

# USB EHCI Host Controller User's Guide

Version 1.20

For use with USBH EHCI Host Controller versions 3.09  
and above

**Date:** 27-Mar-2015 16:40

All rights reserved. This document and the associated software are the sole property of HCC Embedded. Reproduction or duplication by any means of any portion of this document without the prior written consent of HCC Embedded is expressly forbidden.

HCC Embedded reserves the right to make changes to this document and to the related software at any time and without notice. The information in this document has been carefully checked for its accuracy; however, HCC Embedded makes no warranty relating to the correctness of this document.

---

# Table of Contents

---

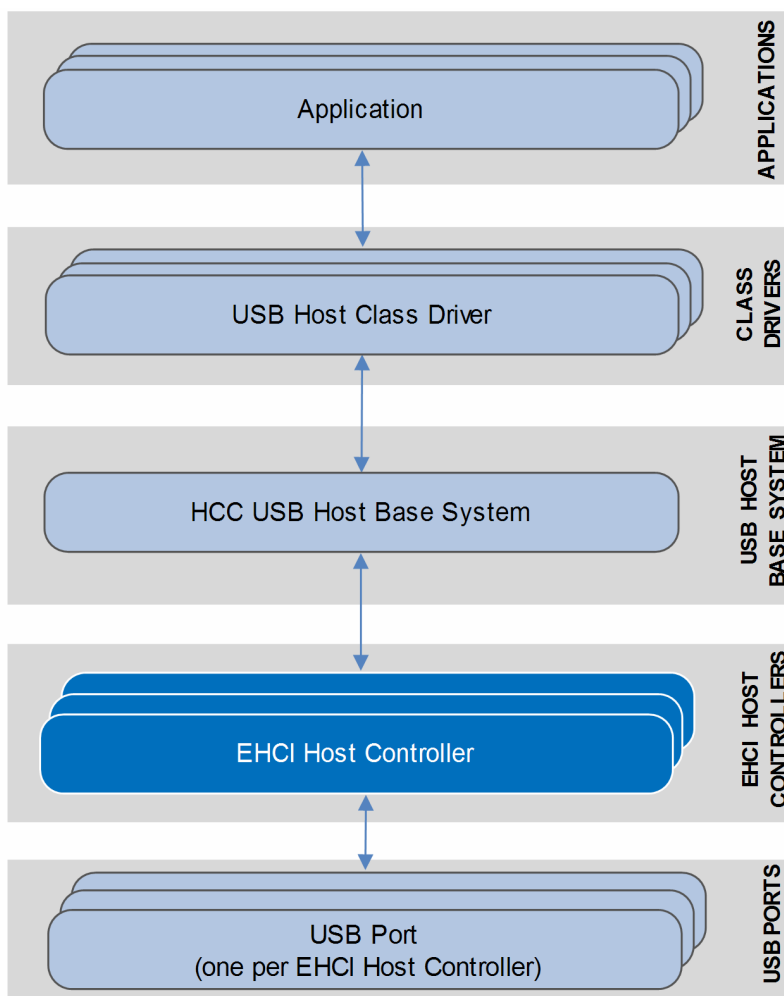
System Overview	3
Introduction	3
Feature Check	4
Packages and Documents	4
Packages	4
Documents	4
Change History	5
Source File List	6
API Header File	6
Configuration File	6
Source Code	6
Version File	6
PSP File	7
Configuration Options	8
Starting the EHCI Controller	11
usbh_ehci_hc	11
Host Controller Task	11
Code Example	12
Integration	13
OS Abstraction Layer (OAL)	13
PSP Porting	14
ehci_hw_init	15
ehci_hw_start	16
ehci_hw_stop	17
ehci_hw_delete	18
ehci_hw_suspend	19
ehci_hw_resume	20
ehci_hw_state	21
ehci_hw_connect	22
ehci_hw_disconnect	23

# 1 System Overview

## 1.1 Introduction

This guide is for those who want to configure and use HCC Embedded's Enhanced Host Controller Interface (EHCI) module with HCC's USB host stack. The EHCI module provides a high speed USB 2.0 host controller which provides both full speed and low speed USB functions. The controller can handle all USB transfer types and, in conjunction with the USB host stack, can be used with any USB class driver.

The EHCI Host Controller package provides a host controller for a USB stack, as shown below.



---

## 1.2 Feature Check

---

The main features of the host controller are the following:

- It conforms to the HCC Advanced Embedded Framework.
- It can be used with or without an RTOS.
- It is integrated with HCC USB Host stack and all its class drivers.
- It can be used with any EHCI-compliant USB host controller.
- It supports multiple simultaneous EHCI controllers, each with multiple devices attached.
- It supports all USB transfer types: Control, Bulk, Interrupt and Isochronous.

