Universal Serial Bus in Industrial Measurement and Control Applications

Gerald Kupris
Motorola Semiconductor Products Sector

The Universal Serial Bus (USB) specification is defined as an interface for the connection of peripheral devices, for example printers, scanners, keyboards and pointing devices, to a PC or a similar host computer. A possible application for USB is in the field of measurement and control. As the need for general-purpose sensor and actuator devices in laboratories, households, offices and industrial applications increases, designers are faced with the problem of interfacing these devices with a computer. USB is an ideal solution for this problem.

1. Introduction to USB

For USB to be successful it is important that the USB devices are user-friendly through having a plug-and-play type functionality. The USB specification is an industry standard, which addresses this by giving a complete definition of the bus system from the electrical interface to the higher protocol layers, guaranteeing the interoperability of all devices. The basis specification of USB is supplemented by specifications for certain frequently used types of device classes, for example a human interface device (HID) class for keyboards, mouse pointers etc. [1]. Plug and play functionality was one of the initial drivers of the USB specification; ease of use by consumers and customers has always been one of its main objectives. The massive penetration of USB became a reality through the integration of the USB hardware into chip-sets on PC Motherboards. Today it is estimated that at least two USB connections are available in every PC.

USB has several key features that make it the first choice for designers now, and will help to keep it as a dominating protocol in the future:

- The same bus system is suitable for all attached devices.
- Using hubs, up to 127 end devices can be attached to the bus.
- Plug and Play: USB devices can be attached to or removed from the PC during operation.
- Small devices can be supplied with energy via the bus.

The Basic Principles

Specification	**Practical Transmission Rate**	**Typical Application**	**Characteristic**
"Low Speed" USB 1.1 (1.5 Mb/s)	Small 10-100 kb/s	Keyboard, Mouse, Joysticks …	Minimum Cost
"Full Speed" USB 1.1 (12 Mb/s)	Medium 0,5-10 Mb/s	Printer, Scanner, Audio-Interfaces	Bandwidth guaranteed
"High Speed" USB 2.0 (500 Mb/s)	High 25-500 Mb/s	Video, Mass storage	High bandwidth guaranteed

Table 1: USB specifications and transmission rates

Several versions of the USB specification exist. In table 1 the varying transmission rates are defined for the low, medium and high-speed USB protocols. However, these rates cover the speed of the entire bus system. If several end devices are attached to the bus, these devices must share the existing bandwidth. The latest specification, 2.0, was complied in the year 2000 by the USB Implementers Forum (USB-IF). It is compatible with version 1.1 and contains the already established “Low Speed” and “Full Speed” classifications. Additionally release 2.0 introduces a High Speed device type, however USB 2.0 compliant devices are only just becoming available today.

Apart from the transmission speed, the USB Specification also standardizes the electrical and mechanical characteristics of the device connection (figure 1). In figure 1 the signals on lines D+ and D- are differential signals with voltage levels of 0V and 3.3V. These two lines transmit the USB data. Additionally, an operating voltage of 5V is made available for low-end devices, which only draw a current of up to 100mA.

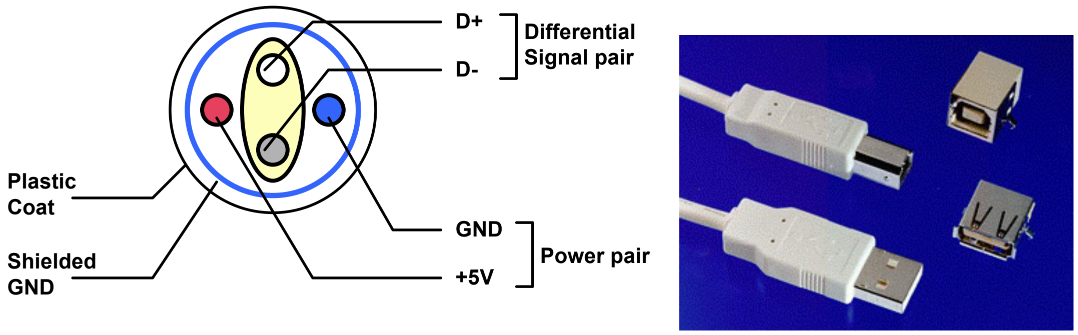

Figure 1: Cross section of a USB cable and different USB plugs

Most 8-bit microcontrollers with integrated USB functionality use the Low Speed implementation. Typical Low Speed applications are PC peripherals such as keyboards, mice and similar input devices. Motorola’s innovative MC68HC908JB8 is a good example of a microcontroller with integrated USB functionality [2]. This component contains the HC08-CPU clocked with 6MHz and the Low Speed USB interface as well as 8K Flash memory. The on-chip programming and debugging interface makes this MCU ideal for use in Low Speed applications and the development of USB applications in the laboratory.

