Understanding USB

Devised to simplify the job of connecting peripherals to a PC, the increasingly popular universal serial bus communicates data at up to 12Mbit/s over two of its four wires. Simple connection, yes, but the protocol needed to make the bus efficient and transparent to the PC user is complex, as Tony Wong explains.

Universal serial bus, or USB, is recommended for the new generation of IBM-compatible PCs in the 'PC 98 System Design Guide'.[1] It is also supported by Windows 98.

This bus provides an easier way of connecting a PC to a variety of peripherals via a serial bus. The universal serial bus is a four-wire cable of which there are two wires for power and ground – namely V_{bus} and Gnd – and two for data transfer, D+ and D–.

Up to 126 devices can be simultaneously connected to a PC via USB without the fear of running out of PC i/o addresses or having conflicts on IRQs and DMA channels used. The bus can also reduce cost and PCB space by removing the need for traditional attachment ports such as keyboard connector and serial ports.

Other benefits of USB are its low cost and that it supports data transfer at up to 12Mbit/s in 'full-speed mode'This is described in USB specification 1.0/1.1. At this speed, it is possible transfer data such as voice and compressed video signals in real time.

At the end of 1999, the USB 2.0 specification will be officially released and will move the maximum transfer rate to between 120 and 240Mbit/s. However, behind of these features, there is a need for sophisticated USB embedded controllers to handle the unique protocol and algorithm for the data communication activities on the bus.

In this article, I present a summary of the USB protocol format based on USB 1.1.

As an example of how the bus is implemented, I describe the Infineon Technologies C541U embedded USB microcontroller. Infineon Technologies was formerly Siemens Microelectronics by the way.

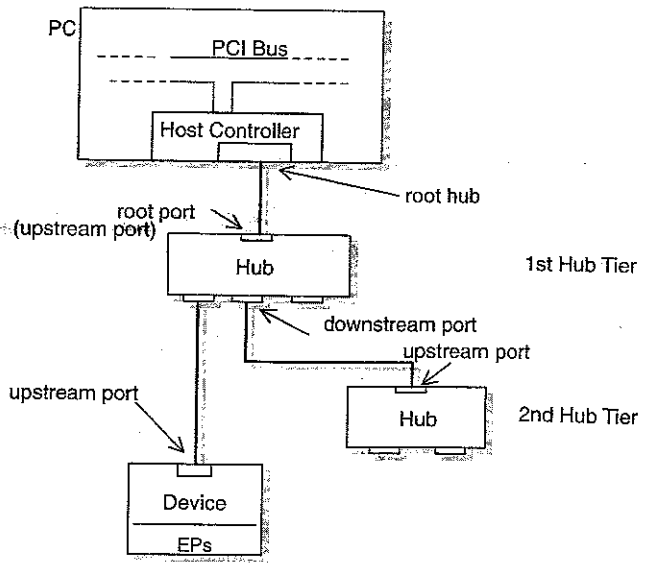


Fig. 1. With universal serial bus, hubs can be used to expand the system, allowing connection of up to 126 devices to one PC.

What is an 'end point'?

An end-point, or device end-point, as used in USB terminology, is not the easiest of concepts to understand. In the specifications, an 'end-point', or EP, is described as, "A uniquely identifiable portion of a USB device that is the source or sink of information in a communication flow between the host and device." Physically, an end-point can be considered as a memory area for data flow.

Take, for example, a CD-ROM drive with USB interface. The drive can be accessed by the file manager tool to read a data file from the CD, but it can also be used as to play audio CDs. As a result, you need to use two end-points to handle these two functions. One end-point is configured as 'bulk transfer', for transferring data files, while the other end-point is configured as 'ISO transfer' for real-time audio data transfer.

USB terminology

Descriptor
A Descriptor carries the information used for identify the device, for example, number of end-points, or EPs, in the device and the end-point type for each. This information is usually accessed by the host from the device.

Function
This is a USB device that provides the host with a capability, such as an ISDN connection.

IN
A packet-identification, or PID, type used to perform the 'in' transaction. Data packets flow from device to host.

HID
Human Interface Devices class. Devices are used by humans to control the operation of computer systems. Examples are keyboards, pointing devices and bar-code readers.

Host
A computer system with the installed host controller driver is a host. This includes the hardware USB root port, which may be a PC system running under the Windows 98 operating system for example.

Host controller
The host's USB interface.

Hub
A full-speed USB device that allows additional connections to the USB bus is know as a hub.

OUT
A packet-identification, or PID, type used to perform the 'out' transaction. Data packets flow from host to device.

PID
Packet identification – a field in a USB packet that indicates the types of packet. Examples are SOF, IN and ACK.

SETUP
A packet identification, or PID, command type used to perform the 'set-up' transaction. Data packets flow from host to device. One use of SETUP is to allocate an address to a device.

SIE
Serial Interface Engine is a hardware circuit and a part of USB module in silicon. It performs USB data processing tasks.

SOF
Start-of-Frame. SOF is a packet-identification type and also the first transaction of a frame. It allows end points to synchronise their internal clock to the host.

USB peripheral/USB device
A device that performs an USB function such as a hub, a USB keyboard, or USB speakers.

USB peripheral silicon
A USB embedded silicon chip.

Table 1. Some dedicated USB chips and microcontrollers with USB support.

