

# Grundkurs

von Benedikt Sauter 10.02.2007

Mit diesem Grundkurs soll jeder auf eine einfache Art und Weise den USB Bus verstehen können.

Inhaltsverzeichnis [[verbergen](#)]

1. [USB verstehen und anwenden](#)
  1. [Warum ist der Einstieg in USB so schwer?](#)
  2. [Ein einfaches Modell zum besseren Verstehen](#)
  3. [Endpunkte](#)
  4. [Interface](#)
  5. [Konfiguration](#)
  6. [Deskriptoren](#)
  7. [Plug and Play - Geräteerkennung und -zugriff](#)
  8. [USB Klassen](#)
2. [Glossar](#)

## USB verstehen und anwenden

### Warum ist der Einstieg in USB so schwer?

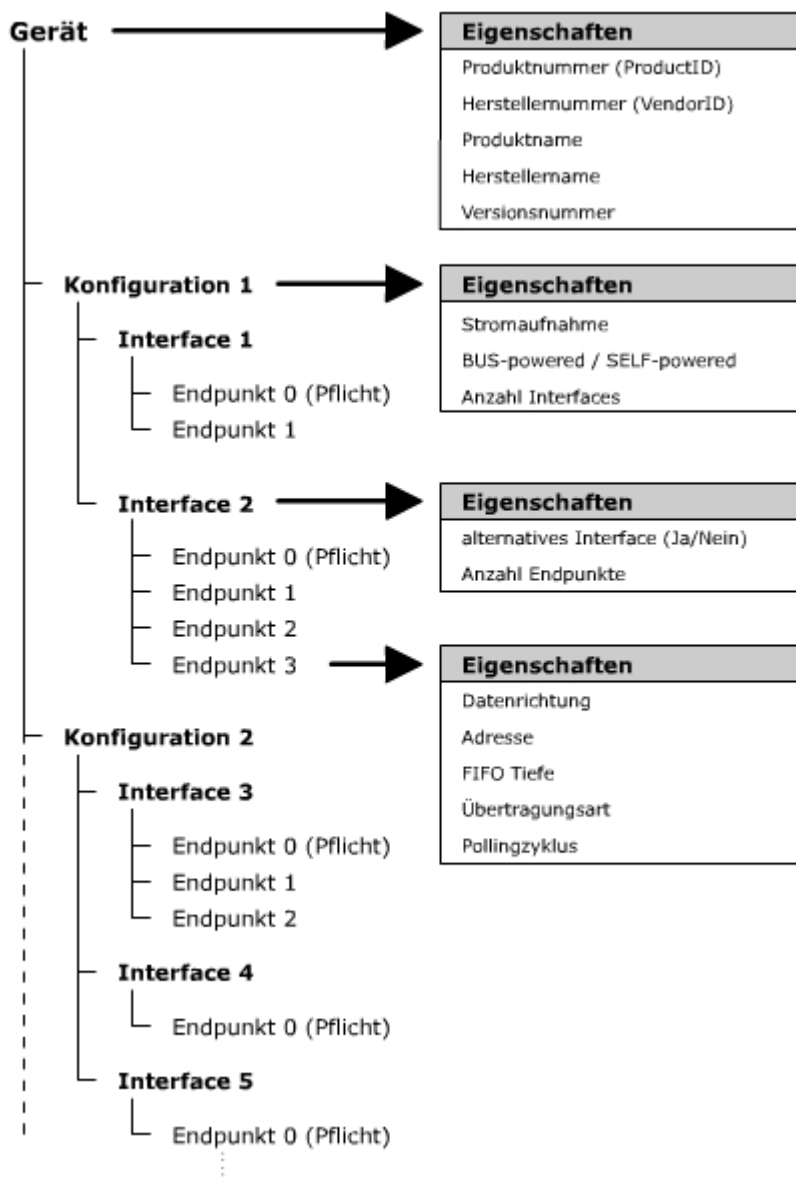
USB ist für Mikrocontroller-Schaltungen eine ideale Ergänzung. Schnelle Übertragungsraten, flexible Kommunikationskanäle, integrierte Stromversorgung und Echtzeitfähigkeit sind die bekanntesten Schlagwörter von USB. Doch obwohl USB bereits seit 1996 existiert, findet man immer noch sehr häufig die klassische RS232 Verbindung in neuen Projekten.

