

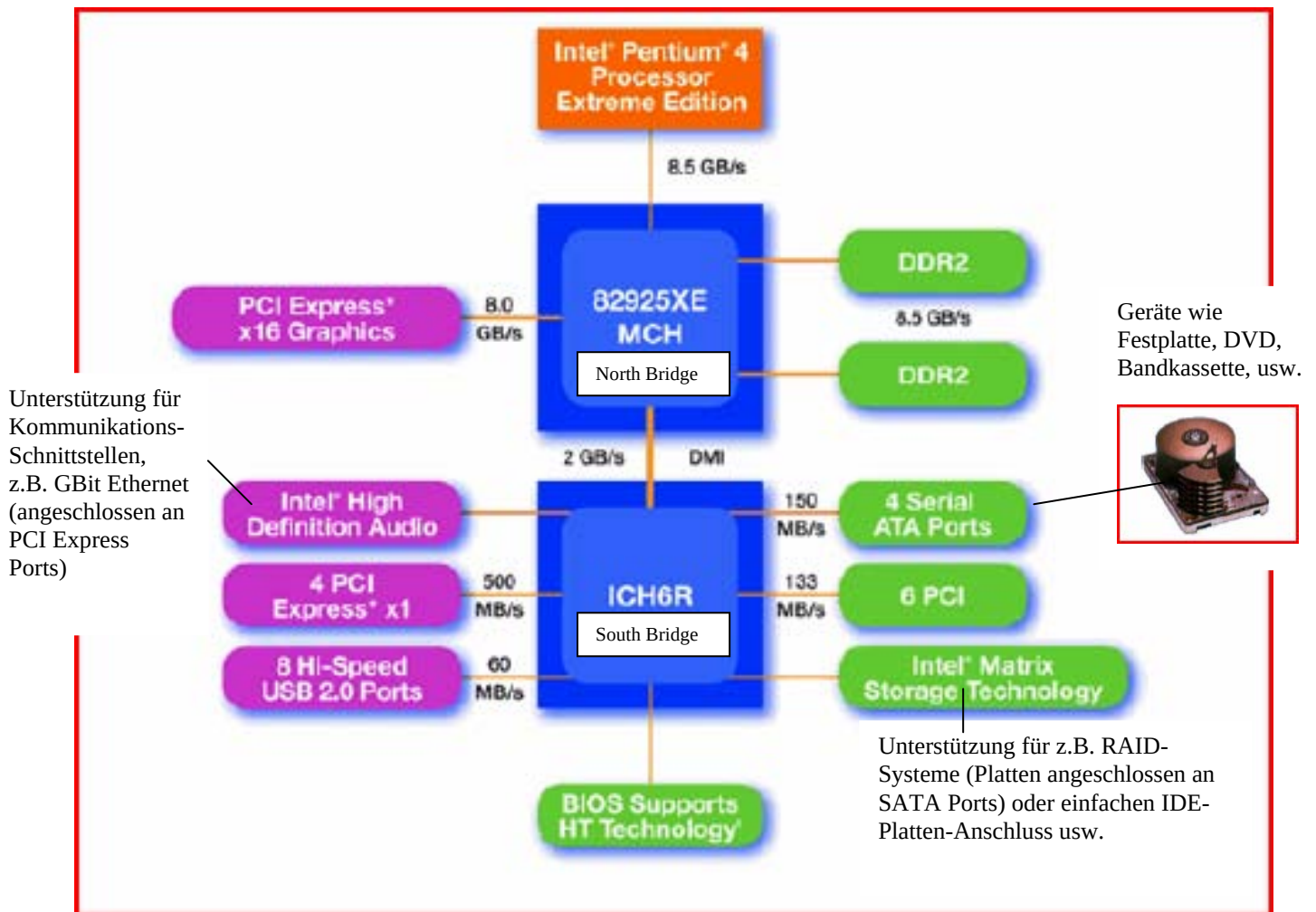
RECHNER - PERIPHERIE

Teil 1: Datenpuffer und Schnittstellen

Inhaltsverzeichnis

RECHNER - PERIPHERIE	1
1. Beispiele externer Diplomarbeiten	5
1.1. Diplomarbeit „CAN-USB-Interface“	5
1.2. Diploma Thesis “Real-Time Image Processing in Cars”	6
1.3. Diplomarbeit „Implementierung einer PCI-Schnittstelle in ein SOC (System on a Chip)“	8
1.4. Diplomarbeit „Entwicklung eines Aufzeichnungssystems für digitale Audio-Daten“	9
1.5. Diplomarbeit „Entwicklung einer elektronischen Drosselklappen-Ansteuerung für den Motorsport“	11
2. Datenpuffer	12
2.1. Datenpuffer-Prinzipien (funktionelle Ebene)	12
2.1.1. Einfach-Puffer	12
2.1.2. Tandem-Puffer (Wechsel-Puffer)	13
2.1.3. Ring-Puffer	14
2.1.4. FIFO-Puffer	14
2.1.5. Dual Port RAM	17
2.2. Realisierung von Datenpuffern (Beispiele)	20
2.3. Aufgaben	21
3. PCI-Bus (Peripheral Component Interconnect)	23
3.1. Einführung	23
3.2. Bus-Varianten	26
3.3. PCI Conventional	27
3.3.1. Adressierung	27
3.3.2. Konfiguration	28
3.3.3. Software-Schnittstelle	35
3.3.4. Bus-Kommunikation	36
3.3.5. Hardware	42
3.4. PCI Express	45
3.4.1. Physikalische Ebene	46
3.4.2. Adressierung	47
3.4.3. Konfiguration	47
3.4.4. Architektur und Buskommunikation	47
3.5. Aufgaben	51

4.	SCSI-Bus (Small Computer Systems Interface).....	52
4.1.	Einführung	52
4.2.	SCSI-3 Architektur.....	54
4.2.1.	Befehlsebenen.....	59
4.2.2.	Physikalische Ebene.....	63
4.3.	SCSI Parallel als Beispiel für ein dezentrales Buskonzept.....	64
4.3.1.	Begriffe.....	64
4.3.2.	Befehlsformat	68
4.3.3.	Bus-Kommunikation	69
4.3.4.	Bussignale	75
4.4.	Serial Attached SCSI (SAS).....	79
4.4.1.	Physikalische Ebene (Interconnects layer)	81
4.4.2.	Adressierung	82
4.4.3.	Konfiguration.....	82
4.4.4.	Architektur und Buskommunikation.....	83
4.5.	Aufgaben.....	86
5.	USB (Universal Serial Bus)	87
5.1.	Einführung	87
5.2.	Host Controller und angeschlossene Busteilnehmer	89
5.3.	Bus-Kommunikation	91
5.4.	Software-Schichten	92
5.5.	Informations-Transfer	95
5.6.	Paket-Formate	99
5.7.	Phasen einer USB-Transaktion (Transaktions-Formate).....	102
5.8.	Low- und Full-Speed-Geräte am High-Speed-Bus bzw. Hub	104
5.9.	Übertragungsreihenfolge innerhalb eines Zeitrahmens.....	106
5.10.	Deskriptoren.....	108
5.11.	USB-Befehle (USB Device Requests).....	110
5.12.	Elektrische Eigenschaften und Datencodierung	112
5.13.	Initialisierung (Bus Enumeration)	116
5.14.	On-The-Go USB (OTG-USB)	117
5.15.	Wireless USB.....	121
5.16.	Aufgaben.....	122
6.	Serielle Schnittstelle RS-232.....	126
7.	Parallele Schnittstelle.....	127



MCH Memory Controller Hub (Memory Bridge)
 DMI Direct Media Interface
 ICH6R I/O Controller Hub (I/O Bridge, I=Input, O=Output)
 ("... 8 multipurpose DMA engines, 4 input and 4 output, support of simultaneous independent data streams")

Bild: PC-Architektur (Fa. Intel)

Über die Schnittstellen **PCI**, **SATA** und **USB** sind alle gängigen Peripherie-Geräte (Funktionen) anschließbar. Weitere Schnittstellen wie **PCI-X**, **Firewire**, **SAS**, **SCSI**, **RS232**, **Enhanced Parallel Port EPP** usw. sind nicht eingezeichnet.

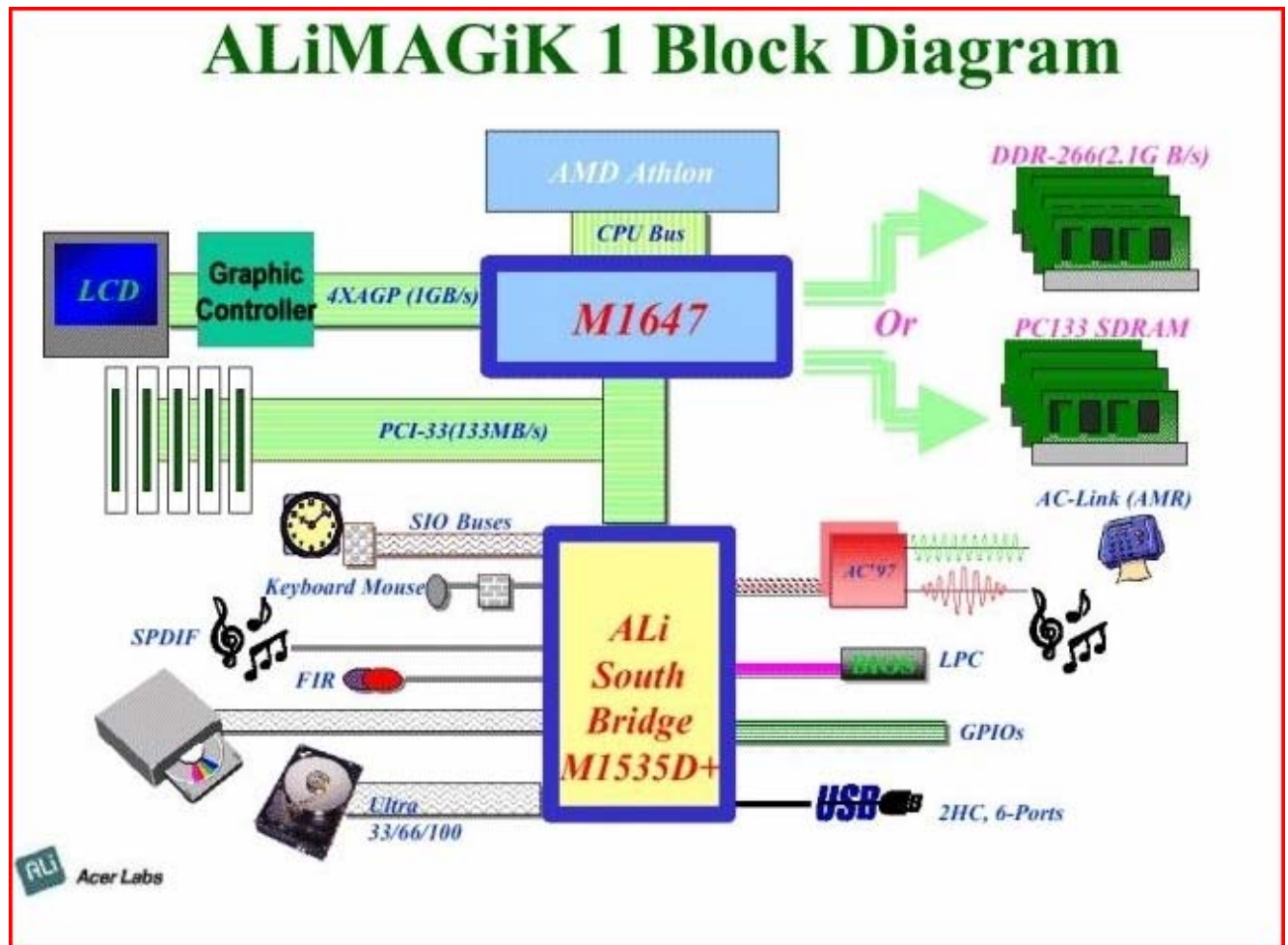
„Schnittstellenauswahl“:

Muss aus Zeit- und Platzgründen für die Vorlesung eine Beschränkung auf nur wenige Schnittstellen erfolgen, so könnte folgende Argumentation für drahtgebundene Schnittstellen stichhaltig sein:

- Der Anschluss vieler Geräte und sonstiger digitaler Hardware an einen Rechner erfolgt derzeit meist über die Standardschnittstelle **USB**, sofern die Anforderungen an Datenrate, Realzeitverhalten usw. nicht extrem sind. Die Kabellängen sind aber begrenzt (einige Meter).
- Sind Geräte und sonstige anzuschließende digitale Hardware räumlich entfernt vom Rechner, dann bietet sich die in Büro- und Industrieräumen häufig vorhandene **Ethernet**-Schnittstelle an.
- Muss aus Gründen von Datenrate, Realzeitverhalten usw. oder wegen der Integration in ein Gehäuse eine Funktion direkt am inneren Rechnerbus angeschlossen werden, dann ist dies derzeit der **PCI**.

USB und **PCI** werden im Detail in dieser Vorlesung abgehandelt, die Ethernet-Schnittstelle in „Computernetze“.

Diskutieren Sie die Datenraten von Geräten und Schnittstellen.
Wegen der „anschaulichen“ Darstellungen:



1. Beispiele externer Diplomarbeiten

Die nachfolgenden Diplomarbeiten der letzten Jahre fanden in verschiedenen Firmen statt. Wichtige Schwerpunkte dieser Arbeiten wie die Realisierung von USB- und PCI-Schnittstellen und von Datenpuffern vom Typ FIFO und Dual-Port-RAM usw. sowie die Beschreibung und Simulation der Hardware-Funktionen in VHDL und deren nachfolgende Synthese in FPGA-Bausteinen sind Hauptkapitel in den Vorlesungen/Praktika „Rechnerperipherie“ und „Rechnergestütztes Schaltungs-Design“.

1.1. Diplomarbeit „CAN-USB-Interface“

(Helmut Nagy bei Fa. TEMIC)

Text aus der Diplomarbeit:

..... Der im Automotive-Bereich stark verbreitete Zweidraht-CAN-Bus sollte mittels eines „Gateways“ dem PC zugänglich gemacht werden (CAN-USB-Interface). Das Gateway besteht aus einem HC12-Mikrocontroller von Motorola und einem Anchor USB-Chip AN2131. Aus Sicht des am USB angeschlossenen PCs stellt das Interface ein Gerät mit 2 Endpunkten dar, zu denen Datenstrecken („pipes“) installiert sind. Die Eigenschaften der Datenstrecken sind in sogenannten Deskriptoren festgelegt. Eine detaillierte Kenntnis der USB-Protokolle usw. war bei der Diplomarbeit notwendig.

Die Abteilung, in der diese Arbeit erstellt wurde, stellt Steuergeräte für die elektronische Sitzverstellung und für Sitzheizungen her. Damit eine Sitzbaugruppe richtig funktionieren kann, benötigt sie Informationen von anderen Steuergeräten im Fahrzeug. Für die Ausstiegshilfe muss z.B. der Zustand der Tür (offen/geschlossen) bekannt sein. Das CAN-Interface stellt alle zum Betrieb der Sitzbaugruppe benötigten Informationen zur Verfügung.

Das CAN-USB-Interface ist auf einer beidseitig bestückten Platine im Euro-Format (160x100mm) aufgebaut. Es wurden bis auf den Quarz und einige Steckverbinder ausschließlich SMD-Bauteile verwendet, damit alles auf der Platine Platz hat. Für den CAN-Physical-Layer ist eine Aufsteckplatine vorgesehen. Die Bauteile erfüllen die Anforderungen für den Automotive-Bereich. Das Interface kann dank der VG-96 Steckerleiste in ein 19“-Rack eingebaut werden.

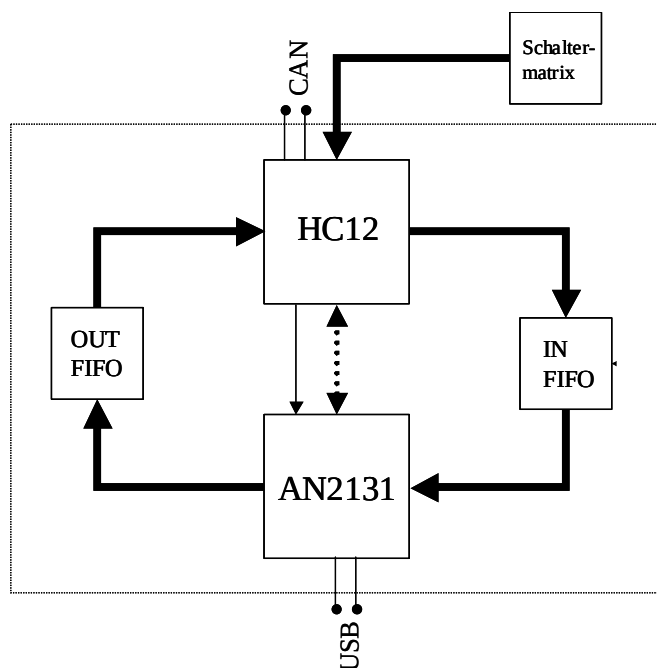


Bild: Hardware-Blockdiagramm

AN2131: USB Mikrocontroller, HC12: CAN 2.0 Protokollcontroller

Der CAN-Bus wurde 1987 in seinen wesentlichen Eigenschaften von der Firma Robert Bosch GmbH entwickelt, um die voranschreitende Vernetzung von Steuergeräten in Kraftfahrzeugen zu erleichtern. Mittlerweile hat sich der CAN-Bus zu einem internationalen Standard etabliert und wird heute in vielen neu entwickelten Kraftfahrzeugen eingesetzt.

1.2. Diploma Thesis “Real-Time Image Processing in Cars”

(Martin Schwarz bei Fa. FREESCALE Semiconductor)

Text aus der Diplomarbeit:

..... Image processing in cars is used for various driver assistance systems. This starts with distance control, overtaking (Überholen) assistants and lane departure. A camera also can be used as a backward driving camera or to detect persons in the car's interior.

The purpose of the present thesis is to develop a PCI card that allows data acquisition from automotive graded CMOS image sensors and high speed serial links.

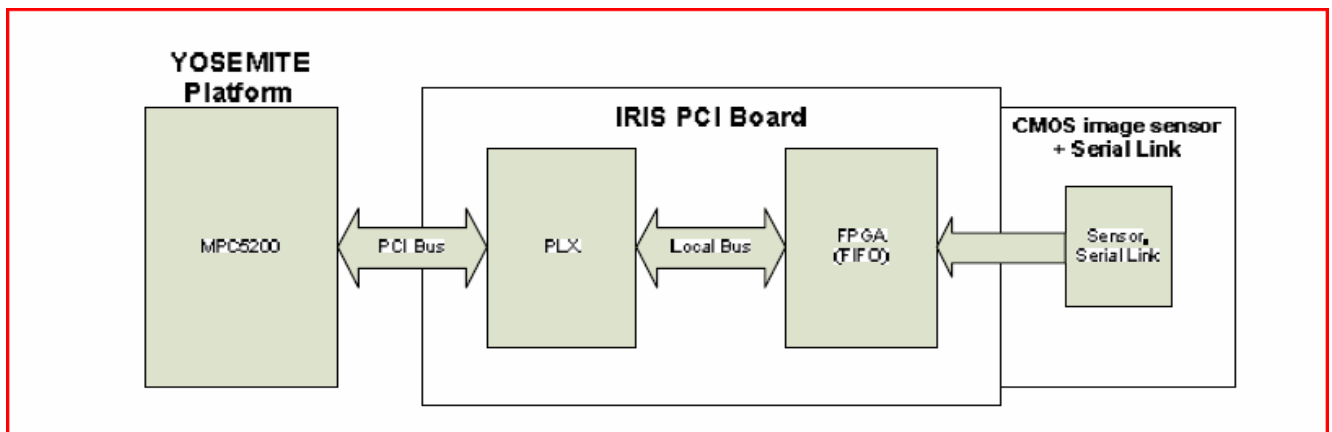


Bild: Hardware Blockdiagramm

Die Hardware wurde in einem ersten Testaufbau mit 4 Platinen realisiert (im Bild von links nach rechts):

MPC5200-uP-Platine mit PCI-Bus, PCI-Bus-Platine (PLX9030 PCI Controller Evaluation Board, senkrecht), FPGA-Platine (Fa. XILINX), Bildsensor-Platine (Kamera).

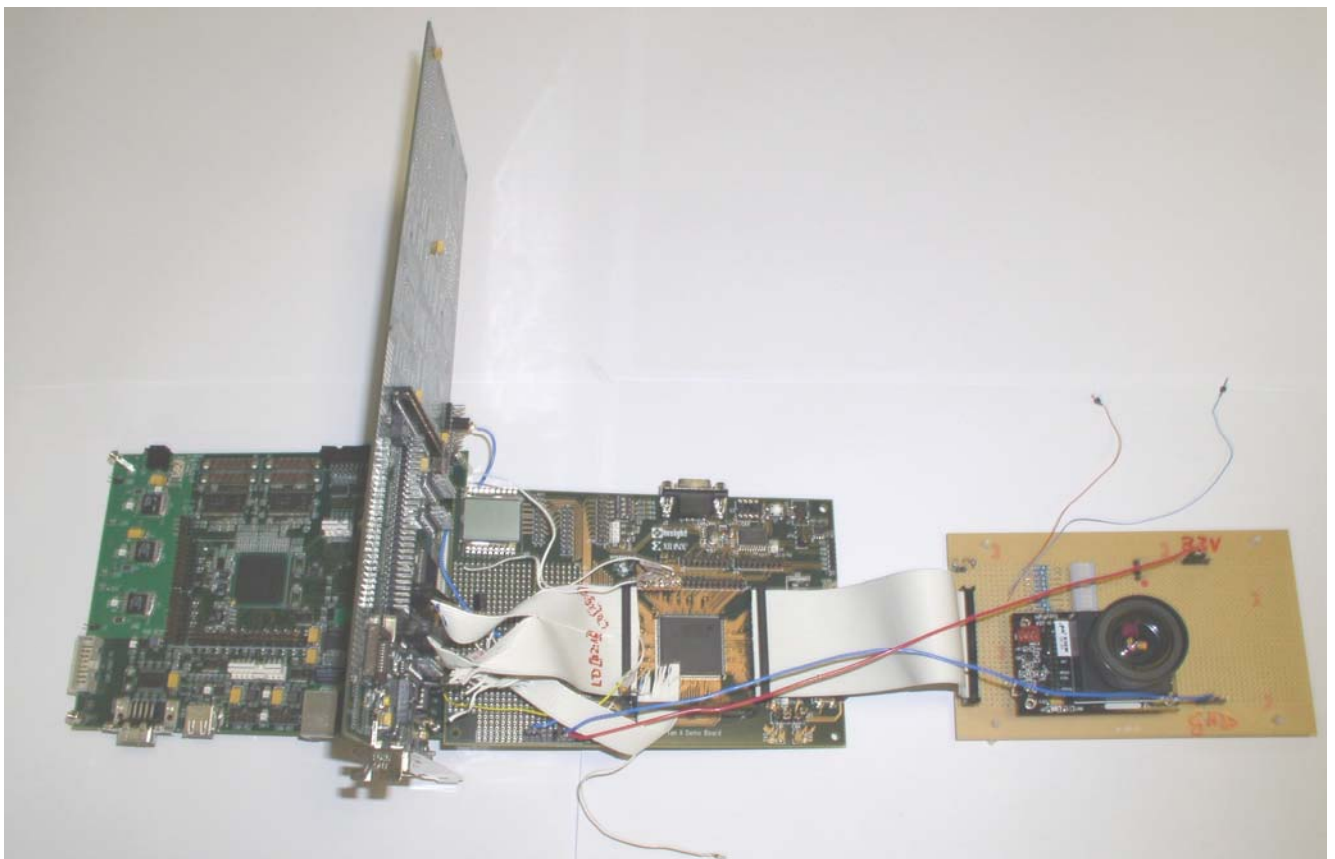


Bild: Erster Testaufbau

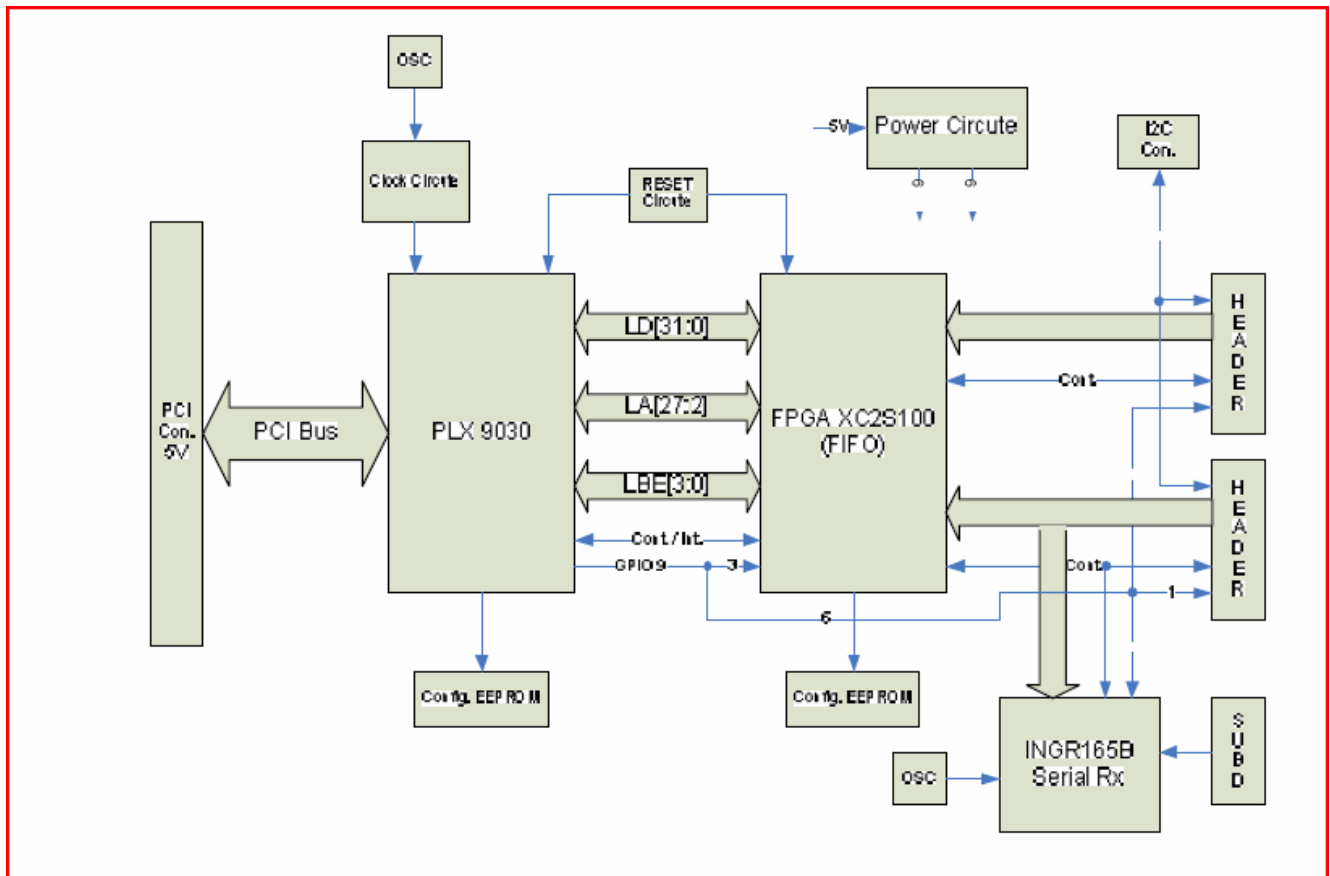


Bild: Blockdiagramm des IRIS PCI Boards

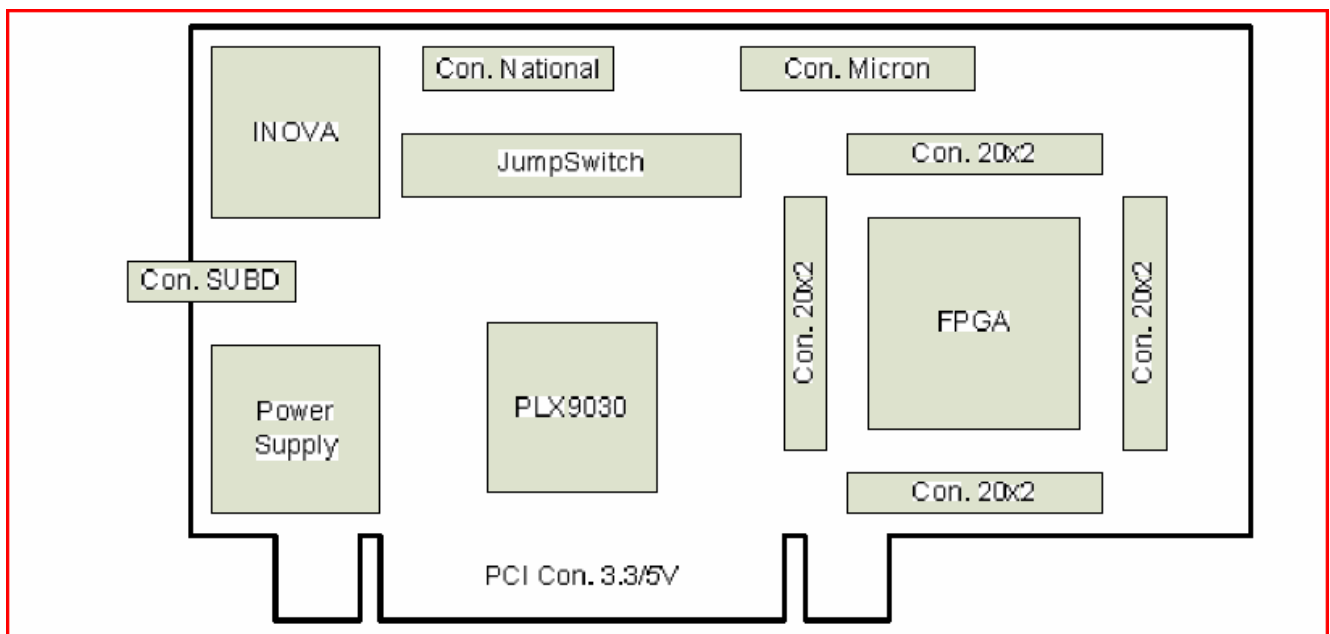


Bild: Obiges Blockdiagramm, realisiert als „IRIS PCI Board mit FPGA (FIFO)“:

1.3. Diplomarbeit „Implementierung einer PCI-Schnittstelle in ein SOC (System on a Chip)“

(Ulrich Benkart bei Fa. OSRAM)

Text aus der Diplomarbeit:

PCI-Schnittstelle:

..... Es wurde ein Vergleich von in Frage kommenden PCI-Controllern und PCI IP-Cores durchgeführt. Die IP-Core-Lösung wurde einem separaten PCI-Controller vorgezogen.