	Samsung	Zilog	Cypress	Infineon Technologies	NetChip Technology	Motorola
Part number	KS86C6004	Z8E520	CY7C64113	C541U	NET2888	MC68HC08KH12
Product type with hub	8-bit controller with USB	8-bit controller with USB	8-bit controller with USB	8-bit controller with USB	USB peripheral controller	USB keybd 8-bit controller
USB speed	Low	Low	Full	Full/low	Full/low	Full/low
ROM (bytes)	4K	6K OTP	8K EPROM/OTP	8K OTP	—	12K OTP/Flash
RAM (bytes)	208	160	256	256	—	384
Clock	6MHz	12MHz	6MHz	12MHz	—	6MHz
USB buffer	8 bytes for transmit	8 bytes	Up to four 8-byte data EPs	Configurable USB buffer	64 bytes each of transmit	8 bytes each of transmit and
	8 bytes for receive		Up to two 32-byte data EPs	8 bytes for control EP	and receive for bulk/ISO	receive for control EP
				Up to 64 bytes for data EPs	transfers	8 bytes of transmit for data
EP				Total buffer size 256 bytes	8 bytes for bulk EP	
					8 bytes for interrupt EP	
End points	1 control EP	1 control EP	1 control EP	Full-speed mode:	1 control EP	1 control EP for hub
	2 data EPs	2 data EPs	4 data EPs	1 control EP	4 data EPs	1 data EP for hub
				4 data EPs		1 control EP for device
				Low-speed mode:		2 data EP for device
				1 control EP		
				2 data EPs		
Op. voltage	4.5V~5.5V	4.0V~6V	4.0V~5.5V	5V	3.3V	5V
Int. transceiver	Yes	Yes	Yes	Yes	Yes	Yes

Notes: With the C541U, each data EP can be configured to support any one of four USB transfer types. The NET2888 acts as a bridge between processor Independent local bus and USB bus. The Motorola chip has 1 full-speed upstream port, 4 full/low-speed downstream ports and it supports interrupt transfer on the data EP.

This device provides a simple solution to the USB peripheral implementation. Its hardware can handle the USB protocol transmissions automatically.[2]

USB system architecture

There are three basic hardware elements in the USB system architecture. They are host, hubs and devices, **Fig. 1**.

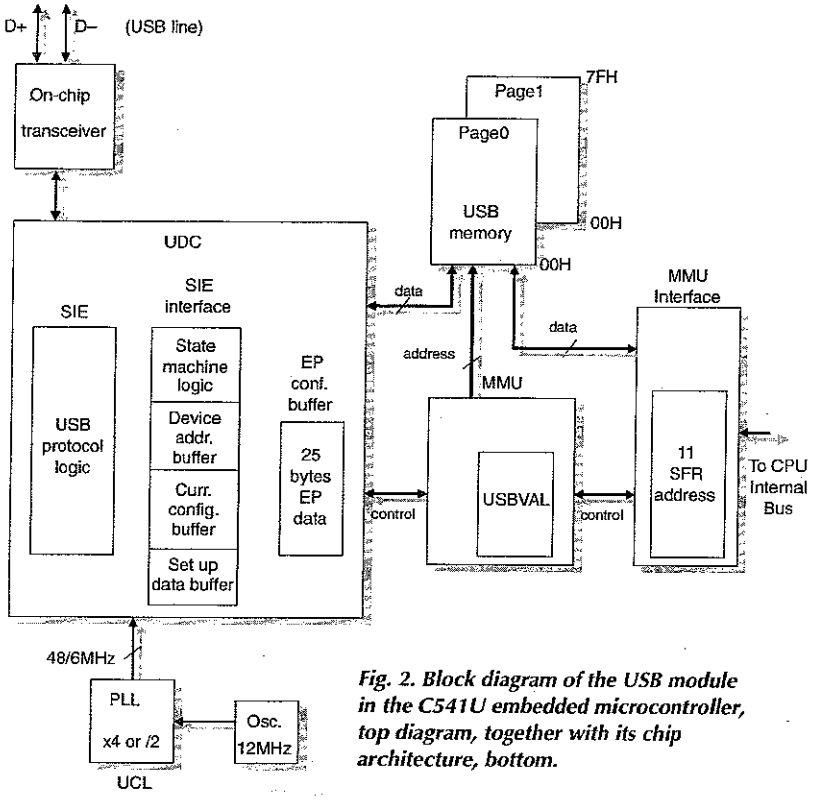
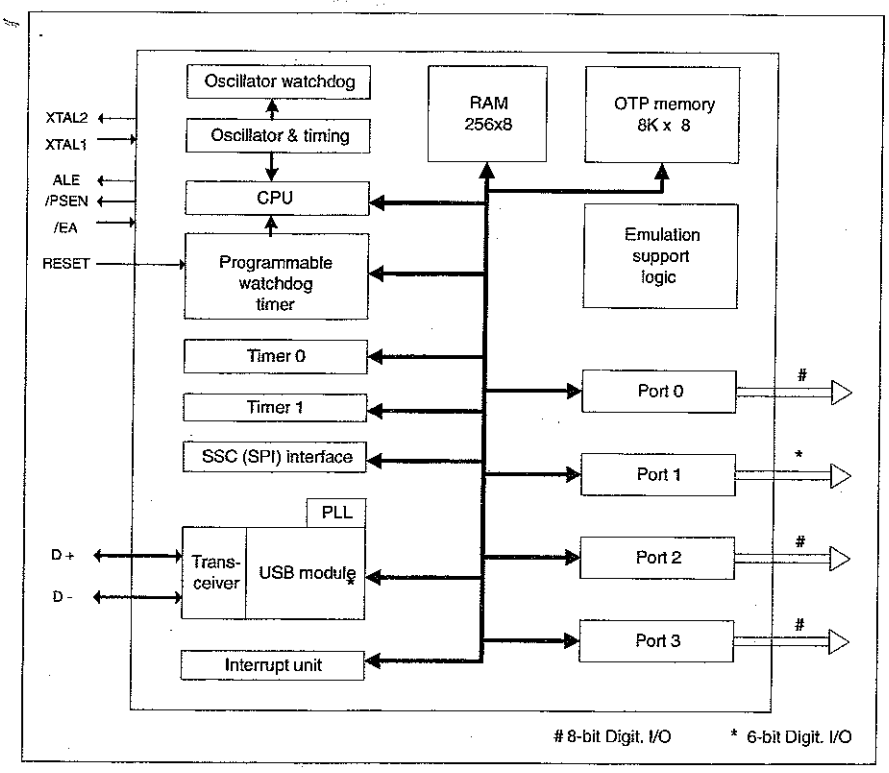


Fig. 2. Block diagram of the USB module in the C541U embedded microcontroller, top diagram, together with its chip architecture, bottom.



