AVR328: USB Generic HID Implementation

Features

- **Supported by all Microsoft® O/S from Windows® 98SE and later**
- **Simple PC Interface with Read/Write Functions**
- **Up to 64Kbytes/s Full Duplex Transfer**
- **Runs on any AVR® USB microcontroller**

1. Introduction

The aim of this document is to describe how to start and implement a USB application based on the HID class to transfer data between a PC and user equipment.

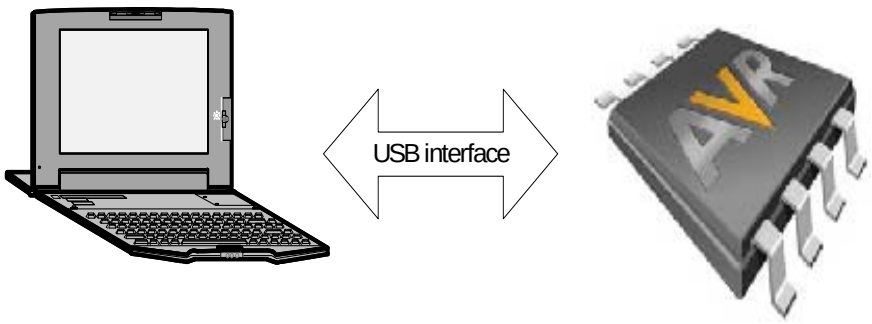
2. Description

The USB interface becomes very complex when the user data does not fit with USB standard classes (Mass Storage, Audio, Video...). Specific drivers must be developed, requiring a significant amount of development time.

Atmel has developed a solution to save time and development efforts. This solution is based on HID class. It ensures a full duplex transfer between the device and the PC (up to 64Kbytes/s). Adding to the data exchange, this application allows the user to upgrade firmware without any hardware setup.

The HID class is supported by all Microsoft O/S from Windows 98SE and later. It is also supported by most other O/S running on PCs (at time of publication).

A familiarity with the *USB Software Library for AT90USBxxx Microcontrollers* (Doc 7675, Included in the USB CD-ROM & Atmel website) and the HID specification (http://www.usb.org/developers/hidpage) is assumed.



8-bit AVR® Microcontrollers

Application Note

Rev. 7599C–AVR–07/08

3. Hardware Requirements

The generic HID application requires the following hardware:

1. AVR USB evaluation board (STK525, AT90USBKey, STK526...or your own board)
2. AVR USB microcontroller
3. USB cable (Standard A to Mini B)
4. PC running on Windows (98SE, ME, 2000, XP) with USB 1.1 or 2.0 host

4. In system programming and Device firmware Upgrade

To program the device you can use the following methods:

- The JTAG interface using the JTAGICE MKII
- The SPI interface using the AVRISP MKII
- The USB interface thanks to the factory DFU bootloader and Flip software
- The parallel programming using the STK500 or STK600

Please refer to the hardware user guide of the board you are using (if you are using Atmel starter kit) to see how to program the device using these different methods.

