
Atmel AVR4950: ASF - USB Host Stack

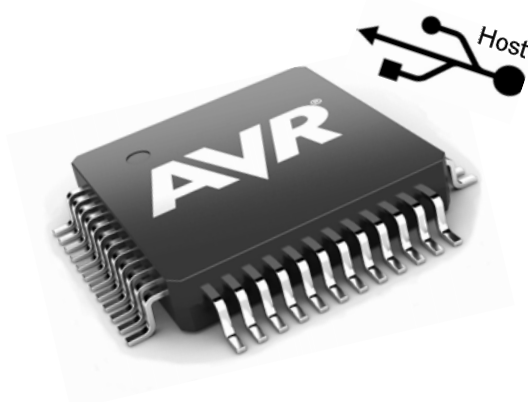


Features

- USB 2.0 compliance
 - Chapter 9
 - Control, Bulk, Isochronous and Interrupt transfer types
 - Low Speed (1.5Mbit/s), Full Speed (12Mbit/s), High Speed (480Mbit/s) data rates
- Low stack size
- Real time (OS compliance, no latency)
- Interrupt driven
- Speed performance using USB DMA
- Low power modes
- USB HUB
- Major USB classes and ready to use (HID, CDC, MSC, PHDC, AUDIO)
- Composite device
- 8-bit and 32-bit AVR® platforms
- GCC and IAR™ compilers

1 Introduction

This document introduces the USB host stack. This stack is included in the Atmel® AVR Software Framework (ASF), and aims to provide the customer with the quickest and easiest way to build a USB embedded host application. Also this USB host stack is oriented low footprint and low power. A full description of this stack is available in this document.



8-bit Atmel Microcontrollers

Application Note

Rev. 8486A-AVR-02/12





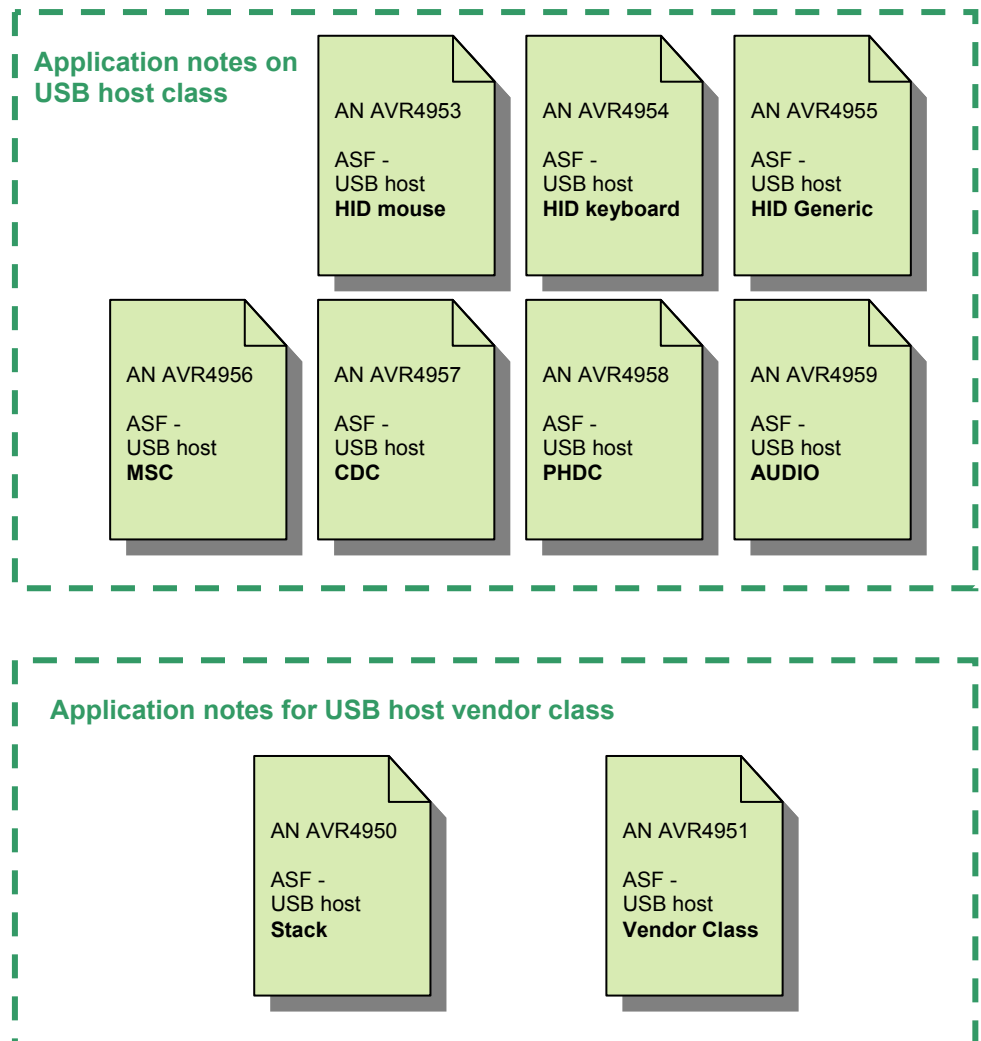
2 Abbreviations

ASF:	AVR Software Framework
CBW:	Command Block Wrapper (from Mass Storage Class)
CDC:	Communication Device Class
CSW:	Command Status Wrapper (from Mass Storage Class)
DP or D+:	Data Plus differential line
DM or D-:	Data Minus differential line
EHCI:	Enhanced Host Controller Interface
FS:	USB Full Speed
HID:	Human interface device
HS:	USB High Speed
LS:	USB Low Speed
MSC:	Mass Storage Class
OHCI:	Open Host Controller Interface
PHDC:	Peripheral Health Device Class
PIPE:	Data buffer to manage an endpoint transfer in host mode
sleepmgr:	Sleep management service from ASF
UDC:	USB Device Controller
UDD:	USB Device Descriptor
UDI:	USB Device Interface
UHC:	USB Host Controller
UHD:	USB Host Descriptor
UHI:	USB Host Interface
USB:	Universal Serial Bus
USBB:	USB hardware interface version B for 32-bit Atmel core
USBC:	USB hardware interface version C for 32-bit Atmel core
STALL:	USB token used to stop a USB transaction
ZLP:	Zero length packet

3 USB host application notes

Several USB host examples are provided by Atmel. For each example, Atmel provides several application notes.

Figure 3-1. USB host application notes.



Basic USB knowledge is necessary to understand the USB host class application notes (Classes: HID, CDC, MSC, PHDC, AUDIO).

To create a USB host with one of the ASF provided classes, refer directly to the related application note for this USB class.

The USB host stack and USB host vendor class application notes are designed for advanced USB developers.

