Embedded USB Host CP210x Class Driver User Guide

Version 1.10

For use with USBH CP210x Class Driver versions 1.01 and above

Date: 20-Apr-2016 16:24

Table of Contents

System Overview 3
Introduction 3
Feature Check 4
Packages and Documents 4
Packages 4
Documents 4
Source File List 6
API Header File 6
Configuration File 6
System File 6
Version File 6
Configuration Options 7
Application Programming Interface 8
Module Management Functions 8
usbh_cp210x_init 9
usbh_cp210x_start 10
usbh_cp210x_stop 11
usbh_cp210x_delete 12
Device Management Functions 13
usbh_cp210x_send 14
usbh_cp210x_get_send_state 15
usbh_cp210x_receive 16
usbh_cp210x_get_status 17
usbh_cp210x_set_control_line_state 18
usbh_cp210x_set_line_coding 19
usbh_cp210x_get_port_hdl 20
usbh_cp210x_present 21
usbh_cp210x_register_ntf 22
Error Codes 23
Types and Definitions 24
t_usbh_ntf_fn 24
Notification Codes 24
Data Bit Codes 25
Modem Status Codes 25
Parity Codes 25
Stop Bit Codes 26
Integration 27
OS Abstraction Layer 27
PSP Porting 27
Sample Code 28
Initialization 29
Data Transfer 30

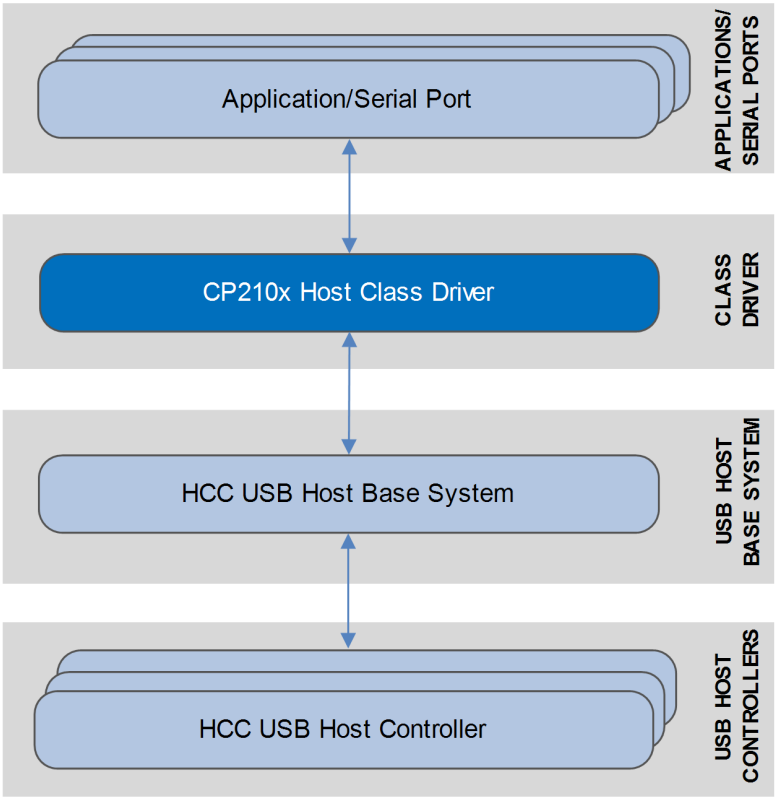
1 System Overview

1.1 Introduction

This guide is for those who want to implement an embedded USB host class driver to control Silicon Labs® CP2102, CP2103, CP2105 and CP2109 USB devices.

The CP210x devices provide a Universal Asynchronous Receiver/Transmitter (UART) over a USB bridge, such that a real or virtual UART port can be accessed at the device end of the USB connection. This UART port can be accessed through the class driver API as if it is a local UART on the host microcontroller.

The system structure is shown in the diagram below:



The lower layer interface is designed to use HCC Embedded's USB Host Interface Layer. This layer is standard over different host controller implementations; this means that the code is unchanged, whichever HCC USB host controller it is interfaced to. For detailed information about this layer, see the *HCC USB Host Base System User Guide* that is shipped with the base system.

The package provides a set of API functions for controlling access to a device. These are described here, with separate sections for module and device management.

Sample code in this guide shows how to initialize the system and perform data transfer.

1.2 Feature Check

The main features of the class driver are the following:

- It conforms to the HCC Advanced Embedded Framework.
- It can be used with or without an RTOS.
- It is compatible with all HCC USB host controllers.
- It supports Silicon Labs® CP2102, CP2103, CP2105 and CP2109 devices.
- It supports multiple devices connected simultaneously.
- It uses a system of callbacks for user-specified events.

