Developing a driver for a film scanner by means of USB sniffing and reverse engineering

Hugo Platzer

University of Salzburg

November 20, 2017

1 The USB standard

1.1 Motivation

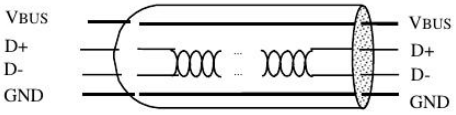
USB is an interface intended to connect various peripherals to PCs. These include: Human interface devices like keyboards and mice; storage devices like card readers, external hard disks, memory sticks and smartphones; multimedia devices like microphones, speakers, cameras and scanners. Some highlights leading to its wide adoption:

- Unified interface for all kinds of peripherals
- Plug and play: The user plugs in the device, the configuration (e.g. loading the appropriate drivers) is done automatically by the operating system
- Number of ports can be increased using hubs. Multiple hubs can be chained allowing for up to 127 devices on a single root port.
- High data rate of 480 Mbit / s (USB 2.0 high-speed mode). Also offers low-latency transfers for real-time audio/video applications
- Backwards compatibility: Older USB 1.1 devices can be used at 2.0 hosts. High-speed USB 2.0 devices can also be used on older machines supporting only USB 1.1 (albeit at lower speed).

1.2 Electrical side

A USB cable has four wires: One as ground, VBUS for a 5 V power supply and two for data transmission. The power line allows it to draw up to 100 mA without any configuration. This is useful for simple devices that are not using the data lanes, just the power, like USB lights. Also it allows for devices not taking much power to be self-powered which eliminates the need for an extra

Figure 1: USB cable cross-section



power supply and connector. Devices can ask the host for more power (up to 500 mA), those that need even more (like a lot of scanners) need an external supply.

1.3 Signaling

1.3.1 Low-level states

USB is a serial bus which means there is only a single path for data transmission. Differential signalling across the D- and D+ wires is used, which means the difference in voltage across the two wires (rather than some absolute) determines the state. This is beneficial because noise during transmission should affect both lines equally, not changing the difference. Higher frequencies and thus data rates become possible.

Table 1: USB speed modes

Mode	Speed	Bit time
Low Speed	1.5 Mbit / s	667 ns
Full Speed	12 Mbit / s	83 ns
High Speed	480 Mbit / s	2 ns

Table 2: Low-level data line states (only applies to Full Speed)

Levels	State
Differential '0'	D- high, D+ low
Differential '1'	D- low, D+ high
Single Ended Zero (SE0)	both low
Single Ended One (SE1)	both high (illegal state, should never happen)
Data 'J' state	Differential '1'
Data 'K' state	Differential '0'
Idle state	Data 'J' state
Start of Packet (SOP)	Switch from idle to 'K'
End of Packet (EOP)	SE0 for 2 bit times followed by 'J' for 1 bit time
Disconnect	SE0 for $\geq$ 2 us
Connect	Idle for 2.5 us
Reset	SE0 for $\geq$ 2.5 us

Low Speed is used for devices where speed is not important (mice, keyboards). It allows for cheaper cables and electronics. High Speed is only available in USB 2.0. For reasons of simplicity, only full speed signaling will be covered here.

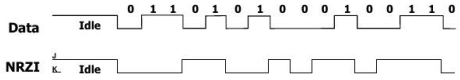
SE0 is the state of the data lines if no device is connected. The host recognizes a device being plugged in by the D+ line being "pulled up" to high. It then most likely initiates a reset so the device is in a known state for communication to begin. Similarly, a disconnect is sensed by a SE0 for some time.

It is important to note that **all communication on the USB bus is initiated by the host**. Devices on the bus can not directly talk to each other and can only talk to the host as a direct response to a request made by it before.

1.3.2 Bitstream encoding

USB uses NRZI encoding for the transmitted data: A zero is represented by a change to the opposite state while a one is represented by staying in the same state.

Figure 2: NRZI bitstream encoding



For keeping the receiver clock in sync with the data it is not ideal if the signal stays at J or K for too long. To prevent this, a technique called "bit stuffing" is used: Before doing NRZI encoding, a zero is inserted after every six consecutive ones in the data. The receiver recognizes the stuffed bits during decoding and discards them.

1.4 Packets

Packets are the atomic unit of data transmission in USB. In between packet transmission, the bus remains in an idle state. Every packet starts with a sync pattern to synchronize the clocks between sender and receiver. Next are the actual data bits. The packet is terminated by an EOP state. Fields in a packet are transmitted least-significant bit first.