4 Organization

4.1 Overview

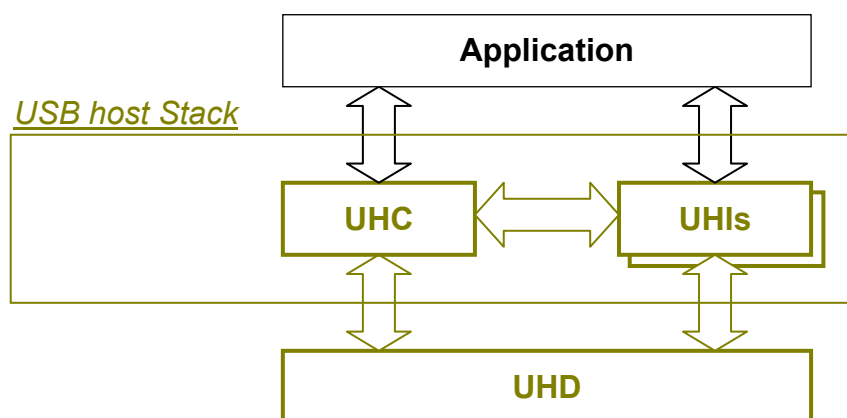
The USB host stack is divided into three parts:

- The USB Host Controller (UHC), providing USB Chapter 9 compliance
- The USB Host Interface (UHI), providing USB Class compliance
- The USB Host Driver (UHD), providing the USB interface for each Atmel product

NOTE

The USB host drivers are implemented in full interrupt mode, thus this UHD is a perfect base to create a USB driver for third party's USB stacks. This is applicable either for custom interface or for OHCI/EHCI interface.

Figure 4-1. USB host stack architecture.



In case of dual (device/host) mode, the application note [AVR4900: ASF – USB Device stack](#) is provided to describe the USB device stack.

4.2 Memory Footprint

The USB host stack footprint depends on:

- Atmel core (megaAVR®, UC3)
- USB hardware version
- USB Class used
- Compiler and optimization level

These parameters give different footprint values, but in average the USB host stack does not exceed 10Kbytes of FLASH and 1Kbytes of RAM using compiler highest optimization level.

4.3 USB host stack files

The USB host stack is available in [Atmel AVR Studio® 5](#) through the New Example Project wizard and Select ASF Modules wizard. Accessing examples is done by browsing the examples list or simply by typing "host" in the search field.

NOTE

Tip: Select *Technology USB* to reduce examples list in AVR Studio 5.

NOTE

The example names including «From ASF V1» refer to former implementation of USB host stack from ASF v1.7 which is beyond the scope of this application note.

4.3.1 Common files for all products

<u>Files</u>	<u>Paths</u>
<ul style="list-style-type: none">Defines USB constants usb_protocol.h* (from usb.org) usb_atmel.h* (from Atmel)	common/services/usb/
<ul style="list-style-type: none">UHC files uhc.c/h uhi.h uhd.h	common/services/usb/uhc/
<ul style="list-style-type: none">Classes Protocols files usb_protocol_<class>.h*	common/services/usb/class/<class>/
<ul style="list-style-type: none">UHI files uhi_<foo>.c/h	common/services/usb/class/foohost/

NOTE

* These files are common with USB device stack.

4.3.2 UHD files depending on selected products

- avr32/drivers/usbb/usbb_host.c/h
- avr32/drivers/usbb/usbb_otg.h*
- avr32/drivers/usbc/usbc_host.c/h
- avr32/drivers/usbc/usbc_otg.h*

NOTE

* These files are common with the USB device stack.

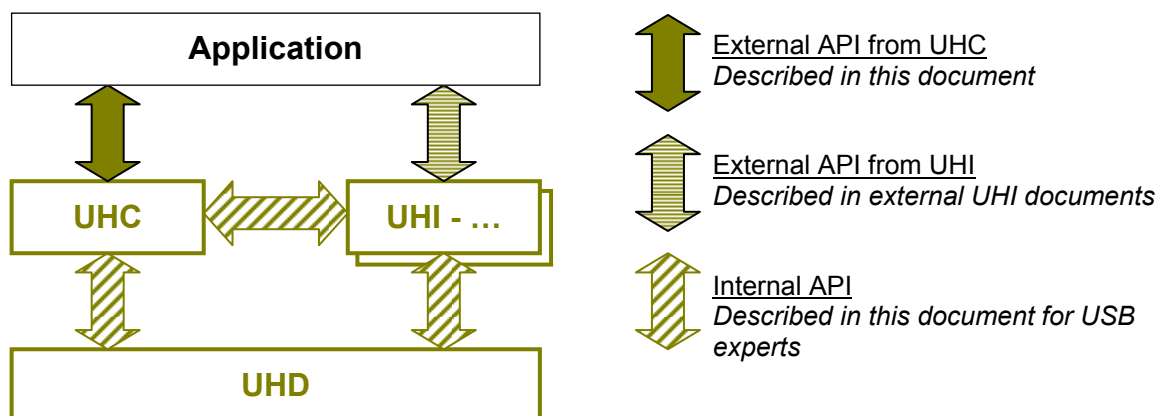
4.3.3 Specific file for each application

- Application file to configure USB host stack (see Chapter 7 [Configuration](#), and [Table 5-3](#))
usb_conf_host.h

5 Application programming interface

This section describes all APIs except the UHI APIs, which are described in a separate document.

Figure 5-1. USB modules.



5.1 External API from UHC

The external UHC API allows the application to manage common USB host behavior and receive common USB host events. These controls and events are common to any USB application.

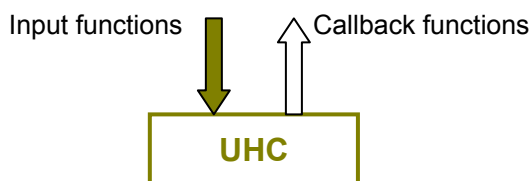


Table 5-1. External API from UHC – Input functions.

Declaration	Description
<code>uhc_start()</code>	Start the host mode.
<code>uhc_stop(bool b_id_stop)</code>	Stop the host mode and stop control of USB ID pin, if requested by <code>b_id_stop</code> .
<code>uhc_suspend(bool b_remotewakeup)</code>	Suspend the USB line. This authorize or not the remote wakeup features of each devices.
<code>uhc_is_suspend()</code>	Test if the suspend state is enabled on the USB line.
<code>uhc_resume()</code>	Resume the USB line.

Table 5-2. External API from UHC – Input functions concerning connected devices.

Declaration	Description
<code>uint8_t uhc_get_device_number()</code>	Return the number of devices connected.
<code>char* uhc_dev_get_str_manufacturer(uhc_device_t*dev)</code>	Return the string ASCII manufacturer corresponding at USB device.
<code>char* uhc_dev_get_str_product(uhc_device_t*dev)</code>	Return the string ASCII product corresponding at USB device.
<code>char* uhc_dev_get_str_serial(uhc_device_t*dev)</code>	Return the string ASCII serial corresponding at USB device.
<code>char* uhc_dev_get_str(uhc_device_t*dev, uint8_t str_id)</code>	Return the USB string ASCII corresponding at the string ID.
<code>uint16_t uhc_dev_get_power(uhc_device_t*)</code>	Get the maximum consumption of a device (mA).
<code>uhd_speed_t uhc_dev_get_speed(uhc_device_t*)</code>	Return the current device speed.
<code>bool uhc_dev_is_high_speed_support(uhc_device_t*)</code>	Return true if it is the device supports high speed.

