



TECHNISCHE UNIVERSITÄT  
CHEMNITZ

## Fakultät für Elektrotechnik und Informationstechnik

Professur für Mess- und Sensortechnik

### Diplomarbeit

Eik Arnold

Thema: Untersuchung der Implementierung und Programmierung von  
USB-Schnittstellen für die Übertragung von Daten und die Steuerung von  
Messaufbauten

Abkürzungsverzeichnis .....	3
1 Einleitung .....	4
1.1 Motivation und Stand der Technik .....	4
1.2 Geschichte des USB .....	4
1.3 USB und die Schnittstelle IEEE 1394 (FireWire) .....	5
2 USB im Überblick .....	6
2.1 PC-Anforderungen .....	6
2.2 USB aus der Sicht des Entwicklers .....	6
2.3 Systemgeschwindigkeit .....	7
2.4 Kosten .....	8
2.5 Grundlegende Eigenschaften und Begriffe des USB .....	9
2.5.1 Komponenten des USB .....	9
2.5.2 Architektur des USB .....	10
2.5.3 Hot-Plug-and-Play .....	11
2.5.4 Vier verschiedene Transferarten .....	11
2.5.5 Kabel und Stecker .....	11
3 Elektrisches Interface und Low-Level-Protokoll von USB 1.1 .....	13
3.1 Buszustände, Pegel und Taktzeiten .....	13
3.2 Connect- und Disconnect-Erkennung .....	15
3.3 Elektrisches Interface .....	16
3.4 Low-Level-Datenkodierung .....	17
3.4.1 Synchronisationsfeld (Sync-Field) .....	18
3.4.2 Genauigkeit der Taktung .....	18
3.5 High-Speed-Interface bei USB 2.0 .....	19
4 Transferarten .....	21
4.1 Pipe- und Endpoint-Konzept .....	21
4.2 Control-Transfer .....	22
4.3 Interrupt-Transfer .....	23
4.4 Isochronous-Transfer .....	24
4.5 Bulk-Transfer .....	25
4.6 Zusammenfassung und Bandbreitenabschätzung .....	26
5 Framework und USB-Pakete .....	27
5.1 Bausteine der Übertragung für USB 1.1 .....	27
5.2 Zusätzliche Pakete für USB 2.0 .....	29
5.3 Framework .....	30
5.4 Fehlerbehandlung .....	34
5.4.1 PID-Fehler .....	34
5.4.2 CRC-Fehler .....	35
5.4.3 Bit-Stuff-Fehler .....	36
5.4.4 Time-Out-Fehler .....	36
6 Standarddescriptoren .....	36
6.1 USB-Konzept der Descriptoren .....	36
6.2 Decive-Descriptor .....	38
6.3 Configuration-Descriptor .....	38
6.4 Interface-Descriptor .....	39
6.5 Endpoint-Descriptor .....	40
6.6 String-Descriptor .....	42
6.7 Device-Qualifier-Descriptor .....	42
6.8 Other-Speed-Configuration-Descriptor .....	43
7 Standard-Device-Requests .....	43
7.1 GetStatus .....	45

7.2 SetFeature und GetFeature .....	45
7.3 SetAddress.....	45
7.4 GetDescriptor .....	45
7.5 SetDescriptor.....	46
7.6 GetConfiguration.....	46
7.7 SetConfiguration .....	46
7.8 GetInterface und SetInterface.....	46
7.9 SynchFrame.....	46
8 Enumeration .....	47
8.1 Enummerierung von Hubs .....	48
8.2 Entfernen eines Gerätes.....	48
9 USB-Host-Controller .....	48
9.1 Universal-Host-Controller-Interface (UHCI).....	49
9.2 Open-Host-Controller-Interface (OHCI).....	50
9.3 Enhanced-Host-Controller .....	51
10 USB-Klassen .....	51
11 Treiber .....	52
11.1 WDM – Win32-Driver-Model .....	52
11.2 USB-Treiber .....	53
11.3 Treiber unter Linux .....	54
12 Schaltkreisauswahl für USB-Geräte.....	55
12.1 Cypress CY7C63000, CY7C63001A.....	56
12.2 FDTI FT232BM, FT245BM .....	56
12.3 Philips Semiconductors PDIUSBD11, PDIUSBD12.....	56
12.4 Microchip PIC16C765, PIC 16C745 .....	57
12.5 Dallas Semiconductors DS2490.....	57
12.6 Intel und Cypress 8x930A, 8x931A .....	57
12.7 Cypress EZ-USB (AN2131SC/QC).....	57
12.8 FX-Serie; CY7C64603, CY7C64613.....	58
12.9 FX2-Serie .....	58
13 Realisierung einer Schrittmotoransteuerung mit USB .....	58
13.1 Cypress AN2131SC/QC.....	58
13.1.1 USB 1.1-fähige SIE und 8051 kompatibler CPU-Kern .....	59
13.1.2 Speicher des EZ-USB.....	59
13.1.3 I/O-Ports .....	59
13.1.4 I <sup>2</sup> C-Bus .....	59
13.1.5 Interrupts .....	60
13.1.6 USB-Kommunikation.....	60
13.1.7 Software .....	61
13.1.8 Abschätzung der Geschwindigkeit.....	61
13.2 Aufbau der Testplatten des Mikrocontrollers.....	63
13.3 Schrittmotortreiberschaltkreis IMT901 .....	64
13.4 Ansteuerung des Fräsmotors .....	64
13.5 Handrad .....	65
13.6 Software der Steuerung .....	65
13.7 Aufbau der Maschine .....	67
14 Zusätzlicher Handlungsbedarf.....	67
15 Zusammenfassung.....	68
Literaturverzeichnis.....	70

## Abkürzungsverzeichnis

- AWG American Wire Gauge; Aderbezeichnung in Kabeln und Leitungen
- CPU Central Processing Unit ;Rechenkern eines Controllers oder Prozessors
- CRC Cyclic Redundancy Checking; Prüfsummenverfahren für die Fehlererkennung
- DDK Driver Development Kit; Software-Paket für die Treiberentwicklung
- DMA Direct Memory Access; Verfahren für den Speicher- und Datenzugriff
- EEPROM Electrically Erasable Programmable Read Only Memory; Speichervariante
- EHCI Enhanced Host Control Interface; USB 2.0 Host-Controller
- EOP End of Package; Paket das beim USB das Ende einer Transaktion abgibt
- EP0 Endpoint 0; Dieser Endpoint wird in der USB-Kommunikation verwendet
- FIFO First in First out; Speichervariante, adressloser Speicher
- GPIB General Purpose Interface Bus; Bussystem, das in der Messtechnik angewendet wird
- GPIF General Programmable Interface; Freiprogrammierbare Schnittstelle bei Mikrocontrollern
- HID Human Interface Device; Gerät das als Mensch-Maschine-Schnittstelle dient
- HPIB Hewlett Packard-Interface-Bus; Vorgänger von GPIB
- I<sup>2</sup>C Serieller Bus zur Kommunikation zwischen Schaltkreisen
- IEEE Institute of Electrical and Electronics Engineers; Amerikanische Organisation
- NRZI Non Return to Zero Inverted; Kodierungsverfahren
- OHCI Open Host Control Interface; Eine Variante von USB 1.1-Controllern
- OTP One Time Programmable; Einmal programmierbarer Speicher
- PCI Bussystem in Computern für Einsteckkarten
- PIC Mikrocontrollerfamilie des Herstellers Microchip
- PID Package Identifier; Identifizierungsnummer von USB-Paketen
- PLL Phased Locked Loop; Phasenregelkreis
- ppm parts per million; Maßeinheit
- RAM Random Access Memory; Speicher in Mikrocontrollern und Computern
- ROM Read Only Memory; Speicher, aus dem man nur lesen kann
- SE0 Single Ended Zero; Buszustand auf dem USB
- SDK Software Development Kit; Programmierumgebung von Microsoft
- SIE Serial Interface Engine; Chip oder Teil eines Chips, der die USB-Kommunikation durchführt
- SPI Bussystem bei Mikrocontrollern
- TIA Telecommunications Industry Association
- UCS-2 Kodierungsart des Unicode-Zeichensatzes
- UHCI Universal Host Control Interface; Eine Variante von USB 1.1-Controllern
- USB Universal Serial Bus; Computerschnittstelle

# 1 Einleitung

## 1.1 Motivation und Stand der Technik

Der Computermarkt ist im Bereich der Prozessoren, Speicher und Bustakte einer rasanten Entwicklung unterworfen. Wie jedoch soll ein Anschluss externer Peripheriegeräte ermöglicht werden, wenn Schnittstellen nicht mit diesem enormen Wachstum ausgebaut und erweitert werden? Die klassischen Standardschnittstellen RS 232 für die serielle und die Drucker-schnittstelle für parallele Kommunikation sind in ihren Grundeigenschaften seit ihrer Einführung ca. 1980 nahezu unverändert geblieben. Die Möglichkeit, Daten universell mit hohen Datenraten zwischen Computer und Peripheriegerät auszutauschen, ist durch die veralteten Schnittstellen nicht mehr gegeben. Häufig können externe Geräte dadurch nur begrenzt genutzt werden.

Im Bereich der Mess- und Sensortechnik ist die Verwendung eines Messrechners zur Messwertaufnahme und Datenverarbeitung zu einem Teil von Messanordnungen geworden, der in der heutigen Zeit unersetzbar ist. Dieser zeitgemäße Trend, vielseitige und hochgenaue Messgeräte und Sensoren zu entwickeln, ist mit einer Erhöhung des Datenvolumens und der Datenrate verbunden. Um leistungsfähige Sensoren und Messapparate an den Computer anschließen zu können, sind schnelle und universelle Datenschnittstellen erforderlich.

Seit 1998 ist an Computern die USB-Schnittstelle (USB – Universal-Serial-Bus) zu finden. Diese schnelle und vielseitig einsetzbare Verbindung zwischen Rechner und externen Geräten hat sich in den letzten Jahren auf dem Computermarkt erfolgreich etabliert.

Messtechnische Anwendungen der Professur für Mess- und Sensortechnik der Fakultät Elektrotechnik/Informationstechnik (TU Chemnitz) sollen zukünftig mit USB-Schnittstellen ausgestattet werden, um einen schnellen Datentransport vom oder zum Computer gewährleisten zu können. Mit dieser Diplomarbeit sollen Grundlagen für das Verständnis der USB-Kommunikation bereitgestellt und Beispiele und Möglichkeiten der Nutzung aufgezeigt werden.

## 1.2 Geschichte des USB

Die unzureichenden Voraussetzungen, Computer mit Peripheriegeräten zu verbinden, machten die Entwicklung von USB erforderlich. Eine Neuentwicklung ist dem bisher Verwendeten vorzuziehen, wenn der Nutzen der Verbesserung höher ist als der Aufwand für eine Änderung. Das Ergebnis ist eine multifunktionale Schnittstelle, welche die vorhandenen Schnittstellen zu Standard- und anwenderspezifischen Peripheriegeräten mit niedrigen bis hohen Geschwindigkeiten an beliebigen Computern ersetzen kann.

Bis dahin wurden Schnittstellen von einzelnen Firmen entwickelt. Hewlett Packard entwickelte den HP-Schnittstellenbus (HPIB), der für Laborausrüstungen unter dem Namen GPIB (Universal-Schnittstellenbus) bekannt wurde. Ein anderes Beispiel ist die Drucker-schnittstelle des Unternehmens Centronics Data Computer Corporation, die bis heute als Centronics-Schnittstelle bekannt ist. Die Unternehmen können Lizenzen für die Nutzung von Schnittstelle und zugehörigen Peripheriegeräten erheben. Sie können die Schnittstellen jederzeit ohne Ankündigung verändern. Aus diesen Gründen würde keine Firma mit ihren Produkten von der Schnittstelle eines Wettbewerbers abhängig sein wollen.

Deshalb sind neuere Schnittstellen meist das Ergebnis der Zusammenarbeit von Firmen aus den unterschiedlichsten Computerbranchen. In manchen Fällen fördert eine Organisation wie IEEE (Institute of Electrical and Electronics Engineers) oder die TIA (Telecommunications

Industry Association) Komitees zur Entwicklung von Spezifikationen und veröffentlicht die Ergebnisse. Viele der älteren Herstellerstandards wurden jedoch von solchen Organisationen übernommen. Der IEEE 1248 ging aus dem Standard der Centronics-Schnittstelle hervor und GPIB war die Grundlage für IEEE 488.

Es gibt mehrere Beispiele dafür, dass die Entwickler einer neuen Schnittstelle eine Organisation gründen, die den Standard veröffentlicht und weitere Fragen der Kompatibilität und der Weiterentwicklung behandelt. Diese Variante wurde auch beim USB gewählt. Die Urheberrechte der USB 1.1 Spezifikation sind unter vier Hardware- bzw. Softwareunternehmen aufgeteilt: Compaq, Intel, Microsoft und NEC. Diese stellen kostenlos ihre erarbeitete Spezifikation auf der Webseite des USB-Implementers-Forum ([www.usb.org](http://www.usb.org)) für Entwickler und andere Endanwender bereit.

Die USB-Spezifikation 1.0 wurde 1996 als Ergebnis einer langjährigen Entwicklungsarbeit und einigen Vorabversionen veröffentlicht. Es folgte 1998 die Version 1.1, die einige Schwachstellen der Vorgängerversion behob und weitere Komponenten neu hinzufügte.

Die wichtige Betriebssystemunterstützung des USB wurde erstmals mit der OEM Service Release 2 von Windows 95 verfügbar. Die USB-Unterstützung wies in dieser Phase noch sehr viele Mängel auf. Außerdem waren nur wenige USB-Peripheriegeräte auf dem Markt zu finden, so dass nur ein geringer Einsatz möglich war. Mit der Veröffentlichung von Windows 98 und der zweiten Version Windows 98 SE wurde die Betriebssystemunterstützung deutlich verbessert. Gleichzeitig begannen die Hersteller, ihre Peripheriegeräte mit USB auszurüsten, so dass die Vorteile der USB-Schnittstelle schnell bekannt wurden. Nach kurzer Zeit war USB Bestandteil jedes Computers.

Mit der Entwicklung von USB 2.0 gelang den Herstellern eine wesentliche Verbesserung der Übertragungsgeschwindigkeit. Die Datenraten wurden um den Faktor 40 gesteigert, was eine maximale Übertragungsrate von 480 MBit/s zur Folge hat. Dieses Merkmal ist besonders attraktiv für Hersteller von Druckern, Scannern und externen Massenspeichern. Der USB 2.0 ist vollständig abwärtskompatibel zum USB 1.1. Alle Geräte nutzen die gleichen Steckverbinder und Kabel. Um Geräte mit diesem schnellen Standard anschließen zu können, ist ein Host und ein Hub notwendig, der nach USB 2.0 kompatibel ist. Diese unterstützen dann auch die langsameren Standards 1.0 und 1.1. Host und Hub werden dabei vom inneren Aufbau komplizierter. Es entfällt dann aber die Notwendigkeit verschiedener Geräte für die unterschiedlichen Standards. Zu den Mitgliedern der USB 2.0-Organisation gehören neben den vier Unternehmen des USB 1.1 auch Hewlett Packard, Philips und Lucent. Die Entwicklungsspezifikation wurde 1999 an das USB-Implementers-Forum übergeben. Eine Release-Version folgte dann 2000. Im Jahr 2001 wurden erste Peripheriegeräte auf dem Markt angeboten.

### **1.3 USB und die Schnittstelle IEEE 1394 (FireWire)**

Eine weitere wichtige Schnittstelle, die etwa zur gleichen Zeit entwickelt wurde, ist die IEEE 1394. Das Unternehmen Apple setzte mit der Realisierung dieses Standards den Namen FireWire durch. Geplant war, dass IEEE 1394 die Anbindung schneller Peripheriegeräte unterstützt und USB für Anwendungen mit niedriger bis mittlerer Geschwindigkeit genutzt wird. Durch Probleme bei der Lizenzvergabe von IEEE 1394 wurde durch Intel der Ausbau des USB weiter gefördert, so dass die Version 2.0 mit IEEE 1394 in Konkurrenz steht. Beide Systeme können kleinere Geräte über den Bus mit Strom versorgen. Im Gegensatz zum USB wird hier eine Peer-To-Peer-Struktur verwendet. Diese eignet sich sehr gut für Videoverbindungen und andere schnelle externe Geräte. USB unterstützt hingegen verschiedene Geschwindigkeitsstufen. Beim USB steuert der Host die Kommunikation aller angeschlossenen Geräte. Eine Kommunikation zwischen zwei Geräten erfolgt immer über den

Host, wobei mit IEEE 1394 die Geräte auch direkt miteinander kommunizieren können. Dadurch ist es möglich, mit mehreren Empfängern gleichzeitig Mitteilungen auszutauschen. Nachteil dabei ist, dass die Elektronik der Peripheriegeräte aufwändiger und deswegen teurer ist.

Datenraten des FireWire von 400 MBit/s sind mit der Einführung von USB 2.0 mit 480 MBit/s auch erreichbar. Durch die Veröffentlichung von IEEE 1394.b, die mit einer Datenrate von bis zu 3,2 GBit/s aufwartet, ist die Datenübertragungsrate von USB 2.0 übertroffen worden.

## 2 USB im Überblick

### 2.1 PC-Anforderungen

Um USB-Peripheriegeräte nutzen zu können, ist ein USB-Host-Controller und ein Betriebssystem mit USB-Unterstützung notwendig.

Fast jeder neue Computer enthält heute mindestens zwei USB 2.0 Schnittstellen. Sollten keine USB-Anschlüsse vorhanden sein, kann man mit einer zusätzlichen PCI-Einsteckkarte zwei, vier oder sechs Anschlüsse nachrüsten.

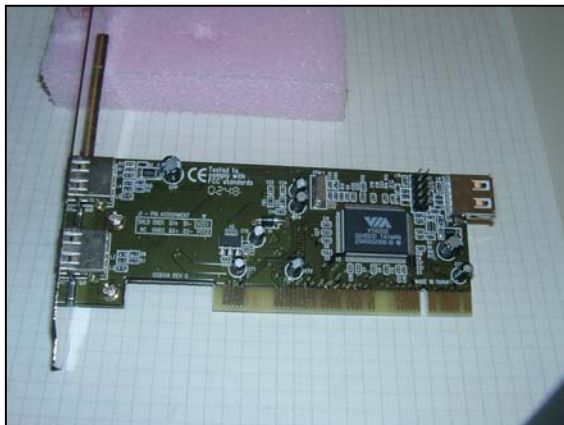


Abb. 1,2 – PCI-Einsteckkarte und PCMCIA-Karte (rechts) für die Nachrüstung von USB 2.0-Schnittstellen

Die zweite Grundlage stellt das Betriebssystem dar. Da Windows 95 zwar USB unterstützt, aber diesbezüglich noch viele Schwachstellen hat, sollte mindestens Windows 98 SE verwendet werden. Windows NT besitzt noch keine Unterstützung. Der Nachfolger Windows 2000 hingegen bietet volle USB-Unterstützung. Sämtliche später entwickelten und veröffentlichten Windows-Versionen wie Windows-Millennium-Edition und Windows XP (Home und Professional) haben in ihrer Standardinstallation volle USB-Unterstützung.

Linux unterstützt USB ab der Kernelversion 2.2.18. Um den aktuellen Entwicklungsstand nutzen zu können, sollte mindestens mit der Kernelversion 2.4 gearbeitet werden.

### 2.2 USB aus der Sicht des Entwicklers

Im klassischen Ein-/Ausgabe-Konzept wurden Peripheriegeräte des Computers immer in den Adressraum der CPU eingeblendet und entsprechenden Interrupt-Leitungen zugeordnet.

Teilweise erhielten die Geräte auch DMA-Kanäle. Die Ressourcen des Systems wurden nach und nach von IBM und anderen Herstellern den verschiedenen Anwendungen zugeteilt.

Damit etablierten sich I/O-Adressräume, Interrupts und DMA-Kanäle für die jeweils spezielle Anwendung. Softwareentwickler konnten sich auf diese, zwischenzeitlich zu festen Standards gewordenen Zuordnungen, beziehen. Durch die steigende Anzahl von Peripheriegeräten wurde die Anzahl freier Systemressourcen geringer, so dass ein Einbau oder eine Konfiguration eines zusätzlichen Gerätes immer öfter erhebliche Probleme mit sich brachte und damit großen Aufwand verursachte.

Interrupt-Leitungen bilden beim klassischen PC mit ISA-Bus-Struktur immer die problematischste Ressource, da sich einige Geräte einen Interrupt teilen, andere IRQ-Leitungen dagegen beliebig vergeben werden können.

USB bietet neben den Vorteilen für Endanwender auch erhebliche Vorteile für Hardware-, Firmware- und Softwareentwickler.

## 2.3 Systemgeschwindigkeit

Ein neuer Standard muss einen deutlich höheren Datendurchsatz ermöglichen als seine Vorgängersysteme RS 232 und Parallelport. Der inzwischen übliche USB 2.0 Standard bietet Übertragungsraten bis 480 MBit/s und entspricht damit den derzeitigen Anforderungen.

In der folgenden Tabelle werden die verschiedenen Anwendungen in Geschwindigkeitsklassen des USB aufgeteilt.

<b>Geschwindigkeitsklasse</b>	<b>Applikation</b>	<b>Genutzte Eigenschaften</b>
Low-Speed (USB 1.0) 10..100 Kbit/s	Interaktive Eingabegeräte (Tastatur, Maus, Joystick, Virtual Reality, etc.)	<ul style="list-style-type: none"> <li>• Sehr preisgünstig</li> <li>• Hot-plug /unplug</li> <li>• Einfache Benutzbarkeit</li> <li>• Mehrere Komponenten gleichzeitig am USB</li> <li>• Hochflexible Kabel</li> </ul>
Full-Speed (USB 1.1) 500..10.000 Kbit/s	Telefonie und Audio (ISDN, Modems, Digital Audio, Scanner, Drucker, etc.)	<ul style="list-style-type: none"> <li>• Preisgünstig</li> <li>• Einfache Benutzung</li> <li>• Garantierte Latenz und Bandbreite</li> <li>• Anschluss mehrerer Komponenten</li> </ul>
High-Speed (USB 2.0) 25..450 Mbit/s	Video, Externe Disk- Laufwerke, LAN (TV und Videoschnittgeräte, LAN und WLAN-Adapter, Festplatten, Memory-Sticks, etc.)	<ul style="list-style-type: none"> <li>• Hohe Bandbreite</li> <li>• Garantierte Latenz</li> <li>• Einfache Handhabung komplexer Geräte</li> </ul>

Tabelle 1 – Einordnung der Peripheriegeräte in die Geschwindigkeitsklassen des USB



Ebenso kann der USB im Vergleich zu anderen Schnittstellen charakterisiert werden.

<b>Schnittstelle</b>	<b>Übertragungsformat</b>	<b>Max. Anzahl der Geräte</b>	<b>Maximallänge in Meter</b>	<b>Maximalgeschwindigkeit in Bit/s</b>	<b>Typische Verwendung</b>
USB	asynchron, seriell, differenziell	127	5 (oder bis 30m mit 5 Hubs)	1,5 M, 12 M, 480 M	Komplettes Spektrum externer Geräte
RS 232	asynchron, seriell	2	15-30	Typ. ca.20 k (bis 115 k )	Maus, Modem, Messgeräte
RS 485	asynchron, seriell, differenziell	32 (durch Repeater erhöhbar)	1200	10 M	Datenerfassungs- und Steuerungssysteme
IrDA	asynchron, seriell, infrarot	2	1,8	115 k	Handys, Palm, Pocket-PC's
Mircowire	synchron, seriell	8	3	2 M	Mikrocontroller-kommunikation
SPI	synchron, seriell	8	3	2,1 M	Mikrocontroller-kommunikation
I <sup>2</sup> C	synchron, seriell	40	5,5	400 k	Mikrocontroller-kommunikation
IEEE 1394 (FireWire)	seriell, differenziell	64	4,5	400 M	Video, Ext. Laufwerke
IEEE 488 (GPIB)	parallel	15	18	8 M	Messgeräte
MIDI	serielle Stromschleife	2	15	31,5 k	Musik, Show-Control
Ethernet	seriell, differenziell (Twisted-Pair)	1024	500	10 M/ 100 M/ 1G	Netzwerk
Parallel-Port	parallel	2	3-10	8 M	Drucker, Scanner

Tabelle 2 – Vergleich verbreiteter Computerschnittstellen

## 2.4 Kosten

Die Kosten konventioneller Peripheriekomponenten werden im Wesentlichen durch die relativ aufwändigen Stecker und Kabel beeinflusst. Sind darüber hinaus spezielle Einsteckkarten notwendig, steigen die Kosten noch einmal. Der USB ermöglicht die Reduzierung der Gesamtbetriebskosten für moderne PC-Systeme (auch als Total-Cost-of-Ownership bezeichnet).

USB liefert einen Low-Cost-Ansatz für den Anschluss von Peripheriegeräten. Mit einem kostenoptimierten Design wurde bereits bei den Steckern und Kabeln begonnen. Die geringe Anzahl von Anschlusspins und die kompakten Abmessungen ermöglichen eine effektive Massenproduktion. Low-Speed-Komponenten benötigen auf Grund des geringeren Datendurchsatzes auch nur ein einfacheres Kabel, das direkt mit dem Gerät verbunden wird.

Eine große Anzahl verfügbarer Mikrocontroller mit USB-Interface bzw. besondere USB-Interface-Bausteine erlauben eine kostengünstige Realisierung von USB-Peripheriegeräten. Mit USB ist es möglich, kompakte und preiswerte Gehäuse einzusetzen. Kleinere Geräte können durch den Bus mit Spannung versorgt werden und benötigen deshalb kein Netzteil. In der Einführungsphase von USB 2.0 entstanden durch den hohen Entwicklungsaufwand von Host, Hub und kompatiblen Geräten erhebliche Kosten, die im Anschaffungspreis spürbar wurden. Nachdem USB 2.0-Komponenten inzwischen serienmäßig produziert werden, haben sich diese anfänglichen Preisverschiebungen wieder normalisiert.

## **2.5 Grundlegende Eigenschaften und Begriffe des USB**

### **2.5.1 Komponenten des USB**

Die physikalischen Komponenten des USB bezeichnet man als Host, Hub und Gerät, die mit Kabeln und Steckverbindern elektrisch verbunden sind. In der Fachliteratur findet man anstelle des Begriffes Gerät für eine Anwendung auch die Bezeichnung Function.

Der Host ist ein PC oder Notebook, der zwei Komponenten enthält: einen Host-Controller und einen Root-Hub. Diese arbeiten immer unmittelbar zusammen, damit das Betriebssystem mit den Geräten am Bus kommunizieren kann. Der Host-Controller formatiert die Daten zur Übertragung auf den Bus und konvertiert empfangene Signale wieder in eine für das Betriebssystem verständliche Form zurück. Er baut eine verkettete Liste von Datenstrukturen auf, welche die Transaktionen darstellt. Dabei legt er deren zeitliche Abfolge (Schedule) fest, die innerhalb eines Zeitrahmens (Frame) auf dem Bus ausgeführt wird. Ebenso ist die Adressverwaltung und die Enummeration der angeschlossenen Geräte eine wichtige Aufgabe des Host-Controllers in Verbindung mit dem Betriebssystem. Der Root-Hub bildet mit einem bis sechs Steckverbindern das erste Glied der USB-Struktur. An ihn können Geräte oder weitere Hubs angeschlossen werden. Dabei erfüllt er folgende Aufgaben:

- Steuerung der Stromzufuhr für die USB-Ports
- Freischalten und Sperren der Ports
- Erkennen, ob ein USB-Gerät an einen der Ports angeschlossen oder abgezogen wird
- Verwaltung des Portstatus

Jede USB-Struktur ist eine Single-Master-Bus Struktur. Das bedeutet, dass im Bus nur ein Master zu finden ist. Alle anderen Geräte sind als Hubs oder Functions zu bezeichnen. Eine Kommunikation zwischen zwei Geräten ist nur über den Host möglich. Host-Controller und Root-Hub sind als Ein-Chip-Lösung auf Motherboards oder PCI-Erweiterungskarten zu finden. Oftmals wird heute ein Chip auf den Motherboards eingesetzt, der einerseits die Steuerung des PCI-Busses übernimmt und gleichzeitig einen USB-Host-Controller nebst Root-Hub enthält.

Die USB 1.1-Spezifikation definiert zwei verschiedene Host-Controller-Designs, den Open-Host-Controller (OHC) und den Universal-Host-Controller (UHC). Beide Varianten erfüllen die Grundfunktionen der USB-Kommunikation. Auf Details wird in Kapitel 9 eingegangen.

USB 2.0 erweitert die Host-Struktur um den Enhanced-Host-Controller (EHC), der für den High-Speed-Transfer zuständig ist und zusätzlich zum UHC bzw. OHC parallel geschaltet wird. Somit enthält jeder USB 2.0 Host-Controller die kompletten Funktionalitäten eines 1.1-Controllers. Die weitere Kommunikation des EHC mit OHC und UHC wird ebenfalls in Kapitel 9 erläutert.

## 2.5.2 Architektur des USB

Die physikalische Struktur des USB ist als kaskadierte Stern-Topologie oder Baumstruktur zu bezeichnen. Die Geräte bilden die Blattknoten und Hubs werden zur Verzweigung eingesetzt. Die folgenden beiden Darstellungen sollen zeigen, dass die beiden oben genannten Strukturbezeichnungen identisch sind.

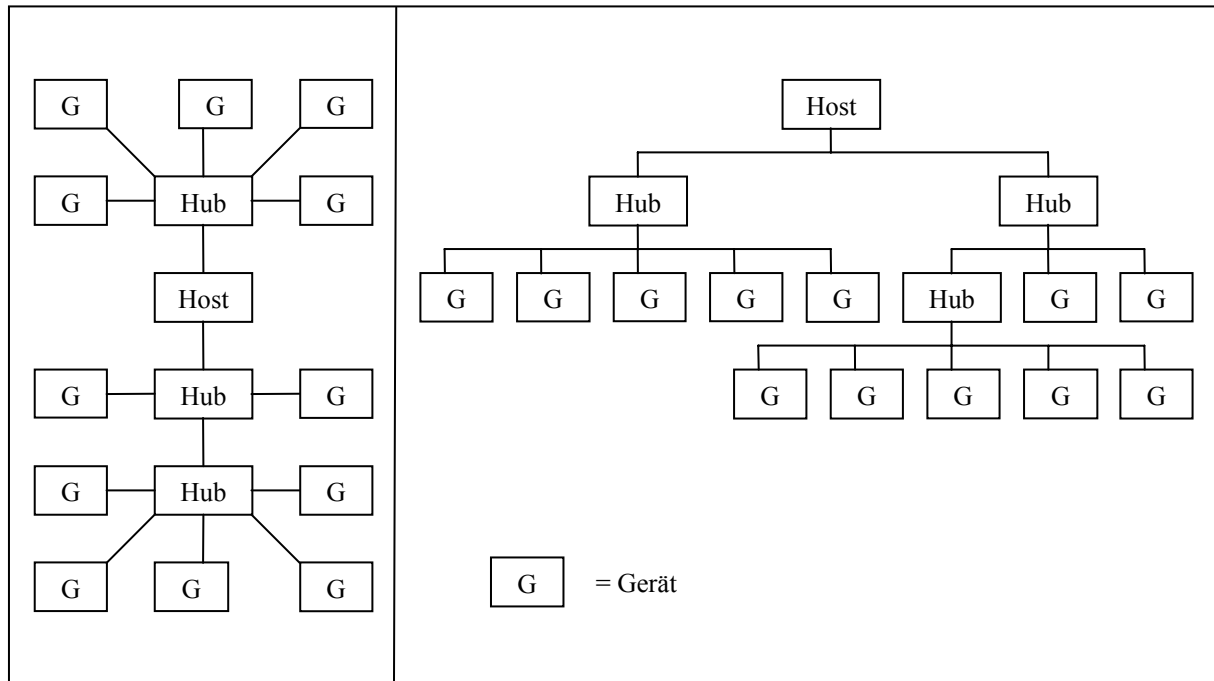


Abb. 3 – Physische Struktur, dargestellt in beiden Strukturvarianten

Die logische computerinterne Struktur ist als reine Sternstruktur zu betrachten. Alle Endgeräte erscheinen sternförmig an den Host angeschlossen.