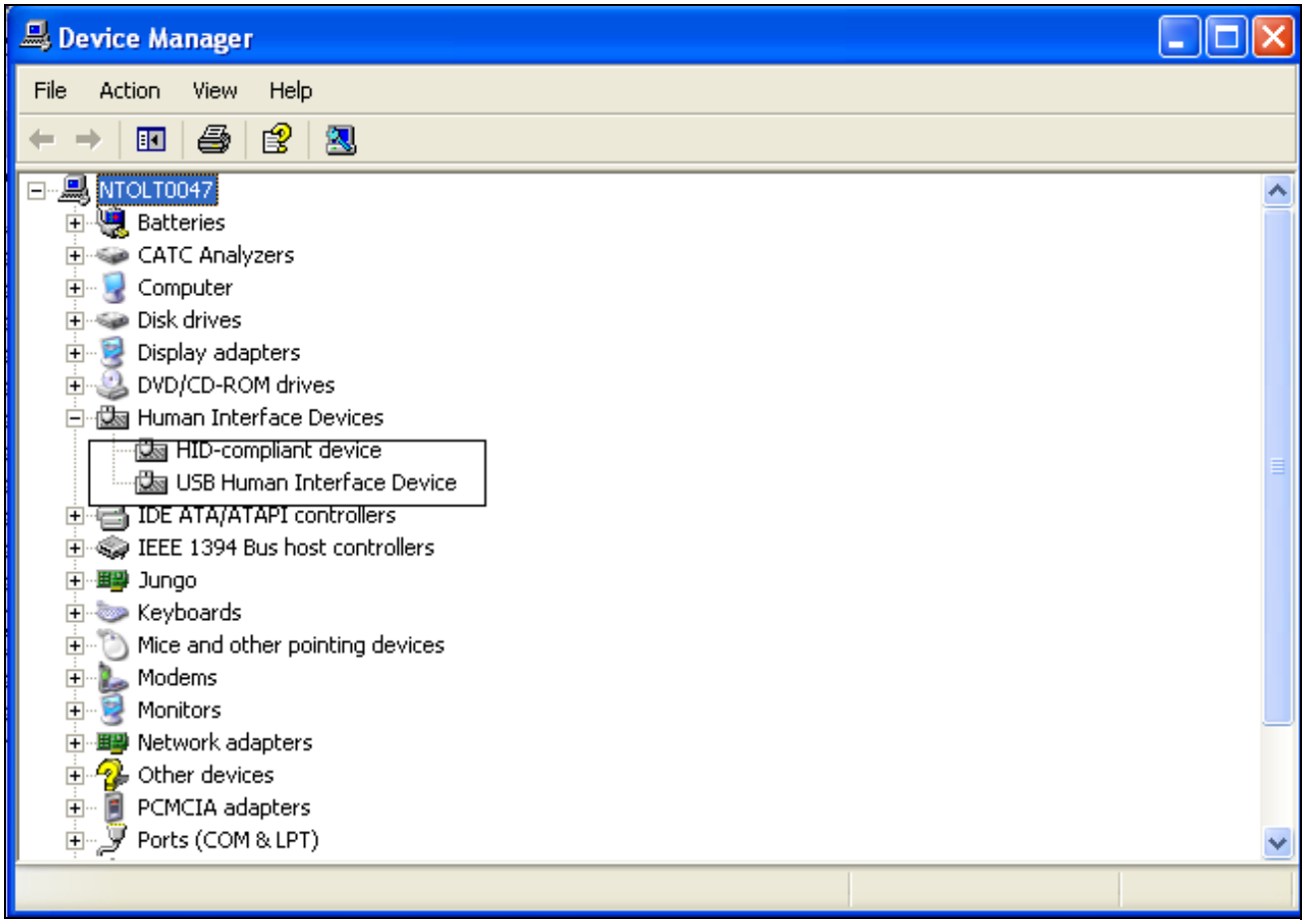
Please refer to Flip[1] help content to see how to install the USB driver and program the device through the USB interface.

Note: 1. Flip is a software provided by Atmel to allow the user to program the AVR USB devices through the USB interface (No external hardware required) thanks to the factory DFU bootloader.

5. Quick Start

Once your device is programmed with *generic hid hex* file, you can start the generic HID demo. Check that your device is enumerated as HID device (see Figure 5-1.), then launch the PC application (see Figure 5-2.) and start exchanging data between the PC and embedded application.