Es stellt sich also die Frage, warum es die USB Schnittstelle bis heute nicht geschafft hat, RS232 abzulösen. Meiner Meinung nach ist eines der Hauptprobleme, dass auf diesem Gebiet viel zu wenig gute Literatur existiert. Oftmals hört man von Entwicklern, dass es bei Weitem nicht ausreicht, etwas flüchtig über die USB Schnittstelle zu lesen, um sie richtig verstehen und einsetzen zu können. Das liegt wohl daran, dass die meisten Texte und Bücher leicht veränderte Übersetzungen der knapp tausendseitigen USB Spezifikation sind.

Die USB Spezifikation ist zwar im Gegensatz zu vielen anderen Spezifikationen sehr ausführlich verfasst, jedoch ist und bleibt sie die komplexe Spezifikation von USB. Daher ist sie als Lernunterlage für das Selbststudium oder als Entwicklungsleitfaden völlig ungeeignet.

Mit dieser Internetseite (PDF Datei) soll diesem Problem entgegengewirkt werden. Es wird ein einfaches Modell eingeführt, mit dem man schnell in der Lage sein soll USB verwenden zu können.

### Ein einfaches Modell zum besseren Verstehen



Eine USB Schnittstelle hat kein typisches Aussehen, wie z.B. eine Schnittstelle vom Typ RS232. Bei einer RS232-Verbindung gibt es immer mindestens je eine TX und RX Leitung. Die TX Leitung ist ausschliesslich dafür da, um Daten zu senden, die RX Leitung, um Daten zu empfangen. Eine solche feste Regel gibt es bei USB nicht. Hier ist es möglich, mehrere RX und TX Leitungen parallel in einer USB Schnittstelle zu definieren. Dadurch kann man verschiedene Daten über eigene Verbindungen übertragen.

Stellen wir uns die USB Schnittstelle kurz einmal folgendermassen vor: Wir können Daten gezielt über die verschiedenen Leitungen senden. Wenn wir beispielsweise eine Soundkarte bauen wollen, benötigen wir zwei TX Leitungen für die beiden Stereokanäle und eine weitere TX Leitung für Steuerkommandos. Um den Zustand der Soundkarte abfragen zu können, wäre zusätzlich noch eine RX Leitung sinnvoll.

In der Realität könnte man eine solche RS232 Verbindung nachbauen. Schliesslich würde sie aber keiner nutzen wollen, denn man würde mindestens drei serielle Schnittstellen auf einem Rechner blockieren. Desweiteren müsste man jedesmal darauf achten, dass man mit dem Gerät genau die richtigen Leitungen vom Computer aus verbindet. An dieser Stelle kommt eine wichtige Funktion von USB ins Spiel. Mit USB kann man sich virtuell die Leitungen so legen, wie man sie benötigt.

Damit ein Computer weiss, welche Leitungen ein Gerät hat, fragt er dies einfach nach dem

Anstecken ab.

## Endpunkte

An dieser Stelle wollen wir die Namen TX und RX vergessen und dafür den Namen Endpunkt verwenden, denn so werden die Datenübertragungskanäle eines USB-Gerätes genannt. Bei USB muss jedem Endpunkt eine Richtung und Adresse zugewiesen werden. Eine Adresse gibt es bei der klassischen RS232 Verbindung nicht, denn jeder Datenkanal (sprich RX, TX, CTS, RTS usw...) hat eine separate Leitung. Hier liegt der wohl grösste Unterschied zur klassischen RS232 Verbindung: USB ist, wie der Name schon in sich trägt, ein Bus. Auf einem Bus werden immer nur Datenpakete übertragen. Daher muss jeder Endpunkt in einem Gerät ebenfalls eine Adresse haben. Weil jedes Gerät nach dem anstecken vom Betriebssystem eine Geräteadresse bekommt, kann man über die Geräte- und Endpunktadresse direkt diese Datenübertragungskanäle antsteuern.