The connection uses the 'tiered-star' topology and can be connected up to five levels – i.e. have five hub tiers. Normally, the host controller and root hub are implemented via a chip set on the PC motherboard.

The host controller controls transactions over the USB system. There are two types of host controller. They are the 'open-host controller interface', or OHCI, and the 'universal-host controller interface, which is shortened to UHCI.

From an application point of view, the OHCI can support multiple transactions for a particular device end point, or EP, within a 1ms 'frame'. There's more on this later. On the other hand, the universal host controller supports one transaction for a particular end point in each frame. Software for USB devices should be able to handle transactions with either of these controllers.

A root hub acts as a port for attaching the USB device, Fig. 1. A USB hub allows multiple connections to the USB system and detects when devices are attached to or detached from the system. It also forwards the bus traffic between its upstream port and downstream ports.

Each USB device is allocated end-point numbers. End point number EP0 is reserved for the device's configuration by the host. It provides a point of communication to the host by means of EP descriptors.

End-point descriptors communicate device attributes and characteristics to the host. According to this information, the host configures the device and locates the USB client software driver.

Other device end points can be considered as a function of the device and can be separately configured for one of the different transfer types to communicate with the host.

For example, a keyboard application, which comes under the USB standard's 'human interface device, or HID, class uses EP0 for the device configuration and may use EP1 as an interrupt transfer to send the key-scanned data to the host. More details on the EP descriptors are discussed in references 3 and 4.

USB supports four types of data transfer:

- Control transfer – transfer request commands from host to device.
- Interrupt transfer – data transfer from an interrupt driven device to host.
- Bulk transfer – transfer for a large amount of data.
- Isochronous transfer – for applications requiring constant data transfer rate.

Implementing USB

Commonly, a USB embedded microcontroller is used to implement the USB functions. There are also other types of USB interface chip to suit different applications.

Table 1 shows some of the examples of USB chip. Only the USB related features are highlighted here; for details of other on-chip features and updated technical specification, refer to the corresponding company's product home page on the web site.

In the Table, the first two columns show two low-speed USB controllers with different clock frequencies and buffer sizes. The third column shows a full-speed chip, and the next one is a full/low-speed chip. The fifth column shows a USB interface chip without an integral controller and the last one is a controller with hub function.

Generally, a USB chip consists of a USB module in which the serial interface engine, or SIE, plays an important role in the USB activities. It performs all the front-end data processing functions such as NRZI and NRZ conversion, token packet decoding, bit stripping and bit stuffing, and cyclic

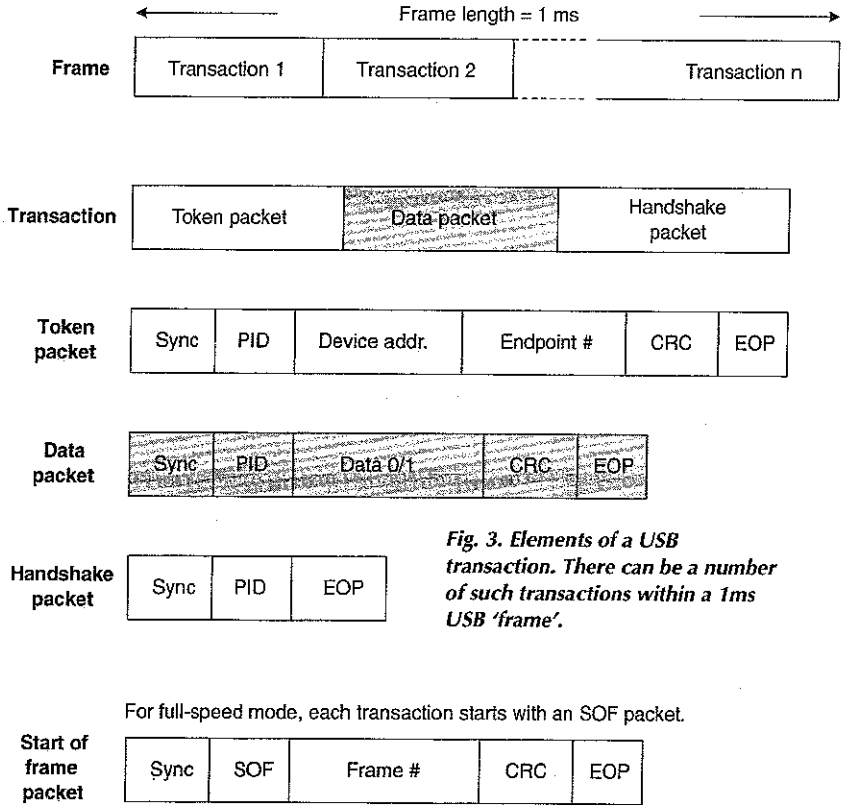


Fig. 3. Elements of a USB transaction. There can be a number of such transactions within a 1ms USB 'frame'.

Table 2: Packet transmission between host and device in full-speed mode.

File ENURATE.USB, recorded using CATC Inspector USB Analyzer S/W Version 2.10, packets 0-377 .

Packet#	
0	RESET(109.46 milliseconds)
109	Sync(00000001) SOF(0xA5) Frame #(0x079) CRC5(0x15) Idle(11964)
110	Sync(00000001) SOF(0xA5) Frame #(0x0E3) CRC5(0x0C) Idle(5)
111	Sync(00000001) SETUP(0xB4) ADDR(0x00) ENDP(0x0) CRC5(0x08) Idle(3)
112	Sync(00000001) DATA0(0xC3) DATA(80 06 00 01 00 00 40 00) CRC16(0xBB29) Idle(5)
113	Sync(00000001) ACK(0x4B) Idle(11801)
114	Sync(00000001) SOF(0xA5) Frame #(0x0E4) CRC5(0x12) Idle(5)
115	Sync(00000001) IN(0x96) ADDR(0x00) ENDP(0x0) CRC5(0x08) Idle(4)
116	Sync(00000001) DATA1(0xD2) DATA(12 01 00 01 00 00 00 08) CRC16(0xC8E7) Idle(6)
117	Sync(00000001) ACK(0x4B) Idle(11799)
118	Sync(00000001) SOF(0xA5) Frame #(0x0E5) CRC5(0x0D) Idle(11964)
119	Sync(00000001) SOF(0xA5) Frame #(0x0E6) CRC5(0x0F) Idle(5)
120	Sync(00000001) OUT(0x87) ADDR(0x00) ENDP(0x0) CRC5(0x08) Idle(3)
121	Sync(00000001) DATA1(0xD2) DATA() CRC16(0x0000) Idle(4)
122	Sync(00000001) ACK(0x4B) Idle(11866)
123	Sync(00000001) SOF(0xA5) Frame #(0x0E7) CRC5(0x10) Idle(1697)
124	RESET(Start of Reset)

redundancy generation and checking.