Figure 5-1. HID Enumeration

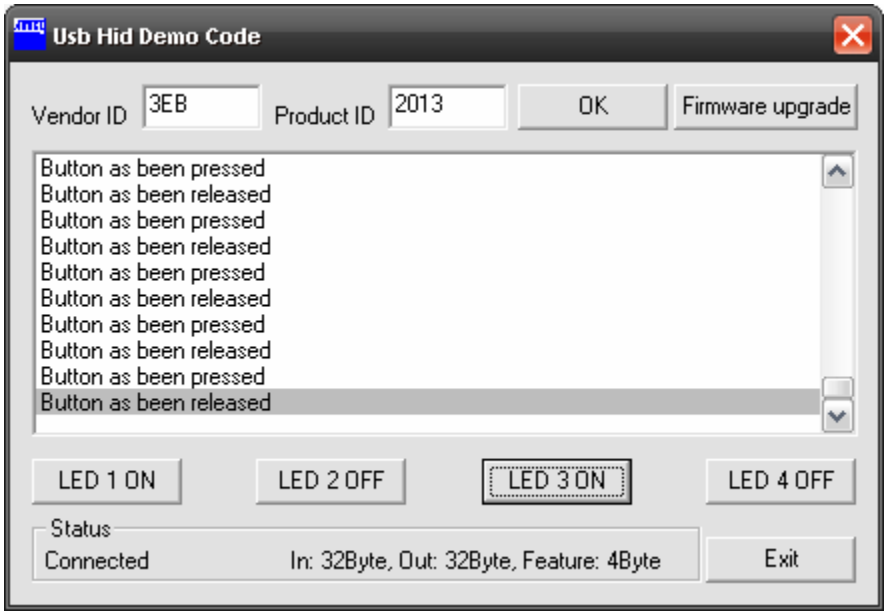


The aim of this demonstration is to demonstrate how to build a full duplex communication between a USB device and a USB host.

The IN communication consists of sending data from the device to the host. These data are sent when you move the joystick of the starter kit. The PC application (see figure below) will allow you to send data to the device to switch ON/OFF the LEDs of your starter kit.

Figure 5-2 shows the GUI used by the PC application:

Figure 5-2. GUI generic HID



Please refer to the Application note *USB PC Drivers Based on Generic HID Class* for further information regarding the PC application.

6. Application Overview

The generic HID application is a simple data exchange between a PC and the device.

The USB data exchange for this application is based on two interrupt endpoints (one IN and one OUT).

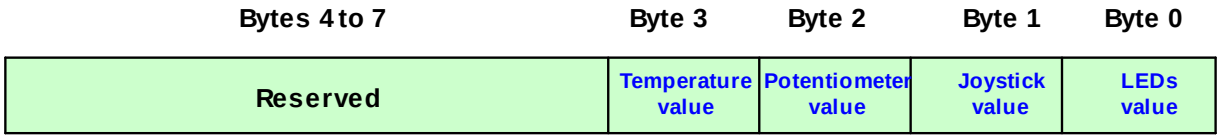
The PC asks the device if there is new data available each *P* time (polling interval time), the device will send data if it is available, otherwise it will send a NAK (No Acknowledge) to tell the PC that there is no data available. The data package sent from the device to the PC is called report IN.

To send data to the device, the PC checks if there is new data available for the application each *P* time (polling interval time). Once data is available, the PC sends it to the device. This data package is called report OUT.

These reports conatin a set of bytes which can be used by the user application to transfer data - depending on the requirement.

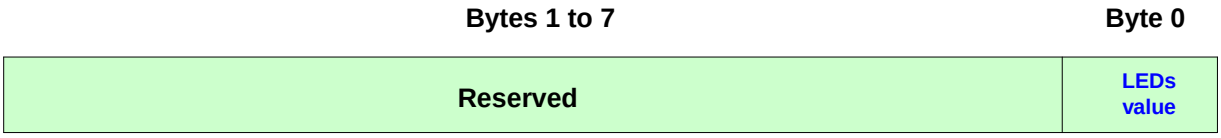
This demo is using the report IN and the report OUT with the structures below:

Figure 6-1. Report IN structure



- LEDs value: This byte contains the current state (ON/OFF) of the LEDs.
- Joystick value: This byte contains the joystick state (Active/Inactive) and the direction of the pointer.
- Potentiometer value: This byte contains the potentiometer value.
- Temperature value: This byte contains the temperature sensor value.