## 1.3 Packages and Documents

---

### Packages

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

Package	Description
<code>hcc_base_doc</code>	This contains the two guides that will help you get started.
<code>usbh_base</code>	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.
<code>usbh_drv_ehci</code>	The USB EHCI host controller package described by this document.

### Documents

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's Guide

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

#### HCC USB EHCI Host Controller User's Guide

This is this document.

## 1.4 Change History

---

This section includes recent changes to this product. For a complete list of all changes, refer to the file **src/history/usb-host/usbh\_hc\_ehci.txt** in the distribution package.

Version	Changes
3.09	Fixed problem that meant some transfers could report incorrect transfer size on certain processors where the last transfer descriptor overlay area could not be DMA'd back to the transfer descriptor from the queue head.
3.08	Used <b>psp_types.h</b> instead of <b>stdint.h</b> and <b>stddef.h</b> .
3.07	Introduced support for test mode.

## 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 of these files except the configuration file and PSP file.

### 2.1 API Header File

The file `src/api/api_usbh_ehci.h` is the only file that should be included by an application using this module. It declares the `usbh_ehci_hc()` function.

### 2.2 Configuration File

The file `src/config/config_usbh_ehci.h` contains all the configurable parameters. Configure these as required. For details of these options, see [Configuration Options](#).

### 2.3 Source Code

These are the source code files. **These files should only be modified by HCC.**

File	Description
<code>src/usb-host/usb-driver/ehci/ehci.c</code>	Source file for EHCI code.
<code>src/usb-host/usb-driver/ehci/ehci.h</code>	Header file for EHCI public functions.
<code>src/usb-host/usb-driver/ehci/ehci_hc.c</code>	Source file for EHCI HC descriptor.
<code>src/usb-host/usb-driver/ehci/ehci_hc.h</code>	EHCI-specific header file.
<code>src/usb-host/usb-driver/ehci/ehci_hub.c</code>	Source file for the EHCI hub.
<code>src/usb-host/usb-driver/ehci/ehci_hub.h</code>	Header file for EHCI hub public functions.
<code>src/usb-host/usb-driver/ehci/ehci_hw.h</code>	Header file for EHCI hardware-specific functions.
<code>src/usb-host/usb-driver/ehci/ehci_reg.h</code>	EHCI register file.

### 2.4 Version File

The file `src/version/ver_usbh_ehci.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.

## 2.5 PSP File

---

The file `src/psp/target/usb-host-ehci/ehci_hw.c` provides functions and elements the core code may need to use, depending on the hardware.

**Note:** These are PSP implementations for the specific microcontroller and development board; you may need to modify these to work with a different microcontroller and/or board. See [PSP Porting](#) for details.

## 3 Configuration Options

Set the system configuration options in the file `src/config/config_usbh_ehci.h`. This section lists the available configuration options and their default values.

### **EHCI\_TRANSFER\_TASK\_STACK\_SIZE**

The stack size of the EHCI transfer task(s). The default is 1024.

### **EHCI\_MAX\_DEVICE**

The maximum number of devices supported. The default is 4. This should correspond to the maximum number of physical devices that will be attached; each externally attached hub counts as a device.

### **EHCI\_MAX\_EP**

The maximum number of Bulk and Interrupt endpoints. The default is 16.

### **EHCI\_MAX\_TRANSFERS**

The maximum number of simultaneous transfers. The default is 10.

### **EHCI\_MAX\_ISO\_EP**

The maximum number of isochronous (ISO) endpoints. The default is 4.

### **EHCI\_MAX\_SISO\_LEVEL**

The maximum number of split ISO levels. This defines how many parallel ISO transfers can be executed. The default is 2.

### **EHCI\_PERIODIC\_LIST\_SIZE**

The periodic list size. The default is 256.

Set this to zero to disable ISO/INT transfers. Valid values are 0, 256, 512, and 1024. Some EHCI controllers only support 1024 so, unless you know that 256 and/or 512 is supported, using 0 or 1024 is recommended.

### **EHCI\_MAX\_SCHEDULE\_IDLE\_TIME**

The time in milliseconds that the async/periodic schedule needs to be idle before it is disabled. The default is 1.

- If this is set to zero the schedule is disabled immediately if there is no transfer. In this case Direct Memory Access (DMA) traffic is only generated if it is necessary, but performance might be poorer.
- If this is set, some needless DMA traffic is generated but the impact on the overall performance is minimal.



## **EHCI\_HC\_COUNT**

The number of EHCI controllers supported. (Some microcontrollers have more than one EHCI controller.) The maximum is 4 and the default is 1.