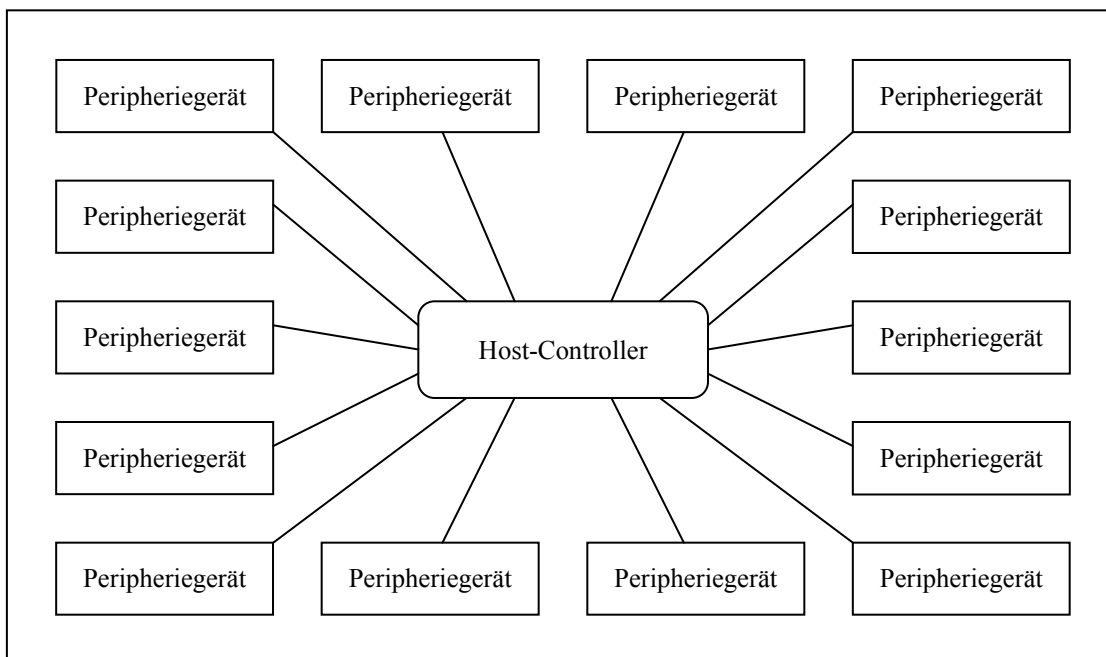


Abb. 4 – Logische Struktur des USB

### 2.5.3 Hot-Plug-and-Play

Das Anstecken eines Gerätes im laufenden Betrieb ist möglich, ohne den Rechner neu starten zu müssen. Es wird durch den USB automatisch erkannt und konfiguriert. Der entsprechende Treiber wird vom Betriebssystem geladen oder beim ersten Anstecken automatisch installiert. Wenn das Gerät vom Bus entfernt wird, so deaktiviert das Betriebssystem den entsprechenden Treiber. Zur Erkennung von Geräten am Bus sind verschiedene Buszustände definiert.

### 2.5.4 Vier verschiedene Transferarten

Die USB-Spezifikation stellt vier verschiedene Transferarten zur Verfügung, um die unterschiedlichen Geräte- bzw. Applikationscharakteristiken zu unterstützen.

- Control-Transfer wird benutzt, um spezielle Anfragen (Request) an ein USB-Gerät zu senden. Dies geschieht insbesondere in der Konfigurationsphase. Diese Transferart wird hauptsächlich für die Übertragung von Initialisierungskommandos vom Host zum Gerät verwendet. Typischerweise erfolgt danach eine Umschaltung in einen anderen Transfermodus.
- Interrupt-Transfer wird für Geräte verwendet, die in einer klassischen PC-Umgebung Interrupts auslösen würden. Da aber USB keine Hardware-Interrupts unterstützt, müssen diese Geräte im Polling-Verfahren in äquidistanten Zeitabständen abgefragt werden. Typisches Beispiel für diese Transferart ist die periodische Abfrage der Tastatur, die keinen Interrupt für das Drücken einer Taste auslösen kann.
- Bulk-Transfer dient der Übertragung großer Datenmengen, die nicht periodisch sind und keine Echtzeitbedingungen stellen. Ein Beispiel dafür sind Druckerdaten. In diesem Modus ist die Geschwindigkeit der ausschlaggebende Faktor.
- Isochronous-Transfer ist für Daten vorgesehen, bei denen eine Latenzzeit der Übertragung im Vordergrund steht. Isochronous-Daten sind beispielsweise Audiodaten, die von einem Gerät zum PC oder umgekehrt gesendet werden. Diese Daten erfordern eine zeitliche Synchronität und eine hohe Kontinuität des Datenstroms. Einzelne Bitfehler treten im Audibereich als kurze Aussetzer auf, die kaum hörbar sind und keine Beeinträchtigung der Qualität darstellen. Dieser Transfermodus verzichtet zu Gunsten einer zeitstabilen Übertragung auf eine Fehlererkennung, die in allen anderen Transferarten vorhanden ist.

### 2.5.5 Kabel und Stecker

#### 2.5.5.1 Aufbau des Kabels

Beim USB wird ein einheitliches Stecker- und Kabelsystem verwendet. Dieses Konzept trägt entscheidend zum Erfolg von USB bei. Vermieden wurde hier, im Gegensatz zu Systemen wie SCSI und RS 232, der Einsatz von dicken Kabeln und verschiedenen Steckern.

Für USB 2.0 können USB 1.1-kompatible Kabel verwendet werden. Die elektrischen Anforderungen an die USB 1.1- bzw. 2.0-tauglichen Kabel sind sehr hoch, so dass nicht jede beliebige Kabelsorte benutzt werden kann. Aus diesem Grund ist vom Selbstbau von Kabeln abzuraten, da USB-Kabel in verschiedenen Längen preiswert angeboten werden. Die USB-Kabel enthalten immer vier elektrische Leitungen: Masse- und Powerleitung für die Stromversorgung des USB-Gerätes über den Bus und die Datenleitungen D+ und D-.

Leitung	Pin-Nummer am Stecker	Adern-Farbe
VCC (+5 V)	1	Rot
D–	2	Weiß
D+	3	Grün
GND	4	Schwarz

Tabelle 3 – Steckerbelegung und Adernfarben

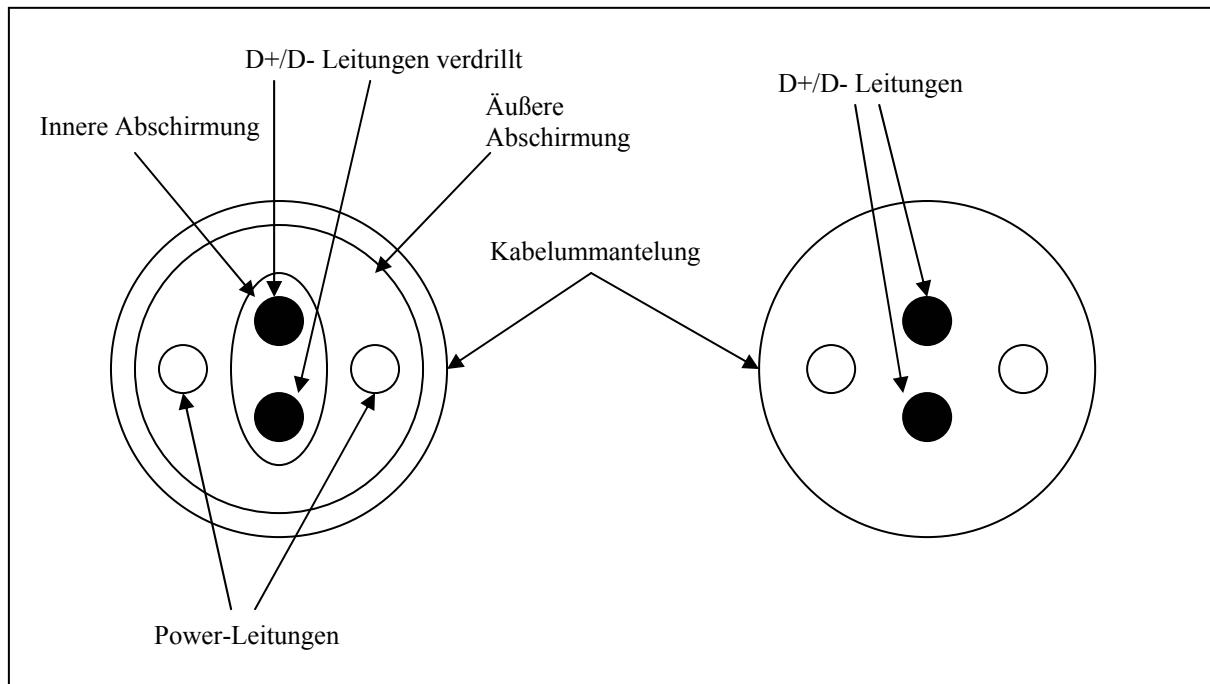


Abb. 5 – Aufbau von Full- und High-Speed-Kabeln (links) und Low-Speed-Kabeln (rechts)

Der physikalische Aufbau des Kabels unterscheidet sich je nach seiner Verwendung für ein Low-Speed- oder Full-/High-Speed-Gerät. In den Kabeln für High- und Full-Speed-Geräte sind die verdrehten Aderleitungen D+ und D– zusätzlich geschirmt. Das dient sowohl der Verringerung von hochfrequenter Störabstrahlung als auch der Verminderung äußerer Störeinflüsse. Die Powerleitungen sind für beide Spezifikationsvarianten nicht verdreht. Die Länge der Full- und High-Speed-Kabel ist auf 5 m begrenzt. Dies ergibt sich aus den geforderten Signallaufzeiten von 30 ns für ein Kabelsegment. Mit zunehmender Kabellänge werden auch die Aderstärken der Powerleitungen angepasst, um Spannungsverluste zu minimieren. Für die Datenleitungen wird durchgängig der Adertyp AWG 28 eingesetzt.

Adertyp	Widerstand pro Meter	Maximale Länge
AWG 28	0,232 $\Omega$ /m	0,81 m
AWG 26	0,145 $\Omega$ /m	1,31 m
AWG 24	0,091 $\Omega$ /m	2,08 m
AWG 22	0,057 $\Omega$ /m	3,33 m
AWG 20	0,036 $\Omega$ /m	5,00 m

Tabelle 4 – Kabellängen und verwendete Kabeltypen

Bei Low-Speed-Kabeln sind die Datenleitungen nicht miteinander verdreht. Deshalb ist ihre Maximallänge auf 3 m begrenzt. Um zu vermeiden, dass Low-Speed-Leitungen für den Datenverkehr anderer Geschwindigkeiten genutzt werden, sind die Low-Speed-Kabel fest an das Gerät montiert. Es existieren also nur USB-Verbindungskabel, die für alle Geschwindigkeiten geeignet sind.

### **2.5.5.2 USB-Steckverbindungen**

Für den USB sind zwei verschiedene Stecker definiert: die Stecker der A-Serie mit flachem, rechteckigem Querschnitt und die Stecker der B-Serie mit quadratischem Querschnitt. Die in den Steckern befindlichen Powerkontakte VCC und GND sind länger als die Kontakte D+ und D-, um die Betriebsspannung vor den Datenleitungen zu kontaktieren.

Jedes lösbare Kabelende in Upstream-Richtung, also das Kabelende, welches zum Host zeigt, ist ein A-Stecker. Da alle Low-Speed-Kabel in Downstream-Richtung fest mit dem Gerät verbunden sind, findet man immer einen Typ-A-Stecker an diesen Geräten. Full- und High-Speed-Geräte besitzen in Downstream-Richtung einen B-Stecker, um das Kabel mit dem Gerät zu verbinden. Das Gerät hat somit eine B-Buchse. Generell gilt, dass alle Kabel Stecker aufweisen. Alle Geräte, Hub's und Computer haben Buchsen.

Die Verwendung von zwei mechanisch unterschiedlichen Steckern soll eine einfache, aber gleichzeitig sichere Handhabung des USB garantieren. Es lassen sich weder Schleifen noch andere verbotene Verbindungen herstellen. Es sollte darauf geachtet werden, dass nur Verbindungskabel mit A-Stecker auf der einen Seite und einem B-Stecker am anderen Ende genutzt werden. Alle anderen im Fachhandel erhältlichen Adapter, Gender-Changer und Verlängerungskabel sind nicht USB-spezifiziert und können zu Beschädigungen am gesamten Bus führen. Für einige Geräte, hauptsächlich Digitalkameras, gibt es seit der Spezifikation des USB 2.0 auch einen Mini-B-Stecker. Leider haben die verschiedenen Hersteller sich eigene Steckervariationen definiert, so dass einheitliche Standards bei den Minivarianten nicht gegeben sind.

## **3 Elektrisches Interface und Low-Level-Protokoll von USB 1.1**

### **3.1 Buszustände, Pegel und Taktzeiten**

Die Datenleitungen werden bis auf einige Sonderfälle (Single-ended-Zero) differenziell getrieben. Die beiden idealen Schaltzustände der Leitungen betragen 0 V und 3,3 V. Eine differenzielle 1 liegt am Treiber vor, wenn der D+ Ausgang mindestens 2,8 V und der D- Ausgang nicht größer als 0,3 V in Relation zum Massesignal des Treibers ist. Bei einer differenziellen 0 sind die angegebenen Potenziale auf den Datenleitungen vertauscht. Der Empfänger detektiert eine 1 wenn D+ mindestens 2 V in Relation zur Masse beträgt und von D+ zu D- eine Potenzialdifferenz von mindestens 200 mV auftritt. Diese zwei Attribute für den Empfänger sind Beispielwerte. Sollte die Potenzialdifferenz höher als 200 mV sein, so kann der 2 V Mindestsignalpegel auch unterschritten werden. Diese Art von Verbindung wird „balanced line“ genannt. In beiden Fällen sorgt die Differenz zwischen den minimal übertragenen und empfangenen Spannungen dafür, dass der Empfänger den richtigen differenziellen Zustand auch dann noch erkennt, wenn die Signale leicht verfälscht sind.

Außer den differenziellen Einsen und Nullen definiert der USB-Standard zwei weitere gültige Zustände, die durch die D+ und D- -Leitungen repräsentiert werden.

Eine untätige Leitung im Idle-Zustand ähnelt den Zuständen von differenziellen Einsen und Nullen. Die Spannungen dieses Zustandes sind abhängig von der Geschwindigkeitsklasse.

Damit kann der Hub die Geschwindigkeit neu angeschlossener Geräte erkennen. Die Full- und High-Speed-Signale für den Idle-Zustand sind unterschiedlich zu denen von Low-Speed-Geräten. Für schnelle Übertragungen wird die D+-Leitung auf ein Potenzial von über 2,8 V und D– auf max. 0,8 V gelegt. Bei Low-Speed sind die Potenziale vertauscht.

Der Single-ended-Zero – Zustand (SE0) tritt ein, wenn sowohl D– als auch D+ am Empfänger weniger als 0,8 V betragen. Dieser Zustand wird für End-of-Packet-, Disconnect- und Reset-Signale genutzt. Ebenso gibt es eine Single-ended-One, die auftritt, wenn an beiden Datenleitungen am Empfänger eine Spannung über 2 V anliegt. Dieser Buszustand ist ungültig und muss unter allen Umständen vermieden werden.

Die differenziellen Nullen und Einsen stehen nicht synonym für die logischen Zustände des Datenstroms, der übertragen wird. Erläuterungen dazu erfolgen im Kapitel 3.4. Die grundlegenden Signallaufzeiten sollen durch die zwei folgenden Abbildungen charakterisiert werden.

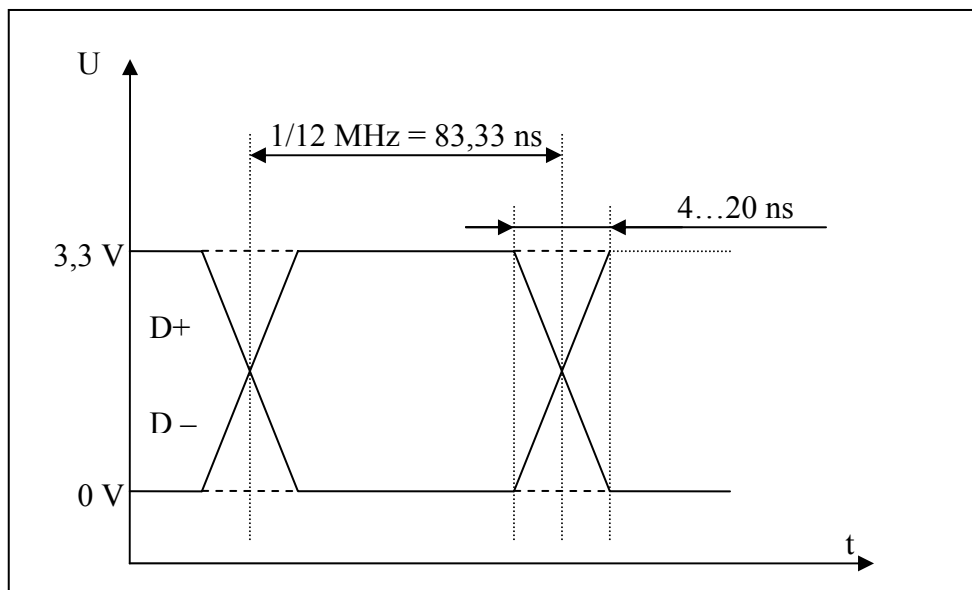


Abb. 6 – Elektrische Signale im Full-Speed-Modus

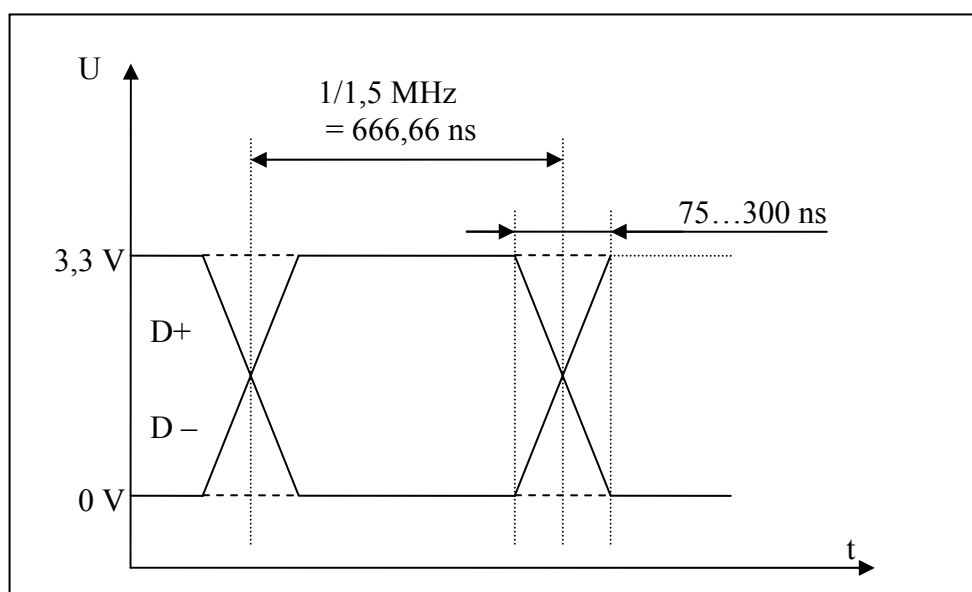


Abb. 7 – Elektrische Signale im Low-Speed-Modus

Um die Störabstrahlung der Kabel zu minimieren, müssen die Signalanstiegs- und abfallzeiten durch die elektrischen Treiber begrenzt werden. Da diese Zeiten für Full-Speed- und Low-Speed-Geräte unterschiedlich definiert sind, werden analoge Treiber mit programmierbarer Slew-Rate eingesetzt. Bei Full-Speed-Geräten darf die Anstiegs- bzw. Abfallzeit zwischen 4 ns und 20 ns und bei Low-Speed-Geräten zwischen 75 ns und 300 ns liegen.

### 3.2 Connect- und Disconnect-Erkennung

Da der USB ein echter Hot-Plug-and-Play-Bus ist, sind Mechanismen implementiert, um das Anstecken oder Abziehen eines USB-Gerätes während des laufenden Betriebes zu erkennen. Diese Connect- und Disconnect-Erkennung wird auf elektrischer Ebene realisiert.

Die beiden Datenleitungen D+ und D- sind am Downstream-Port des Hubs mit jeweils 15 k $\Omega$  an Masse gelegt. Im Gegensatz hierzu ist beim USB-Gerät auf der Upstream-Seite eine Leitung über einen 1,5 k $\Omega$  Pull-Up-Widerstand mit 3,3 V verbunden. Bei Full-Speed-Geräten ist das die D+-Leitung, bei Low-Speed-Geräten dagegen D-.

Der Hub überwacht an jedem Downstream-Port beide Datenleitungen, ohne sie mit einem Pegel zu treiben. Ist kein Gerät angeschlossen, ziehen die 15 k $\Omega$  Pull-Down-Widerstände beide Leitungen auf Low-Pegel. Wird ein Full-Speed-Gerät an diesen Downstream-Port angeschlossen, bewirkt der Pull-Up-Widerstand von 1,5 k $\Omega$  an D+ einen High-Pegel auf dieser Leitung. Der Hub erkennt diese Pegeländerung und signalisiert nach 2,5  $\mu$ s stabilem High-Pegel auf D+ ein Connect-Ereignis an den Host. Automatisch erfolgt dann die Enumeration des neu angeschlossenen Gerätes. Da der Hub unterscheidet, ob durch das neu angesteckte Gerät die D+ oder die D- Leitung auf High-Pegel liegt, besitzt er gleichzeitig die Information über die Geschwindigkeitsklasse des Gerätes.

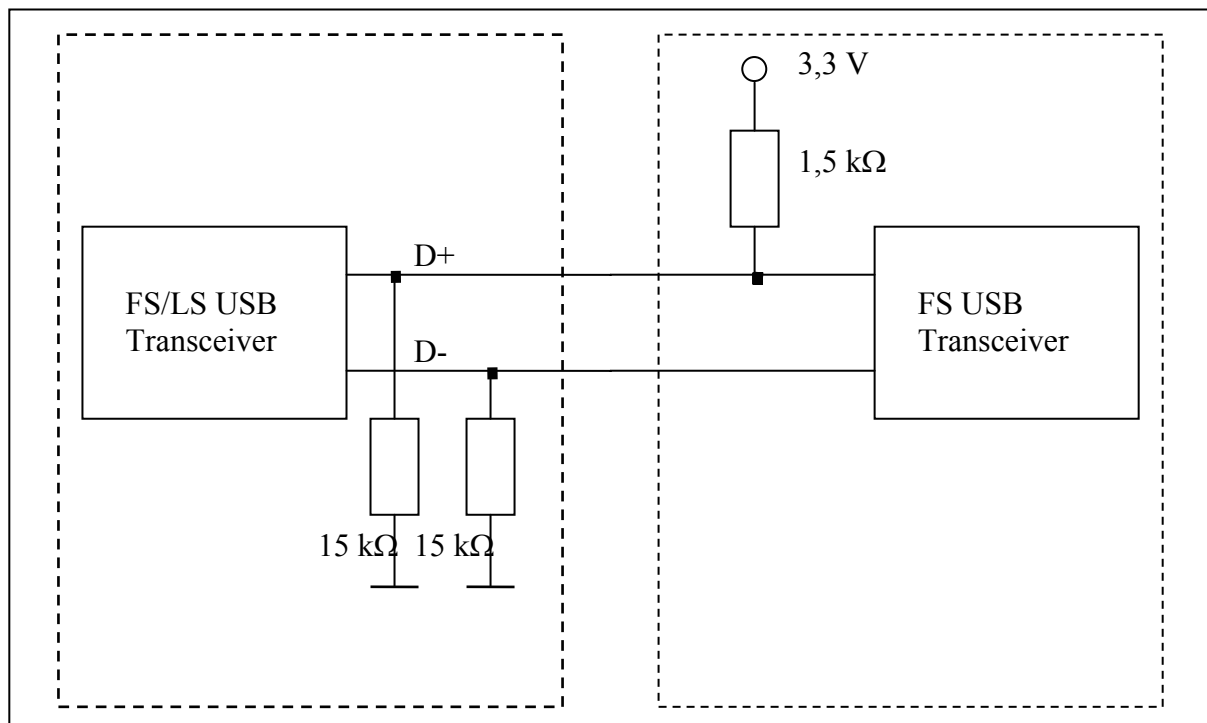


Abb. 8– Connect-Erkennung für Full-Speed-Geräte

Analog zur Connect-Erkennung wird auch ein Disconnect erkannt. Treibt der Hub kein Signal auf seinem Downstream-Port, so befindet sich eine Leitung auf Low und die andere auf High, entsprechend der unterstützten Geschwindigkeit des angesteckten Geräts. Gehen beide



Leitungen auf Low und verbleiben dort länger als 2,5  $\mu$ s, wird dies als Disconnect erkannt. Auch in diesem Fall benachrichtigt der Hub den Host-Controller, der daraufhin das abgezogene Gerät (oder auch mehrere Geräte) aus seiner Konfiguration entfernt.

### 3.3 Elektrisches Interface

Alle USB-Geräte besitzen ein elektrisches Interface, das sich in drei Komponenten einteilen lässt:

- Differentieller Empfänger
- Differentieller Treiber
- Single-Ended Empfänger

Der differentielle Empfänger dient dem Entsymmetrieren der auf den Datenleitungen empfangenen Daten. Die differentiell empfangenen Daten werden hier wieder in normale Logikpegel umgewandelt. Der Empfänger muss eine differentielle Schaltschwelle von 200 mV aufweisen, wenn sich beide Datenleitungen in einem Spannungsbereich zwischen 0,8 V und 2,5 V befinden.

Neben dem differentiellen Receiver befindet sich an jeder Leitung jeweils ein Single-Ended-Receiver. Im engeren Sinne sind dies einfache nichtinvertierte Treiber mit Schmitt-Trigger-Charakteristik und einer Schaltschwelle zwischen 0,8 V und 2,0 V. Mit Hilfe dieser Empfänger werden nichtdifferentielle Zustände auf den beiden Datenleitungen D+ und D- detektiert. Dies sind insbesondere Connect und Disconnect sowie SE0-Zustände (EOP bzw. USB-Reset).

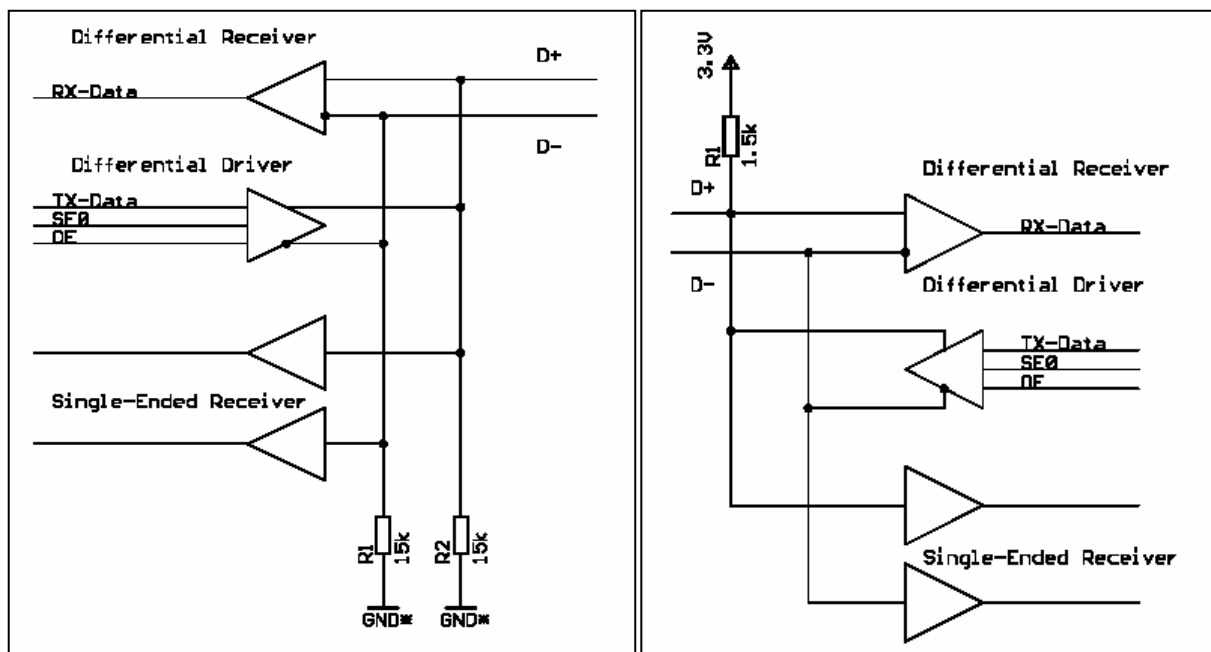


Abb. 9,10 – Elektrisches Interface eines Downstream-Port (links) und Upstream-Port (rechts)

Um die Datenleitungen zu treiben, werden analoge Treiberstufen verwendet. Nur damit können die unterschiedlichen Slew-Rates für Full- und Low-Speed verwirklicht werden.

Neben dem Symmetrieren der zu übertragenden Daten muss auch die Möglichkeit bestehen, einen SE0-Zustand zu treiben.

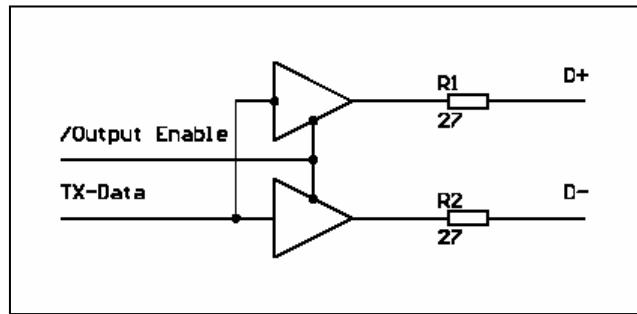


Abb. 11 – Differentieller Treiber

Die Treiber werden meist in CMOS-Technologie gefertigt. Allerdings ist damit nicht die laut USB 1.1- Spezifikation erforderliche Ausgangsimpedanz zwischen 28 und 44  $\Omega$  zu erreichen, da die Treiber niederohmiger sind. Deswegen sind in (fast) jedem Design zwei typische Reihenwiderstände (22..27  $\Omega$ ) in den Datenleitungen zu finden.