Die gesamte Logik – Dual Port RAM (IP-Core), Anpassungslogik und PCI-Schnittstelle (IP-Core) – wurde in einem FPGA der Fa. ALTERA realisiert.

(IP = intellectual property, IP-Core = vorgefertigter Funktionsblock, in VHDL oder Verilog erstellt.)

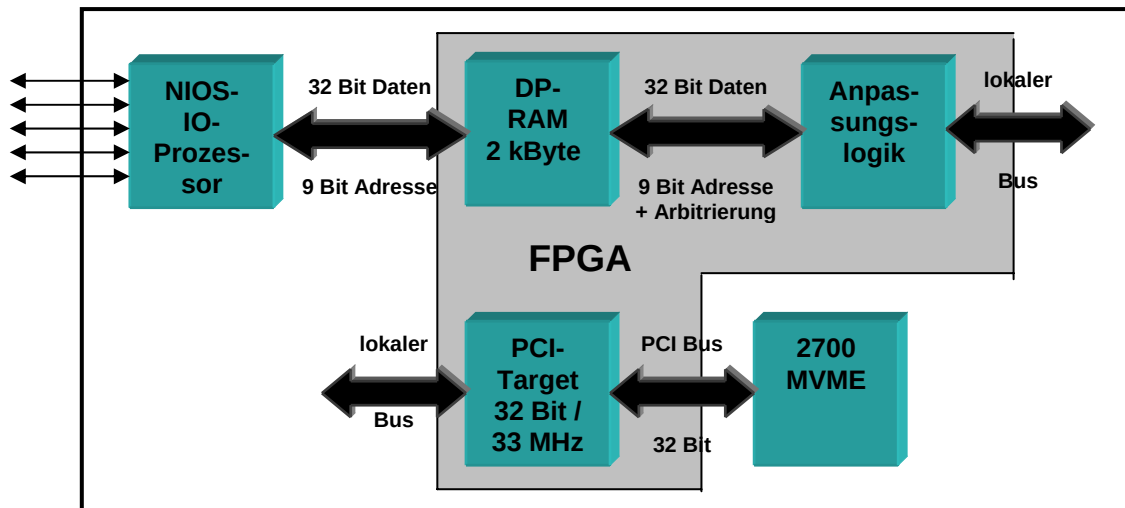


Bild: Anbindung des NIOS IO-Prozessors an die CPU 2700 MVME

DP RAM = Dual Port RAM (IP-Core)

PCI Target = PCI-Controller-Funktion (PCI-Core PLX 9030)

Funktionsteile	Kosten PLX 9030	Kosten IP-Core
Controller	21 €	14 € (bei 100 Stück)
FPGA Kostenanteil	keiner	10 € für 1 000 LEs
Layoutkosten Controller	20 €	keine weiteren
Layoutkosten Pegelanpassung	keine	50 €
Entwicklungssoftware	beide gleich	beide gleich
Gesamtkosten	41 €	74 €

Tabelle: Kostenvergleich PCI-Controller PLX 9030 Lösung vs. PCI IP-Core Lösung

IP-Core: 1400 € ergeben 14 € bei Stückzahl = 100

teurer aber flexibler !

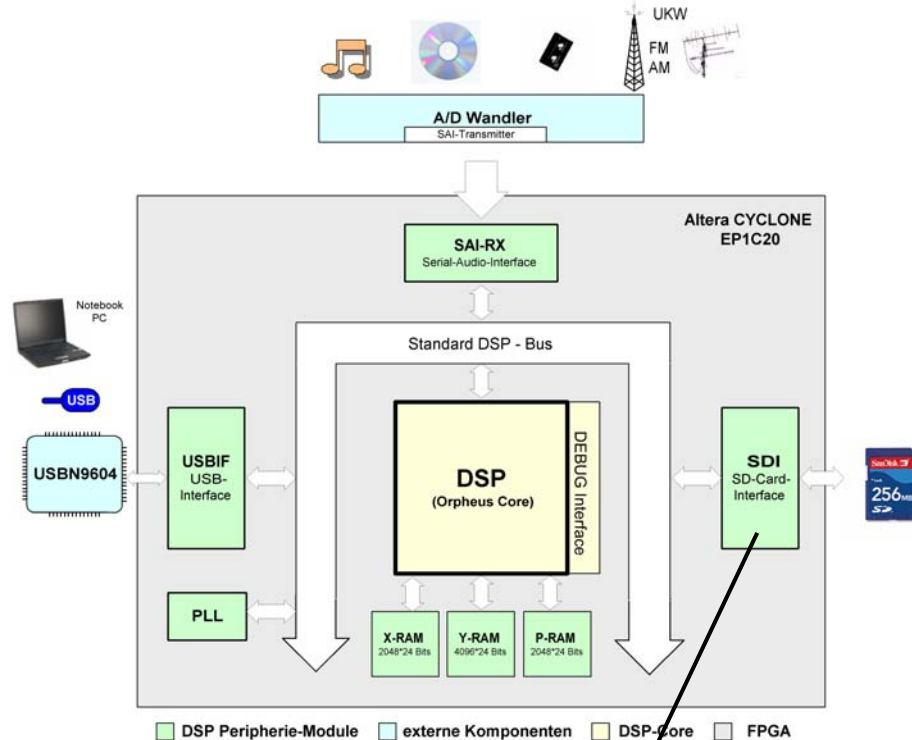
1.4. Diplomarbeit „Entwicklung eines Aufzeichnungssystems für digitale Audio-Daten“

(Stefan Ramböck bei Fa. STMicroelectronics, European Automotive Design Center)

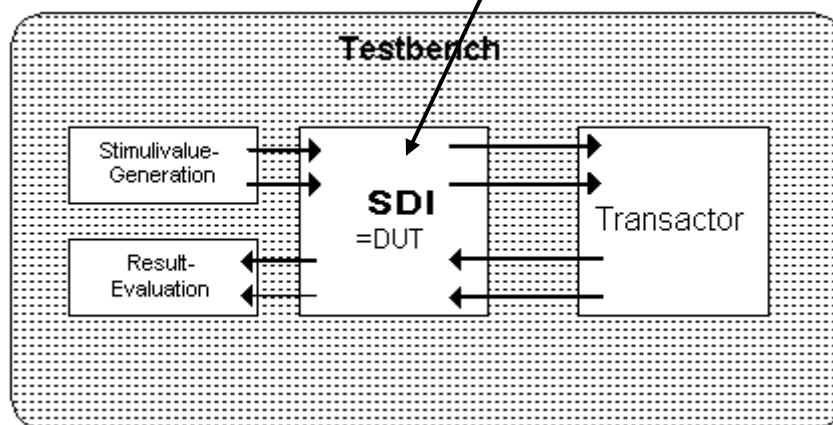
Text aus der Diplomarbeit:

..... Die vorliegende Arbeit beschreibt die nichtflüchtige Aufzeichnung digitaler Audio-Daten auf eine Secure Digital (SD) Memory Card. Der Peripherie-Modul wurde vollständig in der Hardwarebeschreibungssprache Verilog codiert. Dabei wurde der FPGA-Designflow mit Simulation, Synthese und Place&Route durchlaufen.

Das Auslesen der aufgezeichneten Daten via USB zu einem PC sowie die Steuerung des Aufzeichnungssystems über ein Graphical-User-Interface (GUI) war Teil einer weiteren Diplomarbeit.

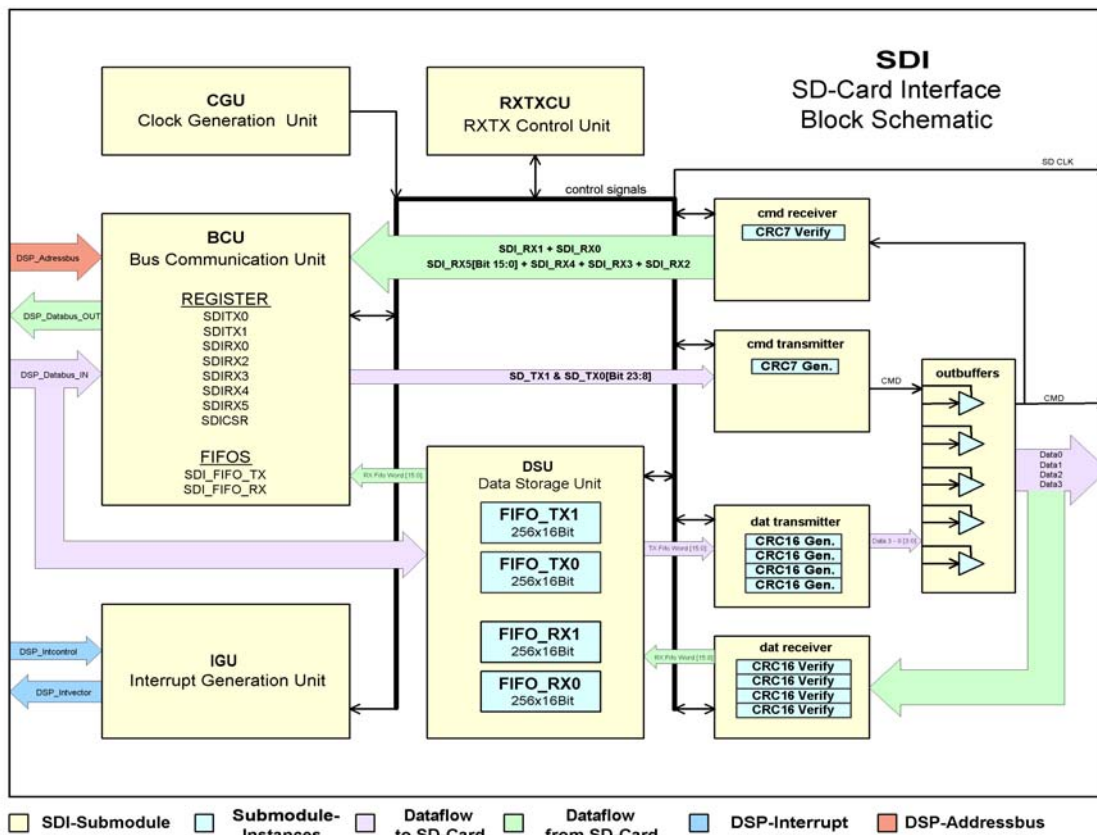


Blockschaltbild des zu entwickelnden Aufzeichnungssystems

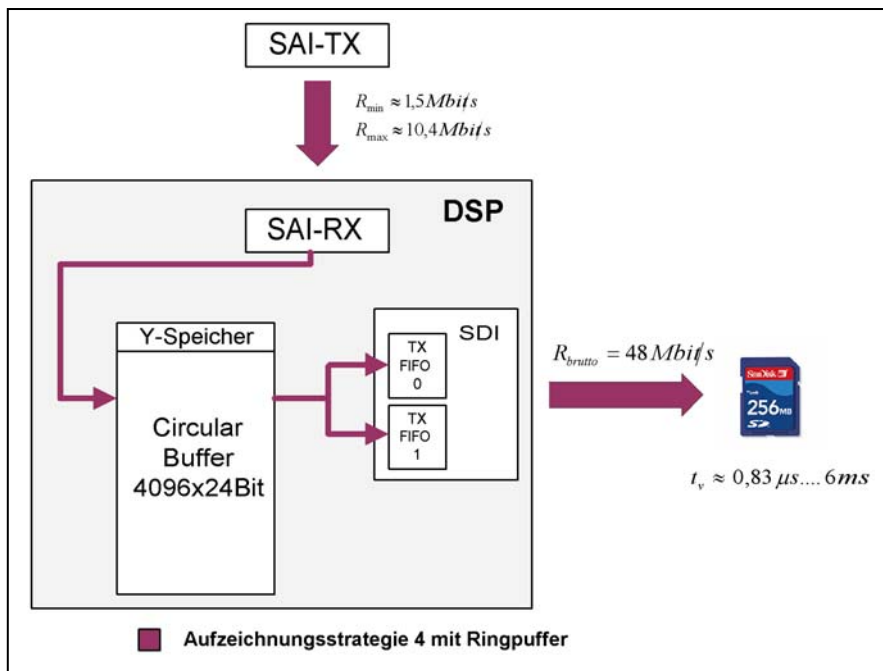


Struktur der Simulationsumgebung

Der hierfür entwickelte Transactor ist nur auf die wesentlichen Funktionen beschränkt und bildet eine SD-Card nicht vollständig ab. So kann er die wichtigsten Kommandos interpretieren und die vom Host erwartete Antwort generieren und senden. Der Transactor beinhaltet alle Antworttypen, jedoch mit statischem Informationsgehalt. Dies bedeutet, dass eine Antwort mit der Nummer Eins immer gleiche Daten besitzt.



Blockschaltbild des SD-Card-Interfaces SDI



Die sporadisch auftretenden enorm hohen **Verarbeitungszeiten** t_v von bis zu 6 ms mussten auf irgendeine Weise kompensiert werden. Die Idee war, einen Puffer zu verwenden, der die Daten in der Zwischenzeit, in der die Karte mit der Verarbeitung der Daten beschäftigt ist, aufammelt. Da der Realisierungsaufwand sehr umfangreich werden würde, weil im Vorfeld nicht vorhergesagt werden konnte, wann die langen Verarbeitungszeiten wirklich auftreten werden und wie oft, wurde entschieden, alle Daten direkt in einem Puffer zu schreiben und von dort wieder auszulesen. Dieser Mechanismus wurde mit Hilfe eines **Ringpuffers** realisiert, für den der gesamte Y-Speicher des DSPs verwendet wurde.

Datenpuffer = Ringpuffer (Circular Buffer) + FIFO

Sobald zwei Datenwörter in den **Ringpuffer** geschrieben werden, werden diese auch gleich wieder ausgelesen und in einen der **FIFOs** weitergereicht. Es wird also nicht gewartet bis ein vollständiger Block im Ringpuffer steht. Der Ringpuffer besitzt zwei Pointer; einen zum Schreiben der Eingangsdaten und einen zum Lesen der Ausgangsdaten.

1.5. Diplomarbeit „Entwicklung einer elektronischen Drosselklappen-Ansteuerung für den Motorsport“

(Thomas Kratz in Fa. BMW, Abteilung Motorsport)

Text aus der Diplomarbeit:

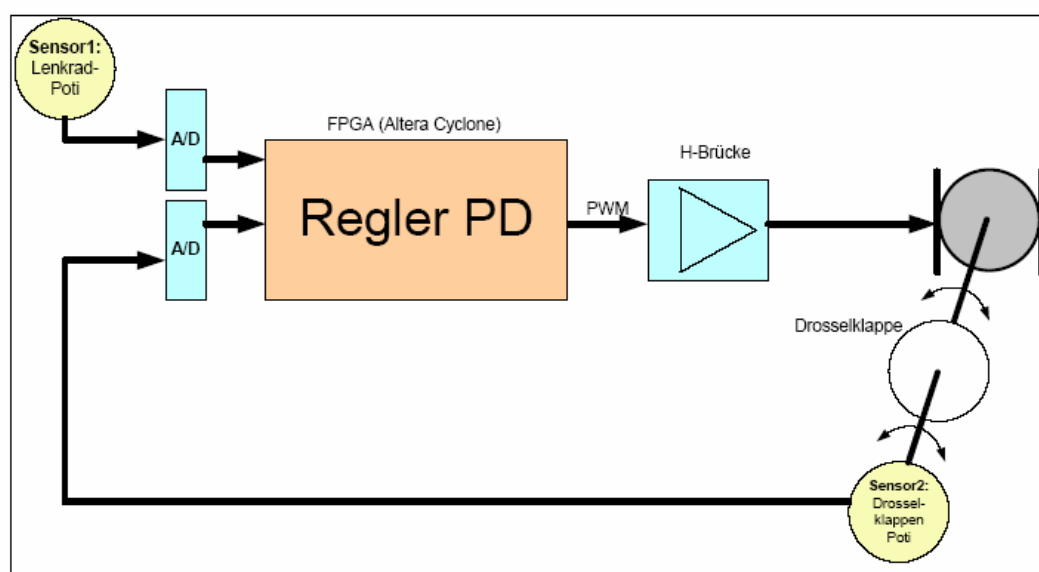
..... Der Inhalt dieser Arbeit beschreibt die Entwicklung einer elektronischen Drosselklappenansteuerung. Der Regler sollte in einem FPGA realisiert und in der Programmiersprache VHDL beschrieben werden. Field Programmable Gate Arrays (FPGAs) sind Hardwarebausteine, die ihre Anfänge in programmierbaren Logikbausteinen für einzelne Gatter (GALs) hatten. FPGAs haben in den letzten Jahren eine rasante Entwicklung erlebt. Sie stellen in einem Chip Ressourcen zur Verfügung, die einigen 10.000 bis zu mehreren Millionen Logikgattern entsprechen. Dadurch lassen sich sehr komplexe Schaltungen implementieren. Die Programmierung erfolgt über Hardware-Entwicklungstools. Diese übersetzen die in einer Hardware-Beschreibungssprache (z.B. VHDL) ausgedrückte Funktion in eine Chip-spezifische Form und generieren ein Konfigurationsfile, womit der Chip programmiert wird.

Selbst in den einfachsten PKWs befindet sich heutzutage ein Motorsteuergerät, welches diverse Steuerungen, Regelungen und Automatisierungseinheiten ansteuert. Eine Innovation hierzu sind die Fahrbefehle wie Lenken, Gasgeben, Bremsen, Kuppeln und Schalten elektronisch zu erzeugen, zu bearbeiten und an die entsprechenden Stellen im Fahrzeug weiterzuleiten. Der Überbegriff, unter welchem diese Aktionen zusammengefasst werden können, nennt sich „Drive-by-Wire“.

Drive-by-Wire:

Mittels elektrischer Leitungen (wire) werden Messsignale von Sensoren zu Steuerungs- oder Regelungseinrichtungen übertragen. Die von diesen Einrichtungen berechneten Steuerbefehle werden ebenfalls über elektrische Leitungen an die Aktoren weitergeleitet. In dieser Arbeit wird speziell die Möglichkeit des „elektronischen Gasgebens“ untersucht. Das elektronische Gaspedal (E-Gas) ersetzt den die Drosselklappe ansteuernden (mechanischen) Bowdenzug. Bei einem E-Gas System befindet sich am Gaspedal ein Potentiometer (Sollwert), welches einen Spannungswert an die Regeleinrichtung liefert. Ein weiteres Potentiometer an der Drosselklappe liefert den aktuellen Öffnungswinkel (Istwert). Der Winkel kann somit neben der vom Fahrer gewünschten Beschleunigung, auch in Abhängigkeit einer Vielzahl weiterer motorspezifischen Messdaten eingestellt werden.

Ziel dieser Arbeit ist die Entwicklung eines E-Gas Systems für ein Rennfahrzeug, jedoch mit der Modifikation, dass nicht mit dem Fuß sondern mit der Hand der Gassollwert vorgegeben wird. Dies wird über eine „Handgasmechanik“ am Lenkrad realisiert. Der Hintergrund besteht darin, dem Rennfahrer Alex Zanardi, der in einem schweren Rennunfall beide Beine verloren hat, die Möglichkeit zu geben weiter Rennfahrzeuge steuern zu können. Das E-Gas System soll das sehr schwergängige mechanische System ersetzen. Die Regelung der E-Gas Drosselklappe nimmt einen wichtigen Stellenwert ein. Es besteht ein großer Anspruch, sowohl an das Führungsverhalten, speziell die Minimierung der Anregelzeit und der Verzögerungszeit wie auch gute Stabilitätseigenschaften des Regelkreises. Der Algorithmus soll in einem FPGA implementiert werden.



„**Gleichstrommotorregelung realisiert in einem FPGA**“. Der konkrete Anwendungsfall, der sich hinter diesem Thema verbirgt ist die elektronische Ansteuerung einer Drosselklappe (vgl. Abbildung).

2. Datenpuffer

www.IDT.com, www.CYPRESS.com

2.1. Datenpuffer-Prinzipien (funktionelle Ebene)

Datenpuffer sind ein zentrales Element bei der Datenübertragung und beeinflussen wesentlich die resultierende Datenrate. Datenpuffer sind an verschiedenen Stellen eines Rechners realisiert, z.B.

- **an der Schnittstelle von zwei (asynchronen) Systemen:**

Der Datenpuffer dient als Zwischenspeicher für Nachrichten und Daten.

- **im Peripherie-System (Geräte-Controller, Host-Adapter):**

Der Datenpuffer dient als Zwischenspeicher für die Daten auf dem Weg vom Gerät zum Arbeitsspeicher (und umgekehrt). Er garantiert dem Gerät eine feste Datenrate. Die Datenrate über den Bus zum Arbeitsspeicher ist meist schwankend, d.h. abhängig von der Busauslastung und von der Priorität des Geräts am Bus.

- **im Arbeitsspeicher:**

Das Betriebssystem kann Teile des Arbeitsspeichers als Datenpuffer oder Befehlspuffer definieren.

Betrachtete Datenpuffer:

Frage: Unterschiede obiger Datenpuffer?

Die Puffer-Prinzipien werden mit dem Ziel diskutiert, den jeweiligen Datendurchsatz zu berechnen (Datenpuffer mit Speichergeräten als Datenquelle und/oder Datensenke). Die (Hardware-) Realisierung der Puffer sollte mit dem Wissen aus anderen Vorlesungen (Digitaltechnik, Mikroelektronik, Rechnergestütztes Schaltungs-Design) möglich sein.

2.1.1. Einfach-Puffer

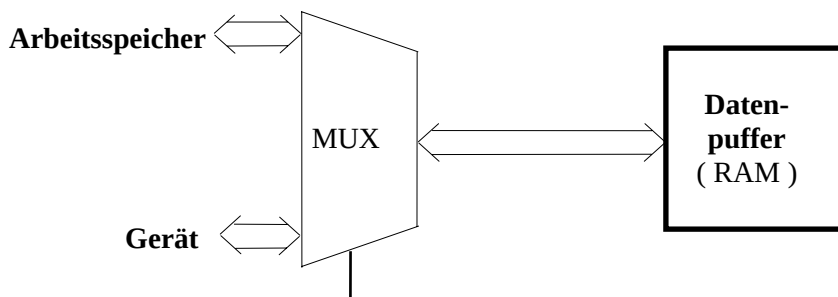
Sequentieller Betrieb Füllen (Schreiben) → Leeren (Lesen) → Füllen →...., d.h., erst wenn der Vorgang "Füllen" beendet ist, kann mit dem "Leeren" begonnen werden. Beispiele für Einfachpuffer sind „normale“ SRAMs und DRAMs.

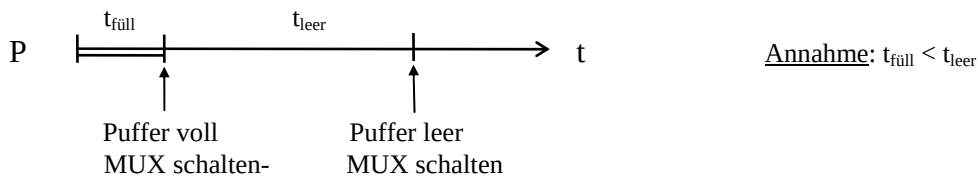
{HW-mäßig wäre Schreiben und Lesen gleichzeitig (verschachtelt) möglich, aber:

-- Zeigerverwaltung Lesen/Schreiben aufwendig (SW)

-- Überholen des Schreibzeigers durch den Lesezeiger: Lesen von ungültigen Daten

-- Überholen des Lesezeigers durch den Schreibzeiger: Überschreiben von gültigen Daten}





Aufgabe: Datentransfer über Einfachpuffer

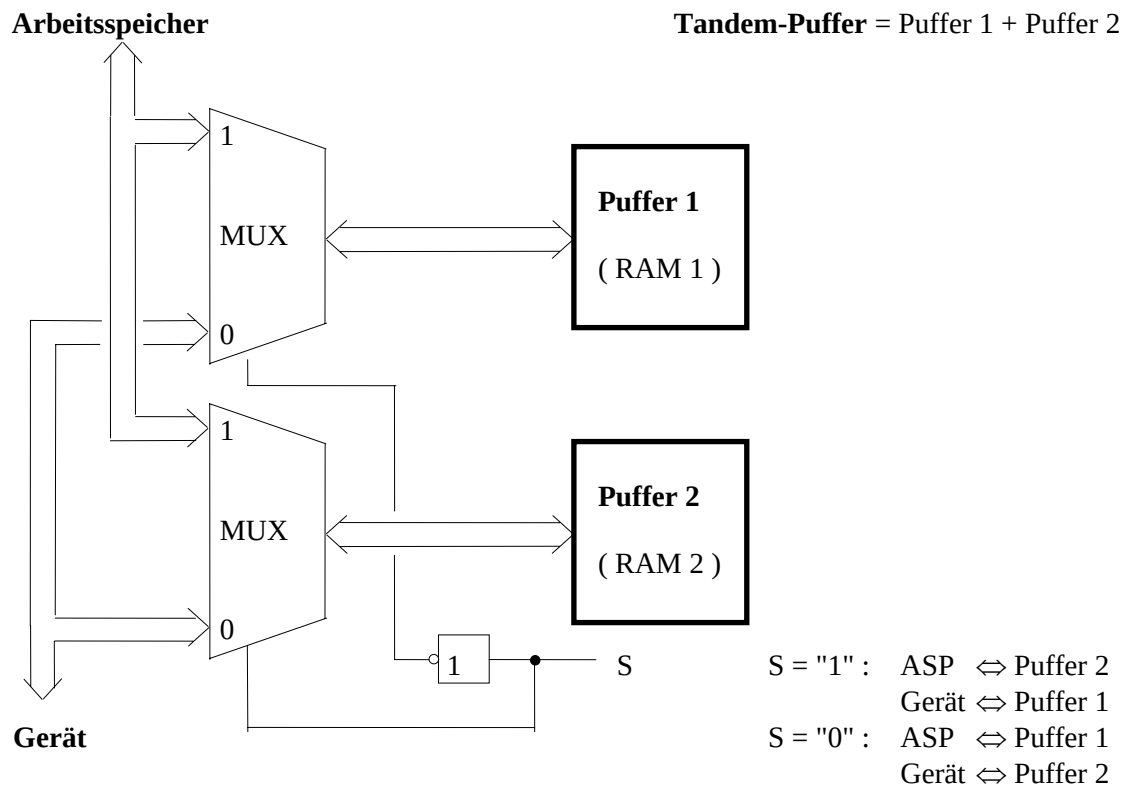
2.1.2. Tandem-Puffer (Wechsel-Puffer)

Tandem-Puffer = 2 x Einfach-Puffer (P1, P2)

z.B. 2 x ... kByte, 2 x ... MByte .. (RAM)

Parallelbetrieb möglich (z.B. Lesen P1 und gleichzeitig Schreiben P2)

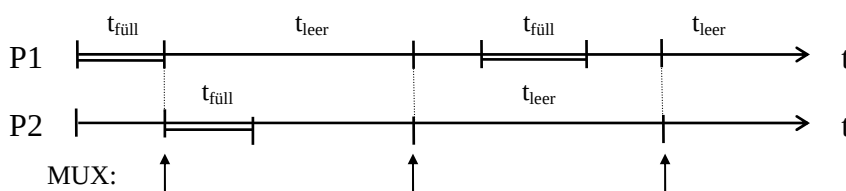
→ höherer Datendurchsatz als bei Einfach-Puffer.



Annahmen: Gerät "füllt" und Transfer zum Arbeitsspeicher "leert" den Puffer

$t_{\text{füll}} < t_{\text{leer}}$

$t=0$: beide Puffer leer



Zeitpunkt für Füllen, Leeren bestimmt durch: Gerät, Arbeitsspeicher (Bus), Pufferzustand leer/voll

Fragen: Wieso nach $t_{\text{füll}}$ in P2 nicht sofort neues $t_{\text{füll}}$ in P1 ?

Wieso nach $t_{\text{füll}}$ in P2 nicht sofort tleer ?

Wieso nach tleer in P1 nicht sofort neues $t_{\text{füll}}$?

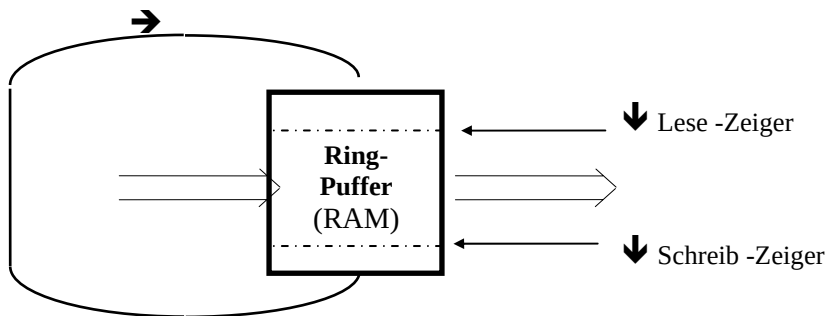
Unterschied zwischen Anlauf-Phase und eingeschwungenem Zustand ?

