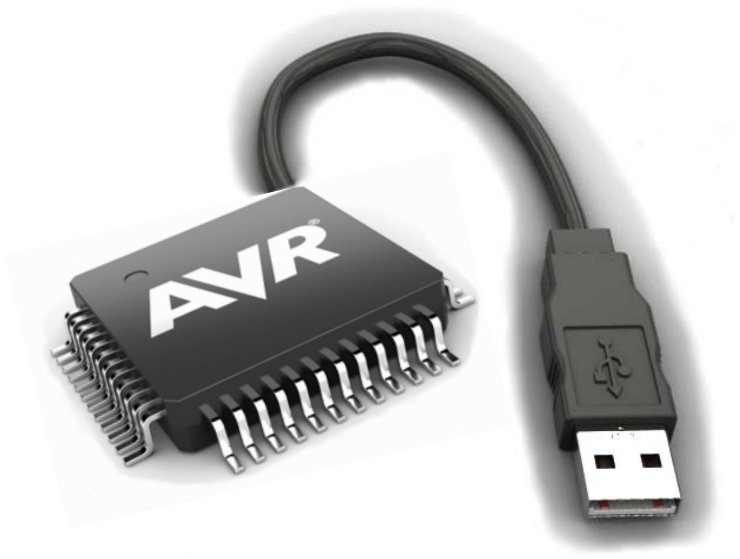
AVR4900: ASF - USB Device stack

Features

- **USB 2.0 compliance**
  - **USB Chapter 9 certified**
  - **Control, Bulk, Isochronous and Interrupt transfer types**
  - **Low Speed (1.5Mbit/s), Full Speed (12Mbit/s), High Speed (480Mbit/s) data rates**
- **Small stack size frees space for main application**
- **Real time (OS compliance, no latency)**
- **Supports 8-bit and 32-bit AVR® platforms**
- **USB DMA support increases speed performance**
- **Supports most USB classes and ready to use (HID, CDC, MSC, PHDC, AUDIO)**

1 Introduction

This document introduces the USB device 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 application. A full description of this stack is available in this document. No specific knowledge is required to use it except basic USB knowledge.



Atmel Microcontrollers

Application Note

Rev. 8360C-AVR-09/11

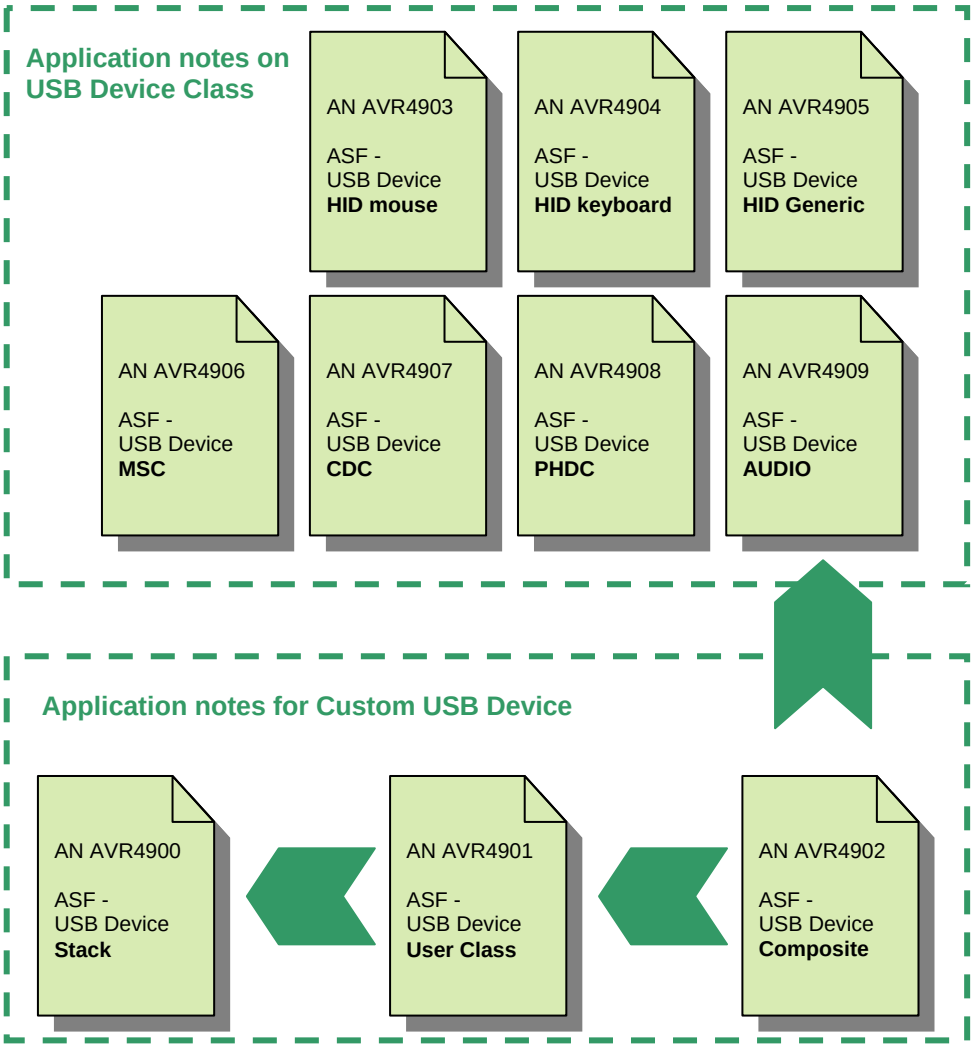
2 Abbreviations

- 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
- FS: USB Full Speed
- HID: Human interface device
- HS: USB High Speed
- UDC: USB Device Controller
- UDD: USB Device Descriptor
- UDI: USB Device Interface
- USB: Universal Serial Bus
- MSC: Mass Storage Class
- PHDC: Peripheral Health Device Class
- sleepmgr : Sleep management service from ASF
- ZLP: Zero length packet

3 USB Device Application Notes

Several USB device examples are provided by Atmel. For each example, Atmel provides an Application Note.