### USB Controller (kurzer Ausflug):

Will man Daten über eine Netzwerkverbindung senden, so macht man sich keine Gedanken darüber, wie man die einzelnen Kupferleitungen schalten muss, sodass die Daten richtig übermittelt werden. Genauso ist es bei USB. Bei Verwendung von USB sollte man mit dieser Problematik in der Regel nicht konfrontiert werden, denn sie wird vom USB Controller übernommen. Hier gibt es ein großes Feld an Controllern, beginnend mit sehr einfachen, die gerade einmal das Signal von Busleitungen generieren, bis hin zu sehr grossen komplexen Controllern mit internen Konfigurationsregistern für die Endpunkte, oder internem Speicher für ein- und ausgehende Daten u. v. m.

Die meisten Controller weisen intern eine verschiedene Anzahl von kleinen FIFO Speichern auf (meist bis zu 64 Byte). Diese FIFO Speicher kann man direkt einem Endpunkt zuweisen (für ein- und ausgehende Daten). In der Endpunktdefinition muss die FIFO Tiefe als Parameter angegeben werden.

Besprochen wurden bisher die Parameter Tiefe des FIFOs, die Adresse des Endpunktes und dessen Richtung. Zusätzlich gibt es einen weiteren Parameter, die Transferart. Für verschiedene Anwendungen bietet USB bereits direkt die passenden Transferarten an. Dadurch muss man keine eigenen Algorithmen schreiben, welche die Daten auf Korrektheit überprüfen.

- |                           |  |
|---------------------------|--|
| <b>Bulk-Transfer</b>      | Der Bulk-Transfer wird am meisten genutzt. Es können grosse und zeitkritische Datenmengen übertragen werden. Zusätzlich überprüft dieser Transfer stets die Korrektheit der Datenübertragung.  |
| <b>Interrupt-Transfer</b> | Diese Übertragungsart darf man nicht falsch verstehen. USB ist und bleibt ein Single Master Bus. Das heisst, dass nur der Master jegliche Kommunikation initiieren kann. Kein Gerät kann sich beim Master selbst anmelden und ihm mitteilen, dass es Daten übertragen will. Der Master muss zyklisch alle Geräte nach neuen Daten abfragen. Im Grunde ist der Interrupt Transfer nichts anderes als der Bulk-Transfer mit dem Unterschied, dass die Interrupt Endpunkte eine höhere Priorität und mehr Bandbreite bekommen. Auf diese Weise kann der Master immer zu dem gewünschten Zeitpunkt auf das Gerät zugreifen, selbst wenn gerade viel Datenverkehr auf dem Bus ist. Diesen zyklischen Zugriff stellt man über ein Pollingintervall in Millisekunden ein. |
| <b>Isochron-Transfer</b>  | Mit dem Isochronen Modus kann man Daten übertragen, die eine konstante Bandbreite erfordern. Typische Anwendungsbeispiele sind die Übertragung von Audio oder Videosignalen. Geht hier ein Bit oder Byte verloren, äussert sich das nur in einem Knacken oder Rauschen. Würden die Daten aber verzögert ankommen, wäre die Sprache oder das Bild völlig verzerrt und daher unbrauchbar.  |

Man muss also genauso wie beim Interrupt-Transfer das Pollingintervall definieren, und angeben, wie oft der Master Daten abholen oder versenden soll.

Pro Endpunkt muss definiert werden:

1. Richtung
2. Adresse
3. FIFO Tiefer
4. Übertragungsart
5. Pollingzyklus

Jeder eigene Endpunkt kann mit diesen Parametern frei definiert werden. Die Daten, die man später über die Endpunkte sendet oder empfängt, sind selbst definierte Datenpakete.

Es gibt nur einen einzigen Endpunkt, der anders arbeitet, der Endpunkt 0. Er wird vom Betriebssystem benötigt, um das USB Gerät zu konfigurieren. Über ihn werden von der USB Spezifikation definierte Nachrichten gesendet. Der Endpunkt 0 ist auch der einzige, der in zwei Richtungen betrieben werden kann, und nicht wie die anderen nur in eine Richtung. Zusätzlich gibt es für den Endpunkt 0 eine eigene Übertragungsart, den Controll Transfer, der auch nur vom Endpunkt 0 unterstützt wird.