All UHC callbacks are optional and defined by the user in *conf_usb_host.h* file for each application.

Table 5-3. External API from UHC – Callback functions.

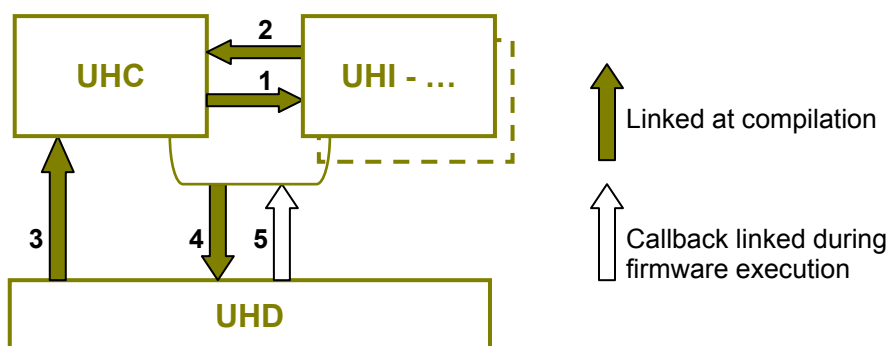
Define name	Description
<code>UHC_MODE_CHANGE(bool b_host_mode)</code>	To notify that the USB mode are switched automatically. This is possible only when ID pin is available.
<code>UHC_VBUS_CHANGE(bool b_present)</code>	To notify that the V_{BUS} level has changed (Available only in USB hardware with V_{BUS} monitoring).
<code>UHC_VBUS_ERROR()</code>	To notify that a V_{BUS} error has occurred (Available only in USB hardware with V_{BUS} monitoring).
<code>UHC_CONNECTION_EVENT(uhc_device_t* dev, bool b_present)</code>	To notify that a device has been connected or disconnected.
<code>UHC_WAKEUP_EVENT()</code>	Called when a USB device or the host have wake up the USB line.
<code>UHC_SOF_EVENT()</code>	Called for each received SOF each 1ms. Note: Available in High and Full speed mode.
<code>UHC_DEVICE_CONF(uhc_device_t* dev)</code>	Called when a USB device configuration must be chosen. Thus, the application can choose either a configuration number for this device or a configuration number 0 to reject it. If callback not defined the configuration 1 is chosen.

Define name	Description
UHC_ENUM_EVENT (uhc_device_t* dev, uhc_enum_status_t status)	Called when a USB device enumeration is completed. The status can be: <ul style="list-style-type: none"> - UHC_ENUM_SUCCESS Device is bus enumerated with at least one supported interface. - UHC_ENUM_UNSUPPORTED All interfaces are not supported by UHIs. - UHC_ENUM_OVERCURRENT Device power can not be supported - UHC_ENUM_FAIL A problem has been occurred during USB enumeration. - UHC_ENUM_HARDWARE_LIMIT USB hardware can not support it. Not enough free pipes. - UHC_ENUM_SOFTWARE_LIMIT USB software can not support it. Implementaion limit. - UHC_ENUM_MEMORY_LIMIT USB software can not support it. Not enough memory.

5.2 Internal APIs

The following definitions are defined for advanced USB users who intend to develop a specific USB host, not provided in ASF.

Figure 5-2. Internal USB host API overview.



NOTE

Numbers are references for tables below.

Table 5-4. UHI input from UHC (1).

Declaration	Description
uhc_enum_status_t (*install)(uhc_device_t*)	Install interface if supported by new device (allocation of endpoints).
void (*uninstall)(uhc_device_t*)	Uninstall the interface if installed for this device.
void (*enable)(uhc_device_t*)	Called to start the USB interface of the device.

Declaration	Description
<code>void (*sof_notify)(bool b_micro)</code>	Called by UHC to notify a SOF event on the enabled USB interface.

Note: The UHI API are stored in USB_HOST_UHI array, see [Table 7-1](#).

Table 5-5. UHC input from UHD (3).

Declaration	Description
<code>uhc_start()</code>	Start the host mode.
<code>uhc_stop(bool b_id_stop)</code>	Stop the host mode and USB ID pin management if requested by <code>b_id_stop</code> .
<code>uhc_notify_connection(bool b_plug)</code>	Called when a device has been connected or disconnected.
<code>uhc_notify_sof(bool b_micro)</code>	Called when a SOF is sent.
<code>uhc_notify_resume()</code>	Called when resume bus state occurs. After a downstream or an upstream resume.

Table 5-6. UHD input (4).

Declaration	Caller	Description
<code>uhd_enable()</code>	UHC	Enables the USB host mode and ID management if the ID pin is available.
<code>uhd_disable(bool b_id_stop)</code>	UHC	Disables the USB host mode and USB ID pin management if requested by <code>b_id_stop</code> .
<code>uhd_speed_t uhd_get_speed()</code>	UHC/UHI	To check the speed of device connected to USB interface port (not device connected on a USB hub).
<code>uint16_t uhd_get_frame_number()</code>	Application	Returns the current start of frame number.
<code>uint16_t uhd_get_microframe_number()</code>	Application	Returns the current micro start of frame number.
<code>uhd_send_reset (uhd_callback_reset_t callback)</code>	UHC	The USB driver enables the reset state on the USB line.
<code>uhd_suspend()</code>	UHC	The USB driver enables the suspend state on the USB line.
<code>uhd_is_suspend()</code>	UHC	Test if the suspend state is enabled on the USB line.
<code>uhd_resume()</code>	UHC	The USB driver enables the IDLE state on the USB line. ("Downstream Resume")
<code>bool uhd_setup_request(usb_add_t add, usb_setup_req_t req, uint8_t *payload, uint16_t payload_size, uhd_callback_setup_run_t callback_run, uhd_callback_setup_end_t callback_end)</code>	UHC/UHI	Add this setup request, in the control endpoint setup queue. (Request timeout is 5s)
<code>bool uhd_ep0_alloc(usb_add_t add, uint8_t ep_size)</code>	UHC	Enables control endpoint 0.
<code>bool uhd_ep_alloc(usb_add_t add, usb_ep_desc_t *ep_desc)</code>	UHC	Enables endpoints (except control endpoint).
<code>uhd_ep_free(usb_add_t add, usb_ep_t endp)</code>	UHC	Disables one endpoint or all endpoints of a device.



Declaration	Caller	Description
<pre>bool uhd_ep_run(usb_add_t add, usb_ep_t endp, bool b_shortpacket, uint8_t *buf, iram_size_t buf_size, uint16_t timeout, uhd_callback_trans_t callback)</pre>	UHI	<p>Starts/stops a data transfer in or out on an endpoint. The schedule of the transfer is managed by UHD.</p> <p>Note: The control endpoint is not authorized here.</p>
<pre>uhd_ep_abort(usb_add_t add, usb_ep_t endp)</pre>	UHI	

Table 5-7. UHD callback (5).