### 3.4 Low-Level-Datenkodierung

Alle USB-Daten sind kodiert. Dies dient einerseits der Erhöhung der Datensicherheit und zum anderen der Taktrückgewinnung. Dieses Kodierungsverfahren, das sogenannte NRZI (Non-Return-to-Zero-Inverted) mit Füllbits (bit-stuffing), sorgt dafür, dass der Empfänger mit dem Sender synchronisiert wird, ohne ein getrenntes Taktsignal oder Start- und Stopbits zu verwenden. Zum Zweck der Kodierung werden die Daten in einem Schieberegister serialisiert, durchlaufen einen Bitstuffer und werden anschließend durch einen NRZI-Encoder geführt. Der daraus gewonnene NRZI-Datenstrom wird durch die differentiellen Treiber symmetriert und auf die D+ und die D- -Leitung getrieben. Auf der Empfängerseite wird der gesamte Prozess umgekehrt betrieben, so dass die Nutzdaten wieder zur Verfügung stehen.

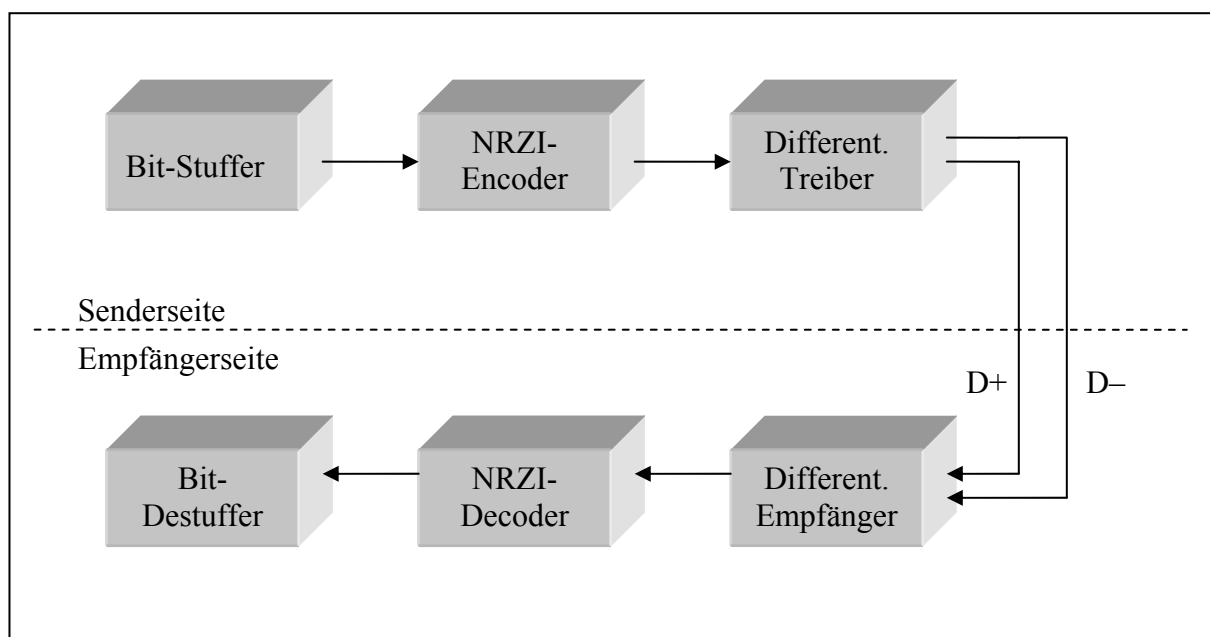


Abb. 12 – Low-Level-Datenkodierung

Bei RS 232 und allen ähnlichen asynchronen Schnittstellen haben Sender und Empfänger eigene Taktreferenzen, die aufeinander synchronisiert werden müssen. Dazu fügt man Start- und Stoppbits ein. Der Empfänger erkennt den Übergang vom Idle-Zustand zum Startbit und synchronisiert so seine Taktgenerierung. Dabei müssen in Sender und Empfänger die Datenübertragungsrate und andere Faktoren eingestellt werden. Dies führt bei RS 232 zu 25 % Verwaltungsdaten (Overhead), die mit übertragen werden müssen und so den eigentlichen Datendurchsatz schmälern. Ein anderes Verfahren benutzen SPI, I<sup>2</sup>C und Microwire. Bei diesen existiert neben der Datenleitung eine Taktleitung. Beide müssen an alle Slave-Bausteine des Busses herangeführt werden. Ebenso wie bei RS 232 sind grundlegende Konfigurationen notwendig.

Für Geräteentwickler und Programmierer verringert sich der Arbeitsaufwand erheblich, da die USB-Hardware die gesamte Konfiguration und Kodierung der Daten selbstständig vornimmt. Analysen mit einem Oszilloskop oder einem Logikanalysator lassen sich mit den kodierten Daten nur schwer durchführen, da einzelne Bits nicht erkennbar sind. Für diese Aufgaben werden spezielle Protokollanalysatoren eingesetzt.

Aus der NRZI-Kodierung erklärt sich der Einsatz von Bit-Stuffing. NRZI ist ein in der Informationselektronik oft genutztes Kodierungssystem. Wird im Eingangsstrom eine logische Null erkannt, so wird der aktuelle Buszustand invertiert. Bei einer logischen Eins bleibt der aktuelle Zustand erhalten. Würde der Datenstrom nur Einsen enthalten, könnten keine Singnalzustandswechsel erzeugt werden. Diese Wechsel des Signalzustandes sind für alle Empfänger zwingend notwendig, da die Taktsynchronisation der Empfänger sonst nicht funktioniert. Um die PLL der Empfänger zu synchronisieren, wird nach der sechsten aufeinander folgenden Eins eine Null eingeschoben. Damit wird ein Taktwechsel erzwungen. Diese Null wird im Empfänger durch einen Bit-Destuffer wieder entfernt. Rechnerisch kann so die Zahl der übertragenen Bits um 17 % steigen. Tatsächlich ist dieser Prozentsatz viel geringer als der berechnete Worst-Case-Wert.

### **3.4.1 Synchronisationsfeld (Sync-Field)**

Das Bit-Stuffing allein reicht nicht aus, um den Empfängertakt zu synchronisieren. Für eine dauerhafte Taktgenauigkeit während der Übertragung wird jedem Paket ein sogenanntes Synchronisationsfeld vorangestellt. Dieser erzwungene Zustandswechsel findet vor jedem Paket statt. Es werden pro Paket 8 Bit vorangestellt. Somit werden lange Pakete effektiver als kürzere. Ein solches Feld vor ein ganzes Paket zu stellen ist außerdem effizienter als vor jedes Byte ein Startbit zu setzen. Nach dem 8 Bit langen Synchronisationsfeld ist der Empfänger in der Lage, alle folgenden Bits eines Pakets auszuwerten.

### **3.4.2 Genauigkeit der Taktung**

Die normale Geschwindigkeit von 12 Mbit je Sekunde ermöglicht eine schnelle Kommunikation zwischen den Geräten. Dies führt allerdings zu hohen Anforderungen an Kabel und Taktgeber. Bei Low-Speed sind demzufolge durch die geringere Geschwindigkeit von 1,5 Mbit pro Sekunde die Qualitätsmerkmale von Taktgeber und Kabel niedriger. Die daraus resultierenden niedrigeren Kosten werden aber durch die hohen Ausgaben für die Halbleiter aufgehoben. Die langsameren Flankenraten von USB 1.0-Chips erfordern einen höheren Entwicklungs- und Technologieaufwand.

Für Full-Speed und High-Speed Geräte sind sehr hohe Anforderungen gesetzt, die sich hauptsächlich auf den angeschlossenen Quarz beziehen. Viele Faktoren können die Frequenz des Quarzes beeinflussen wie z.B. dessen ursprüngliche Genauigkeit, Kapazitätslasten,

Alterungsprozesse sowie Versorgungsspannungen und Temperatur. Die Frequenzgenauigkeit wird in Millionsteln (ppm – parts per million) angegeben. Darunter ist die maximale Anzahl der Zyklen zu verstehen, die der Quarz innerhalb der Nennfrequenz abweichen darf. Daraus lässt sich eine maximale Zeitdifferenz berechnen. Die geforderte Genauigkeit von 0,25 % bei Full-Speed Geräten besagt, dass im Höchstfall eine Taktabweichung von  $\pm 2500$  ppm auftreten darf, bevor das USB-Gerät die Synchronisation mit dem USB-Takt verliert. Für Low-Speed Geräte sind diese Anforderungen mit 1,5 % bedeutend niedriger. Dadurch können an dieser Stelle auch Keramikresonatoren eingesetzt werden. Um den Takt des Busses zu garantieren, muss der Host eine Genauigkeit von  $\pm 500$  ppm aufweisen. Hubs müssen ihre Rahmenintervalle an die des Hosts anpassen können, um die Taktgenauigkeit auf dem Bus aufrecht zu erhalten. Die Genauigkeit für High-Speed-Geräte mit 480 Mbit/s beträgt  $\pm 500$  ppm und ist damit fünfmal höher als bei Full-Speed.

### 3.5 High-Speed-Interface bei USB 2.0

Für die bei USB 2.0 erreichten Datenraten von bis zu 480 Mbit/s konnte der verwendete Busabschluss von 15 k $\Omega$  Pull-Down-Widerständen an der Upstream-Seite und 1,5 k $\Omega$  Pull-Up an der Downstream-Seite nicht weiter eingesetzt werden. Ebenso erwiesen sich der bisher definierte Spannungshub von 3,3 V sowie die Ansteuerung durch analoge Spannungstreiber als ungeeignet. Nach vielen Simulationen und zahlreichen Testreihen wurden folgende elektrische Voraussetzungen für den High-Speed-Modus definiert:

- Einsatz von differenziellen Stromtreibern für 17,78 mA
- Kein Pull-Up-Widerstand auf der Downstream-Seite
- Beidseitiger Abschluss des Busses mit  $Z=45\ \Omega \pm 10\ %$  für eine maximale Reflexionsunterdrückung

Wenn die Datenleitungen auf beiden Seiten mit jeweils 45  $\Omega$  abgeschlossen sind, dann beträgt der effektive Widerstand 22,5  $\Omega$ . Bei Stromtreibern mit 17,78 mA ergibt sich damit ein Spannungshub von 400 mV (Spitze-Spitze). Um diese Anforderungen zu erfüllen, sind wesentlich komplexere Transceiver als die bisher für USB 1.1 verwendeten notwendig. Im Bild 13 ist der prinzipielle Aufbau eines High-Speed-fähigen Transceivers dargestellt. Die differenziellen Receiver und die notwendigen Stromtreiber für den High-Speed-Modus arbeiten parallel zu den bisher genutzten USB 1.1-kompatiblen Full-/Low-Speed-Transceivern.

Man nutzt die vorhandenen Full-Speed-Treiber, damit die Datenleitungen beidseitig mit 45  $\Omega$  abgeschlossen werden können. Im High-Speed-Modus erzeugen die Full-Speed-Transceiver einen SE0-Zustand auf den Datenleitungen. Sie stellen damit einen virtuellen Massepunkt mit einer angekoppelten Impedanz von 45  $\Omega$  dar.

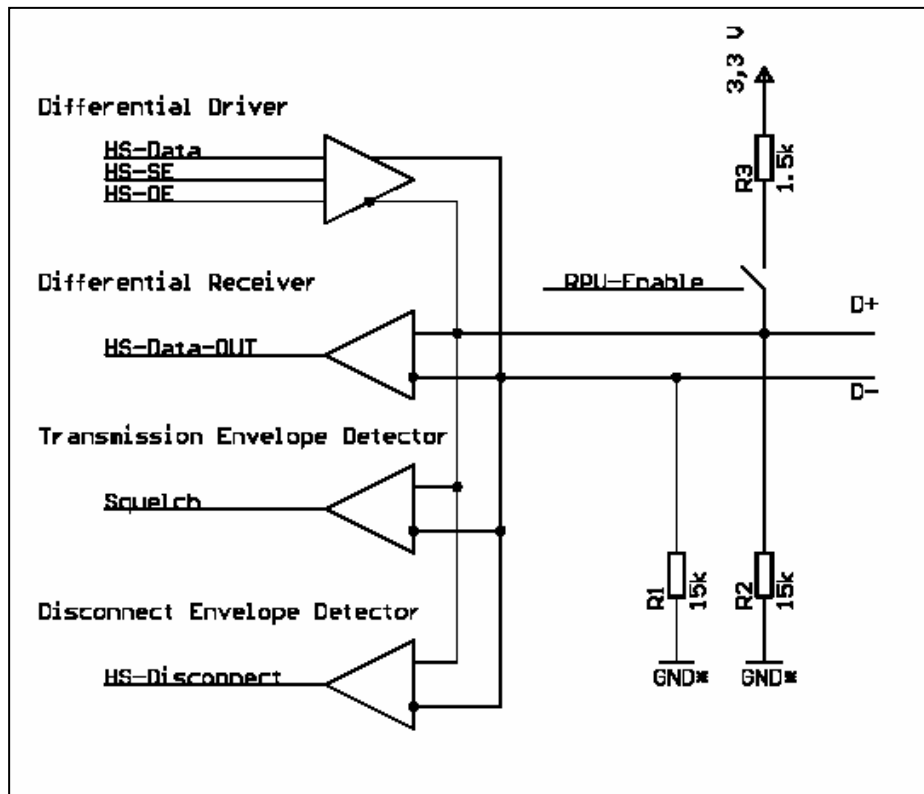


Abb. 13 – USB 2.0-Transceiver

Damit die Verbindung im USB 2.0 auch für Full- und Low-Speed-Geräte möglich ist, sind bestimmte Vorkehrungen notwendig.

Alle USB 2.0-kompatiblen Geräte werden beim Anstecken an den Bus mit Full-Speed-Geschwindigkeit nach USB 1.1 angesprochen. Erst wenn der Host erkannt hat, dass es sich um ein USB 2.0-fähiges Gerät handelt, wird eine Geschwindigkeitsumschaltung durch den Host zum Gerät initiiert. Ebenso muss ein High-Speed-Hub einen Anschluss von Full- und Low-Speed-Geräten ermöglichen. Der Hub enthält neben dem High-Speed-Transceiver auch einen Full- und Low-Speed-Transceiver. Um die Geschwindigkeiten umsetzen zu können, ist neben einem Speicher ein Controller eingebaut, der den Datentransfer koordiniert. Wenn ein Full- oder Low-Speed-Gerät an einen USB 2.0-Hub angeschlossen wird, so werden die Daten bis zum Hub im High-Speed-Modus übertragen. Lediglich die Kommunikation mit dem Gerät erfolgt in einer niedrigeren Geschwindigkeit. Daten, die das Gerät zum Host sendet, werden vom Hub in USB 2.0-fähige Busdaten gewandelt.

Da in der High-Speed-Betriebsart kein Pull-Up-Widerstand am Gerät vorhanden ist, kann die Disconnect-Erkennung nicht mehr in der herkömmlichen Weise erfolgen. Für diese Erkennung nutzt man jetzt die fehlenden Buswiderstände aus. Treibt ein Hub den Bus an  $45\ \Omega$  anstatt an  $22,5\ \Omega$ , so entsteht eine Spannungsüberhöhung von 400 mV auf 800 mV (Spitze-Spitze) auf der Treiberseite. Diese wird durch den implementierten „Disconnect-Envelope“ erkannt. Die Disconnect-Überprüfung erfolgt nur während der letzten 8 Bit des 40 Bit langen EOP-Feldes eines Micro-Frame-Tokens.

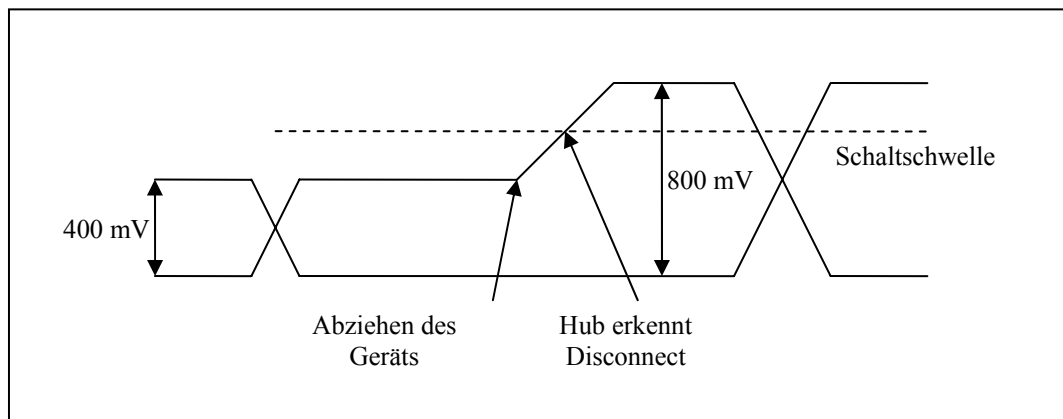


Abb.14 – Disconnect-Erkennung bei USB 2.0

Zusätzlich wurde eine Rauschunterdrückung implementiert. Dieser sogenannte Transmission-Envelope-Detector sorgt dafür, dass schwache Signale des differentiellen Receivers nicht ausgewertet werden (Squelch).

Bei der Low-Level-Kodierung wurde das Sync-Field von 8 Bit auf 32 Bit verlängert. Damit soll das sichere Einrasten der digitalen PLL auch dann ermöglicht werden, wenn das Gerät hinter mehreren Hubs angeschlossen ist.

Da die Datenübertragung im High-Speed-Modus nur unter Verwendung des Differential-Pair-Prinzips sicher gestaltet werden kann und der High-Speed-IDLE-Zustand einem SE0 entspricht, wurde auch eine andere Kodierung des EOP (End-Of-Packet) gewählt. In der USB1.1-Spezifikation wurde ein EOP durch ein zwei Taktzyklen langes SE0, gefolgt vom IDLE-Zustand festgelegt. Dagegen entspricht im High-Speed-Modus ein Bit-Stuff-Error dem EOP. Dieser Fehler wird künstlich erzeugt, indem mindestens sieben aufeinander folgende logische Einsen gesendet werden. Die NRZI-Kodierung wird dabei deaktiviert, so dass keine eingeschobenen Nullen entstehen.

## 4 Transferarten

### 4.1 Pipe- und Endpoint-Konzept

Physikalisch besteht die Verbindung zu einem USB-Gerät nur aus der D+ und der D- -Leitung. Der USB unterstützt jedoch die Einrichtung mehrerer virtueller logischer Datenkanäle, den sogenannten Pipes. Jede Pipe endet in einem Endpoint. Ein USB-Gerät kann mehrere Endpoints und somit mehrere logische Pipes unterstützen. Hierbei wird das Pipe-Konzept von der Software des Treibers im PC angesprochen. Physikalisch gesehen ist ein Endpoint eine FIFO mit einer festgelegten Tiefe, welche USB-Daten empfangen oder senden kann. Bei der Adressierung von USB-Daten auf dem Bus wird neben der 7 Bit langen USB-Adresse des Geräts eine weitere 4 Bit lange Endpoint-Nummer gesendet. Dadurch wird die angesprochene FIFO in einem USB-Gerät eindeutig identifiziert. Gleichzeitig dient die Richtung des Datentransfers als zusätzliches Adressierungsmerkmal für den entsprechenden Endpoint.

Damit ist es möglich, mehrere logische Geräte innerhalb eines physischen USB-Gerätes anzusprechen. Mehrere Endpoints werden meist zu einem Interface zusammengefasst. Hiermit wird deutlich, dass USB, im Gegensatz zu alten Protokollen wie RS 232, keinen privaten Header benötigt, um das adressierte Gerät zu bestimmen.

Alle USB-Geräte müssen einen Control-Endpoint EP0 unterstützen. Dieser ist der einzige bidirektionale Endpoint. Über EP0 können sowohl Daten empfangen als auch gesendet werden. Alle anderen Endpoints arbeiten unidirektional. Sie können Daten entweder senden oder empfangen.

Die Richtung des Datentransfers wird immer aus der Sicht des Host bestimmt. Endpoints in Geräten, die Daten vom Host empfangen können, sind Out-Endpoints. Out-Transaktionen laufen in Downstream-Richtung ab. Als klassisches Beispiel ist hierbei der Transfer von Druckdaten vom PC zum Drucker zu sehen.

Wenn die Daten in Upstream-Richtung von einem Gerät zum Host übertragen werden, dann spricht man von einer In-Transaktion. In-Endpoints sind demnach FIFO's, die Daten zum Host senden können.

Zur Übertragung der Daten kann zwischen den vier Transfermodi gewählt werden:

- Control-Transfer
- Interrupt-Transfer
- Bulk-Transfer
- Isochronous-Transfer

Dem Endpoint EP0 ist dabei immer der Control-Transfer zugeordnet. Allen anderen Endpoints lässt sich jeweils eine beliebige Transferart zuordnen. Bis auf den EP0 können alle Endpointnummerierungen zweimal auftreten. So kann beispielsweise zweimal der EP1 vorhanden sein. Diese Endpoints unterscheiden sich dann in ihrer Datenrichtung. Es gibt einen EP1 für Out-Transfer und einen EP1 für In-Transfer. Unterschieden werden die Endpoints durch die implizite Richtungsangabe in der Endpointadresse. Zwei gleichnamige Endpoints mit den gleichen Transferrichtungen sind nicht zu unterscheiden und dürfen nicht programmiert werden.

Durch die 4 Bit lange Endpointadresse sind demnach 16 mögliche Endpoints in einem Gerät vorhanden. Da immer ein EP0 für den Control-Transfer angelegt ist, bleiben noch 15 weitere Endpoints, die beliebigen Transferrichtungen und Transferarten zugeordnet werden können.

Die Pipes lassen sich in Message- und Stream-Pipes einteilen. Daten, die über Message-Pipes übertragen werden, besitzen durch die USB-Spezifikation eine definierte Struktur. Die Firmware des Gerätes ist damit in der Lage, die Daten zu interpretieren. Diese Message-Pipes nutzen den Control-Transfer zur Übertragung. Auf vom Host empfangene Daten muss das Gerät mit bestimmten Daten antworten. Der in jedem USB-Gerät vorhandene EP0 ist immer als Message-Pipe ausgelegt.

Stream-Pipes übertragen Daten, die nicht durch das USB-Protokoll spezifiziert sind. Durch sie werden alle möglichen Daten gesendet oder empfangen. Stream-Pipes arbeiten unidirektional und verwenden Interrupt-, Bulk- oder Isochronous-Transfer.

## 4.2 Control-Transfer

Um die Konfiguration und Steuerung eines Gerätes zu ermöglichen, wird der Control-Transfer genutzt. Die definierten Datenströme, auch Standard-Device-Requests genannt, werden über den bidirektionalen EP0 übertragen. Die Out-FIFO des Endpoints muss mindestens 8 Byte tief sein, um die Requests aufnehmen zu können. Bei Low-Speed-Geräten beträgt diese FIFO-Tiefe exakt 8 Byte. In Full-Speed-Geräten kann die Tiefe mit den Werten 8, 16, 32 oder 64 Byte variieren. Im High-Speed-Modus ist sie auf exakt 64 Byte festgelegt worden. Für Datenübertragungen vom Gerät zum Host wird immer die maximale FIFO-Tiefe ausgenutzt. Dieser Wert ist im Device-Descriptor angegeben. Für den Control-

Transfer sind auf dem Bus bis zu 10% der Bandbreite reserviert. Sollte das Datenaufkommen niedriger sein, so kann der Control-Transfer auch mehr als 10% nutzen. Die Daten werden garantiert geliefert, da die USB-Spezifikation eine Fehlerkorrektur vorsieht. Zusätzlich greift hier ein doppeltes Handshake-Verfahren, welches die Datensicherheit enorm erhöht. Es lassen sich mittels Control-Transfer auch Daten an das Gerät senden. Diese Anwendung ist in Fachkreisen aber umstritten. Ein Standpunkt ist, dass der Control-Transfer ausschließlich für die Übertragung der Requests vorbehalten werden sollte. Andere vertreten die Auffassung, alle Datenübertragungsmöglichkeiten zu nutzen, die das Gerät anbietet. Eine effektive Datenübertragung ist ohnehin nicht möglich, da ein zu großer Overhead von 45 Byte mit gesendet wird. Die zu erreichende Datenrate ist im Vergleich zu anderen Transferarten sehr gering.

Control-Transfers nutzen eine definierte Struktur aus zwei oder drei Abschnitten: Setup-, Daten- (optional) und Status-Abschnitt. Ein Abschnitt besteht aus einer oder mehreren Transaktionen. Die Einteilung dieser drei Abschnitte ist nicht zu verwechseln mit den unterschiedlichen Transaktionsphasen und -paketen (Token, Data und Handshake).

### 4.3 Interrupt-Transfer

Interrupt-Transfers sind anwendbar, wenn in einer spezifischen Zeitdauer kleinere Datenmengen zu übertragen sind. Typische Anwendungen umfassen Tastaturen, Mäuse und andere Zeigegeräte, Joysticks und Hub-Statusberichte. Der Anwender möchte keine merkliche Verzögerung zwischen dem Drücken einer Taste oder dem Bewegen einer Maus und dem Ergebnis auf dem Bildschirm wahrnehmen. Ebenso muss ein Hub das Anschließen oder Entfernen von Geräten sofort melden. Low-Speed-Geräte, die nur Control- und Interrupt-Transfer unterstützen, verwenden meist Interrupt-Transfers für generische Daten.

Der Name Interrupt-Transfer legt nahe, dass ein Gerät einen Hardware-Interrupt auslösen kann, der zu einer schnellen Antwort vom PC führt. Tatsächlich finden aber Interrupt-Transfers wie alle anderen USB-Übertragungen nur dann statt, wenn der Host ein Gerät abfragt. Damit sind Interrupt-Transfers lediglich in der Bedeutung mit klassischen Interrupts gleichzusetzen. Sie dienen zur schnellen und zyklischen Übertragung von Daten mit garantierten Latenzzeiten.

Es ist nicht notwendig, dass alle Geräte Interrupt-Transfers unterstützen, wobei spezielle Geräteklassen dies aber verlangen können. So muss etwa ein Gerät in der USB-HID-Klasse Interrupt-In-Übertragungen zum Senden von Daten an den Host unterstützen. Nähere Informationen zu den USB-Geräteklassen sind in Kapitel 10 zu finden.

Ein Low-Speed-Gerät hat eine maximale Paketgröße von 8 Byte. Bei Full-Speed-Geräten ist die Paketgröße einstellbar von 1 bis 64 Byte. High-Speed-Geräte unterstützen eine maximale Paketgröße von 1024 Byte pro Microframe. Das heißt, innerhalb eines Microframes können bis zu 6 Pakete übertragen werden. Daraus ergibt sich eine theoretische maximale Datenrate von 49,152 Mbyte/s.

Ein Interrupt-Transfer garantiert das Nichtüberschreiten einer maximalen Latenzzeit bzw. der Zeitdauer zwischen den Transaktionsversuchen. Es gibt also keine garantierte Datenrate, sondern eine garantierte Maximalzeit zwischen den Transaktionen. Der im Gerät gespeicherte Endpoint-Descriptor spezifiziert die Latenzzeit, die für Full-Speed-Geräte einen beliebigen Wert zwischen 1 und 255 ms und für Low-Speed-Geräte einen beliebigen Wert zwischen 10 und 255 ms haben kann. Der Host-Controller stellt sicher, dass zwischen den Transaktionen nicht mehr als die angegebene Zeitdauer liegt. Es muss jedoch nicht die maximale Zeit zwischen zwei Transaktionen vergehen, bevor der Host die darauf folgende Übertragung beginnt. So können fünf Übertragungen bei einem Maximum von 10 ms bis zu 50 ms oder auch nur 5 ms dauern. Normalerweise verwendet der Host Werte, die Zweierpotenzen



entsprechen. Wenn das Gerät eine maximale Latenzzeit von 15 ms fordert, so beginnt der Host alle 8 ms eine Übertragung. Der High-Speed-Modus erlaubt eine Einstellung von einer bis drei Transaktionen pro Microframe.

Weil der Host Daten schneller übertragen kann als das Gerät fordert, kann für die Transaktion mit einer Ausnahme keine genaue Übergabezeit zugesichert werden. Eine für 1 ms konfigurierte Interrupt-Pipe muss genau eine Übertragung pro Millisekunde ausführen, da dies die schnellstmögliche Rate ist.

Vor dem Konfigurieren einer Pipe für den Interrupt-Transfer vergleicht der Host die angeforderte Puffergröße, um zu ermitteln, ob die notwendige Bandbreite zur Verfügung steht. Wenn auf dem Bus kein Platz mehr für die Übertragung ist, lehnt der Host die Anforderung zur Konfiguration ab.

Um Interrupt-Transfers effizient zu nutzen, müssen sowohl das sendende als auch das empfangende Gerät in der Lage sein, die Daten mit der geforderten Rate in die USB-Puffer hinein zu schreiben und aus ihnen wieder heraus lesen zu können. Isochronous und Interrupt-Transfer können gemeinsam nicht mehr als 90% der USB-Bandbreite nutzen.

Zur Fehlererkennung hat der Interrupt-Transfer eine CRC sowie ein Handshake. Falls ein Paket fehlerhaft übertragen wird, versucht der Host noch zwei weitere Übertragungen des Paketes, bevor er einen Fehler an den Treiber meldet und abbricht. Zur Sicherheit gibt es zusätzlich ein Toggle-Bit, das beim Senden oder Empfangen von neuen Daten umgeschaltet wird.

## 4.4 Isochronous-Transfer

Ein wesentliches Merkmal des USB ist die Unterstützung des Isochronous-Transfer. Damit ist die Übertragung von Daten möglich, die eine konstante Bandbreite erfordern. Typische Anwendungen sind die Übertragung von Audio- und Videodaten. Bei dieser Übertragungsvariante müssen geringfügige Datenfehler akzeptiert werden. Verspätet ankommende Daten gelten für Echtzeitanwendungen als unnutzbare und damit fehlerhafte Daten.

Isochron bedeutet, dass die Daten mit einer festen Übertragungsrate und einer definierten Anzahl von Bytes in jedem Frame übertragen werden. Sämtliche anderen Transfertypen sind asynchron, das heißt, sie stellen nicht sicher, dass in jedem Frame eine bestimmte Anzahl von Bytes gesendet wird. Eine Ausnahme bildet hier der Interrupt-Transfer mit 1ms Abfrageintervall.

Isochronous-Transfer ist nur für Full- und High-Speed-Geräte spezifiziert. Bei USB 1.1 kann die FIFO-Tiefe von 1 bis 1024 Byte im Descriptor frei gewählt werden. Im High-Speed-Modus können bis zu 3072 Byte innerhalb eines Microframes übertragen werden, die sich aus drei Datenpaketen von maximal 1024 Byte zusammensetzen.

Bei allen isochronen Transfers wurde auf ein Handshake verzichtet, damit effektive Übertragungsraten erreicht werden können. Der Protokoll-Overhead beträgt 9 Byte pro Transaktion. Das entspricht bei einem 1023 Byte langen Paket weniger als 1%.

Für eine isochrone Datenübertragung muss eine Synchronisation zwischen Datenquelle und Senke erreicht werden. Dazu sieht die USB-Spezifikation folgende Mechanismen vor:

- Asynchron

Isochrone Daten werden als asynchron bezeichnet, wenn sie zu einem externen Takt synchronisiert sind. Dieser Takt beträgt 1ms (USB Framelänge).

