



CLOSED LOOP DESIGN LLC

USB CM40x Bulk Library v.1.0 Users Guide

Users Guide Revision 1.0

For Use With Analog Devices ADSP-CM40x Series Processors

Closed Loop Design, LLC

748 S MEADOWS PKWY STE A-9-202

Reno, NV 89521

support@cld-llc.com

Table of Contents

Disclaimer	3
Introduction.....	3
USB Background	3
CLD CM40x Bulk Library USB Enumeration Flow Chart	4
CLD CM40x Bulk Library Bulk OUT Flow Chart.....	6
CLD CM40x Bulk Library Bulk IN Flow Chart.....	7
Dependencies	8
Memory Footprint.....	8
CLD CM40x Bulk Library Scope and Intended Use	8
CLD Bulk Loopback Example v1.0 Description	8
CLD CM40x Bulk Library API	9
cld_cm40x_bulk_lib_init	9
cld_cm40x_bulk_lib_transmit_bulk_in_data.....	13
cld_cm40x_bulk_lib_resume_paused_bulk_out_transfer.....	15
cld_lib_usb_connect	16
cld_lib_usb_disconnect.....	16
cld_time_get.....	17
cld_time_passed_ms	17
cld_timer_1ms.....	18
Adding the CLD CM40x Bulk Library to an Existing Project	19
Using ADI hostapp.exe	20
ADI hostapp Windows USB Driver Installation.....	21
User Firmware Code Snippets	25
main.c.....	25
user_bulk.c.....	26

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 family processors only. Nothing else gives you the right to use this software.

Introduction

The Closed Loop Design (CLD) Bulk library creates a simplified interface for developing a Bulk IN/Bulk OUT USB 2.0 device using the Analog Devices ADSP-CM408F EZ-Kit. The CLD CM40x Bulk library also includes support for timer functions that facilitate creating timed events quickly and easily. The library's CM408 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 CM40x Bulk 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 CM40x Bulk 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:

- The USB 2.0 Specification: http://www.usb.org/developers/docs/usb20_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 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 the USB Enumeration. The CLD CM40x Bulk 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 `clid_cm40x_bulk_lib_init` section of this document. The CLD CM40x Bulk 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 CM40x Bulk 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.

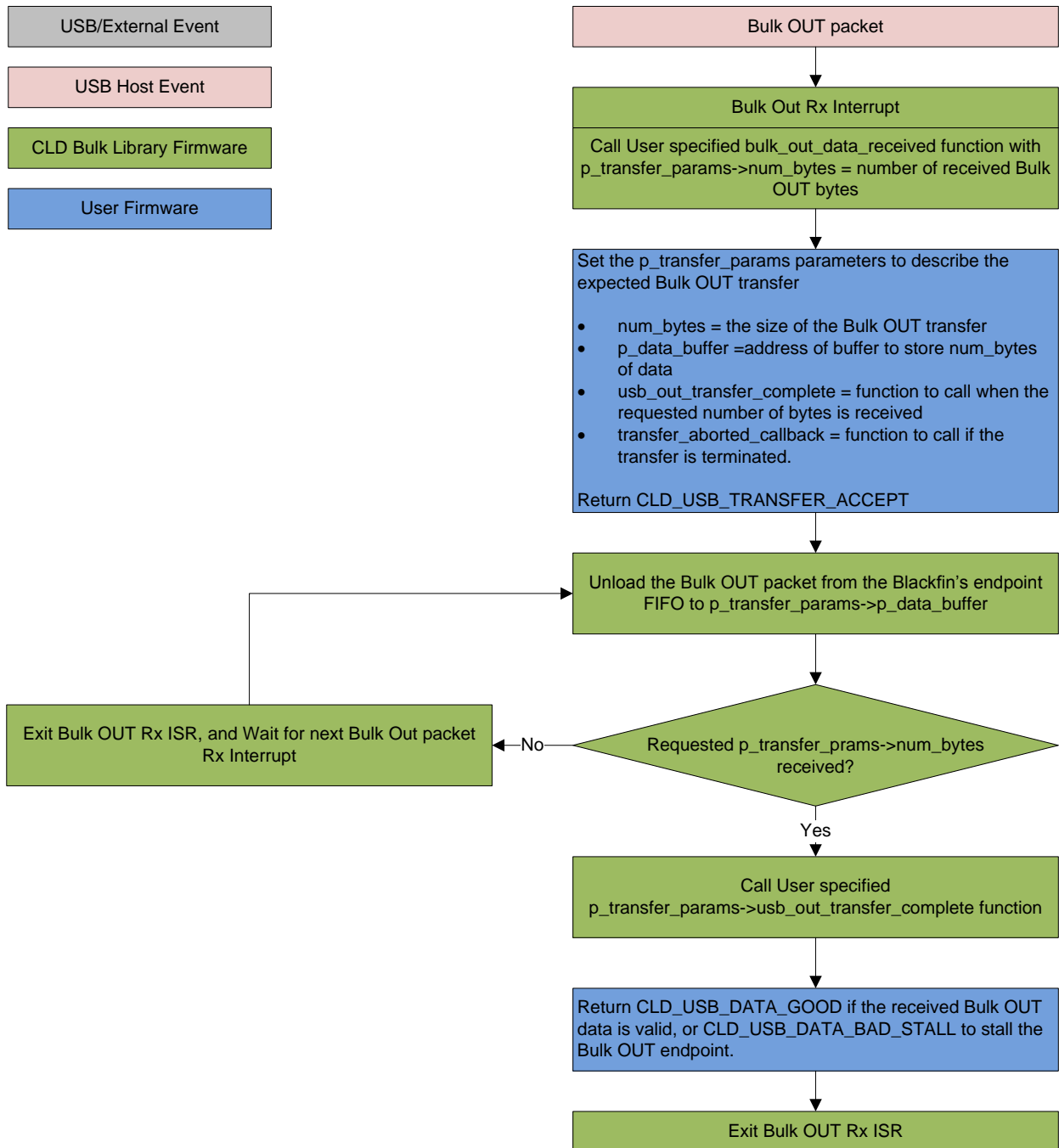
The CLD CM40x Bulk Library uses Control and Bulk endpoints, these endpoint types will be discussed in more detail below.

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 CM40x Bulk Library Endpoint 0 is only used for USB Chapter 9 requests, which are handled by the CLD CM40x Bulk library, thus Endpoint 0 is not accessible by the User application.

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 CM40x Bulk Library includes a Bulk IN and Bulk OUT endpoint, which are used to send and receive data with the USB Host, respectively.

The flow charts below give an overview of how the CLD CM40x Bulk Library and the User firmware interact to process Bulk OUT and Bulk IN transfers. Additionally, the User firmware code snippets included at the end of this document provide a basic framework for implementing a Bulk IN/Bulk Out device using the CLD CM40x Bulk Library.