Figure 6-2. Report OUT structure



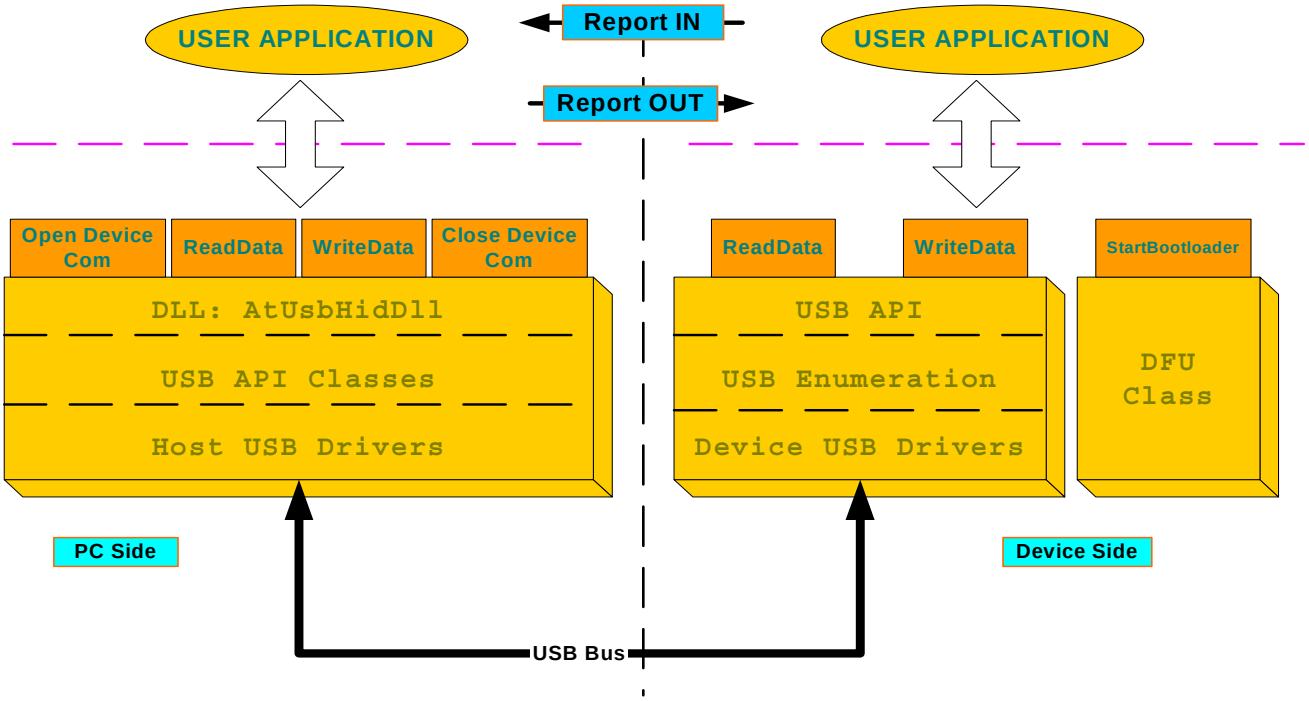
LEDs value: This byte contains the new state of the LEDs.

Note: In this application we use 8 bytes for the report IN/OUT size. This size can be changed by the user. There is no maximum value of the report size, but a maximum of 64 bytes can be sent at one time (see Section "Firmware", page 7).

The Generic HID device allows the user to send data through the endpoint 0 using the ‘setFeature’ function. This function sends a feature report with 4 bytes (the lenght can be modified by the user, refer to the section 7.3):