- Synchron

Synchrone Geräte können sich auf den 1ms isochronen Datentakt synchronisieren. Wenn dieser Mechanismus nicht in einem Gerät integriert ist, so kann der Host-Treiber für dieses Gerät die Framelänge so nachjustieren, dass sich das Gerät wieder synchron zum Datentakt befindet. USB erlaubt diese Framejustage nur einem Master-Gerät auf dem Bus. Für andere, vom Frametakt abhängige Geräte, kann es zu Störungen kommen.

- Adaptiv

Ist ein Gerät in der Lage, sich adaptiv zu synchronisieren, so kann es den Datenstrom in einem gewissen Bereich anpassen. Als Beispiel soll hier eine isochrone Pipe für 16-Bit Audio-Daten mit einer Sampling-Rate von 44,1 kHz dienen. In den meisten Frames werden 88 Byte gesendet. Um die Datenrate wieder anzupassen, wird alle 10 Frames ein 90 Byte langes Datenpaket gesendet.

## 4.5 Bulk-Transfer

Für die Übertragung von zeitunkritischen und großen Datenmengen eignet sich der Bulk-Transfer. Damit können große Datenaufkommen übertragen werden, ohne den Bus zu überlasten. Das ergibt sich daraus, dass Bulk-Transfers bei der Zusammenstellung der Frames erst nach allen anderen Transferarten berücksichtigt werden. Ist im aktuellen Frame kein Platz für die Übertragung frei, wird der Transfer auf das nächste Frame verschoben. Das bedeutet, Bulk-Transfers haben keine zugesicherte Busbandbreite. Die Zeit, die nach allen andern Transfers noch vorhanden ist, kann genutzt werden. Auf einem leeren Bus ist diese Transferart die Schnellste. Für den Einsatz von Bulk-Transfers gibt es unterschiedliche Host-Spezifikationen bei UHCI und OHCI (siehe Kapitel 9).

Bulk-Endpoints werden nur angesprochen, wenn auch wirklich Daten vorhanden sind. In der Out-Richtung startet der Host eine Übertragung ohne vorherige Ankündigung. Für In-Transfers wird meist über einen Interrupt-Transfer eine Ankündigung gesendet und damit der Transfer beim Host angemeldet.

Bulk-Transfer ist ab USB 1.1 spezifiziert. Full-Speed-Geräte können bis zu 64 Byte tiefe FIFOs bereitstellen. Dabei können theoretisch bis zu 19 Transaktionen in einem Frame übertragen werden. Im High-Speed-Transfer sind 512 Byte pro Transaktion möglich. Mit bis zu 13 Transaktionen pro Microframe erreicht man damit die maximale theoretische Übertragungsrate von 53,25 Mbyte/s. Die Buseinteilung in Frames und Microframes wird in Kapitel 5.3 beschrieben

Der Bulk-Transfer verfügt über Fehlererkennung und Handshake. Somit ist eine fehlerfreie Übertragung gewährleistet. Zusätzlich ist ein Data-Toggle-Bit vorhanden, um sicherzustellen, dass keine Daten verloren gehen.

## 4.6 Zusammenfassung und Bandbreitenabschätzung

Die folgenden Tabellen sollen die übersichtliche Zusammenstellung aller vier Transfertypen darstellen.

Transferart	FIFO-Tiefe in Byte		Fehlerkorrektur	Bandbreite
Control	Low-Speed:	8	Ja	10% garantiert
	Full-Speed:	8,16,32,64		
	High-Speed:	64		
Interrupt	Low-Speed:	1..8	Ja	90% garantiert zusammen mit Isochronous
	Full-Speed:	1..64		
	High-Speed:	1..3072		
Bulk	Full-Speed:	8,16,32,64	Ja	Nur bei verfügbarer Bandbreite
	High-Speed:	1..512		
Isochronous	Full-Speed:	1..1024	Nein	90% garantiert mit Interrupt
	High-Speed:	1..3072		

Tabelle 5 – Zusammenfassung der Transferarten

Transferart	Protokoll-Overhead in Byte	Maximale FIFO-Tiefe in Byte	Übertragungen pro Microframe	Maximal erreichbare Bandbreite
Control	45	64	1 (EHCI)	4,0 Mbit/s
Interrupt	13	3 x 1024	1	192 Mbit/s
Bulk	13	512	Max. 13	416 Mbit/s
Isochronous	9	3 x 1024	1	192 Mbit/s

Tabelle 6 – Maximale Übertragungsraten für High-Speed-Geräte

Transferart	Protokoll-Overhead in Byte	Maximale FIFO-Tiefe in Byte	Übertragungen pro Frame	Maximal erreichbare Bandbreite
Control	45	64	Max. 13 (OHCI)	6,6 Mbit/s
			1 (UHCI)	512 Kbit/s
Interrupt	13	64	1	512 Kbit/s
Bulk	13	64	Max. 19	9,7 Mbit/s
Isochronous	9	1024	1	8,2 Mbit/s

Tabelle 7 – Maximale Übertragungsraten für Full-Speed-Geräte

Transferart	Protokoll-Overhead in Byte	Maximale FIFO-Tiefe in Byte	Übertragungen pro Microframe	Maximal erreichbare Bandbreite
Control	46	8	Max. 3 (OHCI)	192 Kbit/s
			1 (UHCI)	64 Kbit/s
Interrupt	13	8	1	8 Kbit/s

Tabelle 8 – Maximale Übertragungsraten für Low-Speed-Geräte

## 5 Framework und USB-Pakete

### 5.1 Bausteine der Übertragung für USB 1.1

In der folgenden Abbildung soll die Bedeutung der Begriffe Transfers, Transaktionen, Abschnitte und Phasen, Data-Transaktionen und Datenpakete, Status-Abschnitte und Handshake-Phasen dargestellt werden.

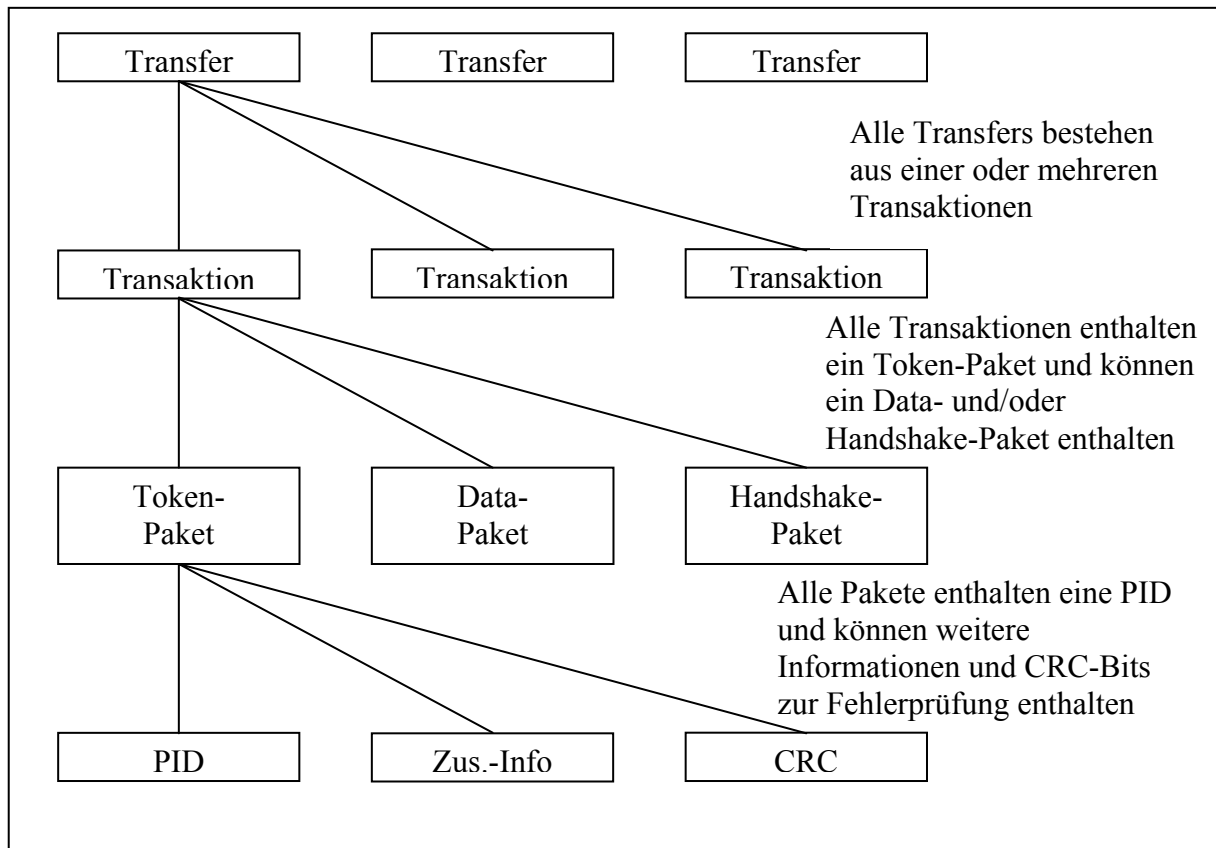


Abb. 15 – Bausteine der USB-Übertragung

Transfers bestehen aus einem der vier Transfertypen. Dabei setzt sich jeder Transfer aus einer oder mehreren Transaktionen zusammen, die wiederum ein bis drei Pakete enthalten können. Die drei Transaktionstypen sind durch ihren Zweck und die Richtung des Datenstroms definiert. Die Setup-Transaktion dient dem Senden von Control-Transfer-Requests an ein Gerät. Während mit In-Transaktionen Daten vom Gerät zum Host gesendet werden, geschieht das auf umgekehrtem Wege durch Out-Transaktionen. In der USB-Spezifikation ist eine Transaktion als Übergabe eines Dienstes an einen Endpoint definiert. Dienst bedeutet hier, dass der Host Daten an das Gerät sendet oder dass er Informationen vom Gerät anfordert und empfängt. Jede Transaktion enthält Identifizierungs-, Fehlerprüf-, Status- und Control-Informationen sowie beliebige auszutauschende Daten. Während ein Transfer mehrere Frames umfassen kann, ist eine Transaktion ein einzelner Kommunikationsabschnitt, der ohne Unterbrechung innerhalb eines Frames stattfindet. Das angesprochene Gerät muss sehr schnell auf Datenanforderungen und Statusinformationen reagieren. Bis auf das Füllen der Datenendpoints wird deshalb ein Großteil des Protokolls durch Hardware realisiert. Kleine Datenmengen können möglicherweise durch eine Transaktion übertragen werden. Ist die Datenmenge größer, wird die Übertragung in mehrere Transaktionen aufgeteilt. Jede Transaktion enthält bis zu drei Phasen oder Bestandteile: Token, Data und Handshake. Jede Phase besteht wiederum aus einem oder zwei übertragenen Paketen. Dabei bildet ein

Paket einen fest definierten Informationsblock, wobei jedes Paket mit einer PID (Paket-Identifizier) beginnt. Abhängig vom Transaktionstyp folgen der PID Adressen, Daten oder Statusinformationen, die mit Fehlerprüfbits abgesichert sind. Die folgende Tabelle soll alle Pakettypen des Standards USB 1.1. auflisten.

<b>Paketttyp</b>	<b>Name</b>	<b>Wert (MSB zuerst)</b>	<b>Beschreibung</b>
Token	OUT	0001	Endpointadresse für Out-Transaktion
	IN	1001	Endpointadresse für In-Transaktion
	SOF	0101	Start-of-Frame-Markierung mit Framenummer
	SETUP	1101	Endpointadresse für Setup-Transaktion
Data	DATA0	0011	Datenpaket mit Data-Toggle-Bit Null
	DATA1	1011	Datenpaket mit Data-Toggle-Bit Eins
Handshake	ACK	0010	Bestätigung Datenpaket fehlerfrei angenommen
	NAK	1010	Empfänger kann Daten nicht annehmen oder der Sender kann die Daten nicht senden oder hat keine Daten zum Senden
	STALL	1110	Ein Control-Request wird nicht unterstützt, oder der Endpoint wurde angehalten
Special	PRE	1100	Präambel wird vom Host ausgegeben und aktiviert beim HUB den Low-Speed-Modus für das Gerät

Tabelle 9 – Pakettypen; die unteren vier Bits der PID geben den Wert an, das obere Nibble dient der Fehlerprüfung

Das PRE-Paket ist ein Spezialfall. In ihm ist ein Präambel-Code enthalten, der dem Hub mitteilt, dass das nächste Paket für ein Low-Speed-Gerät bestimmt ist. Damit ist der Host in der Lage, das angeschlossene USB 1.0-Gerät anzusprechen. Da Low-Speed-Geräte vor dem davor liegenden Hub vor Full-Speed-Datenransfer geschützt sind, können SOF-Pakete nicht versendet werden. Damit das Gerät nicht in den Suspend-Modus umschaltet, muss eine Zeitreferenz übertragen werden. Dazu erzeugt der Hub ein spezielles Low-Speed-Keep-Alive-Signal. Dieses Signal entspricht einem EOP (End-of-Package). Das Kabelsegment wird für zwei Low-Speed-Zyklen auf Low getrieben und geht danach wieder in den Idle-Zustand über.

Die Zeitreferenz SOF (Start-of-Frame) wird am Anfang jedes neuen Frames gesendet. Geräte, die eine Synchronisation auf den USB-Frame-Takt durchführen müssen, nutzen dieses Paket. Zusätzlich enthält dieses Paket eine 11 Bit lange Framenummer, die in jedem neuen Frame inkrementiert wird. Erreicht diese Zahl ihren Maximalwert von 2047, so springt sie im nächsten Schritt wieder auf 0 zurück. Die SOF-Pakete verhindern, dass ein Gerät in den stromsparenden Suspend-Zustand übergeht.

Zu jeder Transaktion gehört ein Token-Paket. Der Host ist immer die Quelle des Paketes. Es konfiguriert die Transaktion in allen Parametern wie z. B. Datenrichtung oder Endpoint. Wenn die Transaktion für ein Low-Speed-Gerät gedacht ist, geht dem ein PRE-Paket voraus. Abhängig davon, ob Daten folgen sollen, wird dem Token-Paket ein Daten-Paket nachgestellt. Ob der Host oder das Gerät dieses Daten-Paket sendet, wird im vorhergehenden Token-Paket festgelegt. Einige Transaktionen enthalten kein Daten-Paket. Verschiedene Requests im Control-Transfer erfordern dies nicht, da die Kodierung meist im Request selbst enthalten ist. Dieser wird im Token-Paket übertragen.

Bei allen Transfertypen, außer beim isochronen Transfer, gibt der Empfänger eines Datenpaketes ein Handshake-Paket zurück, um zu signalisieren, ob die Übertragung erfolgreich war oder Fehler aufweist. Wenn ein erwartetes Handshake-Paket nicht gesendet wird, erkennt der Empfänger dies als schwerwiegender Fehler.

Um eine schnelle Kommunikation zu erreichen, sind die Wartezeiten zwischen den Paketen mitunter sehr kurz. In der Konfigurationsphase eines Gerätes werden aber auch Antwortzeiten bis 5 Sekunden toleriert.

ACK-Pakete sind Bestätigungen, dass ein Gerät oder der Host Daten erfolgreich empfangen hat. Geräte geben außerdem bei einem erkannten Setup-Paket ein ACK-Paket zurück. NAK-Pakete sind negative Bestätigungen. Sie werden von Geräten gesendet, die Daten vom Host erhalten sollen, diese aber nicht annehmen können, weil sie mit der Verarbeitung anderer Daten ausgelastet sind. Das Gleiche geschieht wenn der Host vom Gerät Daten anfordert, diese aber nicht verfügbar sind. Für beide Situationen gilt das NAK-Paket als temporärer Zustand, so dass der Host die Transaktion zu einem späteren Zeitpunkt wiederholt. Der Host sendet nie ein NAK-Paket. Da der Isochronous-Transfer kein Handshake unterstützt, sind Datenpakete, die bei der Übertragung verloren gehen, endgültig verloren.

Eine besondere Bedeutung nehmen STALL-Pakete ein. Sie gelten als Blockierungs-Handshake und können eine der drei Informationen anzeigen:

- Ein Endpunkt ist außer Betrieb
- Ein Control-Request ist fehlgeschlagen
- Ein Control-Request wird nicht unterstützt

Das STALL-Paket wird ausschließlich vom Gerät an den Host gesendet. Wenn der Host das Gerät mit einem Request auffordert, eine Konfiguration einzustellen, die das Gerät nicht unterstützt, dann sendet es ein STALL-Paket zurück. In diesem Fall wird das STALL-Paket als Protokoll-STALL bezeichnet. Wenn ein Endpoint nicht in der Lage ist, Daten zu senden oder zu empfangen, dann sendet das Gerät auf die Anforderung des Hosts ein STALL-Paket. Dieses wird dabei Funktions-STALL genannt.

## **5.2 Zusätzliche Pakete für USB 2.0**

Dieser Abschnitt soll die grundlegenden Veränderungen im USB 2.0-Protokoll zusammenfassen. Auch wenn USB 2.0 eine wesentliche Erhöhung der Übertragungsgeschwindigkeit auf bis zu 480 Mbit/s aufweist, so ist es mit USB 1.1 und USB 1.0 weiterhin kompatibel.

Auf der Protokollebene kommen weitere PID hinzu, die vor allem bei der Kommunikation mit USB 2.0-Geräten von Bedeutung sind.

Pakettyp	Name	Wert (MSB zuerst)	Beschreibung
DATA	DATA2	0111	Wenn im High-Speed-Modus bei einem isochronen Transfer mehr als 3 In-Transaktionen in einem Microframe aufeinander folgen, so wird der Transfer zyklisch mit einem DATA2-Paket begonnen und wird mit DATA1 und DATA0 fortgesetzt.
	MDATA	1111	Wenn im High-Speed-Modus bei einem isochronen Transfer mehr als eine Out-Transaktion pro Microframe stattfindet, muss der Transfer zyklisch mit einem MDATA beginnen, bzw. bis auf das letzte Paket mit MDATA fortgesetzt werden.
Handshake	NYET	0110	Dieses Paket muss von High-Speed-Geräten gesendet werden, wenn der Endpoint noch die Daten der aktuellen Transaktion verarbeiten kann, weitere aber nicht mehr annimmt. Der Host muss dann mit einem Ping-Paket prüfen, ob der Endpoint wieder Daten liefern oder empfangen kann.
Special	PING	0100	Ein nicht empfangsbereites High-Speed-Gerät wird nach einem NYET-Paket vom Host mit PING-Paketen abgefragt, ob es wieder Daten empfangen kann. Das Gerät antwortet darauf mit NAK oder ACK. Nach einer ACK-Antwort sendet der Host wieder Daten. Es muss also nicht das gesamte Datenpaket erneut gesendet werden. Dadurch wird der Bus weniger belastet, so dass mehr Bandbreite verfügbar ist.
	SPLIT	1000	SPLIT-Pakete werden bei der Kommunikation mit High-Speed-Hubs benutzt, wenn ein Full- oder Low-Speed-Gerät angeschlossen wird. Mit Split-Paketen können langsamere Transaktionen bis zum entsprechenden Hub in High-Speed übertragen werden. Diese Methode verhindert, dass der Bus blockiert wird.
	ERR	1100	Mit einem ERR-Paket meldet der Hub die fehlerhafte Übertragung einer langsameren Transaktion, die durch ein SPLIT gekennzeichnet wurde.

Tabelle 10 – Zusätzliche Pakete für USB 2.0

## 5.3 Framework

Die Bandbreite des USB wird durch den Host in 1 ms Zeitabschnitte eingeteilt. Diese werden als Frames bezeichnet. Bei Full-Speed-Modus mit einer Taktfrequenz von 12 MHz können innerhalb eines Frames 12000 Bit übertragen werden. Im Low-Speed-Modus sind bei einem Bustakt von 1,5 MHz entsprechend 1500 Bit möglich. Damit alle Geräte diese Frame-

Einteilung erkennen, sendet der Host jede Millisekunde ein SOF-Token (Start-of-Frame-Token). Dieses Paket besitzt auf dem Bus die höchste Priorität.

Für den High-Speed-Modus wurde das Framework geändert. Ein normales Frame vom 1 ms wird durch den Host in 8 Microframes von je 125  $\mu$ s Länge eingeteilt. Dazu wurde das SOF in ein  $\mu$ SOF verändert. Die 11 Bit lange Framenummer wurde auf 8 Bit gekürzt. Die restlichen 3 Bit dienen jetzt der Identifikation des Microframes.

Als Beispiel soll hier ein leeres Framework aller Geschwindigkeitsstufen dargestellt werden. Der Wert in den Idle- und EOP-Feldern entspricht der Anzahl der Takte.

Idle	EOP
1498	2
Idle	EOP
1498	2
Idle	EOP
1498	2
Idle	EOP
1498	2
Idle	EOP
1498	2

Abb. 16 – Leeres Low-Speed-Framework

Idle	SYNC	SOF	Frame-nr.	CRC 5	EOP
11965	00000001	0xA5	0x7FD	0x1F	2
Idle	SYNC	SOF	Frame-nr.	CRC 5	EOP
11965	00000001	0xA5	0x7FE	0x1D	2
Idle	SYNC	SOF	Frame-nr.	CRC 5	EOP
11965	00000001	0xA5	0x7FF	0x02	2
Idle	SYNC	SOF	Frame-nr.	CRC 5	EOP
11965	00000001	0xA5	0x001	0x08	2
Idle	SYNC	SOF	Frame-nr.	CRC 5	EOP
11965	00000001	0xA5	0x002	0x17	2

Abb. 17 – Leeres Full-Speed-Framework



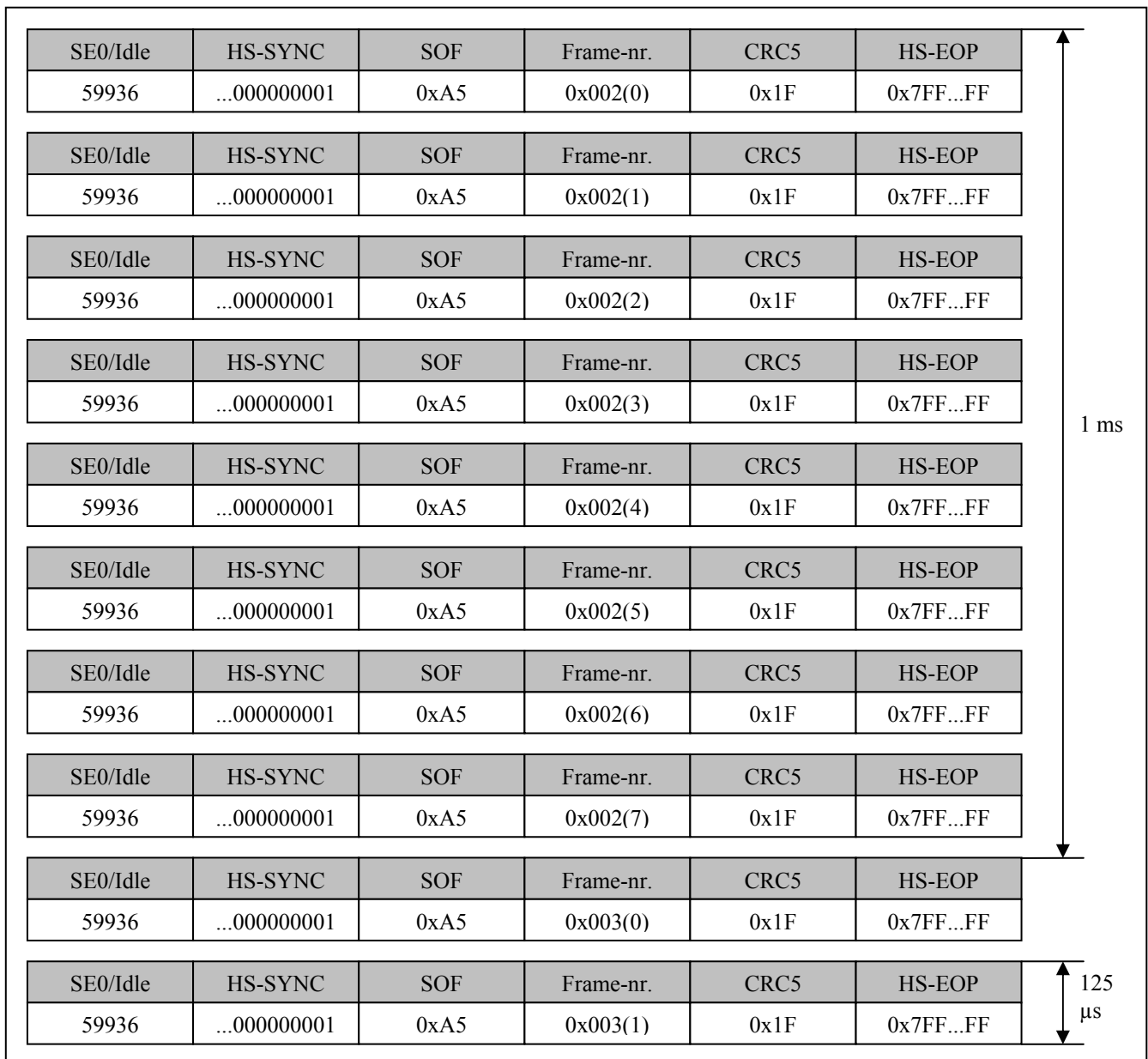


Abb. 18 – Leeres High-Speed-Framework

Die Darstellung eines leeren Frameworks soll grundlegende Buskommunikationen darstellen. In der Fachliteratur /1/, /2/ ist eine Vielzahl von Beispielen für fast alle möglichen Kommunikationssituationen zu finden. Eines der Beispiele soll an dieser Stelle näher erläutert werden:

- Die OUT-Transaktionen über Stream-Pipes bei USB 1.1

Durch Out-Transaktionen werden Daten vom Host zum Gerät übertragen. Dabei wird zuerst vom Host ein Out-Token mit der Adresse und dem Endpoint des Gerätes gesendet. Unmittelbar folgend werden die Daten übertragen. Wenn das Gerät die Daten erfolgreich annimmt, sendet es ein ACK zurück an den Host. Sollten mehrere Daten-Pakete aufeinander folgen, so wechseln die Pakete dann von DATA0 zu DATA1. Im aktuellen Beispiel beschränkt sich der Transfer auf eine Transaktion, die in einem Frame stattfindet. Wenn sich der Transfer auf mehrere Transaktionen aufteilt, wird zu Beginn eines neuen Frames ein SOF-Paket zwischen die Transaktionen geschoben.

HOST	Idle	SYNC	OUT	ADDR	EP	CRC5	EOP
	4	00000001	0xE1	0x02	0x02	0x01	2

HOST	Idle	SYNC	Data0	DATA	CRC16	EOP
	3	00000001	0xC3	00 11 22 9D A5 FF CD 7F	0xCBA8	2

GERÄT	Idle	SYNC	ACK	EOP
	4	00000001	0xD2	2

Abb. 19 – Out-Transaktion ohne Fehler

Der Host prüft bei USB 1.1 nicht, ob der Endpoint weitere Daten aufnehmen kann. Nur in der Handshake-Phase erkennt der Host, ob die Daten fehlerhaft übertragen wurden. Sendet das Gerät ein NAK-Token, dann versucht der Host zu einem späteren Zeitpunkt, die Daten erneut zu senden.

HOST	Idle	SYNC	OUT	ADDR	EP	CRC5	EOP
	4	00000001	0xE1	0x02	0x02	0x01	2

HOST	Idle	SYNC	Data0	DATA		CRC16	EOP
	3	00000001	0xC3	00 11 22 9D A5 FF CD 7F		0xCBA8	2

GERÄT	Idle	SYNC	NAK	EOP
	4	00000001	0x5A	2

Abb. 20 – Out-Transaktion auf ein ausgelastetes Gerät (NAK)

Ist der Out-Endpoint längere Zeit nicht in der Lage, Daten zu empfangen, so wird er von der Firmware des Gerätes deaktiviert. Auf Daten sendet das Gerät dann ein STALL-Paket im Handshake zurück.

HOST	Idle	SYNC	OUT	ADDR	EP	CRC5	EOP
	4	00000001	0xE1	0x02	0x02	0x01	2

HOST	Idle	SYNC	Data0	DATA			CRC16	EOP
	3	00000001	0xC3	00 11 22 9D A5 FF CD 7F			0xCBA8	2

GERÄT	Idle	SYNC	STALL	EOP
	4	00000001	0x1E	2

Abb. 21 – Out-Transaktion auf einen deaktivierten Endpoint (STALL)

Im Vergleich dazu soll dieser Out-Transfer im USB 2.0- Protokoll dargestellt werden. Da man bei den langen Paketen leicht die Übersicht verliert, sind hier nur die Systeminformationen der Pakete dargestellt.

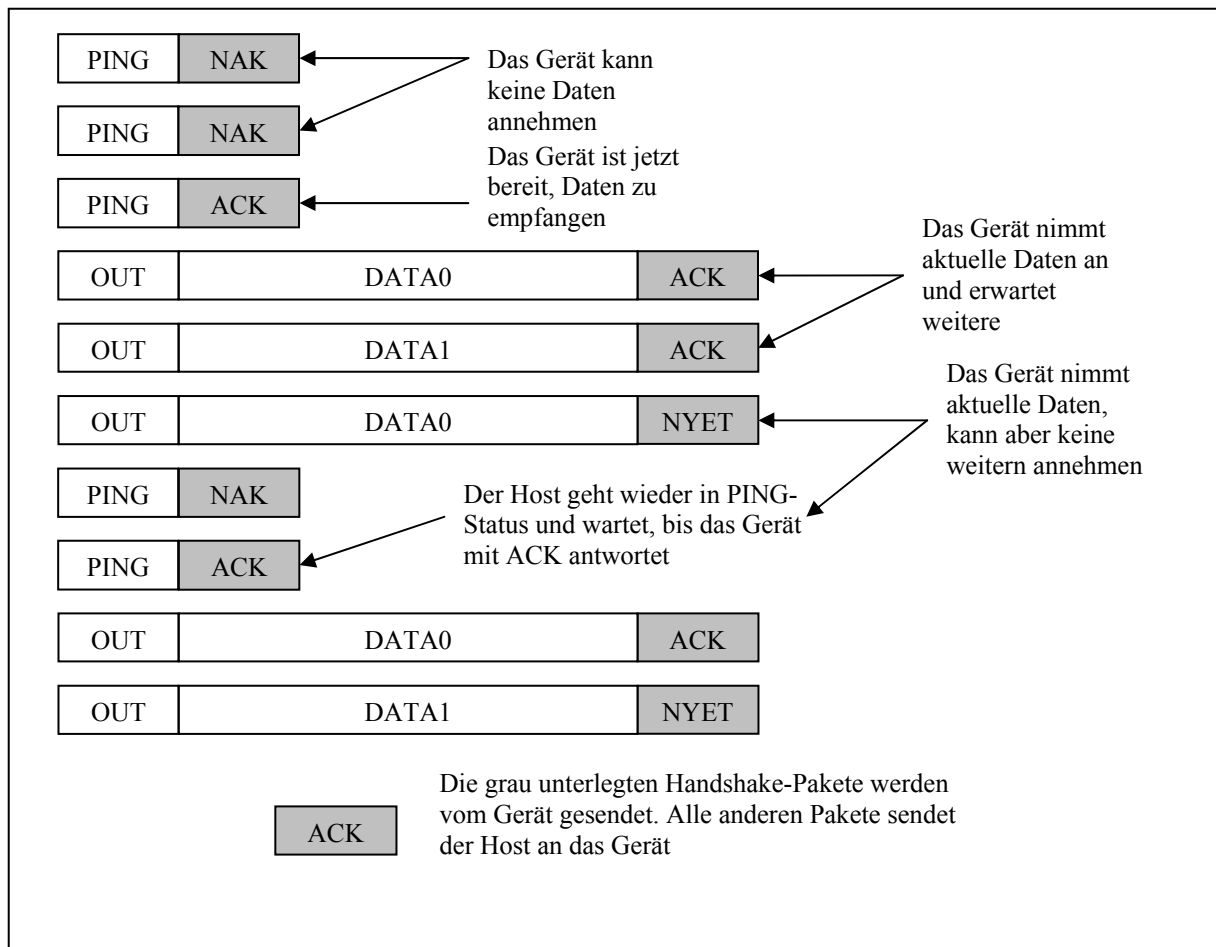


Abb. 22 – Out-Transaktionen im High-Speed-Modus

## 5.4 Fehlerbehandlung

Alle Datentransfers, außer dem Isochronous-Transfer, unterstützen ein Fehlerbehandlungsprotokoll. Diese Maßnahmen stellen sicher, dass Daten garantiert fehlerfrei übertragen werden können.