CLD CM40x Bulk Library Bulk OUT Flow Chart



CLD CM40x Bulk Library Bulk IN Flow Chart

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.

Call cld_bulk_lib_transmit_bulk_in_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_params->num_bytes transmitted?

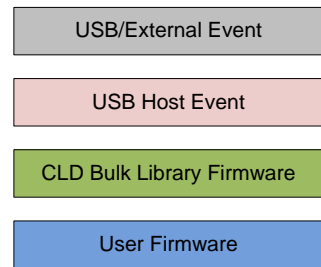
Yes
Call the User specified usb_in_transfer_complete function

usb_in_transfer_complete

Exit Bulk IN Interrupt

No
Load the next the Bulk IN packet into the Blackfin's endpoint FIFO

Exit Bulk IN Interrupt and wait for next Bulk IN Token



Dependencies

In order to function properly the CLD CM40x Bulk Library requires the following resources:

- One General Purpose Timer.
- 30Mhz clock input connected to the CLKIN pin.
- The User firmware is responsible for setting up the clocks, as well as enabling the Interrupt Vector Table (SCB->VTOR).

Memory Footprint

The CLD CM40x Bulk Library approximate memory footprint is as follows:

Code memory:	8240 bytes
Data memory:	1751 bytes
Total:	9991 bytes

Heap memory:	0 bytes
--------------	---------

Note: The CLD CM40x Bulk Library is currently optimized for balanced size and speed.

CLD CM40x Bulk Library Scope and Intended Use

The CLD CM40x Bulk Library implements a Vendor Specific Bulk IN/Bulk OUT USB device, as well as providing time measurements and optional bi-directional UART console functionality. The CLD CM40x Bulk 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 processor configuration must be implemented by the User code.

CLD Bulk Loopback Example v1.0 Description

The CLD_Bulk_loopback_example_v1_0 project provided with the CLD CM40x Bulk Library implements the Analog Devices (ADI) vendor specific Bulk IN/Bulk OUT protocol used by the ADI hostapp.exe program. This example is not intended to be used as a complete stand alone project. Instead, this project only includes the User functionality required to interface with hostapp.exe, and it is up to the User to include their own custom system initialization and any extra functionality they require.

For information about running the ADI hostapp program please refer to the "Using ADI hostapp.exe" section of this Users Guide.

CLD CM40x Bulk 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_cm40x_bulk_lib_init

CLD_RV **cld_cm40x_bulk_lib_init** (CLD_CM40x_Bulk_Lib_Init_Params *
cld_bulk_lib_params)

Initialize the CLD CM40x Bulk Library.

Arguments

cld_bulk_lib_params	Pointer to a CLD_CM40x_Bulk_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 CM40x Bulk 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_cm40x_bulk_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_CM40x_Bulk_Lib_Init_Params structure is described below:

```
typedef struct
{
    unsigned long sclk0;
    void (*fp_lib_msg_out) (char const * msg, const unsigned char msg_len);
    unsigned short vendor_id;
    unsigned short product_id;
    CLD_Bulk_Endpoint_Params * p_bulk_in_endpoint_params;
    CLD_Bulk_Endpoint_Params * p_bulk_out_endpoint_params;
    CLD_USB_Transfer_Request_Return_Type (*fp_bulk_out_data_received)
        (CLD_USB_Transfer_Params * p_transfer_data);
}
```

```

    unsigned char usb_bus_max_power;
    unsigned short device_descriptor_bcdDevice;
    const char * p_usb_string_manufacturer;
    const char * p_usb_string_product;
    const char * p_usb_string_serial_number;
    const char * p_usb_string_configuration;
    const char * p_usb_string_interface;
    unsigned short usb_string_language_id;
    void (*fp_cld_usb_event_callback) (CLD_USB_Event event);
} CLD_CM40x_Bulk_Lib_Init_Params;

```

A description of the CLD_CM40x_Bulk_Lib_Init_Params structure elements is included below:

Structure Element	Description								
sclk0	Used to tell the CLD CM40x Bulk Library the frequency of the ADSP_CM408 SCLK clock.								
fp_lib_msg_out	Pointer to the function that is called when a message is generated by the CLD CM40x Bulk Library. Note: Set to NULL if not required by application If enabled, maximum message length is 128 bytes								
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).								
product_id	The 16-bit product ID that is returned to the USB Host in the USB Device Descriptor.								
p_bulk_in_endpoint_params	Pointer to a CLD_Bulk_Endpoint_Params structure that describes how the Bulk IN endpoint should be configured. The CLD_Bulk_Endpoint_Params structure contains the following elements: <table border="1" data-bbox="630 1146 1422 1724"> <thead> <tr> <th>Structure Element</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>endpoint_num</td> <td>Sets the USB endpoint number of the Bulk endpoint. The endpoint number must be within the following range: $1 \leq \text{endpoint_num} \leq 7$. Any other endpoint number will result in the <code>cld_cm40x_bulk_lib_init</code> function returning <code>CLD_FAIL</code></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>N/A</td> </tr> </tbody> </table>	Structure Element	Description	endpoint_num	Sets the USB endpoint number of the Bulk endpoint. The endpoint number must be within the following range: $1 \leq \text{endpoint_num} \leq 7$. Any other endpoint number will result in the <code>cld_cm40x_bulk_lib_init</code> function returning <code>CLD_FAIL</code>	max_packet_size_full_speed	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.	max_packet_size_high_speed	N/A
Structure Element	Description								
endpoint_num	Sets the USB endpoint number of the Bulk endpoint. The endpoint number must be within the following range: $1 \leq \text{endpoint_num} \leq 7$. Any other endpoint number will result in the <code>cld_cm40x_bulk_lib_init</code> function returning <code>CLD_FAIL</code>								
max_packet_size_full_speed	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.								
max_packet_size_high_speed	N/A								
p_bulk_out_endpoint_params	Pointer to a CLD_Bulk_Endpoint_Params structure that describes how the Bulk Out endpoint should be configured. Refer to the <code>p_bulk_in_endpoint_params</code> description for information about the CLD_Bulk_Endpoint_Params structure.								
<i>fp_bulk_out_data_received</i>	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 process a Bulk OUT transfer:

Structure Element	Description
num_bytes	<p>The number of bytes to transfer to the p_data_buffer before calling the usb_out_transfer_complete callback function.</p> <p>When the bulk_out_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 CM40x Bulk Library will complete the entire bulk transfer without calling bulk_out_data_received again. If num_bytes isn't modified the bulk_out_data_received function will be called for each Bulk OUT packet.</p>
p_data_buffer	<p>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.</p>
<i>fp_usb_out_transfer_complete</i>	<p>Function called when num_bytes of data has been transferred to the p_data_buffer memory.</p>
<i>fp_transfer_aborted_callback</i>	<p>Function called if there is a problem transferring the requested Bulk OUT data.</p>
transfer_timeout_ms	<p>Bulk OUT transfer timeout in milliseconds. If the Bulk out transfer takes longer than this timeout the transfer is aborted and the transfer_aborted_callback is called.</p>