## **EHCI\_REG\_RENDIAN**

The reverse endianness between EHCI registers and the system. Set this to 1 if the EHCI controller is operating in reverse endianness relative to the processor. The default is 1.

## **EHCI\_DATA\_RENDIAN**

The reverse endianness between EHCI structures and the system. The default is 1.

## **EHCI\_64BIT\_MODE**

Set this to 1 if the EHCI controller operates in 64 bit mode. The default is zero.

## **EHCI\_DYNAMIC\_OFFSET**

If you set this, the offset of the EHCI controller can be changed dynamically. The default is zero. See EHCI\_OFFSET\_*n* and OPERATIONAL\_OFFSET\_*n* below.

## **EHCI\_READ\_REG\_32, EHCI\_WRITE\_REG\_32**

These specify the read/write routines to use when accessing an EHCI register. Currently these are mapped to **psp\_rreg32()** and **psp\_wreg32()**.

These macros are defined in **psp/include/psp\_reg.h** and the parameters are (base, offset) for read and (base, offset, value) for write. *base* is always USB\_BASE\_*n*. *offset* is calculated using EHCI\_OFFSET\_*n*+ [OPERATIONAL\_OFFSET\_*n*]+EHCI\_REGISTER\_ADDRESS.

**Note:** For the following options, *n* is 0 to 3. Only set values for the host controllers which are used.

## **USB\_BASE\_n**

The base address of the USB module that includes the EHCI register set, required if processor-specific registers are available for additional settings. The default is zero.

For microcontrollers which have non-EHCI registers belonging to the host controller, you can set USB\_BASE\_*n* to be able to access these registers using the **\_ehci\_[r|w|sb|cb|gb]32\_reg** routines. The register read/write routines can be used relative to USB\_BASE. This can be useful for addressing registers in **ehci\_hw\_\*** functions.

Some platforms need the option of changing the way registers are accessed (see EHCI\_READ\_REG\_32 and EHCI\_WRITE\_REG\_32). In these cases this can represent a unique identifier that needs to be passed to the dedicated read/write function. For example, the parameters identifying the EHCI units can be stored in an array and this number is the index of the array.

**EHCI\_OFFSET\_n**

This can be either of the following:

- The offset of the EHCI register, relative to USB\_BASE\_n. If EHCI\_DYNAMIC\_OFFSET is set, then this is a pointer to a variable that stores the offset.
- A pointer to a uint32\_t that stores the dynamic offset (if EHCI\_DYNAMIC\_OFFSET is set).

The default is ISP1561\_EHCI\_BASE. The type is uint32\_t if EHCI\_DYNAMIC\_OFFSET is zero and uint32\_t\* if EHCI\_DYNAMIC\_OFFSET is set to 1.

**OPERATIONAL\_OFFSET\_n**

The operational register offset relative to EHCI\_OFFSET\_n. The default is 0xC.

- If EHCI\_DYNAMIC\_OFFSET is set, this value is not used and the operational offset is always obtained from the CAPLENGTH register.
- If EHCI\_DYNAMIC\_OFFSET is disabled, the value of CAPLENGTH register needs to be set; this way a lot of unnecessary reads are avoided.

**CC\_EN\_n**

Set this option to 1 (the default) if the EHCI controller can only handle full and low speed devices via a companion controller (generally OHCI) and this is available.

**EHCI\_ISR\_ID\_n**

The ISR identifier of the EHCI controller. The default is 5. This is always platform/RTOS-specific.

**EHCI\_ISR\_PRIORITY\_n**

The ISR priority of the EHCI controller. The default is zero. This is always platform/RTOS-specific.

## 4 Starting the EHCI Controller

This section shows how to start the host controller and describes the task created. It includes a code example.

### 4.1 `usbh_ehci_hc`

This is the only external interface function. This is the host controller descriptor required by the `usbh_hc_init()` function.

#### Format

```
extern void * const usbh_ehci_hc
```

### 4.2 Host Controller Task

The host controller task handles all completed transfers. Callback requested for the transfer is executed from this task.

The task has the following attributes:

Attribute	Description
Entry point	<code>ehci_transfer_task_n</code> ( $n = 0/1/2/3$ )
Priority	OAL_HIGHEST_PRIORITY (USBH_TRANSFER_TASK_PRIORITY)
Stack size	This depends on the RTOS. 1024 is the default.