USB-fähige Module unterscheiden drei verschiedene Arten von Fehlern:

- PID-Fehler
- CRC-Fehler
- Bit-Stuff-Fehler

Sollte einer dieser Fehler auftreten, so sendet das Gerät kein Handshake-Paket und ignoriert die Daten. Der Host erkennt dies als Time-Out-Fehler und versucht erneut eine Datenübertragung.

### 5.4.1 PID-Fehler

In jedem Paket wird nach dem SYNC-Field ein PID (Package-Identifizier) gesendet, der das Paket beschreibt. Ein PID besteht aus 8 Bit. Das obere Nibble (4 Bit eines Byte) enthält die eindeutige Kodierung des Paketes. Das untere Nibble ist das Einerkomplement der oberen 4 Bit. Die SIE (Serial-Interface-Engine) des Gerätes prüft die zwei Nibbles gegeneinander.

Wenn dabei festgestellt wird, dass die beiden Hälften des Bytes nicht komplementär zueinander sind, wird eine interne Fehlermeldung für die Firmware des Gerätes ausgelöst. Die Reaktion auf diesen Fehler ist dabei je nach Firmware unterschiedlich.

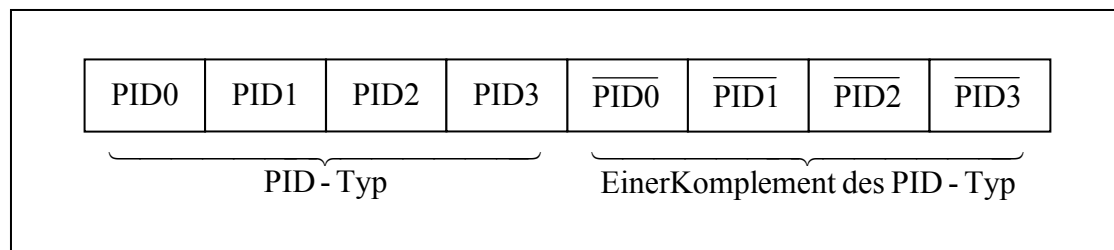


Abb. 23 – Aufbau des PID-Byte

### 5.4.2 CRC-Fehler

Die Daten aller Pakete, außer Handshake- und PRE-Paketen, werden durch eine CRC-Prüfsumme gesichert. CRC (Cyclic-Redundancy-Checking) schützt vor Einzel- und Doppelbitfehlern. Datenpakete nutzen eine 16 Bit CRC. Alle anderen Pakete werden durch eine 5 Bit CRC überwacht. Der PID wird nicht abgesichert.

Pakettyp	Größe	Geschützte Daten	CRC
SOF	11 Bit	11 Bit Framenummer	5 Bit CRC
IN	11 Bit	7 Bit Adresse und 4 Bit EP-Adr.	5 Bit CRC
OUT	11 Bit	7 Bit Adresse und 4 Bit EP-Adr.	5 Bit CRC
SETUP	11 Bit	7 Bit Adresse und 4 Bit EP-Adr.	5 Bit CRC
DATA0	Max. 1024 Byte	Nutzdaten	16 Bit CRC
DATA1	Max. 1024 Byte	Nutzdaten	16 Bit CRC
DATA2	Max. 1024 Byte	Nutzdaten	16 Bit CRC
MDATA	Max. 1024 Byte	Nutzdaten	16 Bit CRC

Tabelle 11 – CRC- geschützte Pakete

Die CRC-Prüfsumme wird von der SIE des sendenden USB-Teilnehmers während der Serialisierung errechnet und dem Paket hinzugefügt. Dies geschieht vor der NRZI-Kodierung. Die Auswertung der Prüfsumme erfolgt ebenfalls in Echtzeit. Die 5 Bit CRC nutzt folgendes Polynom:

$$G[X] = X^5 + X^2 + 1$$

Die 16 Bit CRC basiert auf folgendem Generatorpolynom:

$$G[X] = X^{16} + X^{15} + X^2 + 1$$

Weitere Informationen zum CRC-Algorithmus findet man in der Fachliteratur. Da der USB-Entwickler nichts an der CRC ändern kann und damit keinen Einfluss darauf hat, wird an dieser Stelle die CRC-Fehlerprüfung nicht detaillierter erläutert.

### 5.4.3 Bit-Stuff-Fehler

Durch den Bit-Stuffer im Sender wird nach sechs aufeinander folgenden Einsen eine Bit-Stuff-Null eingefügt. Ein Empfänger zählt die Anzahl der Einsen und erwartet nach sechs Einsen eine Null, die der Bit-Destuffer dann aus dem Datenstrom entfernt. Sollte diese Null nicht vorhanden sein, lehnt der Empfänger das Datenpaket ab.

### 5.4.4 Time-Out-Fehler

In fast jeder Transaktion hat der Empfänger eine Bestätigung an den Sender zu übertragen. Der Sender muss deshalb eine gewisse Zeit einräumen, die als Bus-Turn-Around-Time bezeichnet wird. Nach der USB-Spezifikation kann man diese Zeit berechnen. Mit einer Kabelverzögerung von 30 ns und einer Hub-Verzögerung von 40 ns kann bei einer Hub-Verzweigungstiefe von fünf eine maximale Signallaufzeit von 350 ns in einer Richtung auftreten. Da das Signal hin- und zurückläuft, muss mit 700 ns gerechnet werden. Zusätzlich wird dem Empfänger eine Bearbeitungszeit von 7,5 Taktzyklen eingeräumt. Daraus ergibt sich eine maximale Verzögerung von 1325 ns  $\approx$  16 Taktzyklen.

Der Sender muss also 16 Taktzyklen auf eine Antwort warten. Nach 18 Taktzyklen erkennt der Sender einen Time-Out-Fehler. Die Transaktion wird dann für ungültig erklärt.

## 6 Standarddescriptoren

### 6.1 USB-Konzept der Descriptoren

Für die Unterstützung eines Hot-Plug-and-Play-fähigen Busses ist es notwendig, die Geräte beim Anschließen zu erkennen und zu identifizieren. Die Beschreibung dieser Geräte ist mit den Descriptoren realisiert. Diese Descriptoren werden bei der Enumeration des Gerätes vom Host abgefragt. Descriptoren sind hierbei von der USB-Spezifikation fest vorgegebene Datenstrukturen, die der Host interpretieren kann. Es gibt für USB 1.1 fünf und für USB 2.0 sieben Standarddescriptoren. Dazu gehören auch Stringdeskrpitoren, deren Inhalte als Zeichenketten implementiert sind. Diese Descriptoren sind optional und dienen meist nur dem Betriebssystem zur Anzeige in der Systeminformation. Alle anderen Standarddescriptoren sind zwingend notwendig, und müssen deshalb immer implementiert werden. USB-Geräte können in verschiedene Geräteklassen eingeteilt werden. Für jede Klasse existieren zusätzlich sog. Klassenspezifische Descriptoren.

Descriptor typ	Code	Bemerkungen
Device-Descriptor	0x01	
Configuration-Descriptor	0x02	
String-Descriptor	0x03	
Interface-Descriptor	0x04	
Endpoint-Descriptor	0x05	
Device-Qualifier-Descriptor	0x06	Nur für USB 2.0
Other-Speed-Configuration-Descriptor	0x07	Nur für USB 2.0

Tabelle 12 – Die Standarddescriptoren

Die Descriptoren sind untereinander hierarchisch geordnet. Jedes Gerät besitzt genau einen Device-Descriptor, in dem die wichtigsten Eigenschaften gespeichert sind. Ein Gerät kann jedoch mehrere Konfigurationen unterstützen, wobei jeweils nur eine Konfiguration aktiv ist. Ebenso können mehrere Interfaces unterstützt werden. Dies ermöglicht die Herstellung von Geräten, die mehrere Funktionen gleichzeitig zur Verfügung stellen können. Die logischen Teilgeräte können dann immer mit einem vorgesehenen Treiber angesprochen werden. Dies ermöglicht den Einsatz von Standardtreibern im Betriebssystem.

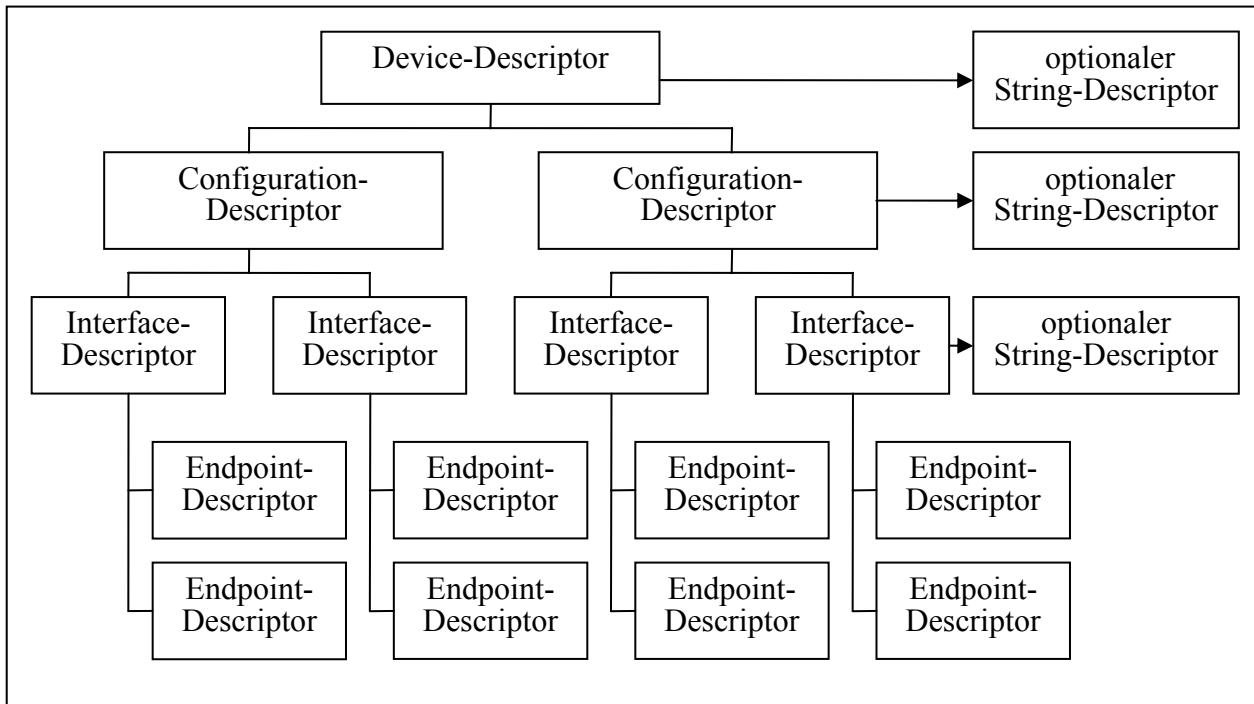


Abb. 24 – Hierarchiestruktur der Standarddescriptoren

## 6.2 Devce-Descriptor

Ein wichtiges Merkmal an den Descriptoren ist die Speicherung von 2-Byte-Werten. Diese wird im Little-Endian-Format vorgenommen. Dabei steht zuerst der Low-Teil und danach der High-Teil des Wertes.

Offset	Feldbezeichnung	Beschreibung	Beispielwerte
0	bLength	Größe des Descriptors in Byte	0x12
1	bDescriptorType	DescriptorTyp (Device)	0x01
2	bcdUSB(L)	Version der USB-Sezifikation	0x10
3	bcdUSB(H)		0x01
4	bDeviceClass	Klassen-Code	0xFF
5	bDeviceSubClass	Subklassen-Code	0xFF
6	bDeviceProtocol	Protokoll-Code	0xFF
7	bMaxPacketSize0()	Tiefe des EP0 in Byte	0x40
8	idVendor(L)	Vendor ID des Herstellers	0x47
9	idVendor(H)		0x05
10	idProduct(L)	Produkt ID	0x31
11	idProduct(H)		0x21
12	bcdDevice(L)	Releasenummer des Produktes	0x21
13	bcdDevice(H)		0x03
14	iManufacturer	String-Index des Herstellers	0x00
15	iProduct	String-Index des Produktes	0x00
16	iSerialNumber	String-Index der Seriennummer	0x00
17	bNumConfigurations	Anzahl möglicher Konfigurationen	0x00

Tabelle 13 – Devicedescriptor des Cypress AN 2131 USB-Mikrocontrollers

Der Device-Descriptor ist in jedem Gerät vorhanden und hat immer eine Länge von 18 Byte. Wichtig hierbei ist die Vendor-ID. Sie wird von der USB-Organisation vergeben. Jedes Mitglied des USB-Implementers-Forum erhält gegen eine Gebühr eine Vendor-ID, die eindeutig identifizierend ist. Geräte mit fremder Vendor-ID dürfen nicht auf dem Markt angeboten werden. Hersteller von Mikrocontrollern stellen ihre Vendor-ID zu Entwicklungszwecken zur Verfügung. Um eine Vendor-ID zu beantragen, werden Formulare auf den Internetseiten des Implementers-Forums angeboten ([www.usb.org](http://www.usb.org)). An letzter Stelle des Descriptors steht die Anzahl der unterstützten Konfigurationen. Dementsprechend sind genau so viele Configuration-Descriptoren abzufragen.

## 6.3 Configuration-Descriptor

Ebenso wie in jedem anderen Descriptor enthält auch der Configuration-Descriptor in den ersten Byte Länge und Typ des Descriptors. Der Configuration-Descriptor ist immer 9 Byte lang. Wird der Descriptor vom Host abgefragt, liefert das Gerät neben dem Configuration-Descriptor auch immer die unterstützten Interface-Descriptoren sowie deren klassenspezifische Descriptoren. Da diese Länge schwanken kann, ist es erforderlich, diese Angabe im Configuration-Descriptor zu verankern. Byte fünf enthält die aktuelle Konfigurationsnummer. Sie kann verschieden von Null sein. Wenn der Host das Gerät nicht konfigurieren

kann, setzt er es auf die Konfiguration Null. Ein Gerät darf nicht zwei Configuration-Descriptoren haben, welche die gleiche Konfigurationsnummer aufweisen. Byte sieben ist als Bitmap zu sehen, das verschiedene Attribute über Remote-Wakeup und den Powerzustand des Gerätes angibt.

Offset	Feldbeschreibung	Beschreibung	Beispielwert
0	bLength	Größe des Descriptors in Byte	0x09
1	bDescriptorType	DescriptorTyp (Configuration)	0x02
2	wTotalLength(L)	Gesamtlänge der Struktur mit allen Descriptoren	0xDA
3	wTotalLength(H)		0x00
4	bNumInterfaces	Anzahl der zugehörigen Interfaces	0x01
5	bConfigurationValue	Konfigurationsnummer	0x01
6	iConfiguration	String-Index für die Konfiguration	0x00
7	bmAttributes	Attribute für die Konfiguration	0x80
8	MaxPower	Stromaufnahme in 2 mA-Schritten	0x32

Tabelle 14 – Configuration-Descriptor

## 6.4 Interface-Descriptor

Dieser Descriptor kann vom Host nicht separat abgefragt werden. Er wird zusammen mit dem Configuration-Descriptor gesendet. Die Bezeichnungsnummerierung wird hier von Null ab hochgezählt. Besonderheit bei den Interface-Descriptoren ist die Nutzung verschiedener Alternate-Settings. Dies wird oftmals bei Geräten genutzt, die verschiedene Bandbreitenanforderungen haben. Im Stand-By-Modus hat ein Gerät beispielsweise eine isochrone Bandbreite von 0 Byte. Wird es aktiviert und soll Daten übertragen, so wechselt es die Alternate-Setting z. B. auf eins. Dann unterstützt das Gerät eine isochrone Bandbreite von 5 Mbit/s, je nach Einstellung der Endpoint-Descriptoren. Durch diese Vorgehensweise kann die Bandbreite des USB effektiver genutzt werden. Zu jedem Paar von Interfacenummer und Alternate-Setting gehört dann ein spezieller Satz Endpoint-Descriptoren, die sich in Anzahl und Bandbreite der Endpoints unterscheiden.

Offset	Feldbeschreibung	Beschreibung	Beispielwert
0	bLength	Größe des Descriptors in Byte	0x09
1	bDescriptorType	DescriptorTyp (Interface)	0x04
2	bInterfaceNumber	Interfacenummer	0x00
3	bAlternateSetting	Alternate-Setting	0x00
4	bNumEndpoints	Anzahl zugehöriger EP ohne EP0	0x01
5	bInterfaceClass	Klassen-Code	0xFF
6	bInterfaceSubClass	Subklassen-Code	0xFF
7	bInterfaceProtocol	Protocol-Code	0xFF
8	iInterface	String-Index für Interface	0x00

Tabelle 15 – Interface-Descriptor



## 6.5 Endpoint-Descriptor

Jeder Endpoint besitzt einen Endpoint-Descriptor. Ausgenommen wird dabei EP0, da dieser im Device-Descriptor ausreichend konfiguriert wird. Wie die Interface-Descriptoren werden auch die Endpoint-Descriptoren bei der Abfrage des Configuration-Descriptors an den Host gesendet. Eine gezielte Einzelabfrage ist somit nicht möglich.

Offset	Feldbeschreibung	Beschreibung	Beispielwert
0	bLength	Größe des Descriptors in Byte	0x07
1	bDescriptorType	DescriptorTyp (Endpoint)	0x05
2	bEndpointAdress	Endpointadresse und Richtung	0x81
3	bmAttributes	Transfer-Typ	0x03
4	wMaxPacketSize (L)	FIFO-Tiefe des EP in Byte	0x10
5	wMaxPacketSize (H)		0x00
6	bInterval	Polling-Intervall	0x0A

Tabelle 16 – Endpoint-Descriptor

Byte 2 enthält im unteren Nibble die Endpointnummer. Das Bit 7 ist das Richtungsbit. Ist es auf Eins gesetzt, ist der Endpoint in In-Richtung konfiguriert. Anderenfalls ist der Endpoint in Out-Richtung gesetzt. In Byte 3 ist in den unteren zwei Bits die Transferart kodiert.

Transferart	bmAttribute[1..0]
Control-Transfer	00
Isochronous-Transfer	01
Bulk-Transfer	10
Interrupt-Transfer	11

Tabelle 17 – Kodierung der Transferarten

Die FIFO-Tiefe wird in den Bytes 4 und 5 angegeben. Da isochrone Endpoints bis zu 1024 Bytes tief sind, müssen zwei Byte reserviert werden. Byte 6 enthält das Polling-Intervall in 1 ms-Einheiten. Diese Angabe wird nur bei Interrupt-Transfer benötigt. Für andere Transferarten ist diese Angabe nicht relevant, und soll auf Null gesetzt werden.

Bei USB 2.0 wurde bmAttributes weiter kodiert, um zusätzliche Informationen an den Host zu übermitteln. Für die Beschreibung der Synchronisation isochroner Endpoints nutzt man die Bits zwei und drei. Für andere Transferarten sind diese Null zu setzen.

Synchronisation	bmAttribute[3..2]
Keine Synchronisation	00
Asynchron	01
Adaptiv	10
Synchron	11

Tabelle 18 – Synchronisationsattribute bei USB 2.0

Durch die Bits vier und fünf wird die Nutzung des Endpoints beschrieben. Angemerkt sollte an dieser Stelle werden, dass isochrone Endpoints, die auf Feedback eingestellt sind, keine Synchronisation unterstützen dürfen.

Endpoint-Nutzung	bmAttribute[5..4]
Daten-Endpoint	00
Expliziter Feedback-Endpoint	01
Impliziter Feedback-Daten-Endpoint	10
Reserviert	11

Tabelle 19 – Nutzungseinstellungen für Endpoints bei USB 2.0

Feedback-Endpoints arbeiten mit Daten-Endpoints zusammen und bilden eine Rückmeldung. Gleichzeitig können über einen Feedback-Endpoint mehrere Daten-Endpoints kontrolliert und synchronisiert werden.

Für USB 2.0 wurde auch das Byte bInterval erweitert. Je nach Transferart und Geschwindigkeitsklasse wird dieses Byte unterschiedlich interpretiert. Bei High-Speed-Geräten wird der Wert als Exponent zur Basis 2 genutzt, um für Interrupt- und Isochronous-Transfer die Polling-Rate anzugeben. Folgende Formel wird dazu angewendet:

$$\text{Polling - Intervall in ms : } t = 2^{bInterval-1} ; bInterval = \{ 1..16 \}$$

Für Bulk- und Control-Transfer wird in bInterval die maximal zulässige Anzahl von NAK-Paketen innerhalb eines Microframe angegeben. Der Wert kann dabei von 0 bis 255 variieren.

Transferart	Wert für Low-/Full-Speed	Wert für High-Speed
Control	0	NAK-Rate (0..255)
Bulk	0	NAK-Rate (0..255)
Interrupt	Polling-Intervall in ms (0..255)	Polling-Intervall als Exponent von 2 (1..16)
Isochronous	1	Polling-Intervall als Exponent von 2 (1..16)

Tabelle 20 – Kodierung des Polling-Intervalls

Im dem Datenwort wMaxPacketSize ist auf den unteren 10 Bits der Wert für die FIFO-Tiefe angegeben. Die Bits 11 und 12 dienen einer weiteren zusätzlichen Kodierung. Sie beinhalten die Anzahl der Transaktionen pro Microframe. Aus der Angabe ist zu entnehmen, dass dieser Wert nur für USB 2.0 gültig ist, da nur der High-Speed-Modus über Microframes verfügt. Diese Angaben beziehen sich auch nur auf den Interrupt- und den Isochronous-Transfer. Für Bulk-Transfer ist der Wert auf 00 zu setzen. Abhängig von der Anzahl der Transaktionen pro Microframe ist auch die minimale FIFO-Tiefe.

Anzahl der Transaktionen	wMaxPacketSize[12..11]	wMaxPacketSize[10...0]
1	00	0-1024
2	01	513-1024
3	10	683-1024
Reserviert	11	

Tabelle 21 – Kodierung der Transaktionen in wMaxPacketSize

## 6.6 String-Descriptor

Um Textbeschreibungen wie z.B. Seriennummer, Hersteller, Gerätenummer, etc. zu übertragen, wird der String-Descriptor genutzt. Es ist nicht zwingend vorgeschrieben, diese Angaben bereitzustellen, jedoch ist es für die Arbeit im Windows-System eine große Erleichterung. Ein String-Descriptor enthält als Text nur Unicode (UCS-2). Um Spracheinstellungen zu setzen, kann der Descriptor auch eine Language-ID in den Bytes zwei und drei enthalten. Demnach ist eine 2-Byte-Kodierung verwendet. Beim Arbeiten mit dem englischen ASCII- Zeichensatz wird vor jedes Zeichen einfach eine 0x00 gesetzt. Die Little-Endian-Notierung vertauscht die zwei Byte des Zeichens. Der Descriptor ermöglicht die Übertragung von 126 Zeichen. Wenn keine String-Descriptoren unterstützt werden, dann ist in allen anderen Descriptorfeldern, die auf Strings verweisen, der Wert 0x00 einzutragen. Zeigt ein Descriptor auf einen nicht vorhandenen String-Descriptor, so wird die Enumeration abgebrochen.

Offset	Feldbeschreibung	Beschreibung	Beispielwert
0	bLength	Größe des Descriptors in Byte	0x0A
1	bDescriptorType	DescriptorTyp (String)	0x04
2	wUnicode(L)	Unicode-Zeichen „I“	0x49
3	wUnicode(H)		0x00
4	wUnicode(L)	Unicode-Zeichen „M“	0x4D
5	wUnicode(H)		0x00
6	wUnicode(L)	Unicode-Zeichen „M“	0x4D
7	wUnicode(H)		0x00
8	wUnicode(L)	Unicode-Zeichen „S“	0x53
9	wUnicode(H)		0x00

Tabelle 22 – Aufbau des String-Descriptors

## 6.7 Device-Qualifier-Descriptor

Dieser Descriptor ist nur bei High-Speed-Geräten zu finden. Er wird im Full-Speed-Modus abgefragt und beschreibt das Verhalten des Gerätes im High-Speed-Modus. Dadurch kann der Host entscheiden, ob er das Gerät in den High-Speed-Modus setzt oder es weiter im Full-Speed-Betrieb anspricht. Dieser Descriptor ist dem Device-Descriptor ähnlich. Für USB 2.0 ist aber zwingend, dass der Wert bcdUSB mindestens 0x0200 sein muss. Dies bedeutet, dass das Gerät USB 2.0 unterstützt. Auch das Byte bMaxPacketSize0 muss auf 0x40 (64 Dez.) gesetzt werden, da die USB 2.0 Spezifikation vorschreibt, dass die Endpoint-Tiefe des EP0 genau 64 Byte betragen muss.

Offset	Feldbeschreibung	Beschreibung	Beispielwert
0	bLength	Größe des Descriptors in Byte	0x0A
1	bDescriptorType	DescriptorTyp (Dev.-Qual.)	0x06
2	bcdUSB(L)	Version der USB-Spezifikation	0x00
3	bcdUSB(H)		0x02
4	bDeviceClass	Klassen-Code	0xFF
5	bDeviceSubClass	Subklassen-Code	0xFF
6	bDeviceProtocol	Protokoll-Code	0xFF
7	bMaxPacketSize0	Tiefe des EP0 in Byte	0x40
8	bNumConfigurations	Anzahl möglicher Konfigurationen	0x01
9	bReserved	Reserviert (zwingend 0x00)	0x00

Tabelle 23 – Device-Qualifier-Descriptor

## 6.8 Other-Speed-Configuration-Descriptor

Der Other-Speed-Configuration-Descriptor beschreibt die Konfiguration eines High-Speed-Gerätes, wenn es im High-Speed-Modus arbeitet. Der Descriptor ist vom gleichen Aufbau wie der Configuration-Descriptor, lediglich der Wert des DescriptorTyps ist zwingend unterschiedlich. Wenn ein USB 2.0-fähiges Gerät keinen High-Speed-Modus unterstützt, so muss es auf die Requests des Hosts nach den zwei USB 2.0-spezifischen Descriptoren mit einem STALL-Paket antworten.

Offset	Feldbeschreibung	Beschreibung	Beispielwert
0	bLength	Größe des Descriptors in Byte	0x09
1	bDescriptorType	DescriptorTyp (Other-Speed-Conf.)	0x07
2	wTotalLength(L)	Gesamtlänge der Struktur mit allen Descriptoren	0x22
3	wTotalLength(H)		0x00
4	bNumInterfaces	Anzahl der zugehörigen Interfaces	0x01
5	bConfigurationValue	Konfigurationsnummer	0x01
6	iConfiguration	String-Index für die Konfiguration	0x04
7	bmAttributes	Attribute für die Konfiguration	0xA0
8	MaxPower	Stromaufnahme in 2 mA-Schritten	0x32

Tabelle 24 – Other-Speed-Configuration-Descriptor

## 7 Standard-Device-Requests

Die USB-Device-Requests enthalten als Teilmenge die Standard-Device-Requests. Auf diese Requests muss das Gerät immer reagieren. Die Kommunikation erfolgt über den bidirektionalen Endpoint 0 im Control-Transfer. Die Standard-Device-Requests sind in allen USB-Geräten implementiert. Sie dienen zur Abfrage der Descriptoren und der Konfiguration des Gerätes. Sollte ein Gerät einen Request bzw. die damit angeforderte Konfiguration nicht unterstützen, so sendet es ein Stall-Paket zurück. USB-Device-Requests sind wie folgt aufgebaut:

Offset	Feldbezeichnung	Belegung
0	bmRequestType	Bitmap
1	bRequest	Byte
2	wValue(L)	16-Bit Wort
3	wValue(H)	
4	wIndex(L)	16-Bit Wort
5	wIndex(H)	
6	wLength(L)	16-Bit Wort
7	wLength(H)	

Tabelle 25 – Kodierung des USB-Device-Requests

Das Bitmap bmRequestType enthält alle Informationen des Requesttyps und wird folgendermaßen kodiert:

Bit-Nummer	Bedeutung	Werte
7	Richtung des Datentransfers	0: Host → Gerät
		1: Gerät → Host
6,5	Requesttyp	00: Standard-Request
		01: Klassenspezifischer Request
		10: Vendorspezifischer Request
		11: Reserviert
40	Empfänger des Requests (Alle anderen Wertkombinationen sind reserviert)	00000: Device
		00001: Interface
		00010: Endpoint
		00011: Andere

Tabelle 26 – Kodierung des Bitmap bmRequestType

Insgesamt existieren 11 Standard-Device-Requests. Diese können wiederum in verschiedenen Ebenen des Gerätes wirken (Device, Interface, und Endpoint).

Request	bRequest	Wert in bRequest	Unterstützung
GetStatus	GET_STATUS	0x00	Immer
ClearFeature	CLEAR_FEATURE	0x01	Immer
SetFeature	SET_FEATURE	0x03	Immer
SetAddress	SET_ADDRESS	0x05	Immer
GetDescriptor	GET_DESCRIPTOR	0x06	Immer
SetDescriptor	SET_DESCRIPTOR	0x07	Optional
GetConfiguration	GET_CONFIGURATION	0x08	Immer
SetConfiguration	SET_CONFIGURATION	0x09	Immer
GetInterface	GET_INTERFACE	0x0A	Immer
SetInterface	SET_INTERFACE	0x0B	Immer
SynchFrame	SYNCH_FRAME	0x0C	Optional

Tabelle 27 – Die Standard-Device-Requests

## 7.1 GetStatus

Wenn der Host den Status des Gerätes abfragen muss, nutzt er diesen Request. Dabei kann er den Device-Status, den Interface-Status und den Endpoint-Status abfragen. Alle drei Requests antworten mit einem 16-Bit-Wort, das entsprechend interpretiert wird. Bei der Device-Abfrage steht das Bit 0 für den Self-Powered-Status. Null gibt an, dass das Gerät über den Bus versorgt wird. Eins steht für eine externe Stromversorgung. Notwendig ist diese Angabe nur bei Geräten, die beide Stromversorgungsarten unterstützen können. Bit 1 gibt den Remote-Wakeup-Status an. Ist dieser Wert auf Eins gesetzt, ist das Gerät in der Lage, einen Bus im Suspend-Zustand zu aktivieren. Nach einem Bus-Reset ist der Wert immer Null und kann bei der Enumeration des Gerätes mit SetFeature gesetzt bzw. mit ClearFeature gelöscht werden. Anzugeben ist hierbei der sogenannte Feature-Selektor Device\_Remote\_Wakeup. Die Bits 2..15 sind reserviert. Bei der Interface-Abfrage sind alle Bits reserviert und auf Null gesetzt und somit für den Entwickler uninteressant. Die Endpoint-Abfrage ist nur im Bit 0 relevant, dass angibt, ob der Endpoint aktiviert ist. Beim Wert Eins ist der Endpoint deaktiviert. Eine Beeinflussung dieses Status ist mit SetFeature oder ClearFeature und dem Feature-Selektor Endpoint\_Stall möglich.