	Setting the timeout to 0 disables the timeout										
	The fp_bulk_out_data_received function returns the CLD_USB_Transfer_Request_Return_Type, which has the following values:										
	<table border="1"> <thead> <tr> <th>Return Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>CLD_USB_TRANSFER_ACCEPT</td> <td>Notifies the CLD CM40x Bulk 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 CM40x Bulk Library pause the current transfer. This causes the Bulk OUT endpoint to be nak'ed until the transfer is resumed by calling cld_cm40x_bulk_lib_resume_paused_bulk_out_transfer.</td> </tr> <tr> <td>CLD_USB_TRANSFER_DISCARD</td> <td>Requests that the CLD CM40x Bulk 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 CM40x Bulk Library that there is an error and the Bulk OUT endpoint should be stalled.</td> </tr> </tbody> </table>	Return Value	Description	CLD_USB_TRANSFER_ACCEPT	Notifies the CLD CM40x Bulk Library that the Bulk OUT data should be accepted using the p_transfer_data values.	CLD_USB_TRANSFER_PAUSE	Requests that the CLD CM40x Bulk Library pause the current transfer. This causes the Bulk OUT endpoint to be nak'ed until the transfer is resumed by calling cld_cm40x_bulk_lib_resume_paused_bulk_out_transfer.	CLD_USB_TRANSFER_DISCARD	Requests that the CLD CM40x Bulk 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.	CLD_USB_TRANSFER_STALL	This notifies the CLD CM40x Bulk Library that there is an error and the Bulk OUT endpoint should be stalled.
Return Value	Description										
CLD_USB_TRANSFER_ACCEPT	Notifies the CLD CM40x Bulk Library that the Bulk OUT data should be accepted using the p_transfer_data values.										
CLD_USB_TRANSFER_PAUSE	Requests that the CLD CM40x Bulk Library pause the current transfer. This causes the Bulk OUT endpoint to be nak'ed until the transfer is resumed by calling cld_cm40x_bulk_lib_resume_paused_bulk_out_transfer.										
CLD_USB_TRANSFER_DISCARD	Requests that the CLD CM40x Bulk 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.										
CLD_USB_TRANSFER_STALL	This notifies the CLD CM40x Bulk Library that there is an error and the Bulk OUT endpoint should be stalled.										
usb_bus_max_power	USB Configuration Descriptor bMaxPower value (0 = self powered). Refer to the USB 2.0 protocol section 9.6.3.										
device_descriptor_bcd_device	USB Device Descriptor bcdDevice value. Refer to the USB 2.0 protocol section 9.6.1.										
p_usb_string_manufacturer	Pointer to the null-terminated string. This string is used by the CLD CM40x Bulk Library to generate the Manufacturer USB String Descriptor. If the Manufacturer String Descriptor is not used set p_usb_string_manufacturer to NULL.										
p_usb_string_product	Pointer to the null-terminated string. This string is used by the CLD CM40x Bulk Library to generate the Product USB String Descriptor. If the Product String Descriptor is not used set p_usb_string_product to NULL.										
p_usb_string_serial_number	Pointer to the null-terminated string. This string is used by the CLD CM40x Bulk 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 NULL.										

p_usb_string_configuration	Pointer to the null-terminated string. This string is used by the CLD CM40x Bulk Library to generate the Configuration USB String Descriptor. If the Configuration String Descriptor is not used set p_usb_string_configuration to NULL.												
p_usb_string_interface	Pointer to the null-terminated string. This string is used by the CLD CM40x Bulk Library to generate the Interface 0 USB String Descriptor. If the Product String Descriptor is not used set p_usb_string_interface to NULL.												
usb_string_language_id	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)												
fp_cld_usb_event_callback	<p>Function that is called when one of the following USB events occurs. This function has a single CLD_USB_Event parameter.</p> <p>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.</p> <p>The CLD_USB_Event has the following values:</p> <table border="1"> <thead> <tr> <th>Return Value</th> <th>Description</th> </tr> </thead> <tbody> <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</td> <td>USB device enumerated (USB Configuration set to a non-zero value)</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> </tbody> </table> <p>Note: Set to CLD_NULL if not required by application</p>	Return Value	Description	CLD_USB_CABLE_CONNECTED	USB Cable Connected.	CLD_USB_CABLE_DISCONNECTED	USB Cable Disconnected	CLD_USB_ENUMERATED_CONFIGURED	USB device enumerated (USB Configuration set to a non-zero value)	CLD_USB_UN_CONFIGURED	USB Configuration set to 0	CLD_USB_BUS_RESET	USB Bus reset received
Return Value	Description												
CLD_USB_CABLE_CONNECTED	USB Cable Connected.												
CLD_USB_CABLE_DISCONNECTED	USB Cable Disconnected												
CLD_USB_ENUMERATED_CONFIGURED	USB device enumerated (USB Configuration set to a non-zero value)												
CLD_USB_UN_CONFIGURED	USB Configuration set to 0												
CLD_USB_BUS_RESET	USB Bus reset received												

cld_cm40x_bulk_lib_transmit_bulk_in_data

CLD_USB_Data_Transmit_Return_Type **cld_cm40x_bulk_lib_transmit_bulk_in_data**
(CLD_USB_Transfer_Params * p_transfer_data)

CLD CM40x Bulk Library function used to send data 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_cm40x_bulk_lib_transmit_bulk_in_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);
    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 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
<i>fp_usb_in_transfer_complete</i>	Function called when the specified data has been transmitted to the USB host. This function pointer can be set to NULL if the User application doesn't want to be notified when the data has been transferred.
<i>fp_transfer_aborted_callback</i>	Function called if there is a problem transmitting the data to the USB Host. This function can be set to 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_cm40x_bulk_lib_resume_paused_bulk_out_transfer

void cld_cm40x_bulk_lib_resume_paused_bulk_out_transfer (void)

CLD CM40x Bulk Library function used to resume a paused Bulk OUT transfer.

Arguments

None

Return Value

None.