Figure 3-1. USB device application notes.



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

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

The New Class and Composite USB Device application notes are designed for advanced USB developers.

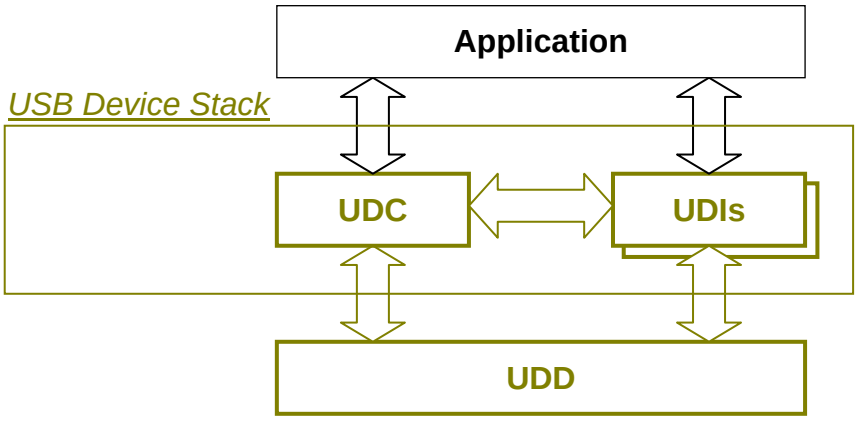
4 Organization

4.1 Overview

The USB Device stack is divided into three parts:

- USB Device Controller (UDC) provides USB Chapter 9 compliance
- USB Device Interface (UDI) provides USB Class compliance
- USB Device Driver (UDD) provides the USB interface for each AVR product

Figure 4-1. USB device stack architecture.



4.2 Memory Footprint

The USB device stack footprint depends on:

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

These parameters give many values, and in average the USB Device stack does not exceed 10Kbytes of FLASH and 1Kbytes of RAM using compiler high optimization level.

4.3 USB Device stack files

The USB Device Stack files are available in Atmel® AVR Studio® 5.

Atmel® AVR Studio® 5 allows creation of a *New Example Project*. Thus, in example list the USB Device examples beginning by *USB Device* (Select *Technology USB* to reduce example list). Note, the example with text *(from ASF V1)* does not corresponding at this USB Device Stack.

4.3.1 Common files for all AVR products

Files	***Paths***
• Defines USB constant usb_protocol.h (from usb.org) usb_atmel.h (from Atmel)	common/services/usb/
• UDC files udc.c/h udc_desc.h udi.h udd.h	common/services/usb/udc/
• Classes Protocols files usb_protocol_*foo*.h	common/services/usb/class/foo/
• UDI files udi_*foo*.c/h udi_*foo*_desc.c udi_*foo*_conf.h	common/services/usb/class/foo/device/

4.3.2 UDD files depending on selected AVR products

- avr32/drivers/usbb/usbb_device.c/h
- avr32/drivers/usbb/usbb_otg.h
- avr32/drivers/usbc/usbc_device.c/h
- avr32/drivers/usbc/usbc_otg.h

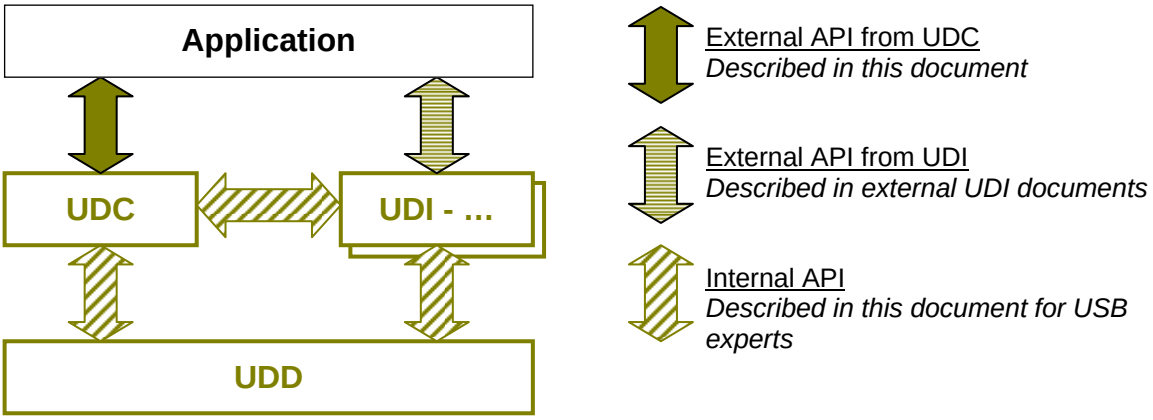
4.3.3 Specific file for each application

- Application file
  usb_conf.h (This configuration file is mandatory)

5 Application programming interface

This section describes all API except the UDI API which has its own document.

