USB Atmel Host Controller User Guide

Version 1.10

For use with USBH Atmel Host Controller versions 2.10 and above

Date: 19-Jun-2017 16:54

Table of Contents

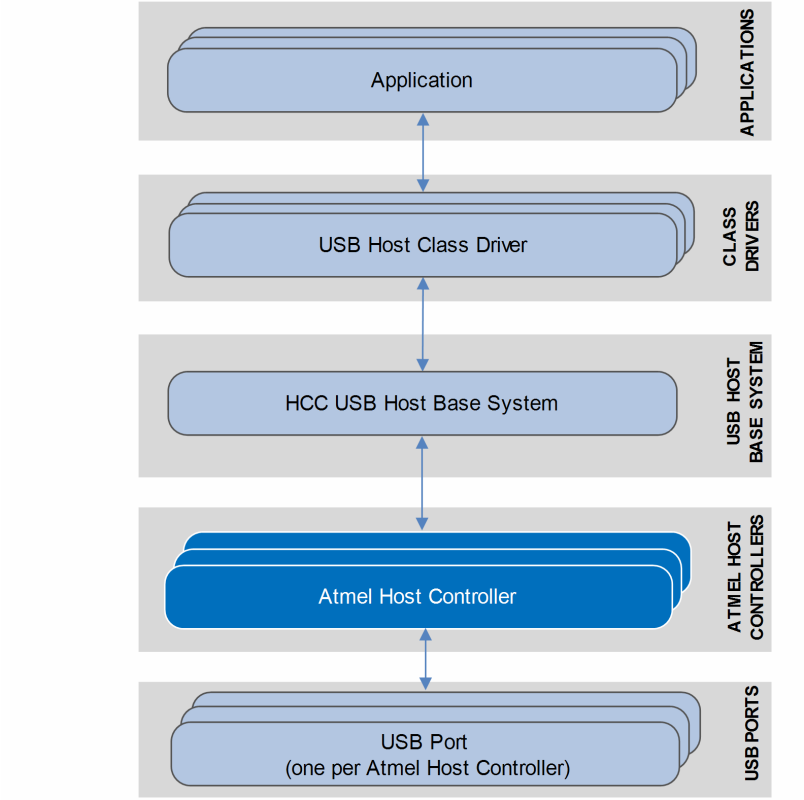
1 System Overview

1.1 Introduction

This guide is for those who want to implement HCC Embedded's Atmel USB Host Controller with the HCC USB host stack.

The Atmel module provides a high speed USB 2.0 host controller which provides both full 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. Supported Atmel micro-controllers include those in the AVR32 and SAM V71 series.

The position of the host controller within the USB stack is shown below:



1.2 Feature Check

The main features of the host controller are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Designed for integration with both RTOS and non-RTOS based systems.
- Integrated with the HCC USB Host stack and all its class drivers.
- Supports multiple simultaneous Atmel controllers, each with multiple devices attached.
- 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
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_drv_atmel	The USB Atmel host controller package described by this document.
psp_template_base	The base Platform Support Package (PSP).

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 USB Atmel Host Controller User Guide

This is this document.

1.4 Change History

This section describes past changes to this manual.

- To view or download earlier manuals, see Archive: USB Atmel Host Controller User Guide.
- For the history of changes made to the package code itself, see History: usbh_drv_atmel.

The current version of this manual is 1.10. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.10	2017-06-19	2.10	New *Change History* format.
1.00	2017-04-13	2.09	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 of these files except the configuration files and PSP files.

2.1 API Header File

The file **src/api/api_usbh_atmel.h** is the only file that should be included by an application using this module. It declares the Application Programming Interface (API) functions. For details, see Starting the Host Controller.

2.2 Configuration Files

These files are in the directory **src/config**:

File	Description
config_usbh_atmel.h	Contains all the configurable parameters. Configure these as required. For details of these options, see Configuration Options.
config_usbh_atmel.c	Controls bank configuration for pipes/endpoints.

2.3 Source Code

The source code files are in the directory **src/usb-host/usb-driver/atmel**. **These files should only be modified by HCC**.

File	Description
usbh_atmel.c	Source file for Atmel code.
usbh_atmel.h	Header file for Atmel public functions.
usbh_atmel_hc.c	Source file for the Atmel HC descriptor.
usbh_atmel_hc.h	HC descriptor header file.
usbh_atmel_hub.c	Source file for Atmel hub.
usbh_atmel_hub.h	Header file for Atmel hub public functions.
usbh_atmel_regs.h	Register values.