Details

The `cld_cm40x_bulk_lib_resume_paused_bulk_out_transfer` function is used to resume a Bulk OUT transfer that was paused by the `fp_bulk_out_data_received` function returning `CLD_USB_TRANSFER_PAUSE`. When called the `cld_cm40x_bulk_lib_resume_paused_bulk_out_transfer` function will call the User application's `fp_bulk_out_data_received` function passing the `CLD_USB_Transfer_Params` of the original paused transfer. The `fp_bulk_out_data_received` function can then chose to accept, discard, or stall the bulk out request.

cld_lib_usb_connect

void cld_lib_usb_connect (void)

CLD CM40x Bulk 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 CM40x Bulk Library has been initialized to connect the USB device to the Host.

cld_lib_usb_disconnect

void cld_lib_usb_disconnect (void)

CLD CM40x Bulk 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 CM40x Bulk Library has been initialized to disconnect the USB device to the Host.

cld_time_get

CLD_Time `cld_time_get(void)`

CLD CM40x Bulk 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 CM40x Bulk Library function used to measure the amount of time that has passed.

Arguments

time	A CLD_Time value returned by a <code>cld_time_get</code> function call.
------	---

Return Value

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

Details

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

cld_timer_1ms

```
void cld_timer_1ms(void)
```

CLD CM40x Bulk Library function 1ms tick function.

Arguments

None

Return Value

None.

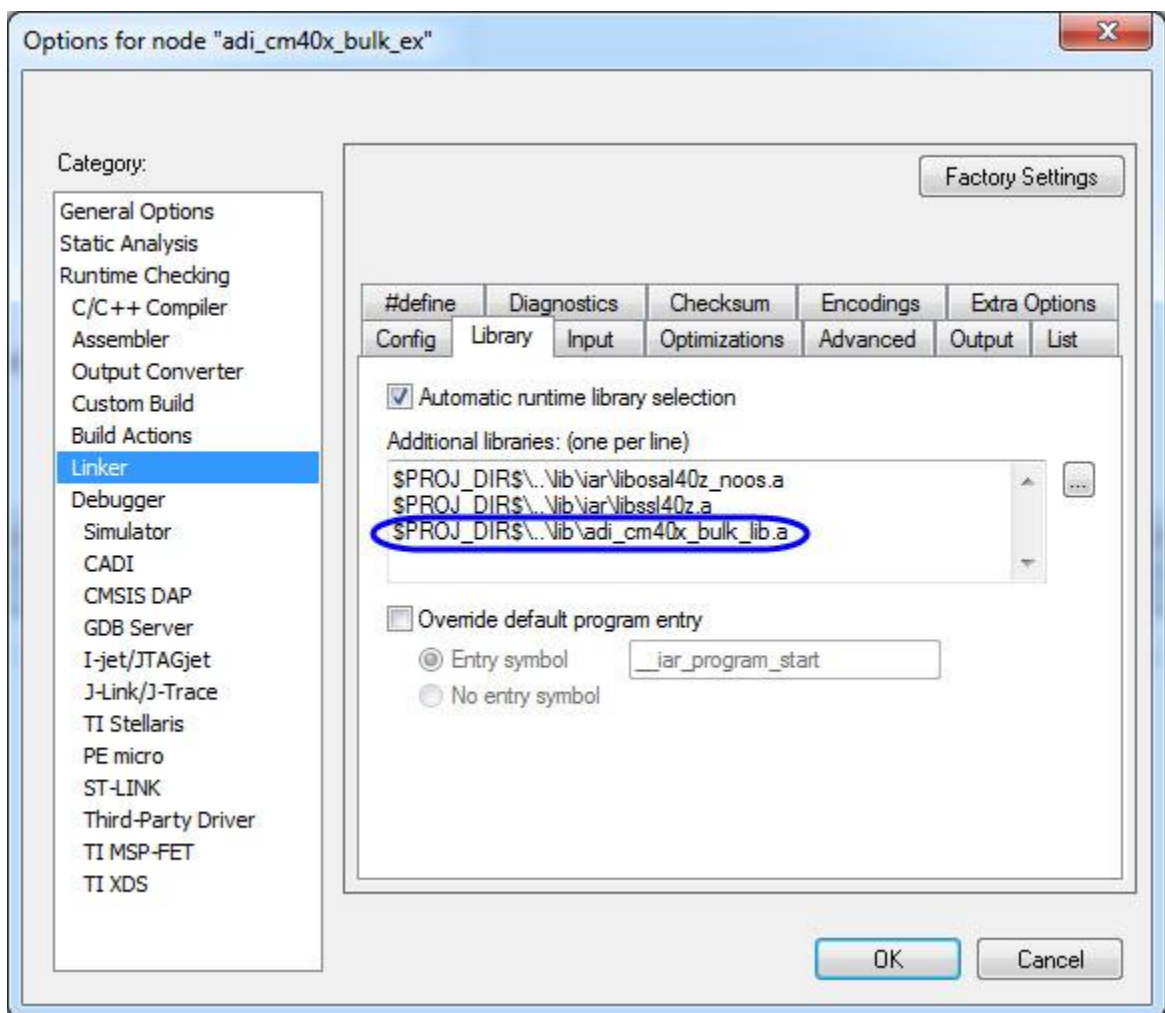
Details

The cld_timer_1ms function is used in 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.

Adding the CLD CM40x Bulk Library to an Existing Project

In order to include the CLD CM40x Bulk Library in an IAR project you must configure the project linker settings so it can locate the library. The following steps outline how this is done.

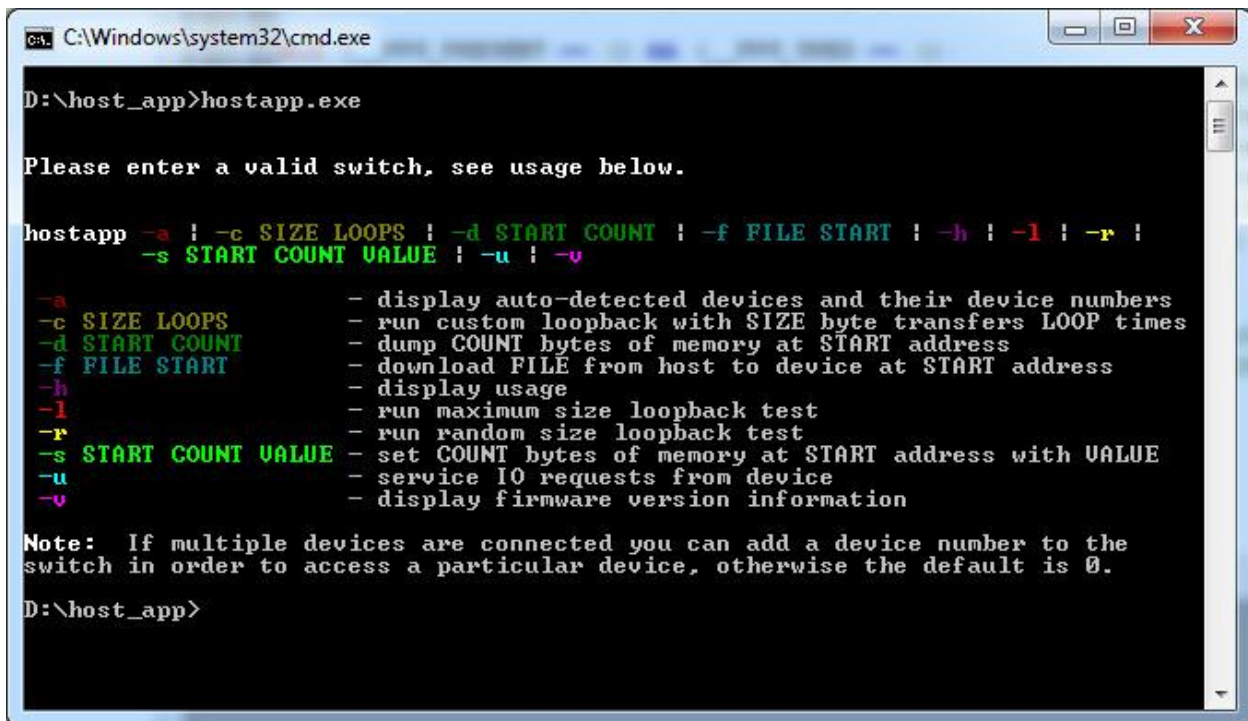
1. Copy the `cld_cm40x_bulk_lib.h` to the project's `src` directory.
2. Copy the `cld_cm40x_bulk_lib.a` to the project's `lib` directory.
3. Open the project in IAR Embedded Workbench.
4. Right click the project in the Files' window and select Add->Files. Add `cld_cm40x_bulk_lib.h` to the project. Include this header file in any source file that references the `cmd_cm40x_bulk_lib` functions.
5. Select Project->Options and select Linker. Click on the Library tab, and in the Additional Libraries box, click on '...' to Edit Additional Libraries. Navigate to the `lib` directory and select `cld_cm40x_bulk_lib.a`. Click OK, and then OK again.