Aufgabe: Datentransfer über Wechsellpuffer, Berechnung der optimalen Pufferkapazität

2.1.3. Ring-Puffer

Daten-Puffer, der zyklisch beschrieben und gelesen wird (interne Lese- und Schreib-Zeiger)

- Lese-, Schreibzeiger:
Lese- und Schreib-Adresse erzeugt der Puffer selbst.
Von außen wird keine Adresse an den Puffer angelegt.
- zyklisch:
Die Zeiger werden bei jedem Lese- oder Schreibzugriff automatisch inkrementiert. Nach Erreichen der höchsten Adresse wird wieder mit der niedrigsten Adresse begonnen.



Problemfälle:

- Lese-Zeiger überholt Schreib-Zeiger: Lesen von ungültigen Daten (Lesen zu schnell)
- Schreib-Zeiger überholt Lese -Zeiger: Überschreiben von noch nicht gelesenen Daten (Lesen zu langsam)

Einsatzgebiet: z.B.

Nach dem Triggerzeitpunkt werden z.B. noch $K/2$ Daten aufgezeichnet. Damit stehen $K/2$ -Aufzeichnungsdaten vor dem Triggerzeitpunkt und $K/2$ -Aufzeichnungsdaten nach dem Triggerzeitpunkt im Speicher (K = Kapazität (kByte) des Aufzeichnungsspeichers).

Anwendungsbeispiel siehe Diplomarbeit Stefan Ramböck.

2.1.4. FIFO-Puffer

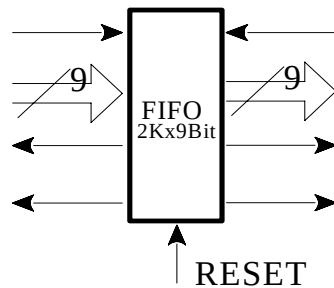
FIFO = ...

Unterschied zu RAM-Bausteinen:

- Adresseingänge ? : ...
- Schnittstellen (Ports) ? :
- Überschreiben noch nicht gelesener Daten ? ...
Lesen schon gelesener Daten ? ...

Write-Port:

Read-Port:



Das Signal WRITE startet den Schreibzyklus (Adresse = Schreibzeiger), das Signal READ den Lesezyklus (Adresse = Lesezeiger). Schreib- und Lesezyklen können unabhängig voneinander gestartet werden.

Der FIFO-Puffer ist ein Ringpuffer, erlaubt aber kein gegenseitiges Überholen von Schreib- und Lesezeiger. Schreib- und Lesezeiger werden von der Hardware (beim FIFO-Baustein) bzw. von einer Programm-Routine (beim Software-FIFO) verwaltet.

Zusätzlich sind in der Regel Füllstands-Anzeigen des FIFO-Puffers vorhanden (= Signale "voll", "halbvoll", "leer" etc.); siehe auch Kapitel "Datenpuffer-Realisierungen". Die Füllstandsanzeige ist bei manchen FIFOs programmierbar.

Asynchrones FIFO:

Die Schreib/Lese-Signale an beiden Ports hängen von unterschiedlichen Takten ab. Dies erfordert im FIFO eine komplexerer Logik gegenüber synchronen FIFOs. Synchroner FIFOs (gleiches Taktsignal an beiden Ports wirksam) erlauben höhere Taktraten.

Datenbreite/Adresstiefe:

FIFOs lassen sich kaskadieren hinsichtlich höherer Datenbreite (z.B. 9 Bit → 18/ 27/ 36/ .. Bit) und hinsichtlich höherer Adresstiefe (z.B. 64k → 128k/ 256k/ ..).

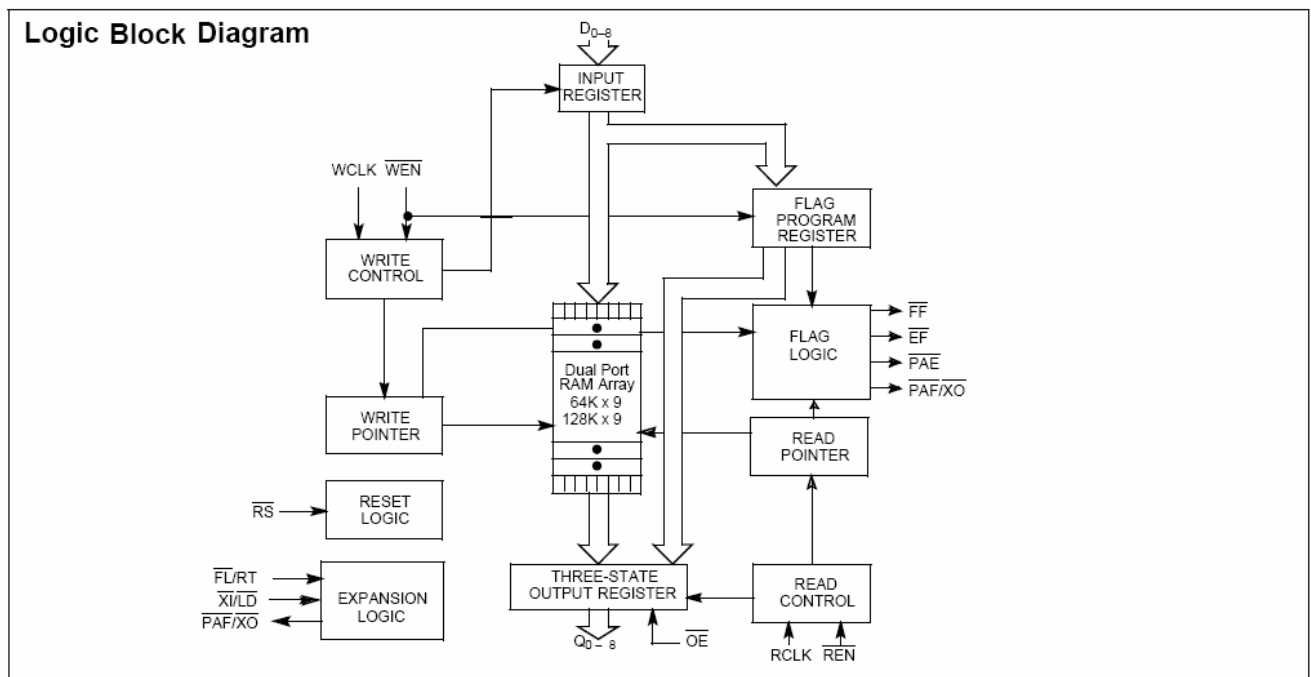
Realisierung:

- FIFO-Chips
- Verwendung eines FIFO-Cores (VHDL-Core), synthetisiert in einem FPGA-Chip
- Standard-Speicherbausteine DRAM oder SRAM, die mit einer Speicherzugriffslogik als FIFO organisiert sind.
Diese Lösung dürfte bei großen Kapazitäten kostengünstig sein.
- Bereich im Arbeitsspeicher, Zugriff über Programmroutine („Software-FIFO“)

Aufgaben: Datentransfer über FIFO-Puffer

Einsatzgebiete:

Detailliertes Blockdiagramm (CY7C4282V):



Signal Name	Description	I/O	Description
WEN	Write Enable	I	The only write enable when device is configured to have programmable flags. Data is written on a LOW-to-HIGH transition of WCLK when WEN is asserted and FF is HIGH.
REN	Read Enable	I	Enables the device for Read operation. REN must be asserted LOW to allow a Read operation.
WCLK	Write Clock	I	The rising edge clocks data into the FIFO when WEN is LOW and the FIFO is not Full. When LD is asserted, WCLK writes data into the programmable flag-offset register.
RCLK	Read Clock	I	The rising edge clocks data out of the FIFO when REN is LOW and the FIFO is not Empty. When LD is LOW, RCLK reads data out of the programmable flag-offset register.
EF	Empty Flag	O	When EF is LOW, the FIFO is empty. EF is synchronized to RCLK.
FF	FullFlag	O	When FF is LOW, the FIFO is full. FF is synchronized to WCLK.
PAE	Programmable Almost Empty	O	When PAE is LOW, the FIFO is almost empty based on the almost empty offset value programmed into the FIFO . PAE is synchronized to RCLK.
PAF/XO	Programmable Almost Full/Expansion Output	O	Dual-Mode Pin. Cascaded – Connected to XI of next device. Not Cascaded – When PAF is LOW, the FIFO is almost full based on the almost full offset value programmed into the FIFO . PAF is synchronized to WCLK.
FL/RT	First Load/Retransmit	I	Dual-Mode Pin. Cascaded – The first device in the daisy chain will have FL tied to VSS; all other devices will have FL tied to VCC. In standard mode or width expansion, FL is tied to VSS on all devices. Not Cascaded – Retransmit function is available in stand-alone mode by strobing RT.
XI/LD	Expansion Input/Load	I	Dual-Mode Pin. Cascaded – Connected to XO of previous device. Not Cascaded – LD is used to write or read the programmable flag offset registers. LD must be asserted LOW during reset to enable standalone or width expansion operation. If programmable offset register access is not required, LD can be tied to RS directly.
OE	Output Enable	I	When OE is LOW, the FIFO's data outputs drive the bus to which they are connected. If OE is HIGH, the FIFO's outputs are in High Z (high-impedance) state.
RS	Reset	I	Resets device to empty condition. A reset is required before an initial read or write operation after power-up.

andere Realisierung der Flags (Ausschnitt aus Datenblatt, anderer FIFO-Baustein):

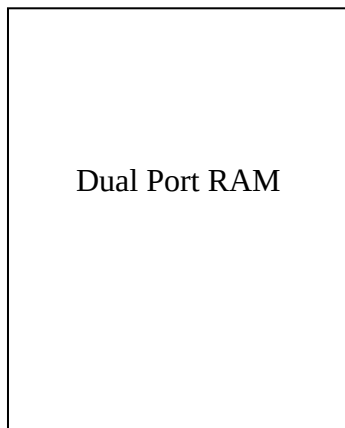
AE	Almost-Empty Flag	O	Programmable Almost-Empty flag synchronized to CLKB. It is LOW when the number of words in the FIFO is less than or equal to the value in the offset register, X.
AF	Almost-Full Flag	O	Programmable Almost-Full flag synchronized to CLKA. It is LOW when the number of empty locations in the FIFO is less than or equal to the value in the Offset register, X.

2.1.5. Dual Port RAM

Im Unterschied zu einem "normalen" RAM-Speicher (= **Single** Port RAM) besitzt der **Dual** Port RAM signalmäßig (Adreß-, Daten- und Steuersignale) zwei getrennte "Ports" (Schnittstellen), über die unabhängig voneinander Zyklen mit dem RAM durchführbar sind. Unabhängig bedeutet hier: Schreib- oder Lesezyklen mit beliebigen, von außen angelegten Adressen. Die Anschlüsse (Pins) für Adreßbus, Datenbus und die Steuersignale sind doppelt vorhanden. Über beide Ports wird auf dieselbe Speichermatrix des RAMs zugegriffen

(Beispiel: Dual Port Static RAM IDT70V27, 32k x 16 Bit).

Aufgabe: Ergänzen Sie den Dual Port RAM mit den wichtigsten Signalen



wichtige Unterschiede zum FIFO-Baustein:

...

...

Konfliktfall:

Gleichzeitige Zugriffe von beiden Ports auf dieselbe Speicheradresse werden über die Signale **/BUSY-R** (R=Right Port) und **/BUSY-L** (L=Left Port) angezeigt; eventuelle Schreibzugriffe werden unterdrückt.

Einsatzgebiete:

- Datenübergabe zwischen (asynchronen) Systemen, z.B. zwei Prozessor-Systemen.
- Datenübergabe zwischen Geräten mit unterschiedlichen Datenraten, usw.

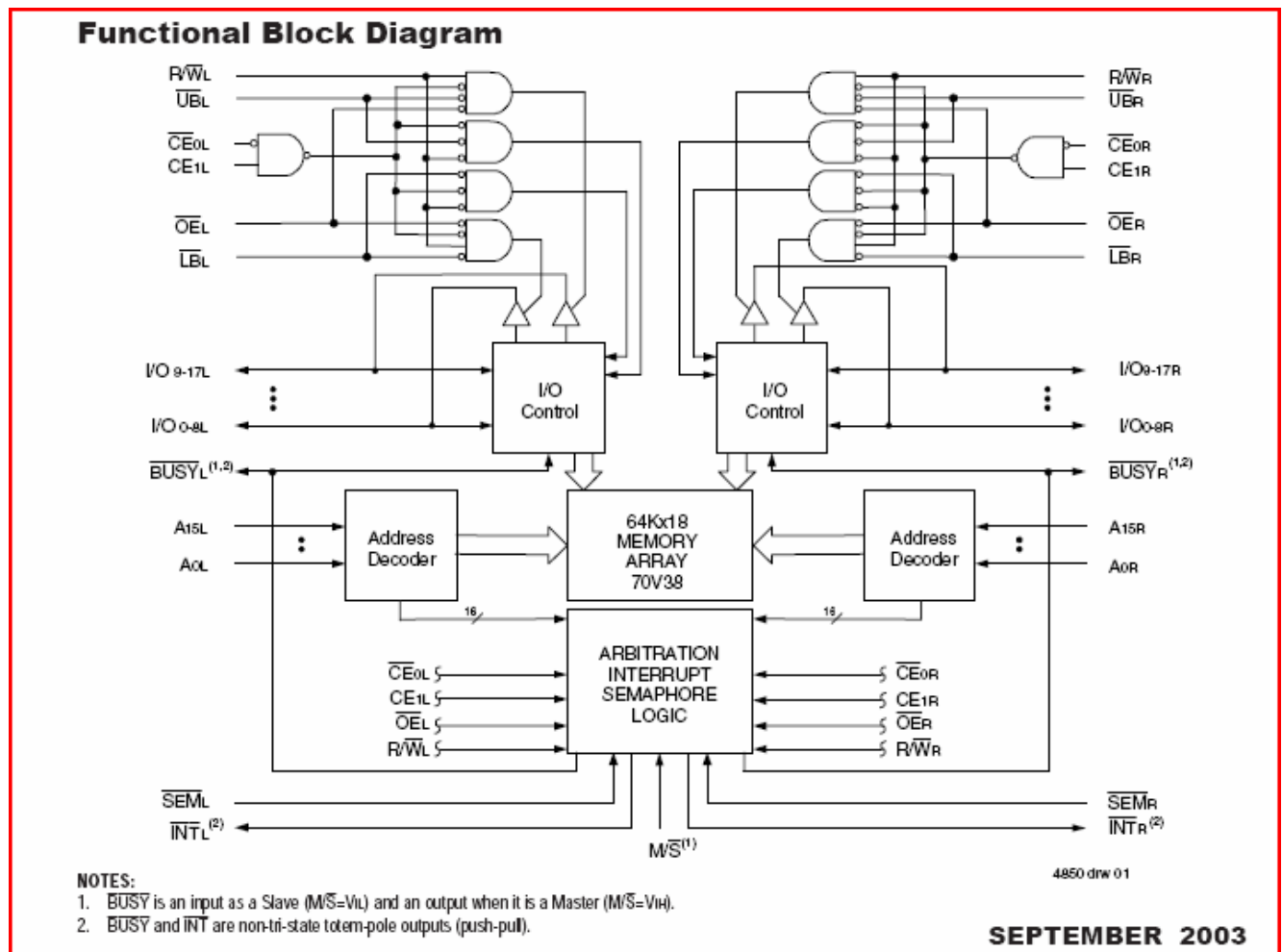


Bild: Blockdiagramm eines Dual Port RAM Bausteins

Signale: / = low-aktiv	
/SEM	Semaphore Enable
/UB	Upper Byte Select
/LB	Lower Byte Select
/INT	Interrupt Flag
/BUSY	Busy Flag
M /S	Master or Slave Select (für Kaskadierung)

Zusätzliche Funktionen:

Dual Port RAMs sind in vielen Anwendungen ein gemeinsamer Speicher, auf den z.B. zwei getrennte Schaltungsteile oder Schaltungssysteme (Prozessorsysteme) über die Ports zugreifen. Für die Steuerung der Datenübergabe und für die konfliktfreie Verwaltung des gemeinsamen Speichers stellen Dual Port RAM Bausteine folgende Funktionen zur Verfügung: → **Kommunikations-Schnittstelle**

-- Interrupt-Signalisierung ("mailing"):

Jeder Port kann durch Beschreiben einer bestimmten Speicheradresse (Bspl.: Adresse 7FFE für rechten Port und 7FFF für linken Port) am anderen Port ein Interrupt-Signal (/INT-R oder /INT-L) auslösen. Durch Auslesen der jeweiligen Speicheradresse (Adresse 7FFE durch den linken Port und 7FFH durch den rechten Port) kann der angesprochene Port das Interrupt-Signal wieder zurücksetzen.

-- Semaphore Flags:

Über ein Semaphore Flag (Semaphore, griech. "Zeichenträger") kann ein Port dem anderen Port signalisieren, daß er z.B. einen Speicherbereich des Dual Port RAM im Moment beschreibt oder liest und daß der andere Port auf diesen Bereich nicht zugreifen soll. Jedes Semaphore Flag kann vom linken oder vom rechten Port gesetzt werden, ist aber nur von dem Port, der es gesetzt hat, rücksetzbar. Welcher Speicherbereich gemeint ist, muß vorher zwischen den Programm-Routinen auf beiden Seiten vereinbart werden.

Bspl.: 8 Semaphore Flags im Baustein, realisiert mit 8 Latches, die außerhalb des normalen Adressraums sind: Adressierung über Signal /SEM_L bzw. /SEM_R aktiv und Auswahl über Adressbits A2..A0. Jedes Semaphore Flag ist mit einem Schreibzyklus (Adresse A2..A0) bei gleichzeitiger Aktivierung des Signals /SEM setzbar (und rücksetzbar) und mit einem Lesezyklus lesbar.

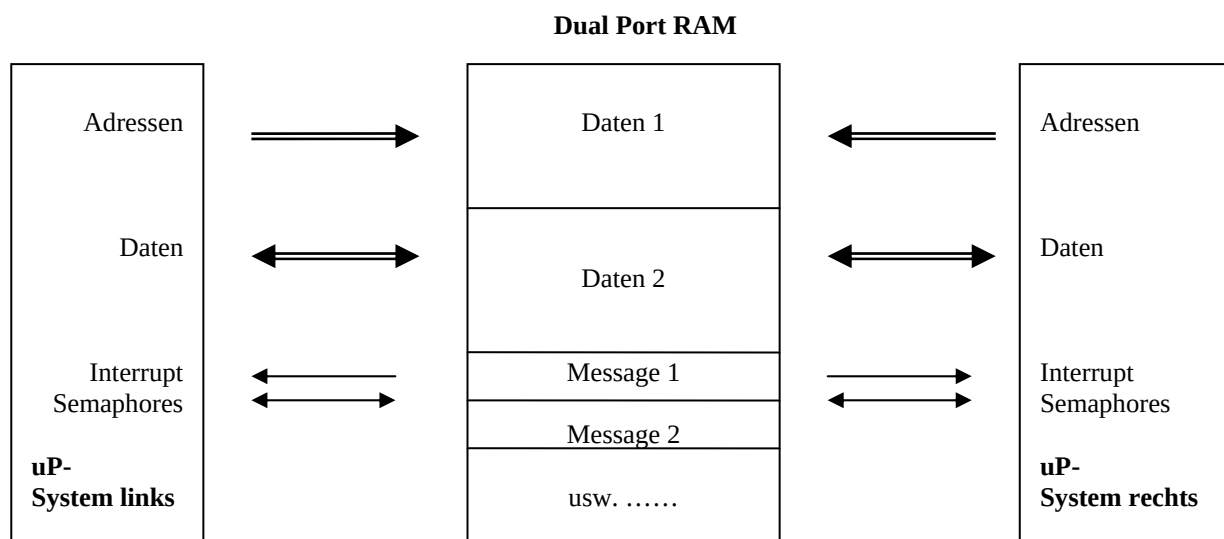
Über Semaphore Flags kann die Software auch andere gemeinsam benutzte Schaltungsteile verwalten.

Frage: Welche Möglichkeiten hat der Entwickler, um einen gleichzeitigen Zugriff von beiden Ports auf dieselbe Speicherzelle zu erkennen bzw. zu vermeiden.

.....
.....

Aufgaben: Datentransfer über Dual Port RAM-Puffer

Anwendung: Dual Port RAM-Speicher verbindet 2 Systeme (Daten- und Nachrichtenaustausch)



2.2. Realisierung von Datenpuffern (Beispiele)

a) FIFO und Dual Port RAM:

- FIFO- und Dual Port RAM-Bausteine
- VHDL/Verilog-Cores, synthetisiert in FPGA-Bausteinen
- Hybrid-Lösung, kostengünstig bei großen Speicherkapazitäten:
 - Speichermedium realisiert mit SRAM- oder DRAM-Bausteinen/Modulen
 - Ansteuerung realisiert mit FPGA-Baustein (VHDL/Verilog-Beschreibung, VHDL/Verilog-Core)
- Arbeitsspeicher-Bereich, entsprechend deklariert. Die Zugriffsroutinen bilden das FIFO- oder Dual Port RAM-Verhalten an.

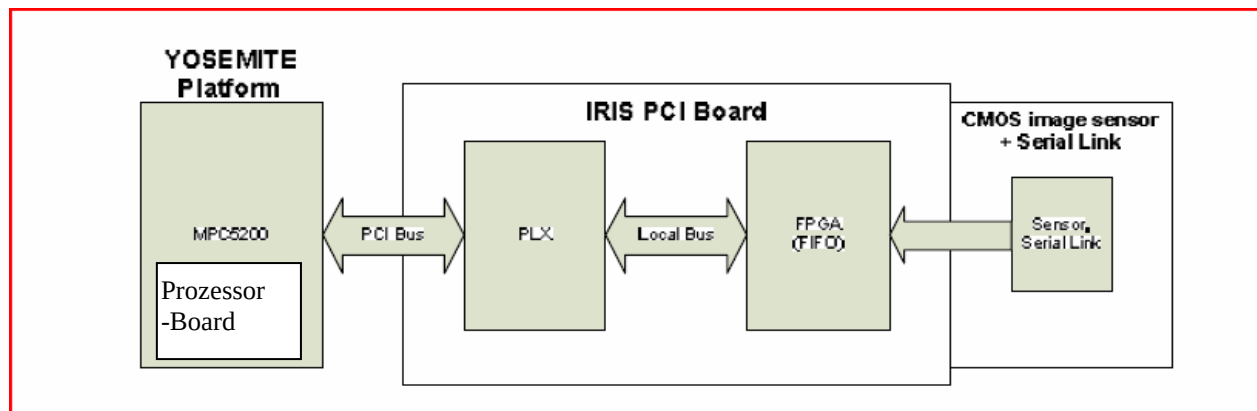
b) Wechsel-Puffer:

Verhalten/Realisierung wie Dual Port RAM-Puffer

Wie hoch ist dann die Datenrate beim Transfer einer Datenmenge $\gg$ RAM-Kapazität ?

Aufgabe DP-3: Datentransfer über FIFO-Puffer, vom Bildsensor zum Prozessorboard

Datentransfer von einem Bildsensor(Kamera) über einen FIFO-Puffer zum Prozessorboard (DA Automobilbereich).



R_{sensor} (MByte/s) = Datenrate Sensor $\rightarrow$ FIFO

R_{proz} (MByte/s) = Datenrate FIFO $\rightarrow$ Prozessorboard

Bei $t=0$: Der Sensor beginnt mit dem Füllen des Puffers (Zu Beginn ist der Puffer leer).
Der Prozessor erhält das Signal zum Leeren des Puffers.
 t_s = Zeit, nach der das Prozessorboard mit dem Leeren beginnt.

1. Berechnen Sie allgemein die übertragene Datenmenge D_1 des Bildsensors als Funktion der Zeit t !
(Der Zeitpunkt $t=0$ ist oben definiert)
2. Berechnen Sie allgemein die übertragene Datenmenge D_2 des Prozessorboards als Funktion der Zeit t !
(Der Zeitpunkt $t=0$ ist oben definiert)
3. Die Datenrate R_{proz} hängt stark von der Auslastung des PCI-Busses durch andere Busteilnehmer ab. Im ungünstigsten Fall ist $R_{\text{proz}} = 5$ MByte/s. Wie lange darf die niedrige Datenrate anhalten ohne dass der Datenstrom vom Sensor gestört wird?
(Datenrate $R_{\text{sensor}} = 10$ MByte/s, $t_s = 300\text{ms}$, FIFO-Kapazität = 16 MByte)
4. Stellen Sie die Ergebnisse von 1. bis 3. grafisch dar als Funktion über der Zeitachse t .
5. Mit dem Signal "Puffer voll" wird das Füllen gestoppt. Nach dem Signal "Puffer leer" wird der Transfer zum Prozessorboard gestoppt. Welche Datenmenge (MByte) wurde insgesamt übertragen ? (Zahlenwerte wie oben).

Aufgabe DP-4: Dual Port RAM- und FIFO-Puffer

Vergleichen Sie Dual Port RAM- mit FIFO-Puffer hinsichtlich der Adressierung bei Lese- und Schreibzyklen.

Dual Port RAM:

FIFO:

3. PCI-Bus (Peripheral Component Interconnect)

www.pcisig.com, www.plxtech.com, www.plda.com

3.1. Einführung

Die Rechnerperipherie ist direkt oder über Peripherie-Schnittstellen wie z.B. den SCSI-Bus am internen PCI-Bus des Computers (Host) angeschlossen, siehe **Bild "Busstruktur/Schnittstellen des PC"**

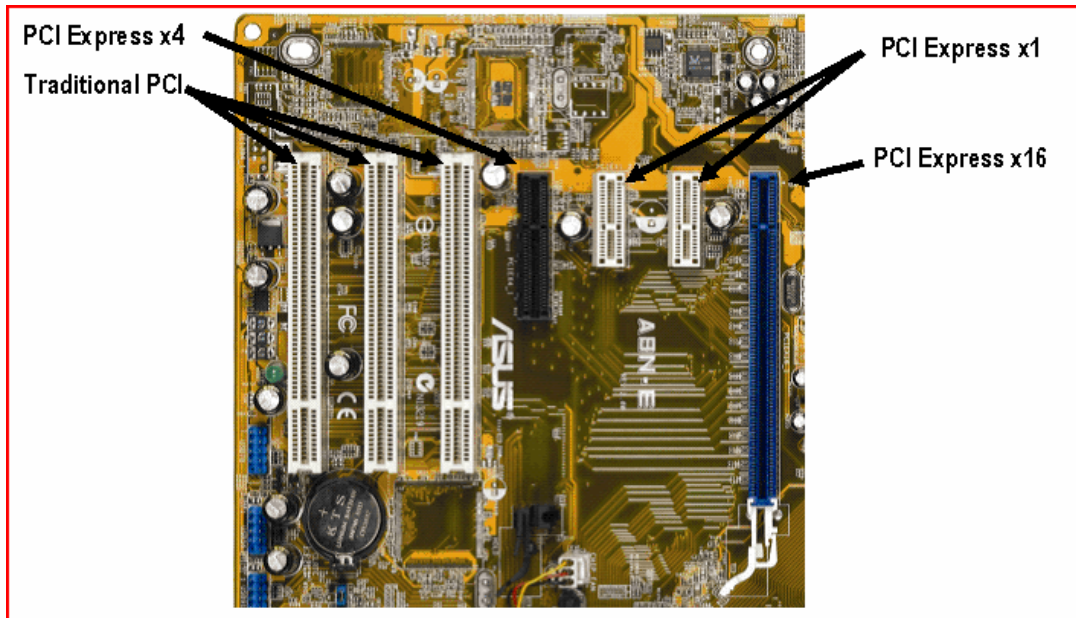


Bild: Platine mit PCI-Bus Steckplätzen (Slots)

3 Steckplätze PCI Conventional (Traditional), 4 Steckplätze PCIe (e=Express)

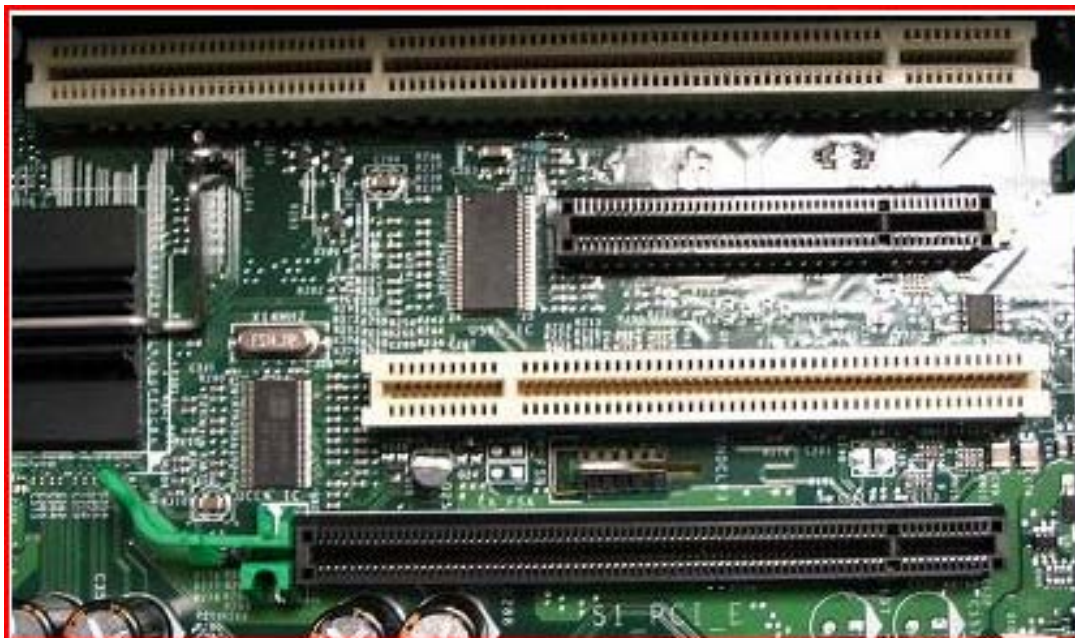


Bild: Platine mit PCI-Bus Steckplätzen (Slots)

Frage: Wie ist die genaue Bezeichnung der 4 PCI-Steckplätze?