Figure 6-3. Feature Report

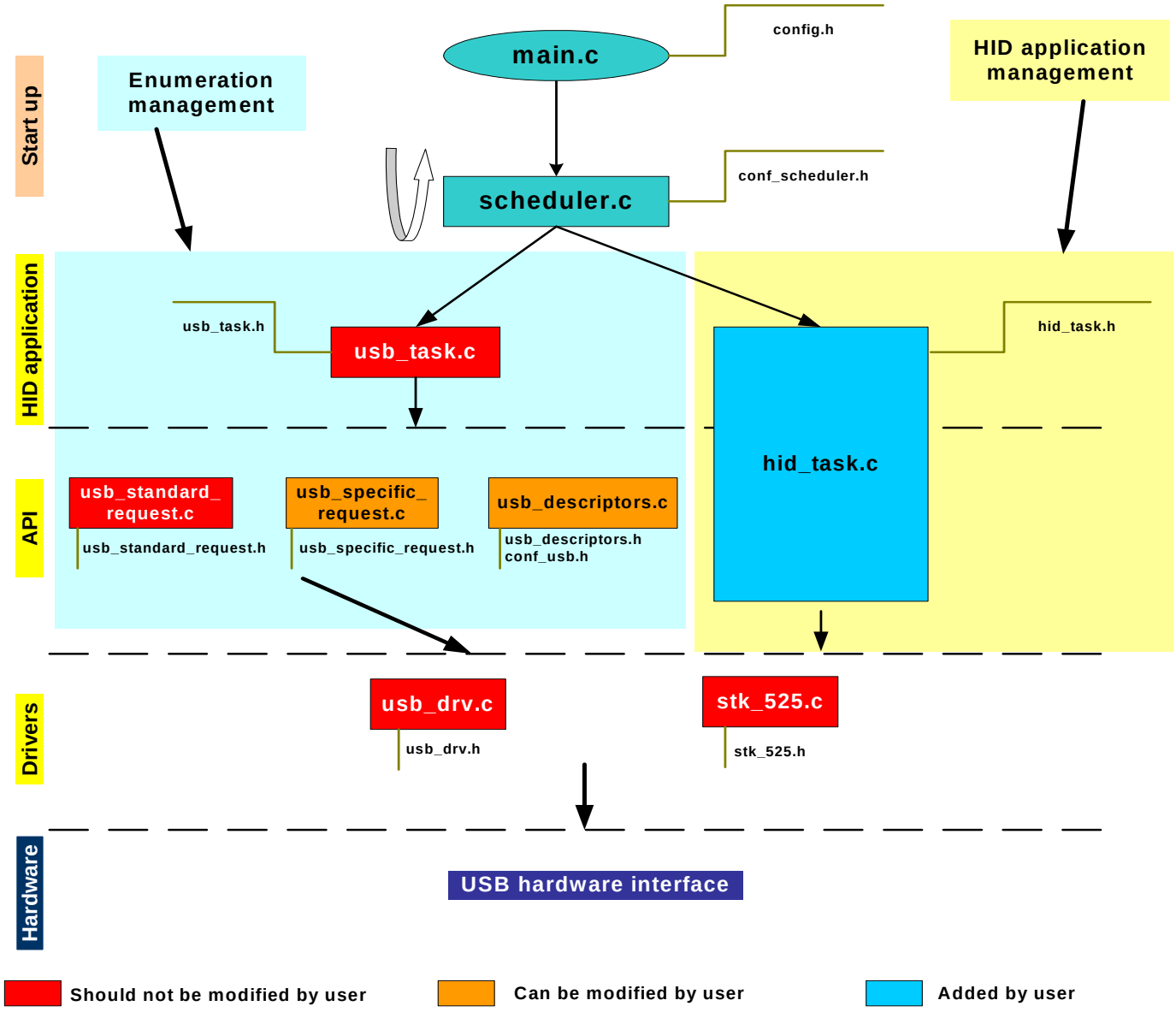
Figure 6-4. Application Overview



7. Firmware

As explained in the *USB Software Library for AT90USBxxx Microcontrollers* document (Doc 7675, included in the USB CD-ROM) all USB firmware packages are based on the same architecture (please refer to this document for more details).

