Embedded USB Host CDC-ECM Class Driver User Guide

Version 2.00

For use with USBH CDC-ECM Class Driver versions 2.08 and above

Table of Contents

1. System Overview 4

1.1. Introduction 5

1.2. Feature Check 6

1.3. Packages and Documents 7

1.4. Change History 8

2. Source File List 9

3. Configuration Options 10

4. Application Programming Interface 11

4.1. Module Management Functions 11

usbh_cdcecm_init 12

usbh_cdcecm_start 13

usbh_cdcecm_stop 14

usbh_cdcecm_delete 15

4.2. Device Management Functions 16

usbh_cdcecm_send 17

usbh_cdcecm_receive 18

usbh_cdcecm_add_buf 20

usbh_cdcecm_get_send_state 21

usbh_cdcecm_get_receive_state 22

usbh_cdcecm_rx_chars 23

usbh_cdcecm_get_connection_speed 24

usbh_cdcecm_get_eth_statistics 25

usbh_cdcecm_get_mac_address 26

usbh_cdcecm_get_max_segment_size 27

usbh_cdcecm_get_network_state 28

usbh_cdcecm_get_port_hdl 29

usbh_cdcecm_present 30

usbh_cdcecm_set_eth_mcast_filters 31

usbh_cdcecm_set_eth_packet_filter 32

usbh_cdcecm_register_ntf 33

4.3. Error Codes 34

4.4. Types and Definitions 35

t_usbh_ntf_fn 35

Notification Codes 35

Ethernet Packet Filters 36
Ethernet Statistics 36
5. Integration 38
5.1. OS Abstraction Layer 38
5.2. PSP Porting 38
6. Sample Code 39
6.1. Initialization 39
7. Version 40

1. System Overview

This chapter contains the fundamental information for this module.

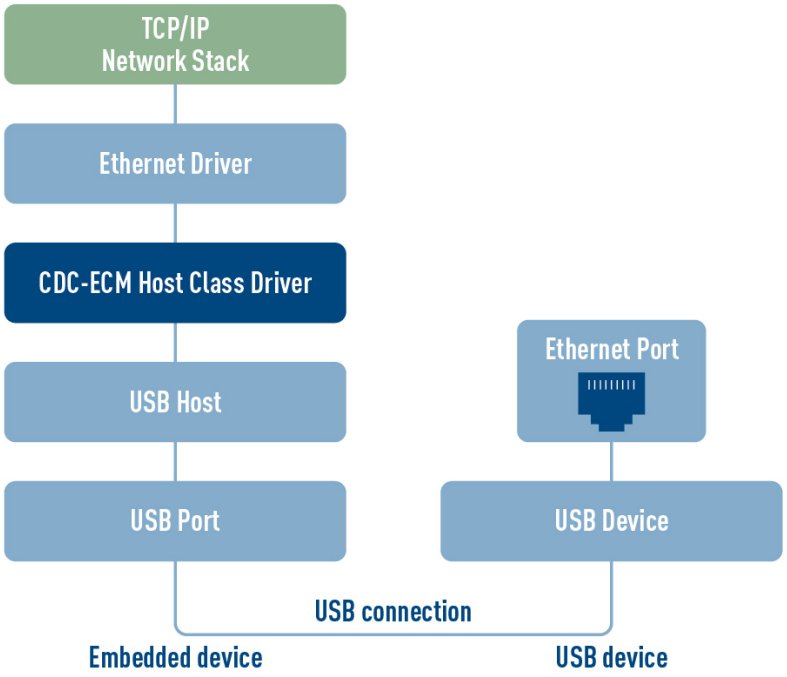
The component sections are as follows:

- Introduction – describes the main elements of the module.
- Feature Check – summarizes the main features of the module as bullet points.
- Packages and Documents – the *Packages* section lists the packages that you need in order to use this module. The *Documents* section lists the relevant user guides.
- Change History – lists the earlier versions of this manual, giving the software version that each manual describes.

1.1. Introduction

This guide is for those who want to implement an embedded CDC-ECM (Control Device Class - Ethernet Control Module) host class driver. This presents the USB ECM device to the system as a network adapter.