## Interface

Ein Interface ist ein Bündel an Endpunkten. Ein Gerät kann mehrere Interfaces anbieten. So kann eine Soundkarte ein Interface für den Mono- und eines für den Stereobetrieb haben. Das Interface für den Monobetrieb hat einen Endpunkt für die Steuerkommandos und einen weiteren für die Daten, die über einen Lautsprecher ausgegeben werden. Das Interface für den Stereobetrieb hat ebenfalls einen Endpunkt für Steuerkommandos, aber zwei für die Signalausgabe (linker und rechter Kanal). Die Software auf dem PC kann jederzeit zwischen den Interfaces hin- und herschalten. Oft liest man auch vom Begriff Alternate Interface. Dieses Interface kann man parallel zu einem anderen Interface definieren. Definiert man ein normales Interface, so gibt man dort die Endpunkte, die zu ihm gehören, an. Entsprechend der FIFO Grösse eines Endpunktes wird die entsprechende Bandbreite auf dem USB Bus reserviert.

Die Bandbreite wäre auf diese Weise sehr schnell aufgebraucht, auch ohne dass Kommunikation auf dem Bus stattfindet. Würde man aber die benötigte Bandbreite immer nur kurz vor dem Senden oder Empfangen reservieren, könnte man viel mehr Geräte über einen Bus bedienen. Daher wurde das Alternate Interface erfunden. Zu jedem Interface kann es also ein alternatives Interface geben.. Die Endpunktstruktur sollte genauso aussehen wie die vom normalen Interface. Der einzige Unterschied ist der, dass überall als FIFO Grösse 0 Byte angegeben ist. Gibt es jetzt ein Alternate Interface, aktiviert das Betriebssystem beim einstecken erst dieses, und nimmt so nicht voreilig anderen die Bandbreite weg. Kurz vor dem Senden und Empfangen wird dann auf das eigentliche Interface gewechselt.

## Konfiguration

Genauso wie Interfaces kann ein Gerät mehre Konfigurationen haben. Hier geht es um die elektrischen Eigenschaften. Bei USB kann können die Geräte direkt über das USB Kabel mit Strom versorgt werden. So kann man von einem Bus max 5V und 500mA beziehen. Bevor ein Gerät den Strom nuzen kann, muss es beim Master erfragen, ob noch genügend freie Kapazitäten vorhanden sind.

In einer Konfiguration muss man folgende Parameter definieren:

1. Stromaufnahme in 2 mA Einheiten
2. Attribute (z.B. Bus-powered, Remote-Wakup-Support)
3. Anzahl der Interfaces unter dieser Konfiguration

## Deskriptoren

Jetzt wissen wir, wie man Endpunkte definiert und diese in Interfaces anordnet. Ebenfalls können wir verschiedene Interfaces einer Konfiguration zuweisen. In der Konfiguration werden dazu noch Parameter für den Stromverbrauch und Anschluss definiert. Alle diese Informationen sind immer in Datenstrukturen verpackt, die von der USB Spezifikation Deskriptoren genannte werden. Ein Descriptor ist nichts anderes als ein Speicherarray, an dem jede Stelle für einen bestimmten Parameter steht. Ein Gerät muss intern an irgendeiner Stelle einen Speicher haben, in dem diese Strukturen liegen, denn das Betriebssystem kann jederzeit diese Informationen abfragen.

Es gibt also einen Endpunkt-Descriptor, Interface-Descriptor, Konfigurations-Descriptor und einen Geräte-Descriptor.

Die Parameter der ersten drei kennen wir bereits. Kommen wir nun zum Geräte-Descriptor.

Der Geräte-Descriptor muss in jedem Gerät vorhanden sein. Hier ist definiert:

### **USB Version**

USB Version die das Gerät unterstützte (z.B. 1.1)

### **Klassen- / Subklassen- / Protokoll-Code**

Das USB Konsortium hat nicht nur den USB Bus definiert, sondern gibt auf Beschreibungen von Endpunkt Bündeln für Geräte heraus. So können Betriebssysteme Standardtreiber anbieten. Mehr zu dieser Technik ist im Bereich USB Klassen dieses Dokuments zu finden.

