3. Augsburger Linux-Infotag 2004

Linux und USB

Stefan Warmuth

luga

http://www.luga.de

http://www.e-enthusiast.de

Einleitung

Es sind zwei Vorträge zum Thema Linux und USB vorgesehen

- USB für Einsteiger
  - ein wenig Hintergrundwissen zu USB und USB mit Linux
  - gängige USB- Geräte unter Linux
  - Werkzeuge und Wege bei Problemen
- Basteln mit USB und Linux für Fortgeschrittene
  - Funktionsprinzip von USB
  - Treiber
  - sdcc

USB für Einsteiger

USB allgemein

Geschichtliches

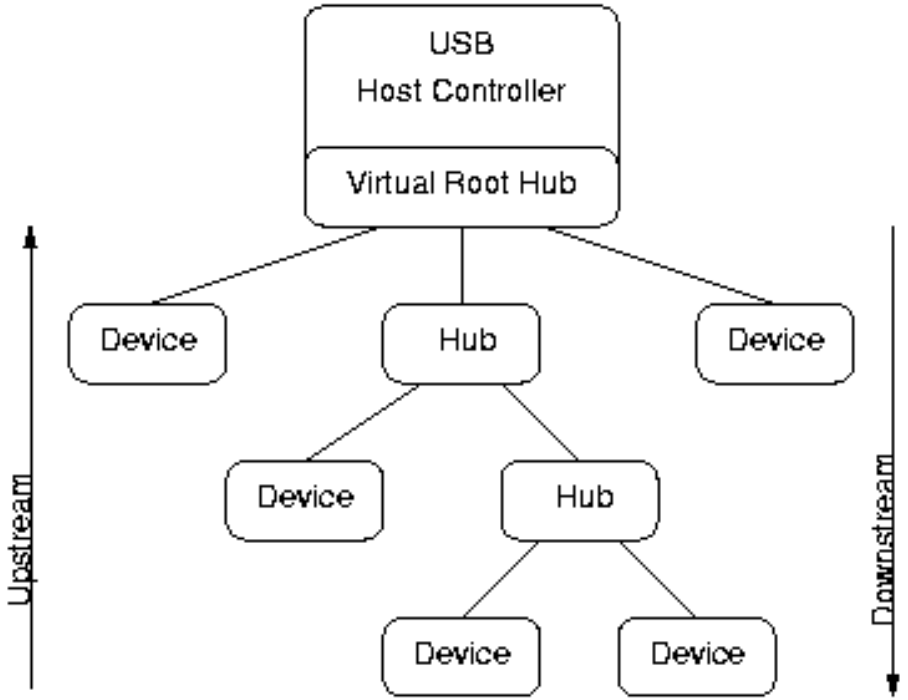
- 1994 spezifizierte eine Allianz von Compaq, Intel, Microsoft, NEC den Universal Serial Bus (USB)
- 1996 wurde die Version 1.0 verabschiedet
- 1998 folgte Version 1.1
- 2000 wurde die zur Zeit aktuelle Version 2.0 herausgegeben

Ziele

- einfach zu verwendendes System um Computer Peripherie anzuschließen
- hotplug fähig
- leicht erweiterbar
- soll das Ressourcen Problem lösen (Interrupts)
- geringe Kosten

Aufbau

- hierarchisches System
- Master/Slave Protokoll
- die Initiative geht immer vom Host aus
- Verteilung erfolgt über Hubs



[1]

Daten

- Geschwindigkeiten
  - Low-Speed 1,5 Mb/s (Mega Bit!)
  - Full-Speed 12 Mb/s
  - High-Speed 480 Mb/s
- max. 127 Geräte an einem Host

“Elektrisches“

- 5V Spannungsversorgung über USB Kabel
- selfpowered / buspowered Devices
- ein USB Port kann max. 500mA liefern
- ein USB Port muss min. 100mA liefern
- Kabellänge für Low-Speed max. 3m
  einfachere Kabel, immer fest “angewachsen“
- Kabellänge für High/Full-Speed max. 5m (Laufzeit)

Host Controllers

In der PC- Welt gibt es hauptsächlich drei Kategorien von Host Controllern.

- **USB 1.1**
  - UHCI
    - von Intel (PIIX4, VIA, ...)
    - “simplere“ Hardware
    - komplexerer Treiber
  - OHCI
    - von Compaq (NEC, iMacs, OPTi, SiS, ALi, NEC, ...)
- **USB 2.0**
  - EHCI

Linux und USB

Kernel Module Host

Die Kernel Module für das USB Host System

- **Kernel 2.4.x**
  - usbcore
  - usb-uhci bzw. uhci.o oder usb-ohci
  - ehci-hcd (zusätzlich)
- **Kernel 2.6.x**
  - usbcore
  - uhci-hcd, ohci-hcd
  - ehci-hcd (zusätzlich)

Kernel Module

Eine lsmod Ausgabe

```
logic:/# lsmod | grep usb
usb-storage            124176   0  (unused)
usb-ohci                19272   0  (unused)
usb-uhci                23600   0  (unused)
usbcore                 63820   1  [usb-storage \
              scanner audio hid ehci-hcd pegasus \
              usb-ohci usb-uhci]
```

usbdevfs

Für den “User Space“ Zugriff auf das USB System dient das “usbdevfs“ bzw. “usbfs“
Hierüber können Informationen über das USB System abgerufen werden aber auch auf USB Geräte zugegriffen werden.

```
logic:/# mount
/dev/hde9 on / type reiserfs (rw)
...
usbfs on /proc/bus/usb type usbfs (rw)
...
```

geg. in die “/etc/fstab“ eintragen

```
usbdevfs  /proc/bus/usb  usbdevfs  noauto  0 0
```

Devices

Um alle verfügbaren USB Geräte und deren Verwendung unter Linux vorzustellen würde zum einen den zeitlichen Rahmen der Veranstaltung sprengen zum anderen ist der Referent auch nicht in der Lage dies zu tun. Es werden deshalb nur einige gängige Geräte und Geräte Klassen vorgestellt. Nachdem das immer noch eine beachtliche Menge ist und dem Referenten kein “Schrank voll“ USB Geräten zur Verfügung steht kann deshalb teilweise nicht auf eigene Erfahrungen zurückgegriffen werden. Somit müssen einige Themen ungeprüft aus den Dokumentationen und Informationen aus dem Internet weitergegeben werden. Eine weiter Problematik ist die distributionsabhängige Handhabung von Konfigurationen und Hotplug Abläufen. Hier muss auf die Dokumentation der Distributionen verwiesen werden.

Devices

Der Zugriff auf USB Devices geschieht entweder über Kernel Module oder über das usbdevfs mittels der User Space Bibliothek libusb. Das hängt von den USB Geräten ab. Manche USB Geräte wie z. B. USB- Speicher- Sticks gehören USB-Klassen an und werden unabhängig vom Hersteller über ein einziges Kernel- Modul angesprochen (z. B. usb-storage).

hotplug

Idealerweise sollten sich, bei den heutigen modernen Distributionen, die benötigten Kernelmodule sowohl für die Host Controller als auch für die USB- Devices automatisch laden. Grundvoraussetzung sind:

- Kernel mit Hotplug- Fähigkeit
- installiertes “hotplug packet“

hotplug

Hotplug muss folgende Aufgaben erfüllen

- einen passenden Treiber finden und laden
- den Treiber mit den Gerät verbinden
- andere Subsysteme informieren bzw. starten

  z. B. Firmware in das USB Device laden oder USB Festplatten mounten

hotplug

Wird ein neues USB Device angesteckt wird vom Kernel ein Event generiert und ein User Space Helper aufgerufen.

```
logic:/# cat /proc/sys/kernel/hotplug
/sbin/hotplug
```

Die Konfigurationsdateien befinden sich hier:

```
logic:/# ls /etc/hotplug
blacklist  usb.distmap  usb.handmap
usb.rc     usb.usermap
usb        usb.agent
```

hotplug

Konfigurationsdateien

- **blacklist**

  hier kann hotplug für bestimmte Module verhindert werden
- **usb.rc**

  Script beim Init-V-Runlevel-Wechsel aufgerufen wird
- **usb.agent**

  Script das bei Veränderungen am USB aufgerufen wird

hotplug

Konfigurationsdateien

- **/lib/modeles/version/modules.usbmap bzw. usb.distmap (2.2.x)**

  Liste mit Device ↔ Modul Zuordnung, wird mit depmod -a erzeugt
- **usb.handmap**

  Liste mit Device ↔ Modul Zuordnung, kann auch erweitert werden
- **usb.usermap**

  Liste mit Device ↔ Modul Zuordnung, meist leer
- **usb/modulname**

  Scripte zu den Modulen

hotplug ...map

- usb module
- match_flags
- idVendor
- idProduct
- bcdDevice_lo
- bcdDevice_hi
- bDeviceClass
- bDeviceSubClass
- bDeviceProtocol
- bInterfaceClass
- bInterfaceSubClass
- bInterfaceProtocol
- driver_info

hotplug


```
/* bitmap values taken from include/linux/usb.h */
#define USB_DEVICE_ID_MATCH_VENDOR 0x0001
#define USB_DEVICE_ID_MATCH_PRODUCT 0x0002
#define USB_DEVICE_ID_MATCH_DEV_LO 0x0004
#define USB_DEVICE_ID_MATCH_DEV_HI 0x0008
#define USB_DEVICE_ID_MATCH_DEV_CLASS 0x0010
#define USB_DEVICE_ID_MATCH_DEV_SUBCLASS 0x0020
#define USB_DEVICE_ID_MATCH_DEV_PROTOCOL 0x0040
#define USB_DEVICE_ID_MATCH_INT_CLASS 0x0080
#define USB_DEVICE_ID_MATCH_INT_SUBCLASS 0x0100
#define USB_DEVICE_ID_MATCH_INT_PROTOCOL 0x0200
#define USB_DEVICE_ID_MATCH_DEVICE
  (USB_DEVICE_ID_MATCH_VENDOR | USB_DEVICE_ID_MATCH_PROD
#define USB_DEVICE_ID_MATCH_DEV_RANGE
  (USB_DEVICE_ID_MATCH_DEV_LO | USB_DEVICE_ID_MATCH_DEV_
#define USB_DEVICE_ID_MATCH_DEVICE_AND_VERSION
  (USB_DEVICE_ID_MATCH_DEVICE | USB_DEVICE_ID_MATCH_DEV_
...
```

hotplug

Was tun wenn ein USB Device von hotplug ignoriert wird?

- Vendor und Produkt ID herausfinden
- zugehöriges Modul herausfinden geg. besorgen, kompilieren
- vielleicht löst ein Lauf von depmod -a das Problem
- hotplug entsprechend händisch konfigurieren

hotplug

Beispiel: Der Prototyp eines TFT LCD USB Displays mit selbst geschriebenen Treiber wird von hotplug ignoriert.

```
nbook:~# lsusb
Bus 002 Device 001: ID 0000:0000
Bus 001 Device 001: ID 0000:0000
Bus 001 Device 007: ID 0547:1002 Anchor Chips, Inc.
```

hotplug


```
nbook:~# tail /etc/hotplug/usb.handmap
...
# ImageLink
dd_lou 0x0003  0x0547  0x1002  0x0000  0x0000  0x00
     0x00  0x00  0x00  0x00  0x00  0x00000000
```

hotplug

Als kleine Spielerei soll hotplug automatisch ein Bild auf das Display laden.

```
#!/bin/bash
/bin/cat /home/stefan/LoU/pix/test_tux.bin > /dev/usb/lo
```

USB Geräte

USB Geräte

Wie bereits angesprochen können nicht alle erdenklichen USB Geräte und deren Betrieb unter Linux vorgestellt werden.
Dennoch soll eine Auswahl der gängigsten USB Geräte die meisten Anwendungsfälle abdecken.
Auf folgende Geräte wird eingegangen

- Maus und Keyboard (Joysticks)
- Scanner
- Drucker
- USB- Sticks, USB- Festplatten und Brenner
- Digital Kameras
- Modem, ISDN, DSL

HID

Unter Human Interface Device (HID) versteht man Eingabe Geräte wie Mäuse, Tastaturen, Joysticks, ...
Für diese Geräte gibt es folgende Module

- hid
- usbmouse
- keyboard

Zudem sollte das Linux Input System geladen sein

HID und Input

Das Input System soll die Nutzung der verschiedenen Eingabegeräte ermöglichen, vereinheitlichen und vereinfachen. Folgende Module sind dafür verantwortlich

- evdev
- input
- joydev
- keybdev
- mousedev

```
logic:/# ls /dev/input/
event0  event2  js0  js2  mice    mouse1  mouse3
event1  event3  js1  js3  mouse0  mouse2

logic:/# lsmod | grep input
input     3488   0  [mousedev hid usbmouse evdev]
```