Declaration	Description
<pre>typedef void (*uhd_callback_reset_t)(void);</pre>	Called when the reset event, requested via uhd_send_reset() , is completed.
<pre>typedef bool (*uhd_callback_setup_run_t)(usb_add_t add, uint8_t **payload, uint16_t *payload_size);</pre>	Called when the (<i>payload</i>) buffer, assigned via uhd_setup_request() is full or empty. Then the setup request is hold. A new buffer can be provided to restart the setup request or the setup request will be aborted.
<pre>typedef void (*uhd_callback_setup_end_t)(usb_add_t add, uhd_trans_status_t status, uint16_t payload_trans);</pre>	Called when the setup request is completed. This callback is given via uhd_setup_request() routine.
<pre>typedef void (*uhd_callback_trans_t) (usb_add_t add, uhd_trans_status_t status, iram_size_t nb_transfered)</pre>	Called when a transfer request is completed or cancelled. This request is registered via uhd_ep_run() .
<pre>uhd_trans_status_t</pre>	<p>This structure is updated after every attempted transaction, whether successful or not. If the transaction is successful, then the status field is set to:</p> <ul style="list-style-type: none"> - UHD_TRANS_NOERROR. <p>Otherwise, it is set according to the following transmission error types:</p> <ul style="list-style-type: none"> - UHD_TRANS_CRC, CRC error in data packet. - UHD_TRANS_DISCONNECT, device is disconnected. - UHD_TRANS_DT_MISMATCH, data toggle PID did not match the expected value. - UHD_TRANS_STALL, the endpoint returned a STALL PID. - UHD_TRANS_NOT_RESPONDING, Device did not respond to token (IN) or did not provide a handshake (OUT). - UHD_TRANS_PID_FAILURE, Check bits on PID from endpoint failed. - UHD_TRANS_TIMEOUT, Data transmission not completed before timeout. - UHD_TRANS_ABORTED, Data transmission has been aborted.

Table 5-8. UHD input for High Speed application only (4).

Declaration	Caller	Description
<code>uhd_test_mode_j()</code>	UHC	Features to enable a specific signal on a USB HS port. These are requested to run a USB host HS certification.
<code>uhd_test_mode_k()</code>	UHC	
<code>uhd_test_mode_se0_nak()</code>	UHC	
<code>uhd_test_mode_packet()</code>	UHC	

5.3 Device connected information

The UHC manages the devices via a structure called `uhc_device_t`, which stores many information about the devices connected on the USB tree.

The global variable `g_uhc_device_root` contains the information about the first device connected and it is the first element of the device list.

The user application can read this list to get information on current USB tree.

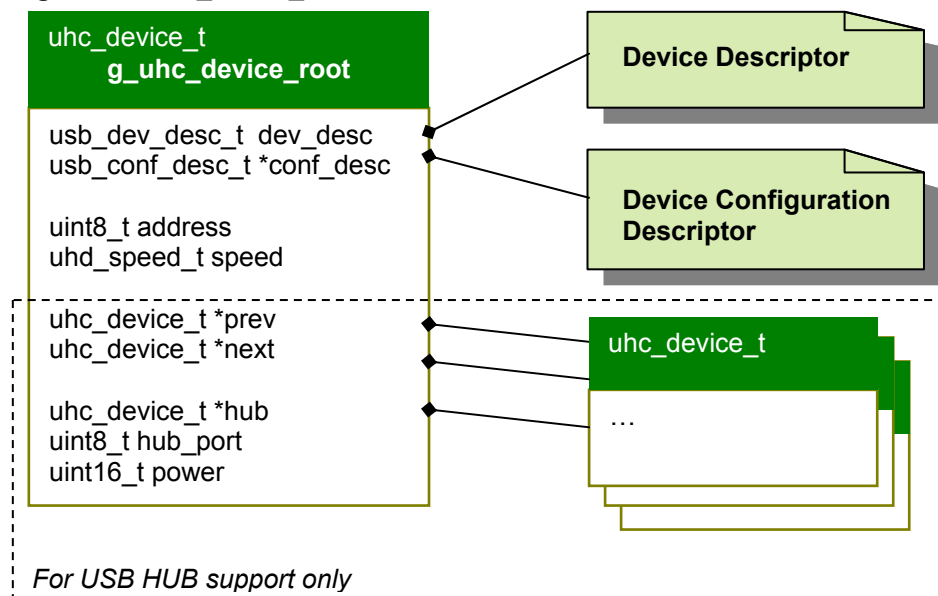
NOTE

There is no USB tree limitation, because the device list and the USB descriptor fields allocations are dynamic and require the standard allocation library (e.g. `alloc()`).

The `uhc_device_t` structure includes:

- A pointer on the USB device descriptor
- A pointer on the USB configuration descriptor
- the USB device address
- the USB device speed
- pointers on parent USB HUB device (for USB HUB support only)
- HUB port number connection (for USB HUB support only)

Figure 5-3. `uhc_device_t` definition.



Legend:

From USB 2.0

ASF struct



6 USB host stack behavior

This chapter aims to answer advanced USB usage questions and introduce the USB host stack behavior.

The following sections describe the interaction between the different layers for these steps:

- USB dual roles management
- USB V_{BUS} management
- USB device connection management
- USB device enumeration management

6.1 Dual roles

When the USB hardware has the possibility to work in USB device and in USB host mode, the user can implement a dual role application. This application will switch between a USB device and a USB host task following the USB ID pin level (from OTG specification) or a custom process.

The USB host stack provides three possibilities to manage the dual role: manual, automatic and semi-automatic.

The **manual** dual role must be used when ID pin is not available. In this case, a custom process manages the dual role.

The **automatic** and **semi-automatic** dual role must be used when ID pin is managed by the USB hardware interface.

The **semi-automatic** dual role allows delaying the start of USB device and USB host mode when the ID pin change. This possibility is enabled when `UHD_START_MODE_MANUAL` is defined (See [Table 7-2](#)).

Figure 6-1. USB manual dual role.

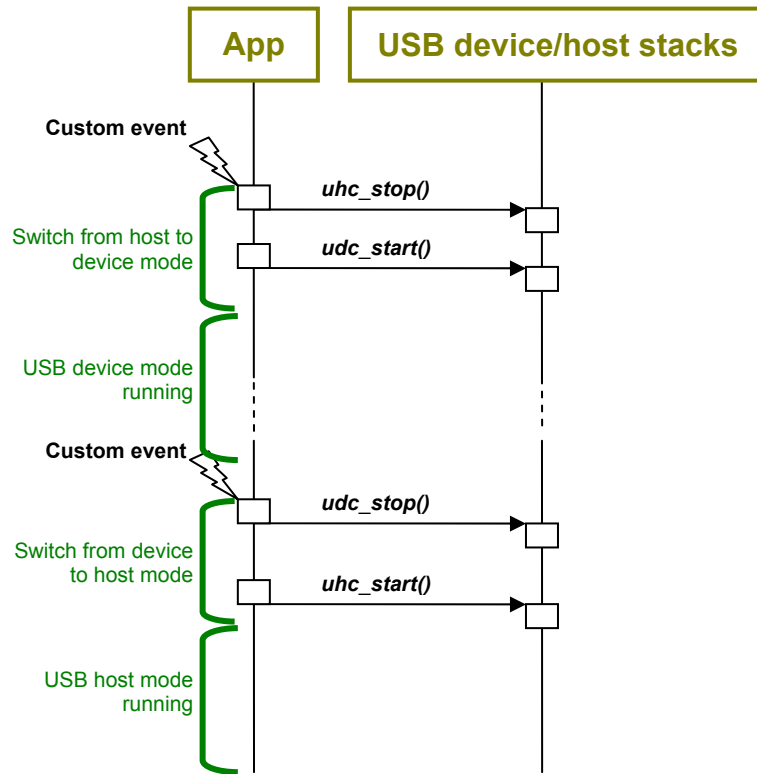


Figure 6-2. USB semi-automatic dual role.