Figure 7-1. Generic HID Firmware Architecture



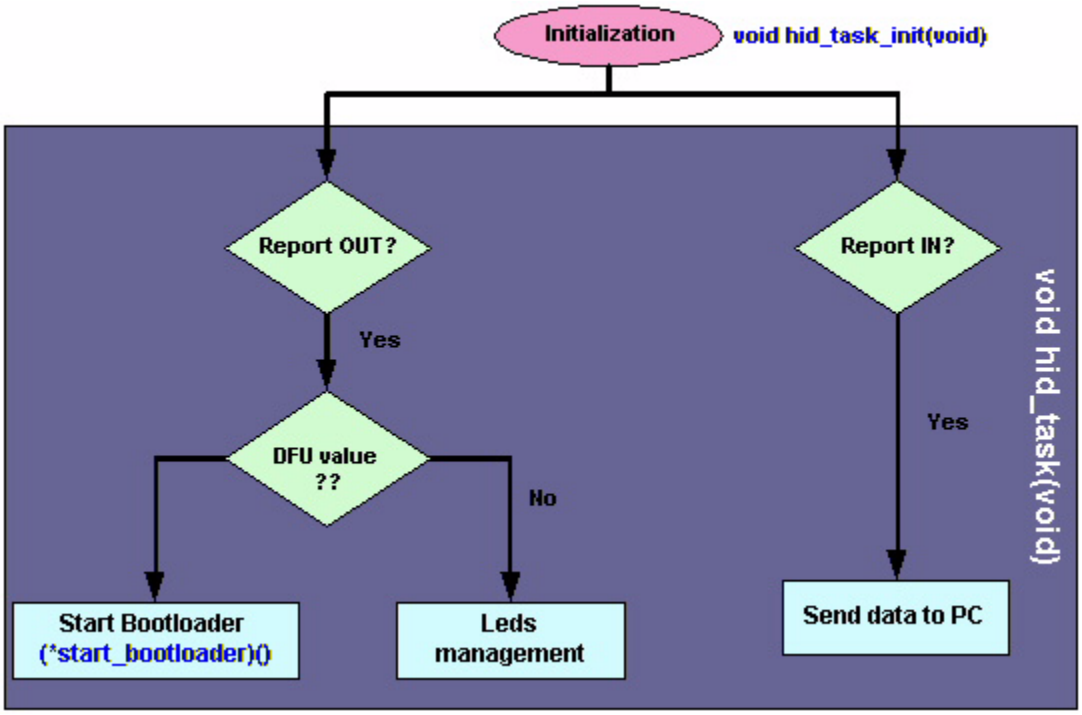
This section is dedicated to the HID module only. The customization of the files described hereafter allow the user to build his own Generic HID Application:

- hid_task.c
- usb_descriptor.h

7.1 hid_task.c

This file contains the functions to initialize the device, manage the data transfer with the PC, and start the bootloader when the user wants to upgrade his firmware.

Figure 7-2. Generic HID Application



7.1.1 hid_task_init

This function performs the initialization of the device parameters and hardware resources (joystick, potentiometer...).

7.1.2 (*start_bootloader)

This function pointer allows the launch of the bootloader.

7.1.3 hid_task

This function manages the data transfer, and the launch of the bootloader, once the DFU command is sent by the PC.

7.2 stk_52x.c.

This file contains all the routines to manage the STK52x board resources (Joystick, potentiometer, Temperature sensor, LEDs...). The user should not modify this file when using the STK52x board. Otherwise he has to build his own hadware management file.

7.3 Report Length Modification

As mentioned in Application Overview section, the user can modify the report IN/OUT length.

The report IN and report OUT lengths are defined in the file *usb_descriptor.h:*

```
#define LENGTH_OF_REPORT_IN 8
#define LENGTH_OF_REPORT_OUT 8
```

Please note that the report length should be equal or less than the endpoint size. If the report is bigger than the endpoint size you must send the report in several times.