1.3 Packages and Documents

Packages

The table below lists the packages that you need in order to use this module:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
usbh_base	The USB host base package. This is the framework used by USB class drivers to communicate over USB using a specific USB host controller package.
usbh_cd_cp210x	The USB device CP210x host class driver package described by this document.

Documents

For an overview of HCC's embedded USB stacks, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC USB Host Base System User Guide

This document defines the USB host base system upon which the complete USB stack is built.

HCC Embedded USB Host CP210x Class Driver User Guide

This is this document.

2 Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

> **Note:** Do not modify any files except the configuration file.

2.1 API Header File

The file **src/api/api_usbh_cp210x.h** should be included by any application using the system. This is the only file that should be included by an application using this module. For details of the API functions, see Application Programming Interface.

2.2 Configuration File

The file **src/config/config_usbh_cp210x.h** contains all the configurable parameters of the system. Configure these as required. For details of these options, see Configuration Options.

2.3 System File

The code file is **src/usb-host/class-drivers/cp210x/usbh_cp210x.c**. **This file should only be modified by HCC**.

2.4 Version File

The file **src/version/ver_usbh_cp210x.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

3 Configuration Options

Set the system configuration options in the file **src/config/config_usbh_cp210x.h**.

USBH_CP210X_MAX_UNITS

The maximum number of units the system can handle. This comprises the maximum number of units per interface and the number of devices connected. The default is 1.

USBH_CP210X_RXBUF_SIZE

The size of the receive buffer. The default is 512.

The minimum size for fulll speed systems is 64 and for high speed systems it is 512. The buffer size should always be a multiple of 64 or 512, respectively.

4 Application Programming Interface

This section documents the Application Programming Interface (API). It includes all the functions that are available to an application program.

4.1 Module Management Functions

The functions are the following:

Function	Description
usbh_cp210x_init()	Initializes the module and allocates the required resources.
usbh_cp210x_start()	Starts the module.
usbh_cp210x_stop()	Stops the module.
usbh_cp210x_delete()	Deletes the module and releases the resources it used.

usbh_cp210x_init

Use this function to initialize the class driver and allocate the required resources.

> **Note:** You must call this before any other function.

Format

```
int usbh_cp210x_init ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cp210x_start

Use this function to start the class driver.

> **Note:** You must call **usbh_cp210x_init()** before this function.

Format

```
int usbh_cp210x_start ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cp210x_stop

Use this function to stop the class driver.

Format

```
int usbh_cp210x_stop ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cp210x_delete

Use this function to delete the class driver and release the associated resources.

Format

```
int usbh_cp210x_delete ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

4.2 Device Management Functions

The functions are the following:

Function	Description
usbh_cp210x_send()	Sends data over the CP210x serial channel.
usbh_cp210x_get_send_state()	Gets the state of the last send.
usbh_cp210x_receive()	Receives data on the CP210x serial channel.
usbh_cp210x_get_status()	Gets the modem status.
usbh_cp210x_set_control_line_state()	Configures the control line.
usbh_cp210x_set_line_coding()	Sets the line coding (baud rate, data and stop bits, and parity).
usbh_cp210x_get_port_hdl()	Gets the port handle.
usbh_cp210x_present()	Checks whether a CP210x device is connected.
usbh_cp210x_register_ntf()	Registers a notification function for a specified event type.

usbh_cp210x_send

Use this function to send data over the CP210x serial channel.

In RTOS Mode the call returns immediately if a notification function has previously been registered for the USBH_NTF_CP210X_TX event. If this function has not been registered the call returns when the transfer completes.

In non-RTOS Mode, before using this function you must register a TX notification function by using **usbh_cp210x_register_ntf()**. This is called when the transfer completes; see USBH_NTF_CP210X_TX.

Note:

- **usbh_cp210x_send()** cannot be called from the notification function.
- The notification function can be used to wake up a task that calls **usbh_cp210x_send()** or to send an event to a task that calls **usbh_cp210x_send().**

Format

```
int usbh_cp210x_send (
    t_usbh_unit_id   uid,
    uint8_t *        p_buf,
    uint32_t         length )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
p_buf	A pointer to the buffer.	uint8_t *
length	The number of bytes to send.	uint32_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cp210x_get_send_state

Use this function to get the state of the last send.

A notification function can call this function to find whether there was an error during transmission, or whether the transfer was successful.

Format

```
int usbh_cp210x_get_send_state ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
The completion code of the last send.	Successful execution.
Else	See Error Codes.

usbh_cp210x_receive

Use this function to receive data on the CP210x serial channel.