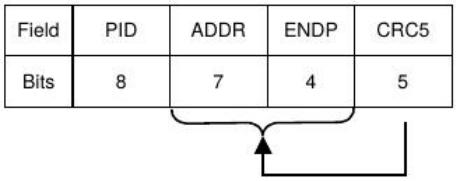
The first 8 bits of every packet contain the packet ID (PID) which identifies its type and thus how the rest of the packet data should be interpreted. The PID is 4 bits long, they are transmitted a second time in reverse bit order to allow the receiver to quickly discard a faulty packet. There are 17 different packet types (PRE and ERR have the same ID, some are only relevant for High Speed):

Table 3: USB packet types; notice how the least-significant two bits identify the packet category

PID type	PID name	PID bits (3..0)	Description
Token	OUT	0001	Address + endpoint number for host-to-device transaction
	IN	1001	Address + endpoint number for device-to-host transaction
	SOF	0101	Start-of-frame marker, frame number
	SETUP	1101	Special host-to-device transaction for device configuration
Data	DATA0	0011	Data packet
	DATA1	1011	Data packet
	DATA2	0111	Data packet (only High Speed)
	MDATA	1111	Data packet (only High Speed)
Handshake	ACK	0010	Receiver accepts error-free data packet
	NAK	1010	Receiver cannot accept data or transmitter cannot send data
	STALL	1110	Endpoint halted or control pipe request not supported
	NYET	0110	Data packet (only High Speed)
Special	PRE	1100	Preamble to enable downstream traffic to low-speed devices
	ERR	1100	Split Transaction error handshake
	SPLIT	1000	High speed Split Transaction token (only High Speed)
	PING	0100	High speed control flow probe (only High Speed)
	Reserved	0000	Reserved PID

1.4.1 Token packet

Figure 3: Token packet format

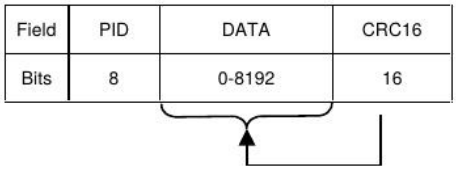


Token packets are used at the start of so-called transactions to specify the target of the transaction on the bus, namely a certain device and endpoint. There are 127 possible devices on a bus (address 0 is reserved for a device that has not been configured yet).

Endpoints are logical entities on a device that are used as sources and sinks of data in so-called pipes. A pipe is either in OUT (to device) or IN (to host) direction. Endpoint 0 is a special bidirectional pipe that must be available on every device right after the reset. It is used mainly for identifying and configuring the device.

1.4.2 Data packet

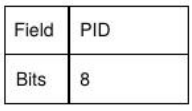
Figure 4: Data packet format



Used to transmit the actual data in a transaction.

1.4.3 Handshake packet

Figure 5: Handshake packet format



Used to report the status of a transaction.

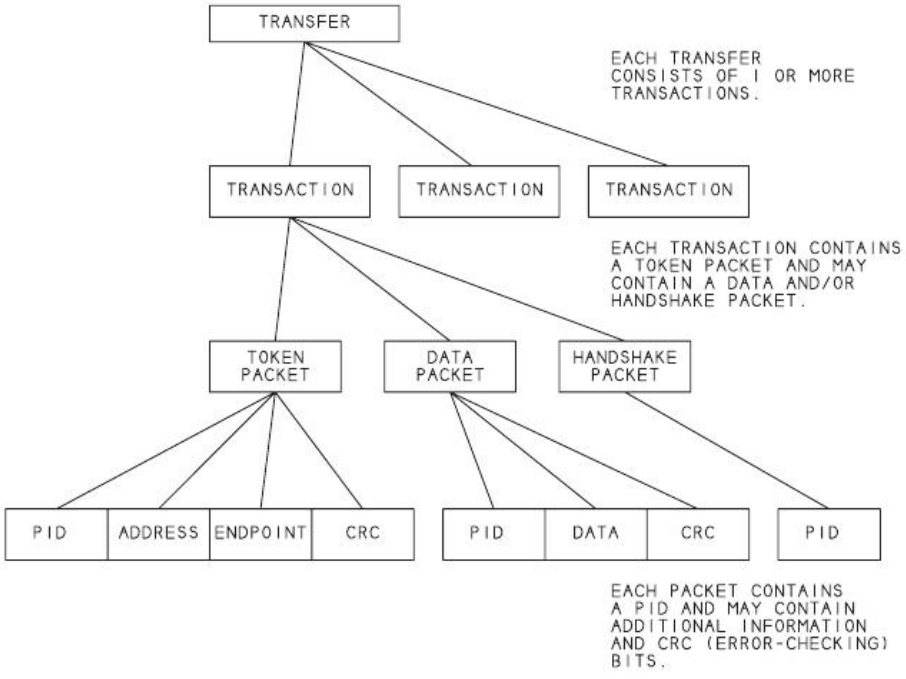
1.5 Transfers

Transfers are the abstraction level of data exchange as seen at the software side of the host. From the host's perspective, a transfer transmits a string of bytes from / to the device. A transfer is directed to one of the device's endpoints, each of them has one of four transfer types plus a transfer direction asscociated with it during device configuration. Transfers are composed of transactions, which usually consist of three packets:

1. A token packet to tell the type of the transaction and its destination (as all USB communication is initiated by the host, the token packet always comes from the host)
2. A DATA packet for the actual payload. Length can vary between 8-64 bytes on Full Speed links. This can go host to device or device to host.
3. A handshake packet (usually ACK, NAK) lets the sender know if the data was received successfully.