Using ADI hostapp.exe

Analog Devices includes the hostapp application as part of the Examples for CM403F_EZ & CM408F_EZ KITS

To launch hostapp navigate to the above directory using the Windows DOS console (type cmd.exe in the Windows Run dialog box). Once there type hostapp.exe and press Enter to see a list of supported command switches as shown in the screen show below.



```
C:\Windows\system32\cmd.exe
D:\host_app>hostapp.exe

Please enter a valid switch, see usage below.

hostapp -a | -c SIZE LOOPS | -d START COUNT | -f FILE START | -h | -l | -r |
        -s START COUNT VALUE | -u | -v

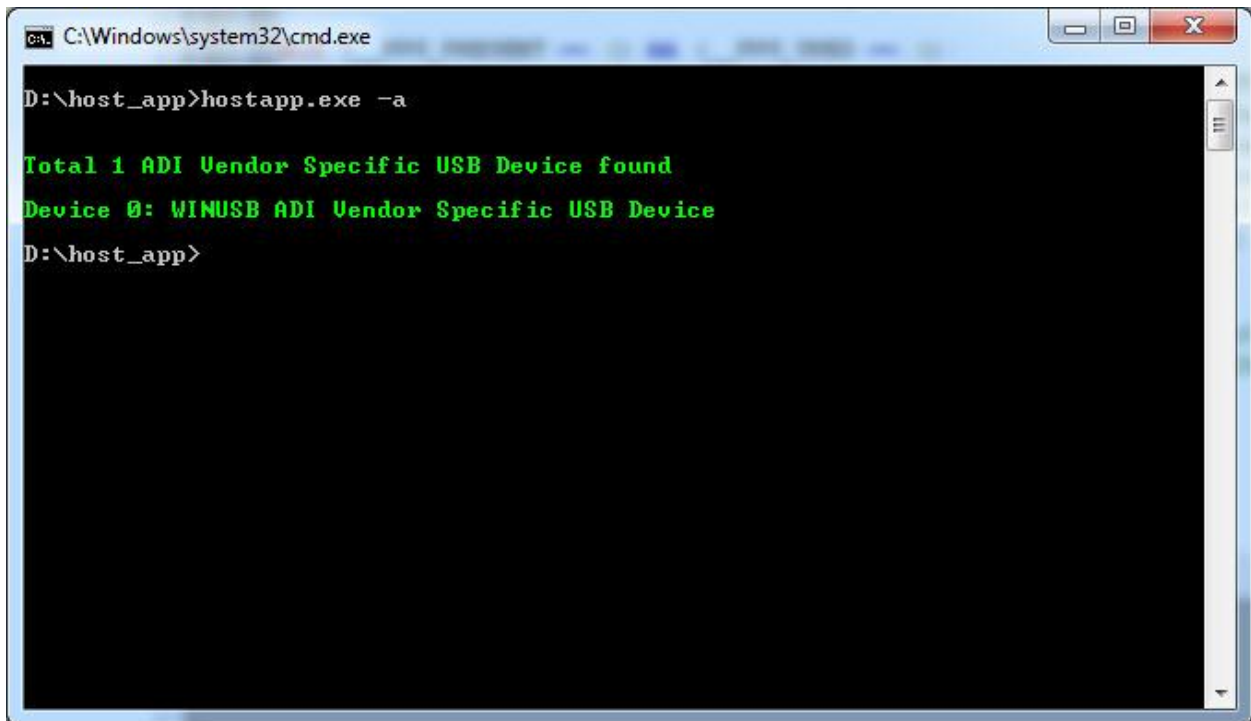
-a          - display auto-detected devices and their device numbers
-c SIZE LOOPS  - run custom loopback with SIZE byte transfers LOOP times
-d START COUNT - dump COUNT bytes of memory at START address
-f FILE START  - download FILE from host to device at START address
-h          - display usage
-l          - run maximum size loopback test
-r          - run random size loopback test
-s START COUNT VALUE - set COUNT bytes of memory at START address with VALUE
-u          - service IO requests from device
-v          - display firmware version information

Note: If multiple devices are connected you can add a device number to the
switch in order to access a particular device, otherwise the default is 0.

D:\host_app>
```

Note: The CLD Bulk Loopback Example supports all of the above command switches except for the '-u' switch.