2. Measurement and Control

For measuriement and control applications typically only small data ranges are needed, which is easily accomplished by using Low Speed USB devices. A Low speed USB device ensures a realistic data rate, which can be compared roughly with a RS232 connection with 9600 Baud. However, while RS232 represents a point-to-point connection, USB uses a bus structure. Several USB devices can be attached to a PC, using it as a bus master. If the connections at the PC host are not sufficient, hubs can be inserted and several hubs can be cascaded in up to 5 levels. Each USB device is addressed by the host via a unique address consisting of 127 addresses. The communication between host and USB device can be explained by a model of different software layers and protocols (figure 2). Communication always takes place between the appropriate layers of the host and the device firmware.

This diagram references USB 1.1, however a full detailed discussion of software communication exceeds the limits of this article. For more information about this, Kelm’s book, USB 1.1 [4] offers a good introduction. In this article condensed descriptions for some of the terms and procedures, important for the initial implementation of a USB design, are given.

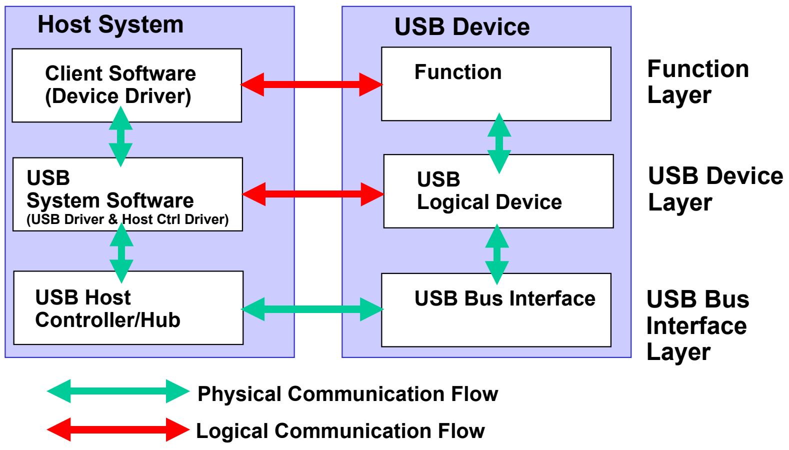

Figure 2: Communication between different software layers

The Packets

Packets form the basic modules of USB communication in the levels above electrical connection. Packets are atomic, so they cannot be interrupted or divided into sections. The packet types shown in Table 2 are relevant for Low Speed USB.

Name	Group	Function
SETUP	Token	Starts a control transfer
IN	Token	Starts a data transfer to the host
OUT	Token	Starts a data transfer to the device
DATA0	Data	Transfers 0 to 8 data bytes
DATA1	Data	Transfers 0 to 8 data bytes
ACK	Handshake	Information was accepted
NAK	Handshake	Busy – send again later
STALL	Handshake	Information was incorrect

Table 2: Low Speed USB Packet Types

In addition to the packets shown, the bus traffic consists of further quasi-static bus conditions (reset, suspend, resume) and the "Keep-alive-EOP". A USB transaction is a series of packets that exchange information between a host and a device. A transaction is always initiated through the host sending a token packet (SETUP, IN, or OUT), as the actual USB devices are incapable of initiating transactions. The token packet contains the address and the desired endpoint of the device. SETUP and OUT packets are supplemented by a DATA packet from the host, which contains up to eight bytes of data. The packets DATA0 and DATA1 are always sent in an alternating sequence. This procedure is called data toggle and serves as error protection. Following the data packet, the device answers with a handshake packet. If the device is able to receive the data, it sends an ACK packet. If the device is not immediately ready, it sends a NAK packet signalling the host that the packet should be sent again at a later time. In the event of an error, the device sends a STALL packet. For IN transactions, the device sends the data packet and the host closes the transaction with a handshake packet. If the device does not hold any data ready for sending, it sends a busy handshake NAK instead of the data packets.

While the actual data communication is based on simple IN and OUT transactions, the management of the device uses more complex control transfers. These control transfers are secured using a double handshake, so control transfers consist of two or three transaction stages.

The set-up stage starts with a SETUP transaction. The data stage is necessary if data has to be sent from the device to the host, and consists of several IN transactions. The status stage serves as back confirmation that the information was processed correctly.