Figure 2 shows the block diagram of the USB module in the C541U controller chip from Infineon, and also the functional block diagram of the chip. More details of the operation of the USB module can be found in reference 5.

USB communication structure

Communication over the universal serial bus is performed with a series of frames. Within a frame, which is 1ms long, there can be a number of transactions.

The number of transactions depends on the number of attached USB devices and how often the host needs to communicate with these devices. A transaction can be viewed as a transfer of data. It consists of three phases.

Figure 3 shows the elements forming a packet phase. A token-packet phase comprises commands sent from host to a device and has four possible packet identifiers, known as PIDs. They are SOF, IN, OUT and SETUP.

Data is transferred during the data-packet phase. Two PID types are available for this, namely DATA0 and DATA1. This allows a data toggling mechanism, which is used to synchronise the transmitter and receiver of a transfer.

Acknowledgement of the data packet transfer is carried out during the handshake packet phase. It carries one of these PID codes, ACK, NACK or STALL, representing the current data receiving status.

Under the USB communication protocol, two kinds of control transfer can be performed. **Figure 4** shows the sequence of the communication for a three-stage control transfer involving a 'get descriptor' command transaction and a 'setup' token. Three-stage control transfer consists of a setup stage, a data stage and a status stage. It is mainly performed by the host to get information from the device.

Figure 5 shows a two-stage control transfer. Such transfers are used by the host to assign data to the device. For example, the host sets an address number to the device using the 'set descriptor' command as shown. Note that there is no data stage for the device to send back data information. The device only performs the 'status' stage to acknowledge the host by sending a zero-length data packet.

In addition to the two control-transfer formats, USB provides another data transfer format that is used to perform interrupt, ISO and bulk transfer types.

Figure 6 shows the data transfer procedure with two examples, namely 'interrupt' and 'ISO' transfers. For the interrupt transfer, the host keeps polling the bus by sending out 'in' tokens to the particular device.

The time interval for polling end-point for data transfer is user defined between 1 and 255ms. If the device has data to be sent, it will transmit the data packets to the host following the 'in' token. If not, it will send out an NAK response, which represents 'negative acknowledge'.

For an ISO transfer, which involves real-time data transmission, the 'in' token should be sent to the device every 1ms.

In order to illustrate the protocol involved in a real application more clearly, we used a monitor system called a CATC USB Inspector. It was set up to capture data transfers between a UHCI host and C541U device controller.

Figure 7 shows the test set-ups involved. The host was running Windows 98 and the device chip contained source code with USB keyboard function.

Test set-up 'A' captures the USB traffic between host and device directly. A full-speed hub and low-speed device are included in the test set-up 'B'. This second set-up can monitor the combined transmissions of full and low-speed packets on the bus at the same time.

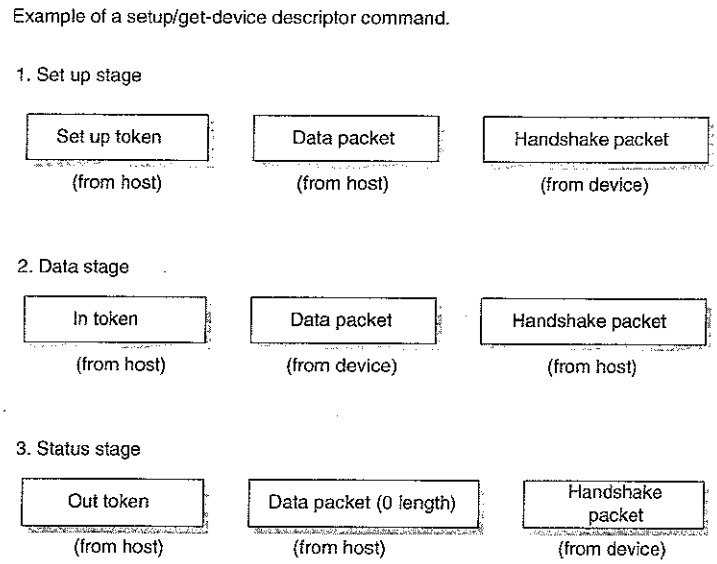


Fig. 4. In the USB protocol, two types of control transfer are possible. This diagram represents a three-stage control transfer, mainly used by the host to get information from a connected device.

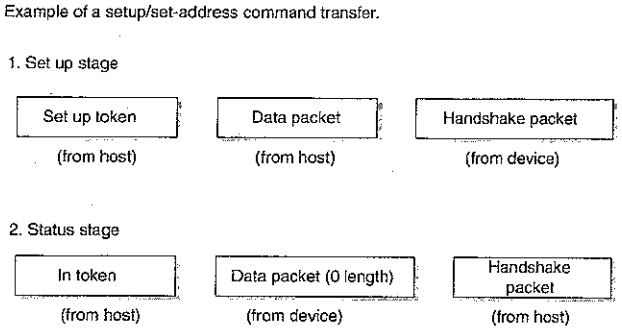


Fig. 5. Two-stage control transfers are mainly used by the host to assign data to the device. In this case, the device is being assigned an address.

Table 3: Packet transmissions in low-speed mode.

File LS-1.USB, recorded using CATC Inspector USB Analyzer S/W Version 2.2, packets 0-394.