## 4.3 Code Example

This example shows how to initialize the EHCI host controller. Note the following:

- There is only one external interface function, **usbh\_ehci\_hc()**. You call the **usbh\_hc\_init()** function with this function as a parameter to link this host controller to the system.
- The last parameter in the **usbh\_hc\_init()** call is the number of the host controller. In this example EHCI has two controller units so the first call uses 0 and the second call uses 1.

```
void start_usb_host_stack ( void )
{
    int rc;
    rc = hcc_mem_init();
    if ( rc == 0 )
    {
        rc = usbh_init(); /* Initialize the USB host stack */
    }
    if ( rc == 0 )
    {
        /* Attach first EHCI host controller */
        rc = usbh_hc_init( 0, usbh_ehci_hc, 0 );
    }

    if ( rc == 0 )
    {
        /* Attach second EHCI host controller */
        rc = usbh_hc_init( 0, usbh_ehci_hc, 1 );
    }
    if ( rc == 0 )
    {
        rc = usbh_start(); /* Start the USB host stack */
    }
    if ( rc == 0 )
    {
        rc = usbh_hc_start( 0 ); /* Start first EHCI host controller */
    }
    if ( rc == 0 )
    {
        rc = usbh_hc_start( 1 ); /* Start second EHCI host controller */
    }

    .....
}
```

## 5 Integration

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

### 5.1 OS Abstraction Layer (OAL)

---

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.

This module requires the following OAL elements:

OAL Resource	Number Required
Tasks	1
Mutexes	1
Events	1
ISRs	One for each supported EHCI host controller ( <a href="#">EHCI_HC_COUNT</a> ).

## 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 functions:

Function	Package	Element	Description
<b>psp_membar()</b>	psp_base	psp_membar	Memory barrier.
<b>psp_memcpy()</b>	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.
<b>psp_memset()</b>	psp_base	psp_string	Sets the specified area of memory to the defined value.
<b>psp_rreg32()</b>	psp_base	psp_reg	Read routine used to access an EHCI register.
<b>psp_wreg32()</b>	psp_base	psp_reg	Write routine used to access an EHCI register.

The host controller makes use of the following functions that must be provided by the PSP. These are designed for you to port them easily to work with your hardware solution. The package includes samples in the `src/psp/target/usb-host-ehci/ehci_hw.c` file.

Function	Description
<b>ehci_hw_init()</b>	Initializes the device.
<b>ehci_hw_start()</b>	Starts the device.
<b>ehci_hw_stop()</b>	Stops the device.
<b>ehci_hw_delete()</b>	Deletes the device, releasing the associated resources.
<b>ehci_hw_suspend()</b>	Suspends a device.
<b>ehci_hw_resume()</b>	Resumes a suspended device.
<b>ehci_hw_state()</b>	Gets the state of a device.
<b>ehci_hw_connect()</b>	Connects a device.
<b>ehci_hw_disconnect()</b>	Disconnects a device.

These functions are described in the following sections.

**Note:** HCC can provide samples for different configurations; contact [support@hcc-embedded.com](mailto:support@hcc-embedded.com).

## ehci\_hw\_init

This function must be provided by the PSP to initialize the device.

### Format

```
int ehci_hw_init ( t_usbh_unit_id unit )
```

### Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

## ehci\_hw\_start

This function must be provided by the PSP to start the device.

### Format

```
int ehci_hw_start ( t_usbh_unit_id unit )
```

### Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.



## ehci\_hw\_stop

This function must be provided by the PSP to stop the device.

### Format

```
int ehci_hw_stop ( t_usbh_unit_id unit )
```

### Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

## ehci\_hw\_delete

This function must be provided by the PSP to delete the device, releasing associated resources.

### Format

```
int ehci_hw_delete ( t_usbh_unit_id unit )
```

### Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

## ehci\_hw\_suspend

This function must be provided by the PSP to suspend a device.

### Format

```
int ehci_hw_suspend ( t_usbh_unit_id unit )
```

### Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

## ehci\_hw\_resume

This function must be provided by the PSP to resume a suspended device.

### Format

```
int ehci_hw_resume( t_usbh_unit_id unit )
```

### Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

## ehci\_hw\_state

This function must be provided by the PSP to get the state of a device.

### Format

```
int ehci_hw_state ( t_usbh_unit_id unit )
```

### Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

## ehci\_hw\_connect

This function must be provided by the PSP to connect a device.

### Format

```
void ehci_hw_connect ( t_usbh_unit_id unit, uint8_t port, uint8_t speed )
```

### Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id
port	The port to connect to.	uint8_t
speed	The line speed.	uint8_t

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

## ehci\_hw\_disconnect

This function must be provided by the PSP to disconnect a device.

### Format

```
void ehci_hw_disconnect ( t_usbh_unit_id unit, uint8_t port )
```

### Arguments

Argument	Description	Type
unit	The unit ID.	t_usbh_unit_id
port	The port to disconnect.	uint8_t

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.