USB Generic Implementation

Each USB device requires a specific driver from a Host point of view. Operating Systems offer standard drivers corresponding to USB classes. For specific implementation or for non-supported USB classes, the device manufacturer has to develop its own USB driver. This requires strong knowledges on the targeted OS and on driver development.

The aims of this document are:
- describe how to use a specific USB implementation, without developing the corresponding USB Host driver.
- describe the way to use USB, without writing a USB code

References
- Doc4136.pdf - AT89C5131 data sheet (available on www.atmel.com)

Acronyms
- USB: Universal Serial Bus
- HID: Human Interface Device
Operating System Basics

Operating System Architecture

In this document, the Operating System used is Microsoft Windows.

Figure 1. Microsoft Windows® 2000 architecture

As we can see on Figure 1., the USB drivers are located in the Kernel (“Kernel Mode Drivers” box). The difficulty is to develop such a driver.
The USB HID Driver

Every Operating System (Microsoft Windows, Linux®, MAC® OS) supports the USB HID class. This USB class has been created to support every Human Interface Device, such as mice, keyboards, gaming pads...

The specificity of this class is to be very flexible and dynamically configurable following the enumerated device. This configuration is performed using special sets of HID and report descriptors.

Please refer to the HID class specification at http://www.usb.org for more details.

ATMEL Generic Use Of HID Driver

ATMEL provides:

- from the device side: a set of functions for enumerating as HID, reading and writing through USB, with an abstraction of the USB firmware stack.
- from the PC side: a set of functions for detecting a specific USB device, reading from it and writing to it, with an abstraction of the kernel stack.

This speeds up the development of the USB application.

Figure 2. ATMEL USB Architecture

NOTE: Atmel USB PC driver is described in another document (doc7645 on Atmel site).
Atmel Generic USB: firmware side

This section describes the variables and the functions available to use the usb_hid_generic_api. An example of use is implemented in user_application.c.

Global Variables

type reset_to_bootloader

This bit is set when the firmware has received a “Jump to Bootloader” command from the Host controller.

Read Functions

type Is_new_data_read()

This function returns TRUE if a new message has been received from the Host. Otherwise, this function returns FALSE.

U8 usb_read_continuous(U8 *buffer)

This function fills the buffer with the data received from the Host. The returned value is the number of bytes stored in buffer.

usage

    U8 data_buffer[LENGTH_OF_REPORT];
    U8 number_of_byte;
    if (Is_new_data_read())
    {
        number_of_byte = usb_read_continuous(data_buffer);
    }

Write Functions

type Is_write_data_ready()

This function returns TRUE if the firmware is allowed to write and send a message to the Host. Otherwise, this function returns FALSE.

U8 usb_write_continuous(U8 *buffer, U8 data_length)

This function writes and sends “data_length” bytes stored in “buffer” to the Host. The returned value is the number of bytes written.

Warning: because the Host waits for a "LENGTH_OF_REPORT” bytes for each transfer, this function will complete with 0xFF if “data_length” < "LENGTH_OF_REPORT”.

usage

    U8 data_to_send[LENGTH_OF_REPORT];
    U8 number_of_bytes_to_send, number_of_bytes_sent;
    if (Is_write_data_ready())
    {
        number_of_bytes_sent = usb_write_continuous(data_to_send,
        number_of_bytes_to_send);
    }
Customize

VID / PID / Serial Number

The application has to be customized with specific VID / PID / Serial Number using the conf_usb.h file.

LENGTH_OF_REPORT

By default, this value is equal to 0x08. This may be customized from 1 to the maximum size of the IN and OUT Endpoints.

Application Example

An application example is delivered: user_application.c.

This example can be customized / re-written for specific applications.

The main features of this example are:

• Turn ON/OFF Leds 0, 1, 2 and 3 following a OUT command from the Host
• Report the INT0 (P3_2) status (1=released / 0=pressed) to the Host with the byte 0 of the IN report.
• Start the bootloader when receiving a “Jump To Bootloader” command from the Host.

IMPORTANT NOTE:

The final application has the responsibility to split a message larger than LENGTH_OF_REPORT in messages, the size of which is equal or less than LENGTH_OF_REPORT.

OUT report example:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LED number

'0'=OFF / '1'=ON

IN report example:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0=released / 1=pressed