HID und Input

Um die Maus mit dem XServer zu Nutzen sollte folgendes in die XF86Config eingetragen werden. Natürlich bieten die meisten Distributionen Hilfsmittel.

```
logic:/# cat /etc/X11/XF86Config-4
...
Section "InputDevice"
        Identifier  "Configured Mouse"
        Driver      "mouse"
        Option      "CorePointer"
        Option      "Device"            "/dev/input/mice"
        Option      "Protocol"          "ImPS/2"
        Option      "ZAxisMapping"      "4 5"
EndSection
...
```

Scanner

Scanner werden mit SANE - Scanner Access Now Easy betrieben. Dabei werden die Eigenarten der unterschiedlichen Scanner in Bibliotheken den “backends“ ausgelagert. Der Zugriff auf die USB Scanner kann auf unterschiedliche Arten erfolgen:

- **Kernel Module**

  scanner, microtek, hpusbscsi

- **usbdevfs** mit libusb

Falls die Distribution keine grafische Konfiguration anbietet findet man die Konfigurationsdateien in /etc/sane.d/

Scanner

Beispiel Konfiguration Epson 1240 Scanner.
Erstmal backend “freigeben“:

```
stefan@stefan:/etc/sane.d$ cat dll.conf
# /etc/sane.d/dll.conf -  Configuration file for the \
    SANE dynamic backend loader
...
# enable the next line if you want to allow access \
    through the network:
net
#canon_pp
epson
fujitsu
...
```

Scanner

Beispiel Konfiguration Epson 1240 Scanner.
Konfiguration des backends:

```
stefan@stefan:/etc/sane.d$ cat epson.conf
# epson.conf
#
...
usb /dev/usb/scanner0
```

Scanner

Beispiel Konfiguration Epson 1240 Scanner.
Netzwerkfreigabe:

```
stefan:/etc/sane.d# cat saned.conf
# saned.conf
...
# NOTE: /etc/inetd.conf (or /etc/xinetd.conf) and
# /etc/services must also be properly configured to star
# the saned daemon as documented in saned(1), services(4
# and inetd.conf(4) (or xinetd.conf(5)).
logic
isa
nbook
```

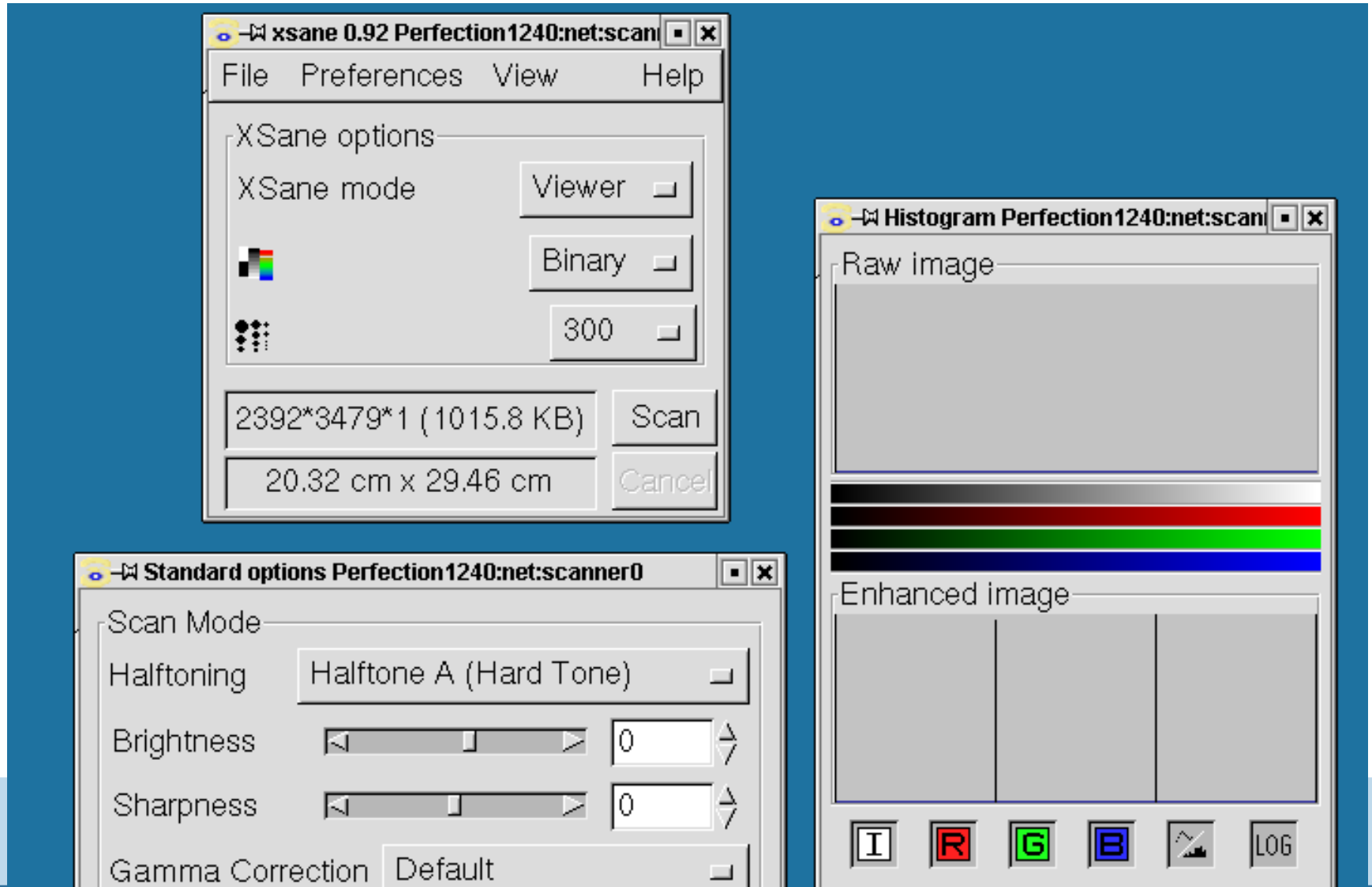
Scanner

Nützliche Tools

```
stefan:~# sane-find-scanner

  # No SCSI scanners found.
...
found USB scanner (vendor=0x04b8, product=0x010b) at \
                                  /dev/usb/scanner0
  # Your USB scanner was detected.
...

stefan:~# scanimage -L
device ‘epson:/dev/usb/scanner0’ is a Epson \
                      Perfection1240 flatbed scanner
```

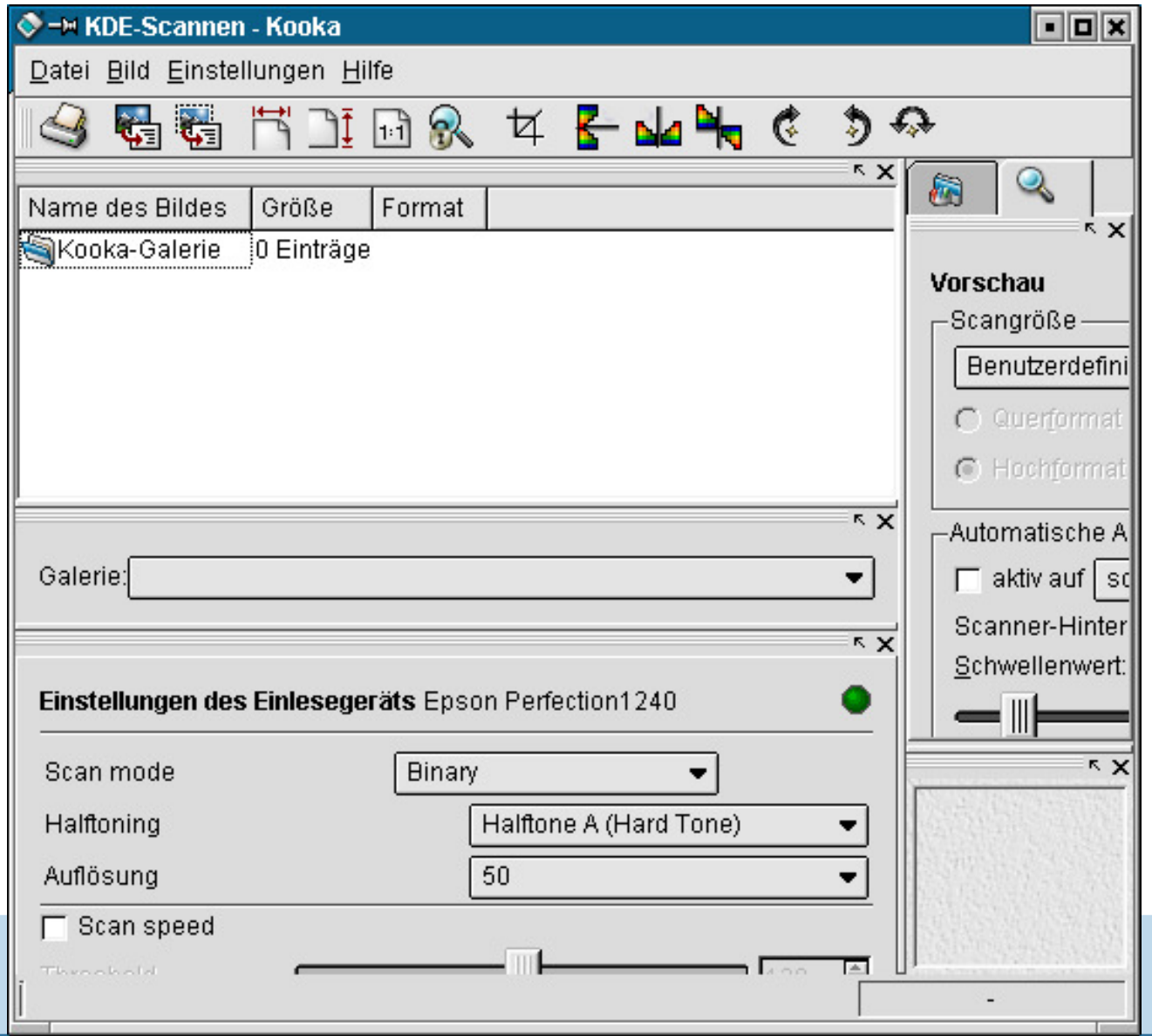
Scanner

Grafisches Frontend “xsane“



Scanner

Grafisches Frontend “kooka“



Drucker

Für USB Drucker gibt es auch einen USB Klassen Treiber “printer“.
Die USB Drucker sind damit über “/dev/usb/lpX“ ansprechbar.

USB- Sticks, HD, CD/DVD/Brenner

Für USB- Speicher- Sticks, USB- Festplatten, USB- DC/DVD- Laufwerke/Brenner gibt es auch einen USB Klassen Treiber “usb-storage“ das Treiber Modul für USB Massenspeicher. Diese funktionieren dann wie SCSI- Geräte. Dies ist naheliegend da das USB- Mass- Storage- Devices- Protokoll dem SCSI- Protokoll angelehnt ist.
Die Speicher Geräte könne dann über “/dev/sdXY“ z. B. “/dev/sda1“ die CD/DVD Laufwerke über “/dev/scdY“gemountet werden. Das “X“ ist für die Reihenfolge der angeschlossenen Geräte das “Y“ für die Partition. Da die USB Geräte normal nur über eine Partition verfügen ist “sdY1“ meist der richtige Eintrag. Schwieriger ist das mit der Reihenfolge.