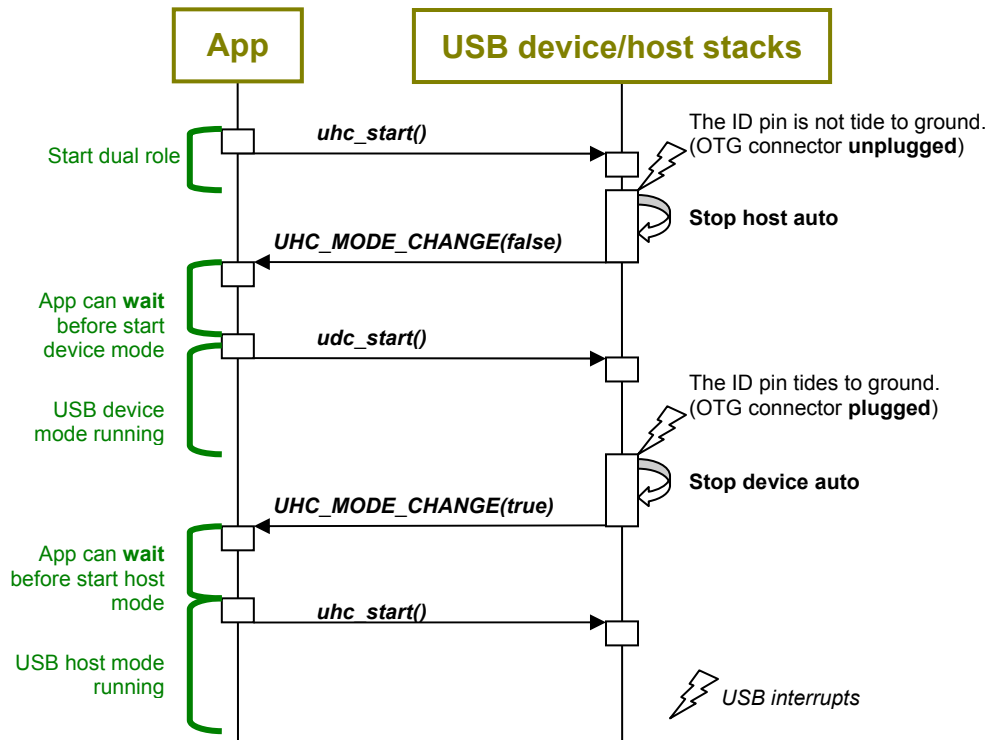
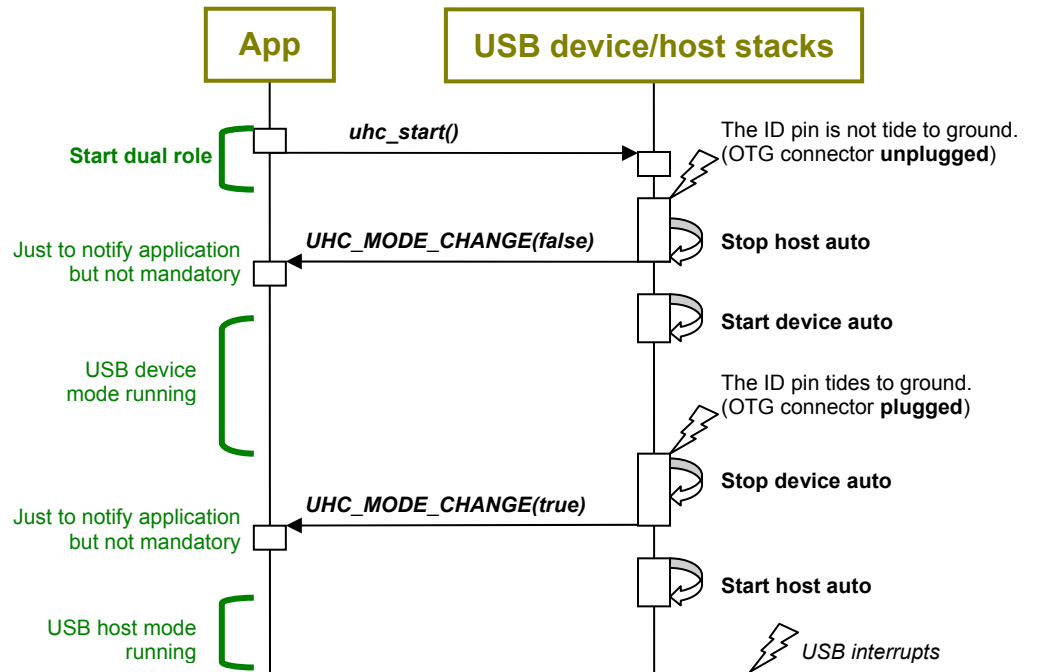
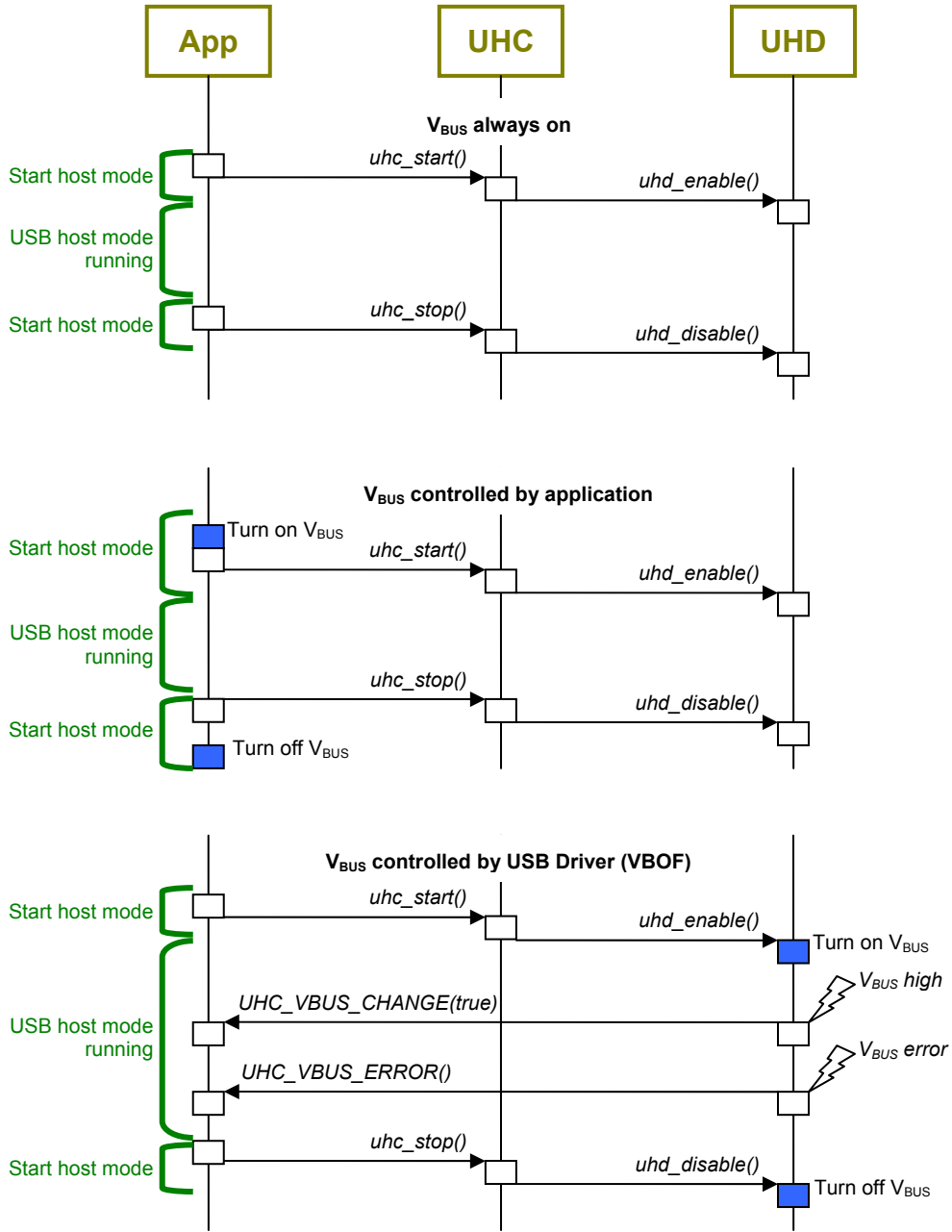