The HCC CDC-ECM package provides a host class driver for a USB stack. The system structure is shown below:



The host implementation allows the host system to transfer Ethernet packets to/from a remote Ethernet port at the other end of the USB connection. It connects to a USB device that has an Ethernet port on it that connects to a real or virtual network.

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 the same, whichever HCC USB host controller it is interfaced to. For detailed information about this layer, refer to 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.

1.2. Feature Check

The main features of the class driver are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Designed for integration with both RTOS and non-RTOS based systems.
- Compatible with all HCC USB host controllers.
- Supports all devices that conform to the USB CDC-ECM specification.
- Supports multiple devices connected simultaneously.
- 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_cdc_ecm	The USB device CDC-ECM host class driver package described by this document.
nw_drv_base	The network driver base package.
nw_drv_eth_usbh_cdcecm	The network driver for USB Host CDC-ECM (required if used with the TCP/IP stack).

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.

Network Driver for CDC-ECM User Guide

This document defines the USB host CDC-ECM network driver that may need to be used with the class driver.

HCC Embedded USB Host CDC-ECM Class Driver User Guide

This is this document.

1.4. Change History

This section describes past changes to this manual.

- To download this manual or a PDF describing an earlier software version, see USB Host PDFs.
- For the history of changes made to the package code itself, see History: usbh_cd_cdc_ecm.

The current version of this manual is 2.00. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
2.00	2020-08-11	2.10	New document format.
1.30	2018-10-16	2.08	Corrected *Packages* list. Added USBH_CDCECM_EXTERNAL_BUF option and **usbh_cdcecm_add_buf()**. Updated *PSP Porting*.
1.20	2017-06-19	2.06	New *Change History* format.
1.10	2016-04-20	2.06	Added function group descriptions to API.
1.00	2015-12-07	2.05	First release.

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.

API Header File

The file **src/api/api_usbh_cdcecm.h** must 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.

Configuration File

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

Source Code File

The file **src/usb-host/class-drivers/cdc-ecm/usbh_cdcecm.c** contains the main code for the class driver. **This file should only be modified by HCC**.

Version File

The file **src/version/ver_usbh_cdcecm.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_cdcecm.h**. This section lists the available configuration options and their default values.

USBH_CDCECM_MAX_UNITS

The maximum number of CDC-ECM units the system can handle. This includes the maximum units per interface and the number of connected devices. The default is 1.

USBH_CDCECM_EXTERNAL_BUF

Keep the default of 1 to use an externally provided buffer or set this to 0 to use internal buffers. If this set to 1, the **usbh_cdcecm_add_buf()** function is used.

> **Note:** The following two options are only used if USBH_CDCECM_EXTERNAL_BUF is set to 0.

USBH_CDCECM_RXBUF_SIZE

The size of a receive buffer. The default is 2048.

- For full speed systems, the minimum size is 64 and the value must be a multiple of 64.
- For high speed systems, the minimum size is 512 and the value must be a multiple of 512.

USBH_CDCECM_RXBUF_COUNT

The number of receive buffers for receiving in the background. When a device is plugged in, reception starts automatically. When data arrives the user gets a notification or polls for data. In the meantime reception to the next buffer can be started to keep the performance as high as possible. The default is 2.

USBH_CDCECM_COMBUF_SIZE

The size of the COM (comms. interface) buffer. It is safe to keep this at the default of 64.

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_cdcecm_init()	Initializes the module and allocates the required resources.
usbh_cdcecm_start()	Starts the module.
usbh_cdcecm_stop()	Stops the module.
usbh_cdcecm_delete()	Deletes the module and releases the resources it used.

usbh_cdcecm_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_cdcecm_init ( void )
```

Arguments

None.

Return Values

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

usbh_cdcecm_start

Use this function to start the class driver.

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

Format

```
int usbh_cdcecm_start ( void )
```

Arguments

None.

Return Values

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

usbh_cdcecm_stop

Use this function to stop the class driver.

Format

```
int usbh_cdcecm_stop ( void )
```

Arguments

None.

Return Values

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

usbh_cdcecm_delete

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

Format