VID and PID: The Identity of the USB Device

The Vendor ID (VID) designates the manufacturer of the equipment. The assignment of this 16-Bit ID takes place via the USB organization, so that a clear allocation from devices to manufacturers can always take place. The use of unauthorized fantasy IDs would, in the long run, cause dangerous conflicts of different devices on the bus.

The Product ID (PID) is specified at the manufacturer's own discretion, and is also a 16-Bit number. VID and PID together usually represent the unique device identifyer and are used to find the suitable driver component.

Members of the USB-IF (Implementors forum) automatically receive a Vendor ID, however the yearly membership costs about $2,500 USD. This could quickly become a cost consideration, particularly for smaller companies, where USB is not the primary focus. However, to bring USB specified devices to market USB-IF membership is required, as it is necessary to have a legal VID. In addition, restrictions exist with use of the official USB logo. This logo is to signal the official "safety" of a product to end-customers, which is extremely useful in marketing mass produced products to consumers, especially in the PC market.

The USB08 Starterkit

To assist engineers in building their own USB applications, Motorola and Elektronikladen have developed a USB starterkit [5] - Figure 3. The kit uses Motorola's MC68HC908JB8, a low cost 8-bit microcontroller. [5].

This starter kit and reference design [6] (available at Elektronikladen [8, 9]) offer a complete reference platform to assist the USB designer's own developments within the USB range. The user will receive, among other things, complete C-Source codes for the microcontroller firmware, demo software for the PC and device help to accelerate development.

Motorola's USB08 Reference Design shows a detailed example of how the integrated USB module on the MC68HC908JB8 can be used. The microcontroller can be used for measuring and control applications and data exchanges with a PC via the Universal Serial Bus. The integrated USB module on the MC68HC908JB8 works at a data rate of 1.5 MBit/s, thus it is defined as Low Speed. Typically only small data rates are needed, making Low Speed USB devices an ideal solution for measurement and control applications. The Reference Design contains a USB08 evaluation board, enabling this plug and play demo application to be evaluated immediately. Beyond that, all source codes are provided in the form of C modules, giving the designer all the building blocks needed for their own design.

The administration of the USB device takes the largest portion of the source code of the reference implementation. This is done via a Standard Device Serve Request and the transfers take place via control endpoint EP0, and are multi-level, complex communication procedures. Since the implementation of these complex functions can be transferred as a block from the Reference Design to any other application, the practical work for the administration of an USB device become easy. The actual data communication functions, which take place using the interrupt endpoints EP1 and EP2, can be simply adapted to the concrete user application.

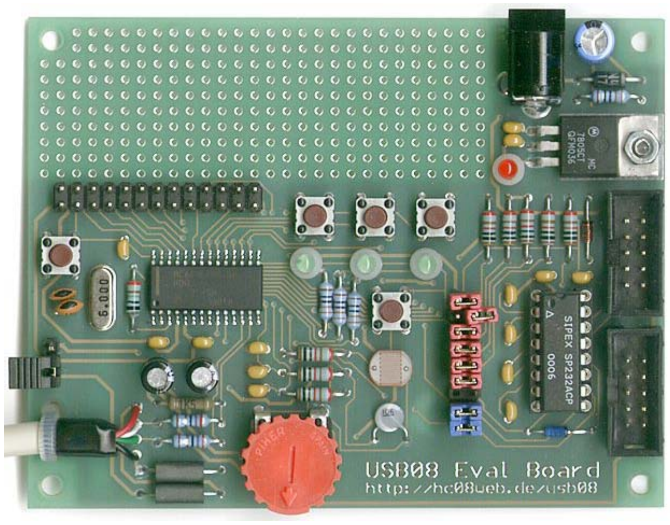

Figure 3: Motorola's USB08 Starterkit

The administration of the USB device takes the largest portion of the source code of the reference implementation. This is done via a Standard Device Serve Request and the transfers take place via control endpoint EP0, and are multi-level, complex communication procedures. Since the implementation of these complex functions can be transferred as a block from the Reference Design to any other application, the practical work for the administration of an USB device become easy. The actual data communication functions, which take place using the interrupt endpoints EP1 and EP2, can be simply adapted to the concrete user application.