Figure 5-1. USB blocks.



5.1 External API from UDC

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

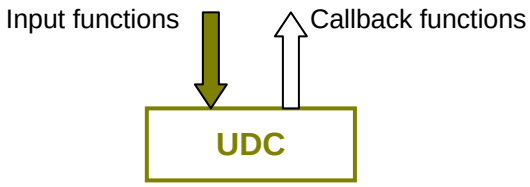


Table 5-1. External API from UDC – Input.

Declaration	Description
udc_start()	Start USB device stack
udc_stop()	Stop USB device stack
udc_attach() udc_detach()	Authorize the device enumeration or not. Enable pull-up on DM or DP.
udc_wakeup()	Wakeup the USB device

All UDC callbacks are optional and defined by user in usb_conf.h for each application.

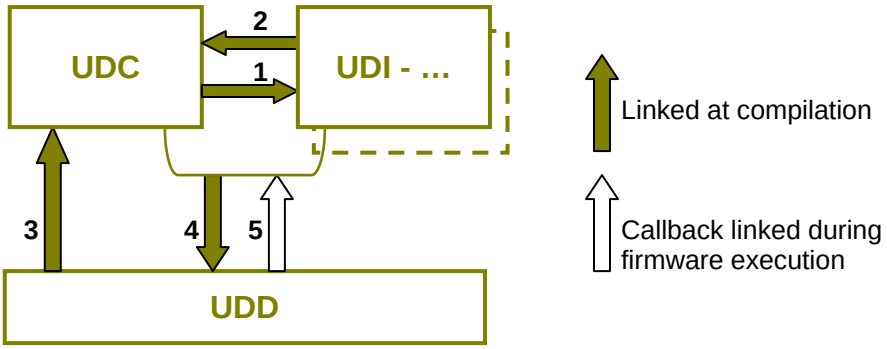
Table 5-2. External API from UDC – Callback.

Define name	Description
UDC_VBUS_EVENT(bool b_present)	To notify VBUS level change (only if USB hardware includes VBUS monitoring)
UDC_SUSPEND_EVENT()	Called when USB enters in suspend mode
UDC_RESUME_EVENT()	Called when USB wakes up
UDC_SOF_EVENT()	Called for each received SOF each 1 ms Note: Available in High and Full speed mode
UDC_REMOTEWAKEUP_ENABLE()	Called when USB host requests to enable/disable remote-wakeup feature when the device supports it
UDC_REMOTEWAKEUP_DISABLE()	
UDC_GET_EXTRA_STRING()	When a extra string descriptor must be supported (other than manufacturer, product and serial string)
UDC_SPECIFIC_REQUEST()	When a specific device setup request must be supported

5.2 Internal APIs

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

Figure 5-2. Internal USB device API overview.



Note: Numbers are references for tables below.

Table 5-3. UDI input from UDC (1).

Declaration	Description
bool (*__enable__)()	Called by UDC to enable/disable a USB interface
void (*__disable__)()	
bool (*__setup__)()	Called when a USB setup interface request is received
uint8_t (*__getsetting__)()	Called by UDC to obtain the current alternate setting of a USB interface
uint8_t (*__sof_notify__)()	Called by UDC to notify a SOF event at USB interface enabled

Note: The UDI API is linked with the UDC module via the UDC descriptor configuration file (see section 7.2)

Table 5-4. UDC input from UDI (2).

Declaration	Description
usb_iface_desc_t* **udc_getiface**()	Gives the USB interface descriptor selected by UDC when UDI is called (Table 5-3. UDI input)

Table 5-5. UDC input from UDD (3).

Declaration	Description
void **udc_reset**()	Called when reset bus state occurs
bool **udc_process_setup**()	Called when a setup packet is received

Table 5-6. UDD input (4).

Declaration	Caller	Description
void **udd_enable**()	UDC	Enables/disables the USB device mode
void **udd_disable**()	UDC	
void **udd_attach**() void **udd_dettach**()	UDC	Inserts or removes pull-up on USB line
void **udd_set_address**(uint8_t add)	UDC	Changes/returns the USB device address
uint8_t **udd_getaddress**()	UDC	
bool **udd_is_high_speed**()	UDC/UDI	In case of USB HS device, then checks speed chosen during enumeration
uint16_t **udd_get_frame_number**()	Application	Returns the current start of frame number
udd_send_wake_up()	Application	The USB driver sends a resume signal called “Upstream Resume”
bool **udd_ep_alloc**(usb_ep_id_t ep, uint8_t bmAttributes, uint16_t wMaxPacketSize)	UDC	Enables/disables endpoints
udd_ep_free(usb_ep_id_t)	UDC	
bool **udd_ep_clear_halt**(usb_ep_id_t)	UDC/UDI	Clears/sets/gets the endpoint state (halted or not)
bool **udd_ep_set_halt**(usb_ep_id_t)		
bool **udd_ep_is_halted**(usb_ep_id_t)		
bool **udd_ep_wait_stall_clear**(udd_ep_id_t endp, udd_callback_nohalt_t callback)		Registers a callback to call when endpoint halt is removed
bool **udd_ep_run**(usb_ep_id_t endp, bool b_shortpacket, uint8_t *buf, uint32_t u32_size_buf, udd_callback_trans_t callback)	UDI	Starts/stops a data transfer in or out on an endpoint Note: The control endpoint is not authorized here
udd_ep_abort(usb_ep_id_t endp)	UDI	

Table 5-7. UDD callback (5).