Frage:

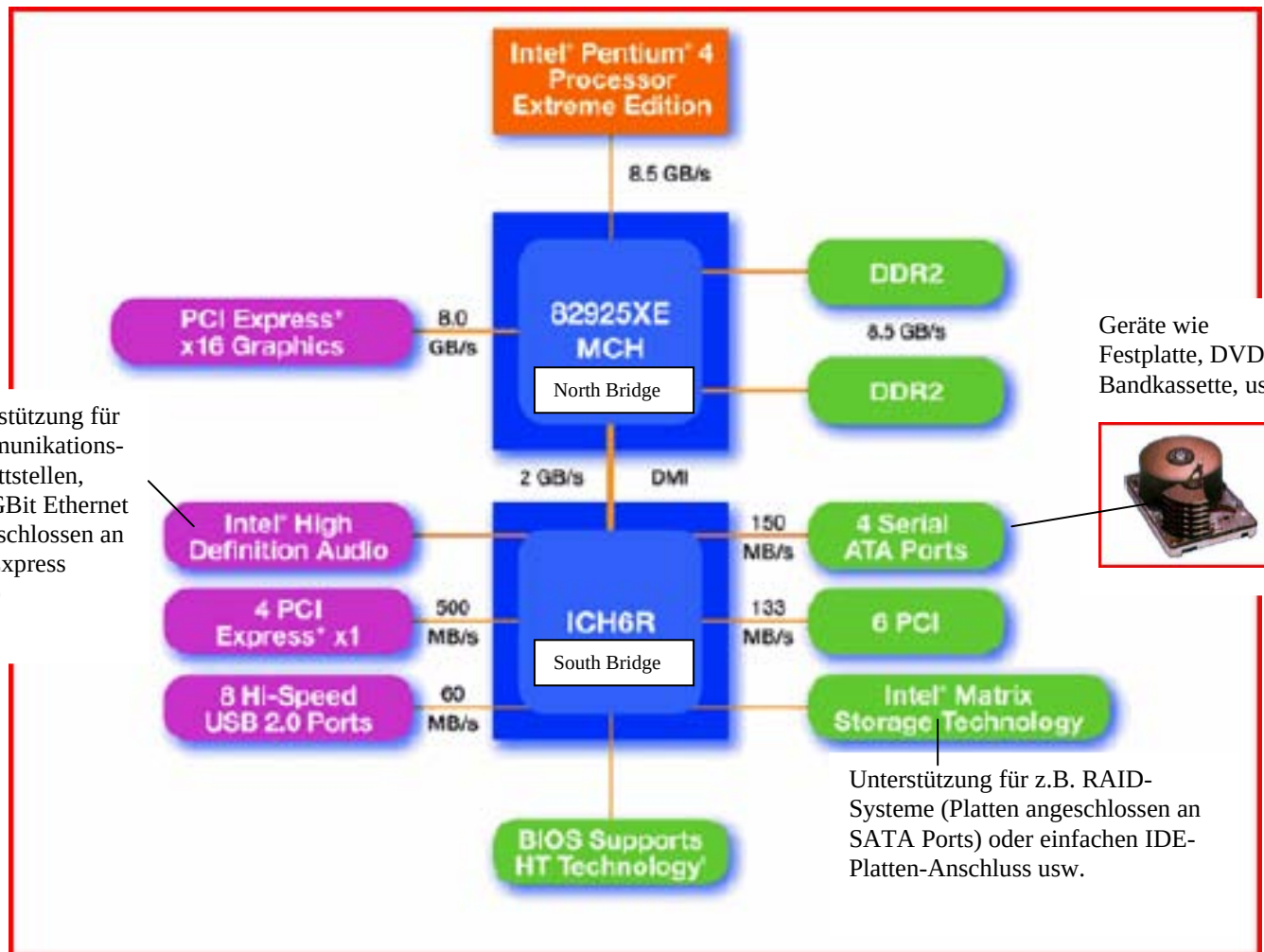
Wieso nicht einfach die Geräteperipherie am uP-Bus anschließen?

Unterstützung für
Kommunikations-
Schnittstellen,
z.B. GBit Ethernet
(angeschlossen an
PCI Express
Ports)

Geräte wie
Festplatte, DVD,
Bandkassette, usw.



Unterstützung für z.B. RAID-
Systeme (Platten angeschlossen an
SATA Ports) oder einfachen IDE-
Platten-Anschluss usw.



MCH Memory Controller Hub (Memory Bridge)

DMI Direct Media Interface

ICH6R I/O Controller Hub (I/O Bridge, I=Input, O=Output)

("... 8 multipurpose DMA engines, 4 input and 4 output, support of simultaneous independent data streams")

Bild: PC-Architektur (Fa. Intel)

Über die Schnittstellen PCI, SATA und USB sind alle gängigen Peripherie-Geräte (Funktionen) anschließbar.

Weitere Schnittstellen wie PCI-X, Firewire, SAS, SCSI, RS232, Enhanced Parallel Port EPP usw. sind nicht eingezeichnet.

Fragen: Was versteht man unter "Bus-Bandbreite"? Zahlenwerte abschätzen für PCI, USB und die Geräte!

Der übliche Chipsatz auf dem Motherboard teilt sich in die North- und die South-Bridge auf. Eine physikalische Trennung in zwei Bausteine ist nicht unbedingt nötig, d.h., sie können in einem Chip integriert sein.

Die North-Bridge regelt die Datenströme zwischen Prozessor, Cache und Arbeitsspeicher. Dort angeschlossen sind auch der AGP-Slot für die Grafikkarte und das gesamte Bussystem.

An der South-Bridge sind die Schnittstellen, die der Chipsatz anbietet, angeschlossen. Dazu gehören USB, serielle und parallele Schnittstelle usw. .

Hersteller von Chipsätzen sind Intel, AMD, Cyrix usw.

Der PCI-Bus verbindet im Host (PC) Prozessor und Arbeitsspeicher mit der Geräteperipherie.

Aufgabe: Bitte Tabelle ausfüllen!

	„Traditional (conventional)“	„Express“

Der PCI-Bus ist genormt und damit herstellerunabhängig (PCI-Standard, Normierungsgremium: PCI Special Interest Group PCISIG).

PC-Karten mit der älteren, langsameren ISA-Schnittstelle (die in gewissem Umfang noch auf dem Markt sind) können am PCI-Bus über einen PCI/ISA-Adapter (PCI/ISA-Bridge) angeschlossen werden.

3.2. Bus-Varianten

PCI Conventional (alt):

- 33MHz, 32 Bit Datenbus: 132MByte/s "Basismodell"
- 66MHz, 64 Bit: 528MByte/s

→ Norm Conventional PCI 3.0

PCI-X:

→ Norm PCI-X 2.0

Weiterentwicklung in Richtung höherer Frequenzen (Busbandbreite) unter Beibehaltung von Architektur, Protokoll, Signalen und Stecker des konventionellen PCI-Busses (→ Abwärtskompatibilität)

- 133MHz, 64 Bit: 1 GByte/s
- 266MHz
- 533MHz

Zusätzlich:

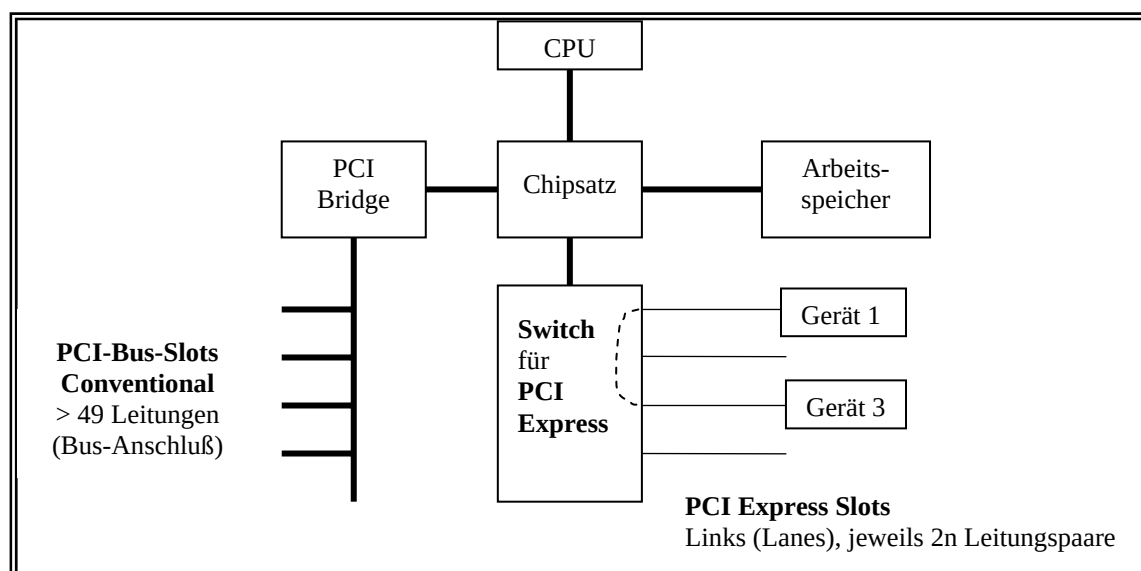
- 16Bit-Interface, spezifiziert für platzsparende Anwendungen.
- Fehlerkorrektur ECC mit 1Bit-Korrektur und 2Bit-Erkennung.
- größerer 4kByte-Konfigurationsbereich
- Device ID Message Transaction für Punkt-zu-Punkt-Verbindungen (z.B. zu einer Bandkassette)

PCI Express:

→ Norm PCI Express 1.1

- Serieller PCI-Bus mit differentieller Übertragung der Bits als Weiterentwicklung des derzeitigen Parallelbusses. Eine 8Bit/10Bit-Codierung der Daten ermöglicht ein im Datenstrom integriertes Taktsignal
- Pro Bitleitung 2 Adern (D+, D-) in Hin- und zwei Adern (D+, D-) in Rückrichtung = 1 Lane
Lane bezeichnet jeweils 2 Adernpaare.
Ein Link umfasst 1 oder mehrere (2x, 4x, 8x, usw.) Lanes. Über einen zentralen elektronischen Schalter (Switch) werden jeweils 2 Links verbunden und ergeben so eine Punkt-zu-Punkt-Verbindung. Maximal sind 32 Lanes pro Link möglich.
- Datenraten:
 - 1 Lane (PCIex1) 250 MByte/s (2,5 GBit/s x 8/10 x 1/8 → ca. 250 MByte/s)
 - 32 Lanes (PCIex32) 8 GByte/s
- Die Daten werden paketweise übertragen (Packet-Nr, CRC-Sicherung).
- 4 Adressräume: Memory-, I/O-, Configuration- und zusätzlich Message-Address Space (z.B. Message signalled Interrupts).

PCI-basierte Verbindungsstruktur mit PCI-Bus und PCI-Express-Links:



- Slot PCI-X: mehr Signale als PCI Conventional Slot (wegen 64 Bit Bus)
- Slot PCI Ex1: weniger Signale als PCI Conventional Slot (da nur 1 Link)

3.3. PCI Conventional

Viele Aussagen gelten unverändert auch für den neuen seriellen Bus PCI Express.

3.3.1. Adressierung

Bus-Teilnehmer:

"PCI Device" = Busteilnehmer ("Agent").

Ein Busteilnehmer ist vom Typ "Initiator Device" oder "Target Device" oder "Bridge Device".

Ein Busteilnehmer kann eine PCI-Steckkarte sein, z. B. ein PCI/SCSI-Adapter, oder eine auf dem "Mother Board" realisierte Funktion (z.B. USB Controller).

"Initiator" Device: beginnt einen Bustransfer (auch "Bus **Master**" genannt).

"Target" Device: vom Initiator adressierter Teilnehmer (auch "Bus **Slave**" genannt).

Jeder Bus-Teilnehmer kann bis zu 8 verschiedene funktionelle Einheiten besitzen (**Functions**, → "Multi-functional Devices").

Nach dem Einschalten ermittelt ("scanning") der Host-Prozessor über die Konfigurations-SW (BIOS, PCI Bus Enumerator) die am Bus angeschlossenen "PCI-Devices" und über deren Konfigurations-Register die Anforderungen an die Konfiguration. Mit "Anforderungen" ist gemeint, wieviel Adreßraum ein PCI-Device benötigt, ob eine Interruptleitung notwendig ist, welcher Device-Typ vorliegt und wer der Hersteller ist.

Mit den Angaben Geräteklasse (Typ) und Hersteller lädt der Host-Prozessor das Treiberprogramm. Manuelle Einstellungen über Schalter oder Steckbrücken entfallen ("plug and play").

PCI-Adresse (32 Bit):

Die Interpretation der Adresse hängt vom jeweiligen Befehlstyp im Zylus ab. Bei Konfigurationsbefehlen (Lesen bzw. Schreiben Konfigurationsregister) wird die Adresse wie gezeigt als Bus|Device|Function-Adresse interpretiert. Die Host/PCI-Bridge dekodiert die Device-Adresse (AD[15..11]) und aktiviert das entsprechende IDSEL-Signal für den dazugehörigen Slot:

HOST→Bridge: Device-Adresse -----Bridge→PCI-Bus: IDSEL-Signal

Konfigurationsadresse:

AD[23..16]=	Sekundärer PCI Bus (Kaskadierung)	{ IDSEL-Signal notwendig, da Konfig-Adressraum
AD[15..11]=	PCI Device (max. 32 Geräte am Bus)	{ aller Teilnehmer Parallel
AD[10..8]=	PCI Funktion (max . 8 Funktionen innerhalb eines Geräts)	
AD[7..2]=	Konfigurations-Register (1 aus 64), nur für Konfigurationstransfers	
AD[1..0]=	Adressierung eines Bytes in einem 4Byte-Wort	

Bei „normalen“ Memory-Read/Write-Befehlen wird die ganze Adresse als Speicheradresse interpretiert (Adressraum: Arbeitsspeicher+PCI-Speicherraum)

Signal IDSEL:

IDSEL wird realisiert mit AD[31..11] → 21 Signale, daher theoretisch 21 Geräte am Bus möglich.

Signal DEVSEL:

Alle Zugriffe, die nicht von einem PCI-Device des Busses beantwortet werden (mit Signal DEVSEL), werden auf den Erweiterungs-Bus geleitet („subtraktiver Dekodiermodus“)

3.3.2. Konfiguration

Die Konfigurations-SW stellt fest, wieviel Speicher- und wieviel I/O-Adressraum die angeschlossenen PCI-Device's benötigen und programmiert danach deren Speicher- und I/O-Adressdekoder so, daß keine Mehrfachbelegungen von Adressen stattfinden. Damit ist sichergestellt, daß bei Speicher- und I/O-Zugriffen vom Host-Prozessor immer nur ein Bus-Teilnehmer angesprochen wird. Sind im Konfigurations-Register Interrupts angezeigt, programmiert die SW die notwendige Routing-Information (Verbindung der PCI-Interruptleitungen INTA#..INTD# zu den Host-Prozessor-Interruptleitungen IRQ15..IRQ0). Dies kann in Abhängigkeit von der Device Address erfolgen.

Es entfallen manuelle Einstellung über Schalter oder Steckbrücken betreffend Adreßraum und Nummer der Interruptleitung.

Befehlscodes für Konfigurationszyklen:

- „Lesen Konfigurations-Register“
- „Schreiben Konfigurations-Register“

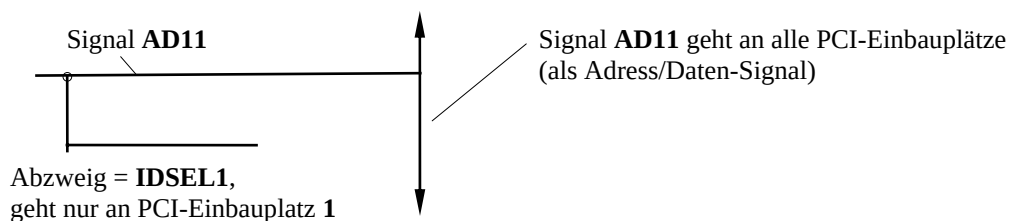
Daher existieren am PCI-Bus 3 Adressräume, die über den Befehlscode (4 Bit) im Buszyklus unterscheidbar sind:

- Configuration-Adressraum,
- Memory-Adressraum und
- I/O- Adressraum.

a) Abfrage der Bus-Einbauplätze (Slots)

Zu Beginn der Konfiguration wird jeder Bus-Einbauplatz über sein Signal **IDSEL_x** (Initialization Device Select) adressiert, d.h., die Software fragt mit Hilfe der IDSEL_x-Signale die PCI-Einbauplätze der Reihe nach ab, ob ein PCI-Device vorhanden ist (= eine PCI-Karte steckt) und liest dann die Konfigurations-Register des PCI-Device aus.

Da während der Konfigurationszyklen in Taktphase T1 die höherwertigen Adressen AD11..AD31 nicht benötigt werden können sie als IDSEL_x-Signale verwendet werden (Abzweig, an den IDSEL-Pins angeschlossen), z.B.



b) Konfigurationsregister

Die Konfigurationsdaten sind in den **Konfigurations-Registern** der Bus-Teilnehmer abgespeichert. Je nach Inhalt sind diese Register

- Nur-Lese-Register: **ROM**-Zellen, vom **Hersteller** programmiert, enthalten
- Schreib/Lese-Register: **RAM**-Zellen, vom **Host** beschrieben, übernehmen

Die **Konfigurations-Register** einer Funktion umfassen maximal 256 Bytes (= 64 doublewords DW = 64 x 4 Bytes) oder erweitert 4 kByte und sind aufgeteilt in:

- DW 00..15 **Configuration Header** (Die **schattierten Felder** sind obligatorisch)
- DW 16..63 (1k) **Device specific data**

Konfigurations-Register, Teil "Configuration Header":

Byte 3	Byte 2	Byte 1	Byte 0	DW
Device ID (Geräte-Typ, Vergabe d. Hersteller)		Vendor ID (fest) (Hersteller, Vergabe durch PCI SIG)		00
Status Register (Gerätestatus)		Command Register Bit2=Master Enable (Bus Master)		01
Class Code (fest) (Geräte-Klasse z.B. Storage Controller/SCSI-Interface)			Revision (Geräte- Version)	02
BIST (Built-In Self Test)	Header Type ³⁾	Latency Timer (Master, Zeit am Bus)	Cache Line Size	03
Base Address 0				04
Base Address 1				05
Base Address 2				06
Base Address 3				07
Base Address 4				08
Base Address 5				09
				10
				11
Expansion ROM Base Address (ROM des PCI-Device mit BIOS, Interrupt service routines, self-test. Enthält Funktionen für die Steuerung des PCI-Devices, die nicht im zentralen BIOS-ROM des Hosts enthalten sind).				12
				13
				14
Max_Lat (Master, max. Zeit am Bus)	Min_Gnt (Master, min. Zeit für GRANT-Signal)	Interrupt Pin ¹⁾ (fest, ROM-Zelle)	Interrupt Line ²⁾ (zugewiesener Host-Interr., RAM-Zelle)	15

fest="hard-wired", d.h., diese Informationen sind fest im Konfigurationsregister des PCI Device eingetragen (ROM).

¹⁾ Zeigt an, welchen Interrupt (INTA/B/C oder D#) das PCI Device benötigt.

²⁾ Die Konfigurations-Software verbindet das Interruptsignal des PCI Device (INTA/B/C oder D#) mit einem Interrupt-Eingang des Hosts (Prozessor). Der Vorgang heißt "routing" und wird durch Programmieren der "Programmable Interrupt Router"-Schaltung in der Host/PCI Bridge erreicht. Der zugewiesene Host-Interrupt (IRQ0..IRQ15) wird hier von der Konfigurations-SW eingetragen.
(Zusätzlich zu den Interruptsignalen des PCI-Busses müssen auch Interruptsignale z.B. des ISA-Busses an den Host weitergereicht werden).

³⁾ Im Register Header Type wird die Art des Device festgelegt, ob Single- oder Multifunktion, PCI-to-PCI Bridge usw. Dadurch entscheidet sich das Aussehen des Config. Space am Register 10h .

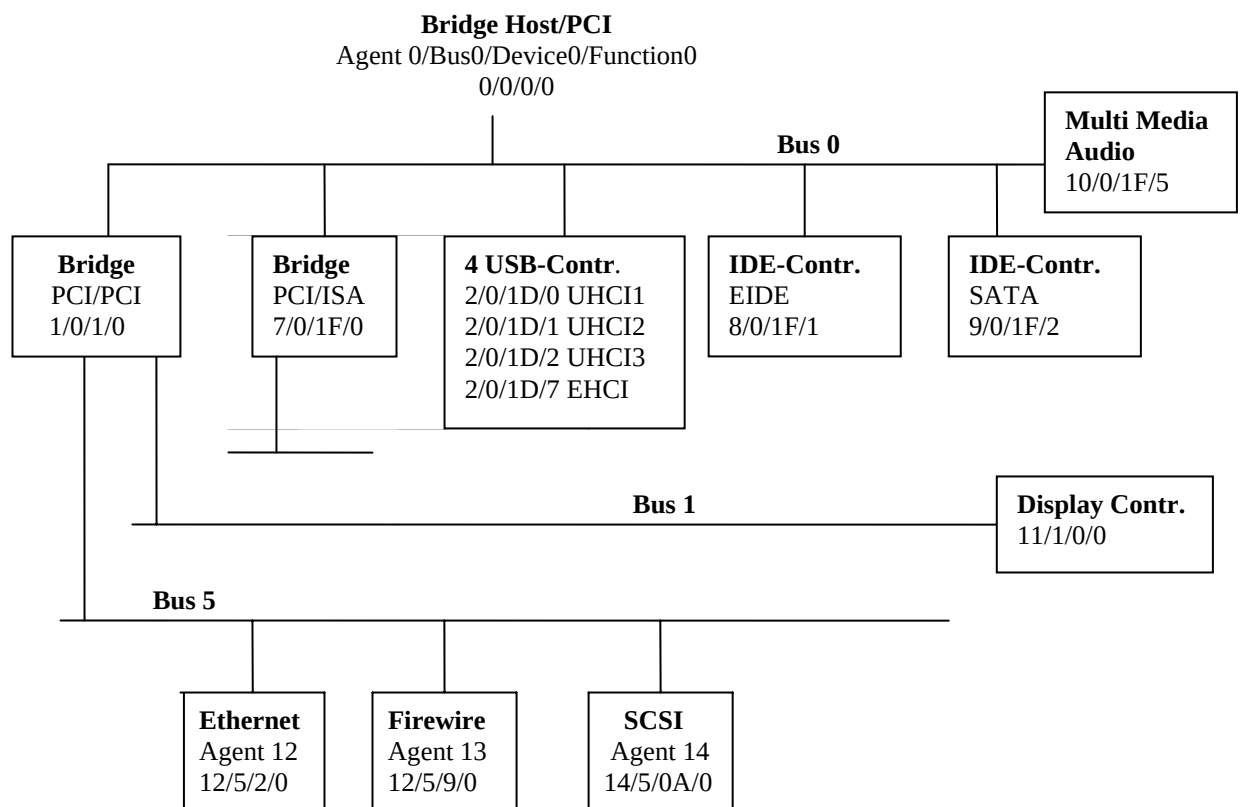
DEMO: Abfrage eines PCI-Busses und einzelner PCI-Devices mittels „PCI Tools“

Diskussion

1) Pcmng (PCI Manager): 15 Agents, Bild an Tafel: Busstruktur

Busstruktur Demo PC5=10.27.201.68: 4 Lasten am Bus (ohne Host-Bridge)

Agentd#	Bus#	Device#	Function#	
0	0	0	0	Host/PCI Bridge
1	0	1	0	PCI/PCI Bridge
2	0	1D	0	USBContr. UHCI#1
3	0	1D	1	USBContr. UHCI#2
4	0	1D	2	USBContr. UHCI#3
5	0	1D	7	USBContr. EHCI
6	0	1E	0	PCI/PCI
7	0	1F	0	PCI/ISA
8	0	1F	1	IDE-Contr. EIDE
9	0	1F	2	IDE-Contr. SATA
10	0	1F	5	MultiMedia Audio
11	1	0	0	Display Contr.
12	5	2	0	Ethernet
13	5	9	0	Firewire
14	5	0A	0	SCSIUltra160



Das Status Register gibt den aktuellen Zustand des PCI Device an.

Einige Bits sind reine Lese-Bits, andere Lese/Schreib-Bits, die bei Schreibzugriff gelöscht werden.

Die Bits 4 ..10 sind vom Hersteller fest verdrahtet (=Vorgaben des Entwicklers).

Die Bits 11 ..15 werden bei entsprechenden Bus-Ereignissen hardwaremäßig gesetzt.

-- Das „Capabilities List Bit“ gibt an, ob das Device einen erweiterten Konfigurationsadressraum besitzt.

-- Fast Back-to-Back Capable:

Fast Back-to-Back Capable ist '1', falls das Device diese Funktionalität unterstützt:

Fast Back-to-Back Capable: optional für Bus Master. Ist die Fähigkeit, 2 aufeinanderfolgende Busbelegungen (transactions) zu 2 verschiedenen Targets ohne idle state (1 Takt Pause) durchzuführen.

Status Reg.=1: Target ist "Fast Back-to-Back Capable" fähig

Command Reg.=1/0: Eigenschaft ist ein/ausgeschaltet

-- DEVSEL timing: Target sendet DEVSEL, wenn es angesprochen wird. Initiator bricht ab, falls kein DEVSEL innerhalb einer bestimmten Zeit kommt.

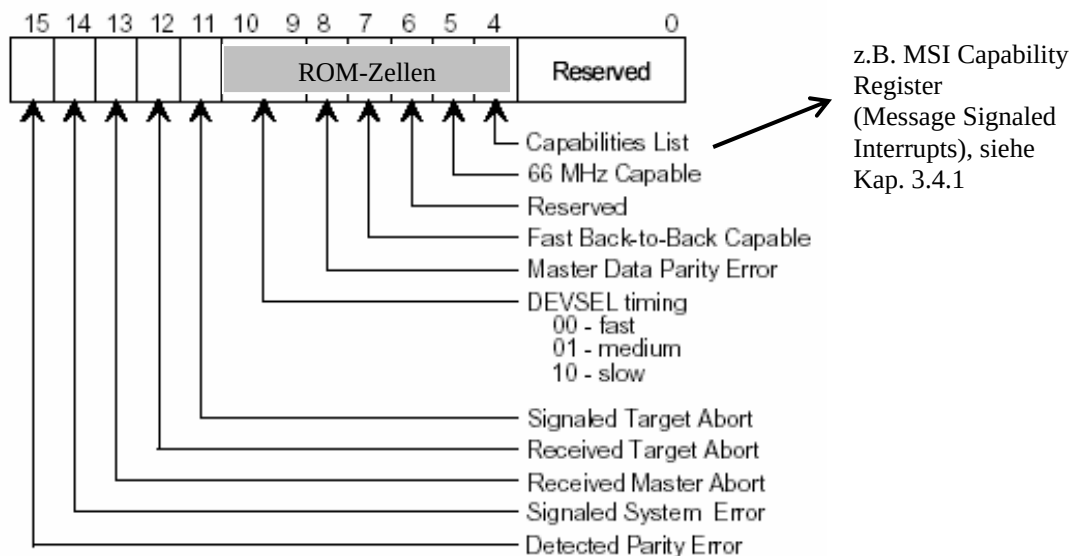


Bild: Status Register

Das Command Register legt das Verhalten des Device am Bus und dessen Eigenschaften fest.

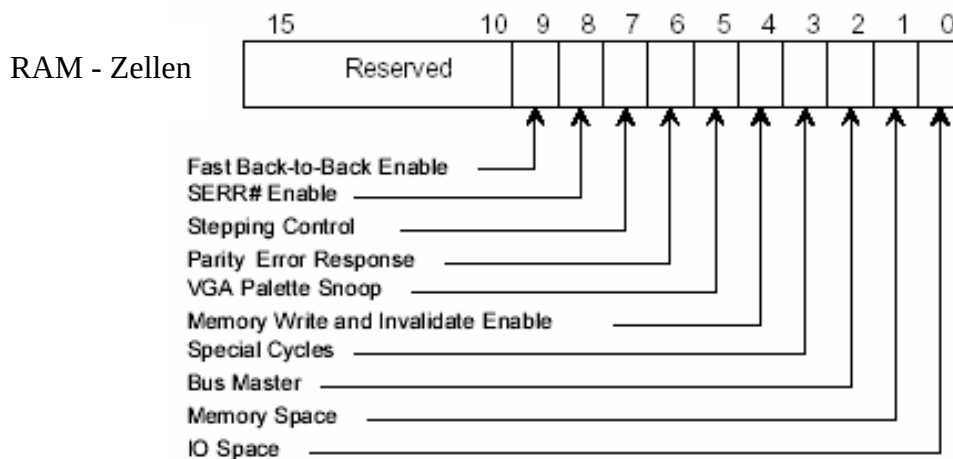


Bild: Command Register

Bei gesetztem Bit sind die im Bild genannten Merkmale vorhanden. Die Bits lassen Lesen und Schreiben zu.

(Nach dem Reset haben z.B. die Bits „Memory Space“ und „I/O Space“ den Wert 0, d.h., der Baustein reagiert nur auf Konfigurationszyklen).