Packet#	
18	EOP( ) Idle(1497)
19	EOP( ) Idle(114)
20	Sync(__000001) SETUP(0xB4) ADDR(0x00) ENDP(0x0) CRC5(0x08) Idle(4)
21	Sync(00000001) DATA0(0xC3) DATA(80 06 00 01 00 00 40 00) CRC16(0xBB29) Idle(6)
22	Sync(00000001) ACK(0x4B) Idle(13)
23	Sync(00000001) IN(0x96) ADDR(0x00) ENDP(0x0) CRC5(0x08) Idle(5)
24	Sync(00000001) NAK(0x5A) Idle(11)
25	Sync(00000001) IN(0x96) ADDR(0x00) ENDP(0x0) CRC5(0x08) Idle(5)
26	Sync(00000001) DATA1(0xD2) DATA(12 01 00 01 00 00 00 08) CRC16(0xC8E7) Idle(6)
27	Sync(00000001) ACK(0x4B) Idle(4)
28	Sync(00000001) OUT(0x87) ADDR(0x00) ENDP(0x0) CRC5(0x08) Idle(4)
29	Sync(00000001) DATA1(0xD2) DATA() CRC16(0x0000) Idle(6)
30	Sync(00000001) ACK(0x4B) Idle(887)
31	EOP( ) Idle(238)
32	RESET(Start of Reset)

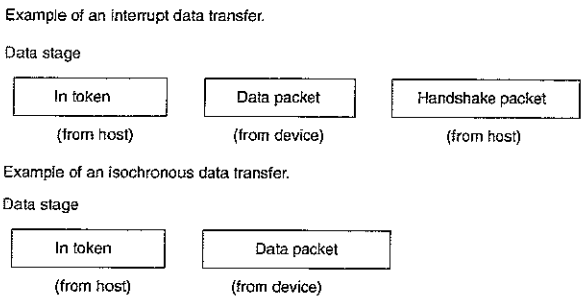


Fig. 6. Two examples of USB data transfer structures. The main difference is that interrupt transfer is polled and the isochronous transfer is real time.

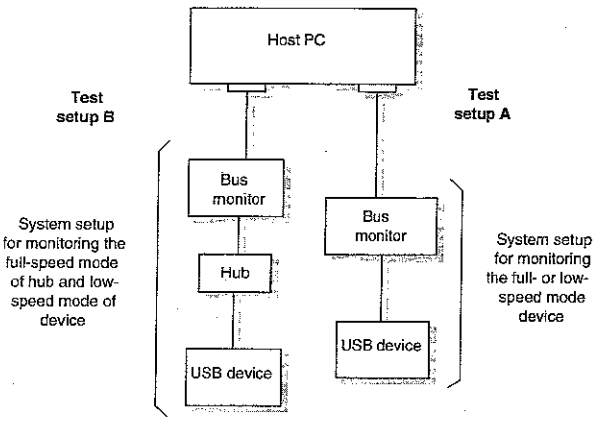


Fig. 7. Set-ups used to analyse USB communications. In this case the host is a UHCI and the device a C541U.

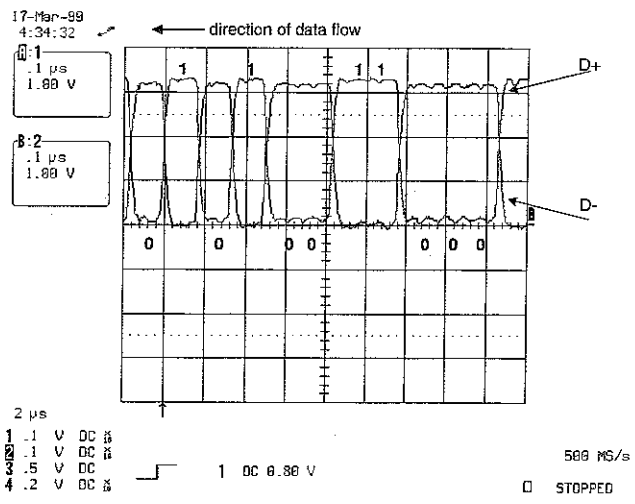


Fig. 8. USB signal, encoded in inverted non-return to zero form during sending of a negative-acknowledge NAK response to a host at full speed.

Snap-Shot of the USB Protocol

Table 2 shows part of the enumeration process of the device in full-speed mode. Enumeration is a procedure that allows a device to be recognised by the host and for setting up a communication pipe between them.

Packets 110 to 122 show a three-stage control transfer involving a 'get-descriptor' command. Packets 110 to 113 form the set-up stage, packets 114 to 117 form a data stage and packets 119 to 122 form a status stage.

Note that packets 110 to 113 are within one frame time, i.e. 1ms. Since there is only one device connected to the host,

Fig. 7 test set-up 'A', the transactions can not use up the whole time frame, resulting in an idle time of 11801 bit times.

For full-speed mode, the data transfer rate is 12Mbit/s. By counting the number of bits in each packet as shown in **Table 2**, it can be worked out that there's around 12K bits within a 1ms time frame. The table below shows the number of bits used for different tokens in the packet.

Field name	No. of bits
Sync	8
ADDR	7
SOF	8
ENDP	4
Frame #	8
DATA0	8
CRC5 / CRC16	5/16
DATA contents	64
SETUP	8
ACK	8

Low-speed mode

The captured packets transferred on the bus in low-speed mode are shown in **Table 3**. You can see that there is no start-of-frame SOF token. The time frame in this case is defined as being between 'end-of-packet' EOP tokens. Packets #20 to #31 are located within one frame.

Packets #23 and #24 show a negative-acknowledge NAK response issued by device to indicate that it is not available to respond to the 'in' packet from the host at that time. The descriptions on the packets are as follows:

- Three-stage control transfer with 8-byte data length:
  Packets #20 ~ #22, setup stage
  get-descriptor command
  Packets #25 ~ #27, data stage.
  Packets #28 ~ #30, status stage.

Full-speed and low-speed signals on the bus

This section looks at how full-speed and low-speed modes can operate simultaneously. Assume test set-up 'B' in Fig. 7. The USB bus between host and hub is in full-speed mode but

Table 4. The Low-speed packet transmission on Full-speed transactions.