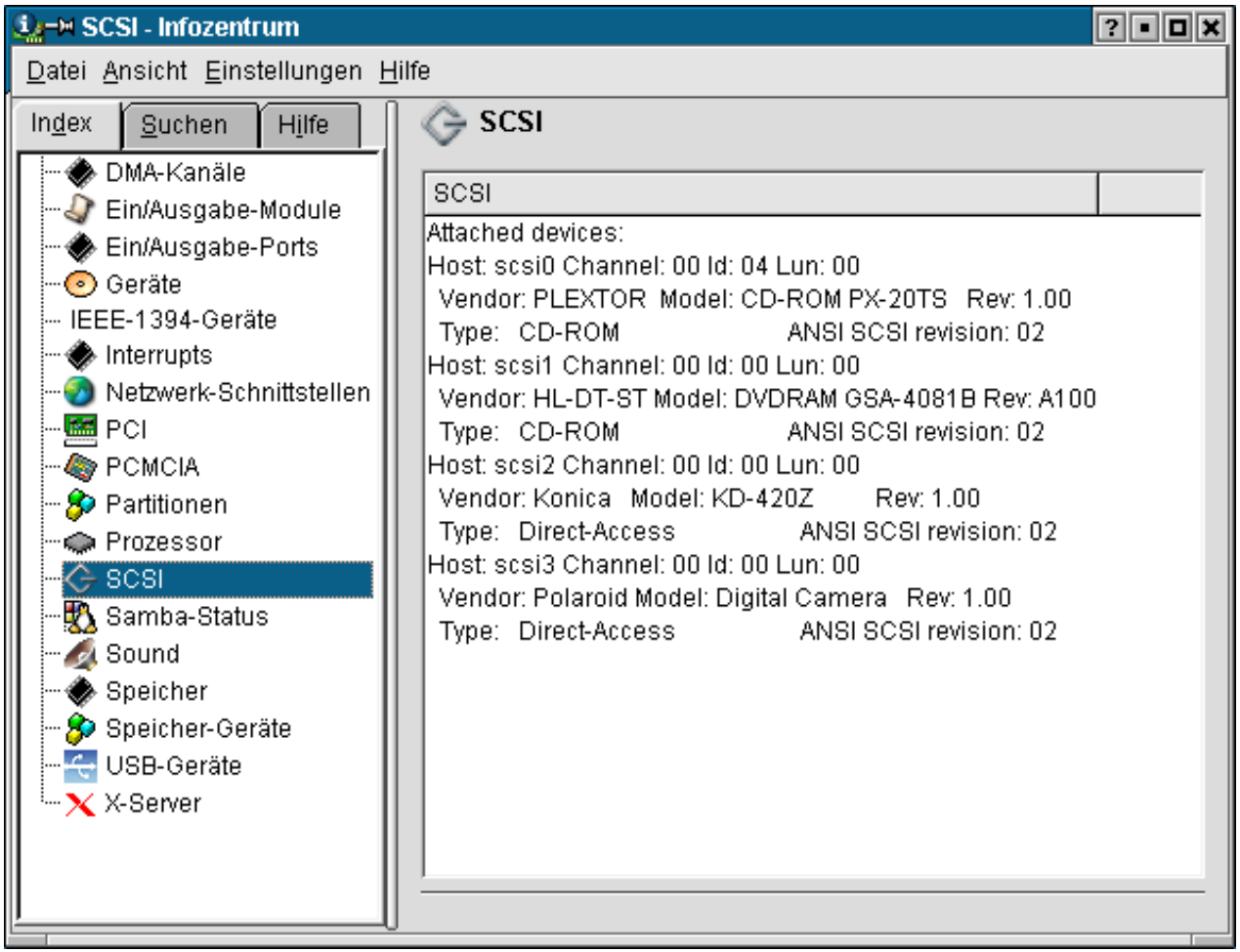
USB- Sticks, USB- HD, Brenner

Um herauszufinden ob “sda“, “sdb“ ... bzw. “scd0“, “scd1“ ... muss man hinter die Kulissen des SCSI Systems schauen.
Dies geht über:

- KDE Infozentrum
- /proc/scsi/scsi

USB- Sticks, USB- HD, Brennern

KDE Infozentrum



USB- Sticks, USB- HD, Brenner


```
logic:/usr/src/linux-2.4.25-rc1# cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 04 Lun: 00
  Vendor: PLEXTOR  Model: CD-ROM PX-20TS   Rev: 1.00
  Type:   CD-ROM                 ANSI SCSI revision: 02
Host: scsi1 Channel: 00 Id: 00 Lun: 00
  Vendor: HL-DT-ST Model: DVDRAM GSA-4081B Rev: A100
  Type:   CD-ROM                 ANSI SCSI revision: 02
Host: scsi2 Channel: 00 Id: 00 Lun: 00
  Vendor: Konica   Model: KD-420Z          Rev: 1.00
  Type:   Direct-Access          ANSI SCSI revision: 02
Host: scsi3 Channel: 00 Id: 00 Lun: 00
  Vendor: Polaroid Model: Digital Camera   Rev: 1.00
  Type:   Direct-Access          ANSI SCSI revision: 02
```

USB- Sticks, USB- HD, Brenner

Entsprechend der angezeigten Reihenfolge sind die Geräte “sda“, “sdb“ ... bzw. “scd0“, “scd1“ ... . Die Unterscheidung ob es sich um ein Speicher Gerät oder CD/DVD handelt sieht man am Type.
Über z. B. “mount /dev/sda1 /mnt/usb-storage“ lassen sich die Geräte mounten. Verwendet man die Geräte häufiger sollten Einträge in die “/etc/fstab“ gemacht werden und vielleicht Icons auf dem Desktop erstellt werden (bei KDE rechte Maustaste).

```
stefan@logic:~$ cat /etc/fstab
...
/dev/hde9 /                 reiserfs defaults    0  0
...
/dev/sda1 /mnt/usb-storage auto noauto,user 0  0
...
```

USB- Sticks, USB- HD, Brenner

Automatische Desktop Icons:
Hierfür gibt es keinen einheitlichen Weg.
Bei SuSE scheint das der “suseplugger“ zu erledigen bei Knoppix “hotplug-knoppix“ und “scanpartitions“.
Hier muss man sich auf seine Distribution verlassen oder selber experimentieren.

Digital Kameras

Digital Kameras werden auf unterschiedliche Arten angesprochen. Viel verhalten sich wie die USB Massenspeicher und werden wie die bereits angesprochenen USB- Speicher- Sticks behandelt.
Daneben verwenden viele Kameras auch ein Protokoll namens PTP (Picture Transfer Protocol). Leider gibt es noch eine Reihe Hersteller bzw. Kamera spezifischer Protokolle. Hierfür und für PTP Geräte kommt gphoto zum Einsatz. Der Zugriff erfolgt über “usbdevfs“ mit der libusb. Für manche Kameras gibt es auch Kernel Module. Diese sollten nicht geladen sein da diese gphoto blockieren.

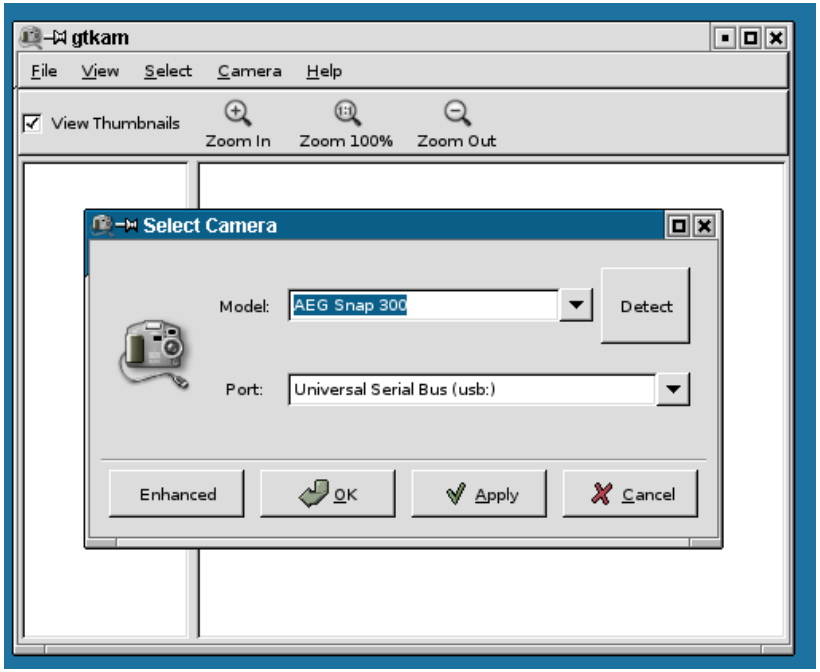
Digital Kameras

Ob eine Kamera unterstützt wird kann über die Homepage in Erfahrung gebracht werden oder mit:

```
logic:/# gphoto2 --list-cameras
Number of supported cameras: 434
Supported cameras:
        "AEG Snap 300"
        "Agfa ePhoto 1280"
        ...
```

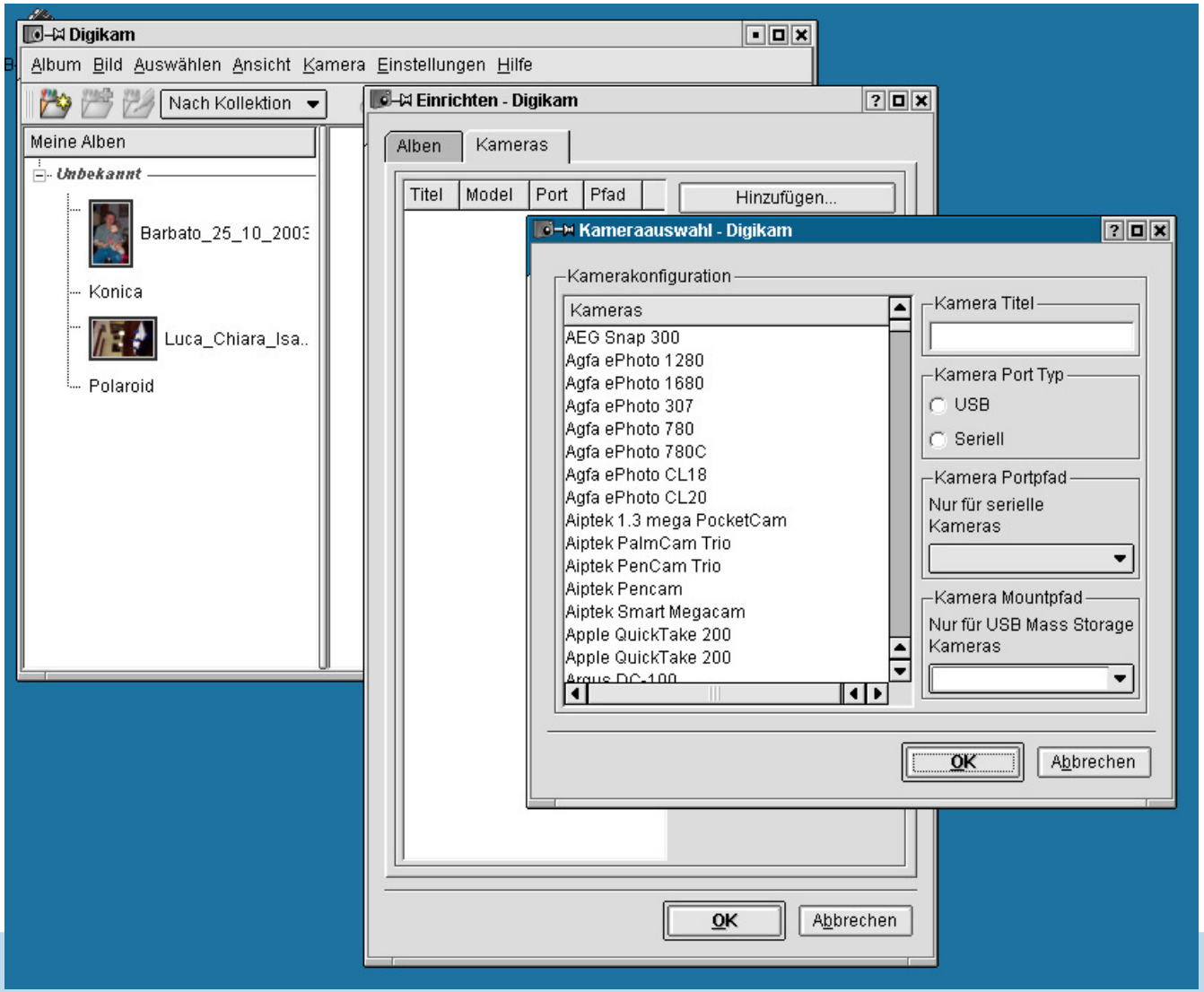
Digital Kameras

Als grafische Frontends gibt es “digikam“ und “gtkam“.
Über die Konfigurationsmenüs lässt sich ebenfalls herauskriegen ob eine Kamera unterstützt wir.



Digital Kameras

digikam



Modem, ISDN, DSL

Für USB- Modem, ISDN-, DSL- Adapter gibt es auch einen USB Klassen Treiber “acm“.
Die USB Drucker sind damit über “/dev/usb/ttyACMX“ bzw. “/dev/ttyACMX“ ansprechbar.