```
int usbh_cdcecm_delete ( void )
```

Arguments

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_cdcecm_send()	Sends data through a channel.
usbh_cdcecm_receive()	Receives available data.
usbh_cdcecm_add_buf()	Adds an external receive buffer (only used if USBH_CDCECM_EXTERNAL_BUF is set).
usbh_cdcecm_get_send_state()	Gets the completion code of the last send.
usbh_cdcecm_get_receive_state()	Gets the completion code of the last receive.
usbh_cdcecm_rx_chars()	Gets the number of characters in the receive buffer.
usbh_cdcecm_get_connection_speed()	Gets the current upstream and downstream connection speeds.
usbh_cdcecm_get_eth_statistics()	Gets Ethernet statistics.
usbh_cdcecm_get_mac_address()	Gets the MAC address of the ECM device.
usbh_cdcecm_get_max_segment_size()	Gets the maximum segment size.
usbh_cdcecm_get_network_state()	Gets the connection state of the network.
usbh_cdcecm_get_port_hdl()	Gets the port handle.
usbh_cdcecm_present()	Checks whether a channel is present.
usbh_cdcecm_set_eth_mcast_filters()	Sets Ethernet multicast filters.
usbh_cdcecm_set_eth_packet_filter()	Sets an Ethernet packet filter.
usbh_cdcecm_register_ntf()	Registers a notification function for a specified event type.

usbh_cdcecm_send

Use this function to send data through a channel.

Format

```
int usbh_cdcecm_send (
    t_usbh_unit_id   uid,
    uint8_t *        buf,
    uint32_t         length )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
buf	A pointer to the data to send.	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_cdcecm_receive

Use this function to receive available data.

Its format depends on the setting of USBH_CDCECM_EXTERNAL_BUF, which has two possible settings:

- 0 - internal buffers are used.
- 1 - an externally provided buffer is used. This is the default.

Format when using an externally provided buffer

```
int usbh_cdcecm_receive (
    t_usbh_unit_id   uid,
    uint8_t * *      pp_buf,
    uint32_t *       rlength )
```

Arguments when using an externally provided buffer

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
pp_buf	A pointer to the buffer that is to receive the data.	uint8_t * *
rlength	A pointer to the number of bytes written to the buffer.	uint32_t *

Format when using internal buffers

```
int usbh_cdcecm_receive (
    t_usbh_unit_id   uid,
    uint8_t *        buf,
    uint32_t         max_length,
    uint32_t *       rlength )
```

Arguments when using internal buffers

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
buf	A pointer to the buffer that is to receive the data.	uint8_t *
max_length	The maximum length of the receive buffer.	uint32_t
rlength	Where to put the number of bytes written to the buffer.	uint32_t *

Return Values

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

usbh_cdcecm_add_buf

Use this function to add an external receive buffer.

This function returns USBH_SUCCESS if there is no active reception in progress.

> **Note:** This is only used if USBH_CDCECM_EXTERNAL_BUF is set, meaning that an externally provided buffer is to be used.

Format

```
int usbh_cdcecm_add_buf (
    t_usbh_unit_id   uid,
    uint8_t          buf[],
    uint32_t         length,
    uint8_t          b_start
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
buf[]	The buffer to add.	uint8_t
length	The length of the buffer.	uint32_t
b_start	TRUE if the transfer has to be started immediately.	uint8_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution. There is no active reception in progress.
Else	See Error Codes.

usbh_cdcecm_get_send_state

Use this function to get the completion code of the last send.

> **Note:** This function is only required if an event is used for send.

Format

```
int usbh_cdcecm_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
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cdcecm_get_receive_state

Use this function to get the completion code of the last receive.

> **Note:** This is not required generally since **usbh_cdcecm_receive()** returns the same information with the possible available data. If events are not used, you can poll this function to find whether data is available or not.

Format

```
int usbh_cdcecm_get_receive_state ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

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

usbh_cdcecm_rx_chars

Use this function to get the number of characters in the receive buffer, if data is available.

Note: Only call this function if **usbh_cdcecm_get_receive_state()** indicates success, otherwise the result is not valid.

Format

```
uint32_t usbh_cdcecm_rx_chars ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
The number of bytes in the receive buffer.	Successful execution.
Else	See Error Codes.

usbh_cdcecm_get_connection_speed

Use this function to get the current upstream and downstream connection speeds.