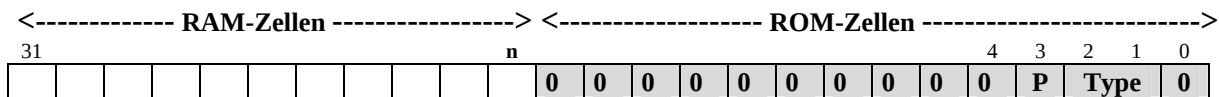
c) Abbildung der Speicherbereiche eines PCI-Device auf den Speicherbereich der CPU

Adreßräume eines PCI-Device:

Die vom PCI-Device benötigten Adreßräume sind in den 6 Feldern **Base Address 0..5 (Konfig.-Register)** eingetragen.

Die CPU ordnet die Adressräume der angeschlossenen PCI Devices so an, dass keine gegenseitige Überlappungen (wäre Mehrfachadressierung) stattfinden.

Format Base Address, Bit[31..0]:



Bit0 = 0 Speicheradressraum

= 1 I/O-Adressraum

Type=00: beliebige 32Bit-Adresse | **=01:** 32Bit-Adresse im ersten MByte

=10: beliebige 64Bit-Adresse | =11: reserved

n festgelegt durch Angabe der Speichergröße

Die im Feld "**Type**" vom PCI Device angegebene Adreßlage muß von der Konfigurations-Software beachtet werden, da der Adreßdekorator des PCI Device danach aufgebaut ist. Z.B. werden bei Type=01 nur die ersten 20 Adreßbit am Bus (AD19..0) dekodiert.

Das Feld "**P**" (prefetchable) gibt an, ob ein Speicherbereich wiederholt lesbar ist oder ob nach dem ersten Lesen die Information u.U. nicht mehr vorhanden ist (z.B. memory mapped I/O).

Die Konfigurations-SW ermittelt aus den Feldern "Base Address 0..5" die Speicher- und die I/O-Adressräume eines PCI-Device:

Schritte:

(1) CPU schreibt Base Address Register AD31..AD0 = "FFFFFFFF". Nur die RAM-Zellen werden verändert.



(2) CPU liest Base Address Register:

Im Register ist die **Speichergröße als 0-Folge** fest in den ROM-Zellen enthalten (Bit3 bis Bit0 sind ohne Bedeutung). Der Rest sind RAM-Zellen. Die erste "1" auf Bitposition >Bit3 liefert daher die Größe des Adreßraums.

z.B. Schreiben="FFFFFFFF", Lesen="FFF0000X" → PCI Device braucht 1MByte Speicher-Adressraum.

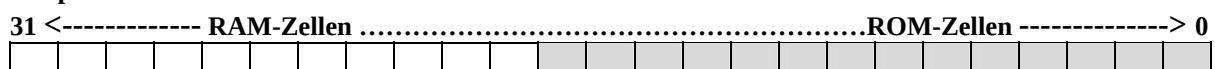
z.B. Schreiben="FFFFFFFF", Lesen="FFFF800X" → PCI Device braucht 32kByte

(3) CPU ordnet die Speicherbereiche aller PCI-Devices "nicht überlappend" an und vergibt die Startadressen.

(4) CPU schreibt Startadresse in die Base Address Register des Device. Diese steuern den Adressdeko- der des Device

Nach dem Lesen der Konfigurationsregister aller PCI-Devices am Bus vergibt die SW die Startadressen für die einzelnen Adreßräume ("**mapping**") durch Schreiben der Startadresse auf das Base Address Register. Da nur die RAM-Zellen beschreibbar sind muß eine Startadresse immer ein ganzzahliges Vielfaches der fest programmierten Speichergröße sein.

Beispiel: Startadresse=2¹⁹=512k



Frage: Was ist die minimal angebbare Speichergröße?

d) Konfigurationszyklen

Die CPU greift über die **HOST/PCI-Bridge** (North/South Bridge) auf den PCI-Bus zu, siehe **Bild "PCI - Architektur"**.

Für die Erzeugung der Konfigurationszyklen hält die Bridge zwei I/O-Ports bereit (I/O-Adressen im I/O-Adreßraum der CPU):

- 32 Bit **Configuration Address Port** (belegt die Adressen 0CF8..0CFB hex)
- 32 Bit **Configuration Data Port** (belegt die Adressen 0CFC..0CFF hex).

Schritt 1: Schreibzyklus

Die CPU schreibt die **Adresse (=Schreibdaten)** des gewünschten Konfigurationsregisters auf den I/O Configuration Address Port:

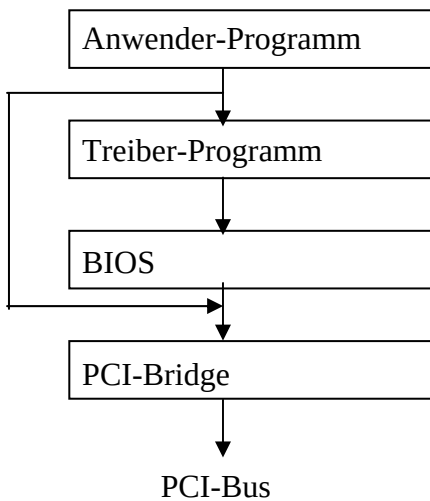
Adresse = PCI-Bus-Nummer (falls mehrere Busse)/
Device-Nummer/
Funktions-Nummer (innerhalb eines Devices)/
DW-Adresse innerhalb des Konfigurations-Registers.

Schritt 2: Lese- oder Schreibzyklus

Die CPU greift mit einem **I/O-Lese- oder Schreibzyklus** auf den Configuration **Data** Port der HOST/PCI-Bridge zu. Dies veranlaßt die HOST/PCI Bridge, einen "Konfigurationszyklus für Lesen" am PCI-Bus durchzuführen und die Lesedaten im Configuration Data Port für die CPU abzuspeichern (oder einen Schreibzyklus mit den im Configuration Data Port enthaltenen Daten am PCI-Bus durchzuführen).

Aufgabe: PCI-Bus

3.3.3. Software-Schnittstelle



C-Funktionen
Variable (Standardparameter-Übergabe)

BIOS-Funktionen
Register-Übergabe

2 I/O Ports für Übergabe von Zyklus-Adresse und Daten

PCI-Bridge:

Wie schon beschrieben, bietet die die Bridge der CPU für die Erzeugung der **Konfigurationszyklen** zwei I/O-Ports (I/O-Adressen im I/O-Adreßraum der CPU). Greift man auf diese Ports zu, generiert die Bridge die entsprechenden Konfigurationszyklen und übermittelt eventuelle Übergabewerte zum Prozessor.

Memory- und I/O-Zyklen der CPU in den Adressraum von PCI-Devices werden von der Host/PCI-Bridge über die Adresse erkannt und auf den PCI-Bus durchgeschaltet, siehe Kap. „Buskommunikation“.

BIOS:

Das **BIOS** (Ein/Ausgabe-Routinen des PC-Betriebssystems) stellt für den Zugriff auf den PCI-Bus **Funktionen** zur Verfügung, z.B.

<code>pci_bios_present</code>	Abfrage: PCI-Bus vorhanden
<code>find_pci_device</code>	Karte mit VENDOR-ID und DEVICE-ID gesucht
<code>read_configuration_area</code>	
<code>write_configuration_area</code>	usw.

Die Funktions-Nummer und sonstige Parameter werden in CPU-Registern übergeben. Alle BIOS-Funktionen werden über den Interrupt 1ah aufgerufen.

Treiber-Programm:

<code>init</code> -Funktion	initialisiert eine Karte am Bus
<code>config_read_byte/word/dword</code> -Funktion	
<code>config_write_byte/word/dword</code> -Funktion	usw.

(C-Funktionen)

3.3.4. Bus-Kommunikation

Die Kommunikation über den PCI-Bus wird über **Kommandos** eingeleitet. Ein Initiator informiert über ein Kommando andere am Bus angeschlossene Teilnehmer über den Typ des folgenden Datentransfers. Die Übertragung eines Kommandos erfolgt während der Adressierungsphase innerhalb eines Buszyklusses auf den Leitungen C/BE#[3..0]. Gleichzeitig mit dem Kommando überträgt der Initiator die Zieladresse für den folgenden Datentransfer auf den Leitungen AD[31..00], siehe Bild Lesezyklus.

Der Datentransfer kann aus einem einzigen Datenwort oder aus mehreren Datenworten bestehen (Burst-Mode). Im Burst-Mode ist das adressierte Target Device selbst für die Bereitstellung der jeweiligen Folgeadresse verantwortlich.

Zum Anschluß eines Target Device am PCI-Bus sind mindestens 47 Signale notwendig.

Ein Initiator (Master) braucht zwei zusätzliche Signale REQ# und GNT# für die Belegung (Arbitrierung) des Busses.

a) Bus-Kommandos:

Folgende Kommandos stehen einem Initiator zur Verfügung, wenn er einen PCI-Buszyklus durchführt (codiert in den Signalen **C/BE#[3..0]** während der Adressierungsphase). Durch das Kommando ist festgelegt, welcher Adreßraum (Speicher, I/O oder Konfigurationsreg.) mit der Adresse angesprochen wird.

C/BE#[3::0]	Command Type
0000	Das Interrupt Acknowledge Kommando zeigt einen Zugriff auf den Interruptcontroller. Dann ist der Zustand der Adressleitungen ohne Bedeutung.
0001	Special Cycle zeigt einen Broadcast an (geht an alle Teilnehmer)
0010	Bei IO Read wird von einem PCI Device, dessen Adressraum sich im I/O Bereich befindet, gelesen.
0011	IO Write wie IO Read jedoch mit schreibendem Zugriff.
010X	Reserved
0110	Memory Read leitet einen lesenden Zugriff auf ein Device ein.
0111	Memory Write wie Memory Read jedoch mit schreibendem Zugriff
100X	Reserved
1010	Configuration Read erzeugt einen lesenden Zugriff auf den Konfigurations- Speicher.
1011	Configuration Write erzeugt einen schreibenden Zugriff auf den Konfigurations-Speicher.
1100	Memory Read Multiple entspricht dem Memory Read, kündigt aber an, dass mehr als ein Datum gelesen wird.
1101	Dual Address Cycle wird zu Übertragung einer 64-Bit Adresse verwendet.
1110	Bei Memory Read Line wird eine komplette Cache Zeile übertragen.
1111	Memory Write and Invalidate entspricht dem Memory Write. Es wird jedoch eine komplette Cache Zeile übertragen.

b) Bustransfers (PCI Transaction Models):

Darstellung der Schreib- und Lesezyklen im Zeitdiagramm siehe Unterkapitel e).

„Programmed I/O“:

CPU $\leftarrow$ Interrupt PCI-Device:	Meldung „Daten abholen“
CPU $\leftarrow$ PCI-Device:	Lesezyklus der CPU (Daten in CPU-Register)
CPU $\rightarrow$ Arbeitsspeicher:	Schreibzyklus der CPU (Daten in Arbeitsspeicher)

Vor/Nachteile: Bus-Master =

„DMA (Direct Memory Access)“:

PCI-Device $\rightarrow$ Arbeitsspeicher Schreibzyklus des PCI-Device
(genauer: PCI-Device $\rightarrow$ North Bridge: Schreibzyklus des PCI-Device (in Bridge-Register),
von der North Bridge über die Adresse erkannt.
North Bridge $\rightarrow$ Arbeitsspeicher: Schreibzyklus der North Bridge auf Speicherbus)

CPU $\leftarrow$ Interrupt PCI-Device: Meldung „Transfer beendet“

Vor/Nachteile: belastet CPU nicht (Datentransfer ohne CPU)

Bus-Master = PCI-Device

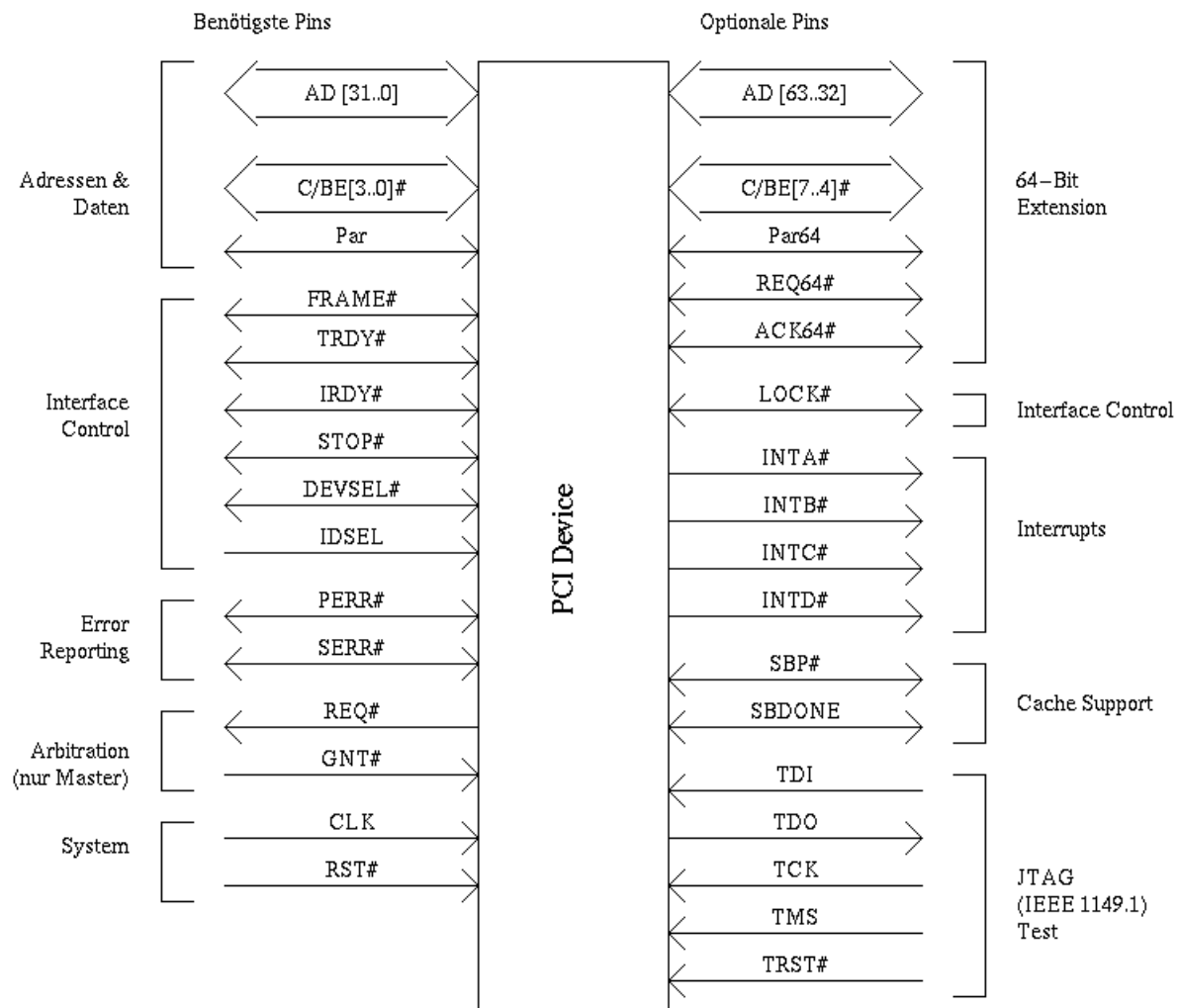
„Device-Device (Peer to Peer)“:

PCI-Device1 $\rightarrow$ PCI-Device2: Schreibzyklus von PCI-Device1 mit Speicheradresse von
PCI-Device2

Vor/Nachteile: nur 1 Zyklus, belastet CPU nicht

Bus-Master = PCI-Device1

c) Signale:



FRAME Initiator Start Signal
TREADY Lesedaten gültig oder Schreibdaten übernommen
IRDY bereit, Lesedaten zu übernehmen oder
STOP Target signalisiert dem Initiator, dass er die Transaktion beenden soll
DEVSEL Target signalisiert mit DEVSEL, dass es seine Adresse auf dem Bus erkannt hat

Beim Target Device (Slave) drehen sich die Richtungen folgender Signale um:

C/BE#[3..0], FRAME#(nur Eingang), TRDY#, IRDY#, STOP#, DEVSEL#, PERR#(nur Ausgang)

Beim Target Device (Slave) entfallen:

REQ#, GNT#

Zusätzlich sind für Target und Initiator noch optional:

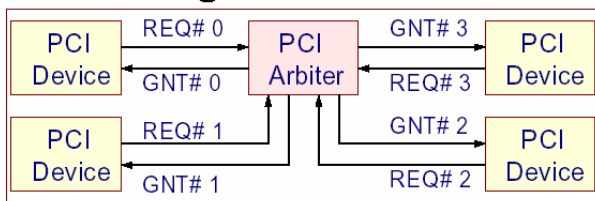
JTAG-Signale, Signale für Erweiterung auf 64Bit-Bus usw.

Fragen:

Wieso sind 2 Ready-Signale IRDY# und TRDY# vorhanden ?

Erklärung und Vergleich mit uP-Bus.

d) Bus-Belegung (Arbitrierung):



PCI-Arbiter: Bussteuerung in der PCI-Bridge
PCI-Device: z.B. PCI-Board im PCI-Slot (Steckplatz)

Ein Initiator fordert mit dem Signal **REQ#** den Bus an und bekommt ihn, falls er frei ist, mit dem Signal **GNT#** von der Bus-Arbiter-Schaltung zugeteilt. Der Bus ist frei (**IDLE state**), wenn die Signale **FRAME#** und **IRDY#** "nicht aktiv" sind. Jeder Initiator hat eine eigene REQ#- und GNT#-Leitung zum Arbiter. Der **Bus-Arbiter** ist Bestandteil der **Host/PCI-Bridge** (der Arbiter sollte programmierbar sein hinsichtlich der Teilnehmer-Priorität).

Eine Busbelegung kann beendet werden durch den Initiator (Wegnahme des Signals FRAME#) oder das Target (Signal STOP# an Initiator)

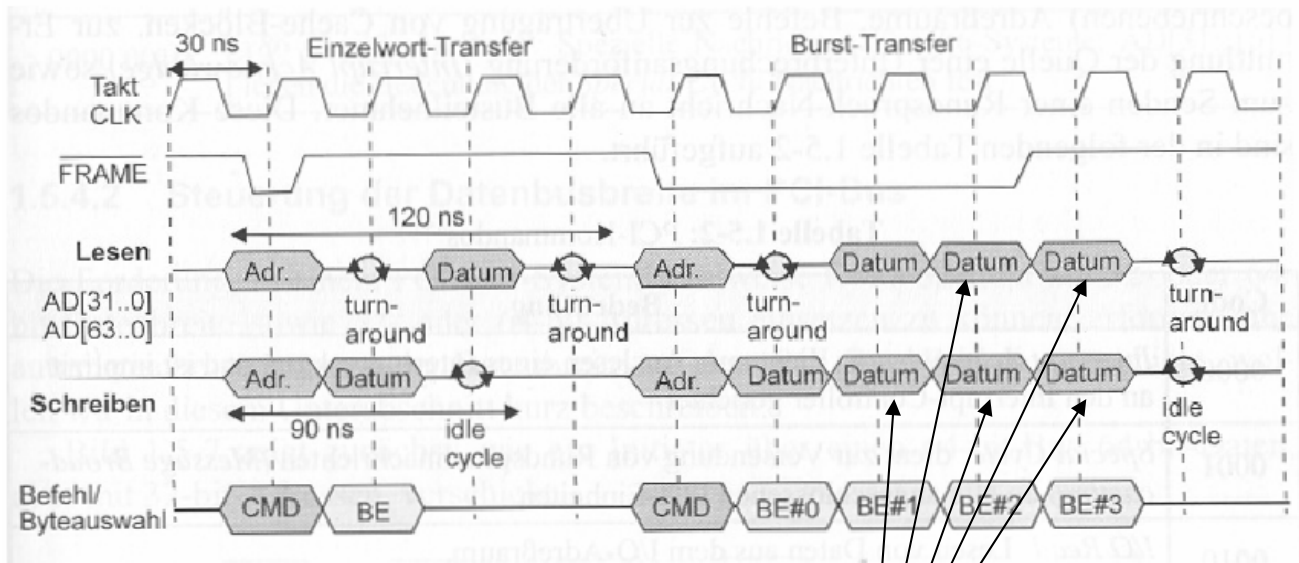
Das Konfigurationsregister des Initiators enthält eine maximale Busbelegungszeit (master latency time).

Zeitdiagramm: Pegel-Darstellung

Die Bussignale sind low-aktiv (#): Signal = 0 V → Signal aktiv

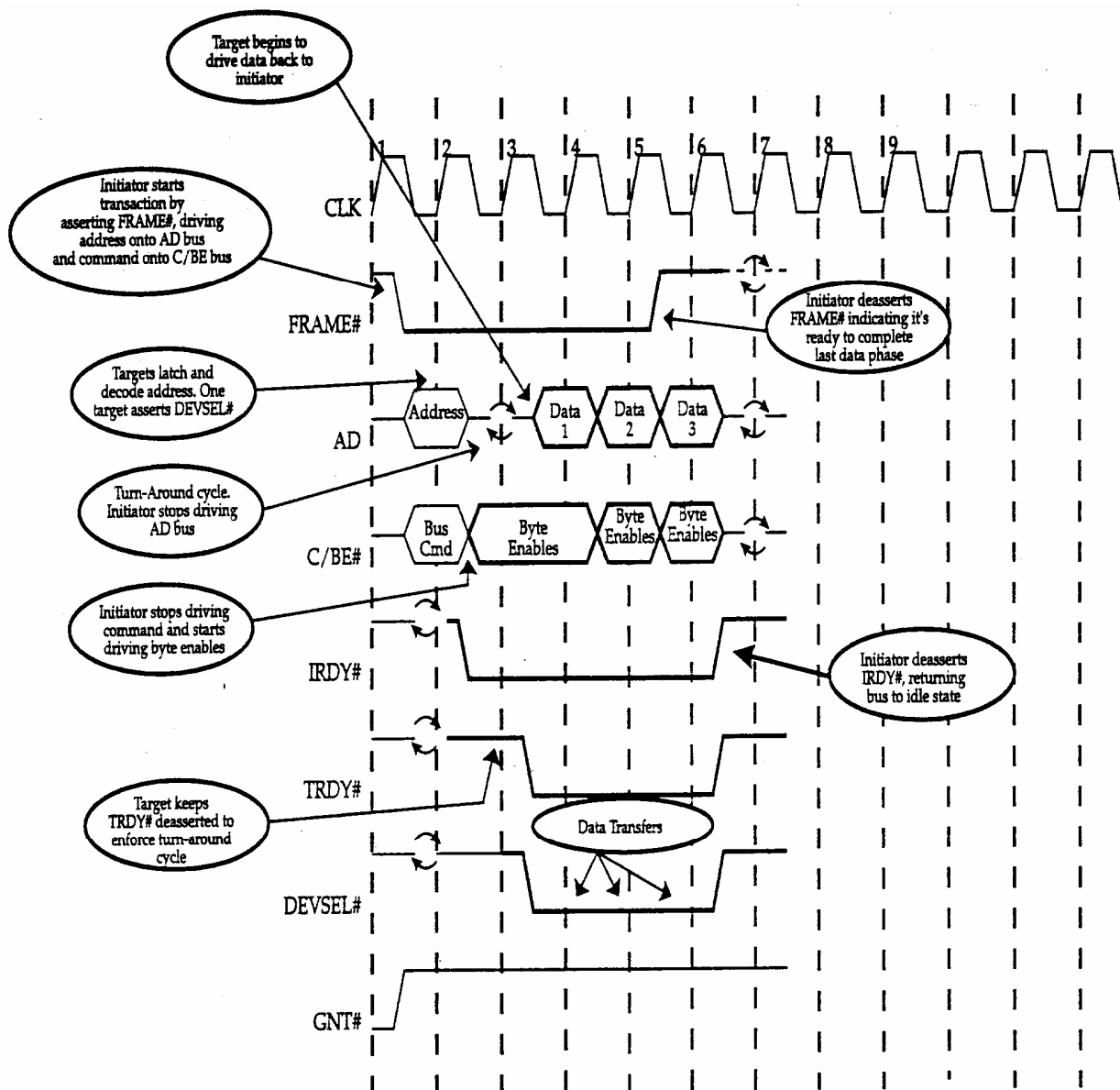
Bussignale	Aktivität Initiator	Aktivität Target
FRAME# =1 IRDY# =1	Bus frei (Idle state)	
GNT# =0	Bus wird dem Initiator zugeteilt (vom Arbiter)	
FRAME# =0 AD[31..2] C/BE#[3..0]	T1: Initiator startet Buszyklus durch Anlegen von FRAME# -Signal (=Start), Adresse und Kommando	Target speichert und dekodiert: Adresse und Kommando
C/BE#[3..0] IRDY# =0 TRDY#=1	T2: Initiator meldet über "Byte Enable" welche Bytes gültig sind Initiator meldet: bin bereit mit IRDY#	Target erzwingt "turn around cycle" für Kontrolle des AD[31..0]-Bus (bidirektional)
DEVSEL# =0 AD[31..0] TRDY# =0	T3:	Target meldet "bin ausgewählt" legt Daten auf den Bus meldet "Daten gültig"
	T4: Initiator usw.	Target inkrementiert die Adresse für die nächsten Daten (Burst) usw.
FRAME# =1	T5: Initiator zeigt dem Target die letzte Datenphase an	
IRDY#=1	T6: Initiator gibt Bus frei	

e) Datenzyklen: vereinfacht (nicht alle Signale), Pegeldarstellung
Lesen, Schreiben, Einzel- und Burst-Transfer
 Schaltzeiten für 33MHz



Target generiert Folgeadressen selbst

Lesezyklus (alle Signale): **Pegeldarstellung** (# = low-aktives Signal)



Frage:

IRDY# = 1 → ?

TRDY# = 1 → ?

3.3.5. Hardware

Bus-Takt:

Alle Aktivitäten auf dem Bus sind synchron zum Bustaktsignal PCI CLK (spezifiziert mit maximal 33MHz, maximal 66MHz usw.). Steckkarten, die z.B. für die Timer-Funktion eine genaue Taktfrequenz benötigen und die in verschiedenen Computern zu Einsatz kommen können, sollten deshalb wegen der nicht genau festgelegten Taktfrequenz eine vom PCI-Takt unabhängige Taktquelle besitzen.

Zahl PCI-Einbauplätze (Slots) am Bus:

Bei 33MHz und 32 Bit Busbreite sind max. 10 elektrische Lasten („loads“) anschließbar. Eine PCI-Einsteckkarte zählt 2 Lasten (Steckverbinder = 1 Last, Gatter-Ein/Ausgang = 1 Last). Da der Chipsatz (Bridge Host/PCI) auf dem Motherboard auch eine Last darstellt, sind normalerweise nur 4 Einbauplätze am PCI-Bus verfügbar. Sind mehr Einbauplätze notwendig, ist der PCI-Bus (BUS 0) mit einer Bridge zu erweitern (Bus 1 ..).

Datentransfers innerhalb von PCI-Bus 1 können gleichzeitig zu Datentransfers innerhalb von Bus 0 ablaufen.

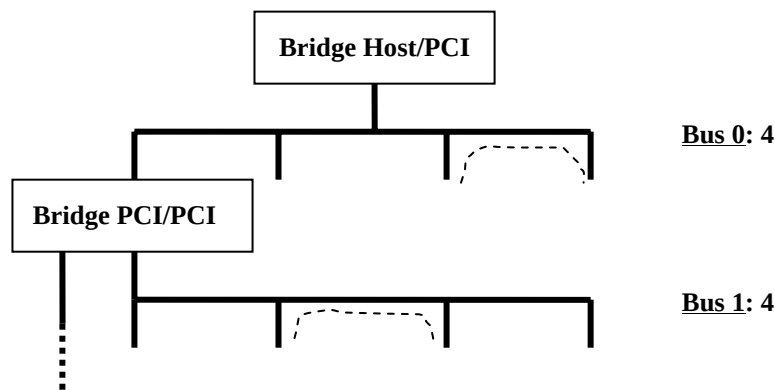


Bild: Kaskadieren von PCI-Bussen

Compact PCI:

Der Bus ist für verschiedene Steckkarten und Stecker spezifiziert, z.B. für Europakartenformat.