Troubleshooting

Troubleshooting

Was kann man tun wenn ein USB Gerät nicht will?
Leider gibt es kein Patentrezept, deshalb:

- Informationen einholen
- Im Internet suchen
- Foren und Mailinglisten besuchen

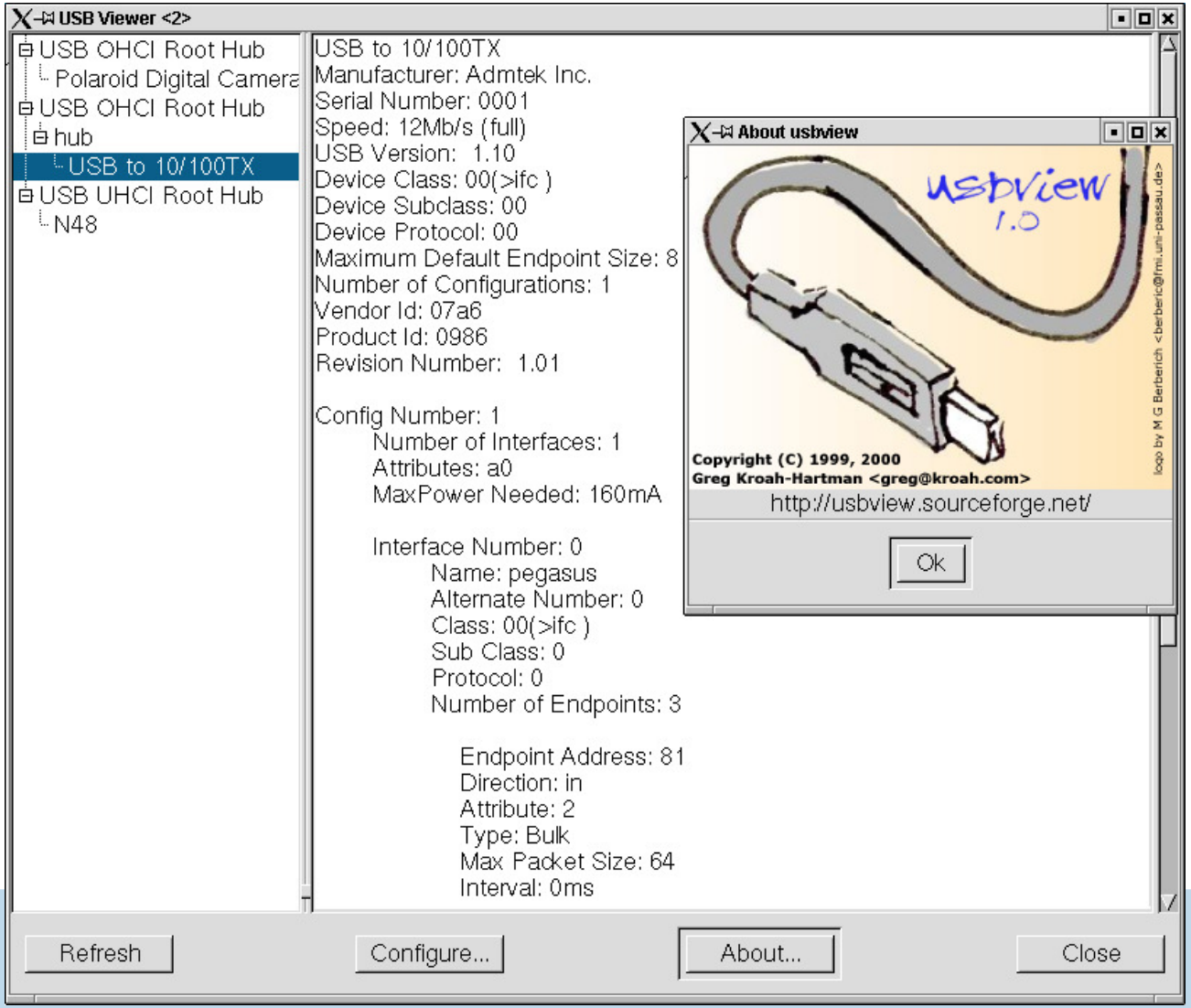
usbdevfs

Um Informationen über das USB System zu bekommen sollte unbedingt das usbdevfs gemountet sein. Ist dies nicht der Fall tut man sich schwer. Außerdem könnte das bereits die Fehlerursache sein da verschiedene Applikationen und Subsysteme usbdevfs benötigen.

```
logic:/# mount
...
usbfs on /proc/bus/usb type usbfs (rw)
...
```

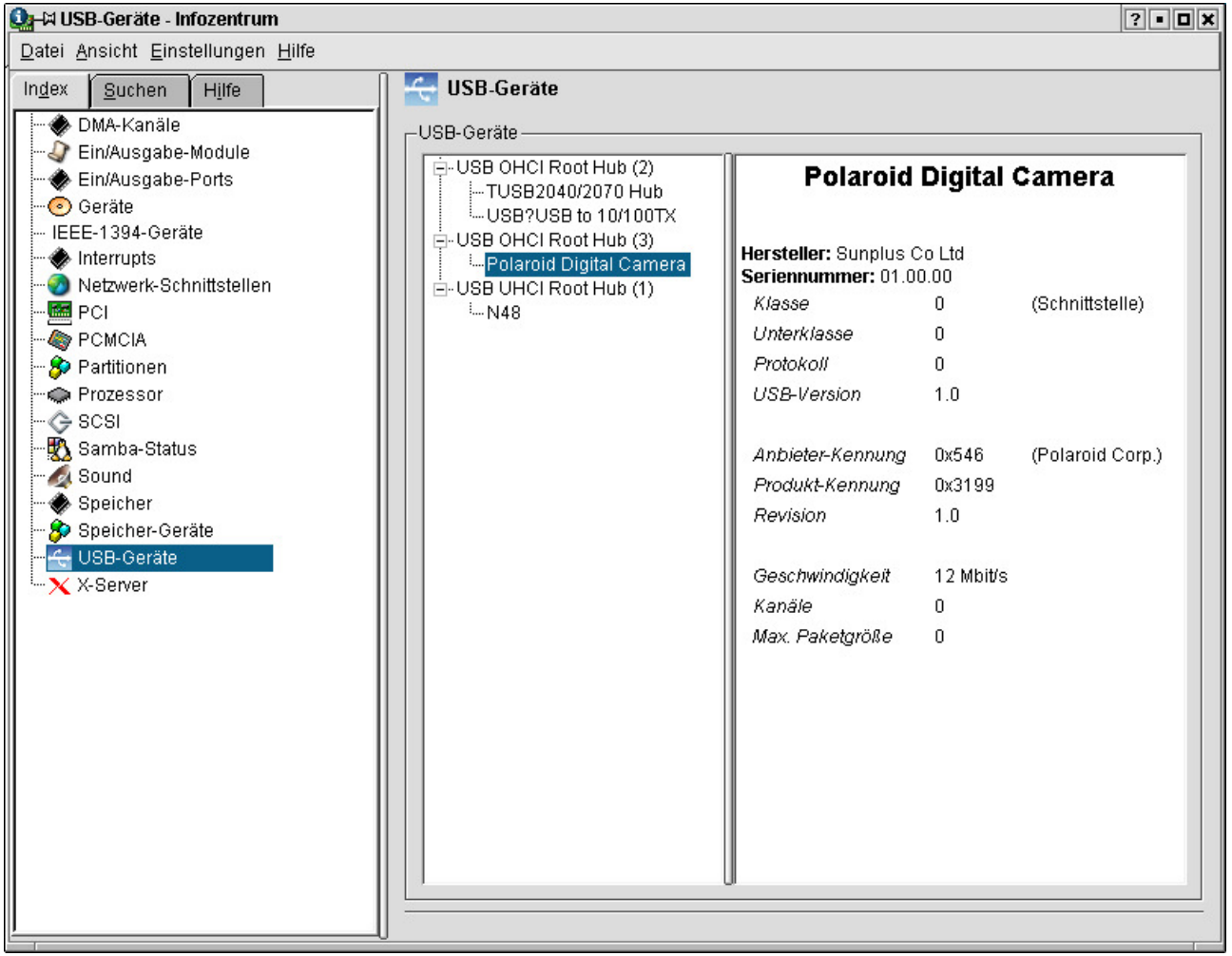
usbview

usbview ist ein grafisches Frontend zum usbdevfs. Damit lassen sich bequem einige Informationen abrufen.



KDE Infozentrum

Ähnlich wie usbview gehts auch mit dem KDE Infozentrum.



usbutils

usbmodules:
Damit kann man herausfinden welches Modul für ein USB Device zuständig ist. Setzt allerdings vorraus das ein solches Modul auf den System existiert.

```
logic:/# usbmodules -d /proc/bus/usb/002/003
pegasus
```

usbutils

lsusb:
Zeigt die Devices die am USB angeschlossen sind. Mit lsusb -v auch etwas ausführlicher.

```
logic:/# lsusb
Bus 003 Device 001: ID 0000:0000
Bus 002 Device 001: ID 0000:0000
Bus 001 Device 001: ID 0000:0000
Bus 001 Device 002: ID 046d:c001 Logitech, Inc. N48/M-BB
```

usbutils


```
logic:/# lsusb -v
...
Bus 001 Device 002: ID 046d:c001 Logitech, Inc. N48/M-BB
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               1.00
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0         8
  idVendor           0x046d Logitech, Inc.
  idProduct          0xc001 N48/M-BB48 [FirstMouse Plus]
  bcdDevice            4.00
  iManufacturer           1 Logitech
  iProduct                2 N48
...
```

modinfo

modinfo:
Die Umkehrung von usbmodules

```
logic:~# modinfo /lib/modules/2.4.25-rc1/kernel/drivers/
filename:    /lib/modules/2.4.25-rc1/kernel/drivers/usb/
description: "Pegasus/Pegasus II USB Ethernet driver"
author:      "Petko Manolov <petkan@users.sourceforge.ne
license:     "GPL"
parm:        loopback int, description "Enable MAC loopb
parm:        mii_mode int, description "Enable HomePNA m
```

/proc/bus/usb

Man kann natürlich auch das usbdevfs direkt bemühen. z. B.

```
logic:~# cat /proc/bus/usb/devices
...
T:  Bus=01 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=1.
D:  Ver= 1.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs
P:  Vendor=046d ProdID=c001 Rev= 4.00
S:  Manufacturer=Logitech
S:  Product=N48
C:* #Ifs= 1 Cfg#= 1 Atr=a0 MxPwr= 26mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=01 Prot=02 D
E:  Ad=81(I) Atr=03(Int.) MxPS=   8 Ivl=10ms
...
```

/var/log/messages

Ein Blick in “/var/log/messages“ kann auch weiterhelfen.

```
logic:/# tail -n 150 /var/log/messages | grep usb
... kernel: usb.c: registered new driver usbdevfs
... kernel: usb.c: registered new driver hub
...
... kernel: usb.c: new USB bus registered, assigned bus
... kernel: usb-uhci.c: v1.275:USB Universal Host Contro
... kernel: usb-ohci.c: USB OHCI at membase 0xd0ced000,
... kernel: usb-ohci.c: usb-02:06.0, NEC Corporation USB
...
... kernel: usb.c: registered new driver usbmouse
... kernel: input0: Logitech N48 on usb1:2.0
... kernel: usbmouse.c: v1.6:USB HID Boot Protocol mouse
... kernel: usb.c: registered new driver hiddev
... kernel: usb.c: registered new driver hid
```

firmware

Manche USB Geräte benötigen eine Firmware die zum einen vom Treiber aber auch von hotplug in das Gerät geladen werden kann. Hotplug benötigt hierfür weitere Programme z. B. fxload. Hier muss geprüft werden ob diese Programme vorhanden sind. Außerdem muss natürlich die Firmware vorhanden sein.

Fragen???

Basteln mit USB und Linux für Fortgeschrittene

USB allgemein

Geschichtliches

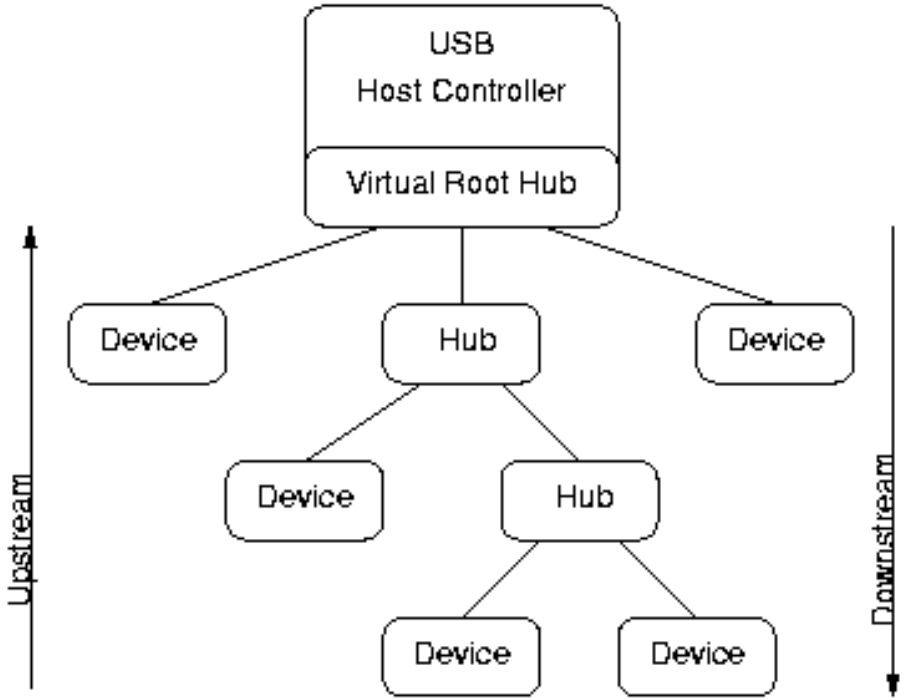
- 1994 spezifizierte eine Allianz von Compaq, Intel, Microsoft, NEC den Universal Serial Bus (USB)
- 1996 wurde die Version 1.0 verabschiedet
- 1998 folgte Version 1.1
- 2000 wurde die zur Zeit aktuelle Version 2.0 herausgegeben