Format

```
int usbh_cdcecm_get_connection_speed (
    t_usbh_unit_id   uid,
    uint32_t *       usbitrate,
    uint32_t *       dsbitrate )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
usbitrate	Where to write the upstream connection speed.	uint32_t *
dsbitrate	Where to write the downstream connection speed.	uint32_t *

Return Values

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

usbh_cdcecm_get_eth_statistics

Use this function to get Ethernet statistics.

Note: This call only returns a valid value if the requested statistic is supported by the ECM device.

Format

```
int usbh_cdcecm_get_eth_statistics (
    t_usbh_unit_id   uid,
    uint16_t         stat,
    uint32_t *       val )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
stat	The requested statistic, one of the Ethernet Statistics options.	uint16_t
val	Where to write the requested statistic.	uint32_t *

Return Values

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

usbh_cdcecm_get_mac_address

Use this function to get the MAC address of the ECM device.

Format

```
int usbh_cdcecm_get_mac_address (
    t_usbh_unit_id   uid,
    uint8_t *        mac_address )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
mac_address	Where to write the MAC address. This is a pointer to a 6 byte unsigned char array.	uint8_t *

Return Values

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

usbh_cdcecm_get_max_segment_size

Use this function to get the maximum segment size.

Format

```
int usbh_cdcecm_get_max_segment_size (
    t_usbh_unit_id   uid,
    uint16_t *       max_segment_size )
```

Arguments

Parameter	Description	Type
uid	The unit ID	t_usbh_unit_id
max_segment_size	Where to write the maximum segment size.	uint16_t *

Return Values

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

usbh_cdcecm_get_network_state

Use this function to get the connection state of the network.

Format

```
int usbh_cdcecm_get_network_state (
    t_usbh_unit_id   uid,
    uint8_t *        state )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
state	Where to write the state (1 = connected, 0 = disconnected).	uint8_t *

Return Values

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

usbh_cdcecm_get_port_hdl

Use this function to get the port handle.

Format

```
t_usbh_port_hdl usbh_cdcecm_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_cdcecm_present

Use this function to check whether a channel is present.

Format

```
int usbh_cdcecm_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 channel is present.
1	A channel is present.
Else	See Error Codes.

usbh_cdcecm_set_eth_mcast_filters

Use this function to set Ethernet multicast filters.

Format

```
int usbh_cdcecm_set_eth_mcast_filters (
    t_usbh_unit_id   uid,
    uint16_t         n,
    uint8_t *        filters )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
n	The number of filters.	uint16_t
filters	A list of 48 bit multicast filters in network byte order.	uint8_t *

Return Values

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

usbh_cdcecm_set_eth_packet_filter

Use this function to set an Ethernet packet filter.

Format

```
int usbh_cdcecm_set_eth_packet_filter (
    t_usbh_unit_id   uid,
    uint16_t         flags )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
flags	A bitmap of packet type filters from the Ethernet Packet Filters list.	uint16_t

Return Values

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

usbh_cdcecm_register_ntf

Use this function to register a notification function for the device.

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_cdcecm_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 be used 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

This section describes the *t_usbh_ntf_fn* and the main elements that are defined in the API Header file.

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

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

Notification Codes

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

Notification	Value	Description
USBH_NTF_CONNECT	1	Connection notification code.
USBH_NTF_DISCONNECT	2	Disconnection notification code.

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

Notification	Value	Description
USBH_NTF_CDCECM_RX	USBH_NTF_CD_BASE + 0	Data received notification.
USBH_NTF_CDCECM_TX	USBH_NTF_CD_BASE + 1	Data sent notification.
USBH_NTF_CDCECM_NTF	USBH_NTF_CD_BASE + 2	CDC notification.

Ethernet Packet Filters

The Ethernet packet filters are used by the **usbh_cdcecm_set_eth_packet_filter()** function:

Code	Value	Description
USBH_CDCECM_PACKET_TYPE_PROMISCUOUS	1U << 0	Promiscuous.
USBH_CDCECM_PACKET_TYPE_ALL_MULTICAST	1U << 1	Multicast all.
USBH_CDCECM_PACKET_TYPE_DIRECTED	1U << 2	Directed.
USBH_CDCECM_PACKET_TYPE_BROADCAST	1U << 3	Broadcast.
USBH_CDCECM_PACKET_TYPE_MULTICAST	1U << 4	Multicast.