## 7.2 SetFeature und GetFeature

Mit diesen beiden Requests lassen sich bestimmte Eigenschaften des Gerätes aktivieren oder deaktivieren. Für die Standard-Device-Requests sind bislang nur zwei Features definiert. Dies ist zum einen das Device\_Remote\_Wakeup mit dem Wert 0x0001 und zum anderen Endpoint\_Stall mit 0x0000. Der Feature-Selektor wird im Feld wValue übertragen. Zusätzlich unterscheidet bRequest zwischen den beiden Anforderungen. Der Wert 0x01 ist für das ClearFeature und 0x03 für SetFeature vorgesehen. Bei der Übertragung ist als Antwort ein leeres Datenpaket, ein sogenanntes Zero-Data-Packet, für die erfolgreiche Erkennung der Requests vorgesehen. Alle anderen Pakettypen und Inhalte werden vom Host als Fehler interpretiert.

## 7.3 SetAddress

Nach einem Reset ist jedem Gerät die Adresse 0x00 zugewiesen. Erst in der Enumeration erhält das Gerät seine Busadresse. Diese wird mit SetAddress übergeben. Die neue Adresse ist im Low-Teil von wValue abgelegt. Um den SetAddress-Request zu übertragen, wird es auf der Adresse 0x00 angesprochen. Als Antwort erwartet der Host ein Zero-Data-Paket. Anschließend kann das Gerät mit seiner neuen Adresse angesprochen werden.

## 7.4 GetDescriptor

Alle im USB-Gerät vorhandenen Descriptoren werden mit GetDescriptor angefordert. Dabei können Device-, Device-Qualifier-, Configuration-, Other-Configuration- und String-Descriptoren abgefragt werden. Alle anderen werden zusammen mit dem Configuration-Descriptor übermittelt. Der High-Teil von wValue enthält dabei den Descriptortyp. 0x01 steht beispielsweise für den Device-Descriptor. Für die verschiedenen Konfigurationen wird der Low-Teil von wValue genutzt. Wenn mehrere Configuration-Descriptoren vorhanden sind, dann wird wValue von 0x00 an bis n-1 inkrementiert.

Wenn ein String-Descriptor abgefragt wird, enthält der Low-Teil von wValue den Index des Strings. Im wLength ist angegeben, wie viele Zeichen der Host empfangen kann. Ist die Zeichenkette länger, muss die Firmware eine Teilung des Strings vornehmen und dementsprechend mehrere Pakete übertragen.

## **7.5 SetDescriptor**

Dieser optionale Descriptor wird von Geräten unterstützt, die nachträgliche Änderungen von Descriptoren oder die Einrichtung zusätzlicher Descriptoren ermöglichen. Der Aufbau dieses Descriptors ist dem GetDescriptor ähnlich. Allerdings werden hier keine Daten mit In-Transaktionen vom Gerät gesendet, sondern der Descriptor wird in einem oder mehreren Out-Transaktionen vom Host zum Gerät übertragen.

## **7.6 GetConfiguration**

Mit diesen Request kann die aktuelle Konfiguration des Gerätes abgefragt werden. Es wird ein Byte bConfigurationValue zurückgeliefert, das die Konfiguration enthält. Ist der Wert 0x00, befindet sich das Gerät im unkonfigurierten Zustand.

## **7.7 SetConfiguration**

Um dem Gerät eine Konfiguration zuzuweisen, wird dieser Request benutzt. Vorher muss der Host alle Device- und Configuration-Descriptoren abgefragt haben. Die Konfiguration wird im Low-Teil des Feldes wValue gesendet. Im unkonfigurierten Zustand darf nur der EP0 aktiviert sein und die Stromaufnahme ist auf 100 mA begrenzt. Wenn die neue Konfiguration aktiviert ist, können auch andere Endpoints angesprochen werden. Die Stromaufnahme ist dann variabel einstellbar.

## **7.8 GetInterface und SetInterface**

In einer Konfiguration können mehrere Interfaces unterstützt werden, die wiederum verschiedene Alternate-Settings besitzen können. Mit GetInterface kann die aktuelle Alternate-Setting abgefragt werden. Dazu muss im Low-Teil von wIndex das abzufragende Interface angegeben sein. Als Antwort erhält der Host ein Byte, in dem die aktuelle Alternate-Setting enthalten ist. Mit SetInterface kann man das Interface und die Alternate-Setting einstellen. Im Low-Teil von wValue wird die Alternate-Setting eingetragen. Der Request enthält das ausgewählte Interface im Low-Teil von wIndex.

## **7.9 SynchFrame**

Dieser Request ist optional und wird nur von isochronen Endpoints genutzt, die eine Framesynchronisation haben. Die Endpoint-Adresse ist im Low-Teil von wIndex enthalten. Sollte der Endpoint diese Synchronisation nicht unterstützen, sendet er ein STALL-Paket an den Host zurück.

## 8 Enumeration

Unter Enumeration versteht man den Ablauf der Identifizierung und Konfiguration eines Geräts, wenn es an den Bus angeschlossen wird. Der Hub erkennt den Anschluss eines neuen Geräts und informiert den Host, der dann die Enummerierung vornimmt. Dieser Prozess findet auch nach dem Einschalten des Computers statt. Der gesamte Bus wird dann in einer festgelegten Reihenfolge enumeriert. Dabei gilt folgender genereller Ablauf der Enumeration:

- Der Hub meldet dem Host auf einer Interrupt-Pipe, dass ein neues Gerät angeschlossen wurde. Auf den Versorgungsleitungen ist immer Spannung vorhanden. Der Hub erkennt durch die Überwachung seiner Downstream-Ports, wo das Gerät angesteckt wurde und mit welcher Geschwindigkeit es kommunizieren kann. High-Speed-Geräte starten immer mit der Full-Speed-Geschwindigkeit und werden erst im Bedarfsfall auf USB 2.0 umgeschaltet. Der Hub führt dabei noch keine Kommunikation mit dem Gerät.
- Der Host wird durch den Hub benachrichtigt und sendet an den Control-Endpoint des Hubs klassenspezifische Requests, die der Hub beantwortet, um dem Host nähere Informationen über das Gerät zu liefern. Dieser veranlasst den Hub, den Port für die Dauer von 10 ms in den Reset-Zustand zu setzen.
- Der Host öffnet jetzt mit Hilfe des Hubs eine Verbindung zum Gerät. Das Gerät ist nach dem Reset immer im Default-Zustand. Dabei sollte nur der Endpoint 0 aktiviert sein und die Stromaufnahme darf nicht mehr als 100 mA betragen. Er fragt den 18 Byte langen Device-Descriptor ab. Der Host wird die Transaktion aber nach den ersten 8 Byte abbrechen und dem Gerät eine Quittierung senden. Theoretisch sollte das Gerät auf diesen vorzeitigen Abbruch richtig reagieren können. Aus Sicherheitsgründen wird das Gerät aber in den Reset-Zustand geschaltet. Diese Phase ist als erster Kommunikationsversuch zwischen Host und Gerät anzusehen. Der Vorgang beginnt nun für das Gerät von Neuem. Der Host hat sich den ersten Kommunikationsversuch gemerkt und wird weitere Konfigurationsmaßnahmen durchführen.
- Nach Beendigung des Reset-Zustands, bekommt das Gerät vom Host sofort eine Adresse zugewiesen. Diese kann bei jedem Enummerationsprozess unterschiedlich sein. Alle weiteren Übertragungen werden mit der neuen Adressierung durchgeführt. Das Gerät befindet sich jetzt im Adress-Zustand.
- Der Host fragt nun mit den Standard-Device-Requests die verschiedenen Descriptoren ab. Als erstes wird der Device-Descriptor übertragen. Die Configuration-Deskriptoren werden mehrmals abgefragt. Beim ersten Empfangen wird der Host nur die 9 Byte des jeweiligen Configuration-Descriptors betrachten. Weitere anhängende Interface- und Endpoint-Descriptoren werden in der zweiten Auswertung berücksichtigt.
- Der Host wertet jetzt die Vendor-ID und die Produkt-ID des Device-Descriptors aus und sucht in der Treiber-Datenbank nach einem Treiber. Bleibt die Suche erfolglos, sucht das System anhand des Klassen-Codes, des Subklassen-Codes und des Protokoll-Codes nach einem klassenspezifischen oder generischen Treiber. Sollte auch hier kein passender Gerätetreiber gefunden werden, erscheint auf dem Bildschirm des Computers ein Fenster und das Betriebssystem fordert einen Treiber für das Gerät an.



- Nachdem der Host alle Konfigurationen des Geräts kennt, überprüft er die Anforderungen an die Bandbreite und den Strombedarf des Busses. Der Gerätetreiber wählt eine Konfiguration an, die der Host nochmals prüft und dann einstellt. Das Gerät ist jetzt im Configured-Zustand. Sollte der Host eine Konfiguration nicht unterstützen können, sucht er nach einer möglichen Alternativkonfiguration. Bietet das Gerät diese Option nicht an, so veranlasst der Host den Hub zur Sperrung des Ports. Für Kombigeräte, die mehrere Applikationen anbieten, versucht der Host für jede einzelne Funktion einen Treiber zu finden. Das Gerät ist nun betriebsbereit und kann verwendet werden.

## 8.1 Enummerierung von Hubs

Einen Hub erkennt der Host genauso als Gerät. Die Enummerierung erfolgt hierbei nach dem gleichen Schema. Erst wenn der Hub betriebsbereit ist, werden auch angeschlossene Geräte enummeriert.

## 8.2 Entfernen eines Gerätes

Wenn ein Gerät vom Bus abgezogen wird, muss der Hub den Port für eine bestimmte Zeitdauer sperren. Dabei wird auch der Host informiert. Der Hosttreiber veranlasst das Betriebssystem, den Gerätetreiber aus der Verwaltung zu entfernen. Der Treiber wird damit geschlossen, ist aber weiterhin in der Datenbank verfügbar. Ein erneutes Anstecken des Geräts bewirkt, dass der Gerätetreiber wieder geladen wird.

## 9 USB-Host-Controller

Der Host-Controller ist der Master auf dem USB. Er ist zuständig für die Generierung von Übertragungen. Diese legt er in Transferdescriptoren im Speicher ab. Transferdescriptoren sind dabei festgelegte Strukturen, die folgende Informationen enthalten:

- Adresse des USB-Gerätes
- Transferart (Control, Interrupt, Bulk oder Isochronous)
- Transferrichtung
- Zeigeradresse auf den Pufferspeicher des Gerätetreibers.

Die Zeigeradresse markiert dabei den Anfang des Speicherinhaltes für bereitgestellte Out- oder Setup-Transaktionen. Zu empfangende Daten aus In-Transaktionen werden ab dieser Adresse in den Speicher geschrieben.

Wie bereits in Kapitel 2.5.1 erwähnt, existieren für Low- und Full-Speed zwei Controllervarianten: Der UHCI von Intel und der OHCI von Microsoft, Compaq und National Semiconductor. Für USB 2.0 gibt es den EHCI. Dieser arbeitet parallel zum OHCI oder UHCI. Man findet mehr OHCI-Hosts als UHCI. Wichtige Unterschiede der beiden Varianten sollen in den dazugehörigen Kapiteln verdeutlicht werden.

## 9.1 Universal-Host-Controller-Interface (UHCI)

Die Übertragungsreihenfolge der Daten für UHCI ist folgendermaßen spezifiziert.

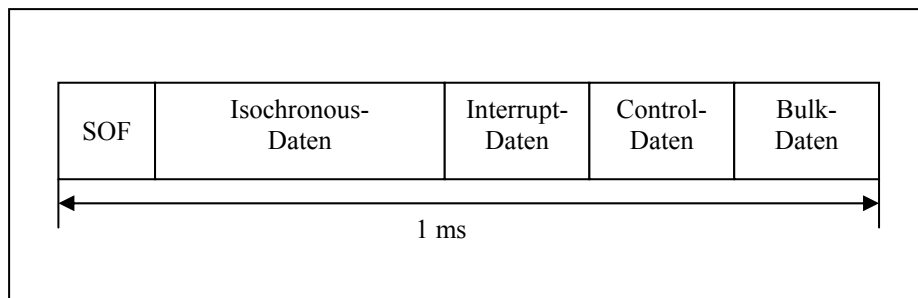


Abb. 25 – Anordnung der Transferarten im Frame bei UHCI

Isochronous- und Interrupt-Transfer haben bis zu 90% garantierte Bandbreite. Der Control-Transfer kann bis zu 10% garantierte Bandbreite nutzen. Bulk-Transfers bleibt nur der Rest möglicherweise ungenutzter Bandbreite.

Die aufeinander folgenden Frames werden von der Software des Hosts zu einer Frameliste zusammengesetzt. In der sind nur Speicheradressen des Framedatenstroms enthalten. Der Zugriff auf die Frameliste funktioniert folgendermaßen:

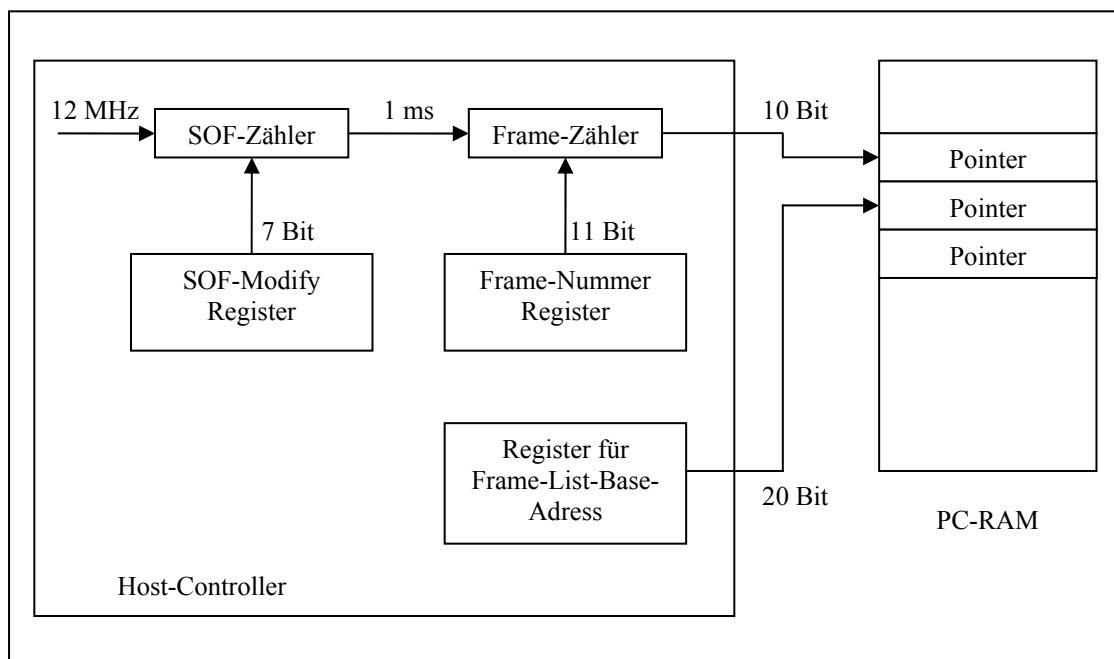


Abb. 26 – Organisation der Frameliste

Im Register der Frame-List-Base-Adresse ist die zugeteilte Basisadresse für den Speicherzugriff enthalten. Die aktuelle Adresse wird aus dem Register und dem Framezähler zusammengesetzt. Um bei isochronen Transfers die Synchronisation über die Framezeit zu erreichen, kann im SOF-Modify-Register der Wert verändert werden. Diese Zahl wird im Betriebstakt dekrementiert, so dass im Normalfall der Framezähler im 1 ms Abstand inkrementiert wird.

Die Frame-Pointer im Systemspeicher verweisen auf Felder von Transfer-Deskriptoren. Diese bestehen aus vier 32-Bit (Double-Word) Speicherzellen und enthalten alle relevanten Transfer- und Transaktionsinformationen. Diese Transfer-Deskriptoren werden nacheinander

abgearbeitet und auf dem Bus gesendet oder dementsprechende Daten empfangen. Man unterscheidet bei der Abfolge der Descriptoren je nach Transfertype zwischen der horizontalen und vertikalen Arbeitsrichtung. Die Transfer-Descriptoren der vier verschiedenen Transferarten werden mit Queue-Heads verbunden. Das sind Datenstrukturen, die Zeiger auf die nächsten Transfer-Descriptoren und zusätzliche Statusinformationen enthalten.

Für die Steuerung des Hosts durch das Betriebssystem werden Steuerregister in den PCI-Adressraum eingeblendet.

## 9.2 Open-Host-Controller-Interface (OHCI)

Die Übertragungsreihenfolge der Transferarten weicht vom Konzept des UHCI ab. Eine klassische Einteilung in die vier Transfertypen ist hier nicht vorzufinden. Der OHCI teilt die Daten in periodische und nichtperiodische Übertragungen ein. Nach einem SOF beginnt der Host zuerst mit der Verarbeitung nichtperiodischer Daten. Das sind Control- und Bulk-Transfers. Der Host-Controller legt vorher eine maximale Zeitdauer für diese Übertragungen fest. Der typische Wert dafür sind 10% der Framedauer. Anschließend setzt der Host das Frame mit der Übertragung von periodischen Daten fort. Diese bestehen aus Interrupt- und Isochronous-Transfers. Noch vorhandene Bandbreite nutzt der Host wieder für die nichtperiodischen Daten.

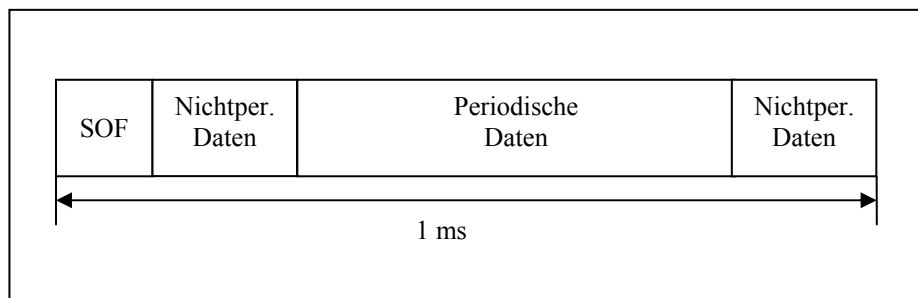


Abb. 27 – Übertragungsreihenfolge des OHCI

Der Transfermechanismus ist beim OHCI anders aufgebaut als beim UHCI. Das Framelisting ist im gleichen Schema realisiert wie für UHCI und OHCI in Abbildung 26 beschrieben. OHCI nutzt Endpoint-Descriptoren und Transfer-Descriptoren für den Übertragungsmechanismus. Der Host generiert zu jedem verfügbaren Endpoint einen eigenen Endpoint-Descriptor. Um Verwechslungen zu vermeiden, muss strikt zwischen dem Endpoint-Descriptor des Gerätes und dem des Hosts unterschieden werden. Die Endpoint-Descriptoren des Host enthalten folgende Informationen:

- Adresse des Geräte-Endpoints
- Richtung des Endpoints
- Maximale Paketgröße
- Geschwindigkeitsunterscheidung für Low- und Full-Speed
- Statusinformationen

Alle Endpoint-Descriptoren werden in einer Liste zusammengefasst. Jedem Endpoint der Liste werden Transfer-Descriptoren zugeordnet, die entsprechende Transferinformationen beinhalten. Damit ergibt sich folgende Listenstruktur:

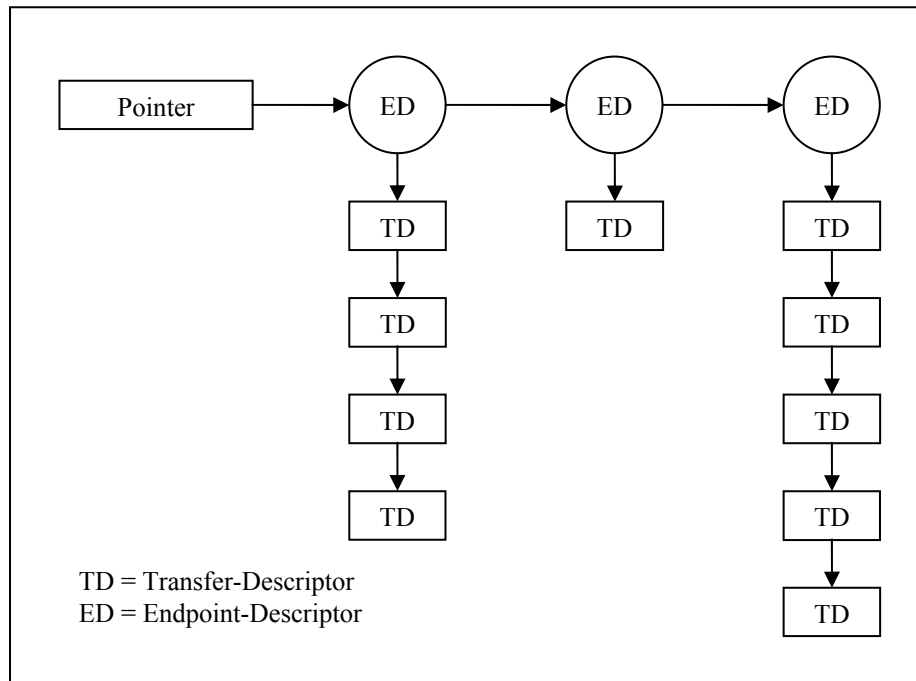


Abb.28 – Listenstruktur des OHCI

### 9.3 Enhanced-Host-Controller

Der EHCI ist für den High-Speed-Modus zuständig. Er ist in USB 2.0-Contollern parallel zum bisher verwendeten UHCI oder OHCI implementiert und besitzt darüber hinaus einen eigenen USB 2.0-Root-Hub. Eine Routing-Logik sorgt für den fehlerfreien Datenverkehr in den verschiedenen Geschwindigkeitsstufen. Die Unterstützung für USB benötigt auch ein Betriebssystem, das USB 2.0-fähig ist. Microsoft bietet bisher nur Windows 2000 an, bei dem USB 2.0 unterstützt wird. Für die Betriebssysteme Windows ME und Windows XP sind zusätzliche Treiber notwendig, um USB 2.0 verwenden zu können. Nach dieser Unterstützung richtet sich die Routing-Strategie des Host-Contollers.

Im PCI-Adressraum verfügt der Host-Controller über ein Bit im Configure-Flag-Register für die Erkennung. Sollte das Betriebssystem die USB 2.0-Unterstützung haben, setzt es das Bit. Der Host verbindet dann alle Geräte mit dem USB 2.0-Root-Hub. Geräte, die mit einer niedrigeren Geschwindigkeit arbeiten, werden nachträglich mit dem USB 1.1-Root-Hub verbunden. Ist das Bit nicht gesetzt, werden alle Ports mit angeschlossenen Geräten mit dem USB 1.1-Controller koordiniert. Alle USB 2.0-fähigen Geräte sind auch in der Lage, mit Full-Speed zu kommunizieren. Sie werden vorerst vom Full-Speed-Controller enumeriert. Wenn ein High-Speed-Treiber vorhanden ist, wird der Port, an dem das Gerät angeschlossen ist, an den EHCI umgeschaltet. Für den Test von EHCI hat Intel spezielle Kompatibilitätsprogramme im Implementers-Forum bereitgestellt. Die Funktionsweise des Hosts spielt bei der Entwicklung von USB-Anwendungs- und Messgeräten eine untergeordnete Rolle. Als weiterführende Literatur dienen die Quellen /1/, /2/ und /3/.

## 10 USB-Klassen

Geräte oder Interfaces, die gleiche oder ähnliche Eigenschaften haben, können zu USB-Klassen zusammengefasst werden. Wenn die Geräte einer Geräteklasse angehören, können sie mit einem allgemeinen Klassentreiber angesprochen werden. Dies erleichtert die Kommuni-

kation für Entwickler von Klassengeräten, zu denen man fast alle handelsüblichen Geräte zuordnen kann. Das Betriebssystem muss lediglich einen Treiber für eine ganze Klasse bereitstellen. Um die Klassenspezifikation zu erarbeiten, ist eine erfolgreiche Zusammenarbeit von Geräteherstellern und Betriebssystemanbietern notwendig.

Die Mitgliedschaft im Implementers-Forum der USB-Organisation ist dazu Voraussetzung. Auf der dazugehörigen Webseite findet sich auch die aktuelle Klassenliste, die alle bis jetzt existierenden Klassen und Subklassen beinhaltet. Da es unwahrscheinlich ist, dass ein Entwickler eine eigene USB-Klasse erstellt, soll hier auf die Nutzung vorhandener Klassenspezifikationen hingewiesen werden.

Neben den in der USB-Spezifikation enthaltenen Descriptoren und Requesets unterstützen die Klassen jeweilige zusätzliche klassenspezifische Descriptoren und Requests. Der USB-Entwickler kann die Klasseneigenschaften gezielt nutzen, um sein Gerät mit dem Klassentreiber anzusprechen. Das erspart dem Entwickler, eigene Treiber für sein Gerät programmieren zu müssen. Wichtigste Klasse ist die HID-Class (Human-Interface-Device). Alle vom PC-Anwender gesteuerten Eingabegeräte gehören zu dieser Klasse. Dazu zählen Geräte wie Maus, Tastatur, Schalter, Regler, Joysticks, Barcode-Scanner und kleinere Messgeräte. Unter /3/ und /4/ sind Beispiele mit klassenspezifischen Geräten und Treibern enthalten.

## **11 Treiber**

### **11.1 WDM – Win32-Driver-Model**

Heutige Gerätetreiber müssen dem von Microsoft definierten Win32-Treibermodell entsprechen. Dieser Standard ist seit Windows 98 verbreitet und ermöglicht eine Treiberentwicklung für alle Windows Systeme ab 98. Die frühere Version 95 nutzte VxD-Treiber (virtual-device-drivers). Windows NT4 hatte einen anderen Treibertyp, den sogenannten Kernel-Mode-Treiber, eingesetzt. Wollte man ein Gerät mit zwei Betriebssystemen ansteuern, so mussten zwei verschiedene Treiber programmiert werden. Unter der Voraussetzung, dass nur WDM-Bibliotheken zur Programmierung verwendet werden, kann der WDM-Treiber in allen neueren Betriebssystemen von Microsoft eingesetzt werden.

WDM-Treiber sind im Schichtmodell aufgebaut und ermöglichen so einen modularen Aufbau mehrerer Treiberebenen. Das eigentliche Anwendungsprogramm arbeitet im User-Mode und kann den Treiber mit einer beliebigen Programmiersprache ansprechen. Der Treiber ermöglicht, im Kernel-Mode mit anderen Treibern oder direkt mit I/O-Bereichen zu kommunizieren. Dadurch kann das Betriebssystem die Systemressourcen zuteilen und überwachen. Anwendungsfreundliche Eigenschaften wie Plug-and-Play und das Power-Management werden im Treiberkonzept unterstützt.

Um Treiber für Windows programmieren zu können, benötigt man eine spezielle Entwicklungsumgebung. Das DDK (Driver-Development-Kit) kann man von Microsoft für das jeweilige Betriebssystem gegen eine Gebühr von ca. 30 € beziehen. Bis vor kurzer Zeit war es möglich, das DDK kostenlos im Internet zu erhalten. Wenn man Mitglied im Microsoft-Development-Network (MSDN) ist, bekommt man das DDK neben vielen anderen Produkten zugeschickt. Die bevorzugte Programmiersprache ist Visual C++. Das DDK-Paket kann in eine normale Entwicklungsumgebung von Visual C++ eingebunden werden. Neu erstellte Treiber können mit zwei Varianten debuggt werden. Microsoft bietet dazu im SDK (Software-Development-Kit) das Programm WinDbg an. Der Hersteller sieht für eine komplette Entwicklungsumgebung einen Aufbau von zwei Computern vor. Ein Rechner enthält neben dem Betriebssystem nur den Treiber und eine Testapplikation zum Ansprechen

des Treibers. Der zweite Rechner ist mit einem seriellen Kabel verbunden. Auf ihm läuft die Debug-Umgebung. Dieser klassische, aber auch aufwändige Aufbau sollte nur für die kommerzielle Entwicklung genutzt werden.

Eine Alternative dazu stellt das Programm SoftIce von NuMega dar. Der Debugger wird vor dem Betriebssystem in den Speicher geladen und an der entsprechenden Stelle aufgerufen. Es ist zu beachten, dass der Debugger immer aktiv ist. Mit SoftIce ist ein zweiter Rechner nicht notwendig.

## **11.2 USB-Treiber**

Um USB-Geräte steuern zu können, ist ein Treiber unverzichtbar. Ehe Signale von der Anwendungssoftware bis zum Gerät gelangen, durchlaufen sie mehrere Treiberschichten. Dazu gehören Host-Treiber, Hub-Treiber und spezifische Gerätetreiber. Untere Treiberschichten spielen für den Entwickler von USB-Geräten nur eine geringe Rolle. Anwendungsspezifische Treiber selbst zu schreiben, erfordert umfangreiche Programmierkenntnisse und Erfahrung. Eine große Hilfe dabei sind die vielen Beispiele des DDK. Oftmals werden aus Abwandlungen der Beispiele eigene Gerätetreiber entwickelt. Eine weitere Strategie ist die Nutzung vorhandener Treiber. Viele Hersteller von USB-Entwicklungskits bieten Treiber für ihre Produkte an. Diese sind dann meist speziell auf den USB-Chip zugeschnitten. Für Anwendungen mit USB-Klassen wird der vorgefertigte Klassentreiber verwendet. Gegebenenfalls muss für spezielle Geräte ein Minitreiber oder ein Systemkomponententreiber auf den Klassentreiber aufgesetzt werden. Eine weitere Alternative stellt der generische Treiber dar. Diese Treibervariante ist in der Lage, jedes USB-Gerät anzusteuern. Der Treiber kann auf das Gerät adaptiv eingestellt werden. Dies geschieht in der Inf-Datei. Alle WDM-Treiber haben als Dateiendung .sys. Zu jedem Treiber gehört zusätzlich noch eine Inf-Datei mit der Endung .inf. Wenn ein Gerät an den Bus angeschlossen wird, sucht das Betriebssystem im Rahmen der Enumeration im Inf-Verzeichnis nach einer passenden Inf-Datei. Diese enthält neben Vendor-ID, Produkt-ID und Device-ID Informationen über die Konfiguration des Gerätes und des Treibers. Anhand dieser Daten kann der passende Treiber geladen werden.