Before going further connect the ADSP-CM408 EZ-Kit running the CLD Bulk Loopback Example and try running 'hostapp -a' to display the detected USB devices that support hostapp. If everything is working correctly you should see the following:



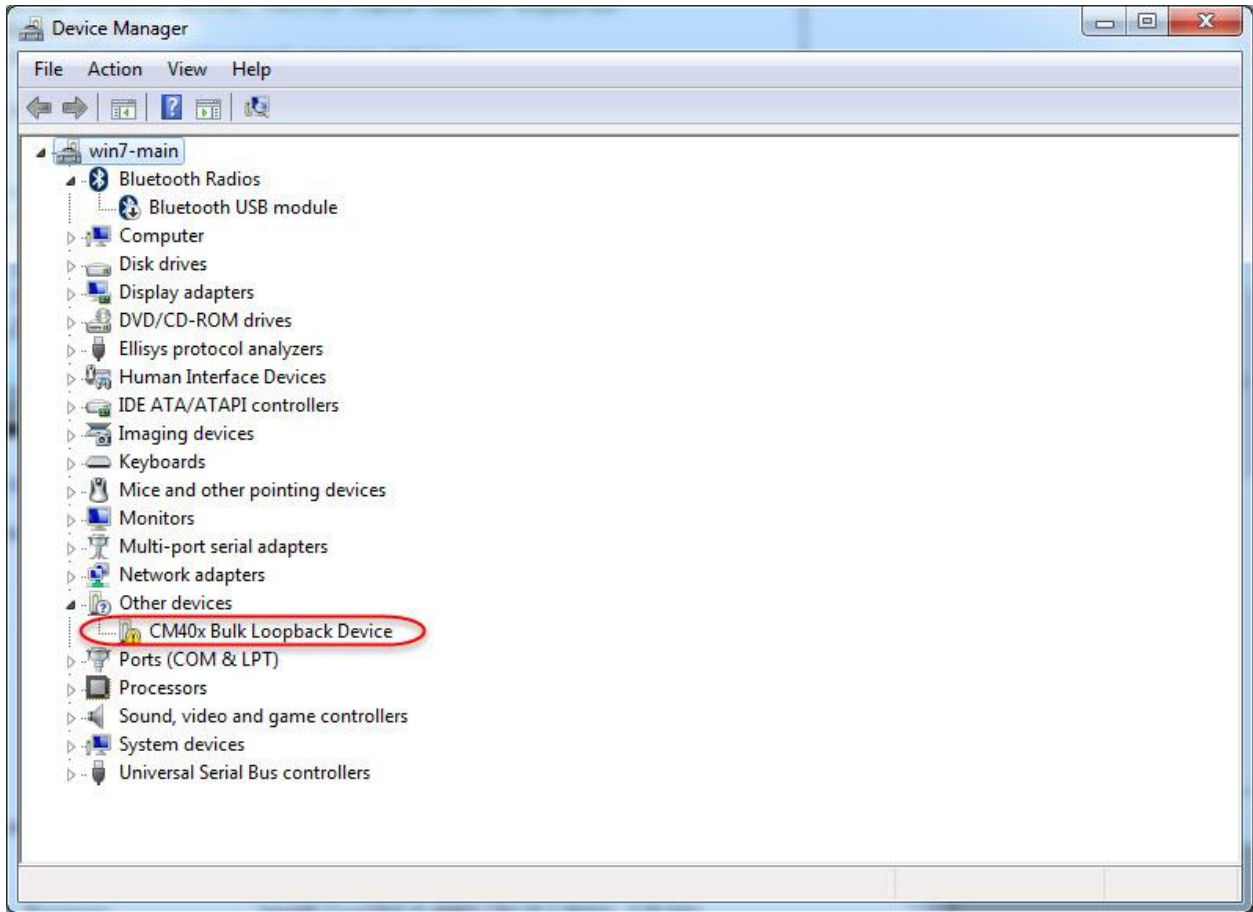
```
C:\Windows\system32\cmd.exe
D:\host_app>hostapp.exe -a
Total 1 ADI Vendor Specific USB Device found
Device 0: WINUSB ADI Vendor Specific USB Device
D:\host_app>
```

However, if hostapp.exe outputs "Total 0 ADI Vendor Specific USB Device found" it means that hostapp was not able to detect a hostapp compatible device. If this occurs first check to make sure the CLD Bulk Loop Back Example is running on the ADSP-CM408 EZ-Kit, and that you have a USB connected between the USB port and one of your PC USB ports. If this doesn't correct the problem the next step is to install the ADI hostapp USB driver as shown in the 'ADI hostapp USB Windows Driver Installation' section of this document.

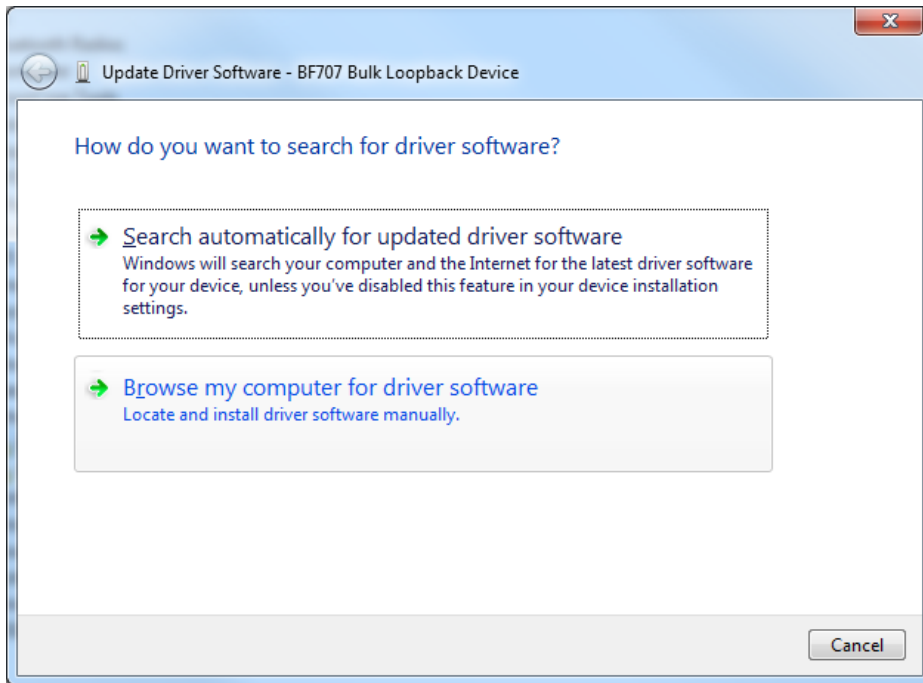
Once the USB driver has been installed you should be ready to run the remaining hostapp command switches (type hostapp.exe or hostapp -h to see the list of supported command switches).

ADI hostapp Windows USB Driver Installation

To install the ADI hostapp Windows USB driver open the Windows Device Manager by running "devmgmt.msc" from the Windows run dialog box. You should see a Device Manager window similar to the one below.