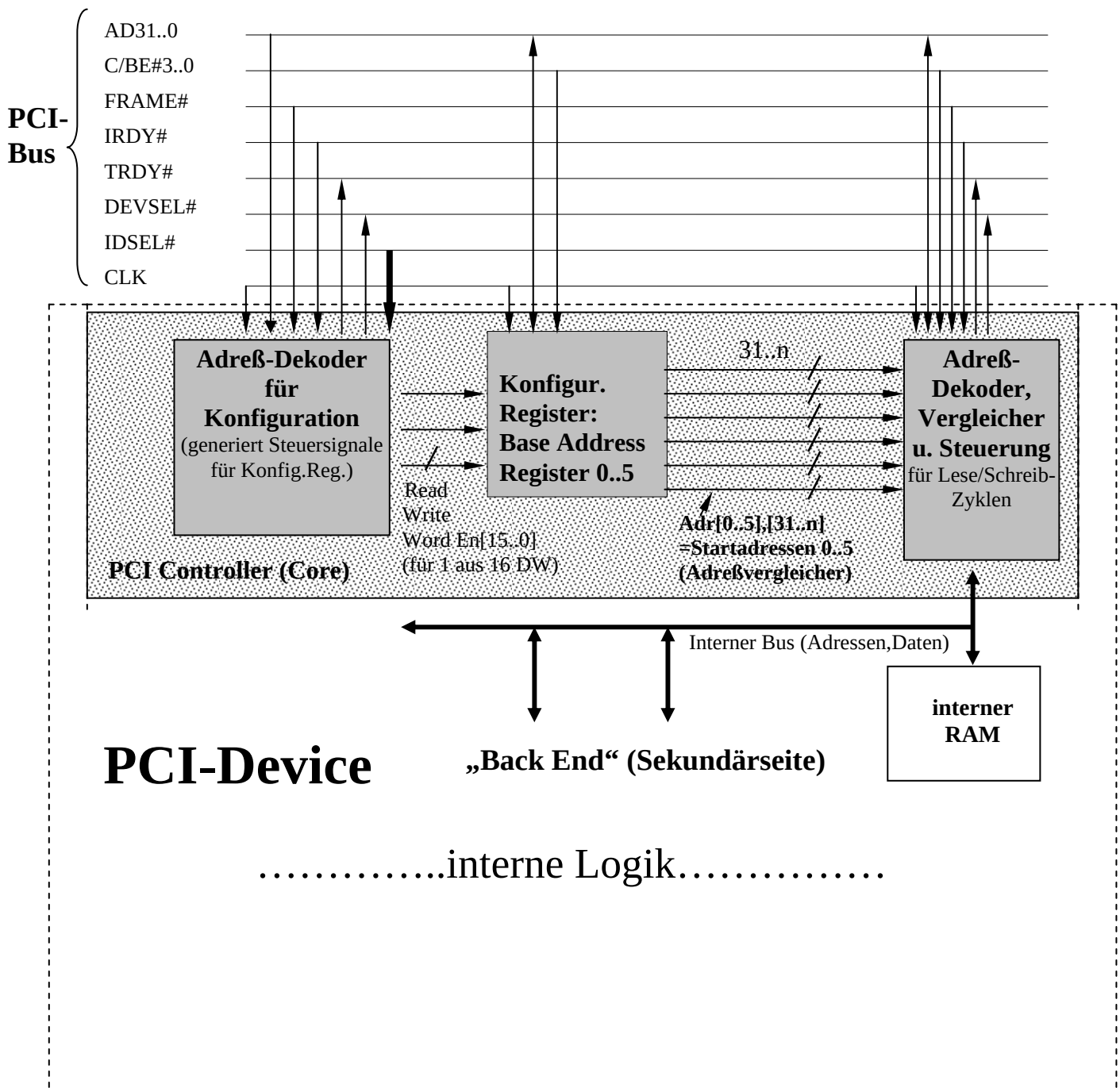
Weitere Themen:

Interrupt-Funktion / Hardware: kein Leitungsabschluß (Unterschied zu SCSI-Bus) / Adressen sind in AD31..2 enthalten.
(=Grenze eines Doublewords=4Byte)

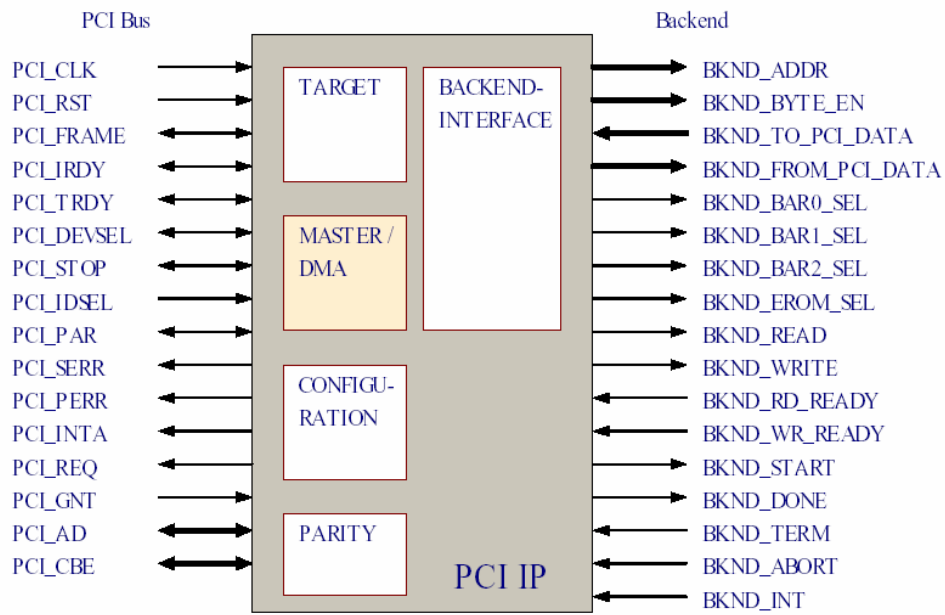
Realisierung:

Bei der Entwicklung einer PCI-Karte kommen derzeit zwei Methoden zur Anwendung:

- bei kleinen Stückzahlen werden PCI-Controller-Chips verwendet, die auf der Primär-Seite das PCI-Bus-Protokoll abwickeln und auf der Sekundär-Seite Signale (Address-, Daten-, Dekoder- und Steuersignale) für die Anwenderfunktionen zur Verfügung stellen.
- bei größeren Stückzahlen wird die PCI-Bus-Schnittstelle in programmierbaren Logikbausteinen wie z.B. FPGAs implementiert. Dazu existieren z.B. VHDL-Makros (PCI-Cores) für die PCI-Schnittstellenfunktionen, die normalerweise nicht kostenfrei sind.



Darstellung am Beispiel eines PCI-Core (Fa. IMG)



IP = Intellectual Property

Vorteile der Backend-Schnittstelle für den Anwender:

Tabelle „Ihr Wissen über den PCI-Bus“ am Kapitel-Anfang ergänzen/korrigieren!

3.4. PCI Express

Der PCI-Bus stellt das „Rückgrat“ im PC und in anderen Computern dar. Die steigenden Anforderungen an den Bus wie höhere Datenrate, begrenzte Pinzahl für den Bus, Datentransfer in Realzeit für isochrone Daten, Power Management (Umschalten eines PCI-Device in Power-Down-Mode) usw. sind mit den bisherigen Konzepten von PCI Conventional und auch von PCI-X nicht oder nur teilweise erreichbar. Deshalb wurde mit dem PCI Express ein neues Konzept entwickelt, das die obigen Forderungen für die nächste Zukunft erfüllen soll.

	Anforderung	PCI Conventional, PCI-X	PCI Express
Datenrate	hoch, aber skalierbar (wegen Pinzahl) Festplatte, Gigabit-LAN, Grafikkarte usw.	PCI Conv: 132 MByte/s PCI-X: 1 GByte/s bei 64 Bit, 133 MHz	PCIe x1: 250 MByte/s bis PCIe x32: 8 GByte/s (auf Basis von 2,5 GBit/s Übertragungsrate)
Pinzahl (Kabel)	möglichst klein	PCI Conv: ca. 80 Pins (49 Sign.+GNDs+Vcc) PCI-X: ca. 110 Pins	PCIe x1: 8 Pins (4 Sign. +GNDs+Vcc) PCIe x8 : 40 Pins (32 Sign. +GNDs+Vcc) angepasst an Datenrate
Datentransfer in Realzeit	keine Verzögerung	alle PCI-Devices teilen sich die Busbandbreite → kurzzeitige Verzögerungen	Punkt-zu-Punkt-Verbindung, für Hin- und Rückrichtung getrennt → keine Verzögerungen
Power Management	reduzierte Stromaufnahme falls kein Datentransfer		

Der PCI Express ist software-kompatibel zum PCI Conventional (Spec. 2.3). Die Anwenderprogramme müssen nicht geändert werden.

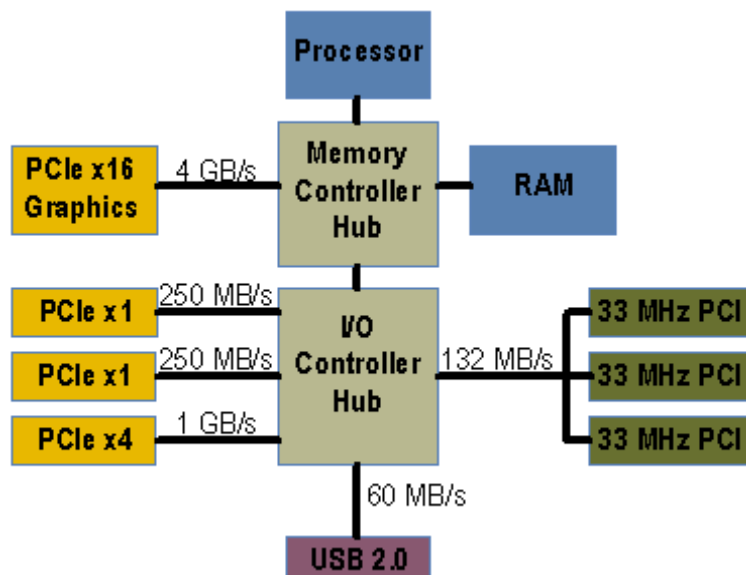


Bild: Busstruktur einer CPU (PC) betreffend PCI-Steckplätze (Kombination PCI Express und PCI Conventional)

Jeder PCI-Express-Steckplatz verfügt über die angegebene Busbandbreite zum RAM-Arbeitsspeicher wogegen sich die 3 PCI-Conventional-Steckplätze (33MHz PCI) die Bandbreite von 132 MB/s des Busses „teilen“ müssen.

3.4.1. Physikalische Ebene

Im Gegensatz zu den Parallelbussen PCI Conventional und PCI-X ist der PCI Express ein serieller Bus mit differentieller Datenübertragung. Im Minimalfall PCIe x1 (250MByte/s) besteht der Bus aus nur 4 Signaladern. Er ist entsprechend den Anforderungen modular erweiterbar (skalierbar) auf PCIe x2/ x4/x8/ x12/ x16 und x32.

Damit bei einer Datenübertragung auf nur wenigen Adernpaaren ein hoher Datentransfer zustande kommt, ist eine hohe Datenrate (Übertragungsfrequenz) notwendig. Mit den bei den Gigahertz-Prozessorboards gemachten Erfahrungen sind jetzt Übertragungsraten pro Signal von 2,5 GBit/s und mehr möglich.

Weitere Signale wie z.B. Interrupt- Fehler-, Buszuteilungs- und Fehlersignale sind nicht vorhanden. Diese Funktionen werden von Nachrichten-Paketen (Messages) übernommen.

Beispiel Interrupt:

Da Interrupt-Signale nicht vorhanden sind erfolgt die Signalisierung über Message Signaled Interrupts MSI. Ein MSI ist eine Schreib-Transaktion. Sie unterscheidet sich von einer „normalen“ Schreib-Tansaktion (Memory Write Transaction) nur durch die Ziel-Adresse (Target Memory Address, reserviert vom System für Interrupt-Meldungen) und den Datencode. Beide werden bei der Konfiguration vom Host in das MSI Capability Register geschrieben, siehe auch Kap.3.3.2 b) Konfiguration, Stausregister, Capability List.

Signale:

Nur differentielle Datensignale (D+, D-).

Lane = 2 Signalpaare: je ein differentielles Signalpaar (D+, D-) für jede Übertragungsrichtung

2,5 GBit/s pro Übertragungsrichtung ergibt $2,5 \text{ GBit/s} * 8/10 * 1/8 = 250 \text{ MByte/s}$

(8/10 wegen 8Bit $\rightarrow$ 10 Bit Codierung, siehe auch Kapitel Serial Attached SCSI)

(5 und 10 GBit/s sind in Entwicklung)

Link = physikalische Verbindung zwischen 2 PCI-Devices, bestehend aus ≥ 1 Lane.

Da der Bus eine Punkt-zu-Punkt-Verbindung ist, gehört die Datenrate eines Links

ausschließlich zu dieser Verbindung und muss nicht mit anderen Geräten geteilt werden wie dies beim PCI Conventional der Fall ist.

Beispiele für PCI Express Links:

PCIe x1 = 250 MByte/s

max. PCIe x32 = 8 GByte/s

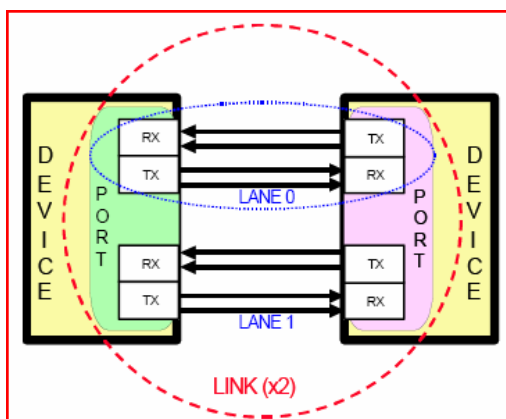


Bild:

Link PCIe x2 = Lane 0 + Lane 1

Besteht ein Link aus mehreren Lanes (x2, x4, x8 usw.), dann werden die zu übertragenden **Daten byte-weise auf die Lanes aufgeteilt**.

hier:

Lane 0: Byte 0, Byte 2, Byte 4, usw.

Lane 1: Byte 1, Byte 3, Byte 5, usw.

eines Datenblocks

Switch (Schalter), PCI Express Fabric (PCI Netzwerk):

verbindet zwei PCI-Devices miteinander (im Bild von Kap. 3.4 im Controller Hub enthalten)

PCI-Devices mit unterschiedlicher Lane-Zahl, z.B. PCIe x2 und PCIe x4, können miteinander verbunden werden auf Basis der kleineren Lane-Zahl (hier PCIe x2).

Power Management:

Es sind 4 Zustände Full on/ Light Sleep/ Deep Sleep und Full Off definiert (Zustandsübergänge: 200 us bis 10 ms).

Power Budgeting:

Bei der Konfiguration wird erkannt ob ein PCI Device hinsichtlich Stromaufnahme (und Kühlung) anschließbar ist.

Signalpegel:

Stecker mit Signalbelegung:.....

3.4.2. Adressierung

Es existieren wie bei PCI Conventional eigene Adressräume für Konfiguration, für Memory (4 GByte bei 32 Bit Adressierung bzw. 16 Exabyte bei 64 Bit Adressierung) und für I/O.

Der Konfigurationsraum wurde von 256 Byte auf 4 kByte erweitert. Die ersten 256 Byte des Konfigurationsregisters sind identisch zu PCI Conventional.

Zusätzlich gibt es einen Adressraum für Messages (z.B. Interruptsignalisierung, Fehlersignalisierung, usw.).

3.4.3. Konfiguration

Die Konfiguration ist kompatibel zum PCI Conventional (PCI-compatible Configuration Mechanism). Sie verwendet dieselben Ports (Configuration Address Port, Configuration Data Port) und dieselbe Adressierung. Daneben existiert ein PCI Express Enhanced Configuration Mechanism.

3.4.4. Architektur und Buskommunikation

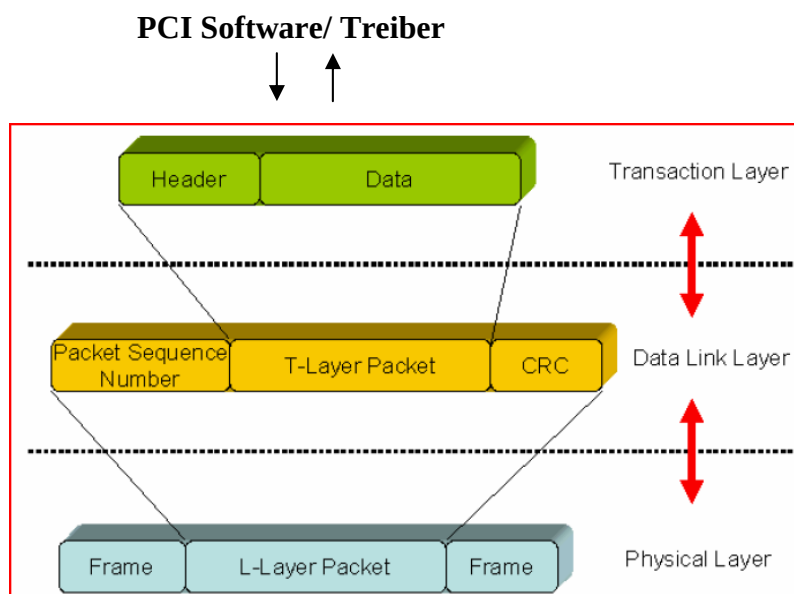


Bild:

Schichtenmodell und Datenpakete

Die PCI Express Spezifikation definiert ein Schichtenmodell für die Entwicklung von PCI Devices

Die Schichten werden im PCI Device beim **Senden** von **Requests** von oben nach unten und beim **Empfangen** von **Requests** von unten nach oben durchlaufen.

Transaction Layer:

Erhält Lese- und Schreib Anforderungen (Requests) von der Software und generiert daraus sog. Request Packets (Header+Data) für die Busübertragung

Data Link Layer:

Fügt eine fortlaufende Packetnummer und eine CRC-Fehlersicherung hinzu.

Physical Layer:

Fügt eine Start-Kennung (Frame-Bitmuster links) und eine Ende-Kennung (Frame-Bitmuster rechts) hinzu.

Header-Feld (max. 16 Byte):

Die Inhalte hängen von der Transaktionsart (Memory-/ IO-/ Configuration-/ Message- oder Completion Request) ab und sind sehr vielfältig. Einige wichtige Inhalte sind:

- PCI Express Command
- Transaction Type (Read/ Write)
- Requester-ID (Bus/ Device/ Function Nummer des sendenden PCI Device)
- Completer-ID (Bus/ Device/ Function Nummer des empfangenden PCI Device)
- Transferclass TC des Pakets (Priorität beim Durchschalten des Pakets zum empfangenden PCI Device)
- Länge des nachfolgenden Datenfelds
- Paket-Nummer (dieses 8Bit-Tag wird benötigt zur Identifizierung ausstehender Pakete)
- alternativ :
- Memory-Adresse
- usw.

Data-Feld (max. 1024 Byte): Enthält die Daten.

Transaction:

Umfasst ein oder mehrere Pakete, um einen Informationstransfer durchzuführen.

Split Transactions, Completions:

Wie schon beim PCI-X können Transaktionen über den Bus sog. Split-Transaktionen sein, d.h., bei einer Anforderung (Request), die vom Empfänger nicht sofort bearbeitbar ist, wird vom Empfänger nur der Erhalt der Anforderung mit einem Completion-Paket quittiert. Das Ergebnis der Anforderung (z.B. Lesedaten) wird erst später gesendet. Dadurch werden Verbindungswege und Datenpuffer nicht unnötigerweise blockiert.

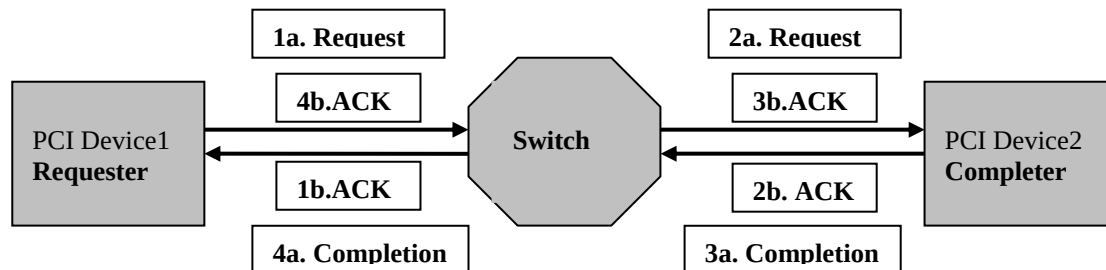


Bild: Split Transaction am Bspl. eines Memory Read (Device1 liest aus Speicher von Device2)

Reihenfolge: 1a→1b→2a→2b→3a→3b→4a→4b

1a: Requester sendet Memory Read Packet MRd. Switch speichert MRd und überprüft CRC (LCRC)

1b: Switch quittiert mit ACK. Requester löscht Packet MRd aus seinem VC Buffer.

2a: Switch leitet Packet MRd auf grund der Memory-Adresse an den richtigen Port weiter. Completer erhält (speichert) MRd und überprüft CRC (LCRC).

2b: Completer quittiert mit ACK. Switch löscht Packet MRd aus seinem VC Buffer.

3a: Completer überprüft CRC (ECRC) und sendet ein Completion Packet CplD mit den Lesedaten. Switch speichert Packet CplD und überprüft CRC (LCRC).

3b: Switch quittiert mit ACK. Completer löscht Packet CplD aus seinem VC Buffer.

4a: Switch dekodiert Requester ID in CplD und leitet das Paket an den richtigen Port weiter. Requester erhält (speichert) CplD und überprüft CRC (LCRC).

4b: Requester quittiert mit ACK. Switch löscht Packet CplD aus seinem VC Buffer. Requester überprüft CRC (ECRC) in CplD. Requester vergleicht 8Bit-Tag des erhaltenen CplD Pakets mit dem 8Bit-Tag des ursprünglichen MRd Pakets.

Traffic Class TC (0 bis 7):

Legt die Priorität eines Pakets beim Durchschalten durch das PCI-Netzwerk (PCI Switch, PCI Express Fabric) fest. TC7 ist die höchste Priorität. Ein hochprioriges Durchschalten (TC=7) ist besonders wichtig bei Interrupt-Messages und bei isochronen Daten (z.B. Daten von einer Filmkamera, die eine garantierte Bandbreite brauchen).

Normale Dateitransfers von/zum SCSI-Controller (Festplatte, DVD) besitzen in der Regel TC=0.

Die Traffic Class TC wird jedem Paket von der Anwendersoftware oder der Treibersoftware zugewiesen.

Virtual Channel VC:

Die momentane Verbindung vom Datenpuffer des Senders (Requester) über die Datenpuffer des PCI-Netzwerks (PCI Switch, PCI Express Fabric) bis zum Datenpuffer des Empfängers (Completer) wird auch als Virtual Channel VC bezeichnet. Die im PCI Device und im PCI-Netzwerk enthaltenen Datenpuffer werden als **VC Buffer** bezeichnet. Es sind dies in einem PCI Device maximal 8 (VC Buffer0 bis VC Buffer7). Da im PCI-Netzwerk mehrere VC Buffer vorhanden sind können gleichzeitig mehrere Verbindungen (Virtual Channels VCs) mit unterschiedlicher Priorität geschaltet sein.

TC/VC-Mapping:

Jedem VC Buffer ist ein Wert für eine Traffic Class TC zugeordnet. Datenpakete werden entsprechend ihrer TC# in VC Buffern mit derselben TC# zwischengespeichert (→ Virtual Channels mit unterschiedlichen Prioritäten).

Flow Control:

Ein Paket mit Schreibdaten wird im VC-Datenpuffer des empfangenden PCI-Device abgespeichert. Der Empfänger versorgt den Sender periodisch mit Informationen, wie viel freie Pufferkapazität er zur Verfügung hat (Flow Control Mechanism Protocol). Somit ist sichergestellt, dass kein Transfer wegen zu kleiner Pufferkapazität wiederholt werden muss.

Switch mit Port:

Über Multi-Port-Switches können PCI-Geräte direkt miteinander kommunizieren. Pro Port ist ein PCI-Device anschließbar, das maximal in 8 Funktions („Teilgeräte“) unterteilt sein kann. Eine Funktion muss function# = 0 besitzen. Ein Switch verwendet die prioritätsgesteuerte Arbitrierung.

Ingress Port: Port, der ein Paket empfängt.

Egress Port: Port, der ein Paket sendet.

Hot Plug/ Unplug:

Kommandos:

Es sind prinzipiell die gleichen Memory-, I/O- und Configuration-Kommandos verfügbar wie beim PCI Conventional, siehe die dortige Tabelle. Zusätzlich gibt es Message-Request- und Completion-Kommandos. Completion-Kommandos steuern die Split Transactions, Message-Request-Kommandos übernehmen die Funktionen von (beim PCI Express fehlenden) Signalen wie Interrupt-, Fehler-, Buszuteilungs- oder Ready-Signal usw.

Die Kommando-Codes sind im Header eines Request Packets enthalten.

3.5. Aufgaben

Aufgabe PCI-1: PCI-Bus, Varianten

Bitte nur kurze Antworten (Stichpunkte)

Neben dem bisherigen „PCI Conventional“ gibt es 2 neuere PCI-Bus-Varianten.
Nennen Sie die beiden Bezeichnungen.
Vergleichen Sie beide mit dem PCI Conventional bei den wichtigen Merkmalen
--Buskonzept(u.a. parallel/seriell),
--Busbreite (Signaladern) und
--Taktrate.

Aufgabe PCI-2: PCI-Bus, Konfiguration

1. Erklären Sie den Begriff "Bus-Bandbreite"! Welche Werte bietet der PCI-Bus?
2. Was ist ein "Target-Device" ?
3. *Beim Konfigurieren des PCI-Busses wird von der CPU über das Konfigurationsregister (Feld "Base Address") zuerst die Größe des Speicherbereichs eines PCI-Device ermittelt und dann die Beginnadresse festgelegt ("Mapping"). Beantworten Sie dazu folgende Fragen:*
 - a) Welche Speichergröße liegt vor unter den Annahmen:
Schreiben "Base Address" AD31..AD0 = FFFFFFFF
Lesen "Base Address" AD31..AD0 = FFFF00X
 - b) Welche Beginnadressen für den obigen Speicherbereich kann die CPU vergeben ?

Aufgabe PCI-3: PCI-Bus, Konfiguration

Gegeben ist ein PCI-Board mit 64 kByte Datenpuffer.

Im 32Bit-Konfigurationsregister (Feld: Base Address) steht nach der Konfiguration: FC400004 hex.

- a) Welche Bitpositionen (31 .. 00) sind als ROM-Zellen realisiert?
- b) Wie ermittelt der Host beim Zugriff auf das Konfigurationsregister (Feld: Base Address) die Größe des geforderten Speicherbereichs? (Antwort stichpunktartig).
- c) Welche Beginnadresse (Hex-Wert) wurde dem Speicherbereich zugewiesen?
- d) Was wäre die kleinstmögliche von 0 verschiedene Beginnadresse (Hex-Wert) bei obiger Konfiguration?

Aufgabe PCI-4: PCI Express

1. Nennen Sie 3 wichtige technische Vorteile des PCI Express Bus (gegenüber dem PCI Conventional):
2. Erklären Sie die Bezeichnung PCIe x2! Wie ist der Bus realisiert? Wie viele Signaladern sind vorhanden?

Aufgabe PCI-5: PCI-Bus, Adressräume

1. Was ermöglicht es dem Host bei der Konfiguration, die Steckplätze einzeln abzufragen? Sind dadurch mehr Bussignale notwendig?
2. Welche Adressräume gibt es am PCI-Bus und woran erkennt ein Target diese unterschiedlichen Adressräume bei einem Buszyklus?
3. Im Unterschied zu einem "normalen" Mikroprozessorzyklus gibt es beim PCI-Zyklus zwei Ready-Signale: TRDY# (Target-Ready) und IRDY# (Initiator-Ready)
Erklären Sie die Funktion der Ready-Signale am Beispiel eines Leszyklus. Wieso ist ein Initiator-Ready notwendig?
4. Was verbirgt sich hinter der Bezeichnung PCISIG?

4. SCSI-Bus (Small Computer Systems Interface)

(gesprochen: "Skuzzy"-Bus)

www.scsita.org, www.google (SCSI-2 oder SCSI-3) |

4.1. Einführung

Das Konzept der SCSI-Schnittstelle ist mehr als „nur“ eine Geräteschnittstelle. Sie erlaubt den Aufbau eines kompletten Peripherie-Systems (= Anschluß vieler Geräte am Computer).

Inzwischen wurde auch der Standard für einen seriellen SCSI-Bus verabschiedet.

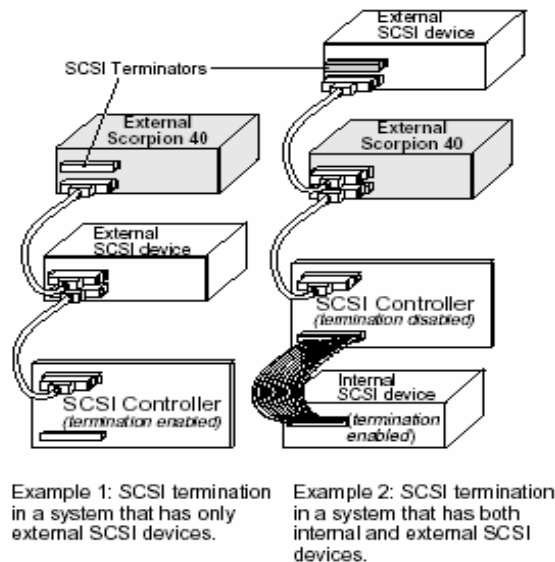


Bild:
Anschluß von SCSI-
Geräten am Host
(SCSI-2-Controller)

Der SCSI-Bus als reine Geräteschnittstelle wurde vielfach von der einfacheren und daher billigeren Geräteschnittstelle IDE/EIDE/ATA bei Geräten wie Festplatte, DVD-Laufwerken usw. verdrängt.

Mit Verabschiedung der Norm SCSI-3 nimmt das SCSI-Konzept eine zentrale Rolle im Bereich der Rechnerperipherie ein, da die SCSI-Befehlsebene von vielen Schnittstellen übernommen wurde, siehe Bild „SCSI Architecture“.

SCSI-Befehle werden nicht nur über den SCSI-Bus sondern auch über viele andere Schnittstellen transportiert.

Normen:

SCSI-1: 1986 ANSI-X3.131-Norm für die SCSI-Parallelschnittstelle

SCSI-2: 1993 Weiterentwicklung von SCSI-1

SCSI-3: 2003 „Multiple Standard“

Die SCSI-Befehlssätze gelten für viele andere Schnittstellen. SCSI-3 fächert in verschiedene Standards auf wie SCSI-3 parallel, SCSI-3 serial, iSCSI usw.