### **FIFO Tiefe von EP0**

Tiefe des FIFOs, der für den Endpunkt 0 zuständig ist. Dieser ist bei USB 1.1 meist 8 Byte tief.

### **Hersteller Nummer**

Jeder Hersteller von USB Geräten muss sich bei [www.usb.org](http://www.usb.org) registrieren. Dafür bekommt man dann eine eindeutige Nummer, die für die Treibersuche vom Betriebssystem von Bedeutung ist.

### **Produkt Nummer**

Die Produktnummer wird ebenfalls (wenn sie definiert ist) vom Treiber verwendet, um das Gerät eindeutig zu identifizieren. Mehr zu diesen Nummern kann man im Bereich Plug and Play - Geräteerkennungsfahren.

### **Versions Nummer**

Versionsnummer für das Gerät

### **String Index für Hersteller**

Hier kann ein Name für den Hersteller angegeben werden, der vom Betriebssystem angezeigt werden kann.

### **String Index für Produkt**

Hier kann ein Name für das Produkt angegeben werden.

### **String Index für Seriennummer**

Und hier eine Seriennummer.

In dem Gerätedeskriptor wird nicht direkt der Name für Hersteller, Produkt oder Seriennummer gespeichert, sondern nur eine Nummer eines sogenannten String-Deskriptors. Er ist wiederum eine einfache Datenstruktur im USB Gerät, in dem dann die einzelnen ASCII-Buchstaben stehen.

### **Anzahl der Konfigurationen**

Das ist die Anzahl der vorhandenen Konfigurationen für das Gerät. Eine Kamera könnte hier zwei Konfigurationen haben. Es gibt eine Konfiguration in der die Kamera, die den Strom vom USB Bus bezieht und eine, von der sie den Strom aus den eigenen Batterien bekommt.

## **Plug and Play - Geräteerkennung und -zugriff**

Dadurch, dass im USB Gerät alle Eigenschaften mit den Deskriptoren gespeichert sind, kann das Betriebssystem direkt nach dem anstecken viele Details von dem Gerät erkennen. Es kann z.B. dem Nutzer anzeigen, dass es ein Gerät X vom Hersteller Y gefunden hat.

### *Wie genau sieht eigentlich der Ablauf dahinter aus?*

Kommen wir auf den Endpunkt 0 zurück. Über den EP0 werden definierte Nachrichten gesendet. Mit diesen Nachrichten kann das Betriebssystem alle definierten Deskriptoren abfragen. So gibt es z.B. eine Nachricht Get Descriptor, um einen beliebigen Deskriptor abfragen zu können, oder Get Configuration, um das Gerät nach der aktuellen Gerätekonfiguration zu fragen. Wenn man auf einen Endpunkt zugreifen muss, dann immer über die Geräte- und Endpunktadresse.

### *Wie kommt aber ein Gerät zu einer Geräteadresse?*

Direkt nach dem Anstecken hat das Gerät die Adresse 0. Der Master erkennt, dass ein neues Gerät eingesteckt worden ist, und sendet an dieses Gerät die Anfrage GetDescriptor? (Gerätedeskriptor). Daraufhin antwortet das Gerät mit dem entsprechenden Deskriptor. Jetzt weiss der Master, dass es sich um ein echtes USB Gerät handelt, und ordnet ihm eine neue endgültige Adresse zu. Ab diesem Zeitpunkt ist das Gerät über die neue Adresse erreichbar.

Nur woher soll der Programmierer wissen, welche Adresse das USB Gerät später auf einem Computer hat? Die Adresse wird schliesslich nach der Anzahl der angesteckten Geräte erhöht.

Wenn man eine Verbindung zu einem USB Gerät aufbaut, kann man dies nicht über einen eindeutigen Punkt (wie z.B. c:/ oder /dev/hda0 bei einer Festplatte) machen. Die Prozedur ist immer folgende: Das Betriebssystem legt in einer eigenen internen Struktur alle abgefragten Deskriptoren von den angesteckten USB Geräten ab. Sucht man eine Adresse für ein Gerät, muss man in den Datenstrukturen nach dem dazu passenden Gerät suchen. Dies kann man entweder über die Hersteller- und Produkt-Nummer machen, oder über einen Klasse- / Sub- / oder Protokoll-Code. Falls ein Gerät nur über einen Stringdeskriptor beschrieben wird, kann man die Adresse auch über diesen ermitteln.