The Endpoint Configuration

The integrated USB module on the MCHC908JB8 supports three endpoints EP0, EP1 and EP2. Beside the mandatory, bi-directional control endpoint EP0 two unidirectional interrupt endpoints EP1 and EP2 are available. The data direction is always indicated from the view of the host. Thus an IN endpoint serves for data transfer from the device to the host, and an OUT indicates data transfer from the host to the device. EP1 is always configured as an IN endpoint, since the microcontroller sends data this way. The MC68HC908JB8 endpoint EP2 can be configured as an IN or as an OUT endpoint. To perform data transfers to the microcontroller, the USB08Reference Design uses this endpoint as an OUT endpoint. Some adjustments to the control registers of the USB module are necessary for the configuration of the endpoints. At the end of the USB reset interrupt, the control endpoint EP0 will be enabled for receiving, thus the configuration instructions which follow after the reset (device requests) will be received by the USB receiver. The USB device is now in default status. This means, the device is attached to the bus, is supplied with current, it has received a USB reset, and has reacted to instructions with the default address. After this fundamental initialization further set-up steps follow, as described in the next section.

The Enumeration

After the basic initialization, the USB module is now able to react with an interrupt on packets, addressed to the control endpoint EP0. Enumeration is the process in which the configuration is integrated into the USB system. The host is now able to assign a clear USB address to the device, located in the range from 1 to 127. In another step in the enumeration, the host will request the configuration of the device. For this purpose descriptors are made available by the device, which contain information about the status and about one or more possible configurations. The host loads these descriptors, selects thereafter a suitable driver and forces the device to take a certain configuration. In consequence, the device will be ready for use and will be able to transfer data via the interrupt endpoints EP1 and EP2.

Data Communication Via Endpoints EP1 and EP2

The transmission of user data from or to the USB device takes place via the endpoints EP1 and EP2. EP1 is an IN type endpoint and sends information to the host. EP2 possesses the direction OUT and is used by the device to receive data from the host.

The host polls all interrupt endpoints cyclically, taking into account a guaranteed maximum latency time. That polling interval can be specified in the endpoint descriptor. For Low Speed USB devices with interrupt endpoints the shortest specified polling interval amounts to 10ms. This means, after the polling interval the host asks whether further data has to be fetched from endpoint EP1. In practice, the host uses only intervals of 2^n ms, the demanded 10ms is then rounded to 8ms.

The Windows™ Device Driver

Two identifiers are used to mark a USB device and make it possible for the host to assign a suitable driver: The Vendor ID and the Product ID. Both IDs are registered in the device descriptor of the USB equipment. Both the VID and the PID represent the search criteria for the suitable Windows™ device driver.

With the help of the Microsoft Software Development Kit, it is possible for USB device manufacturers to develop the necessary Kernel Mode Drivers for themselves. However, this kind of programming task requires a deep understanding of the structure and working principles of the Windows™ driver modules. For engineers who do not program regularly, it is strongly recommended they not try this task. A possible workaround could be the operation of the USB device using a Windows™ standard driver for Human Interface Devices (HID). The device class HID covers PC peripheral input such as a keyboard attached to the PC. It is possible to mask the measuring data as a HID input packet and send them through the HID driver. The advantage of this approach includes not having to develop a Kernel Mode Driver. Instead you can program your own PC application on top of the existing free operating system driver.

However, there are two major disadvantages of the HID approach. First, the complexity of the USB handling increases, in particular on the firmware side. There are additional procedures, protocols and descriptors to implement, which are defined in some specifications (not a part of the original USB specification, but defined in a HID specification, which can be loaded from the USB website as well).

Second, several implementation variants exist within each version of Windows™. With each new service pack or release the user risks changes in the USB drivers may change. In this case, the testing expenses may be quite high. To alleviate these problems it is recommended that a universal 3rd-party USB driver be used, which is professionally developed and maintained, as soon as new operating systems occur. The driver specialist Thesycon [5] offers now a "Light version " for the driver library USBIO. By using the generic USB device driver USBIO, it is possible to get any USB device up and running without spending the time and the effort of developing a device driver.

USBIO is a generic Universal Serial Bus (USB) device driver for Windows™ 98, Windows™ Millennium (ME), and Windows™ 2000. It is able to control any type of USB device and provides a convenient programming interface that can be used by Win32 applications. This might be especially useful during development or test of a new device. But in many cases it is also suitable to include the USBIO device driver in the final product. So there is no need to develop and test a custom device driver for the USB-based product at all.

Literature:
[1] Universal Serial bus Specification, revision 1,1 23 September 1998
[2] MC68HC908JB8 Technical Data, Rev. 1.0; Motorola 2000
[3] http://www.usb.org
[4] Kelm, H. J.: USB1.1; Franzis 2000
[5] http://www.thesycon.com
[6] http://hc08web.de/usb08
[7] http://www.motorola.com/semiconductors
[8] http://www.elektronikladen.de
[9] http://www.startergate.com

Contact Address:
Motorola GmbH
Gerald Kupris
Schatzbogen 7
81829 Munich / Germany
Gerald.Kupris@motorola.com
http://www.motorola.com/semiconductors