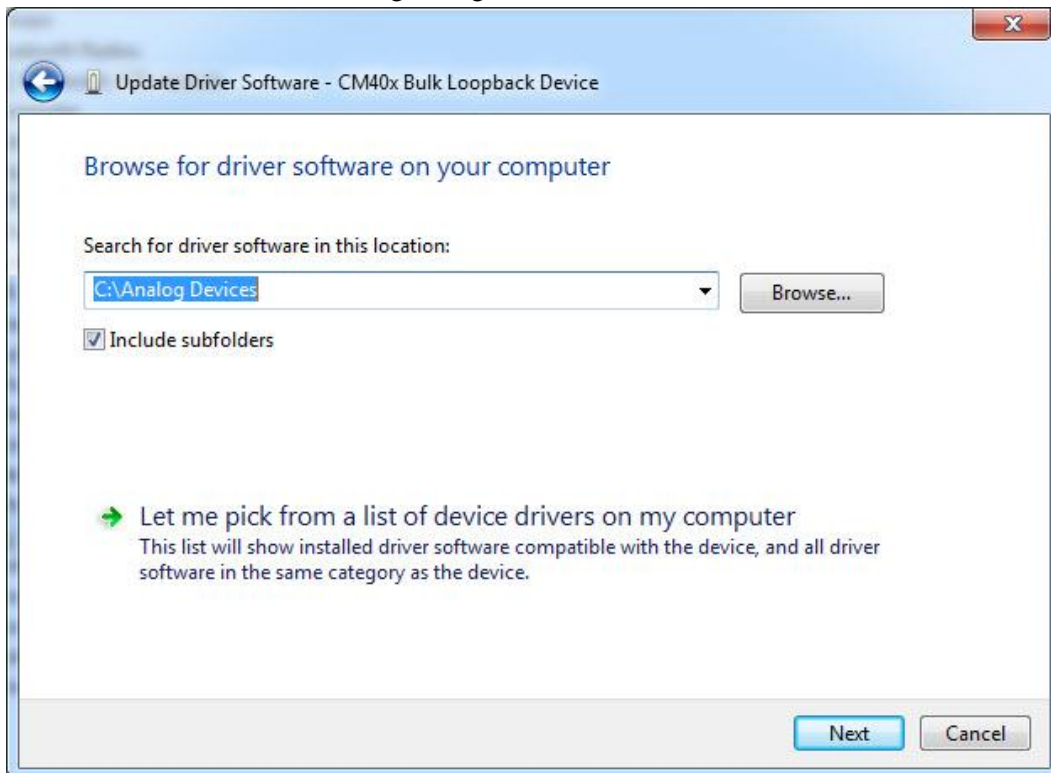


Notice the 'CM40x Bulk Loopback Device' circled in red. This is the CM408 running the CLD Bulk Loopback Example that is missing the ADI hostapp USB driver. To install the USB driver right click the 'CM40x Bulk Loopback Device' device and select Update Driver Software. You should now see the Update Driver Software dialog box shown below.



Click 'Browse my computer for driver software'

You should now see the following dialog box:

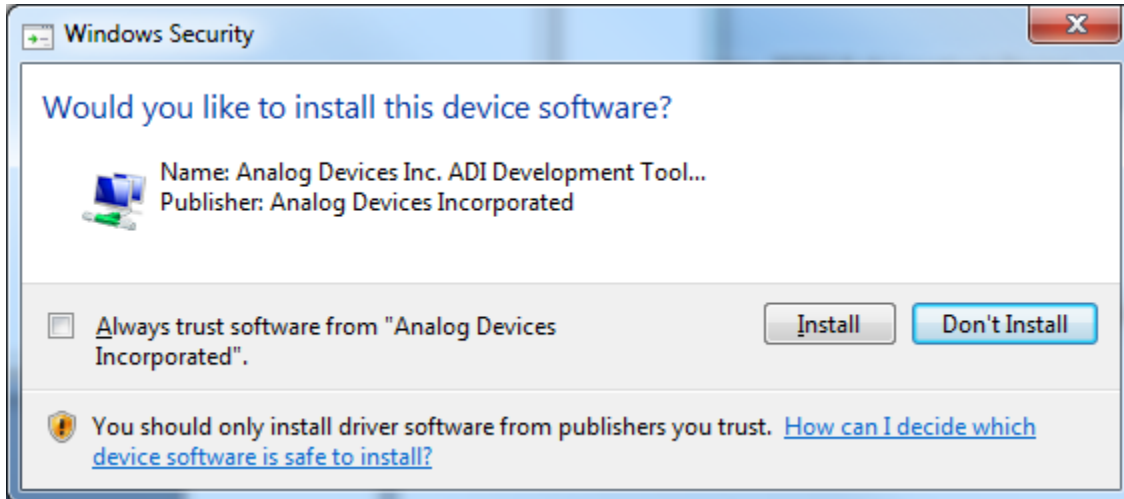


Click 'Browse...' and navigate to the Analog Devices Install directory containing the ADI hostapp USB driver shown below and click ok.

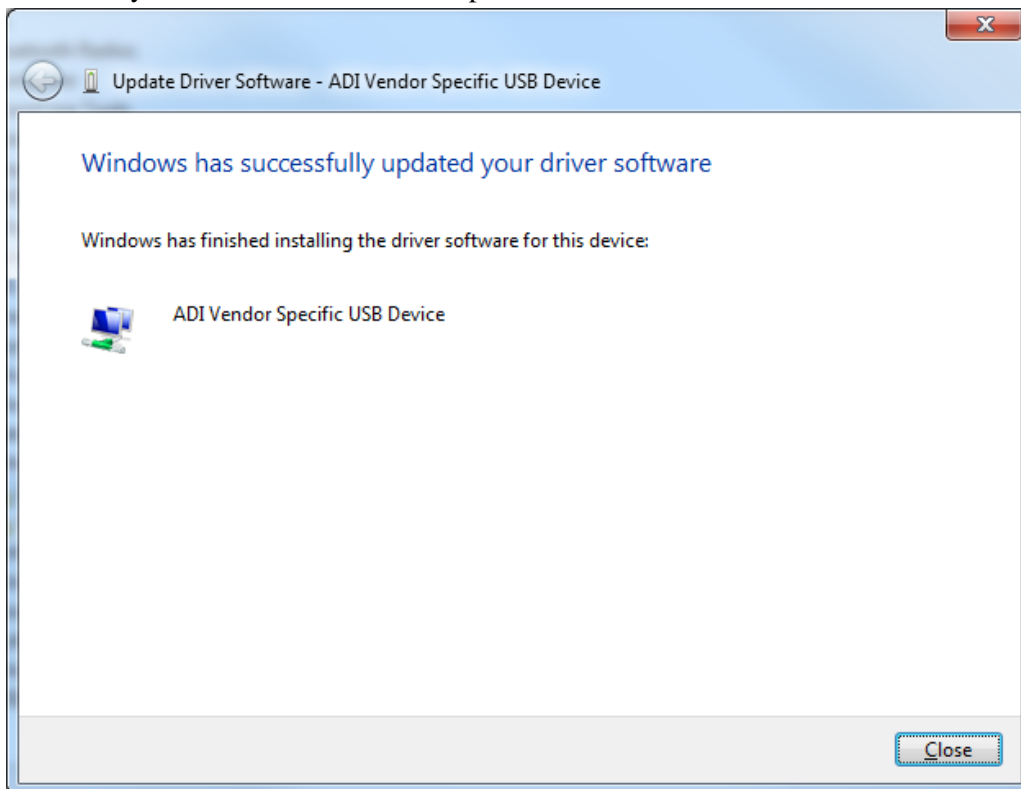
C:\Analog Devices

Click 'Next'

After clicking next you might see a Windows Security dialog box like the one shown below. If you do, click 'Install' to continue the driver installation.