Ziele

- einfach zu verwendendes System um Computer Peripherie anzuschließen
- hotplug fähig
- leicht erweiterbar
- soll das Ressourcen Problem lösen (Interrupts)
- geringe Kosten

Aufbau

- hierarchisches System
- Master/Slave Protokoll
- die Initiative geht immer vom Host aus
- Verteilung erfolgt über Hubs



[1]

Daten

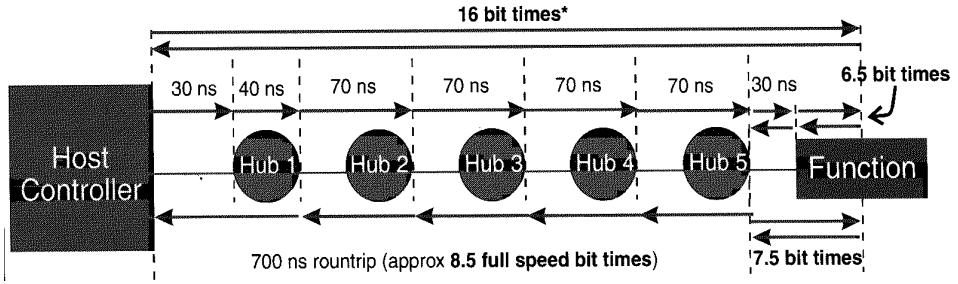
- Geschwindigkeiten
  - Low-Speed 1,5 Mb/s (Mega Bit!)
  - Full-Speed 12 Mb/s
  - High-Speed 480 Mb/s
- max. 127 Geräte an einem Host

“Elektrisches“

- 5V Spannungsversorgung über USB Kabel
- selfpowered / buspowered Devices
- ein USB Port kann max. 500mA liefern
- ein USB Port muss min. 100mA liefern
- Kabellänge für Low-Speed max. 3m
  einfachere Kabel, immer fest “angewachsen“
- Kabellänge für High/Full-Speed max. 5m (Laufzeit)

“Elektrisches“

Die maximale Kabelläge ist vor allem durch die Laufzeit begrenzt.



[6]

Host Controllers

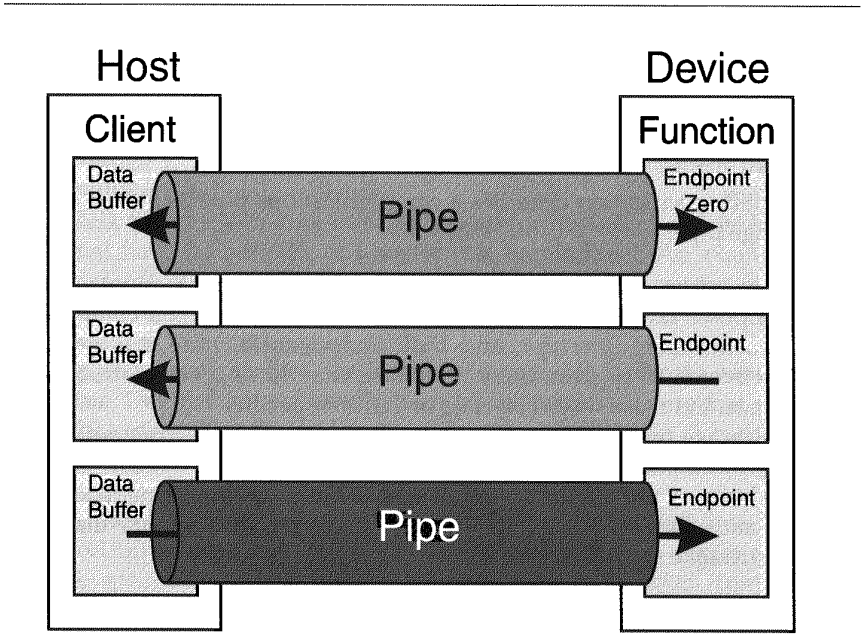
In der PC- Welt gibt es hauptsächlich drei Kategorien von Host Controllern.

- USB 1.1
  - UHCI
    - von Intel (PIIX4, VIA, ...)
    - “simplere“ Hardware
    - komplexerer Treiber
  - OHCI
    - von Compaq (NEC, iMacs, OPTi, SiS, ALi, NEC, ...)
- USB 2.0
  - EHCI

Entsprechend sind die Linux Kernel Module zu laden.

Bindung zum Device

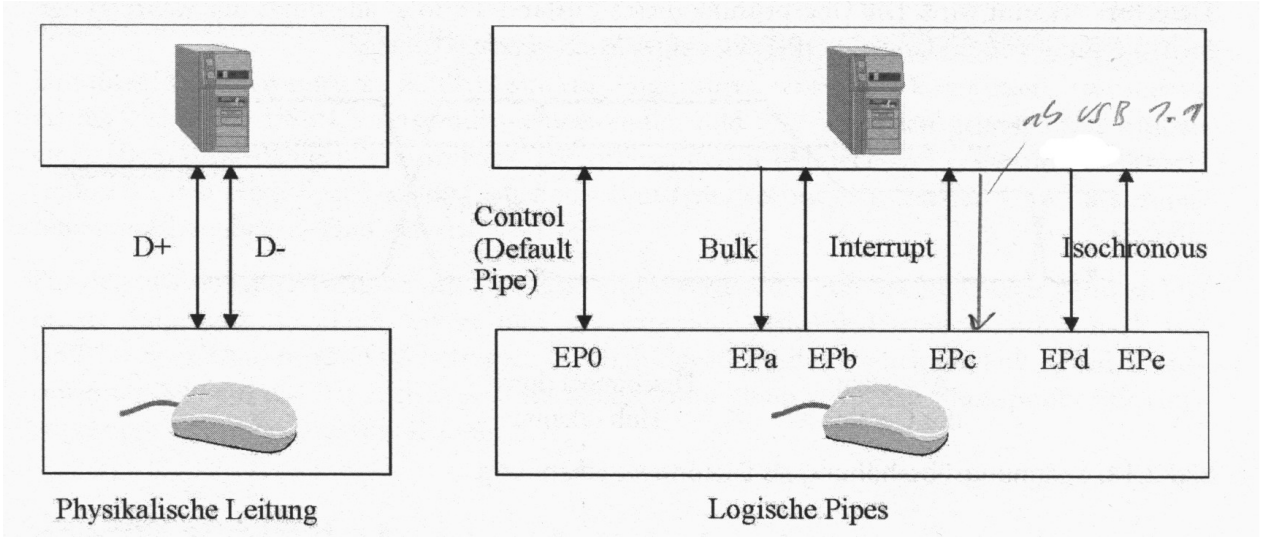
Die Kommunikation zwischen Host und Devices läuft über Endpoints.



[6]

Bindung zum Device

Es gibt unterschiedliche Arten von Endpoints bzw. deren Transferarten.



[5]

Transfer Arten

- **Control transfers**

  für kurze sichere Datenpakete.

  für die Konfiguration der Devices

  jedes Device benötigt ein Minimum an Control Commands
  - GET_STATUS
  - CLEAR_FEATURE
  - SET_FEATURE
  - SET_ADDRESS
  - GET_DESCRIPTOR
  - SET_DESCRIPTOR
  - GET_CONFIGURATION
  - SET_CONFIGURATION
  - GET_INTERFACE
  - SET_INTERFACE

Transfer Arten

- **Bulk Transfers**
  für lange sichere Datenpakete
  verwendet fast die gesammte Busbandbreite
  hat aber keine Priorität auf die Bandbreite
- **Interrupt Transfers**
  ist dem Bulk Transfer ähnlich
  wird periodisch gepollt
  max. 90% Bandbreite zusammen mit Isochronous Transfers
- **Isochronous Transfers**
  sendet/empfangt Daten in Echtzeit
  Datenübertragung nicht sicher
  für Audio und Video
  max. 90 % Bandbreite zusammen mit Interrupt transfers

Transfer Arten

Transferart	*FIFO-Tiefe in Byte*		*Fehlerkorrektur*	*Bandbreite*
Control	Low-Speed:	8	ja	max. 10 % garantiert
	Full-Speed:	8, 16, 32, 64		
	High-Speed:	64		
Interrupt	Low-Speed:	1...8	ja	max. 90% garantiert
	Full-Speed:	1...64		(zusammen mit Isochronous)
	High-Speed:	1...3.072		
Bulk	Full-Speed:	8, 16, 32, 64	ja	nur bei verfügbarer Bandbreite
	High-Speed:	512		
Isochronous	Full-Speed:	1...1.023	nein	max. 90% garantiert
	High-Speed:	1...3.072		(zusammen mit Interrupt)

[5]

Bandbreite

Transferart	*Protokoll-Overhead in Byte*	*Maximale FIFO-Tiefe in Byte*	*Übertragungen pro Microframe*	*Max. erreichbare Bandbreite*
Control	45	64	max. 1 (EHCI)	4,0 MBit/s
Interrupt	13	3 x 1.024	1	192 MBit/s
Bulk	13	512	max. 12	384 MBit/s
Isochronous	9	3 x 1.024	1	192 MBit/s

Tab. 2.6: Maximale Datenraten für High-Speed-Geräte

Transferart	*Protokoll-Overhead in Byte*	*Maximale FIFO-Tiefe in Byte*	*Übertragungen pro Frame*	*Max. erreichbare Bandbreite*
Control	45	64	max. 13 (OHCI)	6,6 MBit/s
			Max. 1 (UHCI)	512 KBit/s
Interrupt	13	64	1	512 KBit/s
Bulk	13	64	max. 19	9,7 MBit/s
Isochronous	9	1.023	1	8,2 MBit/s

Tab. 2.7: Maximale Datenraten für Full-Speed-Geräte

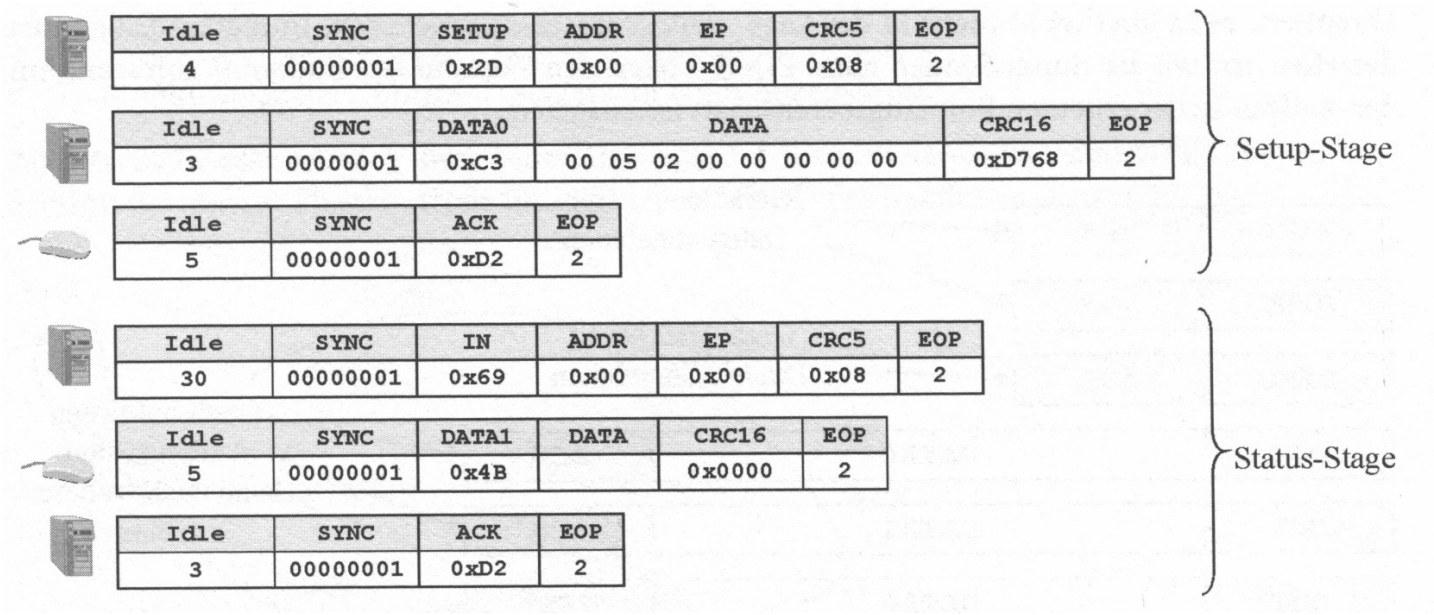
[5]

Transferart	*Protokoll-Overhead in Byte*	*Maximale FIFO-Tiefe in Byte*	*Übertragungen pro Frame*	*Max. erreichbare Bandbreite*
Control	46	8	Max. 3 (OHCI)	192 KBit/s
			Max. 1 (UCHI)	64 KBit/s
Interrupt	13	8	1 je 8 Frames	8 KBit/s