Declaration	Description
typedef void (***udd_callback_nohalt_t**) (void);	Called when the halt on endpoint is removed. This one is registered via **udd_ep_wait_stall_clear()**.
typedef void (***udd_callback_trans_t**) (udd_ep_status_t status, iram_size_t nb_transfered)	Called when a transfer request is finished or cancelled. This one is registered via **udcdrv_ep_run()**.

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

Declaration	Caller	Description
uint16_t **udd_get_microframe_number**()	Application	Returns the current micro start of frame number
udd_test_mode_j()	UDC	Features to test the USB HS device. These are requested to run a USB certification.
udd_test_mode_k()	UDC	
udd_test_mode_se0_nak()	UDC	
udd_test_mode_packet ()	UDC	

The global variable ***udd_g_ctrlreq*** is declared by UDD, and contains two parts:

- Values updated by UDD and used by UDC & UDIs (Table 5-9)
- Values updated by UDC & UDIs and used by UDD (Table 5-10)

Outside the UDD, this variable is processed by **udc_process_setup**() for UDC and ***setup**() for UDI (see Figure 6-2).

Table 5-9. *udd_g_ctrlreq* field updated by UDD.

Declaration	Description
usb_setup_req_t **req**	Values included in SETUP packet and used to decode request.
uint8_t ***payload**	The content of the buffer is sent or filled by UDD. Can be NULL if **u16_size** is equal to 0.

Table 5-10. *udd_g_ctrlreq* updated by UDC or UDI.

Declaration	Description
uint8_t ***payload**	Pointer value of the buffer to send or fill
uint16_t **u16_size**	Buffer size to send or fill It can be 0 when no DATA phase is needed
bool **over_under_run**(void)	Called by UDD when the buffer given (.***payload***) is full or empty. Can be NULL
void ***callback**(void)	Called by UDD when the setup request is finished (setup+data+ZLP). Can be NULL

6 Behavior

This section aims to answer advanced USB usage questions and introduce the USB device stack behavior.

The stack implementation provides a UDC stack which is managed only by USB interruption. This solution guarantees lowest latency, does not require any wait loop, and guarantees OS compatibility.

How long can the USB interrupt routine be blocked?

The USB interrupt routine is enabled with low priority, and can be blocked by another interrupt routine with higher priority or a critical code section.

Following the USB specification, the AVR USB hardware and software should not have any time limitation except the USB “Set Address” request (performed during USB enumeration phase to assign USB address).

Table 6-1. Set Address timings.

USB host	Time maximum (1)
Specification	*2ms*
USB org certification tools	12ms
Windows® XP	48ms
Windows 7, Vista	32ms
Mac Mini OSX 10.5.8	77ms
Ubuntu 8.04, Ubuntu 9, Open Suse 11.1	29ms
Fedora 9, Fedora 10	24ms

Note: 1. These timings can depend on USB host hardware, but gives a range. Also, this time includes the time for setup retry.

The USB host uses a timeout to reset a non-answering USB device (this time is not specified by the USB specification).

Hereunder examples of operating systems’ timeouts:

Table 6-2. OS timeout.

	Timeout				
	Control endpoint		Mass storage		
USB host	Data phase	ZLP phase	CBW	Data read	CSW
Specification	*No timeout*				
Windows XP	5.3s	5.3s	19s	9.3s	9.3s
Windows 7, Vista	5.3s	5.3s	19s	160s/60s	160s/60s
Mac Mini OSX 10.5.8	5.9s	5.6s	11s	31s	22s
Ubuntu 8.04, Ubuntu 9, Open Suse 11.1	5s	5s	30s	30s	30s
Fedora 9, Fedora 10	5s	5s	30s	60s	30s

The following figures describe the interaction between the different layers.

Figure 6-1. USB device startup and stop.