Figure 6-3. USB automatic dual role.



6.2 V_{BUS} management

Figure 6-4. V_{BUS} management.

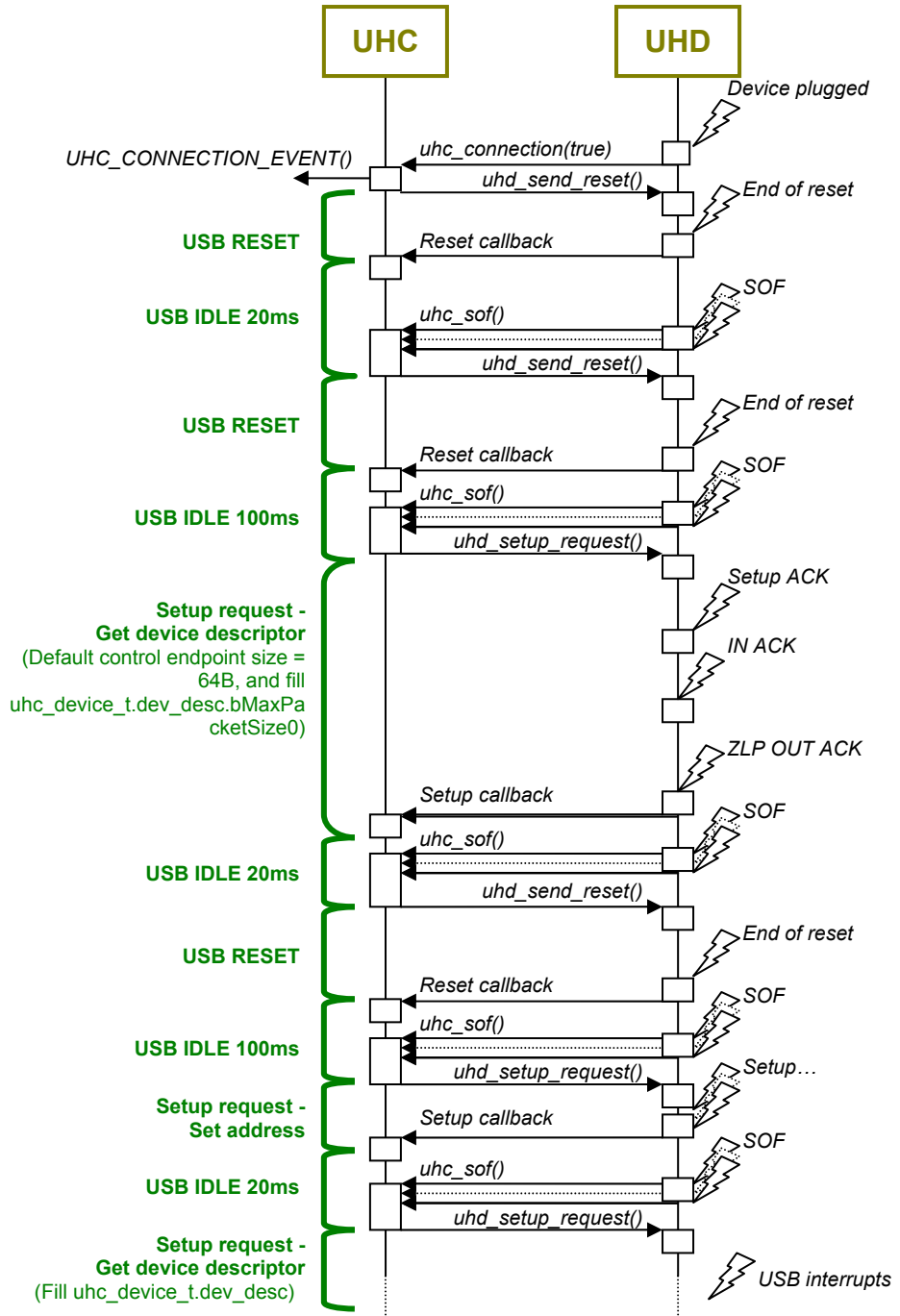


 USB interrupts



6.3 Device connection

Figure 6-5. USB device connection.