[5]

Low-Speed

Control transfer



Idle	SYNC	SETUP	ADDR	EP	CRC5	EOP
4	00000001	0x2D	0x00	0x00	0x08	2

Idle	SYNC	DATA0	DATA	CRC16	EOP
3	00000001	0xC3	00 05 02 00 00 00 00 00	0xD768	2

Idle	SYNC	ACK	EOP
5	00000001	0xD2	2

Setup-Stage

Idle	SYNC	IN	ADDR	EP	CRC5	EOP
30	00000001	0x69	0x00	0x00	0x08	2

Idle	SYNC	DATA1	DATA	CRC16	EOP
5	00000001	0x4B		0x0000	2

Idle	SYNC	ACK	EOP
3	00000001	0xD2	2

Status-Stage

[5]

Control transfer

Idle	SYNC	SETUP	ADDR	EP	CRC5	EOP
4	00000001	0x2D	0x00	0x00	0x08	2

Idle	SYNC	DATA0	DATA	CRC16	EOP
3	00000001	0xC3	80 06 00 01 00 00 40 00	0xBB29	2

Idle	SYNC	ACK	EOP
5	00000001	0xD2	2

Setup-Stage

Idle	SYNC	IN	ADDR	EP	CRC5	EOP
30	00000001	0x69	0x00	0x00	0x08	2

Idle	SYNC	DATA1	DATA	CRC16	EOP
5	00000001	0x4B	12 01 00 01 09 01 00 08	0x82DD	2

Idle	SYNC	ACK	EOP
3	00000001	0xD2	2

n x Data-Stage

Idle	SYNC	OUT	ADDR	EP	CRC5	EOP
30	00000001	0xE1	0x00	0x00	0x08	2

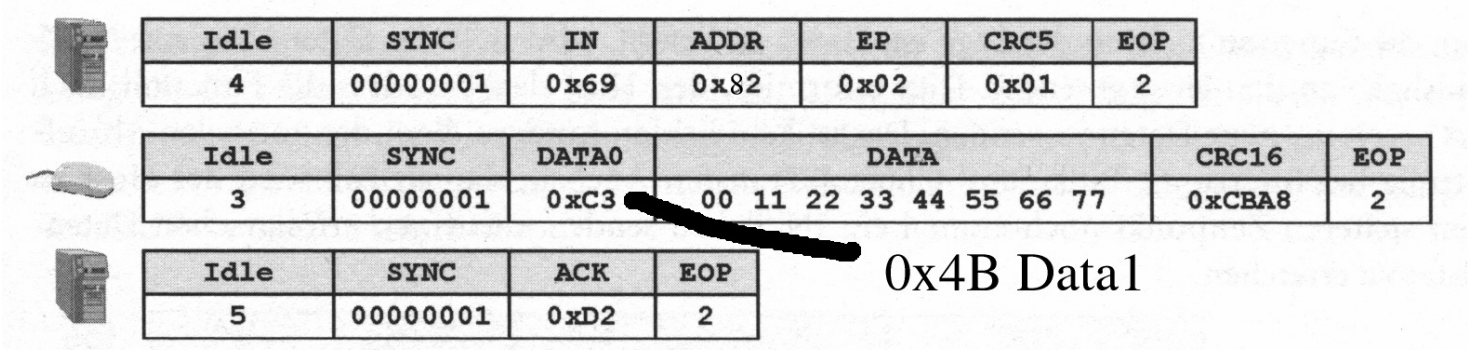
Idle	SYNC	DATA1	DATA	CRC16	EOP
3	00000001	0x4B		0x0000	2

Idle	SYNC	ACK	EOP
5	00000001	0xD2	2

Status-Stage

[5]

IN transfer



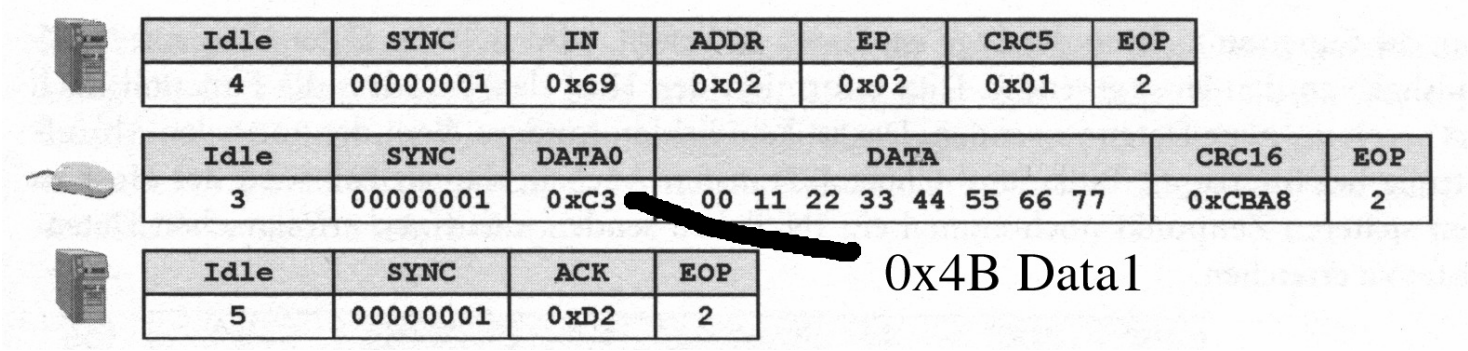
Idle	SYNC	IN	ADDR	EP	CRC5	EOP
4	00000001	0x69	0x82	0x02	0x01	2

Idle	SYNC	DATA0	DATA	CRC16	EOP
3	00000001	0xC3	00 11 22 33 44 55 66 77	0xCBA8	2

Idle	SYNC	ACK	EOP
5	00000001	0xD2	2

[5]

OUT transfer



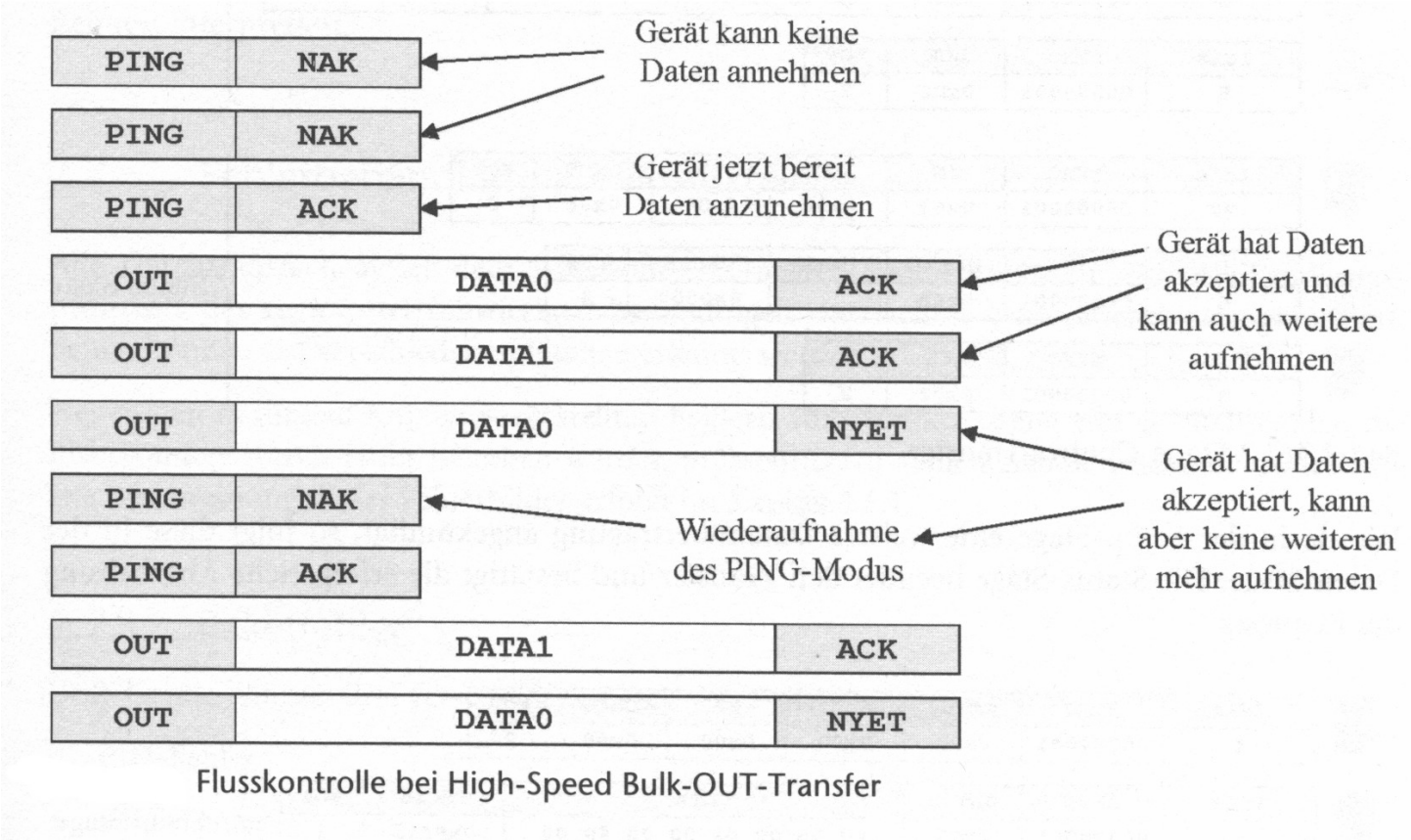
Idle	SYNC	IN	ADDR	EP	CRC5	EOP
4	00000001	0x69	0x02	0x02	0x01	2

Idle	SYNC	DATA0	DATA	CRC16	EOP
3	00000001	0xC3	00 11 22 33 44 55 66 77	0xCBA8	2

Idle	SYNC	ACK	EOP
5	00000001	0xD2	2

[5]