To modify the endpoint size (maximum 64 bytes, since the HID class is using the interrupt transfer), you must modify the following parameters from the file *usb_descriptors.c*:

```
#define EP_IN_LENGTH 8
#define EP_OUT_LENGTH 8
```

And the ***SIZE_n*** parameters from the file *usb_specific_request.c,* and more specifically from the function *usb_user_endpoint_init():*

```
usb_configure_endpoint(EP_HID_IN,       \
                       TYPE_INTERRUPT,      \
                       DIRECTION_IN,   \
                       SIZE_8,         \
                       ONE_BANK,       \
                       NYET_ENABLED);
usb_configure_endpoint(EP_HID_OUT,       \
                       TYPE_INTERRUPT,      \
                       DIRECTION_OUT,   \
                       SIZE_8,         \
                       ONE_BANK,       \
                       NYET_ENABLED);
```

The ***SIZE_n*** can have the following values:

```
SIZE_8: 8 bytes
SIZE_16: 16 bytes
SIZE_32: 32
SIZE_64: 64 bytes
SIZE_128: 128 bytes
SIZE_256: 256 bytes
SIZE_512: 512 bytes
SIZE_1024: 1024 bytes
```

To modify the feature report, you must modify the following parameter from file *usb_descriptor.h*:

```
#define LENGTH_OF_REPORT_FEATURE 4
```

Like the report IN/OUT, this report ha no limit length. If its length is equal or less than the endpoint 0 size, it will be sent in one time, unless you must send it in several times. You may need to modify the endpoint zero length to send a bigger report. to do so, you must modify the following parameter from the *usb_descriptors.h*:

```
#define EP_CONTROL_LENGTH     8
```

The maximum size could be set to the endpoint 0 is 64 bytes (USB specification).

7.4 How to Handle the setFeature Request

The setFeature() function is provided by the DLL to allow the user to send a control command to the device. These commands are sent through the endpoint 0 and have to be treated as a set_report request (refer to the HID specification for further information).

The set_report request has to be managed by the firmawre as a control transfer.

First the host will send the set_report as shown below:

bmRequestType	00100001
bRequest	SET_REPORT (0x09)
wValue	Report Type (0x03) and Report ID 0x00)
wIndex	Interface (0x00)
wLength	Report Length (0x0004)
Data	Report (4 bytes)

This request is specific to the HID class, this is why it is not managed by the *usb_standard_request.c* file but with the *usb_specific_request.c*. In this file the request is decoded following the value of the bmRequest and the bRequest using the *usb_user_read_request()* function. The report type (0x03) corresponds to a Set_Feature_Report. To handle this request the *usb_user_read_request()* will call the *usb_set_feature*() function. This function will acknowledge the setup request and than allow you to get the sent data (you can check the size using the wLength parameter).

```
void usb_set_feature(void)
{U16 wInterface;
  U16 wLength;
  U8 tab[];
  LSB(wInterface)=Usb_read_byte();
  MSB(wInterface)=Usb_read_byte();
  LSB(wlength)=Usb_read_byte();
  MSB(wLength)=Usb_read_byte();
  Usb_ack_receive_setup(); //Clear the setup flag
  while(!Is_usb_receive_out()); //Wait data stage
for(i=0;i<wLength;i++)
    tab[i] = Usb_read_byte() //Read data sent by the host using setFeaure
Usb_ack_receive_out();
Usb_send_control_in(); //send a ZLP zero length packet to ack the setup
while(!Is_usb_in_ready());
}
```

Note: The setFeature is used to send a request from the PC to the device to jump to the bootloader.

7.5 How to modify the polling interval of each endpoint

Since the HID Generic is using the interrupt transfer for both endpoints, each endpoint should have its own polling inteval (specified in the enpoint descriptor).