NOTE

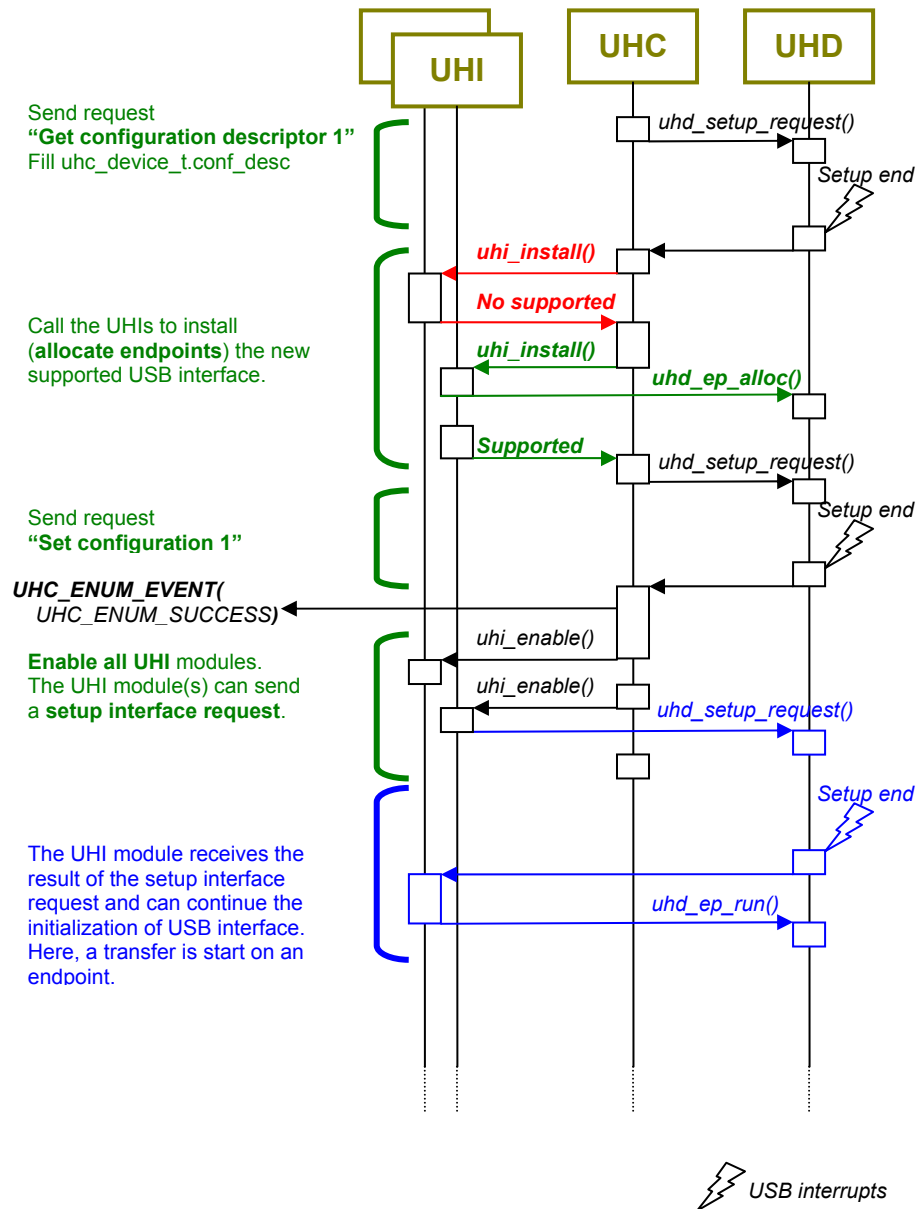
This sequence is the same for the device connected thru a USB HUB, but the connection and reset events are managed via USB HUB setup request.

NOTE

When the USB device enumeration fails four times, the USB line of the device is set in USB SUSPEND mode.

6.4 Device enumeration

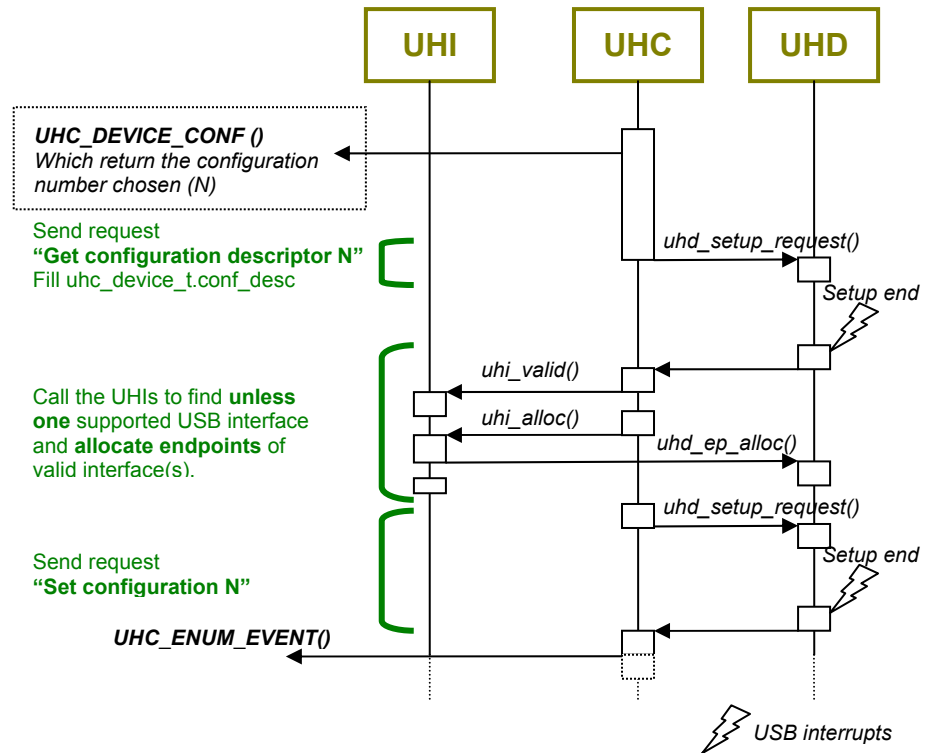
Figure 6-6. USB device enumeration (single USB configuration).



NOTE

Here the callback `UHC_DEVICE_CONF()` is not defined, thus the device configuration 1 is chosen by default.

Figure 6-7. USB device enumeration (multiple USB configurations).



User code example to support a known device with multiple configurations:

```
<conf_usb.h>
#define UHC_DEVICE_CONF(dev) user_choose_usb_conf(dev)

<User C file>:
uint8_t user_choose_usb_conf(uhc_device_t* dev) {
    if ((dev->dev_desc.idVendor == 0x1234)
        && (dev->dev_desc.idProduct == 0x5678)) {
        Assert(dev->dev_desc.bNumConfigurations>=2);
        return 2; // The configuration 2 is chosen
    }
    return 0; // No conf chosen because of unknown device
}
```

7 Configuration

The application's configurations are defined in the `conf_usb_host.h` file. This file must be created for each application, and this action requires a basic USB knowledge.

The `conf_usb_host.h` file defines the following configurations:

- USB host configuration
- USB host interface configuration
- USB host driver configuration

7.1 USB host configuration

The following configuration must be included in the `conf_usb.h` file of the application, which is the main USB device configuration.

Table 7-1. USB host configuration.

Define name	Type	Description
USB_HOST_UHI	Array of UHI APIs	Define the list of UHI supported by USB host. Example: <code>#define USB_HOST_UHI UHI_MSC, UHI_HID_MOUSE</code>
USB_HOST_POWER_MAX	mA	Maximum current allowed on V _{BUS} .
USB_HOST_HUB_SUPPORT ⁽¹⁾	Only defined	Authorize the USB HUB support. (Available for release ASF3)
USB_HOST_HS_SUPPORT ⁽¹⁾	Only defined	Authorize the USB host to run in High Speed.