File PRE-1.USB, recorded using CATC Inspector USB Analyzer S/W Version 2.2, packets 0-840.

Packet#	Idle(10654)
777	Sync(00000001) IN(0x96) ADDR(0x02) ENDP(0x0) CRC5(0x15) Idle(5)
778	Sync(__000001) DATA1(0xD2) DATA(03 03 00 00) CRC16(0xF0F9) Idle(7)
779	Sync(00000001) ACK(0x4B) Idle(42)
780	Sync(00000001) OUT(0x87) ADDR(0x02) ENDP(0x0) CRC5(0x15) Idle(3)
781	Sync(_0000001) DATA1(0xD2) DATA() CRC16(0x0000) Idle(5)
782	Sync(__000001) ACK(0x4B) Idle(765)
783	Sync(00000001) SOF(0xA5) Frame #(0x270) CRC5(0x0E) Idle(5078)
784	Sync(00000001) PRE(0x3C) Sync(00000001) SETUP(0xB4) ADDR(0x00) ENDP(0x0) CRC5(0x08) Idle(10)
785	Sync(00000001) PRE(0x3C) Sync(00000001) DATA0(0xC3) DATA(80 06 00 01 00 00 40 00) CRC16(0xBB29) Idle(48)
786	Sync(00000001) ACK(0x4B) Idle(87)
787	Sync(00000001) PRE(0x3C) Sync(00000001) IN(0x96) ADDR(0x00) ENDP(0x0) CRC5(0x08) Idle(40)
788	Sync(00000001) NAK(0x5A) Idle(73)
789	Sync(00000001) PRE(0x3C) Sync(00000001) IN(0x96) ADDR(0x00) ENDP(0x0) CRC5(0x08) Idle(38)
790	Sync(00000001) DATA1(0xD2) DATA(12 01 00 01 00 00 00 08) CRC16(0xC8E7) Idle(26)
791	Sync(00000001) PRE(0x3C) Sync(00000001) ACK(0x4B) Idle(10)
792	Sync(00000001) PRE(0x3C) Sync(00000001) OUT(0x87) ADDR(0x00) ENDP(0x0) CRC5(0x08) Idle(10)
793	Sync(00000001) PRE(0x3C) Sync(00000001) DATA1(0xD2) DATA() CRC16(0x0000) Idle(48)
794	Sync(00000001) ACK(0x4B) Idle(2865)
795	Sync(00000001) SOF(0xA5) Frame #(0x271) CRC5(0x11) Idle(1619)

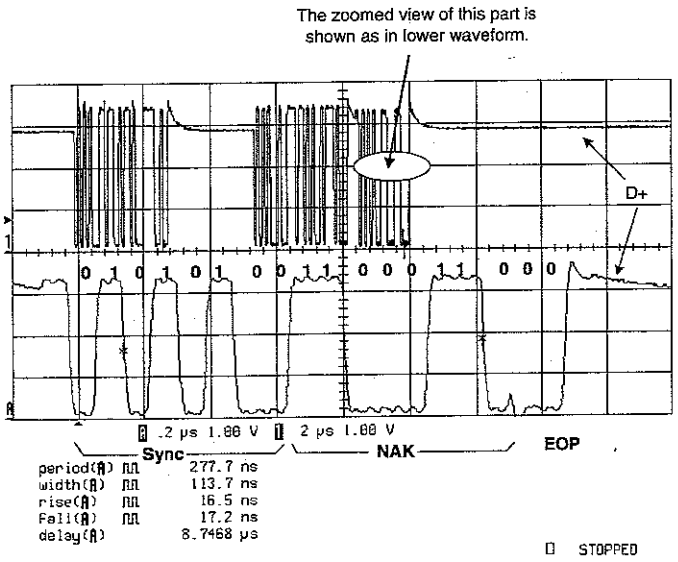


Fig. 9. Signal on USB D+ data line showing the sync, NAK and EOP data formats.

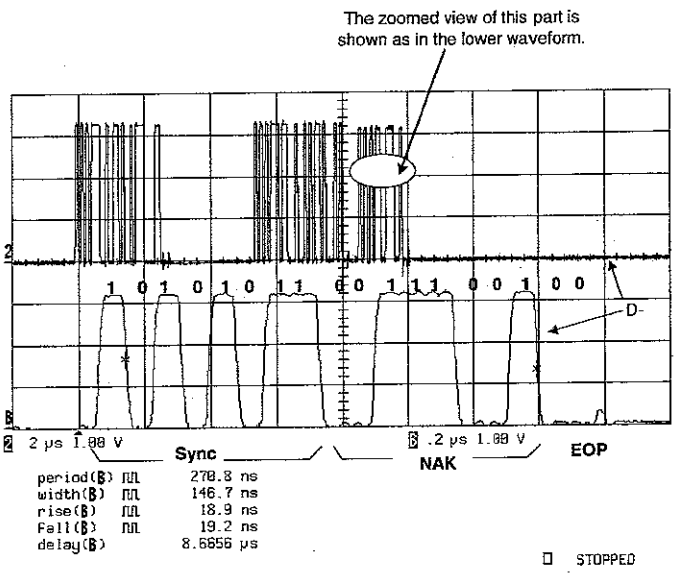


Fig. 10. Signal on USB D– data line showing the sync, NAK and EOP data formats.