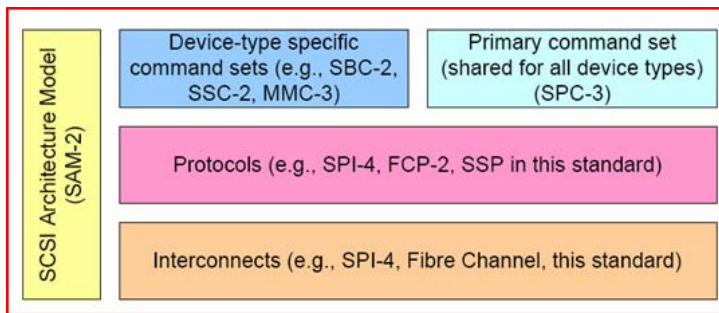
Die Norm trennt (unterteilt) SCSI-3 in Standards für:

- Command Sets (Befehlssätze)
- Transport Protocols
- Physical Interfaces

z.B. iSCSI: Standard zur Übertragung der SCSI-Befehle über TCP/IP-Protokoll; er soll die (preiswerte) Gigabit-Ethernet-Technik für den Transport von Speicherdaten anwendbar machen.

- Befehlsebene: SCSI
- physikalische Verbindung: Ethernet

4.2. SCSI-3 Architektur



SAM: scsi architecture model
SBC: scsi block commands
SSC: scsi streaming commands
MMC: multi media commands
SPC: scsi primary commands
SPI: scsi parallel interface
FCP: fibre channel protocol
SSP: scsi serial protocol

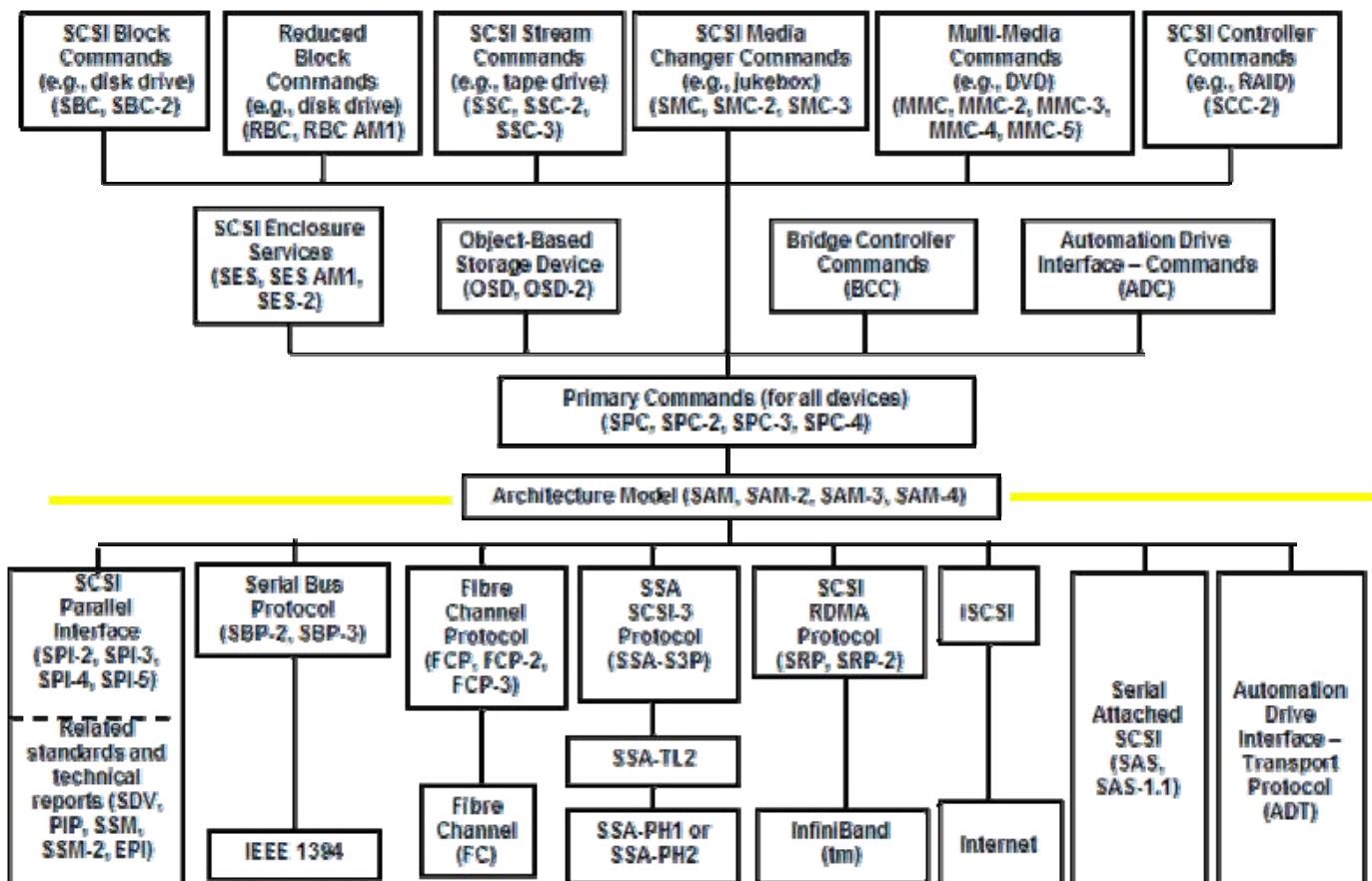
Primary Command Set:
gemeinsamer Befehlssatz aller Geräte

Das SCSI-Architektur-Modell enthält demnach 4 Schichten:

1.
2.
3.
4.

Durch die gemeinsamen Befehlssätze für die verschiedenen physikalischen Schnittstellen wird Software-Kompatibilität erreicht.

Schichten detailliert:



SSA: Serial Storage Architecture

Bild aus SCSI-3 Norm (Es fehlt die USB-Schnittstelle im Bild)

DOCUMENT LINKS

Command Sets:

<http://www.t10.org/drafts.htm#sbc> SCSI-3 Block Commands [first generation disk drive command set]
<http://www.t10.org/drafts.htm#sbc2> SCSI Block Commands - 2 [second generation disk drive command set]
<http://www.t10.org/drafts.htm#rbc> Reduced Block Commands [simplified disk drive command set]
<http://www.t10.org/drafts.htm#rbc-a1> Reduced Block Command Set Amendment 1 [first amendment to above standard]
<http://www.t10.org/drafts.htm#ssc> SCSI-3 Stream Commands [first generation tape drive command set]
<http://www.t10.org/drafts.htm#ssc2> SCSI Stream Commands - 2 [second generation tape drive command set]
<http://www.t10.org/drafts.htm#ssc3> SCSI Stream Commands - 3 [third generation tape drive command set]
<http://www.t10.org/drafts.htm#smc> SCSI-3 Media Changer Commands [first generation jukebox command set]
<http://www.t10.org/drafts.htm#smc2> SCSI Media Changer Commands - 2 [second generation jukebox command set]
<http://www.t10.org/drafts.htm - smc3> SCSI Media Changer Commands - 3 [third generation jukebox command set]
<http://www.t10.org/drafts.htm - mmc> Multi-Media Commands [first generation CD-ROM command set]
<http://www.t10.org/drafts.htm - mmc2> Multi-Media Commands - 2 [second generation CD and DVD command set]
<http://www.t10.org/drafts.htm - mmc3> Multi-Media Commands - 3 [third generation CD and DVD command set]
<http://www.t10.org/drafts.htm - mmc4> Multi-Media Commands - 4 [fourth generation CD and DVD command set]
<http://www.t10.org/drafts.htm - mmc5> Multi-Media Commands - 5 [fifth generation CD and DVD command set]
<http://www.t10.org/drafts.htm - scc2> SCSI Controller Commands - 2 [second generation RAID controller command set]
<http://www.t10.org/drafts.htm - ses> SCSI-3 Enclosure Commands [command set for enclosures (fans, power supplies, etc.)]
<http://www.t10.org/drafts.htm - ses-a1> SCSI-3 Enclosure Commands Amendment # 1 [first amendment to above standard]
<http://www.t10.org/drafts.htm - ses2> SCSI-3 Enclosure Commands - 2 [second generation command set for enclosures (fans, power supplies, etc.)]
<http://www.t10.org/drafts.htm - osd> Object-Based Storage Devices [object-oriented (e.g., files) command set for disk drives]
<http://www.t10.org/drafts.htm - osd2> Object-Based Storage Devices - 2 [second generation object-oriented (e.g., files) command set for disk drives]
<http://www.t10.org/drafts.htm - bcc> Bridge Controller Commands [command set for SCSI bridges between protocols]
<http://www.t10.org/drafts.htm - adc> Automation/Drive Interface - Commands [command set for the Automation/Drive interface]

Primary Command Sets:

<http://www.t10.org/drafts.htm - spc> SCSI-3 Primary Commands [first generation command set for all SCSI devices]
<http://www.t10.org/drafts.htm - spc2> SCSI Primary Commands - 2 [second generation command set for all SCSI devices]
<http://www.t10.org/drafts.htm - spc3> SCSI Primary Commands - 3 [third generation command set for all SCSI devices]
<http://www.t10.org/drafts.htm - spc4> SCSI Primary Commands - 4 [fourth generation command set for all SCSI devices]

Architectural Models:

<http://www.t10.org/drafts.htm - sam> SCSI-3 Architecture Model [first generation architectural model for SCSI devices]
<http://www.t10.org/drafts.htm - sam2> SCSI Architecture Model - 2 [second generation architectural model for SCSI devices]
<http://www.t10.org/drafts.htm - sam3> SCSI Architecture Model - 3 [third generation architectural model for SCSI devices]
<http://www.t10.org/drafts.htm - sam4> SCSI Architecture Model - 4 [fourth generation architectural model for SCSI devices]

Parallel Interface:

<http://www.t10.org/drafts.htm - spi2> SCSI Parallel Interface - 2 [second generation parallel interface (Ultra2)]
<http://www.t10.org/drafts.htm - spi3> SCSI Parallel Interface - 3 [third generation parallel interface (Ultra3 or Ultra160)]
<http://www.t10.org/drafts.htm - spi4> SCSI Parallel Interface - 4 [fourth generation parallel interface (Ultra320)]
<http://www.t10.org/drafts.htm - spi5> SCSI Parallel Interface - 5 [fifth generation parallel interface (Ultra640)]

Parallel Interface Related Standards and Technical Reports:

<http://www.t10.org/drafts.htm - sdv> SCSI Domain Validation [technical report on discovering maximum supported speeds]
<http://www.t10.org/drafts.htm - pip> SCSI Passive Interconnect Performance [standard on testing and measuring cables, connectors, etc.]
<http://www.t10.org/drafts.htm - ssm> SCSI Signal Modeling [technical report on modeling SCSI signals]
<http://www.t10.org/drafts.htm - ssm2> SCSI Signal Modeling - 2 [standard on modeling SCSI signals]
<http://www.t10.org/drafts.htm - epi> SCSI Enhanced Parallel Interface [technical report on enhancements to the parallel interface]

IEEE 1394:

<http://www.t10.org/drafts.htm - sbp2> Serial Bus Protocol - 2 [second generation protocol for transporting SCSI over IEEE 1394]

<http://www.t10.org/drafts.htm - sbp3> Serial Bus Protocol - 3 [third generation protocol for transporting SCSI over IEEE 1394]

Fibre Channel (FC):

<http://www.t10.org/drafts.htm - fcp> Fibre Channel Protocol [first generation protocol for transporting SCSI over Fibre Channel]

<http://www.t10.org/drafts.htm - fcp2> Fibre Channel Protocol - 2 [second generation protocol for transporting SCSI over Fibre Channel]

<http://www.t10.org/drafts.htm - fcp3> Fibre Channel Protocol - 3 [third generation protocol for transporting SCSI over Fibre Channel]

Serial Storage Architecture (SSA):

<http://www.t10.org/drafts.htm - s3p> SSA SCSI-3 Protocol [protocol for transporting SCSI over SSA]

<http://www.t10.org/drafts.htm - tl2> SSA Transport Layer - 2 [second generation SSA transport layer]

<http://www.t10.org/drafts.htm - ph1> SSA Physical Layer 1 [first generation SSA physical layer]

<http://www.t10.org/drafts.htm - ph2> SSA Physical Layer 1 [second generation SSA physical layer]

SCSI Remote Direct Memory Access Protocol:

<http://www.t10.org/drafts.htm - srp> SCSI RDMA Protocol [protocol for transporting SCSI over RDMA interfaces, e.g., InfiniBand (tm)]

<http://www.t10.org/drafts.htm - srp2> SCSI RDMA Protocol - 2 [second generation protocol for transporting SCSI over RDMA interfaces, e.g., InfiniBand (tm)]

Serial Attached SCSI:

<http://www.t10.org/drafts.htm - sas> Serial Attached SCSI [protocol and physical interface for transporting SCSI over serial links]

<http://www.t10.org/drafts.htm - sas1> Serial Attached SCSI - 1.1 [SAS protocol and physical interface enhancements for transporting SCSI over serial links]

Automation/Drive Interface:

<http://www.t10.org/drafts.htm - adt> Automation/Drive Interface - Transport Protocol [protocol/transport principally for Automation/Drive Commands]

SCSI / ATA Translation:

SCSI / ATA Translation (SAT) is a software translation layer that maps ATA devices to make them appear to be SCSI devices:

<http://www.t10.org/drafts.htm - sat> SCSI / ATA Translation [first generation of SAT]

About SCSI-2:

SCSI-2 is the predecessor to the SCSI-3 family of standards. It is a single standard containing all SCSI architecture layers for only the parallel interface. There is a companion standard for SCSI-2 called the *SCSI-2 Common Access Method and SCSI Interface Module (CAM)* that describes a layered software driver method for usage with SCSI-2. While most software drivers today do not exactly conform to CAM, they generally do follow its layered concepts:

<http://www.t10.org/drafts.htm - s2> Small Computer System Interface - 2 [second generation of SCSI]

<http://www.t10.org/drafts.htm - cam> SCSI-2 Common Access Method and SCSI Interface Module [software layering standard]

Der allgemeine Trend bei Schnittstellen, parallele (Bus-)Schnittstellen zunehmend durch serielle (Punkt-zu-Punkt-) Schnittstellen zu ersetzen (Vorteil: weniger Signale, dünnere Kabel), gilt auch für SCSI.

z.B.: PCI→PCI Express, ATA→SATA, SCSI→SAS (Serial Attached SCSI)

DEMO:

<http://www.t10.org/drafts.htm#rbc>-Reduced Block Commands [simplified disk drive command set]

[http://www.t10.org/drafts.htm - mmc5](http://www.t10.org/drafts.htm#mmc5) Multi-Media Commands - 5 [fifth generation CD and DVD command set]

4.2.1. Befehlsebenen

The first group are the *command sets*, which define commands used for various SCSI commands. These include the following: → **unterstützte Geräte (SCSI devices)**

Command Set	Description	Document	Abbreviation and Generation	Status	Standard or Project
Shared	Commands defined for <u>all</u> SCSI devices	<i>SCSI-3 Primary Commands SPC</i>	SPC	Published	X3.301-1997
			SPC-2	Pending Publication	T10 1236-D
			SPC-3	Development	T10 1416-D
Block	Commands defined for random-access devices that transfer data in blocks, such as <u>hard disks</u>	<i>SCSI-3 Block Commands</i>	SBC	Published	NCITS.306-1998
			SBC-2	Development	T10 1417-D
Block (Reduced)	A "simplified" version of the block command set	<i>SCSI-3 Reduced Block Commands</i>	RBC	Published	NCITS.330-2000
Stream	Commands for streaming, sequential-access devices such as <u>tape drives</u>	<i>SCSI-3 Stream Commands</i>	SSC	Published	NCITS.335-2000
			SSC-2	Development	T10 1434-D
Medium Changer	Commands for medium-changing devices such as <u>tape or disk "jukeboxes"</u>	<i>SCSI-3 Medium Changer Commands</i>	SMC	Published	NCITS.314-1998
			SMC-2	Development	T10 1383-D
Multimedia	Commands for "multimedia devices" (typically, <u>optical drives</u>)	<i>SCSI-3 Multimedia Commands</i>	MMC	Published	X3.304-1997
			MMC-2	Published	NCITS.333-2000
			MMC-3	Development	T10 1363-D
Multimedia (Reduced)	A "simplified" version of the multimedia command set	<i>SCSI-3 Reduced Multimedia Commands</i>	RMC	Development	T10 1364-D
Controller	Commands for <u>RAID controllers</u>	<i>SCSI-3 Controller Commands</i>	SCC	Published	X3.276-1997
			SCC-2	Published	NCITS.318-1998

Enclosure Services	Commands for SCSI device enclosures (power supply, cooling etc. ...)	<i>SCSI-3 Enclosure Services</i>	<i>SES</i>	Published	NCITS.305-1998
Object Based Storage Devices	Defines an object-oriented command set for accessing data	<i>Object Based Storage Device Commands</i>	<i>OSD</i>	Development	T10 1355-D

Beispiele für verschiedene Befehlstypen:

a) gemeinsamer Befehlssatz aller Geräte

-- **müssen in einem Gerät implementiert sein:**

- Four commands are **required** for all logical units:

Command	Type	Description
INQUIRY	R	Returns miscellaneous information about the logical unit. Peripheral Device Type (disk, tape, etc.), Vendor ID, Product ID, Serial Number, etc. Also returns vital product data (VPD) pages which contain Device Identifiers (worldwide names).
REPORT LUNS	R ⁰	Returns a list of the logical unit numbers present in the SCSI target device (mandatory for LUN 0)
REQUEST SENSE	R	Returns logical unit's current sense data
TEST UNIT READY	N	Returns GOOD if the logical unit is ready to accept media-access commands

inquiry=Abfrage
miscellaneous=verschiedene

sense data=Fehlerdaten
inquiry=Abfrage

sense data=Abfragedaten

-- **optional (können in einem Gerät implementiert sein):**

Command	Type	Description
LOG SELECT/ LOG SENSE	W/R	Read (sense) or write (select) statistical information from the logical unit. Information is defined by log pages .
MODE SELECT/ MODE SENSE	W/R	Read (sense) or write (select) control information from the logical unit. Information is defined by mode pages .
PERSISTENT RESERVATION IN/OUT	R/W	Prevent other initiators from accessing the logical unit.
READ BUFFER/ WRITE BUFFER	R/W	Read or write a memory buffer (for testing and firmware downloads)
REPORT SUPPORTED OPERATION CODES	R	Return a list of commands supported
REPORT SUPPORTED TASK MANAGEMENT FUNCTIONS	R	Returns a list of task management functions supported
SEND DIAGNOSTIC/ RECEIVE DIAGNOSTIC RESULTS	W/R	Perform diagnostics. Information is defined by diagnostic pages .

b) Befehle für die Platte

Auszug aus Inhaltsverzeichnis der Normschrift

5 Commands for direct-access block devices	5.15 READ LONG (16) command
5.1 Commands for direct-access block devices overview	5.16 REASSIGN BLOCKS command
5.2 FORMAT UNIT command	5.16.1 REASSIGN BLOCKS command overview
5.2.1 FORMAT UNIT command overview	5.16.2 REASSIGN BLOCKS parameter list
5.2.2 FORMAT UNIT parameter list	5.17 START STOP UNIT command
5.2.2.1 FORMAT UNIT parameter list overview	5.18 SYNCHRONIZE CACHE (10) command
5.2.2.2 Parameter list header	5.19 SYNCHRONIZE CACHE (16) command
5.2.2.3 Initialization pattern descriptor	5.20 VERIFY (10) command
5.2.2.4 Address descriptor formats	5.21 VERIFY (12) command
5.2.2.4.1 Address descriptor formats overview	5.22 VERIFY (16) command
5.2.2.4.2 Short block format address descriptor	5.23 VERIFY (32) command
5.2.2.4.3 Long block format address descriptor	5.24 WRITE (6) command
5.2.2.4.4 Bytes from index format address descriptor	5.25 WRITE (10) command
5.2.2.4.5 Physical sector format address descriptor	5.26 WRITE (12) command
5.3 PRE-FETCH (10) command	5.27 WRITE (16) command
5.4 PRE-FETCH (16) command	5.28 WRITE (32) command
5.5 READ (6) command	5.29 WRITE AND VERIFY (10) command
5.6 READ (10) command	5.30 WRITE AND VERIFY (12) command
5.7 READ (12) command	5.31 WRITE AND VERIFY (16) command
5.8 READ (16) command	5.32 WRITE AND VERIFY (32) command
5.9 READ (32) command	5.33 WRITE LONG (10) command
5.10 READ CAPACITY (10) command	5.34 WRITE LONG (16) command
5.10.1 READ CAPACITY (10) overview	5.35 WRITE SAME (10) command
5.10.2 READ CAPACITY (10) parameter data	5.36 WRITE SAME (16) command
5.11 READ CAPACITY (16) command	5.37 WRITE SAME (32) command
5.11.1 READ CAPACITY (16) command overview	5.38 XDREAD (10) command
5.11.2 READ CAPACITY (16) parameter data	5.39 XDREAD (32) command
5.12 READ DEFECT DATA (10) command	5.40 XDWRITE (10) command
5.12.1 READ DEFECT DATA (10) command overview	5.41 XDWRITE (32) command
5.12.2 READ DEFECT DATA (10) parameter data	5.42 XDWRITEREAD (10) command
5.13 READ DEFECT DATA (12) command	5.43 XDWRITEREAD (32) command
5.13.1 READ DEFECT DATA (12) command overview	5.44 XPWRITE (10) command
5.13.2 READ DEFECT DATA (12) parameter data	5.45 XPWRITE (32) command
5.14 READ LONG (10) command	

Protokoll-Ebene

Die Protokoll-Ebene spezifiziert das Format der Informationspakete (Befehlspakete, Datenpakete), eine mögliche Blockfolge, Codierung der Information wie z.B. 8/10 Bit-Codierung usw.

The other two groups of standards are *protocols* and *interconnects*. Protocols define how data is interchanged, and transports describe the physical ways that protocols are implemented. They are closely related and in some cases now are defined by a single document, which makes them a bit difficult to present in an organized way. I will start by discussing the various protocols (also called *transports* by some):

Protocol	Description	Document	Abbreviation and Generation	Status	Standard or Project
Interlocked (Parallel Bus)	Defines the protocol for "regular" parallel SCSI	<i>SCSI-3 Interlocked Protocol</i>	<i>SIP</i>	Withdrawn; now incorporated into later versions of the SCSI-3 Parallel Interface	--
Fibre Channel	Defines the protocol for running SCSI on the Fibre Channel interface	<i>SCSI-3 Fibre Channel Protocol</i>	<i>FCP</i>	Published	X3.269-1996
			<i>FCP-2</i>	Pending Publication	T10 1144-D
Serial Bus	Defines the protocol for transporting commands over the IEEE-1394 (serial) interface	<i>Serial Bus Protocol</i>	<i>SBP</i>	Withdrawn	--
			<i>SBP-2</i>	Published	NCITS.325-1998
Serial Storage Architecture	Defines the transport layer for Serial Storage Architecture, an advanced interface used in servers and enterprise hardware; there are two documents that specify the protocol	<i>Serial Storage Architecture SCSI-3 Protocol</i>	<i>SSA-S3P</i>	Published	NCITS.309-1998
		<i>Serial Storage Architecture Transport Layer</i>	<i>SSA-TL2</i>	Published (replaced SSA-TL1)	NCITS.308-1998

4.2.2. Physikalische Ebene

Die physikalische Ebene (Interconnect) spezifiziert die eigentliche Datenübertragung hinsichtlich Eigenschaften wie parallel/seriell, asymmetrisch/symmetrisch (differentiell), Signalpegel, Datenrate usw.

Finally, this table shows the interconnect standards, also sometimes called *physical layer* documents (since they describe how devices are physically connected):

Interconnect	Description	Document	Abbreviation and Generation	Status	Standard or Project
Parallel Bus	Describes the electrical signaling, connectors and related issues associated with "regular" parallel SCSI; starting with SPI-2 these include the formerly separate SIP protocol document	SCSI-3 Parallel Interface	SPI	Published	X3.253-1995
			Fast-20 (addendum to SPI)	Published	X3.277-1996
			SPI-2	Published	X3.302-1999
			SPI-3	Pending Publication	T10 1302-D
			SPI-4	Development	T10 1365-D
Fibre Channel	Several documents define alternative physical layer standards for Fibre Channel; these are maintained by the T11 technical committee and include Fibre Channel Arbitrated Loop (FC-AL) and several revisions of the Fibre Channel Physical Interface (FC-PHx)				
Serial Bus	The physical layer standards for the serial bus (IEEE-1394) are developed by the IEEE High Performance Serial Bus Bridges Working Group (P1394)				
Serial Storage Architecture	Defines the physical connections for the Serial Storage Architecture interface	Serial Storage Architecture Physical Layer	SSA-PH	Published	X3.293-1996
			SSA-PH2	Published	NCITS.307-1998

Whew. Well, you can certainly understand now why some people find SCSI-3 "a bit confusing". :^) I hope that this "road map" helps to make things a bit more clear for my readers though!



Next: [SCSI\(-3\) Parallel Interface \(SPI\)](#)

4.3. SCSI Parallel als Beispiel für ein dezentrales Buskonzept

Der SCSI-Bus wurde als parallele Geräteschnittstelle entwickelt mit (derzeit) 16+2 Bit Datenbus sowie zusätzlich 9 Steuersignalen. Am Host-Controller können maximal 15 Geräte angeschlossen sein. Die differentielle Signalübertragung erfolgt über ein 68-adriges Kabel. Die parallele SCSI-Schnittstelle wird zunehmend durch die serielle SCSI-Schnittstelle ersetzt.

Jedes Gerät, das am Bus angeschlossen wird, benötigt einen SCSI-Controller für das Busprotokoll ("intelligente" Geräte). Ziel des SCSI-Busses ist es, die Geräte herstellerunabhängig am Computer anzuschließen, d.h., z.B. unterschiedliche Festplatten unterscheiden sich aus Sicht des Computers nur noch durch ihre Kapazität.

Ein SCSI-Bus ist über verschiedene Host-Adapterkarten an Computern wie PC, Workstation usw. anschließbar.

4.3.1. *Begriffe*

Initiator und Target:

Es gibt zwei Funktionen eines Busteilnehmers am Bus:

Ein „**Initiator (Client)**“ startet eine Bus-Kommunikation und ein „**Target (Server)**“ reagiert darauf. Ein „**Initiator**“ verlangt die Ausführung eines Befehls (ist üblicherweise der PC); ein „**Target**“ ist ein Gerät, das einen Befehl ausführt.

Die „**Master**“-Funktion (=Bus anfordern) und die „**Slave**“-Funktion (auf die Bus-Anforderung reagieren) am Bus wird zwischen dem Initiator und dem Target hin- und hergereicht.

Z.B. fordert der Initiator (Host-Adapter im PC) den Bus an (=Master), um einen Befehl an ein Target (Gerät) zu übergeben (=Slave) und gibt dann den Bus wieder frei. Das Target (Gerät) führt den Befehl aus und fordert dann den Bus an (=Master), um die Daten an den Initiator (Host-Adapter im PC, Slave) zu übergeben.

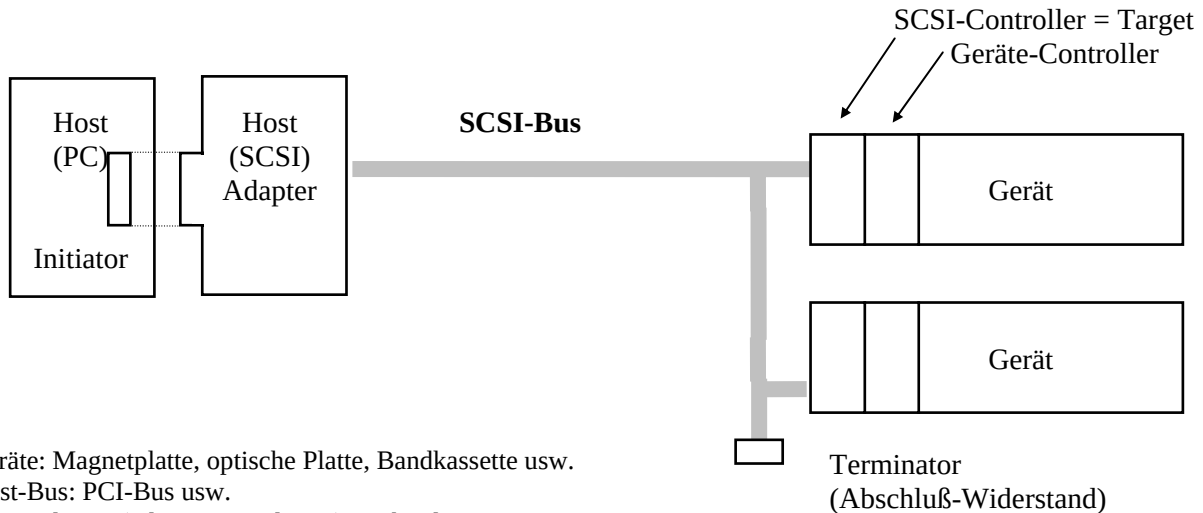
Bei bestimmten Befehlen wie dem „Copy“-Befehl kann ein Gerät auch ein Initiator sein.

Aufgabe:

Vergleichen Sie Master- und Slave-Funktion beim PCI-Bus und beim SCSI-Bus.

.....

Anschluß der Geräte:



Geräte: Magnetplatte, optische Platte, Bandkassette usw.

Host-Bus: PCI-Bus usw.

Host-Adapter (oder SCSI-Adapter): verbindet SCSI-Bus mit Host-Bus.

Adressierung:

a) Device ID (Identification)

Die ID legt fest: und

Es sind soviele Busteilnehmer (Host-Adapter + SCSI-Controller) anschließbar wie Datenleitungen an SCSI-Bus vorhanden sind. Jeder Busteilnehmer ist über eine Datenleitung eindeutig identifizierbar (Device ID oder SCSI ID). Dadurch ist die Anzahl der Busteilnehmer limitiert. Z.B. sind maximal 8 Busteilnehmer beim 8Bit-Datenbus (ID0 .. ID7) oder 16 Busteilnehmer beim 16Bit-Datenbus (ID0 .. ID15) anschließbar.