This call returns immediately, even if nothing is received.

You may register an RX notification function by using **usbh_cp210x_register_ntf()**. This will be called when at least one character is received; see USBH_NTF_CP210X_RX.

Note:

- **usbh_cp210x_receive()** cannot be called from the notification function.
- The notification function can be used to wake up a task that calls **usbh_cp210x_receive()**, or to send an event to a task that calls **usbh_cp210x_receive()**.

Format

```
int usbh_cp210x_receive (
    t_usbh_unit_id   uid,
    uint8_t *        p_buf,
    uint32_t         max_length,
    uint32_t *       rlength )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
p_buf	A pointer to the buffer to hold incoming data.	uint8_t *
max_length	The maximum length of the receive buffer.	uint32_t
rlength	The number of bytes received.	uint32_t *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cp210x_get_status

Use this function to get the modem status.

The status values are returned as masks.

Format

```
int usbh_cp210x_get_status (
    t_usbh_unit_id   uid,
    uint8_t *        p_modem_status )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
p_modem_status	Where to write the modem status.	uint8_t *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cp210x_set_control_line_state

Use this function to configure the control line.

Format

```
int usbh_cp210x_set_control_line_state (
    t_usbh_unit_id   uid,
    uint8_t          rts,
    uint8_t          dtr,
    uint8_t          use_rts,
    uint8_t          use_dtr )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
rts	The Request to Send (RTS) state.	uint8_t
dtr	The Data Terminal Ready (DTR) state.	uint8_t
use_rts	Use RTS state.	uint8_t
use_dtr	Use DTR state.	uint8_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cp210x_set_line_coding

Use this function to set the line coding.

Format

```
int usbh_cp210x_set_line_coding (
    t_usbh_unit_id   uid,
    uint32_t         br,
    uint8_t          b,
    uint8_t          p,
    uint8_t          s )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
br	The baud rate.	uint32_t
b	The Data bits.	uint8_t
p	The Parity code.	uint8_t
s	The Stop bits.	uint8_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cp210x_get_port_hdl

Use this function to get the port handle.

Format

```
t_usbh_port_hdl usbh_cp210x_get_port_hdl ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
The port handle.	Successful execution.
USBH_PORT_HDL_INVALID	Invalid port handle.
Else	See Error Codes.

usbh_cp210x_present

Use this function to check whether a CP210x device is connected.

Format

```
int usbh_cp210x_present ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
0	No CP210x device is present.
1	A CP210x device is present.
Else	See Error Codes.

usbh_cp210x_register_ntf

Use this function to register a notification function for a specified event type.

When a device is connected or disconnected, or one of the specific events for this type of device occurs, the notification function is called.

> **Note:** It is the user's responsibility to provide any notification functions required by the application. Providing such functions is optional.

Format

```
int usbh_cp210x_register_ntf (
    t_usbh_unit_id   uid,
    t_usbh_ntf       ntf,
    t_usbh_ntf_fn    ntf_fn )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
ntf	The notification ID.	t_usbh_ntf
ntf_fn	The notification function to use when an event occurs.	t_usbh_ntf_fn

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

4.3 Error Codes

If a function executes successfully it returns with a USBH_SUCCESS code, a value of 0. The following table shows the meaning of the error codes:

Return Code	Value	Description
USBH_SUCCESS	0	Successful execution.
USBH_SHORT_PACKET	1	IN transfer completed with short packet.
USBH_PENDING	2	Transfer still pending.
USBH_ERR_BUSY	3	Another transfer in progress.
USBH_ERR_DIR	4	Transfer direction error.
USBH_ERR_TIMEOUT	5	Transfer timed out.
USBH_ERR_TRANSFER	6	Transfer failed to complete.
USBH_ERR_TRANSFER_FULL	7	Cannot process more transfers.
USBH_ERR_SUSPENDED	8	Host controller is suspended.
USBH_ERR_HC_HALTED	9	Host controller is halted.
USBH_ERR_REMOVED	10	Transfer finished due to device removal.
USBH_ERR_PERIODIC_LIST	11	Periodic list error.
USBH_ERR_RESET_REQUEST	12	Reset request during enumeration.
USBH_ERR_RESOURCE	13	OS resource error.
USBH_ERR_INVALID	14	Invalid identifier/type (HC, EP HDL, and so on).
USBH_ERR_NOT_AVAILABLE	15	Item not available.
USBH_ERR_INVALID_SIZE	16	Invalid size.
USBH_ERR_NOT_ALLOWED	17	Operation not allowed.
USBH_ERROR	18	General error.

4.4 Types and Definitions

t_usbh_ntf_fn

The **t_usbh_ntf_fn** definition specifies the format of the notification function. It is defined in the USB host base system in the file **api_usb_host.h**.

Format

```
int  ( * t_usbh_ntf_fn )(
    t_usbh_unit_id   uid,
    t_usbh_ntf       ntf )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
ntf	The notification code.	t_usbh_ntf

Notification Codes

The standard notification codes shown below are defined in the USB host base system in the file **api_usb_host.h**.