You should now see the following dialog box showing that the ADI USB driver was installed successfully. Click 'Close' to exit the Update Driver Software wizard.



You should now be able to run `hostapp-a` and see that `hostapp` is now successfully detecting the CM408 running the CLD Bulk Loopback Example project.

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 CM40x Bulk Library please refer to the CLD_Bulk_loopback_Ex_v1_0 project included with the CLD CM40x Bulk Library. The CLD_Bulk_loopback_Ex_v1_0 project implements a Bulk IN/Bulk OUT device used by the Analog Devices hostapp.exe included with the Analog Devices CrossCore Embedded Studio.

main.c

```
void main(void)
{
    Main_States main_state = MAIN_STATE_USER_INIT;
    User_Bulk_Init_Return_Code rv;
    SystemInit();
    power_init();

    while (1)
    {
        switch (main_state)
        {
            case MAIN_STATE_USER_INIT:
                rv = user_bulk_init();
                if (rv == USER_BULK_INIT_SUCCESS)
                {
                    main_state = MAIN_STATE_RUN;
                }
                else if (rv == USER_BULK_INIT_FAILED)
                {
                    main_state = MAIN_STATE_ERROR;
                }
                break;

            case MAIN_STATE_RUN:
                break;

            case MAIN_STATE_ERROR:
                break;
        }
    }
}
```

user_bulk.c

```
/* Bulk IN endpoint parameters */
static CLD_Bulk_Endpoint_Params user_bulk_in_endpoint_params =
{
    .endpoint_number           = 1,
    .max_packet_size_full_speed = 64,
    .max_packet_size_high_speed = 512,
};

/* Bulk OUT endpoint parameters */
static CLD_Bulk_Endpoint_Params user_bulk_out_endpoint_params =
{
    .endpoint_number           = 1,
    .max_packet_size_full_speed = 64,
    .max_packet_size_high_speed = 512,
};

/* cld_cm40x_bulk_lib library initialization data. */
static CLD_CM40x_Bulk_Lib_Init_Params user_bulk_init_params =
{
    .sclk0                     = 60000000u,
    .fp_lib_msg_out            = user_console_rx_msg,
    .vendor_id                 = 0x064b,
    .product_id                = 0x7823

    .p_bulk_in_endpoint_params = &user_bulk_in_endpoint_params,
    .p_bulk_out_endpoint_params = &user_bulk_out_endpoint_params,
    .fp_bulk_out_data_received = user_bulk_bulk_out_data_received,
    .usb_bus_max_power         = 0,
    .device_descriptor_bcdDevice = 0x0100

    /* USB string descriptors - Set to CLD_NULL if not required */
    .p_usb_string_manufacturer = "Analog Devices Inc",
    .p_usb_string_product      = "CM40x Bulk Loopback Device",
    .p_usb_string_serial_number = CLD_NULL,
    .p_usb_string_configuration = CLD_NULL,
    .p_usb_string_interface    = "CM40x Bulk Loopback Demo",

    .usb_string_language_id    = 0x0409, /* English (US) language ID */
    .fp_cld_usb_event_callback = user_bulk_usb_event,
};
```

```
User_Bulk_Init_Return_Code user_bulk_init (void)
{
    static unsigned char user_init_state = 0;
    CLD_RV cld_rv = CLD_ONGOING;
    User_Bulk_Init_Return_Code init_return_code = USER_BULK_INIT_ONGOING;

    switch (user_init_state)
    {
        case 0:
            /* TODO: add any custom User firmware initialization */
            user_init_state++;
            break;
        case 1:
            /* Initialize the CLD CM40x Bulk Library */
            cld_rv = cld_cm40x_bulk_lib_init(&user_bulk_init_params);

            if (cld_rv == CLD_SUCCESS)
            {
                /* Connect to the USB Host */
            }
    }
}
```

```

        cld_lib_usb_connect();
        init_return_code = USER_BULK_INIT_SUCCESS;
    }
    else if (cld_rv == CLD_FAIL)
    {
        init_return_code = USER_BULK_INIT_FAILED;
    }
    else
    {
        init_return_code = USER_BULK_INIT_ONGOING;
    }
}
return init_return_code;
}

/* Function called when a bulk out packet is received */
static CLD_USB_Transfer_Request_Return_Type
    user_bulk_bulk_out_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 Bulk transfer complete callback function. */
    p_transfer_data->fp_callback.usb_out_transfer_complete = user_bulk_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 */;

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

/* The function below is an example if the bulk out transfer done callback specified
in the CLD_USB_Transfer_Params structure. */
static CLD_USB_Data_Received_Return_Type user_bulk_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)*/
}

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

static void user_bulk_usb_event (CLD_USB_Event event)
{
    switch (event)
    {
        case CLD_USB_CABLE_CONNECTED:
            /* TODO: Add any User firmware processed when a USB cable is connected. */
            break;
        case CLD_USB_CABLE_DISCONNECTED:
            /* TODO: Add any User firmware processed when a USB cable is
            disconnected.*/
            break;
        case CLD_USB_ENUMERATED_CONFIGURED:
            /* TODO: Add any User firmware processed when a Device has been
            enumerated.*/
            break;
        case CLD_USB_UN_CONFIGURED:
            /* TODO: Add any User firmware processed when a Device USB Configuration
            is set to 0.*/
            break;
        case CLD_USB_BUS_RESET:
            /* TODO: Add any User firmware processed when a USB Bus Reset occurs. */
            break;
    }
}
}

```

```

/* The following function will transmit the specified memory using
the Bulk IN endpoint. */
static user_bulk_transmit_bulk_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.fp_usb_in_transfer_complete = /* TODO: Set to User
                                                            callback function or
                                                            CLD_NULL */;
    transfer_params.callback.fp_transfer_aborted_callback = /* TODO: Set to User
                                                            callback function or
                                                            CLD_NULL */;
    p_transfer_params->transfer_timeout_ms = /* TODO: Set to desired timeout */;

    if (cld_cm40x_bulk_lib_transmit_bulk_in_data(&transfer_params) ==
        CLD_USB_TRANSMIT_SUCCESSFUL)
    {
        /* Bulk IN transfer initiated successfully */
    }
    else /* Bulk IN transfer was unsuccessful */
    {
    }
}

```