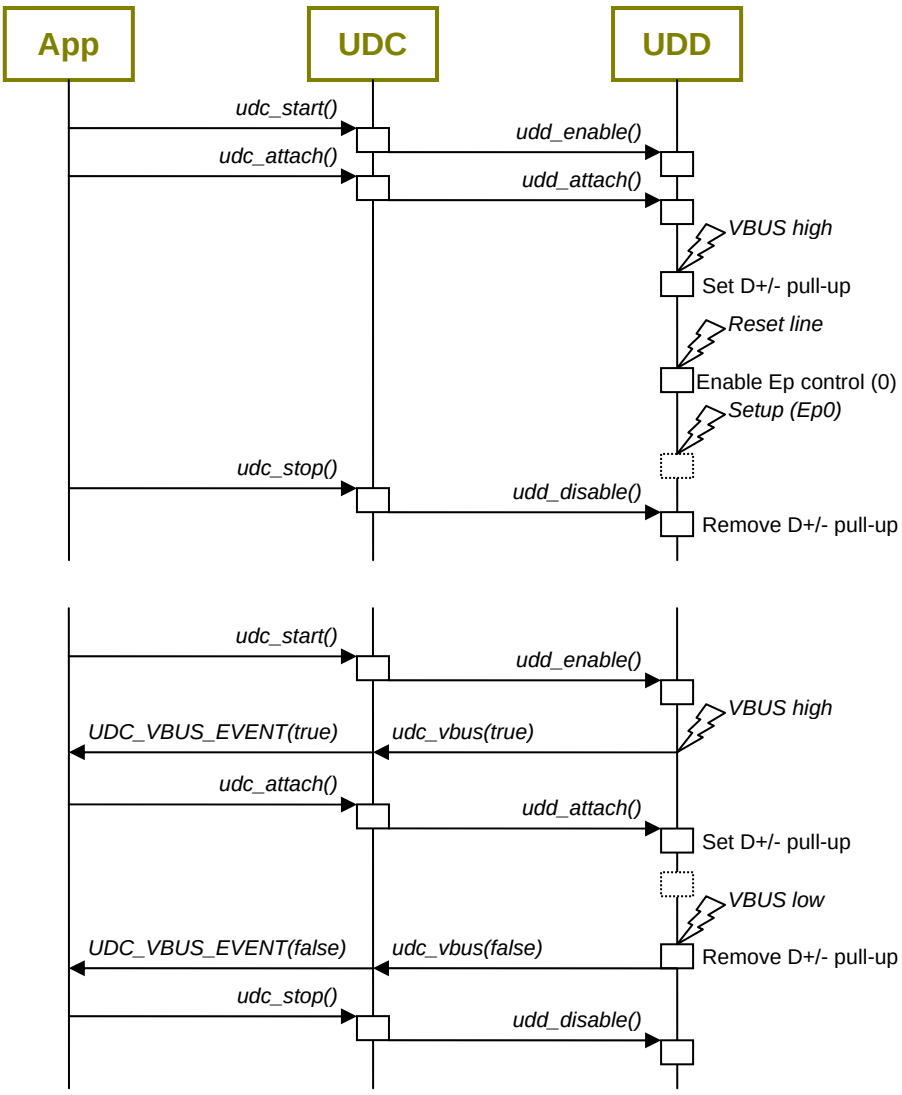
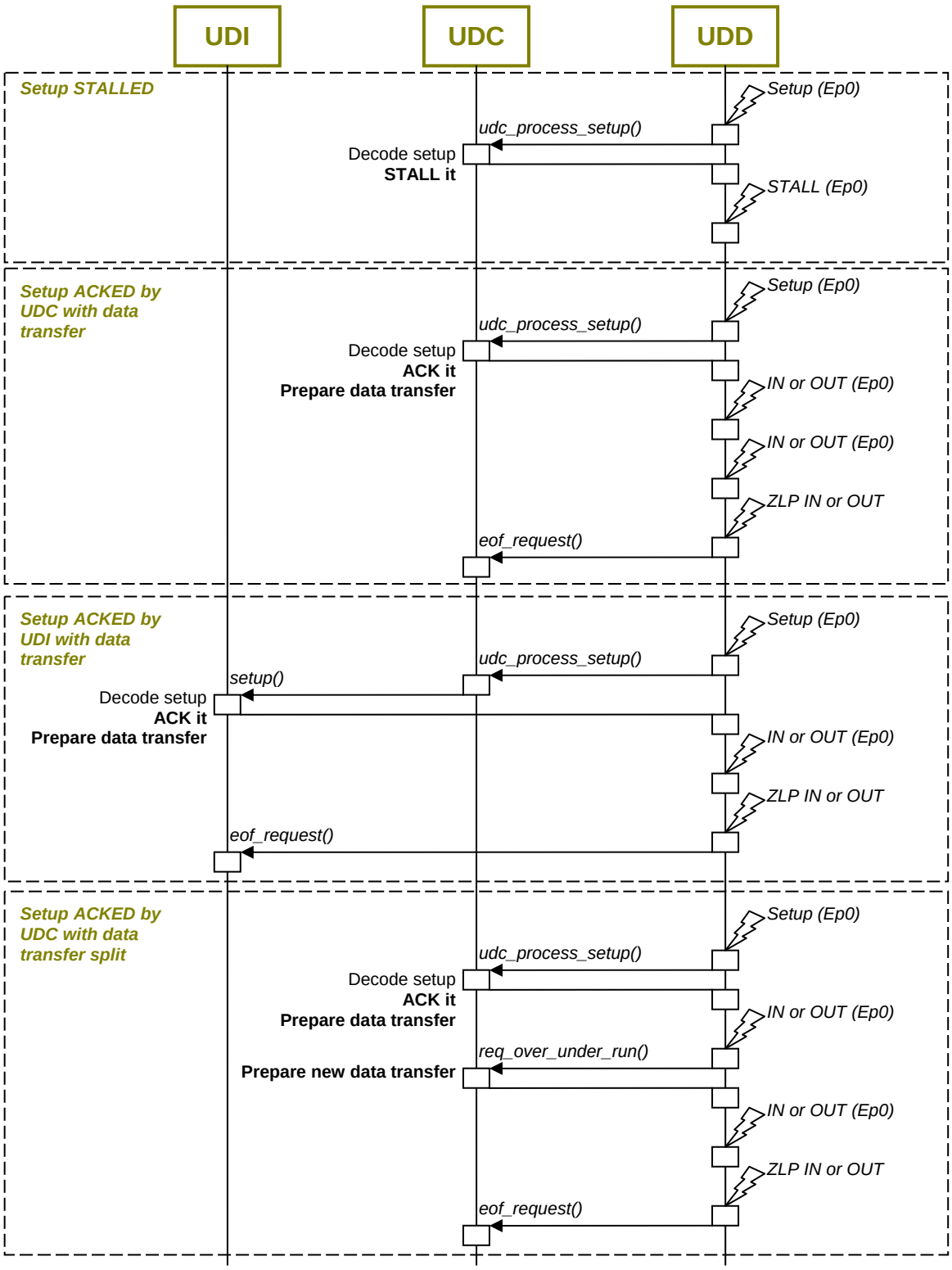
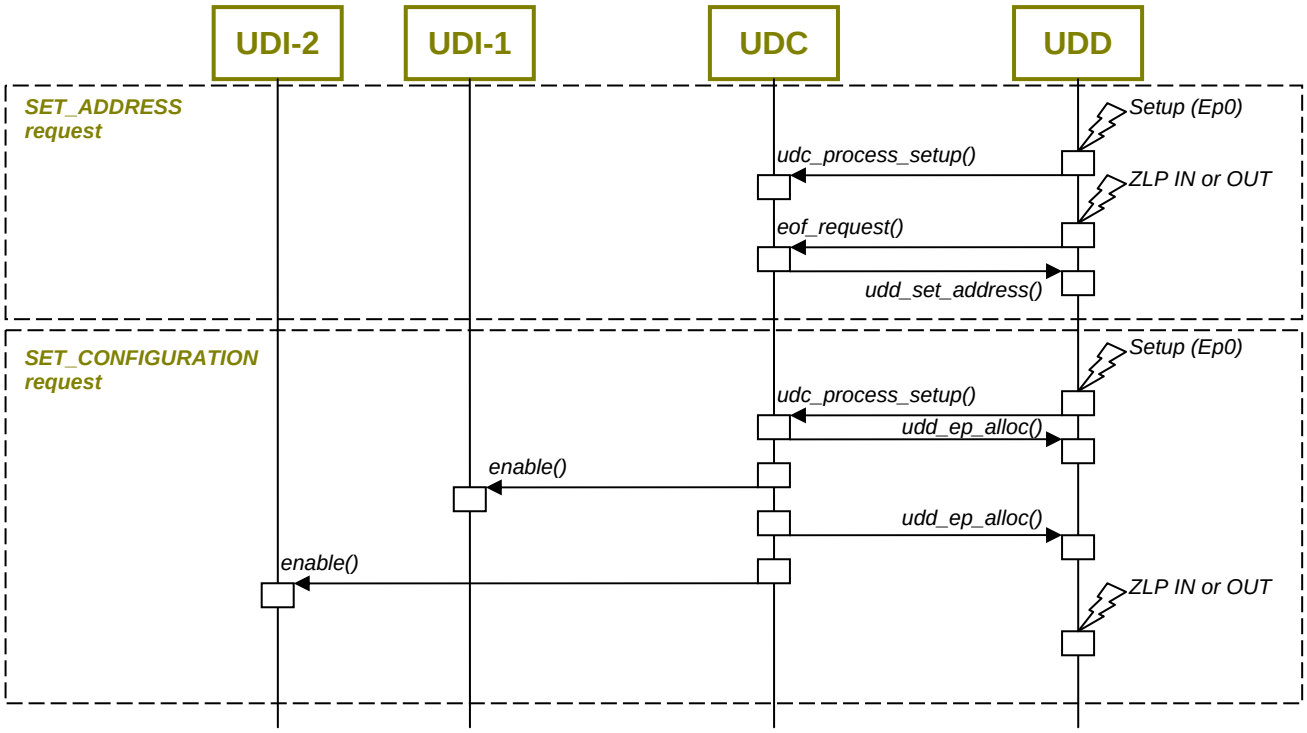