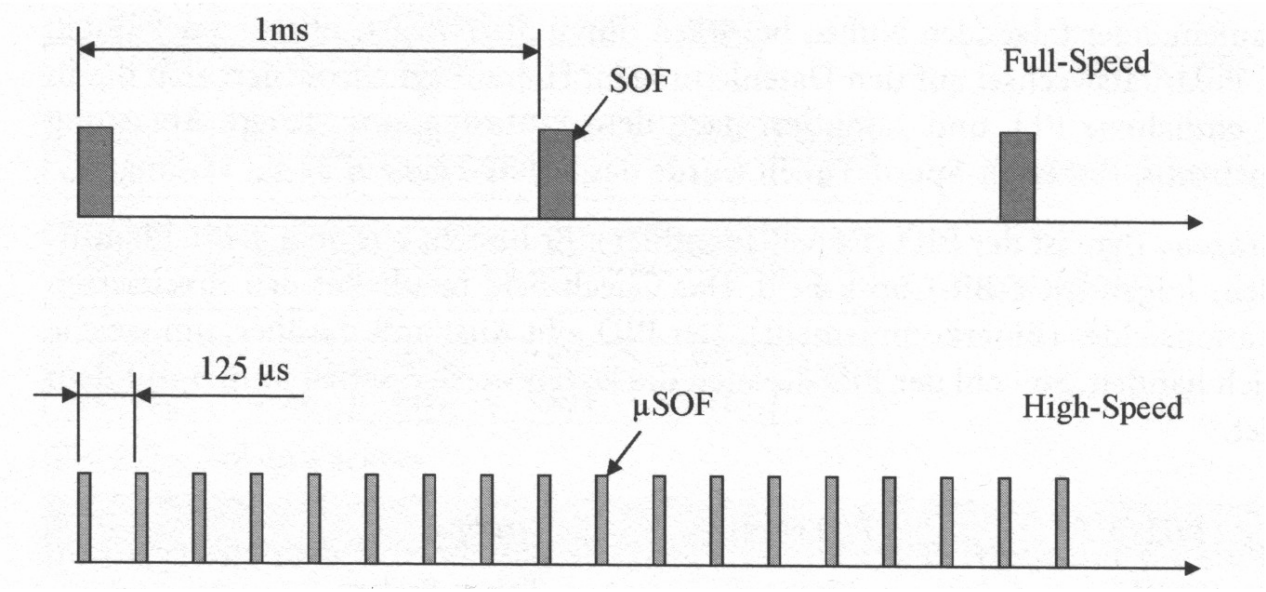
PING - NAK



Flusskontrolle bei High-Speed Bulk-OUT-Transfer

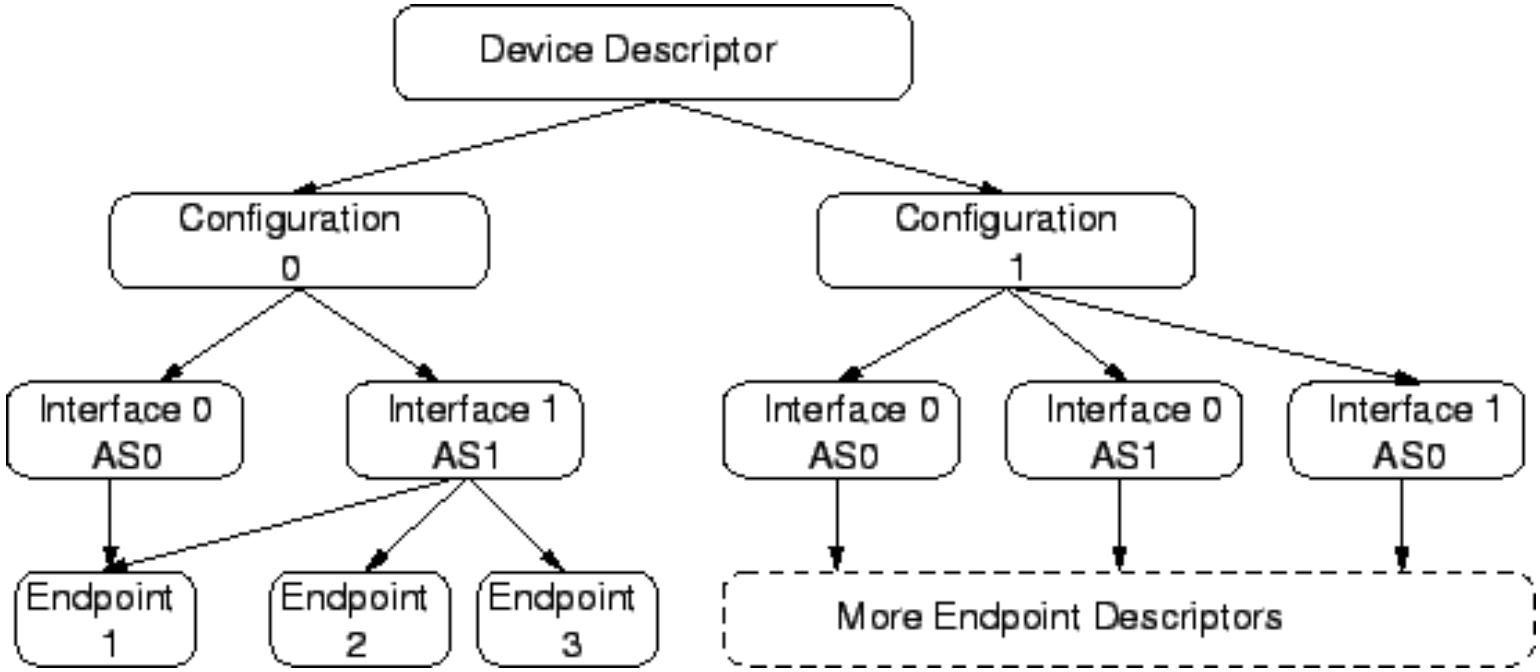
[5]

frames



[5]

Descriptor



[8]

Descriptor

Offset	*Feldbezeichnung*	*Länge*	*Beschreibung*	*Beispielwerte*
0	*bLength*	1	Größe dieses Deskriptors in Byte	0x12
1	*bDescriptorType*	1	Deskriptortyp = Device-Descriptor	0x01
2	*bcdUSB*	2	Version der USB-Spec.(z.B. 1.1)	0x10, 0x01
4	*bDeviceClass*	1	Klassen-Code	0x00
5	*bDeviceSubClass*	1	Subklassen-Code	0x00
6	*bDeviceProtocoll*	1	Protokoll-Code	0x00
7	*bMaxPacketSize0*	1	Tiefe der EP0-FIFO in Byte (z.B. 8)	0x08
8	*idVendor*	2	Vendor-ID des Herstellers (z.B. IMMS)	0x3C, 0x05
10	*idProduct*	2	Produkt-ID (z.B.0x9084)	0x84, 0x90
12	*bcdDevice*	2	Release-Nr. des Produkts (z.B.1.02)	0x02, 0x01
14	*iManufacturer*	1	String-Index für »Hersteller«	0x01
15	*iProduct*	1	String-Index für »Produkt«	0x02
16	*iSerialNumber*	1	String-Index für »Seriennummer«	0x03
17	*bNumConfigurations*	1	Zahl möglicher Konfiguratione	0x01

Tab. 2.15: Aufbau des Device-Descriptor

[5]

Descriptor

Offset	*Feldbezeichnung*	*Länge*	*Beschreibung*	*Beispielwerte*
0	*bLength*	1	Größe dieses Deskriptors in Byte	0x09
1	*bDescriptorType*	1	Deskriptortyp = Configuration-Descriptor	0x02
2	*wTotalLength*	2	Länge aller zu dieser Konfiguration gehörenden Deskriptoren (z.B.34)	0x22, 0x00
4	*bNumInterfaces*	1	Anzahl der zu dieser Konfiguration gehörenden Interfaces (z.B.1)	0x01
5	*bConfigurationValue*	1	Nummer dieser Konfiguration (z.B.1) (Argument für SetConfiguration)	0x01
6	*iConfiguration*	1	String-Index für »Konfiguration«	0x04
7	*bmAttributes*	1	Attribute für diese Konfiguration (z.B. Bus-powered, Remote-Wakeup-Support)	0xA0
8	*MaxPower*	1	Stromaufnahme in dieser Konfiguration in 2 mA-Einheiten (z.B.50 mA)	0x1A

Tab. 2.16: Aufbau des Configuration-Descriptor

[5]

Descriptor

Offset	*Feldbezeichnung*	*Länge*	*Beschreibung*	*Beispielwerte*
0	*bLength*	1	Größe dieses Deskriptors in Byte	0x09
1	*bDescriptorType*	1	Deskriptortyp = Interface	0x04
2	*bInterfaceNumber*	1	Interface-Nummer (z.B.0)	0x00
3	*bAlternateSetting*	1	AlternateSetting für dieses Interface	0x00
4	*bNumEndpoints*	1	Anzahl der zu diesem Interface gehörenden Endpoints ohne EP0 (z.B.1)	0x01
5	*bInterfaceClass*	1	Klassen-Code (Tastatur)	0x03
6	*bInterfaceSubClass*	1	Subklassen-Code (Boot-Device)	0x01
7	*bInterfaceProtocol*	1	Protokoll-Code (z.B.0)	0x00
8	*iInterface*	1	String-Index für »Interface«	0x05

Tab. 2.18: Aufbau des Interface-Descriptor

[5]

Descriptor

Offset	*Feldbezeichnung*	*Länge*	*Beschreibung*	*Beispielwerte*
0	*bLength*	1	Größe dieses Deskriptors in Byte	0x07
1	*bDescriptorType*	1	Deskriptortyp = Endpoint	0x05
2	*bEndpointAddress*	1	Endpoint-Adresse (z.B.IN, EP1)	0x81
3	*bmAttributes*	1	Transferart (z.B.Interrupt-Endpoint)	0x03
4	*wMaxPacketSize*	2	FIFO-Größe des Endpoints (z.B.8 Byte)	0x08, 0x00
6	*bIntervall*	1	Polling-Intervall (z.B.16 ms)	0x10

Tab. 2.19: Aufbau eines Endpoint-Descriptor (USB 1.1)

[5]

Descriptor


```
logic:/# lsusb -v
...
Bus 001 Device 002: ID 046d:c001 Logitech, Inc. N48/M-BB
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               1.00
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0         8
  idVendor           0x046d Logitech, Inc.
  idProduct          0xc001 N48/M-BB48 [FirstMouse Plus]
  bcdDevice            4.00
  iManufacturer           1 Logitech
  iProduct                2 N48
...
```

Linux und USB

zwei Zurgriffsarten

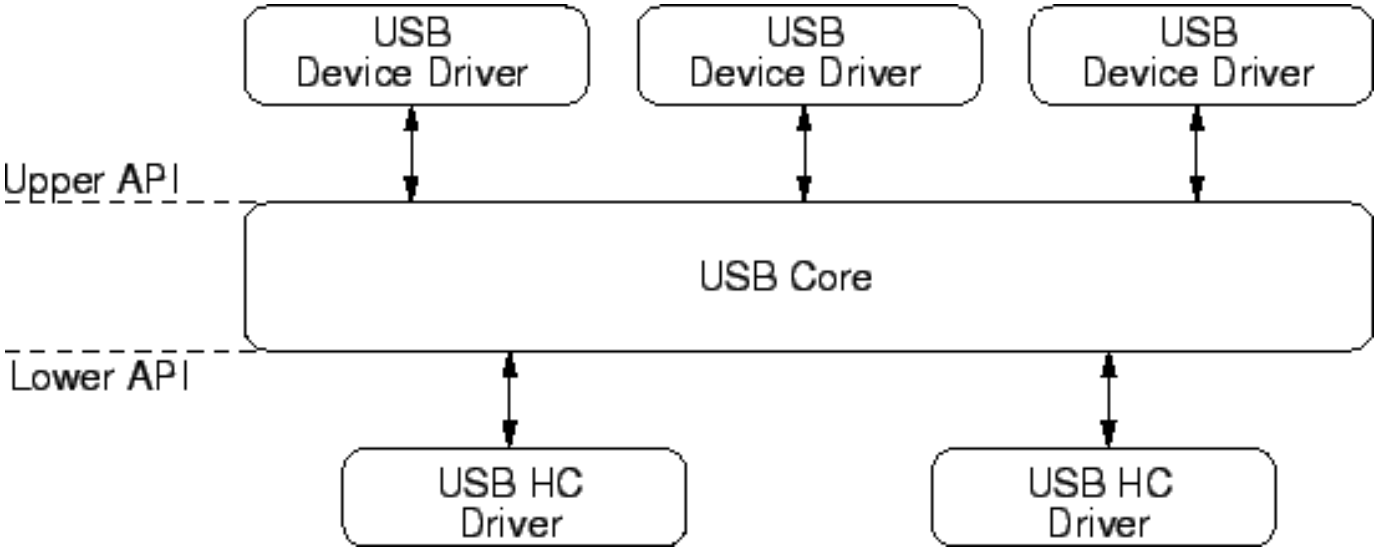
Um auf USB Devices zuzugreifen gibt es zwei Möglichkeiten

- usbdevfs (libusb)
- Kernel Treiber

Der Referent beschränkt sich auf die Kernel Treiber

Linux Kernel USB Subsystem

Das Linux USB Subsystem stellt eine API zur Verfügung die die **Treiberimplementierung unabhängig vom Host Controller** macht.



[8]

Linux USB Kernel Modul Treiber

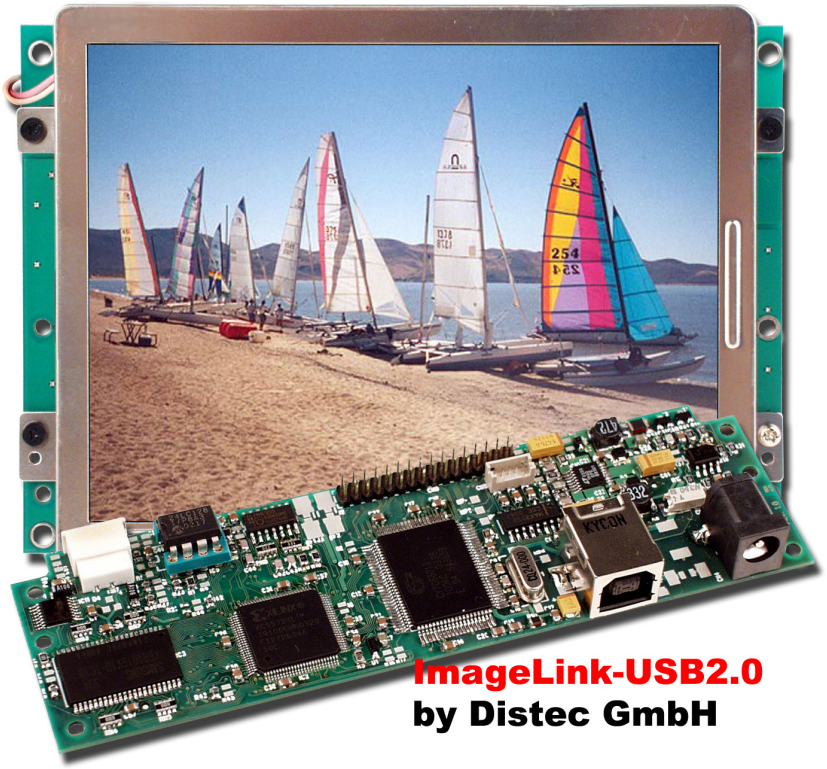
Wird ein Kernel Modul geladen bzw. entladen werden die bekannten “**init**“ bzw. “**exit**“ Funktionen aufgerufen. Innerhalb dieser Funktionen wird der USB Treiber registriert bzw. abgemeldet.
Da es sich bei USB Devices um **Hotplug Geräte** handelt findet die Initialisierung in der “**probe**“ Funktion statt. Diese wird aufgerufen wenn ein USB Device eingesteckt wird bzw. ein vorhandenes Gerät noch keinen passenden Treiber zugeordnet hat und der neu geladene Treiber dafür zuständig ist. Die Zuordnung erfolgt anhand der USB IDs. Wenn ein USB Device abgesteckt wird räumt “**disconnect**“ auf.

Linux USB Treiber


```
#include <linux/usb.h>

static struct usb_driver sample_usb_driver = {
        name:         "sample",
        probe:        sample_probe,
        disconnect:   sample_disconnect,
};

int init_module(void)
{
    /* nur registrieren, gibt 0 oder einen Fehler-Code zurück */
    return usb_register(&sample_usb_driver);
}

void exit_module(void)
{
    usb_deregister(&sample_usb_driver);
}
```

[9]

Beispiel ImageLink



[10]

Beispiel ImageLink


```
static struct usb_driver dd_lou_usb_driver = {
        name:         "dd_lou",
        probe:        dd_lou_probe,
        disconnect:   dd_lou_disconnect,
        id_table:     dd_lou_id_table,
        minor:        DD_LOU_MINOR_BASE,
        fops:         &dd_lou_file_operations,
};
```

Beim 2.6.x Kernel müssen die “file_operations“ mit der struct usb_class_driver mit usb_register_dev() in der “probe“ Funktion übergeben werden und mit usb_deregister() in “disconnect“ wieder entfernt werden.

Beispiel ImageLink


```
static struct usb_device_id dd_lou_id_table [] = {
  { USB_DEVICE_VER(USB_VENDOR_ID_DD, USB_PROD_ID_DD_LOU, \
       0x0000, 0x9999), driver_info: (unsigned long)"lou" },
    {
      0, /* no more matches */
    }
};
```

Beispiel ImageLink


```
static void *dd_lou_probe(struct usb_device *udev, unsigned int ifnum,
                          const struct usb_device_id *id)
{
...
if (interface->bNumEndpoints != 2) return NULL;   // we have EP1 and E
...
```

Beim 2.6.x Kernel ist der erste Parameter vom Typ struct usb_interface. Mit interface_to_usbdev(intreface) bekommt man wieder die struct usb_device.

URB (USB Request Block)

Der Datentransfer zwischen Host und Device geschieht über **URB**s (USB Request Block).
Das ist eine Datenstruktur die neben den Nutzdaten auch Device- und Adress- Informationen enthält.

URB (USB Request Block)

Eine Hilfsfunktion um einen URB für einen Bulk Transfer zu füllen. Weiter sind in **usb.h** zu finden.

```
static inline void usb_fill_bulk_urb (struct urb *urb,
          struct usb_device *dev,
          unsigned int pipe,
          void *transfer_buffer,
          int buffer_length,
          usb_complete_t complete,
          void *context)
```

URB (USB Request Block)

Eine Hilfsfunktion um einen URB für einen **Bulk Transfer** zu füllen.

```
* usb_fill_bulk_urb - macro to help initialize a bulk urb
* @urb: pointer to the urb to initialize.
* @dev: pointer to the struct usb_device for this urb.
* @pipe: the endpoint pipe
* @transfer_buffer: pointer to the transfer buffer
* @buffer_length: length of the transfer buffer
* @complete: pointer to the usb_complete_t function
* @context: what to set the urb context to.
```

Beispiel ImageLink


```
/* fill the URB data structure using the FILL_INT_URB macro */
 {
   int pipe = usb_rcvintpipe(udev, endpoint1->bEndpointAddress);
   int maxp = usb_maxpacket(udev, pipe, usb_pipeout(pipe));

   if (maxp > sizeof(IntTransfer) )
     maxp = sizeof(IntTransfer);
   dd_lou->maxp = maxp; /* remember for later */
   // ###sw1 macro deprecated
   FILL_INT_URB(&dd_lou->irq_urb, udev, pipe, dd_lou->data, maxp,
                              dd_lou_irq, dd_lou, endpoint1->bInterval
 }
```

URB (USB Request Block)

Dann muss man den URB dem USB Subsystem noch unterbreiten (submiten).

```
int usb_submit_urb(struct urb *urb);
```

Beispiel ImageLink

submit eines Interrupt Transfers

```
  /* register the URB within the USB subsystem */
    if ( (test = usb_submit_urb(&dd_lou->irq_urb) ) ) {
      printk(KERN_WARNING "usbdd_lou: WARNING usb_submit_urb() %d fail
      kfree(dd_lou->send_buff);
      minor_dd_lou[lou_dev_nr] = NULL;
      kfree(dd_lou);
      dd_lou = NULL;
      goto bye;
    }
```

Beispiel ImageLink

Die Callback Funktion

```
static void dd_lou_irq(struct urb *irq_urb)
{
  IntTransfer *trans_data;
  struct dd_lou_device *dd_lou = irq_urb->context;

  trans_data = (IntTransfer *)dd_lou->data;
...
  if ( trans_data->error ){
    D1(printk(KERN_WARNING "usbdd_lou: dd_lou_irq(): dd_lou->error =0x
    dd_lou->error = trans_data->error;
   }
...
```

URB (USB Request Block)

Da es sich bei dem Beispiel um einen Interrupt Transfer handelt der periodisch wiederholt wird muss der URB wenn er nicht mehr gebraucht wird wieder aus dem System entfernt werden.

```
int usb_unlink_urb(struct urb *urb);
```

Beim 2.6.x Kernel gibt es einen weiteren Parameter der die Speicherreservierung regelt.

Beispiel ImageLink


```
static void dd_lou_disconnect(struct usb_device *udev, void *clientdat
{
    /* the clientdata is the dd_lou_device we passed originally */
    struct dd_lou_device *dd_lou = (struct dd_lou_device *)clientdata;

..
    usb_unlink_urb(&dd_lou->irq_urb);
...
```

URB (USB Request Block)

Eine andere Möglichkeit Daten auszutauschen ist:

```
int usb_bulk_msg(struct usb_device *usb_dev, \
    unsigned int pipe, void *data, int len,  \
    int *actual_length, int timeout);
```

Beispiel ImageLink


```
static ssize_t dd_lou_write(struct file *filp , const char *buffer, si
                            loff_t *offp)
{
...
   result = usb_bulk_msg(dd_lou->udev,
            usb_sndbulkpipe(dd_lou->udev, 2),
            send_buff, thistime, &partial, 10 * HZ);
...
```

URB (USB Request Block)

- usb_bulk_msg()
  ist synchron
- usb_submit_urb()
  ist asynchron

URB (USB Request Block)

Nachdem nun ganz nebenbei Interrupt und Bulk Transfer vorgestellt wurden wirn nun anhand eines Beispiels ein **Control Transfer** gezeigt.

```
static int dd_lou_ioctl(struct inode *inode, struct file *filp, \
                unsigned int cmd, unsigned long arg)
{...
  case DD_LOU_IOC_Q_GET_BKL:
    usb_control_msg(udev,usb_rcvctrlpipe(udev,0),VX_GET_BKL,0xc0,\
                    (__u16)arg,0x00,&dd_lou->ctrlret,0x02,HZ);
    return dd_lou->ctrlret;
  break;
...
```

Für Bastler

Cypress FX2

Um noch ein wenig Lust aufs Basteln zu machen ein paar Worte zum verwendetem USB- Microcontroller.

- Hersteller

  Cypress `http://www.cypress.com`
- Typ:

  CY7C68013
- CPU:

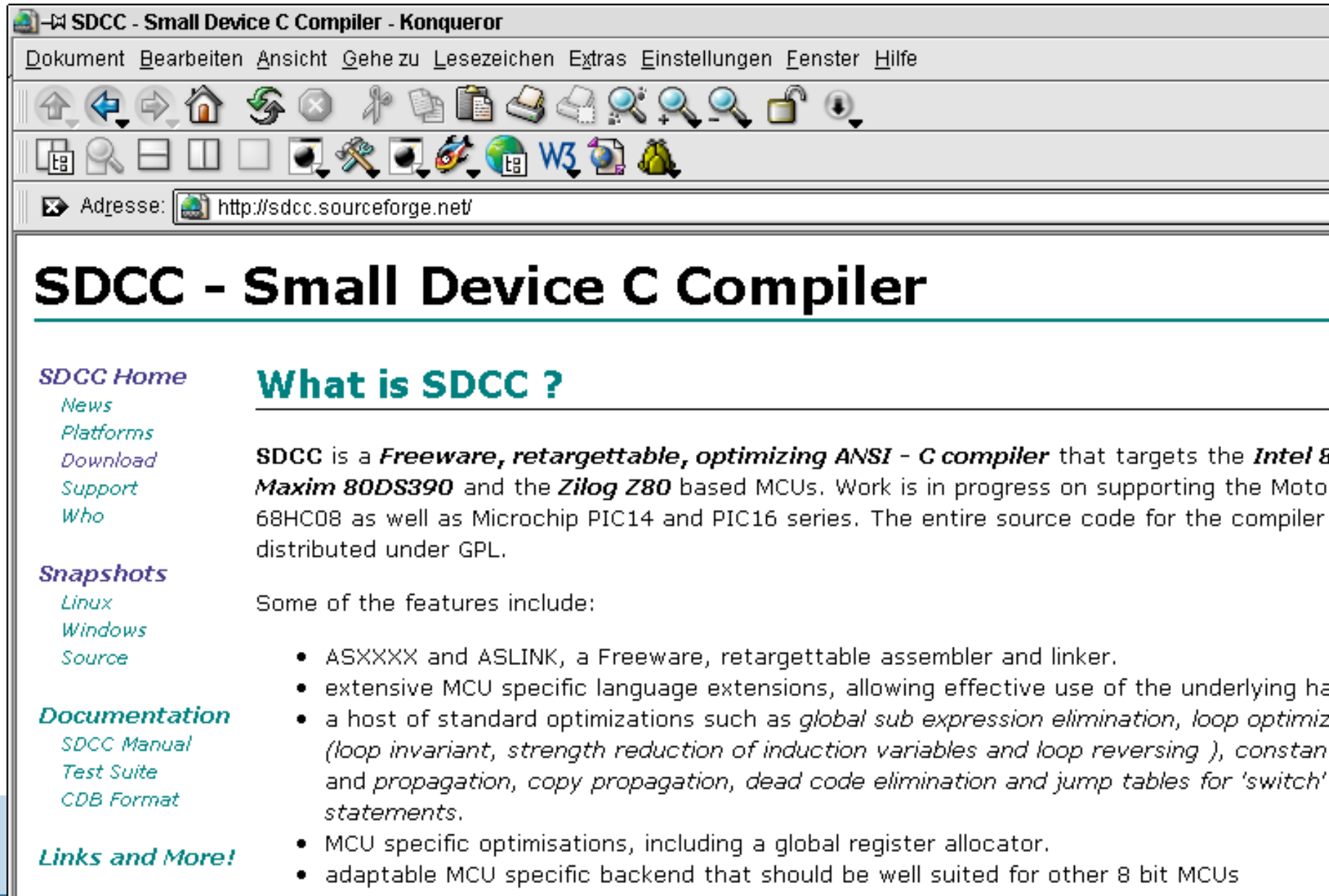
  8051

Cypress FX2

Natürlich gibt es noch andere Microcontroller mit USB Anschluss. Der FX2 bzw. ein anderer der FX Familie bietet dem Bastler aber ein paar nette Eigenschaften

- High-Speed tauglich (FX2)
- über USB programmierbar
- befindet sich z. B. in Keyspan USB Serial Adaptern (lässt sich hacken!!!)
- Dokumentation zum Download
- 8051 CPU läst sich mit sdcc programieren

sdcc



Fragen???

Quellen

[1] Brad Hards, The Linux USB sub-system - Sigma Bravo Pty Ltd., `http://www.linux-usb.org/USB-guide/book1.html`

[2] SANE - Scanner Access Now Easy, `http://www.sane-project.org/`

[3] USB - Universal Serial Bus, `http://www.usb.org/home`

[4] Linux USB Project, `http://www.linux-usb.org/`

[5] H.-J. Kelm (Hrsg.), USB 2.0 - Franzis Verlag, Poing, 2001

[6] Don Anderson, Universal Serial Bus System Architecture, Addison Wesley, 2003

Quellen

[7] gphoto, http://www.gphoto.org

[8] Detlef Fliegl, Programming Guide for Linux USB Device Drivers, http://wwwbode.cs.tum.edu/Par/arch/usb/usbdoc/usbdoc

[9] Alessandro Rubini, Jonathan Corbet - Linux-Gerätetreiber, 2. Auflage, http://www.oreilly.de/german/freebooks/linuxdrive2ge

[10] Distec GmbH, ImageLink-50-USB, http://www.distec.de/