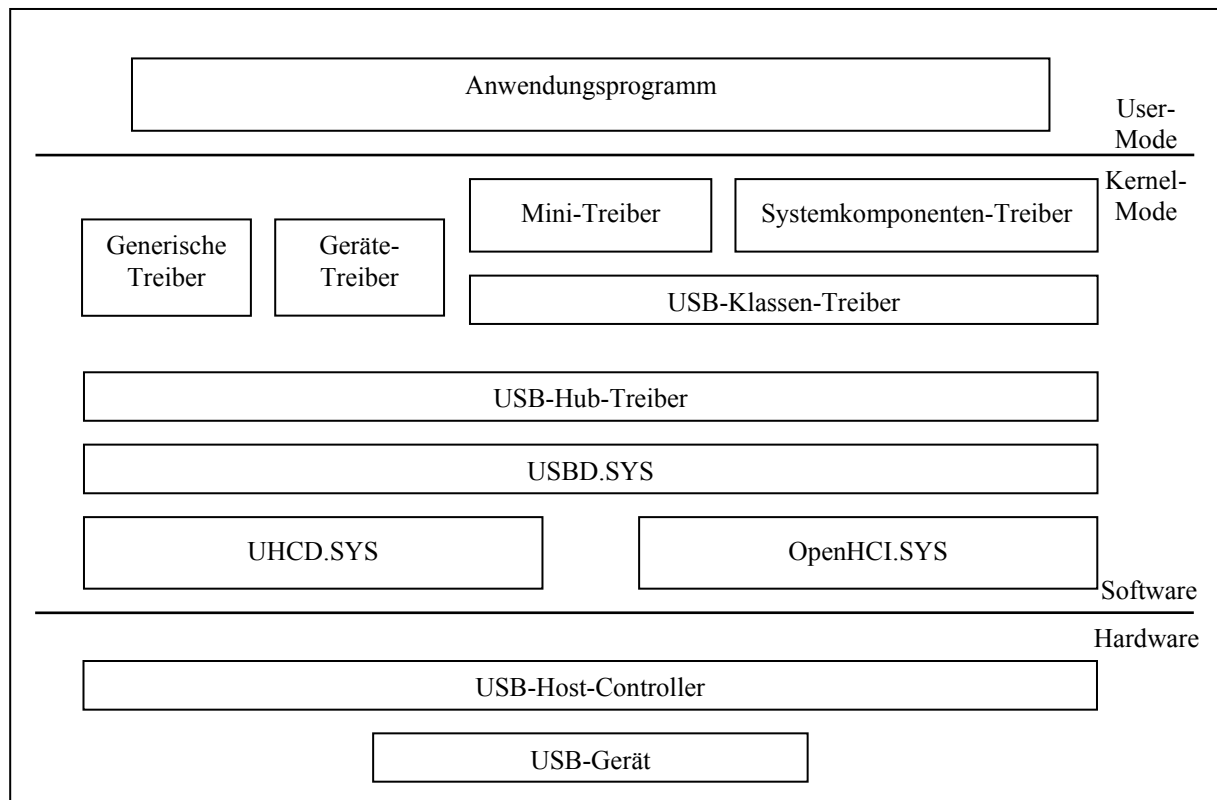


Abb. 29 – USB-Treiber-Struktur im Windows Betriebssystem

### 11.3 Treiber unter Linux

Linux unterstützt USB ab der Kernelversion 2.4 im gleichen Maße wie Windows. Viele Treiber werden in den aktuellen Distributionen mitgeliefert und sind sofort einsetzbar. Alle Standardgeräte, die Windows erkennt, können auch unter Linux angesprochen werden. Spezielle Gerätetreiber findet man im Internet unter [www.linux-usb.org](http://www.linux-usb.org/) /22/. Diese sind durch das GNU-Konzept fast immer als Quellcode angeboten und müssen nach dem Download kompiliert und eingebunden werden. Dafür wird immer eine Make-Datei angeboten, die alle erforderlichen Schritte ausführt. Bei den meisten Treibern wird für die Kompilierung das Kernel-Source-Paket benötigt. Es empfiehlt sich daher, dieses Paket zu installieren. Das fertig kompilierte Modul muss dann in das Betriebssystem integriert werden. Der zugehörige Befehl ist „insmod\_ < Modulname.o >“. Viele Treiberpakete bieten dazu auch eine Make-Install-Datei an, die diesen Vorgang automatisch erledigt. Mit „lsmod“ kann man überprüfen, ob der Treiber geladen wurde und aktiv ist.

Linux nutzt zur Kommunikation mit dem Treiber das URB-Modell. Das ist die äquivalente Struktur zur Windows-API-Programmierung, die auf Linux angepasst ist. Die Befehle sind hierbei anders, um den Linux-Code anzusprechen. Eine Hilfe findet man unter [http://usb.cs.tum.edu/usbdoc.](http://usb.cs.tum.edu/usbdoc/) /23/

## 12 Schaltkreisauswahl für USB-Geräte

Für die Entwicklung von USB-Geräten stehen eine große Anzahl von verschiedenen Chips zur Auswahl. Diese Gesamtmenge muss der Entwickler mit Hilfe gerätespezifischer Kriterien eingrenzen. Generelle Faktoren sind hierbei Kosten und Komplexität des Chips. Der Entwickler muss die Kosten für die Erstellung des Geräts sowie den Geräteendpreis berücksichtigen. Die Komplexität des Schaltkreises ist je nach Anwendungsgebiet zu wählen. Sind im Projekt schon andere Mikrocontroller verwendet worden, so kann man USB-Chips auswählen, die lediglich über eine SIE mit zusätzlicher Hardwareanbindung verfügen. Anderenfalls kann ein Mikrocontroller mit USB-Schnittstelle für das Projekt eingesetzt werden. Dieser verfügt dann neben einer SIE auch über eine CPU und kann programmiert werden. Im Internet finden sich auf den Seiten der Chiphersteller viele USB-Schaltkreise. Eine kleine Auswahl an Geräte-Controllern ist in der folgenden Tabelle aufgelistet.

Hersteller	Bezeichnung	Eigenschaften
AMD	AM186CC	186-basierend mit 2 UARTS, 1 SPI, 12 DMA Kanälen, USB-Peripheral and 4HDLC-Kanälen
	AM186CU	gleiches Modell wie CC aber ohne HDLC Kanäle
Atmel	AT76C711	USB-Controller mit Anpassung auf schnelle serielle Schnittstellen (Netzwerk/IRDA) / AVR
	AT8xC5131	USB mit 8051 Architektur und allen notwendigen Schnittstellen / Flash
	AT43USB320A	USB Hub mit AVR Mikrocontroller
	AT43USB326	USB Keyboard-Controller mit AVR Architektur
	AT43USB353M	AVR mit USB Schnittstelle, ADC, PWM nur maskenprogrammierbar
	AT43USB355	AVR mit USB Schnittstelle, ADC, PWM Pingleich mit ...353M, Programmierung über externes EEPROM/Flash
Cypress	AN2131	USB mit 8051 Architektur und allen notwendigen Schnittstellen, über USB programmierbar
	AN2720SC-01	USB zu USB Chip, dient zur USB Kommunikation zwischen zwei PC's
	CY7C64603	überarbeitete Version von AN 2131 mit zusätzlich DMA, GPIF und Memory extention
	CS5954AM	maskenprogrammierbarer Controller mit allen Funktionen des CY7C64603 aber 16bit Risc Architektur
	SL11R-(DIE)	USB – IDE(Atapi) Controller, maskenprogrammierbar
	SL811S	USB-Interface zum Anschluss an weitere Mikrocontroller
	CY7C64013	8-bit Risc-Controller mit PROM und DAC
	CYC7C63001A	USB-Interface mit EPROM ,OTP weiterer Mikrocontroller notwendig
Infinion	SAB-C541U-1EN	OTP mit 8K ROM, SPI
Microchip	PIC16C745	PIC Risc-Architektur mit USB und EPROM, 5xADC 8-bit, 22 I/O
	PIC16C765	wie 745 nur 8xADC 8-bit, 33 I/O
Motorola	68HC705JB3	Motorola Architektur mit Lowspeed USB, OTP
	68HC908JB8	Motorola Architektur mit Flash, nicht mit USB programmierbar
Philips	ISP1181-A	USB Interface für IDE
SMSC	USB97C102	USB Controller mit ISA/DMA Funktion mit 8051Kern
ST Microelectronics	ST72611F1M1	Low-Speed-Controller mit maskenprogrammierbarem ROM
	ST72621J4T1	Low-Speed-Controller mit maskenprogrammierbarem ROM
Texas Instruments	TUSB 3210	Full-Speed-Controller mit 8052-Architektur, GPIO, I <sup>2</sup> C, mit 8 Kbyte RAM oder ROM

Tabelle 28 – Auswahl an USB-Chips



Die in Tabelle 28 gezeigte Auswahl an Chips ist nur ein geringer Bruchteil aller USB-Schaltkreise. Durch fortschreitende Entwicklung wird die Gesamtzahl noch steigen. In der Liste sind keine USB 2.0-fähigen Schaltkreise enthalten.

Während einige Chiphersteller ihre bekannten Produkte nachträglich mit USB ausstatteten, haben andere neue Produkte rund um die USB-Schnittstelle entwickelt. Neben Schaltkreisfamilien, die nur für USB entwickelt wurden, hat Cypress zusammen mit Intel Derivate des Mikrocontrollers 8051 mit USB ausgestattet. Die Entwicklungswerkzeuge hat sich Cypress von Intel lizenzieren lassen. Mit der Übernahme von Anchor Chips, welche die Baureihe EZ-USB herstellten, ist Cypress führend auf dem Markt der USB-Controller für Peripheriegeräte geworden.

Einige Chips findet man sehr häufig in Entwicklungsanwendungen. Diese sollen in den folgenden Teilabschnitten kurz vorgestellt werden.

## **12.1 Cypress CY7C63000, CY7C63001A**

Einer der ersten Chips war der CY7C63000 von Cypress. Dieser Low-Speed-Schaltkreis ist besonders für Eingabegeräte wie Maus und Joystick geeignet. Neben einem Entwicklungskit existieren viele Anwendungen für kleine Messanforderungen. Bekannt wurde der Schaltkreis in seinem Gebrauch als USB-Thermometer. Der Schaltkreis hat inzwischen einen Nachfolger, den CY7C63001A. Dieser ist in einem 20- oder 24-poligen Gehäuse untergebracht und verfügt über bis zu zwei Acht-Bit-Ports. Der Programmspeicher ist als OTP ausgelegt, was sehr hohe Entwicklungskosten verursacht. Der Chip lässt sich mit einem Resonator betreiben, so dass nur wenige preisgünstige Bauelemente für den Betrieb notwendig sind. Seine Risc-CPU kann mit CYASM, einem erhältlichen Assembler von Cypress, programmiert werden. /5/, /6/, /7/

## **12.2 FDTI FT232BM, FT245BM**

Um vorhandene Mikrocontroller mit einer USB-Schnittstelle ausrüsten zu können, dient dieser Chip, der eine Interface-Lösung von USB-Full-Speed auf RS 232 anbietet. Der FT245BM beinhaltet dafür auch den Pegelumsetzer auf RS 232, während der FT232BM die Signale im TTL-Pegel liefert. Somit kann mit dem Baustein die komplette RS 232-Schnittstelle mit allen Daten- und Statusleitungen nachgebildet werden. Die dadurch entstandene Schnittstelle kann dann von Standardtreibern angesprochen werden. Sie unterstützt alle Geschwindigkeitsstufen der RS 232. Der Hersteller FDTI wird in Deutschland von der Firma Unitronic vertreten. /8/, /9/

## **12.3 Philips Semiconductors PDIUSB11, PDIUSB12**

Weiterhin gibt es zur Applikation an Mikrocontrollern von Philips den PDIUSB11 mit einer I<sup>2</sup>C-Schnittstelle und den PDIUSB12 mit paralleler Anbindung. Damit kann man nahezu jeden vorhandenen Mikrocontroller mit USB 1.1 ausstatten. Die volle USB-Bandbreite ist beim PDIUSB11 aber nicht nutzbar, da die Ausgabeschnittstellen die maximale Bandbreite bestimmen. Der PDIUSB11-Chip enthält 4 Endpoints mit je 8 Byte tiefen FIFO's. Mögliche Transferarten sind Control-, Bulk- und Interrupt-Transfer. Der andere Chip besitzt drei Endpoints mit unterschiedlichen FIFO-Tiefen. Maximum sind hier 128 Byte für Isochronous-Transfer. Alle Transferarten werden unterstützt. /10/, /11/

## 12.4 Microchip PIC16C765, PIC 16C745

Die Firma Microchip hat mit den PIC-Controllern eine sichere Position auf dem Mikrocontroller-Markt. Die zwei Chips aus der 16C-Serie sind die ersten PICs, welche die Risc-CPU mit USB 1.1 verbinden. Programmierbar ist der Schaltkreis durch das kostenlos angebotene MPLAB von Microchip. Für erste Versuche stellt Microchip viele Anwendungsbeispiele zur Verfügung. In diesen wird das Gerät oft als HID-Klassengerät mit dem spezifischen Klassentreiber angesprochen. Die Programme dieser Baureihe lassen sich nur mit UV-Licht löschen. Zu Testzwecken stellt Microchip seine Vendor-ID und Produkt-ID zur Verfügung. /12/, /13/

## 12.5 Dallas Semiconductors DS2490

Dallas Semiconductors hat mit dem 1-Wire-Bus eine preisgünstige Alternative für kleine Sensoren- und Aktorennetzwerke geschaffen. Der Hersteller bietet neben einer großen Anzahl verschiedener Sensoren und Interfacebausteinen auch diesen USB 1.1 zu 1-Wire-Bus-Adapter an. Diese sehr einfach aufzubauende Applikation nutzt mit dem zugehörigen Treiber den USB, um mit der eigenen Software zu kommunizieren. Die eigentliche USB-Kommunikation wird hierbei nicht betrachtet. /14/, /15/, /16/

## 12.6 Intel und Cypress 8x930A, 8x931A

Intel hat in Zusammenarbeit mit Cypress die beiden Mikrocontroller mit USB 1.1 ausgestattet. Der 8x931A ist vom klassischen 8051-Kern abgeleitet worden und besitzt drei Endpoints mit 8 oder 16 Byte tiefen FIFO's. Mit 265 Byte RAM, 3 Timern und 10 Interruptquellen ist er in der Lage, die USB-Kommunikation selbst zu führen und Daten zu verarbeiten. Der 8x930A ist von einem späteren Derivat des 8051 abgeleitet worden. Als Vorlage diente hier der 80251, der über wesentlich mehr Rechenleistung verfügt als der 8051. Der 8x930A kann vier oder sechs Endpoints verwalten. Mit 16 MByte möglichem externen Speicherbus und 1024 Byte internem RAM ist der Anschluss von zusätzlicher Peripherie unproblematisch. Beide Chips verfügen über eine serielle Schnittstelle und können sowohl in Assembler als auch in C programmiert werden. Als Programmspeicher dient entweder ein externer Speicher oder der eingebaute ROM. /17/

## 12.7 Cypress EZ-USB (AN2131SC/QC)

EZ-USB ist ein von Anchor Chips entwickeltes 8051-Derivat, das jetzt von Cypress angeboten wird. Zu dieser Baureihe gehören vier Chips. Der hierbei Interessanteste ist der AN2131 in der kleinen Ausführung mit SC als Präfix und in der großen Variante mit QC am Ende. Vorlage für diesen Chip ist der 80C320, eine Weiterentwicklung des 8051. Der Schaltkreis unterstützt USB 1.1 und kann bis zu 32 Endpoints verwalten. Die maximale FIFO-Tiefe liegt bei 2048 Byte. Die 8 KByte Daten- und Programmspeicher sind kombiniert und können extern auf 64 KByte erhöht werden. Neben der Verarbeitung von USB-Daten ist der AN2131 in der Lage, die Daten schneller als USB zu Peripheriebausteinen zu transportieren. Er verfügt über zwei serielle Ports und eine I<sup>2</sup>C-Busverbindung. Das Programm kann aus einem I<sup>2</sup>C-EEPROM geladen werden oder direkt über USB übertragen werden. In der

Diplomarbeit wurde dieser Chip ausgewählt. Detaillierte Informationen sind im folgenden Kapitel zu finden. /5/, /18/, /19/, /20/, /21/

## **12.8 FX-Serie; CY7C64603, CY7C64613**

Cypress bietet nicht nur die Chips von Anchor an, sondern hat auch eine eigene Serie auf der Basis des 8051. Die beiden ausgewählten USB 1.1-Schaltkreise verfügen über die gleichen Merkmale wie die EZ-USB-Serie. Zusätzlich besitzen einige Schaltkreise ein sogenanntes GPIF (General Programmable Interface) und DMA. /21/

## **12.9 FX2-Serie**

Alle bisher vorgestellten Schaltkreise unterstützen Full- oder Low-Speed. Die FX2-Serie ist einer der ersten Chips, die im High-Speed-Modus arbeiten können. Angelehnt an die FX-Serie hat Cypress mit dem CY7C68013 einen 8051-kompatiblen Mikrocontroller mit USB 2.0 auf den Markt gebracht. Um Daten mit dieser Geschwindigkeit verarbeiten und transportieren zu können, sind externe FIFO's vorgesehen. Periphere Bausteine greifen dann auf die FIFOs zu. Somit wird der Chip nicht zur Bandbreitenbegrenzung für die USB 2.0-Übertragung. /21/, /22/

# **13 Realisierung einer Schrittmotoransteuerung mit USB**

Am Beispiel der Schrittmotorsteuerung ist die Eignung und Funktionsweise des USB-Mikrocontrollers veranschaulicht worden. Um eine weitere Verwendung des USB-Controllers zu ermöglichen, ist ein modularer Aufbau vorgesehen. Ein wichtiges Kriterium war die Auswahl des Mikrocontrollers unter folgenden Vorgaben:

- USB 1.1-Spezifikationen sollen unterstützt werden
- Vielseitige Verwendung für verschiedene Projekte der Mess- und Sensortechnik
- Möglichst Anwendung bekannter Mikrocontrollerfamilien
- Einfache und kostengünstige Beschaffung
- Bereitstellung von Entwicklungswerkzeugen, Treibern und Programmierung des Mikrocontrollers

Für die Demonstration der Schrittmotorsteuerung wurde der AN2131 ausgewählt. Dieser Schaltkreis soll im folgenden Kapitel detailliert beschrieben werden:

## **13.1 Cypress AN2131SC/QC**

Die Schaltkreise der EZ-USB-Familie, zu der der AN2131 gehört, sind oft von Entwicklern eingesetzt worden. Neben den Distributoren von Cypress kann der Chip auch in vielen Elektronikatalogen wie Reichelt oder RS bestellt werden. Cypress bietet für US\$ 495 auch ein Entwicklungskit an. Benötigte Treiber und Software sind neben dem Handbuch auf der Webseite von Cypress /21/ erhältlich. Der AN2131 besitzt folgende Eigenschaften:

### **13.1.1 USB 1.1-fähige SIE und 8051 kompatibler CPU-Kern**

Der AN2131 vereint zwei Hauptschaltkreise auf einem Chip. Die SIE verarbeitet alle USB-Signale und besitzt die Möglichkeit, Daten über ein USB-Interface mit der CPU des Mikrocontrollers auszutauschen. Im Standardmodus übernimmt somit die SIE den kompletten USB-Verkehr. Die CPU hat dadurch viel Rechenzeit zur Verfügung, um Daten zu verarbeiten und mit anderen externen Schaltkreisen zu kommunizieren. Mit Hilfe der CPU kann man die Descriptoren und damit die Identität des Gerätes verändern. In diesem Falle verarbeitet die CPU selbst die USB-Kommunikation. Die Rechenzeit ist damit begrenzt. Der Vorteil dieses Konzeptes besteht darin, Programmcode mit USB übertragen zu können. Das Gerät meldet sich als Cypress-Controller am USB an und kann mit dem Treiber und mitgelieferter Software programmiert werden. Das neue Programm enthält dann einen kompletten Satz neuer Descriptoren. Das Gerät simuliert anschließend eine Trennung vom USB und meldet sich mit den neu erhaltenen Descriptoren an. Es wird jetzt als völlig anderes Gerät erkannt und kann mit einem anwendungsspezifischen Treiber angesprochen werden. Diesen Prozess nennt Cypress Renumeration. Der Hersteller hat sich diesen Begriff schützen lassen.

### **13.1.2 Speicher des EZ-USB**

Beide Varianten des EZ-USB haben die gleiche interne von-Neumann-Speicherstruktur. Im Bereich von 0x0 bis 0x1B3F können Code und Daten gespeichert werden. Von 0x1B40 bis 0x1F3F wird der Speicher für die FIFO-Puffer der Endpoints genutzt. Intern gibt es dann noch einen Speicherbereich von 0x7B40 bis 0x7FFF, in dem alle Register für die CPU und die USB-Kommunikation enthalten sind. Extern können an die 80-polige Variante AN22131QC Speicher für Programm und Daten angeschlossen werden. Das Adressvolumen beträgt bis zu 64 KByte für Daten und 64 KByte für Programme. Die Zusammenlegung von Programm- und Datenspeicher ist auch möglich. Der Chip stellt dazu verknüpfte Signale der Schreib- und Leseleitungen zur Verfügung.

### **13.1.3 I/O-Ports**

Der AN2131SC besitzt 18 I/O-Ports, die als Eingang, Ausgang oder als Spezialfunktion beschalten werden können. Um die Ports in die gewünschte Funktion zu setzen, gibt es für jeden Port vier Steuerregister. Die SC-Variante stellt den Port B und C, sowie A(4) und A(5) zur Verfügung. Die größere Variante QC hat zusätzlich den kompletten Port A.

Da der Chip mit einer Betriebsspannung von 3,3 V arbeitet, werden auch die logischen Pegel der Ports mit diesen Spannungen dargestellt. Diese Pegel werden sind LVTTTL-Spezifiziert (Low-Voltage-Transistor-Transistor-Logic). Zu beachten ist dies bei der Signalzuführung an andere Digitalschaltkreise. Diese müssen Eingänge mit TTL-Logikpegeln ausweisen.

### **13.1.4 I<sup>2</sup>C-Bus**

Viele Schaltkreise haben zur Kommunikation mit anderen Chips eine I<sup>2</sup>C-Bus-Schnittstelle. Neben A/D-, D/A-Wandlern, Portexpandern und Speichern gibt es zahlreiche Bauelemente, die mit I<sup>2</sup>C-Schnittstellen ausgerüstet sind. Der AN2131 hat eine I<sup>2</sup>C-Schnittstelle, bei der er als Master die Kommunikation kontrolliert. Zur Steuerung des Busses besitzt er zwei Register und einen Interrupt. Wichtigstes Merkmal des Busses ist die Möglichkeit, Descriptoren und

Programm in einem EEPROM abzulegen. Beim Start des Controllers werden die Daten automatisch übertragen. Dazu muss das EEPROM auf die passende Adresse eingestellt sein. Bei kleinen EEPROM's bis 2 KByte werden die Daten mit einer 8-Bit Adresse abgefragt. Dabei ist der Speicher in 256 Byte große Seiten aufgeteilt. Mit den vorhandenen Adresspins kann eine Seite ausgewählt werden. Der automatische Zugriff ist nur für die 256 Byte der ersten Seite möglich. Die größeren ab 4 KByte bis 64 KByte nutzen eine 16-Bit Adresse. Die Geräteadresse im I<sup>2</sup>C-Bus ist für die beiden Speichertypen unterschiedlich. Um eine Unterscheidung zwischen den 8 Bit und den 16 Bit adressierbaren Speichern zu erreichen, werden die Speicher auf verschiedene I<sup>2</sup>C-Busadressen geschaltet. Das geschieht über die Adresspins der EEPROMs, die als Anschlüsse herausgeführt sind. Bei kleinen EEPROMs werden alle drei Adresspins mit Masse verbunden. Für große Speicher wird das Pin A(0) an die Betriebsspannung angelegt. Alle anderen Adresspins sind mit Masse kontaktiert. Man kann den EEPROM auch auf eine andere Adresse setzen. Descriptoren und Programm werden dann aber nicht mehr automatisch in die CPU geladen. Wenn der Speicher auf die richtige Adresse eingestellt ist, entscheidet das erste Byte im Speicherinhalt, wie die Daten interpretiert werden. Ist das erste Byte 0xB0, so werden die folgenden sechs Byte ausgelesen. In ihnen ist dann die Vendor-ID, Produkt-ID und die Device-ID enthalten. Wenn das erste Byte 0xB2 ist, werden zusätzlich zu den ID-Bytes Programmdatei in den Speicher des Mikrocontrollers übertragen. Diese Speicherblöcke können bis zu 1023 Byte lang sein. Damit ist es möglich, das Programm dauerhaft in den Controller laden zu können, ohne es vorher mittels USB zu übertragen. Der Controller kann dadurch auch ohne die USB-Schnittstelle arbeiten.

### **13.1.5 Interrupts**

Der AN2131 besitzt 13 Interruptquellen. Neben den Standardinterrupts für Timer, serielle Schnittstellen, und externe Interrupts bietet der Chip Interrupts für I<sup>2</sup>C, Wakeup und USB an. Um alle USB-Aktivitäten zu überwachen, reicht ein Interrupt nicht aus. Deshalb hat der Chip ein sogenanntes Autovectoring. Der erste Befehl in der Interruptroutine für USB sollte deshalb ein Sprungbefehl vom Typ „ljmp“ sein. Die Sprungadresse ist bei diesem Befehl drei Byte lang. Der Autovektor ersetzt automatisch das zweite Byte der Sprungadresse. In einer Sprungtabelle, die bei 0XXX00 beginnt, werden dann an den entsprechenden Einsprungadressen des Autovektors weitere Sprünge zu den verschiedenen Bearbeitungsroutinen programmiert. Jede Routine steht dann für ein USB-Ereignis. Damit ist es möglich, Endpoints zu bedienen, USB-Reset zu detektieren und SOF- oder Setup-Token zu erkennen. Es ist nicht zwingend, dass die automatischen Routinen genutzt werden. Ebenso kann aus einem entsprechenden Register die Sprungadresse gelesen und verarbeitet werden.

### **13.1.6 USB-Kommunikation**

Im Startzustand übernimmt die SIE die komplette USB-Kommunikation. Sie kann über die Interrupts, Register und Endpoint-Puffer Daten mit der CPU austauschen. Wenn eigene Descriptoren verwendet werden sollen, übernimmt die CPU des Controllers die Koordination des USB-Protokolls. Im Grundzustand ist das Alternate-Setting 0 aktiviert. In dieser Einstellung verfügt der Controller nur über den Endpoint EP0. Daher ist nur der Control-Transfer zulässig. Mit den Alternate-Settings 1 und 2 sind zusätzlich weitere Endpoints unterschiedlicher Puffergröße vorhanden, die auch die anderen drei Transferarten anbieten. Durch den Treiber können alle diese Endpoints angesprochen werden. Um keine Bandbreitenbegrenzung für die Kommunikation mit zusätzlichen Peripheriebausteinen

darzustellen, sind verschiedene Mechanismen in den Controller implementiert. Zwei Endpoints gleicher Richtung können für ein Endpoint-Pairing verwendet werden. Dabei werden zwei gleichgroße FIFOs genutzt. Der Host nutzt eine Pipe zum Endpoint um Daten zu empfangen oder zu senden. Während der Host in einen Puffer schreibt oder liest, kann der Controller die Daten der anderen FIFO verarbeiten. Über Register werden die FIFOs anschließend umgeschaltet. Somit wird verhindert, dass sich Host und CPU gegenseitig blockieren. Um die Daten schnell transportieren zu können, eignet sich der Fast-Transfer. Die Daten werden dann vom externen Daten- und Adressbus eingelesen oder dahin geschrieben. Dies ist nur beim AN2131QC möglich, da nur diese Variante über den externen Daten- und Adressbus verfügt. Dazu nutzt man einen der beiden 16-Bit Datenpointer. Dieser wird dann selbstständig bei jedem Datenzugriff inkrementiert. Es ist garantiert, dass Daten schneller transportiert werden als bei USB 1.1. Damit ist die Bandbreite nicht durch den Chip eingeschränkt.

### **13.1.7 Software**

Cypress bietet im Internet ein Softwarepaket an, in dem eine komplette Entwicklungsumgebung enthalten ist. Dazu gehört der Treiber in verschiedenen Varianten. Dieser wird sowohl als fertige .sys-Datei und als Quellcode in C bereitgestellt. Zur Kontrolle des Chips gibt es von Cypress zwei Programme. Das ist einerseits der SIE-Master zur Kommunikation mit der SIE und zum anderen das EZ-USB Control-Panel für die Verbindung mit der CPU. Mit diesen Programmen sind alle Funktionen des Controllers steuerbar. Für die Entwicklung sind die beiden Programme sehr wichtig, da alle Funktionen der eigenen Applikation überprüft werden können. Um Programme für den Mikrocontroller schreiben zu können, ist eine Demoversion der Firma Keil enthalten. Das Programm  $\mu$ Vision 2 dient als komplette Entwicklungsumgebung. Man kann zwischen den Programmiersprachen Assembler und C wählen. Die Programme können auf dem Simulator oder dem Debugger getestet werden. Einzige Beschränkung ist die maximale Programmgröße von 4 KByte. Zu jedem der einzelnen Programme werden Beispiele mitgeliefert, die den Einstieg in die Programmierung des AN2131 erleichtern.

### **13.1.8 Abschätzung der Geschwindigkeit**

Die größte USB-Übertragungsgeschwindigkeit ist mit Bulk-Transfer auf einem ansonsten freien Bus möglich. Cypress hat diese Geschwindigkeit getestet und einen In-Transfer mit 17 Bulk-Transfers pro Frame erreicht.

Daraus ergibt sich eine Übertragungsrate von 1.08 MByte/s. Um den Out-Transfer zu messen, wurde ein Messprogramm in Visual C++ geschrieben. Das Projekt hat den Namen „test\_3“ und ist auf der CD gespeichert. In der folgenden Abbildung ist das Messergebnis dargestellt.