Figure 6-2. Management of control endpoint.



NOTE The udd_g_ctrlreq variable is used to communicate between UDD and UDC/UDIs (see Table 5-9 and Table 5-10).

Figure 6-3. Typical enumeration.



7 Configuration

The configuration is divided into two sections: Application and USB Descriptors.

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

The conf_usb.h file must define the following configurations:

- USB device configuration
- USB interface configuration
- USB driver configuration

The USB Descriptors configuration is requested when the default configuration provided by Atmel is not used. This configuration information is available for advanced USB users.

7.1 USB configuration

7.1.1 USB device 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 Device Configuration.

Define name	Type	Description
USB_DEVICE_VENDOR_ID	Word	Vendor ID provided by USB org (ATMEL 0x03EB)
USB_DEVICE_PRODUCT_ID	Word	Product ID (referenced in usb_atmel.h)
USB_DEVICE_MAJOR_VERSION	Byte	Major version of the device
USB_DEVICE_MINOR_VERSION	Byte	Minor version of the device
USB_DEVICE_MANUFACTURE_NAME (2)	String (1)	Static ASCII name for the manufacture
USB_DEVICE_PRODUCT_NAME (2)	String (1)	Static ASCII name for the product
USB_DEVICE_SERIAL_NAME (2)	String (1)	Static ASCII name to enable and set a serial number
USB_DEVICE_GET_SERIAL_NAME_POINTER() (2)	const uint8_t* *function*(void)	Give a pointer on a dynamic ASCII name to enable and set a serial number. Require USB_DEVICE_GET_SERIAL_NAME_LENGTH and ignore USB_DEVICE_SERIAL_NAME.
USB_DEVICE_GET_SERIAL_NAME_LENGTH() (2)	uint8_t *function*(void)	Give the length of dynamic ASCII name used to enable a serial number.
USB_DEVICE_POWER	Numeric	Maximum device power (mA)
USB_DEVICE_ATTR	Byte	USB attributes to add to enable feature: - USB_CONFIG_ATTR_SELF_POWERED - USB_CONFIG_ATTR_REMOTE_WAKEUP Note: If remote wake feature is enabled, then defines *remotewakeup* callbacks (see Table 5-2)
USB_DEVICE_LOW_SPEED (2)	Only defined	Force the USB device to run in Low Speed
USB_DEVICE_HS_SUPPORT (2)	Only defined	Authorize the USB device to run in High Speed
USB_DEVICE_MAX_EP	Byte	Define the maximum endpoint number used by the device (don't include control endpoint)