Note: These are unique bits in a set of bit values.

Ethernet Statistics

These are the options for the *stat* parameter of the **usbh_cdcecm_get_eth_statistics()** function:

Option	Value	Description
USBH_CDCECM_XMIT_OK	1	Successful transmits.
USBH_CDCECM_RCV_OK	2	Successful receives.
USBH_CDCECM_XMIT_ERROR	3	Transmit errors.
USBH_CDCECM_RCV_ERROR	4	Receive errors.
USBH_CDCECM_RCV_NO_BUFFER	5	Lost because receive buffer full.
USBH_CDCECM_DIRECTED_BYTES_XMIT	6	Directed bytes sent.
USBH_CDCECM_DIRECTED_FRAMES_XMIT	7	Directed frames sent.
USBH_CDCECM_MULTICAST_BYTES_XMIT	8	Multicast bytes sent.
USBH_CDCECM_MULTICAST_FRAMES_XMIT	9	Multicast frames sent.
USBH_CDCECM_BROADCAST_BYTES_XMIT	10	Broadcast bytes sent.
USBH_CDCECM_BROADCAST_FRAMES_XMIT	11	Broadcast frames sent.
USBH_CDCECM_DIRECTED_BYTES_RCV	12	Directed bytes received.
USBH_CDCECM_DIRECTED_FRAMES_RCV	13	Directed frames received.
USBH_CDCECM_MULTICAST_BYTES_RCV	14	Multicast bytes received.
USBH_CDCECM_MULTICAST_FRAMES_RCV	15	Multicast frames received.
USBH_CDCECM_BROADCAST_BYTES_RCV	16	Broadcast bytes received.
USBH_CDCECM_BROADCAST_FRAMES_RCV	17	Broadcast frames received.

Option	Value	Description
USBH_CDCECM_RCV_CRC_ERROR	18	Receive CRC errors.
USBH_CDCECM_TRANSMIT_QUEUE_LENGTH	19	Length of transmit queue.
USBH_CDCECM_RCV_ERROR_ALIGNMENT	20	Receive alignment errors.
USBH_CDCECM_XMIT_ONE_COLLISION	21	Transmit single collisions.
USBH_CDCECM_XMIT_MORE_COLLISIONS	22	Transmit multiple collisions.
USBH_CDCECM_XMIT_DEFERRED	23	Transmit deferrals.
USBH_CDCECM_XMIT_MAX_COLLISIONS	24	Transmit collisions.
USBH_CDCECM_RCV_OVERRUN	25	Receive overruns.
USBH_CDCECM_XMIT_UNDERRUN	26	Transmit underruns.
USBH_CDCECM_XMIT_HEARTBEAT_FAILURE	27	Transmit heartbeat failures.
USBH_CDCECM_XMIT_TIMES_CRS_LOST	28	Transmit CRS lost.
USBH_CDCECM_XMIT_LATE_COLLISIONS	29	Transmit late collisions.

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 CDC-ECM 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. For full details of its functions and macros, see the *HCC Base Platform Support Package User Guide*.

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

Function	Package	Component	Description
psp_memcpy()	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.
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	Element	Description
PSP_RD_LE16	psp_base	psp_endianness	Reads a 16 bit value stored as little-endian from a memory location.
PSP_RD_LE32	psp_base	psp_endianness	Reads a 32 bit value stored as little-endian from a memory location.
PSP_WR_LE16	psp_base	psp_endianness	Writes a 16 bit value to be stored as little-endian to a memory location.

6. Sample Code

This section shows example code for the class driver.

6.1. Initialization

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

```
/*
** Initialize USB host with CDC-ECM class driver.
*/

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

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

    /* Initialize specific USB host controller */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_hc_init( 0, usbh_stm32uh_hc, 0 );
    }

    /* Initialize the CDC-ECM Class driver module */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_cdcecm_init();
    }

    /* Start the CDC-ECM Class driver */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_cdcecm_start();
    }

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

    return rc;

} /* usb_host_init */
```

7. Version

Version 2.00

For use with USBH CDC-ECM Class Driver versions 2.08 and above