Note: 1. Optional configuration. Comment the define statement to disable it (ex: `// #define USB_HOST_X`).

7.2 USB host interface configuration

The UHI configurations are described in USB host class application notes.

7.3 USB host drivers configuration

The USB hardware interface can provide specific features which can be enabled in `conf_usb_host.h` file, in `conf_board.h` file and in project options.

Table 7-2. USB host driver configuration in `conf_usb_host.h`.

Define name	Values	UHD	Description
UHD_NO_SLEEP_MGR	Only defined	All	Remove the management of sleepmgr service.
UHD_ISOCHRONOUS_NB_BANK	1, 2, 3	USBB	Reduces or increases isochronous endpoint buffering. Default value if not defined: 2
UHD_BULK_NB_BANK	1, 2, 3	USBB	Reduces or increases bulk endpoint buffering. Default value if not defined: 2
UHD_INTERRUPT_NB_BANK	1, 2, 3	USBB	Reduces or increases interrupt endpoint buffering. Default value if not defined: 1
UHD_INT_LEVEL	0 to 3	USBB USBC	Sets the USB interrupt level on AVR32 cores. Default value if not defined: 0 (recommended)



Define name	Values	UHD	Description
UHD_START_MODE_MANUAL	Only defined	USBB USBC	By default, when the ID pin is available (USB_ID defined), the host or device mode are stopped and started automatically when the ID pin changes. This automatically start can be disabled, but the automatically stop mode remains.

Table 7-3. USB host driver configuration in conf_board.h.

Define name	Values	UHD	Description
USB_ID	AVR32_USBB_USB_ID_x_x	AVR32 - USBB	Define the input pin connected to ID pin from USB connector. This enables usage of USB ID pin from OTG connector. If not defined, the ID pin usage is disabled.
USB_ID	AVR32_USBC_USB_ or AVR32_USBC_USB_x	AVR32 - USBC	Define the input pin connected to ID pin from USB connector. This enables usage of USB ID pin from OTG connector. If not defined, the ID pin usage is disabled.
USB_VBOF	AVR32_USBB_USB_VBOF_x_x	AVR32 - USBB	Define the output pin connected to enable pin of V _{BUS} generator. This enables usage of V _{BUS} control by the USB hardware interface. If not defined, the V _{BUS} control is disabled.
USB_VBOF	AVR32_USBC_USB_VBOF or AVR32_USBC_USB_VBOF_x	AVR32 - USBC	Define the output pin connected to enable pin of V _{BUS} generator. This enables usage of V _{BUS} control by the USB hardware interface. If not defined, the V _{BUS} control is disabled.
USB_VBOF_ACTIVE_LEVEL	LOW or HIGH	AVR32 - USBB AVR32 - USBC	Active level of the USB_VBOF output pin.

Table 7-4. USB host/device driver configuration in project.

Define name	Values	UHD	Description
UHD_ENABLE	Defined	USBB	These defines allow to know if the dual role (host & device) is enabled.
UDD_ENABLE	Defined	USBC	

8 USB HUB support

The USB host stack is able to support a USB tree through a USB HUB. However, the USB hardware interface provided on Atmel parts can limit the USB tree.

Table 8-1. USB HUB hardware limitation.

USB hardware interface	Description
USBB	Number of pipes limited to seven or eight, but control endpoint multiplexed in one pipe. Does not support multiple USB speeds in same time on a USB tree.
USBC	Number of pipes limited to seven, but control endpoint multiplexed in one pipe. Supports pipe multiplexing ⁽¹⁾ . Does not support multiple USB speeds in same time on a USB tree.

Note: 1. To avoid the limitation due to pipes number a multiplexing of the pipes is possible. However, the transfer scheduling must be done by software (UHD driver) instead of by USB hardware.



9 Power consumption

The power modes available on Atmel products are supported by the USB hardware. All USB drivers implement the feature from sleepmgr service. Thus, the sleepmgr service initialization *sleepmgr_init()* must be called by the application prior any calls to the USB stack.

9.1 AVR32 core

On AVR32 core, the clocks and oscillators can be automatically switched off during idle periods by using the sleep instruction on the CPU.

Table 9-1. Sleep level supported in by USBB and USBC host drivers.

USB state		Maximum sleep mode authorized
ID pin no active (wait host mode)		STATIC
No V_{BUS} (wait V_{BUS} event)		STOP or FROZEN on UTMI parts (UTMI = high speed capability)
V_{BUS} present and device disconnected		IDLE
V_{BUS} present and device connected	Suspend	STATIC
	IDLE without DMA	IDLE
	IDLE + DMA running	IDLE

NOTE

The sleep modes listed here may not be reached if any other software module than the USB stack requires a less profound sleep mode.

10 Table of Contents

<i>Atmel AVR4950: ASF - USB Host Stack</i>	1
<i>Features</i>	1
<i>1 Introduction</i>	1
<i>2 Abbreviations</i>	2
<i>3 USB host application notes</i>	3
<i>4 Organization</i>	4
4.1 Overview.....	4
4.2 Memory Footprint.....	4
4.3 USB host stack files.....	4
4.3.1 Common files for all products.....	5
4.3.2 UHD files depending on selected products.....	5
4.3.3 Specific file for each application.....	5
<i>5 Application programming interface</i>	6
5.1 External API from UHC.....	6
5.2 Internal APIs.....	8
5.3 Device connected information.....	11
<i>6 USB host stack behavior</i>	12
6.1 Dual roles.....	12
6.2 V_{BUS} management.....	15
6.3 Device connection.....	16
6.4 Device enumeration.....	17
<i>7 Configuration</i>	19
7.1 USB host configuration.....	19
7.2 USB host interface configuration.....	19
7.3 USB host drivers configuration.....	19
<i>8 USB HUB support</i>	21
<i>9 Power consumption</i>	22
9.1 AVR32 core.....	22
<i>10 Table of Contents</i>	23

**Atmel Corporation**

2325 Orchard Parkway
San Jose, CA 95131
USA

Tel: (+1)(408) 441-0311

Fax: (+1)(408) 487-2600

www.atmel.com

Atmel Asia Limited

Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road

Kwun Tong, Kowloon

HONG KONG

Tel: (+852) 2245-6100

Fax: (+852) 2722-1369

Atmel Munich GmbH

Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY

Tel: (+49) 89-31970-0

Fax: (+49) 89-3194621

Atmel Japan

16F, Shin Osaki Kangyo Bldg.
1-6-4 Osaki Shinagawa-ku
Tokyo 104-0032

JAPAN

Tel: (+81) 3-6417-0300

Fax: (+81) 3-6417-0370

© 2012 Atmel Corporation. All rights reserved.

Atmel®, Atmel logo and combinations thereof, AVR®, AVR Studio®, megaAVR®, and others are registered trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.