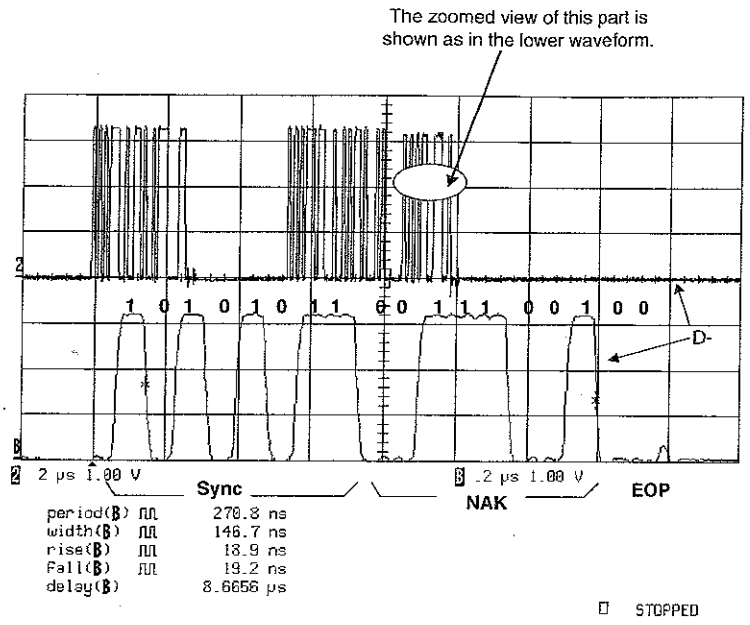


Fig. 10. Signal on USB D– data line showing the sync, NAK and EOP data formats.

(a)

Actual data		0							
Transmitted data (NRZI)	1	0	1	0	1	0	1	0	0

last bit of previous packet

(b)

"Sync" data ID

Actual data		0	0	0	0	0	0	0	1
Transmitted data (NRZI)	1	0	1	0	1	0	1	0	0

(c)

"NAK" data ID

Actual data		0	1	0	1	1	0	1	0
Transmitted data (NRZI)	0	1	1	0	0	0	1	1	0

Fig. 11. Decoding the inverted non-return to zero signal from the bus.

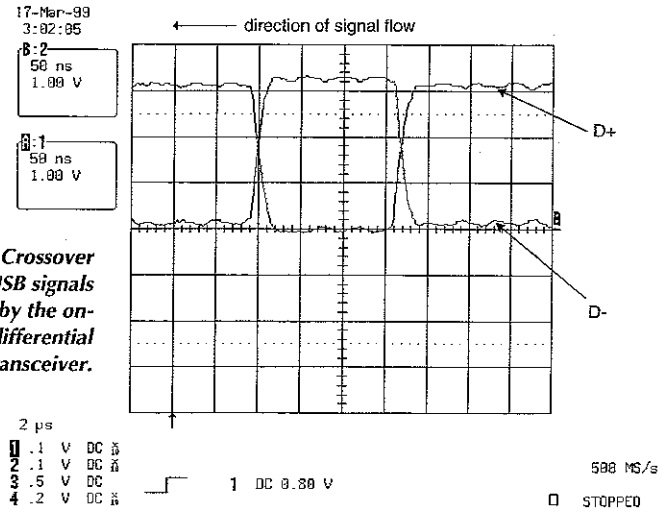


Fig. 12. Crossover points of USB signals generated by the on-chip differential transceiver.

the bus between hub and device is in low-speed mode since the device is low-speed.

We used the bus monitor to observe how the device packets are transmitted on a full-speed bus. **Table 4** shows part of the packet sequences. Packets #777 to #782 are the data and status stages of communication between host and hub using the normal full-speed transfer format.

Packets #784 to #794 perform three-stage control transfer between host and the low-speed device through the full-speed hub. In order to differentiate the low-speed signals from the high-speed for the hub to broadcast them to the downstream ports, a preamble (PRE) packet is required as shown.

Packets following the PRE (0x3C) are the low-speed data. Packets #784 and #785 are the get-descriptor command from host to the device, and packet #786 is the ACK from device.

How USB signals are transmitted

USB protocol involves non-return to zero, inverted, or NRZI, encoding to encode the data before transmitting onto the bus. This encoding method does not need a separate clock signal. In NRZI encoding, a transition between two consecutive data bits represents logic '0' while no transition represents logic '1'.

Figure 8 shows a serial data stream transmitted on the USB bus, encoded in NRZI format. The waveform was captured at the host side by probing on the data lines, D+ and D– while the device was sending an NAK response to the host in full-speed mode.

Logic bits for the D+ signal are shown in bold. For clarity, the D+ and D– signals are separated as in **Figs 9** and **10** respectively.

The upper waveform in Fig. 9 shows two packet transactions and the bottom waveform shows the magnified view of the last part of data transfer. It is easy to work out the actual data value from the waveform. **Figure 11** shows the transmitted NRZI data from Fig. 9.

The left-most bit '1' in the table of Fig. **11a)** represents the last bit transferred from the previous packet. This bit is used to decode the following 8 data bits. The second NRZI bit is '0'. The change from '1' to '0' is a transition, so the first bit of actual data is logic '0'.

In Fig. **11b)**, logic '1' in the actual data row indicates that there is no data transition between the previous NRZI bit, in this case '0'. Following the same procedure for the rest of NRZI bits, the 8-bit actual data decodes as 00000001, which is a specific synchronisation data pattern.

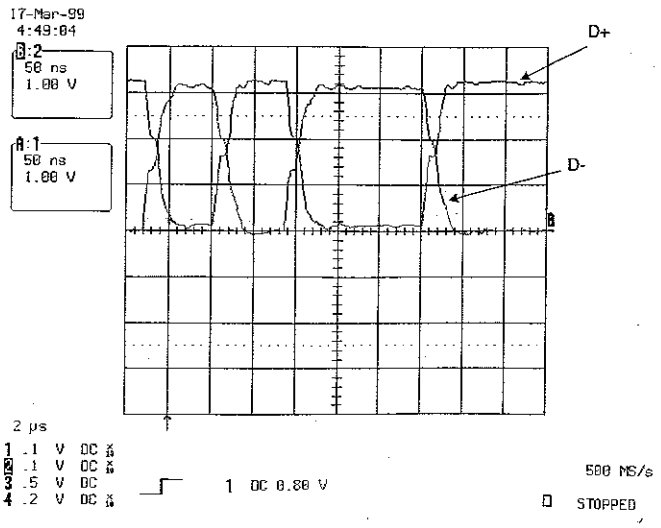


Fig. 13. Full-speed USB signals captured at the device side of the bus during sending of a negative-acknowledge NAK signal to the host.