Transfers usually consist of multiple transactions.

Figure 6: Composition of a USB transfer



1.5.1 Control transfer

Every device must have endpoint 0 for control transfers. Control endpoints are message-based. This means each transfer is for a specific message which has defined length, format and purpose. There can be transfers in either direction on the same endpoint, although each transfer has a specified (data stage) direction. Control transfers start with a Setup stage / transaction specifying the target device and endpoint plus an extra 8 bytes of data called the Device Request.

Table 4: Device Request format

Offset (bytes)	Field	Size (bytes)	Description
0	bmRequestType	1	Bits 0..4: Recipient 0 Device 1 Interface 2 Endpoint 3 Other 4..31 Reserved Bits 5..6: Type 0 Standard 1 Class 2 Vendor 3 Reserved Bit 7: Transfer direction 0 Host to device 1 Device to host
1	bRequest	1	Specific request
2	wValue	2	Varies according to request
4	wIndex	1	Varies according to request, typically used for a index or offset
6	wLength	2	Number of bytes to be transferred in the data stage (0 means no data stage)

Depending on the kind of control transfer, there may be a data stage consisting of data transactions, but this is not required. If there are data transactions, all are going in the same direction as specified by the request type. The transfer is completed with a Status transaction to report back about the success of the whole transfer.

Control transfers are used right after the device reset to get information about its capabilities and set a configuration. For this, there are 11 bRequest values called Standard Requests that have to be supported by all devices as part of the standard.

Table 5: Some important Standard Requests

bRequest	bmRequestType (7..0)	wValue	wIndex
GET_DESCRIPTOR	10000000	Descriptor type and index	Zero or Language ID
SET_ADDRESS	00000000	Device address	Zero
GET_STATUS	10000000 10000001 10000010	Zero	Zero Interface Endpoint
SET_CONFIGURATION	0000000	Configuration value	Zero

bRequest	wLength	Data
GET_DESCRIPTOR	Descriptor length	Descriptor
SET_ADDRESS	None	None
GET_STATUS	Two	Device, Interface or endpoint status
SET_CONFIGURATION	None	None

They are also used to control the device after the configuration phase during normal operation. There are standardized requests (Class Requests) for certain device classes like keyboards, storage devices etc.

Other non-standard devices have custom, vendor-specific requests that are not documented in the USB standard.

1.5.2 Bulk transfer

Bulk transfers are used to transmit large amouts of data with guaranteed delivery and low overhead but no latency / bandwidth constraints. Compared to control transfers, they are stream-based. This means there is no defined message format or size. For an IN endpoint, the host would poll the device endpoint for data until it receives (cumulative) as many bytes as requested by the software. A transfer can also be ended by the device sending a data transaction of less size than the maximum of this endpoint (data is always split so there is at most one smaller packet at the end) or a zero-size data transaction. To avoid data packets being lost, they alternate between the DATA0 / DATA1 PIDs. A single bulk endpoint is only for incoming or only for outgoing transfers. The host schedules them at times when the bus bandwidth is not used by other transfer types. Typical uses for bulk transfers are transferring data from/to an external harddrive, to a printer or from a scanner.

1.5.3 Interrupt transfer

On the bus, interrupt transfers look the same as bulk transfers. The main difference is scheduling: The host guarantees that a transfer attempt is made as often as specified in the endpoint descriptor (1 - 100 ms). Despite the name, interrupt transfers have nothing to do with hardware interrupts. All communication on the USB bus is initiated by the host, so it has to poll the device for data. Typical uses for interrupt transfers are receiving keypresses from keyboards and movements from mice.

1.5.4 Isochronous transfer

Isochronous transfers are for transmitting data at a constant rate with guaranteed latency. Compared to interrupt transfers, they offer a guaranteed transfer rate (with interrupt transfers the interval between two transfers can be anywhere between zero and the specified maximum). The drawback is a lack of any handshake / retry mechanism. Typical uses for isochronous transfers are USB sound cards or webcams.

1.6 Enumeration, Configuration

Enumeration is the process of the host learing about a new device after it has been plugged in. For this, the host assigns a new address to the device, queries a variety of descriptors to find out the device's capabilities, selects a configuration profile and loads a suitable driver.

The host will learn about a new device that is plugged in by being notified by the hub (every USB bus has at least a root hub). It asks the hub to apply a reset to the device, then does a SET_ADDRESS device request to move the new device from its default address zero to an unused one.

1.6.1 Device descriptor

After setting the address, a GET_DESCRIPTOR device request is made, asking for the device descriptor.

1.6.2 Configuration descriptor

1.6.3 Interface descriptor

1.6.4 Endpoint descriptor

References

[1] Universal Serial Bus Specification, Revision 2.0. April 27, 2000. Available at `http://www.usb.org/developers/docs/usb20_docs/`

[2] USB Complete: Everything You Need to Develop Custom USB Peripherals (3rd edition). Jan Axelson, Lakeview Research LLC 2012. ISBN: 978-1-931448-03-1