To modify this parameter to fit with the requirment of your application, you must modify the following values from the *usb_descriptors.c:*

```
// USB Endpoint 1 descriptor FS
#define ENDPOINT_NB_1       (EP_HID_IN | 0x80)
#define EP_ATTRIBUTES_1     0x03          // BULK = 0x02, INTERUPT = 0x03
#define EP_IN_LENGTH       8
#define EP_SIZE_1           EP_IN_LENGTH
#define EP_INTERVAL_1       20 //interrupt pooling from host
// USB Endpoint 1 descriptor FS
#define ENDPOINT_NB_2       (EP_HID_OUT)
```

```
#define EP_ATTRIBUTES_2     0x03          // BULK = 0x02, INTERUPT = 0x03
#define EP_OUT_LENGTH      8
#define EP_SIZE_2           EP_OUT_LENGTH
#define EP_INTERVAL_2       20 //interrupt pooling from host
```

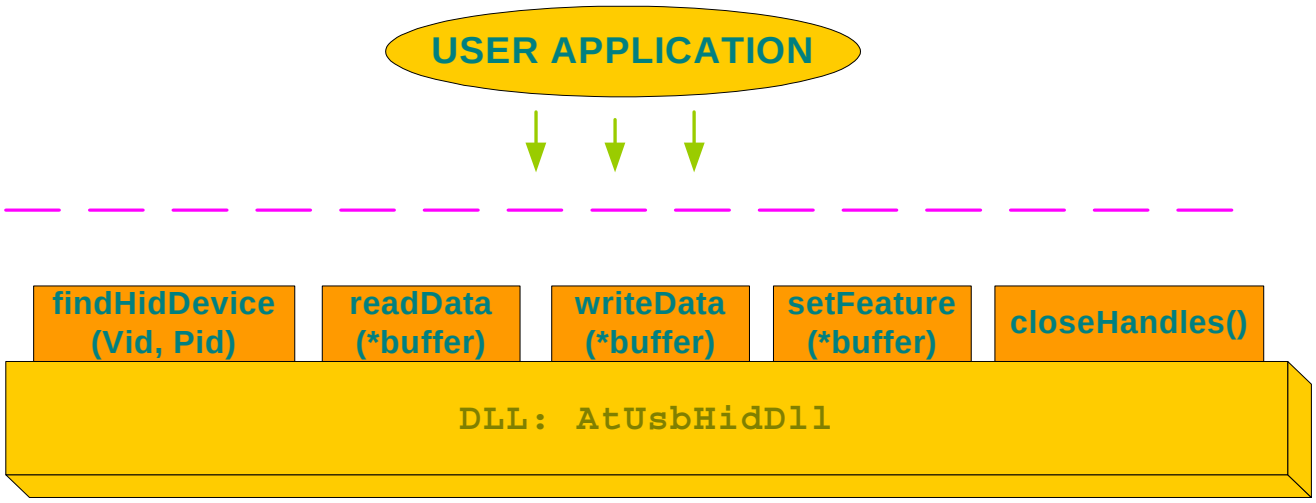
The allowed valued should be between 1 and 255 and the unity is ms.

8. PC Software

As shown in the Figure 6-4, the Generic HID application requires a software package. This package contains a DLL (Dynamic Link Library) and a user application.

The DLL allows the user to interface with the host USB drivers and avoids heavy development. This DLL contains five functions explained below:

Figure 8-1. HID DLL



8.0.1 findHidDevice

This function returns true and creates the handle if the right device is found (the VID & PID are correct). Otherwise, it returns false.

Parameters: VID: The vendor ID of the device

PID: The Product ID of the device.

8.0.2 readData

This function ensures the reading of data sent by the device and saves it on the buffer.

Parameters:*buffer: Pointer of the buffer on which data will be saved.

8.0.3 writeData

This function ensures the sending of data from the PC to the device. It also ensures the sending of the DFU command.

Parameters:*buffer: Pointer of data buffer.

8.0.4 setFeature

This function allows the user to send a control data to the device through the endpoint 0.

Parameters:*buffer: Pointer of data buffer

8.0.5 closeHandles

This function closes all the handles and the threads. It should be called when you close the application.

Parameters : none.

Please refer to the application note *USB PC Drivers Based on Generic HID Class* for further information regarding the USB HID DLL.

9. Related Documents

AVR USB Datasheet (doc 7593)

USB Software Library for AT90USBxxx Microcontrollers (doc 7675)

USB HID class specification.

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature