Serielle Busse: Agenda

- Motivation für serielle Busse
- USB - Architektur
- USB - Treiber
- IEEE 1394 (FireWire)

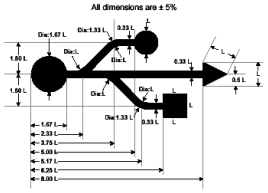


Figure 6-5. USB Icon

Serielle Busse: Literatur

www.usb.org	
developer.intel.com	
www.microsoft.com/hwdev/usb/	
Kelm (Hrsg).	USB, Universal Serial Bus, Franzis Verlag, 1999
usb.sourceforge.net	(USB for Linux homepage)
www.emsys.de/usb	(Hardware-Übersicht)
Don Anderson	Universal Serial Bus System Architecture, MindShare, 1997
	www.mindshare.com

www.1394.org	
www.microsoft.com/hwdev/1394	
www.firewire.org (=www.apple.com)	www.apple.com/firewire/firewireproducts.html
Don Anderson	FireWire System Architecture, MindShare, 1999
1394.sourceforge.net	(FireWire for Linux)
Mike Teener, 1394 overview	www.zayante.com

Serielle Busse: Motivation

warum neue Busse?

- Schnittstellengewirr des "legacy"-PC
- Überwinden der lästigen IRQ/DMA/IO Resourcenkonflikte
- Autokonfiguration
- Hot-Plugging
- Plattformunabhängige Standards (PC/Audio/Video/...)

warum serielle Busse?

- so billig wie möglich, nicht schneller als nötig
- möglichst lange Kabel
- Probleme mit Übersprechen/Skew bei paralleler Übertragung
- Chips sind billig, Kupfer ist teuer
  NE2000-Karte ca. DM 20.00, U2W-SCSI Kabel DM 300.00

Serielle Busse: USB vs. 1394

welcher Bus für welche Anwendung? (Microsoft unterstützt beide)

- USB: ausgerichtet auf PC und PC-Peripherie
  Single-Master-Architektur
- FireWire: eigenständige Multimediageräte
  Audio/Video-Protokolle etabliert
  kein PC als Master notwenig

Bus	Geschwindigkeit	Host-Komplexität	Peripherie-Komp.
IEEE 1394	400 Mb/sec.	12000-20000 gates	5000 - 7000 gates
USB 1.1	12 Mb/sec	10000 gates	2500 - 3000 gates

USB:

entwickelt seit ~1990 von
- Compaq DEC IBM Intel Microsoft NEC Northern Telecom
- große Ähnlichkeit zu Apples FireWire/IEEE 1394
- aber Ausrichtung auf PC und PC-Peripherie

- serieller, billiger Bus, bis zu 127 Geräte
- low/full speed mit 1.5 Mb/s und 12 Mb/s
- volle Autokonfiguration mit Hot-Plugging
- Geräte identifizieren sich für das Betriebssystem
- isochrone Transfers für Media-Processing

- USB 1.0 Q1 / 1996
- USB 1.1 Q3 / 1998
- USB 2.0 Q1 / 2000 (480 Mb/s)

USB: Ziele

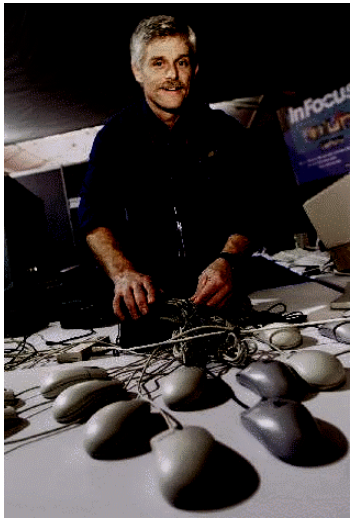
- preisgünstig (Stecker/Kabel/ASICs)
- einheitliche Stecker und Kabel (4 Pins, Typ A/B, max. 5m)
- unverwechselbare Stecker (Hub Typ A, Client Typ B)
- sehr viele Geräte (max. 127 Geräte)

- Hot-Plugging (Anstecken im Betrieb)
- zwei Geschwindigkeiten (1.5 Mb/s und 12 Mb/s)
- flexible Datenübertragung (4 Transferarten)

- Stromversorgung über das Kabel (5V, max. 500 mA)
- benötigt keine ISA-Ressourcen (IRQ/DMA/IO-Ports)
- Schlafzustand der Geräte (suspend nach 3 msec.)

- USB 2.0 deutlich schneller (bis 480 Mb/s)

USB: Intel IDF Demo



- 1996 nur wenige Geräte verfügbar
- seit HX/PIIX3 in jedem Intel Chipsatz

- mittlerweile etabliert
- Gerätevielfalt

USB: Gerätespektrum . . .

bisherige Schnittstelle:
- Tastaturen, Mäuse (PS/2, seriell)
- Joysticks, Gamepads (force feedback) (gameport)
- Monitore mit integrierten Hubs

- Drucker, Scanner (SCSI/parallel)
- Digitalkameras (seriell)

- Modems, ISDN-Adapter, USB-Netze (seriell)
- Wechselplatten, CDROM, ... (SCSI/parallel)

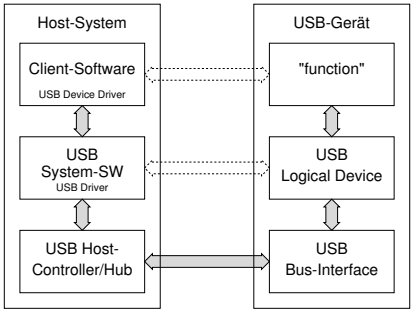
- Audio-Geräte (analog audio in/out)
- MIDI-Geräte (Tonstudios) (seriell)
- Kopierschutz-Adapter (seriell/parallel)
- . . .

USB: Aufgaben der Host-SW:

- detect attachment and removal of USB devices
- managing control flow between host and USB devices
- managing data flow between host and USB devices
- collecting status and activity statistics
- providing power to attached USB devices

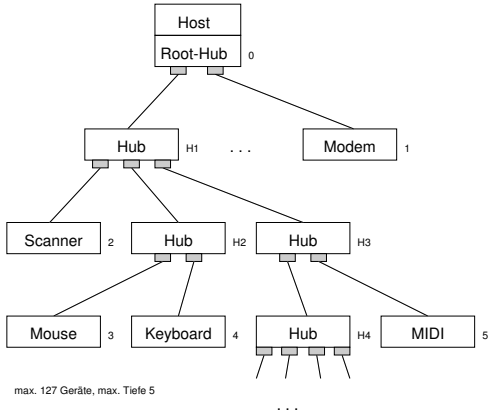
- device enumeration and configuration
- isochronous data transfers
- asynchronous data transfers
- power management
- device and bus management information

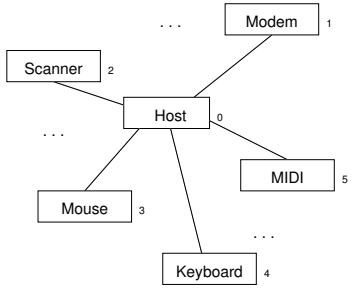
USB: Software-Architektur



- USB-Device-Driver — Client Requests -> IO request packets
- USB-Driver — IRPs -> Transactions
- Host-Controller-Driver — Frame List, Scheduling

USB: Architektur

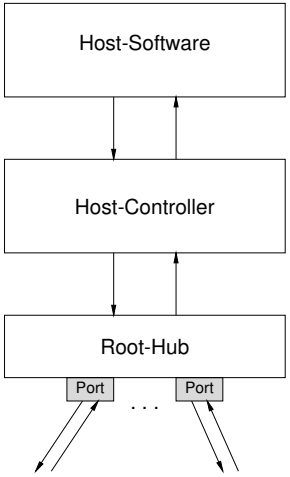




physikalische Struktur:
- Baum, Host an der Wurzel

logische Topologie:
- Stern, Host steuert alles

USB: Host-Controller und Root-Hub



- alle USB-Transfers unter Kontrolle der Host-Software

- Varianten: Open/Universal-HC
- Erzeugen der Bus-Transaktionen
- Abfolge der Transaktionen (Scheduling)
- De-/Serialisierung der Daten

- Steuerung der Stromzufuhr
- Freischalten/Sperren der Ports
- Dis-/Connect-Erkennung
- Status-Verwaltung der Ports

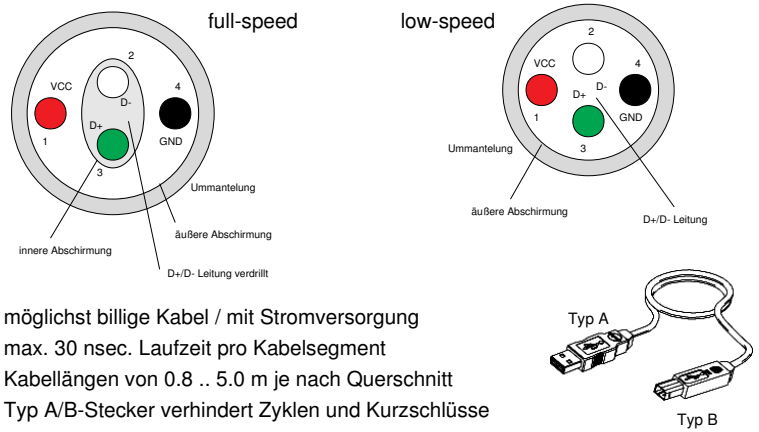
USB: Transferarten

vier separate Datentransfers:

- *Control-Transfer* — Host sendet Befehle an die Clients
  10% Bandbreite garantiert
- *Interrupt-Transfer* — Host fragt Clients nacheinander per Polling
  ob "Interrupt"-Ereignisse vorliegen
- *Bulk-Transfer* — eigentliche Datenübertragung ohne
  Latenz / Echtzeitanforderungen
- *Isochronous-Transfer* — Datenübertragung mit garantierter
  Latenz / Bandbreite, etwa für Audiodaten.

  90% Bandbreite für Interrupt/Isochron.
  Bulk-Transfers nur wenn Bus frei

USB: Kabel und Stecker

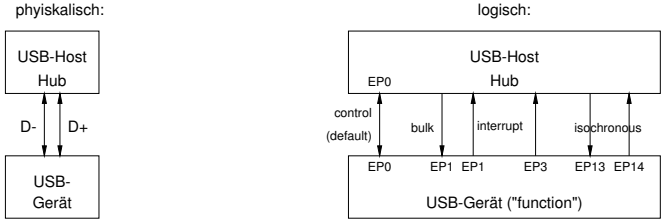


- möglichst billige Kabel / mit Stromversorgung
- max. 30 nsec. Laufzeit pro Kabelsegment
- Kabellängen von 0.8 .. 5.0 m je nach Querschnitt
- Typ A/B-Stecker verhindert Zyklen und Kurzschlüsse
- low-speed Geräte haben festes Kabel (< 3m)

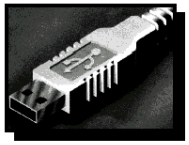
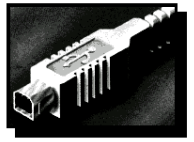
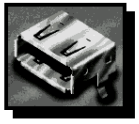
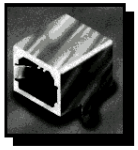
USB: Pipe / Endpoint

USB-"Pipe" :=

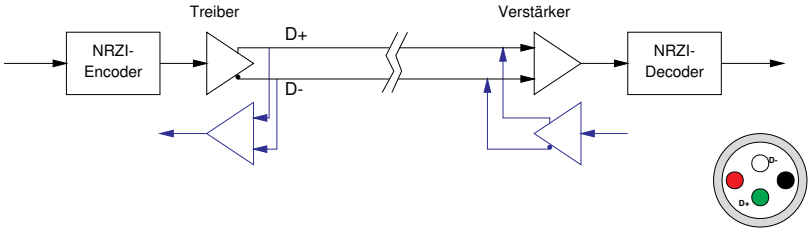
- einzelner logischer Datenkanal
- als Verbindung zwischen Endpoints an Root und "Function'"
- bis zu 16 Endpoints pro Function
- Adressierung: 7-bit Function-ID, 4-bit EP-ID, Richtung
- Endpoints werden im Device-Deskriptor definiert
- EP0 ist immer Control



USB: Stecker

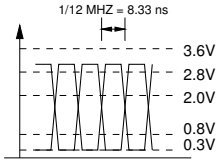
Series "A" Connectors	Series "B" Connectors
♦ Series "A" plugs are always oriented **upstream** towards the *Host System*	♦ Series "B" plugs are always oriented **downstream** towards the *USB Device*
 **"A" Plugs** (*From the USB Device*)	 **"B" Plugs** (*From the Host System*)
"A" Receptacles (*Downstream Output from the USB Host or Hub*) 	**"B" Receptacles** (*Upstream Input to the USB Device or Hub*) 

USB: Pegel / Kodierung

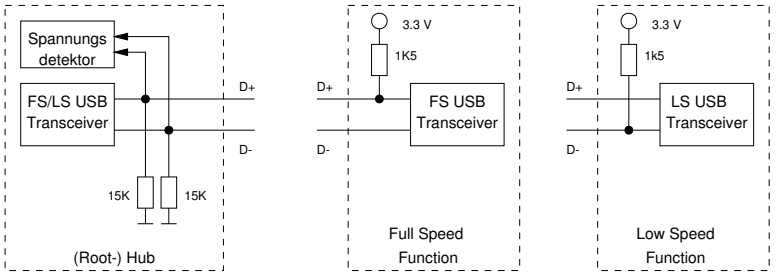


- Hin- und Rückrichtung über dieselben Leitungen
- Störsicherheit durch differentielle Signale
- NRZI-Kodierung und Bit-Stuffing

Differential "1": (D+) - (D-) > 200 mV
Differential "0": (D+) - (D-) < -200 mV



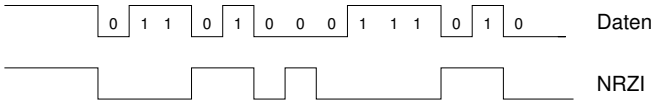
USB: Connect / Disconnect



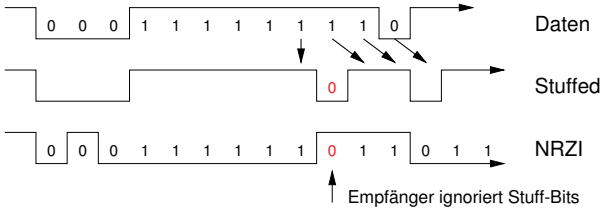
- kein Gerät am Hub: (D+) 0.0 V, (D-) 0.0 V (2x pulldown)
- FS-Gerät am Hub (D+) 3.3 V, (D-) 0.0 V
- FS-Gerät am Hub (D+) 0.0 V, (D-) 3.3 V
- Hub misst (D+)/(D-) Connect/Disconnect wird erkannt

USB: NRZI und Bitstuffing

- NRZI-Kodierung ("non return to zero inverted")
- 0-Datenbit: Signalwechsel, 1-Datenbit: kein Signalwechsel

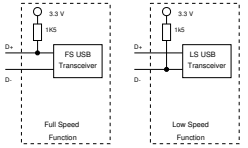


- Bit-Stuffing nach 6 Einsen zur Taktrückgewinnung:



USB: J/K-Zustände

Bus inaktiv: (D+)/(D-) nicht aktiv angesteuert:



- FS/LS-Erkennung über die Pullups
- aber unterschiedliche Spannungen

Bus aktiv: Hub oder Function treibt (D+)/(D-)

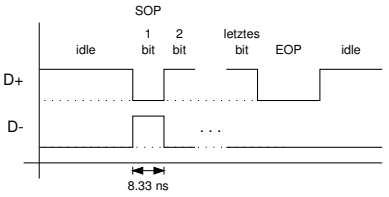
- differentielle Kodierung, kein Unterschied zwischen FS/LS

	Zustand	D+	D-	differentiell
low-speed	J (idle)	low	high	0
	K (resume)	high	low	1
full-speed	J (idle)	high	low	1
	K (resume)	low	high	0

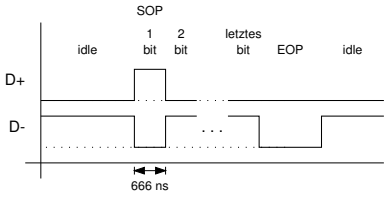
USB: Synchronisation

Synchronisation von Hub und Functions:

- inaktiver Bus wird nicht getrieben: J-Zustand
- SOP (start of packet) Wechsel in K-Zustand
- EOP (end of packet) (D+)/(D-) beide 0, < 2.5 µs
- RESET (D+)/(D-) beide 0, > 2.5 µs

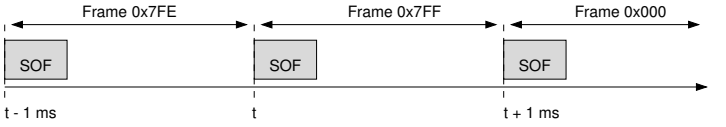


full speed function



low speed function

USB: leerer full-speed Bus



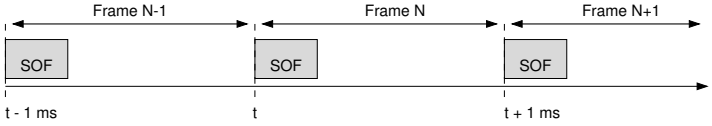
SYNC	SOF	Frame #	CRC5	EOP	idle
00000001	0xA5	0x7FD	0x1F	*0*	11965 Takte

SYNC	SOF	Frame #	CRC5	EOP	idle
00000001	0xA5	0x7FE	0x1D	*0*	11964 Takte

SYNC	SOF	Frame #	CRC5	EOP	idle
00000001	0xA5	0x7FF	0x02	*0*	11964 Takte

SYNC	SOF	Frame #	CRC5	EOP	idle
00000001	0xA5	0x700	0x08	*0*	11964 Takte

USB: Frames und Packets



- 1-ms Zeitabschnitte (=: Frames) (12.000 bit/frame FS)
- Frame beginnt mit SOF-Token inkl. 11-bit Frame-Nummer
- Suspend-Mode, wenn keine SOF-Token
- besondere Verwaltung von low-speed Geräten

- SOF-Token:

SYNC	SOF	Frame #	CRC5	EOP	idle
00000001	0xA5	0x7EE	0x1D	*0*	11964 Takte

NRZI: sieben Wertewechsel | Packet Identifier: SOF | (D+)/(D-) beide Null | Bus nicht getrieben

USB: Paket-Typen

PID-Name	PID (3..0)	PID (hex)	Gruppe
SOF	0101b	0xA5	Token
SETUP	1101b	0x2D	Token
IN	1001b	0x69	Token
OUT	0001b	0xE1	Token
DATA0	0011b	0xC3	Daten
DATA1	1011b	0x4B	Daten
ACK	0010b	0xD2	Handshake
NAK	1010b	0x5A	Handshake
STALL	1110b	0x1D	Handshake
PRE	1100b	0x39	Special

- 10 verschiedene USB Paket-Typen
- PID 7..4 = NOT (PID 3..0)

- Auswahl der Target-Function über Adresse und Endpoint (abhängig vom PID)

USB: Paket-Beispiele

SYNC	SOF	Frame #	CRC5	EOP
00000001	0xA5	0x7FD	0x1F	*0*

Start-of-Frame

SYNC	SETUP	ADDR	EP	CRC5	EOP
00000001	0x2D	0x01	0x00	0x17	*0*

Setup-Token

SYNC	IN	ADDR	EP	CRC5	EOP
00000001	0x69	0x03	0x01	0x07	*0*

In-Token

SYNC	OUT	ADDR	EP	CRC5	EOP
00000001	0xE1	0x02	0x02	0x01	*0*

Out-Token

SYNC	DATA0	data	CRC16	EOP
00000001	0xC3	00 11 22 33 44 55 66 77	0xCBA8	*0*

USB: Datentransfer

	idle	SYNC	SOF	Frame #	CRC5	EOP
PC	11966	00000001	0xA5	0x002	0x15	**

	idle	SYNC	IN	ADDR	EP	CRC5	EOP
PC	4	00000001	0x69	0x02	0x02	0x01	**

	idle	SYNC	DATA0	data	CRC16	EOP
Maus	5	00000001	0xC3	00 11 22 33 44 55 66 77	0xCBA8	**

	idle	SYNC	ACK	EOP
PC	3	00000001	0xD2	**

	idle	SYNC	SOF	Frame #	CRC5	EOP
PC	11804	00000001	0xA5	0x003	0x0A	**

	idle	SYNC	IN	ADDR	EP	CRC5	EOP
PC	4	00000001	0x69	0x02	0x02	0x01	**

	idle	SYNC	DATA1	data	CRC16	EOP
Maus	5	00000001	0x4B	88 99 AA BB CC DD EE FF	0x8705	**

	idle	SYNC	ACK	EOP
PC	3	00000001	0xD2	**

USB: 3-Phasen Datentransfer

Token	Sender Token-Phase	Sender Daten-Phase	Sender Handshake-Phase
SETUP	Host	Host	Function
OUT	Host	Host	Function
IN	Host	Function	Host

Beispiel: Host sendet Daten, Function quittiert (ACK oder NAK)

	idle	SYNC	OUT	ADDR	EP	CRC5	EOP
PC		00000001	0xE1	0x02	0x02	0x01	**

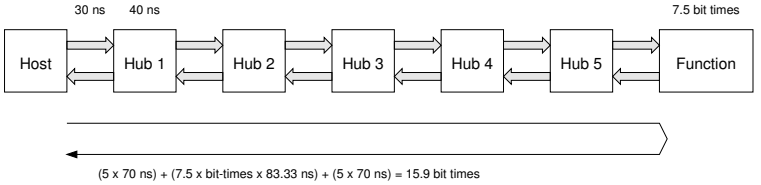
	idle	SYNC	DATA0	data	CRC16	EOP
PC		00000001	0xC3	00 11 22 33 44 55 66 77	0xCBA8	**

	idle	SYNC	ACK	EOP
Maus		00000001	0xD2	**

USB: Timeout

Datenübertragung zwischen Sender und Empfänger

- Mindestzeit bis zur Antwort / Handshake des Empfängers ?
- Bus-Turn-Around-Zeit:



- Empfänger hat 7 Takte Zeit für Antwort / Handshake: max. 16 Takte
- Timeout nach 18 Takten

USB: Deskriptoren

Plug and Play:

- Host muß alle USB-Functions identifizieren können
- wie findet der Host die benötigten Treiber?

=> Deskriptoren: Code

• Device Descriptor	0x01	
• Configuration Descriptor	0x02	(jeweils eine aktiv)
• String Descriptor	0x03	(optional)
• Interface Descriptor	0x04	
• Endpoint Descriptor	0x05	(FIFO-Größe etc.)

- Protokoll definiert Transfers zum Auslesen der Deskriptoren

USB: Device Descriptor

Name	Offset	Länge	Beschreibung	Beispielwert
bLength	0	1	Größe in Bytes	0x12
bDescriptorType	1	1	= Device-Descriptor	0x01
bcdUSB	2	2	USB-Version (1.0,1,1.2.0)	0x0001
bDeviceClass	4	1	Klassen-Code	0x00
bDeviceSubClass	5	1	Subklassen-Code	0x00
bDeviceProtocol	6	1	Protokoll-Code	0x00
bMaxPacketSize0	7	1	EP0-FIFO in Byte	0x08
idVendor	8	2	Vendor-ID	0x3C05
idProduct	10	2	Produkt-ID (etwa 0x9084)	0x9084
bcdDevice	12	2	Versions-Nr. (1.02)	0x0102
iManufacturer	14	1	String-Index für	0x01
iProduct	15	1	Hersteller / Produkt	0x02
iSerialNumber	16	1	Seriennummer	0x03
bNumConfigurations	17	1	# Konfigurationen	0x01

- Vendor-ID wird vom USB Implementers Forum vergeben

USB: Configuration Descriptor

Name	Offset	Länge	Beschreibung	Beispielwert
bLength	0	1	Größe in Bytes	0x09
bDescriptorType	1	1	= Device-Descriptor	0x02
wTotalLength	2	2	Länge aller Desr. (zB 34)	0x0022
bNumInterfaces	4	1	Interface pro Konfiguration	0x01
bConfigurationValue	5	1	Nummer der Konfiguration	0x01
iConfiguration	6	1	String-Index	0x04
bmAttributes	7	1	z.B. Bus-powered, wakeup	0xA0
maxPower	8	1	in 2 mA Schritten	0x32

- Host liest Configuration-D. mitsamt allen Interface/Endpoint-D.
- SetConfiguration(0) deaktiviert das Gerät ("unkonfiguriert")

- weitere Datendefinitionen für Interfaces / Endpoints / Strings

USB: Device-States

• -	nicht mit dem Bus verbunden
• attached	angeschlossen, Hub benachrichtigt Host
• powered	Hub aktiviert Versorgung, noch kein Reset
• default	nach Reset, Device reagiert an Adresse 0
• address	nach SetAdress()
• configured	nach SetConfiguration() etc. das Gerät kann jetzt benutzt werden
• suspended	Stromsparen, wenn 3 ms keine Busaktivität

USB: Request-Codes

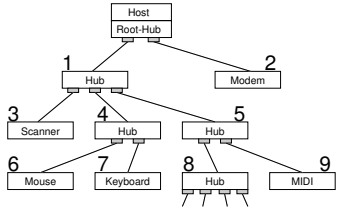
bRequest	Code
GET_STATUS	0
CLEAR_FEATURE	1
reserved	2
SET_FEATURE	3
reserved	4
SET_ADDRESS	5
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7
GET_CONFIGURATION	8
SET_CONFIGURATION	9
GET_INTERFACE	10
SET_INTERFACE	11
SYNC_FRAME	12

- Control-Transfers
- auf EP0 Control-Pipe

USB: Enumeration-Init

Erkennung aller Geräte beim Einschalten:

- alle Hub-Downstream-Ports zunächst deaktiviert
- Host sieht nur je ein Gerät pro Root-Hub Port
- Host kann oberstes Gerät konfigurieren
- Downstream-Ports aller Hubs werden nacheinander konfiguriert
- wahlweise Tiefen/Breitensuche



USB: Enumeration

Plug and Play erfordert automatische Erkennung aller Geräte:

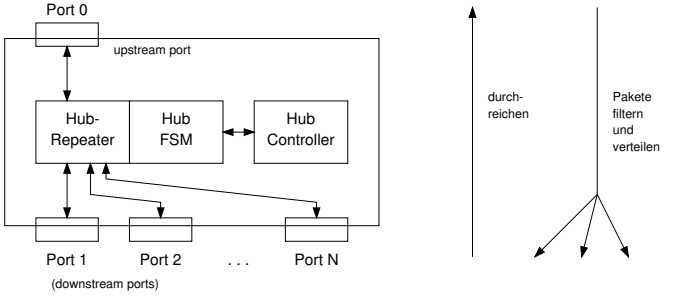
0 Hub erkennt Connect/Disconnect elektrisch
1 meldet dies beim nächsten Interrupt-Transfer

2 Host sendet Standard-Device-Request auf Adresse 0
3 Function antwortet mit seinem Device-Descriptor
4 Host sendet Reset an das Gerät
5 Host sendet SetAddress an das Gerät (noch auf Adresse 0)

6 Host fragt vollen Device-Descriptor ab
7 Host sucht geeigneten Treiber für Vendor-ID und Product-ID
8 Host fragt nach Configuration- und Device-Descriptor

9 Host konfiguriert das Gerät [siehe Kelm: USB S94ff]

USB: Hub



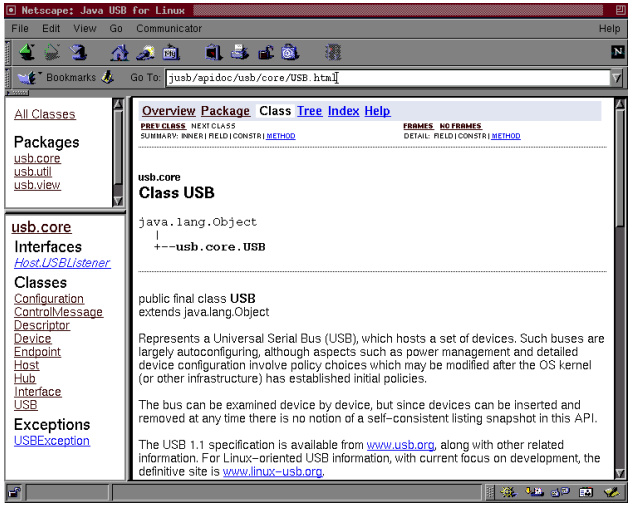
- Hub-Controller ist eigenständige USB-Function
- Diagramm Downstream-Port-FSM
- Funktionalität zur Ansteuerung von low-speed functions

USB: Device-Klassen

USB-Klasse :=

- Gruppe von Geräten/Interfaces mit gemeinsamen Eigenschaften
- z.B. gleiche Datenformate, Kommunikationsverhalten, ...
  - Human-Interface-Device
  - Audio-Device
  - Communication-Device
  - Printer-Device
  - Monitor-Device
  - Power-Device
  - Mass-Storage-Device
  - . . .
- Betriebssystem benötigt nur einen Treiber pro Klasse
- Treibersuche über Device/Config-Descriptor-Infos

USB: Java USB: API

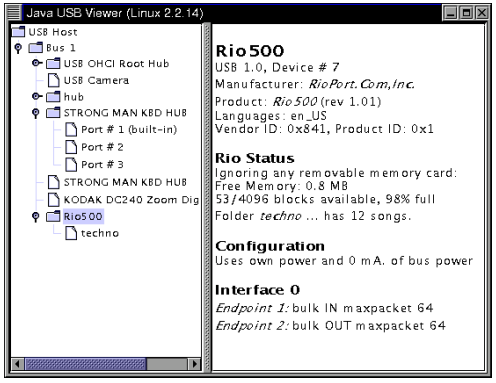


USB: Human Interface Device Class

Human-Interface-Device Class:

- Tastatur, Maus
- Schalter und Regler am PC
- Spielkomponenten wie Joystick, Datenhandschuh, Pedale, ...
- Geräte ohne menschliche Interaktion,
  aber mit ähnlichen Datenformaten:
  Barcode-Leser, Thermometer, Voltmeter, ...
- Definition eines universellen Protokollformats
- Report- und Physical-Descriptor
  www.usb.org/developers/data/hidpage.htm
- besondere Behandlung von Tastatur und Maus (für Systemboot)

USB: Java USB Browser



- USB-Wrapper in Java, für Linux 2.4
- alpha-Status, fast keine Gerätetreiber

jusb.sourceforge.net

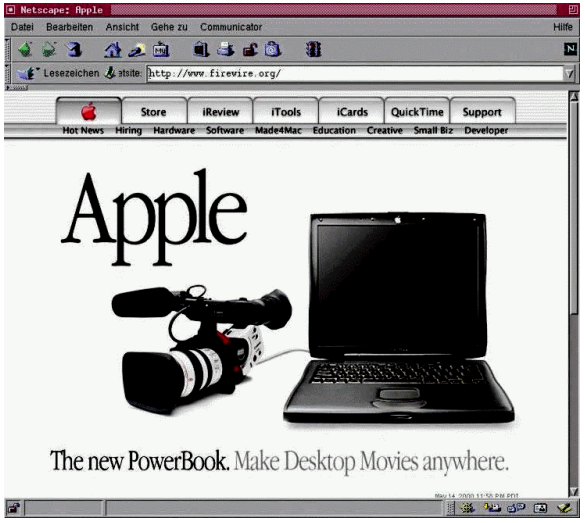
FireWire:

- entwickelt von Apple seit 1986
- IEEE Standard 1394-1995
- komplexer und schneller als USB: 100/200/400 Mb/s
- volle Autokonfiguration, kein zentraler Host
  insbesondere: kein PC notwendig

- derzeit ca. 40 Geräte (www.apple.com)
- etabliert für digitales Video (8 Kanäle MPEG2: <120 Mb/s)
- diverse Erweiterungen / Zusatzprotokolle, zum Beispiel:
- Audio and Music Transmission Protocol
- IP over 1394

- Lizenzprobleme (derzeit 0.25$ / Gerät)
- Microsoft: "IP-Landmines"

FireWire: Homepage



FireWire: Architektur

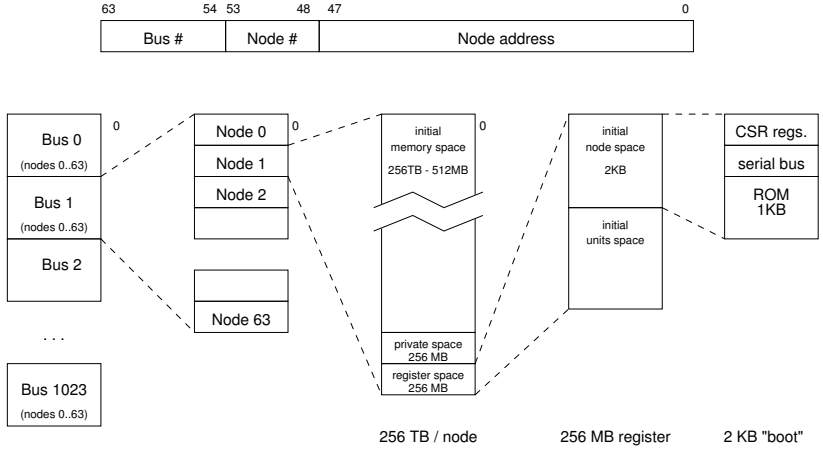
FireWire-System:

- bis 1023 separate Busse
- bis 63 Geräte pro Bus (bis 64.449 Geräte)
- 64-bit Adressraum (10 Bus, 6 Gerät, 48 bit pro Gerät)
- volle Autokonfiguration, keine Jumper/Terminatoren/...
- asynchrone und isochrone Nachrichten (@8KHz)

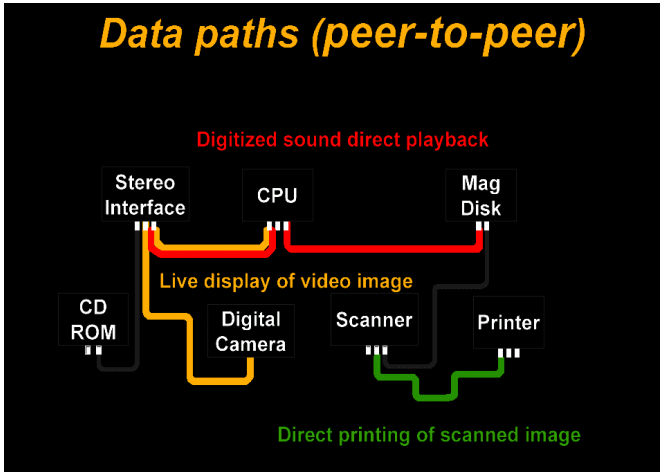
- bis 16 Kabelverbindungen (Hops) pro Bus
- Kabellängen bis 500m (gesamt)
- Kabel 6-adrig, 2x Daten differentiell, Versorgungsspannung
- 400 Mb/s bis 10m Kabellänge

- Peer-to-Peer Verbindungen (statt Master-Slave)
- Beispiel: direktes Drucken von Digitalkamera auf 1394-Drucker

FireWire: Adressraum

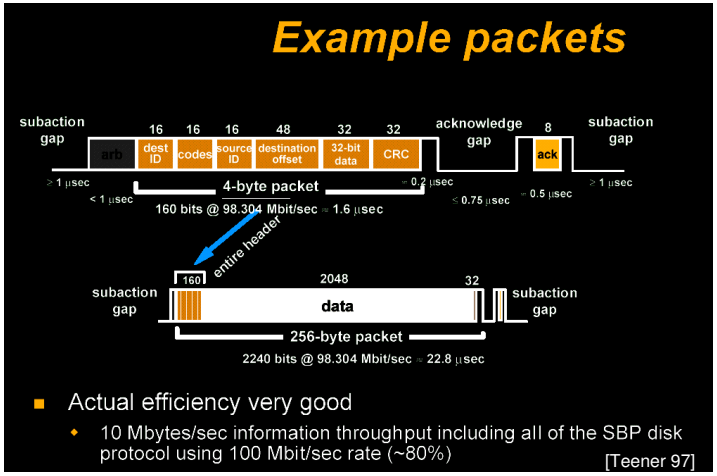


1394: Peer-to-peer transfers



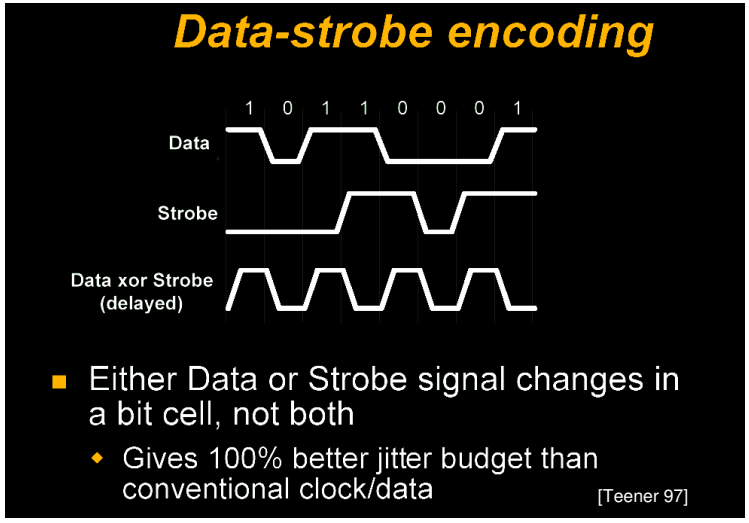
- andere Busse: fast immer zentrale Steuerung via Master

1394: Beispiel Datenpaket



- aber geringe Effizienz für sehr kurze Pakete

1394: Daten- und Taktsignal

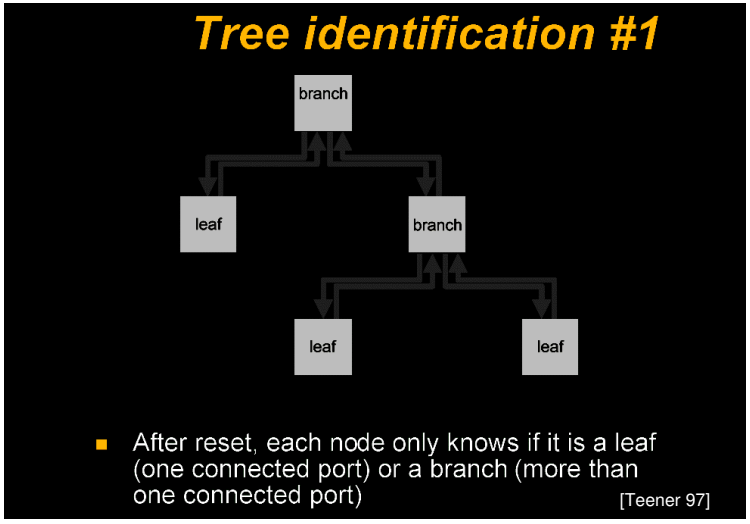


FireWire: Arbitrierung

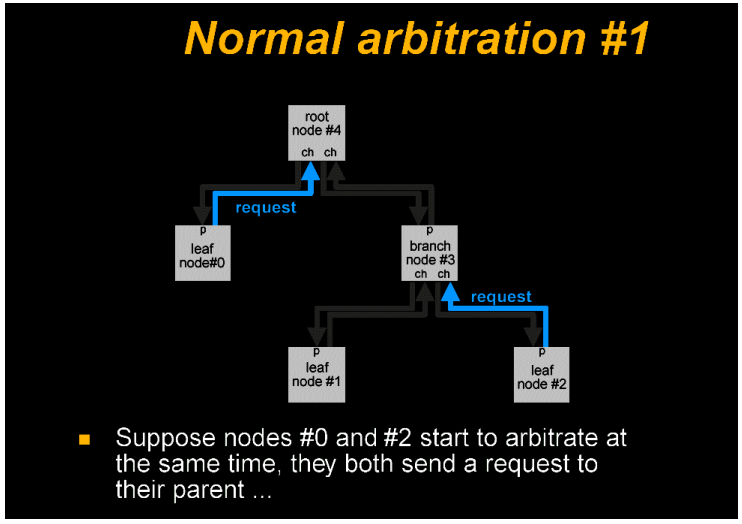
- kein zentraler Bus-Master
- direkte Kommunikation aller Geräte
- verteilte Arbitrierung, Fairness-Garantien

- nach jedem Anstecken/Abnehmen eines Geräts:
  - Bus-Reset (80 .. 300 ms)
  - Tree-Identification (10 .. 166 µs)
  - Self-Identification (70 µs)
  - ermittelt neuen Root-Node
  - sofern zyklenfrei
- Bus automatisch wieder betriebsbereit

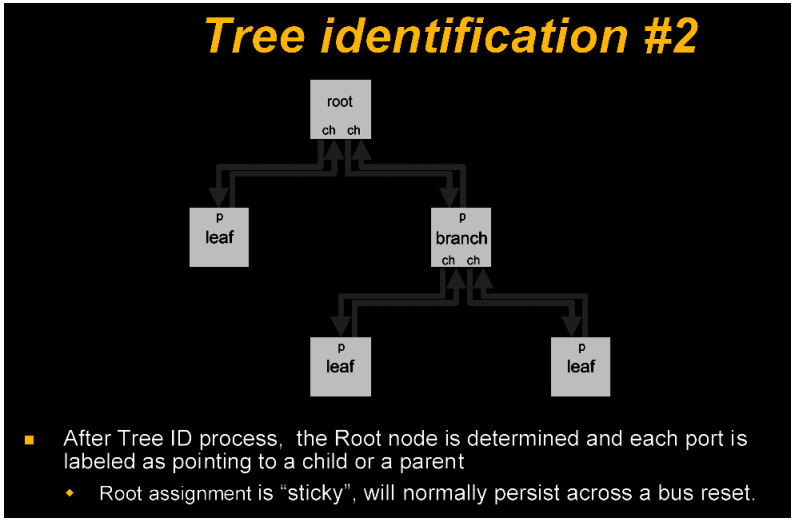
1394: Tree Identification (1)



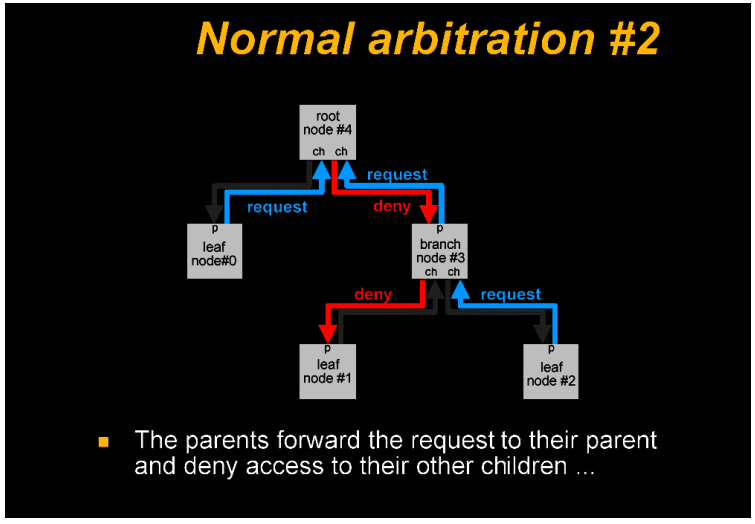
1394: Arbitration (1)



1394: Tree Identification (2)

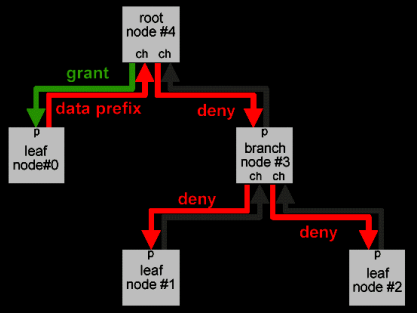


1394: Arbitration (2)



1394: Arbitration (4)

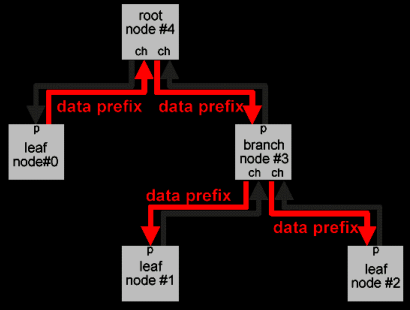
Normal arbitration #4



- The winning node #0 changes its request to a data transfer prefix, while the loosing node #2 withdraws its request …

1394: Arbitration (5)

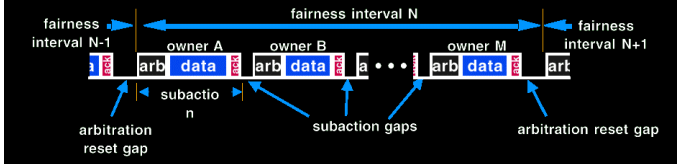
Normal arbitration #5



- The parent of node 1 sees the data prefix and withdraws the grant, and now all nodes are correctly oriented to repeat the packet data (a "deny" is a "data prefix!) …

1394: Fairness

Fairness interval



- Fairness Interval is bounded by “arbitration reset gaps”
- Reset gaps are longer than normal subaction gaps

- andere Busse: fast immer zentrale Steuerung via Master

Leerseite