Notes:
1. Examples of String syntax:
   #define USB_DEVICE_MANUFACTURE_NAME "ATMEL".
   The Define can be omitted, thus the string is removed of USB enumeration.
2. Optional configuration. Comment the define statement to disable it (ex: // #define USB_DEVICE_X).

7.1.2 USB Interface configuration

The UDI configurations are described in USB Device Class application notes.

7.1.3 USB Drivers configuration

The following configuration must be included in the conf_usb.h file of the application.

The AVR products provide specific hardware features which can be enabled here.

Table 7-2. USB Device Driver Configuration.

Define name	Values	UDD	Description
UDD_NO_SLEEP_MGR	Only defined	**All**	Remove the management of sleepmgr service
UDD_ISOCHRONOUS_NB_BANK	1, 2, 3	**AVR32 - USBB**	Reduces or increases isochronous endpoint buffering. Default value: 2
UDD_BULK_NB_BANK	1, 2, 3	**AVR32 - USBB**	Reduces or increases bulk endpoint buffering. Default value: 2
UDD_INTERRUPT_NB_BANK	1, 2, 3	**AVR32 - USBB**	Reduces or increases interrupt endpoint buffering. Default value: 1
UDD_USB_INT_LEVEL	0 to 3	**AVR32 - USBB** **AVR32 - USBC**	Sets the USB interrupt level on AVR32 core. Default value: 0 (recommended)

7.2 USB Descriptors

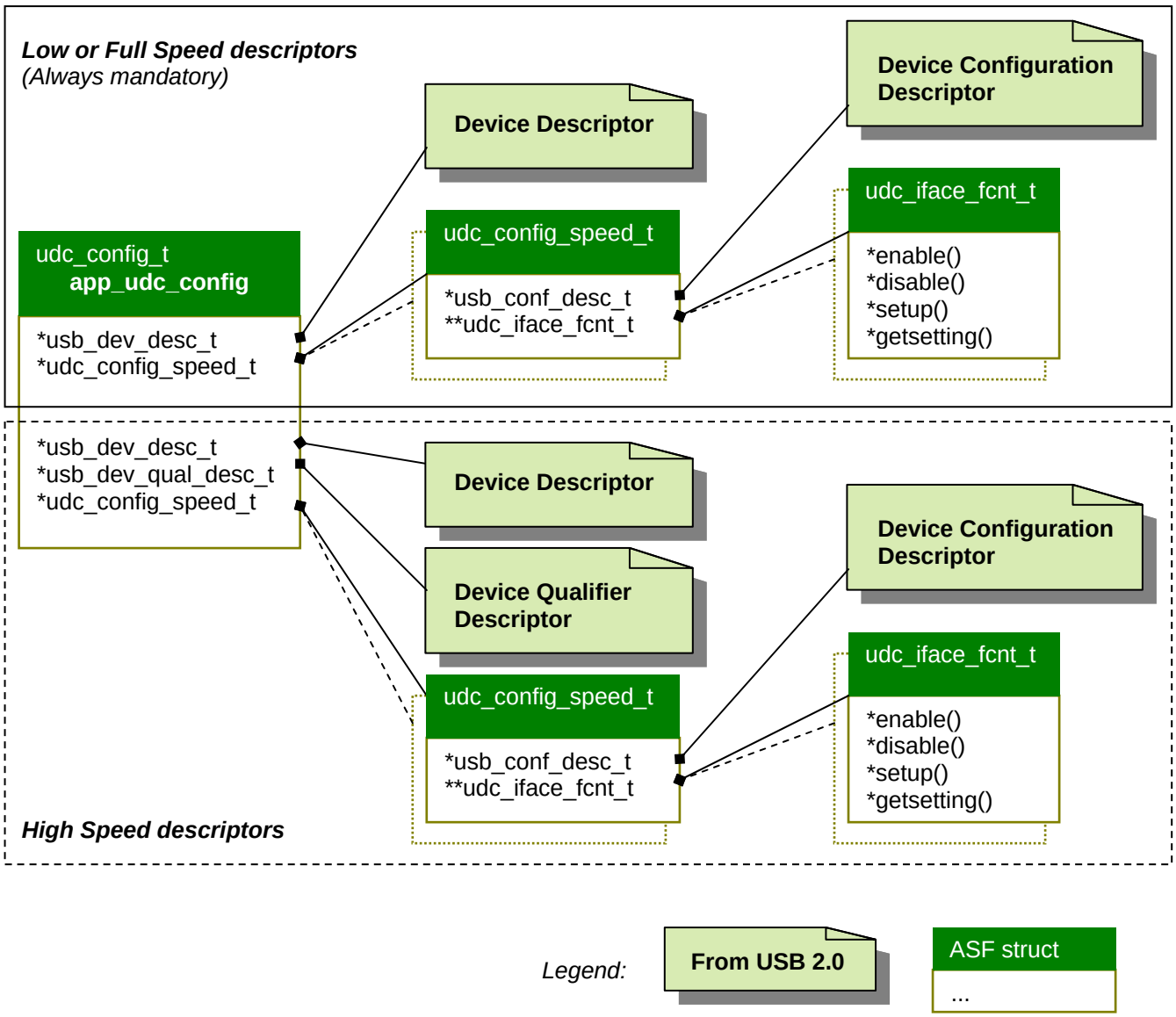
This section is oriented to USB developers who want to create a new UDI or a composite USB device.