2.4 Version File

The file **src/version/ver_usbh_atmel.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 Platform Support Package (PSP) Files

These files are in the directory **src/psp/target/usbh_atmel**. These provide functions and elements the core code may need to use, depending on the hardware.

> **Note:** These are PSP implementations for the specific micro-controller and board; you may need to modify these to work with a different micro-controller and/or development board. See PSP Porting for details.

The files are as follows:

File	Description
psp_usbh_atmel.c	Functions source code.
psp_usbh_atmel.h	Header file for functions.

3 Configuration Options

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

USBH_ATMEL_MAX_EP

The maximum number of software endpoints: Bulk, Isochronous, and Interrupt. The default is 4.

USBH_ATMEL_MAX_TRANSFERS

The maximum number of simultaneous transfers The default is 6.

USBH_ATMEL_HOST_ISR

The host ISR. The default is 2.

USBH_ATMEL_HOST_INT_PRIO

The interrupt priority. The default is 1.

USBH_ATMEL_TRANSFER_TASK_SIZE

The stack size of the transfer task. The default is 1024.

USBH_ATMEL_USE_DMA

Keep the default of 1 to use the bus master DMA capability of the USB module. Set it to 0 to disable this.

USBH_ATMEL_VBUSEN_ACTIVE_LOW

Keep the default of 1 if the USB_VBOF output signal must be inverted (active low). Set it to 0 to disable this.

USBH_ATMEL_FORCE_FSLS

Set this to 1 to force the USB core to operate at Full Speed/Low Speed only, even if it supports High Speed. The default is 0.

USBH_ATMEL_BULK_SOF_SCHEDULE

Keep the default of 1 to use the Bulk pipe fair bus access. Set it to 0 to disable this.

USBH_ATMEL_BULK_PIPE_WORKAROUND

This is the workaround for blocking Bulk pipes. Use this for evaluating performance if not using USBH_ATMEL_BULK_SOF_SCHEDULE (above). Possible values are:

- 0 - do not use the workaround (the default).
- 1 - treat Bulk IN pipes as Interrupt pipes.
- 2 - treat Bulk OUT pipes as Control pipes.

4 Starting the Host Controller

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

4.1 usbh_atmel_hc

This external interface function provides the host controller descriptor required by the **usbh_hc_init()** function.

Format

```
extern void * const  usbh_atmel_hc
```

4.2 Host Controller Task

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

The task has the following attributes:

Attribute	Description
Entry point	*atmel_transfer_task*
Priority	USBH_TRANSFER_TASK_PRIORITY
Stack size	USBH_ATMEL_TRANSFER_TASK_SIZE. The default is 1024.

4.3 Code Example

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

- There is only one external interface function, **usbh_atmel_hc()**. To link this host controller to the system, you call the **usbh_hc_init()** function with this function as a parameter.
- The last parameter in the **usbh_hc_init()** call is the number of the host controller.

```
void start_usb_host_stack ( void )

{
int  rc;
rc = hcc_mem_init();


    if ( rc == 0 )
    {
        rc = usbh_init();    /* Initialize USB host stack */
    }

    if ( rc == 0 )
    {
        /* Attach Atmel host controller */
        rc = usbh_hc_init( 0, usbh_atmel_hc, 0 );
    }

    if ( rc == 0 )
    {
        rc = usbh_start();  /* Start USB host stack */
    }


    if ( rc == 0 )
    {
        rc = usbh_hc_start( 0 );  /* Start Atmel 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

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	1

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
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 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 for the AVR32 and SAM V71 devices in the **psp_usbh_atmel.c** file.

Function	Description
psp_usbh_atmel_init()	Initializes the device.
psp_usbh_atmel_start()	Starts the device.
psp_usbh_atmel_stop()	Stops the device.
psp_usbh_atmel_delete()	Deletes the device, releasing the associated resources.

These functions are described in the following sections.

Note: HCC can provide samples for different configurations; contact support@hcc-embedded.com.

psp_usbh_atmel_init

This function is provided by the PSP to initialize the device.

Format

```
extern int psp_usbh_atmel_init ( void )
```

Arguments

None.

Return Values

None.

psp_usbh_atmel_start

This function is provided by the PSP to start the device.

Format

```
extern int psp_usbh_atmel_start ( void )
```

Arguments

None.

Return Values

None.

psp_usbh_atmel_stop

This function is provided by the PSP to stop the device.

Format

```
extern int psp_usbh_atmel_stop ( void )
```

Arguments

None.

Return Values

None.

psp_usbh_atmel_delete

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

Format

```
extern int psp_usbh_atmel_delete ( void )
```

Arguments

None.

Return Values

None.