Notification	Description
USBH_NTF_CONNECT	Connection notification code.
USBH_NTF_DISCONNECT	Disconnection notification code.

The additional notification codes provided by this module are as follows:

Notification	Description
USBH_NTF_CP210_RX	Data has been received.
USBH_NTF_CP210_TX	Data has been successfully transmitted.

Data Bit Codes

The data bit codes are as follows:

Code	Value
USBH_CP210_BITS_5	5
USBH_CP210_BITS_6	6
USBH_CP210_BITS_7	7
USBH_CP210_BITS_8	8

Modem Status Codes

These modem status codes may be returned by the **usbh_cp210_get_status()** function.

Code	Value	Description
USBH_CP210_DTR	1U << 0	Data Terminal Ready (DTR).
USBH_CP210_RTS	1U << 1	Request to Send (RTS).
USBH_CP210_CTS	1U << 4	Clear to Send (CTS).
USBH_CP210_DSR	1U << 5	Data Set Ready (DSR).
USBH_CP210_RI	1U << 6	Ring Indicator (RI).
USBH_CP210_DCD	1U << 7	Receive line signal detect.

Parity Codes

The parity codes are as follows:

Code	Value
USBH_CP210X_PARITY_NONE	0
USBH_CP210X_PARITY_ODD	1
USBH_CP210X_PARITY_EVEN	2
USBH_CP210X_PARITY_MARK	3
USBH_CP210X_PARITY_SPACE	4

Stop Bit Codes

The stop bit codes are as follows:

Code	Value
USBH_CP210X_STOP_1	0
USBH_CP210X_STOP_1_5	1
USBH_CP210X_STOP_2	2

5 Integration

This section specifies the elements of this package that need porting, depending on the target environment.

5.1 OS Abstraction Layer

All HCC modules use the OS Abstraction Layer (OAL) that allows the module to run seamlessly with a wide variety of RTOSes, or without an RTOS.

The class driver uses the following OAL components:

OAL Resource	Number Required
Tasks	0
Mutexes	1
Events	0

5.2 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer.

The module makes use of the following standard PSP function:

Function	Package	Element	Description
psp_memset()	psp_base	psp_string	Sets the specified area of memory to the defined value.

The class driver makes use of the following standard PSP macros:

Macro	Package	Component	Description
PSP_WR_LE16	psp_base	psp_endianness	Writes a 16 bit value stored as little-endian to a memory location.
PSP_WR_LE32	psp_base	psp_endianness	Writes a 32 bit value stored as little-endian to a memory location.

6 Sample Code

This section gives example code for the class driver.

6.1 Initialization

This example shows the code used to initialize a USB host with the CP210x class driver.

```
/* Initialize USB host with CP210x class driver  */

int usb_host_init ( void )
{
    int  rc;
    rc = hcc_mem_init();

    /* Initialize the USB host module */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_init();
    }

    /* Initialize the specific USB host controller */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_hc_init( 0, usbh_cp210x_hc, 0 );
    }

    /* Initialize the CP210x Class driver module */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_cp210x_init();
    }

    /* Start the CP210x Class driver */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_cp210x_start();
    }

    /* Set line coding for the specified CP210x serial line */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_cp210x_set_line_coding( 0, 115200, USBH_CP210X_BITS_8,
USBH_CP210X_PARITY_NONE, USBH_CP210X_STOP_1 );
    }

    /* Start the USB host stack */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_start();  /* Start the USB host */
    }

    return rc;

} /* usb_host_init */
```

6.2 Data Transfer

This example shows the code of a simple CP210x class driver data handler.

```
/*
** Simple CP210x class driver data handler
** usbh_cp210x_echo()
*  Send back data received by CP210x host.
*/

void usbh_cp210x_echo ( void )

{
    uint32_t  len;
    int  rc;

    /* CP210x echo demo */

    if ( usbh_cp210x_present( 0 ) )    /* Check that device is present */
    {
        rc = usbh_cp210x_receive( 0, dbuf, BUF_SIZE, &len );  /* Wait for data */
        if ( ( rc == USBH_SUCCESS ) && ( len != 0 ) )
        {
            usbh_cp210x_send( 0, dbuf, len );   /* Echo data back to device */
        }
    }

}
```