The USB classes that are already provided by ASF include default USB Device descriptors. These descriptors are defined in the UDI files udi_foo_desc.c and udi_foo_conf.h, and allow an easy implementation described in all UDI application notes.

The descriptor file declares the global variable ***app_udc_config*** which includes:

- a device descriptor for each speed possible *(usb_dev_desc_t*)
- one device qualifier in case of High Speed device *(usb_dev_qual_desc_t)*
- a configuration descriptor for each configuration *(usb_conf_desc_x_t)*
- a link between UDI and configuration descriptor *(udc_iface_fcnt_t)*

Figure 7-1. USB descriptors.



For more information, see Atmel application note, “ASF - USB Device New Class.”

8 Power consumption

The power modes available on AVR products can be supported by the USB hardware according to USB line state. The USB drivers use the sleepmgr service to manage all these power save modes. When a USB application is created, the sleepmgr service initialization *sleepmgr_init*() is required.

8.1 USBB and USBC sleep modes

All AT32UC3 sleep modes are described in each AT32UC3 datasheet §Power Manager. The Sleep modes supported by USBB and USBC drivers are:

- in USB IDLE state, the driver needs of the USB clock and authorizes up to IDLE mode.
- in USB SUSPEND state, the driver does not need USB clock but requests a minimum timing for restarting clock. Thus, it is supported up to STATIC or STANDBY mode.
- VBUS monitoring used in USB Self-Power mode authorizes up to STOP mode.

Table 8-1. Maximum sleep levels supported in USB suspend state on AT32UC3.

USB Power Mode	USB Speed Mode	USB Clock Startup	Sleep mode authorized
Bus and self power	LS, FS	>10ms	STANDBY
Bus and self power	HS	>3ms	STANDBY
Self power	LS, FS	<=10ms	STOP
Self power	HS	<=3ms	STOP
Bus power	LS, FS	<=10ms	STATIC
Bus power	HS	<=3ms	STATIC

Note: Often an external oscillator is used to generate the USB clock. Thus, USB clock startup timing corresponding at oscillator startup timing.

The AT32UC3 family supports easily the power limit (2.5mA) in USB suspend mode.

Thus, for any Bus power device application, it is require to:

- Remove *USB_CONFIG_ATTR_SELF_POWERED* bit in *USB_DEVICE_ATTR* define from *conf_usb.h* file.
- Use an external oscillator with a low startup time.
  This value is specified in the board header via *BOARD_OSC0_STARTUP_US* define. Take care at startup time possibility and see OSCCTRL0 register possibility in AT32UC3 datasheets.
- An external low drop regulator is required to generate the 3.3V AT32UC3 power supply. When selecting this regulator, be sure its quiescent current does not consume a too large proportion of the global 2.5mA suspend current.

8.2 ATxmega sleep modes

All ATxmega sleep modes are described in each ATxmega datasheet §Power Manager. The Sleep modes supported by USB drivers are:

- in USB IDLE state, the driver needs of the USB clock and authorizes up to IDLE mode.
- in USB SUSPEND state, the driver does not need USB clock but requests a minimum timing for restarting clock. Thus, it is supported up to POWER DOWN or STANDBY mode.

Table 8-2. Maximum sleep levels supported in USB suspend state on ATxmega.

USB Power Mode	USB Speed Mode	USB Clock Startup	Sleep mode authorized
Bus and self power	LS, FS	>10ms	STANDBY
Bus and self power	LS, FS	<=10ms	POWER DOWN

Note: Often the internal oscillator is used to generate the USB clock. Thus, USB clock startup timing is < 10ms.

The ATxmega family supports easily the power limit (2.5mA) in USB suspend mode.

Thus, for any Bus power device application, it is require to:

- Remove *USB_CONFIG_ATTR_SELF_POWERED* bit in *USB_DEVICE_ATTR* define from *conf_usb.h* file.
- Use the internal oscillator to have a low startup time.
- An external low drop regulator is required to generate the 3.3V ATxmega power supply. When selecting this regulator, be sure its quiescent current does not consume a too large proportion of the global 2.5mA suspend current.

9 Changelog

9.1 8360C-08/12

1. Add new option to implement a dynamic serial number in **Error! Reference source not found.**.
2. Add ATxmega product information.
3. In features list, fix the High Speed 48Mbit/s by 480Mbit/s.

9.2 8360B-04/11

1. Updated all section concerning Power consumption.
2. Updated UDI and UDD APIs.

   See “sof_notify()” and udd_get_microframe_number() description.

9.3 8360A-12/10

Initial revision

10 Table of Contents

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, Milennium 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
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chou-ku, Tokyo 104-0033
JAPAN
Tel: (+81) 3523-3551
Fax: (+81) 3523-7581

8360C-AVR-09/11