the diagram below (circled in blue). This lets the linker know where the cld_sc598_cdc_lib_Core0.a file is located.



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

User Firmware Code Snippets

The following code snippets are not complete, and are meant to be a starting point for the User firmware. For a functional User firmware example that uses the CLD SC598 CDC Library please refer to the CLD example projects included available with the CLD SC598 CDC Library.

main.c

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

    while (1)
    {
        switch (main_state)
        {
            case MAIN_STATE_SYSTEM_INIT:
                /* Initialize the SC598 clock, and power systems.*/

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

            case MAIN_STATE_RUN:
                 user_main();
            break;

            case MAIN_STATE_ERROR:

            break;
        }
    }
}
```

user.c

```
/*!< CDC Notification Interrupt IN endpoint parameters. */
static CLD_CDC_Notification_Endpoint_Params user_cdc_notification_ep_params =
{
    .endpoint_number               = 4,
    .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               = 5,
    .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               = 5,
    .max_packet_size_full_speed    = 64,
    .max_packet_size_high_speed    = 512,
};


/*!< CLD Library initialization data. */
static CLD_SC598_CDC_Lib_Init_Params user_cdc_init_params =
{
    .vendor_id = 0x064b,        /* Analog Devices Vendor ID */
    .product_id = 0x0008,       /* Product ID. */
    .usb_bus_max_power = 0,

    .device_descriptor_bcdDevice = 0x0100,

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

    .fp_init_usb_phy                = user_init_usb_phy,

    .p_serial_data_rx_endpoint_params = &user_cdc_serial_data_rx_ep_params,
    .p_serial_data_tx_endpoint_params = &user_cdc_serial_data_tx_ep_params,
    .p_notification_endpoint_params   = &user_cdc_notification_ep_params,

    .fp_cdc_cmd_send_encapsulated_cmd = user_cdc_cmd_send_encapsulated_cmd,
    .fp_cdc_cmd_get_encapsulated_resp = user_cdc_cmd_get_encapsulated_resp,

    .fp_cdc_cmd_set_line_coding       = user_cdc_cmd_set_line_coding,
    .fp_cdc_cmd_get_line_coding       = user_cdc_cmd_get_line_coding,

    .fp_cdc_cmd_set_control_line_state= user_cdc_cmd_set_control_line_state,

    .fp_cdc_cmd_send_break            = user_cdc_cmd_send_break,

    .support_cdc_network_connection   = 1,
    .cdc_class_bcd_version            = 0x0120,       /* CDC Version 1.2 */
    .cdc_class_control_protocol_code  = 0,            /* No Class Specific protocol */
```

```
    /* USB string descriptors - Set to CLD_NULL if not required */
    .p_usb_string_manufacturer  = "Analog Devices Inc",
    .p_usb_string_product       = "SC598 CDC Device",
    .p_usb_string_serial_number = CLD_NULL,
    .p_usb_string_configuration = CLD_NULL,
    .p_usb_string_communication_class_interface  = "CLD CDC Ctrl",
    .p_usb_string_data_class_interface           = "CLD CDC Data",

    .user_string_descriptor_table_num_entries = 0,
    .p_user_string_descriptor_table = CLD_NULL,

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

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

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

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

    switch (user_init_state)
    {
        case 0:

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

            user_init_state++;
        break;
        case 1:
            /* Initialize the CLD Library */
            cld_rv = cld_sc598_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_INIT_SUCCESS;
            }
            else if (cld_rv == CLD_FAIL)
            {
                init_return_code = USER_INIT_FAILED;
            }
            else
            {
                init_return_code = USER_INIT_ONGOING;
            }
    }
    return init_return_code;
}

void user_main (void)
{
    cld_sc598_cdc_lib_main();

}

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

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

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

/* 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_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);
}
```