## **USB Klassen**

Das USB Klassenmodell soll die Entwicklung von Treibern erheblich vereinfachen. Die Idee dahinter ist ganz einfach. Mit den Deskriptoren beschreibt man das Aussehen der Schnittstelle. Welche Controller und Techniken dahinter stehen ist aus der USB Sicht unerheblich. Wenn wir unseren Soundkarte ansehen, so haben wir nur gesagt das ein bestimmter Endpunkt für die Ausgabe der Soundsignale für den rechten Stereokanal da ist. Was genau mit dem Byte Strom hinter der USB Schnittstelle passiert, ist dem USB Treiber nicht wichtig. Er schickt nur die definierten Bytes, die das Audiosignal widerspiegeln. Und das ist die wesentliche Idee hinter den USB Klassen. Es sollen für Geräte mit gleichen Merkmalen und Eigenschaften Gruppen von Interfaces und Endpunkten

definiert werden. Betriebssysteme können für diese Geräte Treiber anbieten, da sie kein einziges spezielles Register oder Zeitverhalten irgendeines Controllers kennen müssen. Der Treiber muss nur so geschrieben werden, dass er die Daten richtig formatiert an die Endpunkte verteilt und abholt. Dies entlastet Hersteller typischer PC Komponenten (Tastatur, Maus, Soundkarte, Scanner, Drucker, ...) von der Bereitstellung spezieller Treiber.

Leider funktioniert diese Idee nur selten. Einzig bei Tastaturen und Mäusen nehmen die Hersteller an diesem Konzept teil. Irgendetwas hält die Hersteller davon ab, sich an diesen Standard zu halten. Sie entwickeln lieber eigene proprietäre Treiber.

Es gibt Klassenspezifikationen für:

1. Tastaturen, Mäuse, ... (Human Interface Device Class)
2. Soundkarten (Audio Device Class)
3. Kommunikationsschnittstellen z.B. RS232, Ethernet (Communication Device Class)
4. Content Security Class
5. Chip-/Smart Card Device Class
6. IrDA Bridge Device Class
7. Imaging Device Class
8. Printer Device Class

Für die meisten Klassen gibt es in allen bekannten Betriebssystemen Standardtreiber. Speziell für die Communication Device- oder Human Interface Class, gibt es Beispielprojekt auf dieser Seite (Netzwerkstack in USB Gerät, PS2 zu USB Wandler).

Wenn man einen solchen Standardtreiber verwenden will, muss man im Geräte-Deskriptor die richtigen Klassen- und Protokoll-Codes angeben. Man kann sogar einem Interface eine Klasse zuweisen. In der Praxis bedeutet dies z.B. bei einem dieser bekannten Multifunktionsgeräte (Fax/Kopierer/Scanner/Drucker), dass es für jedes virtuelle Gerät ein Interface mit dem entsprechenden Klassencode gibt. Im Idealfall muss man sich nur um die Firmware im Gerät kümmern, da man keinen einzigen Treiber selbst schreiben muss.

## Glossar

**FIFO** (engl. etwa Erster rein - Erster raus), häufig abgekürzt mit FIFO, gleichbedeutend mit **Speicher**: First-Come First-Served bzw. FCFS, bezeichnet jegliche Verfahren der Speicherung, bei denen diejenigen Elemente, die zuerst gespeichert wurden, auch zuerst wieder aus dem Speicher entnommen werden (entnommen aus [<http://de.wikipedia.org/wiki/FIFO>]).

**Polling**: Polling bezeichnet in der Informatik die Methode, den Status von bestimmter Hard- oder Software mittels zyklischem Abfragen zu ermitteln.

---

<http://www.usb-projects.net/cwiki.php?page=Grundkurs>  
Letzte Änderung: 12.02.07 09:20 Version 3.5.16

[Zurück](#)