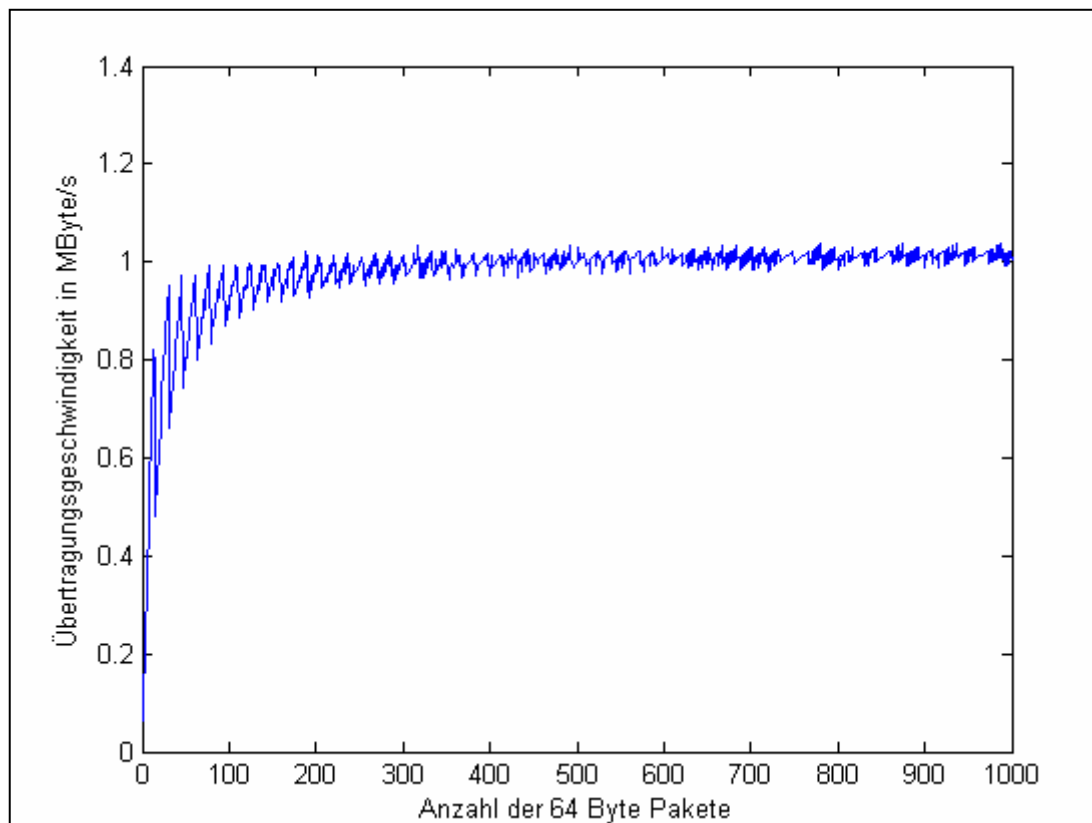


Abb. 30 – Messung der Übertragungsgeschwindigkeit

Dabei wurde eine Datenrate von 1 MByte/s gemessen. Die entspricht ca. 16 Paketen pro Frame. Dies gilt nur, wenn genügend Daten bereitgestellt werden. Die zackenförmigen Einbrüche in der Kurve sind mit der Frameauslastung zu erklären. Für die Messung können nur ganze Frames gezählt werden. Im Beispiel ist sichtbar, dass die Rate bei 14 Paketen ca. 0,8 MByte/s beträgt. Dazu wird ein Frame benötigt. Bei 15 Paketen fällt die Rate enorm auf ca. 0,5 MByte/s ab. Man benötigt ein zweites Frame, das nur ein Paket enthält, aber trotzdem 1ms dauert. Ein Frame kann theoretisch maximal 19 Bulk-Pakete enthalten. Diese Auslastung kann aber praktisch nicht erreicht werden. Die anderen vorhandenen, aber nicht genutzten Isochronous- und Interrupt-Endpoints schränken die Übertragungsrate ein. Verzögerungen bei der Verarbeitung der Frameliste des Hosts sorgen für eine zusätzliche Verringerung der Bandbreite. Um Messungenauigkeiten zu vermeiden, wurde der Versuch mehrmals durchgeführt. Je geringer die zu erwartenden Messzeiten waren, desto häufiger ist die Messung wiederholt worden. Die Messungen wurden mit den Entwicklungsboards AN2131SC und AN2131QC durchgeführt. Im Ausschnitt der Messdaten ist der rasante Abfall der Messwerte gut zu erkennen. Für weitere Anwendungen kann mit einer Übertragungsbandbreite von 1 MByte/s gerechnet werden. Mit diesen Messungen ist die Funktion und die Leistungsfähigkeit der Schnittstelle nachgewiesen worden.

Anzahl bereitgestellter 64 Byte-Pakete	Anzahl der Schleifendurchläufe	Gemessene Zeit in ms	Datenrate in MByte/s
13	769	781	0,819216
14	714	812	0,787862
15	666	1328	0,481446
16	625	1250	0,512
17	588	1188	0,538505

Tabelle 29 - Daten zur Messung der Übertragungsgeschwindigkeit

## 13.2 Aufbau der Testplatinen des Mikrocontrollers

Für erste Versuche ist der AN2131SC genutzt worden. Diese kleinere Variante hat keinen Daten- und Adressbus nach außen geführt. Der Port A ist nicht komplett vorhanden. Intern ist der Chip jedoch identisch mit der QC-Ausführung. Für weitere Anwendungen stehen im Labor zwei erste Testplatinen, die einen I<sup>2</sup>C-EEPROM vorweisen, zur Verfügung. Die Schaltung ist aus der Zusammenfassung vieler Aufbaubeispiele entstanden. Alle Anschlüsse sind auf Steckerleisten herausgeführt und können abgegriffen werden. Für kleine Steuerungsaufgaben und erste Testprogramme hat sich dieser Aufbau bewährt. Im Internet bieten verschiedene Hersteller die gleiche Schaltung als fertig bestückte Platine an, um Geräte mit einer USB-Schnittstelle ausrüsten zu können. /27/

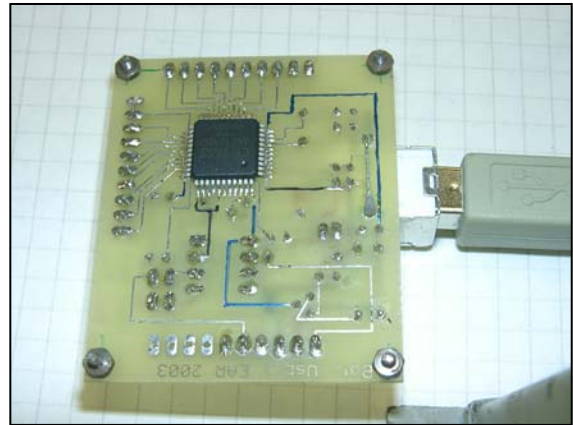
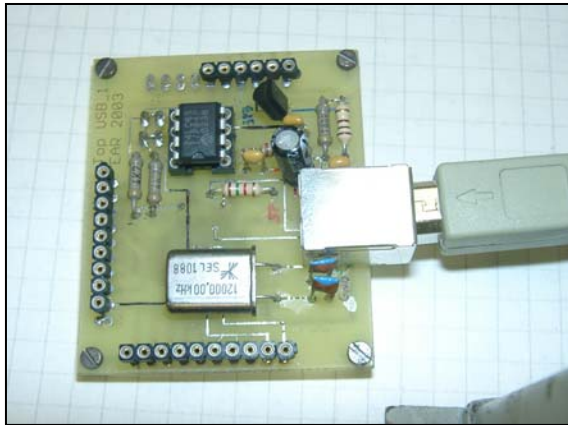


Abb. 31, 32 – AN2131SC-Entwicklungsboard Ansicht von oben (links) und unten (rechts)

Für die Schrittmotorsteuerung wurde eine Platine mit der QC-Variante aufgebaut. Diese hat einen externen Speicher und Pegelwandler für die seriellen Schnittstellen. Zusätzlich kann das Entwicklungsboard auch mit einer externen Stromversorgung betrieben werden.

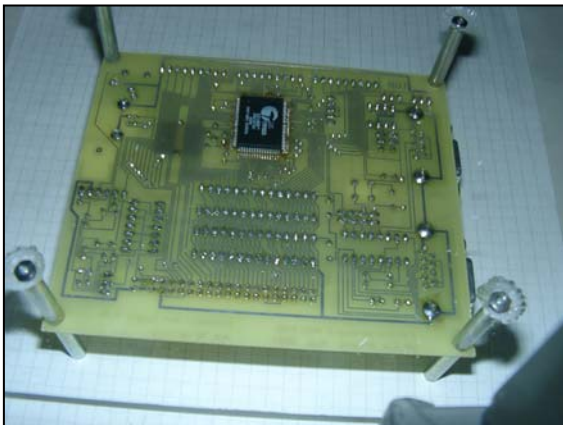


Abb. 33, 34 – AN2131QC-Entwicklungsboard in Ansicht von oben (links) und unten (rechts)

Alle Schaltpläne sind mit Eagle 4.01 erstellt worden und können im Anhang und auf der CD eingesehen werden. Um die Platinen vielseitig einsetzen zu können, ist der Aufbau so modular wie möglich gestaltet worden. Das Entwicklungsboard mit dem AN2131QC wird für die Schrittmotorsteuerung eingesetzt. Das folgende Blockschaltbild zeigt den modularen Aufbau der Schrittmotorsteuerung.



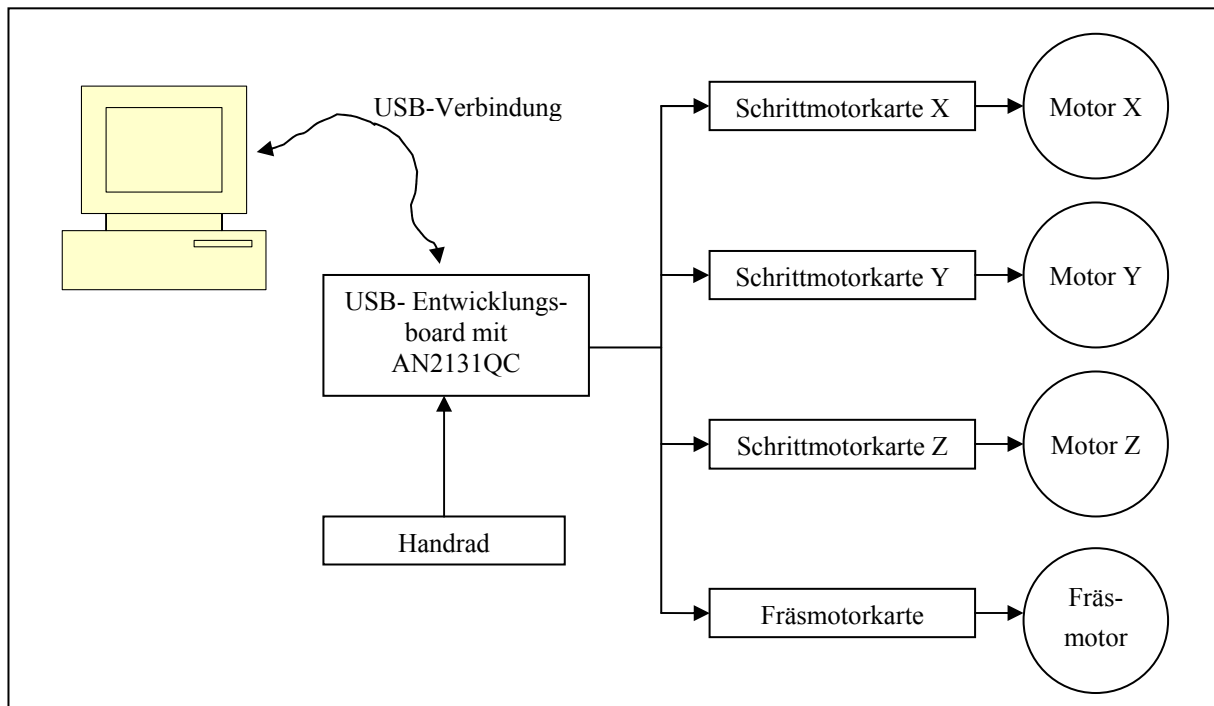


Abb. 35 – Blockschaltbild der Schrittmotorsteuerung

### 13.3 Schrittmotortreiberschaltkreis IMT901

Für die minimale Ansteuerung von Schrittmotoren sind lediglich vier Transistoren notwendig. Der hier verwendete Konstantstromtreiber vereint mehrere Vorteile verschiedenster Ansteuermöglichkeiten in einem Halbleiter. Sowohl Logik als auch Leistungsansteuerung befinden sich in einem Gehäuse. Der IMT901 ist in der Lage, die Motorschritte in acht Mikroschritte aufzuteilen. Damit erhöht sich die Winkelauflösung der Welle, ohne ein zusätzliches Getriebe einzusetzen. Für die Ansteuerung sind nur sehr wenige Bauteile zur äußeren Beschaltung notwendig. Durch die Verwendung eines Adresskonzepts, konnte das modulare Aufbauprinzip umgesetzt werden. Für jeden Motor ist eine Motorsteuerkarte notwendig. Diese Karte bekommt dann über einen Wahlschalter auf der Platine eine Adresse zugeteilt. Insgesamt können mit drei Adressleitungen sieben Motoren angesteuert werden. Die achte Adresse ist als Umschaltposition für den Mikrocontroller vorgesehen. Unter der zugeteilten Adresse lassen sich die Parameter Drehrichtung, Schrittweite, Motorfreigabe und Motorleistung einstellen. Über eine separate Adressleitung werden die Takte aller Motoren ausgewählt. Das Datenbyte, das unter dieser Adresse an alle Motoren gesendet wird, kodiert in jedem der einzelnen acht Bit einen Takt für einen Motor. Der Schaltplan ist im Anhang und auf der CD zu finden. Der IMT901 wird über den Hersteller Nanotec vertrieben. /24/, /25/, /12/

### 13.4 Ansteuerung des Fräsmotors

Die Anwendung der Schrittmotorsteuerung ist ursprünglich für einen Verschiebetisch am Lasermessplatz für Ultraschallsensoren konzipiert worden. Da der Messplatz aber von den Mitarbeitern der Professur für Mess- und Sensortechnik benötigt wird, musste ein anderer Demonstrator eingesetzt werden. Dieser ist eine Dreiachsmaschine mit Fräsmotor. Sie könnte beispielsweise zum Bohren von Leiterplatten verwendet werden. Für den Fräsmotor, in diesem Falle ein Dremel-Fräser, benötigt man eine Ansteuerung, die den Motor in der Drehzahl regeln kann. Dazu wurde eine Phasenanschnittsteuerung aufgebaut, deren

wichtigstes Bauelement ein Mikrocontroller vom Typ PIC 16F84A04 ist. Dieser erkennt über eine galvanische Trennung mittels Optokoppler den Nulldurchgang der Netzspannung. Je nach angelegtem Digitalwert des Mikrocontrollers AN2131 verzögert der PIC-Controller die Zündung des Triacs. Das Programm des Mikrocontrollers ist mit der kostenlosen Software MPLAB von Microchip erstellt worden und ist auf der CD gespeichert. Durch die Verschiebung des Zündzeitpunktes des Triacs verändert sich die effektive Spannung des Motors und damit seine Drehzahl. Diese Schaltung wurde ebenfalls in das modulare Adresskonzept eingegliedert. Die Steuerplatine ist wie ein Schrittmotor zu betrachten. Sie erhält ebenfalls eine Adresse und wird wie die Schrittmotorkarten angesprochen. Die verwendete Datenstruktur wurde für die Ansteuerung verändert. Beim Aufbau der Platine ist auf Gefahren beim Arbeiten mit der Netzspannung zu achten. Entsprechende Sicherheitsmaßnahmen sind zu treffen. /13/, /25/, /26/

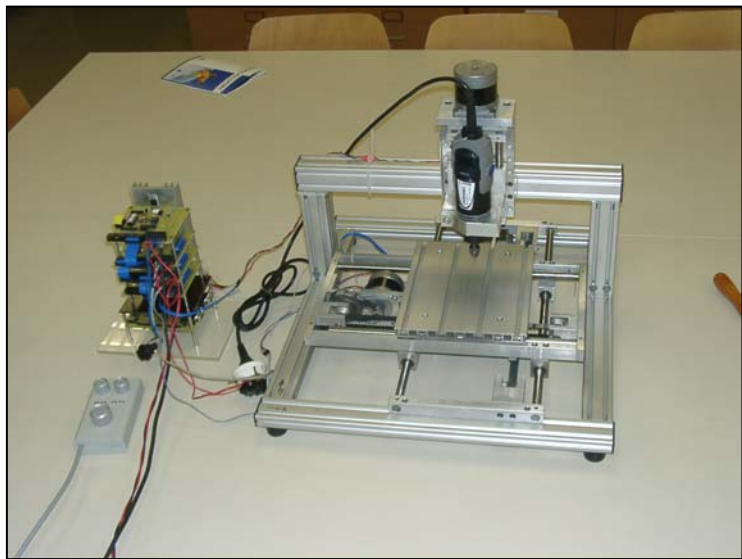


Abb. 36, 37 – Modularer Gesamtaufbau aller Schaltungen (links); Demonstrator (rechts)

## 13.5 Handrad

Um die Steuerung der Maschine zu vereinfachen, wird ein Handrad verwendet. Dieses enthält einen inkrementellen Geber und zwei Wahlschalter. Damit lassen sich die Motoren und deren Schrittweite einstellen. Der Inkrementalgeber zählt die Schritte in der gewünschten Drehrichtung. Zum Anschluss des Handrads an den AN2131 werden ein externer Interrupt sowie einige Portleitungen genutzt.

## 13.6 Software der Steuerung

Die PC-Software der Steuerung wurde mit Visual C++ programmiert. Dabei wird der mitgelieferte USB-Treiber angesprochen. Das entsprechende Projekt heißt „Motorsteuerung“ und ist mit allen Quelltexten auf der CD einzusehen. Die Steuerungsaufgaben wurden so unterteilt, dass die PC-Software neben der Initialisierung und der Kommunikation mit dem Mikrocontroller nur Anzeigefunktionen erfüllt. Das Programm wurde aus früheren Testbeispielen entwickelt. Es ermöglicht, Bohrdaten vom Leiterplattenentwurfsprogramm Eagle zu öffnen und mit der Maschine abzuarbeiten. Für exakte Bohrergebnisse muss die Maschine aber genau kalibriert werden.

Das Programm des Mikrocontrollers ist mit dem Entwicklungspaket von Keil in der Programmiersprache C erstellt worden. Als Berechnungstakt für die Bahnsteuerung der

Motoren wurde das SOF-Paket des USB verwendet. Es löst jede Millisekunde einen Interrupt aus. In der entsprechenden Routine wird dann die Bewegungsansteuerung des Motors vorgenommen. Die Schritte werden in 255 Subschritte aufgeteilt. Über USB werden als erstes die Motorinitialisierungsdaten übertragen. Steuerdaten der Motoren werden nur als Sollwerte in ganzen Schritten angegeben. Der Mikrocontroller rechnet diese in Subschritte um. In äquidistanten Abständen fragt die PC-Software die Istwerte der Motoren ab. Zur Berechnung der Bewegung wird mit einer Geschwindigkeitsrampe gearbeitet, welche die Beschleunigung und Verzögerung des Motors beschreibt. Die Rampe dient zur Taktgenerierung der Motorschritte. Um Beschleunigungen zu erzielen, werden die Zeiten zwischen den Schritttakten variiert. Für die Berechnung werden die Parameter maximale Geschwindigkeit und maximale Beschleunigung benötigt. Diese können im Applikationsprogramm des Computers eingestellt werden. Da die Ansteuerung der Maschine nur der Veranschaulichung dienen soll, ist die Software nicht mit allen notwendigen Funktionen ausgestattet worden. Für den späteren Einsatzfall kann dies nachgeholt werden.

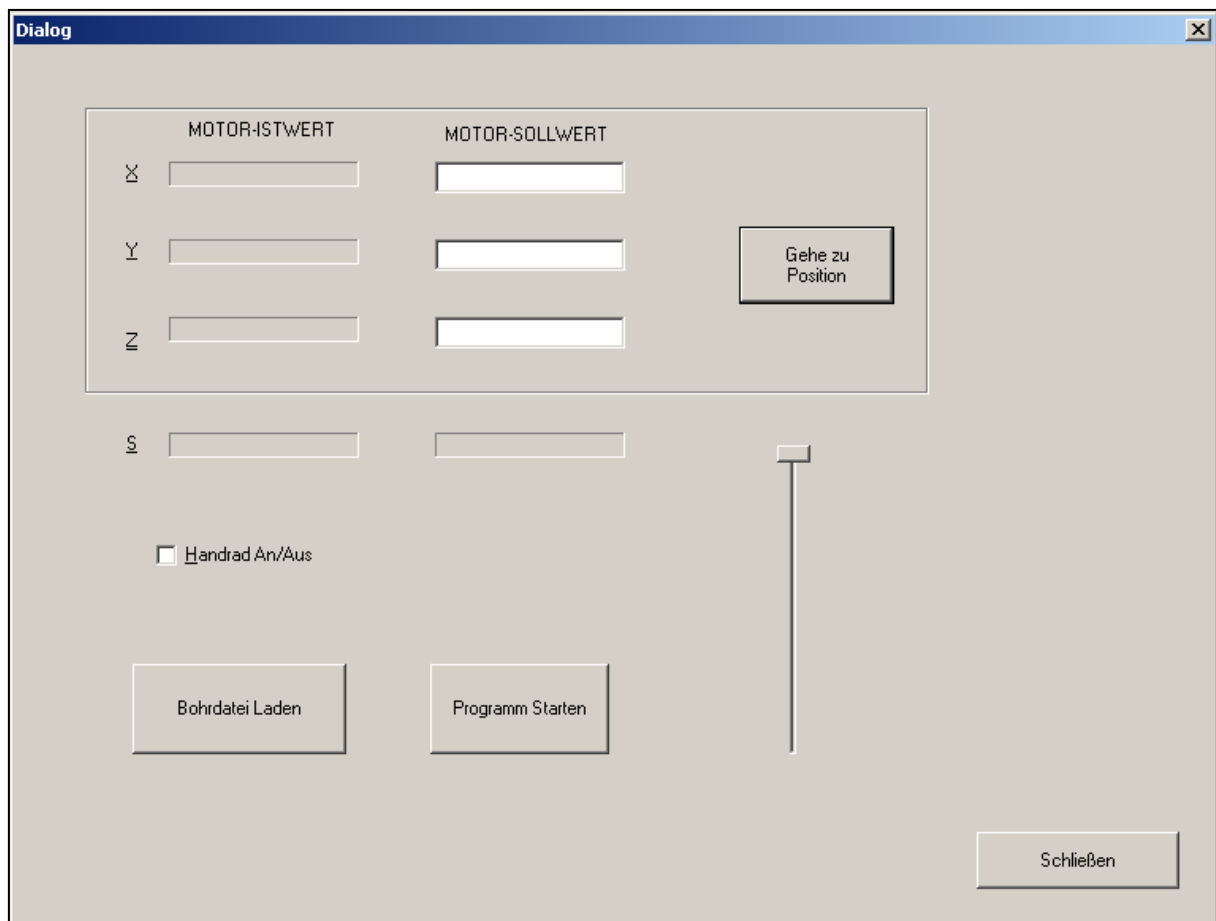


Abb. 38- Dialogfenster der PC-Software

## 13.7 Aufbau der Maschine

Für die Demonstration der Schrittmotorsteuerung ist eine Maschine geplant und konstruiert worden. Die Fertigung aller Einzelkomponenten übersteigt aber den Zeitrahmen der Diplomarbeit. Aus diesen Gründen wird ein anderer Demonstrator verwendet, der für die Präsentation der Steuerkarte geeignet ist. Die Abb. 39 zeigt die Konstruktionszeichnung des ursprünglichen geplanten Maschinenaufbaus. Abb. 37 zeigt den für die Präsentation verwendeten Demonstrator. Es wurde mit Hilfe des Demonstrators gezeigt, dass eine Schrittmotorsteuerung mit USB-Schnittstelle realisierbar ist.

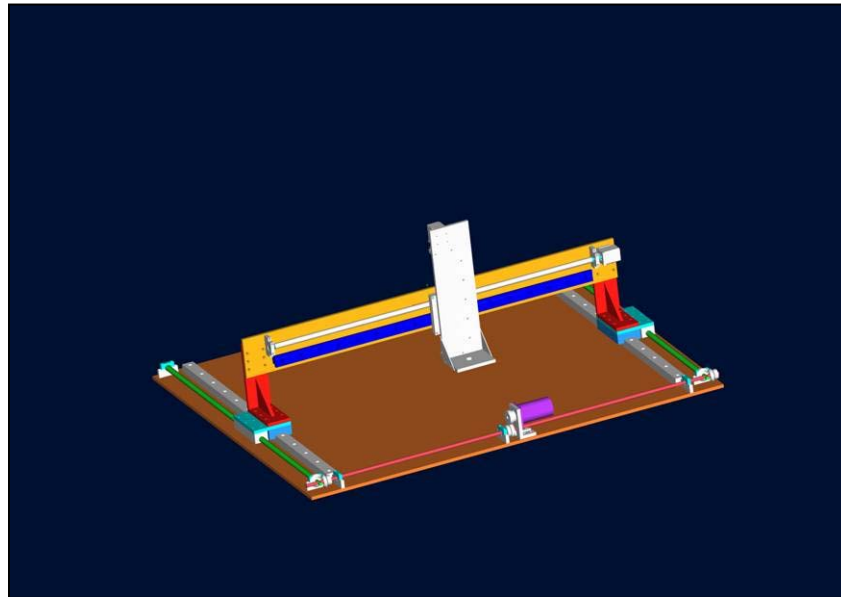


Abb. 39 – Abbildung der Konstruktionszeichnung

## 14 Zusätzlicher Handlungsbedarf

Die Verbesserung der USB-Schnittstelle ist Gegenstand laufender Forschungsarbeiten. Die neuste Veröffentlichung ist USB-On-The-Go. Mit diesem Konzept können Peripheriegeräte miteinander kommunizieren, ohne an einen Host angeschlossen zu sein.

Für verschiedene Anwendungen mit der USB-Schnittstelle sind im Rahmen der Diplomarbeit mehrere Entwicklungsboards entstanden. Diese können für die vielseitigen Aufgaben im Rahmen der Forschungstätigkeit eingesetzt werden. Zusammen mit der Software bieten sie Lösungen für unterschiedliche Anwendungen. Für die einzelnen Applikationen können anhand der Beispiele schnell spezifische Programme erarbeitet werden.

Um Ultraschall-Empfangswandler mit 64 Elementen abfragen zu können, ist der Aufbau einer 64-fachen Empfangsstufe mit USB geplant. Zu diesem Zweck soll ein Mikrocontroller der FX2-Serie zum Einsatz kommen. Er bietet mit der USB 2.0-Schnittstelle eine hohe Übertragungsrate. Für den Einsatz an der Professur für Mess- und Sensortechnik wird deshalb ein Entwicklungsboard mit dem Controller CY7C68013 aufgebaut. Die vorhandene Software kann weiterhin genutzt werden. Damit ist es möglich, Messdaten schnell von den Ultraschallwandlern in den Computer zu übertragen und auszuwerten. Ebenso sollte das Entwicklungsboard derart gestaltet sein, dass es als Ergänzung oder Ersatz für die vielen Anwendungen der anderen Entwicklungsplatinen eingesetzt werden kann.



Abb.39- Die Chips CY7C68013 (li.) und AN22131QC (re.) im Größenvergleich

## 15 Zusammenfassung

Steigende Datenraten wirken sich auf die Komplexität der Schnittstellen aus. Die vielseitigen Funktionen des USB stellen für den Entwickler eine große Herausforderung dar. USB bietet eine Vielzahl von Lösungsmöglichkeiten zum Anschluss von Peripheriegeräten an den Computer. Um sich schnell einen Überblick für die Arbeit mit USB verschaffen zu können, wurden in der Diplomarbeit wichtige Eigenschaften und Funktionsweisen der Schnittstelle in zusammengefasster Form dargestellt. Das soll eine Grundlage sein, eigene, spezifische USB-Lösungen zu erarbeiten und in der Professur für Mess- und Sensortechnik einzusetzen. Anwendungen mit älteren Schnittstellen können modernisiert und verbessert werden. Große Datenaufkommen sind damit leichter, schneller und effizienter zu verarbeiten und auszuwerten. Mit den verschiedenen Lösungsansätzen für Schaltungsaufbauten ist eine Grundlage für zukünftige Mess-, Steuer- und Regelungsgeräte geschaffen worden.

Ich möchte mich recht herzlich beim gesamten Team der Professur für Mess- und Sensortechnik für die erfolgreiche Zusammenarbeit bedanken. Mein besonderer Dank gilt meinem Diplombetreuer, Herrn Dipl. Ing. Haftmann, für die Unterstützung und Beratung in dieser Phase meines Studiums.

**Selbstständigkeitserklärung:**

Hiermit erkläre ich, Eik Arnold, die im Rahmen der Diplomarbeit gestellten Aufgaben selbstständig bearbeitet und gelöst zu haben.

Chemnitz, den 01.12.2003

.....

Eik Arnold

## Literaturverzeichnis

- /1/ [www.usb.org](http://www.usb.org); USB-Spezifikationen von USB 1.0, USB 1.1 und UBS 2.0;2000
- /2/ H. J. Kelm (Hrsg.); USB 2.0 – Das Praxisbuch; 2. Aufl. Franzis' Verlag; 2001; ISBN 3-7723-7966-4
- /3/ Jan Axelson; USB – Handbuch für Entwickler; MITP-Verlag; 2001; ISBN 3-8266-0698-1
- /4/ John Hyde; USB By Example; 2nd Edition; Wiley/Intel University Press; 2001; ISBN 0-9702-8465-9
- /5/ Burkhard Kainka; Messen Steuern und Regeln mit USB; Franzis' Verlag; 2000; ISBN 3-7723-5874-8
- /6/ Burkhard Kainka; USB-Interface; Elektor 9/2000 S.20 ff; ISSN 0932-5468
- /7/ Burkhard Kainka; USB-UART; Elektor 1/2002 S.26 ff; ISSN 0932-5468
- /8/ USB-RS 232-Interface; Elektor 4/2003 S.60 ff; ISSN 0932-5468
- /9/ [www.unitronic.de](http://www.unitronic.de); Datenblätter und Beschreibung USB-RS 232-Interface
- /10/ Philips Datasheets; Datenblatt PDIUSB11
- /11/ Philips Datasheets; Datenblatt PDIUSB12
- /12/ Eddie Brador; USB I/O-Modul; Elektor 11/2003 S.14 ff; ISSN 0932-5468
- /13/ [www.microchip.com](http://www.microchip.com); Hersteller der PIC-Controller, Softwarepaket MPLAB
- /14/ Luc Lemmens; USB-Interface für den 1-Wire-Bus; Elektor 6/2002 S.24 ff; ISSN 0932-5468
- /15/ Luc Lemmens; RS 232-Adapter für 1-Wire-Bus; Elektor 4/2002 S.68 ff; ISSN 0932-5468
- /16/ [www.ibutton.com/software/tmex](http://www.ibutton.com/software/tmex); Anbieter von Software für 1-Wire-Bus
- /17/ [www.intel.com](http://www.intel.com);
- /18/ Markus Müller, Christian Ehmer; USB für alle; Elektor 10 u.11/2002 S. 48 ff und.S.72 ff; ISSN 0932-5468
- /19/ [www.mmvisual.de](http://www.mmvisual.de); Webseite des oben genannten Autors Markus Müller
- /20/ Technical Reference Manual EZ-USB; Cypress Semiconductor
- /21/ [www.cypress.com](http://www.cypress.com); Development-Softwarepaket zu EZ-USB, FX und FX2
- /22/ [www.linux-usb.org](http://www.linux-usb.org); Sammlung vieler Treiber und Applikationsbeispiele für Linux
- /23/ <http://usb.cs.tum.edu/usbdoc>
- /24/ Microstep-Treiber IMT901; Elektor Doppelheft 7/8 /2000 S.24 ff; ISSN 0932-5468
- /25/ [www.nanotec.de](http://www.nanotec.de); Hersteller des IMT901 (Datenblatt)
- /26/ [www.sprut.de](http://www.sprut.de); Webseite über den PIC16F84 mit vielen Beispielen
- /27/ [www.braintechology.de](http://www.braintechology.de); Lieferant eines Schaltungsaufbaus mit AN2131SC