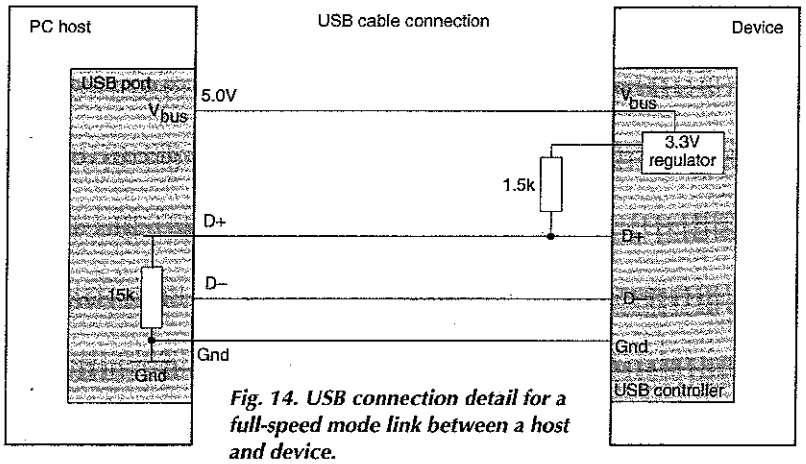


Fig. 14. USB connection detail for a full-speed mode link between a host and device.

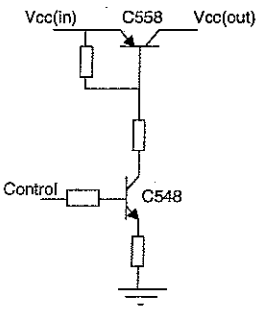


Fig. 15. Power switch control circuit used to switch off high-current circuitry during standby mode.

In Fig. **11c**), the next 8-bit NRZI data stream is 11000110_2, so the decoded actual data will be 01011010 which is an negative-acknowledge NAK signal.

The three bit times at the end of the waveform in Fig. 9 indicate the end-of-packet (EOP). Both data lines D+/D– are driven low for two bit times and back to high again at the third bit time for D+. The D– line stays low after the EOP as shown in Fig. 10. Some more NRZI examples are shown below:

Name (packet ID)	Actual data (hex)	NRZI code
Data1	D2	0 0 1 1 0 1 1 0
SOF	A5	0 1 1 0 1 1 0 0
ACK	4B	1 1 0 1 1 0 0 0
IN	96	0 1 0 0 1 1 1 0
Setup	B4	0 1 1 1 0 0 1 0

The two data lines should be overlapped in order to show the cross-over point. **Figure 12** shows a close-up view of the waveform of Fig. 8. The cross-over voltage point is about 1.7V and the maximum signal level is about 3.3V.

Signal quality issues

When measuring signals at the device side, a different signal quality can be obtained. The signal shown in **Fig. 13** was captured at the device side while the device was driving NAK to the host.

The shape of waveform is not very smooth at the cross-over points; this is due to impedance mismatch. The signal sent out from the device is partially reflected at the host side and back to the device. This makes it confusing for engineers measuring the signal quality as required in the USB Compliance checklist.

The signals should be captured at the host side when measuring the device transmission signal quality, and captured at the device side when the receiving signal quality is required. According to the USB specification, the rise/fall times for the transmitted signal should be both less than 20ns and 30ns for the received signals.

Power management

A pull-up resistor connected at the downstream end of the cable is used to determine the operating speed of the device. D+ is pulled high for full-speed mode, as shown in **Fig. 14**, while D– is pulled high for low-speed mode.

When a device is attached to the host or hub, there is a pull-down terminator with resistance of 15kΩ at the host/hub side to form a complete loop.

It follows that there is a constant current of about 200μA flowing from the pull-up resistor, through the USB cable and to the pull-down resistor. This current needs to be taken into account when designing USB equipment.

Current consumed by a USB device is an important issue for designers trying to meet the USB specification. This is especially true where devices are involved that obtain power from the host/hub through the V_{bus} power line of USB cable.

To avoid overloading the host or hub, a current of less than 100mA should be drawn by each attached device during normal operation. It should be less than 500μA in suspend mode.

Suspend mode and power down mode in the device and USB controller chip are interpreted differently. During suspend mode, the device should be able to receive the wake-up signal from the host so that it can resume normal operation.

This can be done by turning on the signal receiving circuit of the transceiver, which is normally an embedded on-chip module. The rest of the on-chip modules can then be set to power-down mode in order to reduce power consumption.

However, the constant current through the pull-up, and the current consumed by the transceiver in suspend mode result in less of the 300μA current being available to support other hardware in the device. For this reason, it is recommended to power-down the external hardware circuits to meet the specification. A simple power-control switch circuit is introduced, **Fig. 14**, to switch off the high-current components in the device while in suspend mode.

This circuit can be controlled by an on/off signal from the controller. It also provides a 'soft-turn-on' function that prevents excessive surge current drawn from the V_{bus} by all components at the same time during power on. There is a commercial chip, Infineon's *BCR 48 PN*, that can provide a similar function.

In summary

More details on the USB HID class device,[6] such as a keyboard interface, can be found in the application note mentioned in reference 7.

The introduction of the USB 1.1 standard provides an easy way for the connection of PC peripherals and more and more peripherals are being embedded with USB. Dedicated USB controllers play an important role in USB products.

The imminent USB 2.0 specification will enhance the capability of USB for transmitting multi-media signals. Its higher bandwidth provides a wider range of applications to the next-generation peripherals. ■

References

1. 'PC 98 System Design Guide', Intel Corporation and Microsoft Corporation.
2. 'An Embedded Chip for USB Applications: from the architecture to implementation', Proc. of 1999 International IC Conference & Exhibition, p. 1, Apr. 1999.
3. Don Anderson, 'Universal Serial Bus System Architecture', MindShare Inc., 1997.
4. 'USB 1.1 Specification', http://www.usb.org/
5. 'C541U User's Manual', 04.99, http://www.infineon.com/
6. 'USB Device Class Definition for Human Interface Devices', http://www.usb.org/
7. 'USB Device Enumeration Using C541 MCU', Application note #AP0833-01, Infineon Technology, July 99.