Die Identifikation ID wird in den Bus-Phasen "Arbitrierung" und "Selection" ("Reselection") benötigt.

Reihenfolge der Prioritäten (beginnend mit der höchsten Priorität):

.....

Welche ID welchem Busteilnehmer zugeordnet ist, wird vom Anwender festgelegt (z.B. über Schalter am Gerät einstellbar).

Übliche Einstellungen (Bspl.: Netzwerk-Server im Labor LDS):

Hostadapter: ID7 (=höchste Priorität),

SCSI-Controller Bootplatte: ID0

"Gespiegelte" Datenplatten: SCSI-Controller Primärplatte: ID4,
SCSI-Controller Sekundärplatte: ID5

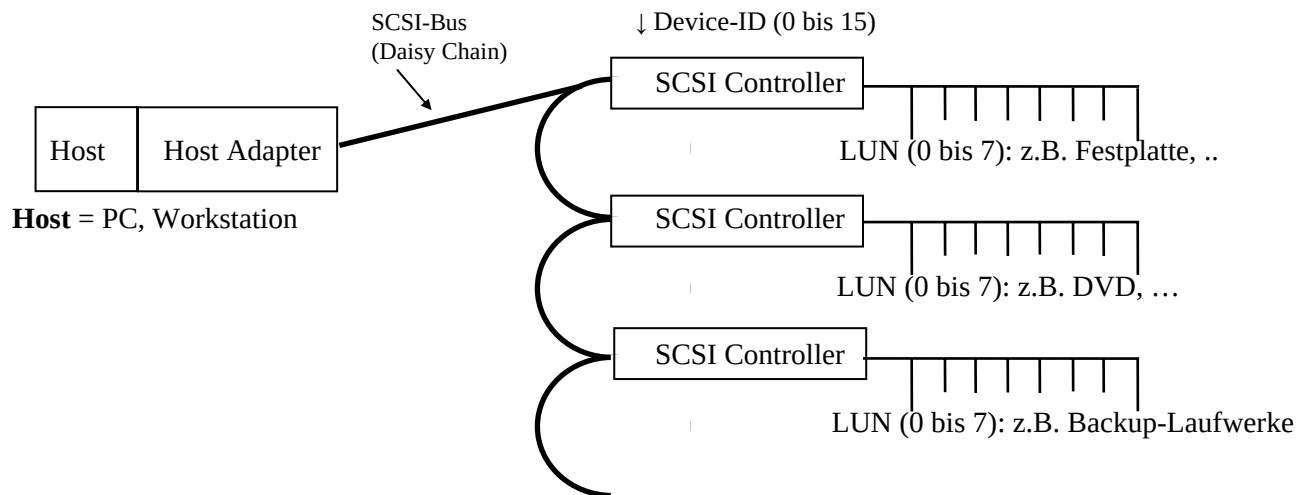
SCSI-Controller CD-ROM: ID...

SCSI-Controller DAT-Laufwerk: ID...

b) LUN (Logical Unit Number)

Eine Logical Unit ist eine „Untermenge“ eines Targets. Im Befehlsformat ist ein 3Bit-Feld für die Geräte-Nummer LUN reserviert. Die LUN kann verwendet werden um maximal 8 Geräte (LUN0 .. LUN7) an einem SCSI-Controller anzuschließen (z.B. mehrere Laufwerke in einem Kassetten Backup System) oder um ein Gerät an einem SCSI-Controller in maximal 8 logische Geräte (z.B. Zonen bei einer Festplatte) zu unterteilen.

Konfigurations-Beispiel:



Der SCSI-Bus kann innerhalb des Rechners verlaufen (um im Rechner integrierte Geräte mit dem Hostadapter zu verbinden) oder er kann außerhalb des Rechners befindliche freistehende Geräte ("stand-alone") anschließen oder er verbindet "gemischt" (interne und externe Geräte) mit dem Hostadapter. Im letzteren Fall ist der Hostadapter in der Mitte des SCSI-Busses.

Task:

Ein Auftrag (Anwendung), der von einem Target (Logical Unit) ausgeführt wird.

Wide/Fast und Transferraten bei SCSI Parallel:

„Wide“: 2Byte-Datenbus, definiert in SCSI-2

"Ultra2 SCSI" (68pin LVD) = 80 MByte/s (bei 2 Byte), LVD (low volt. diff.): max. 12 m, HVD: 25m

"Ultra 320 SCSI" (68pin LVD) = 320 MByte/s (bei 2 Byte), LVD: max. 12m, HVD: 5m

"Ultra 640 SCSI" (68pin LVD) = 640 MByte/s (bei 2 Byte), LVD: max. 7m, HVD: 5m

Die Option "Wide" gilt nur für die Daten-Phasen "Data In" und "Data Out".

Grundsätzlich beginnt der Befehlstransfer auf 1Byte-Busbreite und mit asynchroner Übertragung. Der Übergang auf 2Byte-Bus wird per Bus-Message WDTR (wide data transfer request) eingestellt. Es können daher 1Byte-Geräte an einen 2Byte-Bus angeschlossen werden.

Die mittlere Datenrate des Busses liegt deutlich unter den obigen Werten da die niedrigeren Datenraten bei der Befehlsübertragung, die Zeiten für Busbelegung und Partnerwahl und die sonstigen Programmlaufzeiten im Host (Betriebssystem und Treibersoftware) die Ausführung eines SCSI-Befehls sehr verzögern können ("Overhead").

Asynchron/synchron bei SCSI Parallel:

Es gibt drei Datenübertragungsprotokolle (handshaking protocols), welche die Übergabe der Daten zwischen dem Initiator und dem Target steuern.

„Asynchron“ bedeutet, daß jedes Wort (Byte) per Signal angefordert und quittiert werden muß. Dadurch ist die Datentransferrate (MByte/s) - vor allem bei längeren Kabeln - begrenzt.

„Synchron“ verlangt ebenfalls ein Anforderungs- und ein Quittungssignal pro Wort (Byte). Diese können jedoch summarisch übertragen werden, z.B. 8 Anforderungssignale im Block, dann 8 Datenworte, dann 8 Quittungssignale im Block, was höhere Datentransferraten ermöglicht.

Synchrone Datenübertragung ist nur beim Daten-Transfer möglich. Befehls-Transfers erfolgen asynchron.

Grundsätzlich können alle SCSI-Geräte asynchron arbeiten. Die synchrone Datenübertragung wird per Bus-Message SDTR (synchronous data transfer request) eingestellt.

SE(single ended)/ DI(differential) bei SCSI Parallel:

siehe Kap. Festplatte „Elektrische Übertragungsarten“

Terminierung der Kabel bei SCSI Parallel:

Beide Kabelenden müssen mit einer Terminierung abgeschlossen sein.

Die Terminierung kann erfolgen:

--durch

--durch

--durch

In der Regel liegt der Hostadapter (mit integrierter Terminierung) an einem der Kabelenden. Liegt der Hostadapter in der Kabelmitte muß seine Terminierung ausgeschaltet werden (Konfiguration Host Adapter).

Aufgabe dieser Terminierung ist es,

-- und

--

Passive Terminierung: Widerstandsteiler, siehe "Elektrische Übertragungsarten auf dem Schnittstellenkabel", Kap. Festplatte.

Aktive Terminierung: Integrierter Spannungsregler.

Liefert kürzere low→high-Schaltzeiten (Übergang in den nicht-aktiven Zustand), vor allem wichtig bei Anforderungs- und Quittungssignal (REQ, ACK).

Übertragung störsicherer als bei passiver Terminierung.

4.3.2. Befehlsformat

Befehlsebenen: siehe Kapitel in „SCSI-3 Architektur“.

Befehlsformat:

Der Initiator sendet in der Befehlsphase (**command phase**) Befehle an ein Target. Der **Command Descriptor Block CDB** enthält den Befehl und die Befehlsparameter. Es existieren verschiedene Blocklängen dafür: 6Byte, 10Byte, 12Byte, 16 Byte oder variable Byteanzahl.

Command Descriptor Block CDB für Lesen (12Byte-Format):

Bit	7	6	5	4	3	2	1	0
Byte								
0	Operation code							
1	Logical unit number			Reserved				
2	(MSB)							
3	Logical block address (if required)							
4								
5	(LSB)							
6	(MSB)							
7	Transfer length (if required)							
8	Parameter list length (if required)							
9	Allocation length (if required)							
10	(LSB)							
11	Reserved							
12	Control							

Logical Block: z.B. 512 Byte bei Festplatte

Logische Blockadresse: logische Blockadresse, bei der der Lesevorgang beginnt.

Transferlänge: Zahl der zu übertragenden Datenblöcke (aufeinanderfolgend)

Control Byte mit Flags, Link:

Verknüpfung mit weiteren Befehlen möglich. Erster Befehl endet nicht mit Busfreigabe, sondern es wird sofort der nächste Befehl vom Initiator übertragen ohne Unterbrechung durch einen anderen Initiator. Z.B. die Folge „Lesen Block/Ändern Block“.

Frage: Welche maximale Blockadresse kan mit dem 12Byte-Befehlsformat adressiert werden ?

Befehlswarteschlange im Gerät (Command Queueing):

Ein Target kann mehrere Befehle aufnehmen und in einer Warteschlange ordnen bzw. die Reihenfolge der Ausführung ändern (Command Queueing ist eine optionale Funktion).

Vorteil:

Logische Blockadressierung:

Der Initiator übergibt im Befehl eine logische Blockadresse (und keine physikalische Adresse wie z.B. Zyl.Nr./ Kopf Nr./ Sektor Nr.).

4.3.3. Bus-Kommunikation

Für „SCSI parallel“ wurde ein dezentrales Buskonzept entwickelt. Die Belegung des Busses wird mit einem Verfahren ohne zentrale Bussteuerung abgewickelt. Jeder Busteilnehmer (Host oder Gerät) der den Bus belegen will, um einen Transfer (Befehls- oder Datentransfer) durchzuführen, überprüft zunächst ob der Bus frei ist. Erst dann wird der Bus belegt, der Partner adressiert und der Transfer durchgeführt.

Eine zentrale Bussteuerlogik wie z.B. beim PCI Conventional (Kap. 3.3.4. d) entfällt somit.

Befehlsphasen (command phases):

Ein Befehl wird in mehreren Schritten ausgeführt:

Schritt 1: Bus-Belegung und Partnerwahl (Einzelheiten siehe Zustandsdiagramm):

- Phase Arbitration: Bus belegen (durch Initiator oder Target)
- Phase Selection: Partner auswählen (durch Initiator oder Target)
- (Phase Re-Selection)

Schritt 2: Informationstransfer:

Die Information wird übertragen in Form von:

- Phase Messages:
betrifft die allgemeine Information über einen Gerätebefehl (z.B. "Befehl beendet" oder "Disconnect während Befehlsausführung erlaubt") oder für die Buseinstellung (z.B. "Synchrone Übertragung" oder "2Byte-Transfer").
- Phase Commands:
betrifft die eigentliche Befehlsausführung auf dem Gerät (Anzahl Blöcke, Adresse)
- Phasen Data Out, Data IN
- Phase Status: Meldung über (nicht) erfolgreiche Befehlsausführung

Die Phasen werden vom Target gesteuert, das über das BSY-Signal den Bus hält und über die Signale MSG, C/D und I/O die Informationstransfer-Phasen auswählt.

- Message Out: Initiator sendet an Target die Geräte-Nr. LUN (falls Signal ATN vom Initiator gesetzt ist)
- Message In: Target sendet an Initiator
- Command: Initiator sendet Befehl (Command Descriptor Block)
- Data Out
- Data In
- Status: Target meldet dem Initiator ob Befehl erfolgreich ausgeführt
- Bus Free: Target gibt den Bus frei durch Wegnahme des BSY-Signals

Interruptfähigkeit (Asynchronous Event Notification AEN):

Ein Target kann einen Initiator über ein „asynchrones Ereignis“ informieren, d.h., über ein Ereignis, das nicht über einen Befehl ausgelöst wurde. Ein Target (jetzt kurzzeitig Initiator) informiert den eigentlichen Initiator (jetzt kurzzeitig Target) über eine Message von diesem Ereignis. Z.B. Geräte mit wechselbarem Datenträger informieren den Initiator (PC) über einen Datenträgerwechsel. Diese Funktion ist optional (Target schickt mit MSG=1 eine AEN-Message an Host).

Bussteuerung:

Das SCSI-Kabel ist von Teilnehmer zu Teilnehmer geschleift („daisy chain“). Es kann immer nur 1 Teilnehmer am Bus aktiv werden. Deshalb müssen bei mehr als 2 Teilnehmern am Bus die Busbelegung (Arbitration) und die Auswahl des Partners (Selection) geregelt werden. Die Busbelegung (z.B. durch den Initiator), und die Teilnehmergeauswahl (z.B. Initiator wählt Target aus) werden durch ein → **Zustandsdiagramm** beschrieben.

Eine zentrale Bussteuerung (wie z.B. beim PCI-Bus) ist nicht vorhanden; es entfallen daher Request- und Grant-Signale (Request=Busanforderung, Grant=Buszuteilung).

Frage:

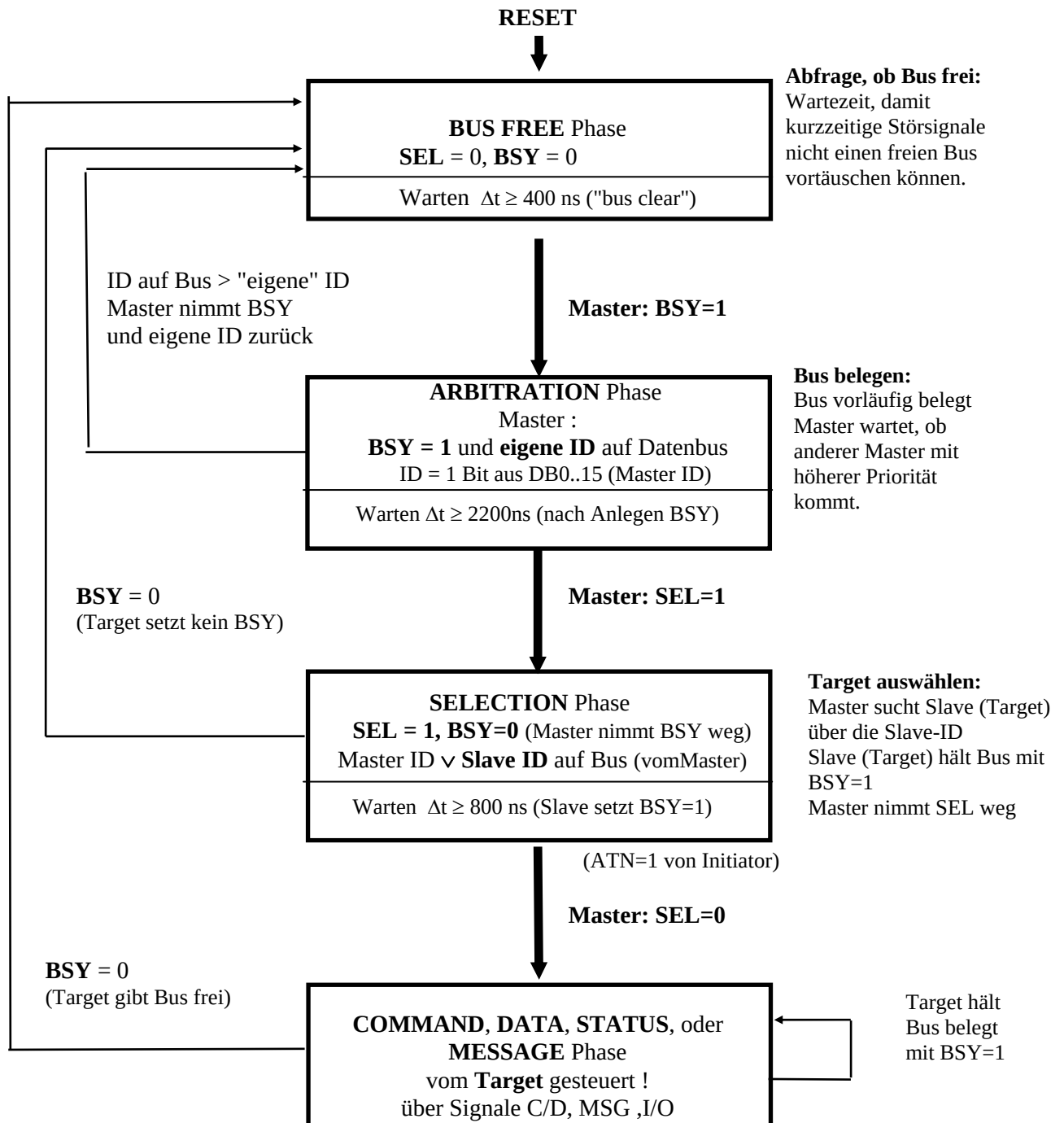
Wie viele Signaladern bräuchte man beim 16Bit-SCSI-Bus, single ended, bei einer zentralen Bussteuerung mit Anforderungs- und Quittungs-Signalen ?
Vergleich mit PCI Conventional.

Zustandsdiagramm:

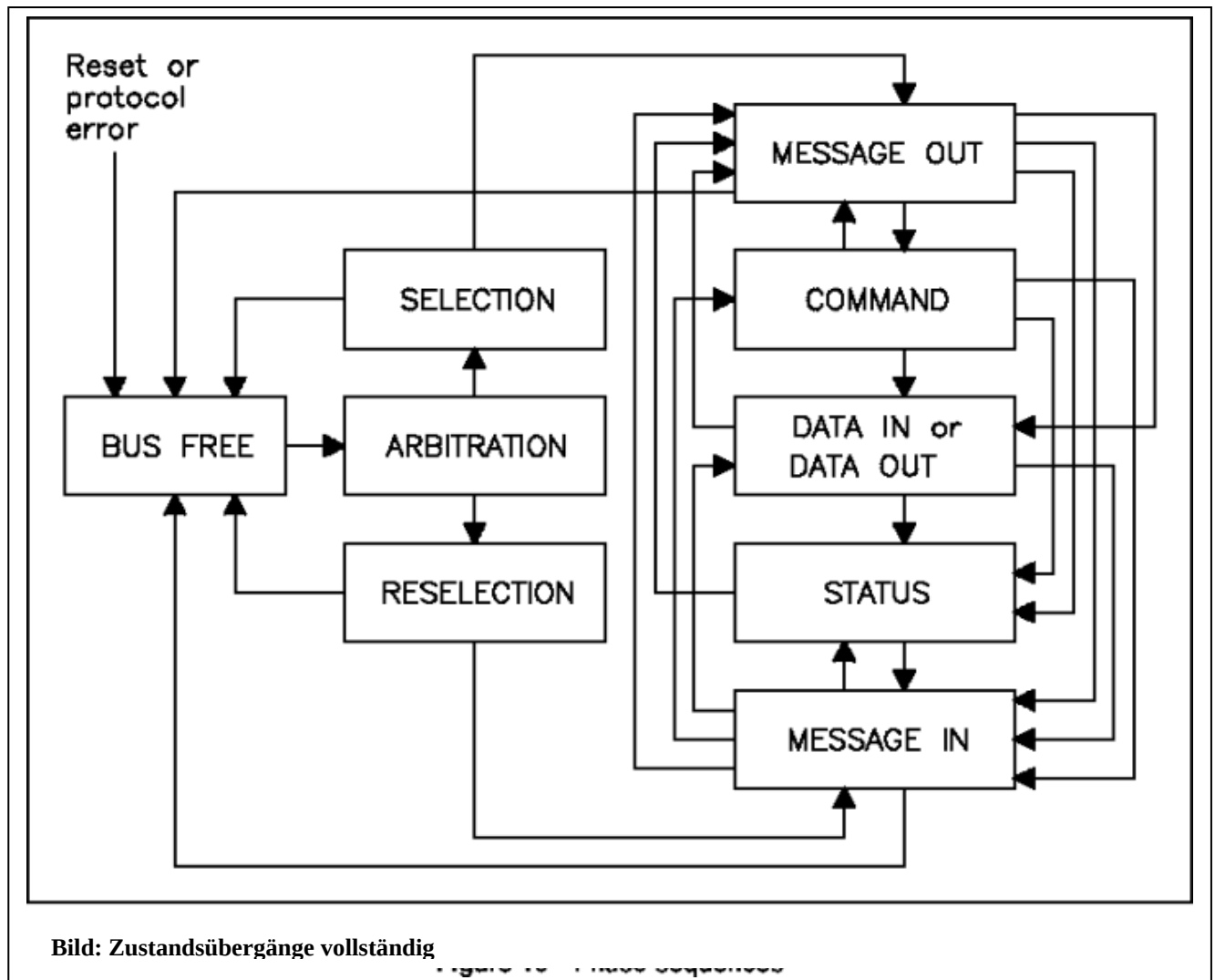
Signal-Zustände: **Signal=1** (= aktiv), **Signal=0** (= nicht aktiv)

Frage: „wie funktioniert Buszuteilung (Arbitrierung) ohne zentrale Bussteuerung?“

in DEMO: Datei sampleex.dtt



Bei einem Befehl werden nicht immer alle Phasen ausgeführt. Wenn ein Target während eines Befehls erkennt, daß bis zum nächsten Informationstransfer noch viel Zeit vergeht, kann es die Verbindung zum Initiator unterbrechen und den Bus freigeben ("**Disconnect**" und "**Reconnect**"). Z.B. kann nach der Befehlsübergabe ein Target den Bus freigeben und danach den Befehl auf dem Gerät ausführen. Dann erfolgt wieder eine Belegung des Busses (= "Reselection" Phase) und die Übertragung der Daten (z.B. "Data In" Phase beim Lesen von Platte).



Bspl. Read Command: Code=08 (Lesen von Platte), Host (ID7), Platte (ID0)

Befehls-Phase	Datenbus D7 .. D0 (Hex)	Erklärung
Bus free		
Arbitration	80	ID=7 binär: 1000 0000
Selection with ATN	81	ID=7, ID=0
Message Out	C0	LUN0, Disconnect o.k. Initiator erlaubt Disconnect
Command	08 02 00 00 07 00	6Byte Command
Message In	04	Disconnect
Bus free		
Bus free		Geräteaktivität:
Bus free		Positionieren, Block Lesen
Bus free		
Arbitration ?		
Reselection ?		
Message In	80	LUN 0
Data In	XX XX	Lesedaten vom Target zum Initiator
Status	00	Geräteaktivität o.k.
Message In	00	Befehl beendet
Bus free		

Ergänzen Sie die Felder für Arbitration und Reselection! (Message In: Standpunkt Initiator)

Fragen:

--Wieviel Blöcke werden von welcher Beginndresse gelesen ?.....

--Welche Datenleitungen sind dem Initiator und dem Target für ihre Identifikation zugeteilt ?.....

Führt der Initiator mit mehreren Targets Befehle gleichzeitig aus, dann können einzelne Phasen der verschiedenen Befehle miteinander verschachtelt sein.

Die temporäre Freigabe kann z.B. bei Multitasking/Multiuser-Betriebssystemen wichtig sein. Der Bus ist dann nicht während der Positionier- und Drehwartezeit der Platte von einer Task blockiert!

Messages:

(Interface zwischen Initiator und Target betreffend, z.B. Einstellung "synchrone Übertragung", Einstellung "Wide" beim Datenbus):

Das Target steuert die Informationsübertragung in den Message-Phasen durch die Bussignale. Will der Initiator dem Target eine Nachricht (Message) senden, fordert er das Target über das ATN-Signal auf, die Nachricht abzuholen. **Die Steuerung der dafür notwendigen Signale liegt beim Target.**

DEMO: „Aufzeichnung von Busaktivitäten mittels SCSI-Analyzer“

Diskussion

SCSI-2 Spec aus Internet (google.de/SCSI-2)

????????????????????????????

Status:

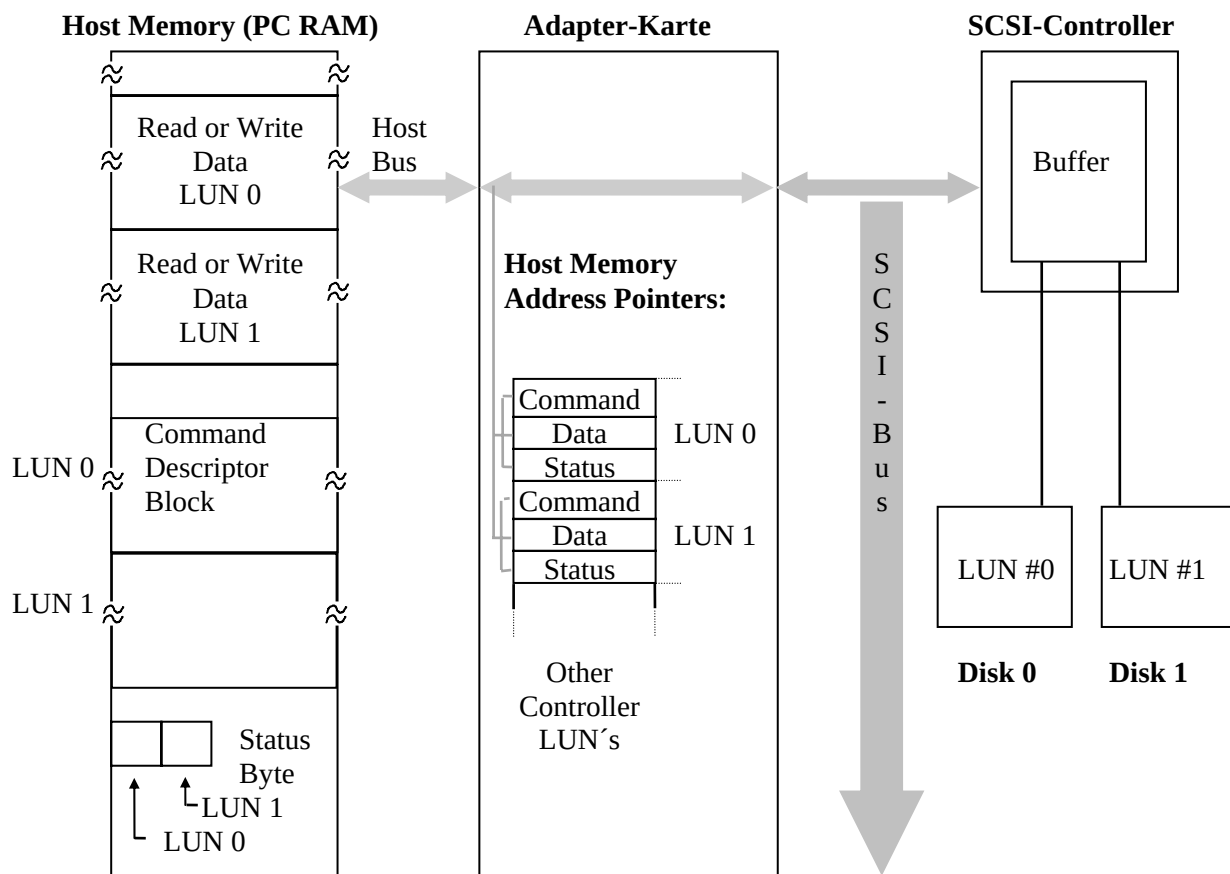
(Meldung des Targets über die Befehlsausführung im Gerät):

Der Status des Targets kann dem Initiator auf zweifache Weise gemeldet werden:

- Im Status-Byte am Ende eines Befehls (kurze Meldung) oder
- Über einen Request Sense Befehl. Diesen fordert das Target mit dem Status-Byte=02H vom Initiator an. In der Datenphase des Befehls sendet das Target dann den Status (ausführlichere Meldung).

Befehls-, Daten- und Statusfelder:

Die Adress-Zeiger (Address Pointer) auf Befehls-, Daten- und Statusfeld im Arbeitsspeicher werden vom Host in die Adapter-Karte geladen.



LUN = Logical Unit (Gerät)

4.3.4. Bussignale

Der Bus ist low-aktiv: Signal=1 (Signal aktiv) bei Pegel=low
Signal=0 (nicht aktiv) bei Pegel=high

Bussignale:

Signalname	Funktion	Sender
BSY	"Bus belegen", Wired-OR	Initiator/Target
SEL	"Auswahl Busteilnehmer", Wired-OR Init. wählt Target oder Target wählt Init. (Reselection)	Initiator/Target
ATN	"Achtung", zeigt dem Target an, daß Initiator Message hat. Das Target holt die Message vom Initiator ab über "Message Out Phase".	Initiator
RST	"Reset", Wired-OR. Empfehlung: nur Initiator	Initiator/(Target)
C/D	Die 3 Signale steuern den Informations- transfer und zeigen an, welche Information am Bus übertragen wird, siehe Tabelle unten.	Target
I/O		Target
MSG		Target
REQ	"Anforderung", 2 Bedeutungen, siehe Tab. unten	Target
ACK	"Quittung"	Initiator
DB 15..0	"Datenbus" 16 Bit	Initiator(Data-Out)/ Target(Data-In)
P	"Parität" (odd) pro Byte	

Anzahl Signale bei asymmetrischer Übertragung (TTL-Pegel):

26 Signale (50-adriges Kabel)

Anzahl Signale bei symmetrischer Übertragung (differentiell Üb.):

51 Signale (68-adriges Kabel)
(RST nur 1 Ader)

Frage:

Wieso ist BUSY-Signal als "Wired-OR" ausgeführt?

Wie könnte die "Wired-OR"-Verknüpfung des BSY-Signals der Busteilnehmer realisiert sein?

Signale C/D, I/O und MSG:

Sie legen die Informations-Transfer-Phasen fest

Signale			PHASE Name	Richtung des Transfers	Kommentar
C/D	MSG	I/O			
0	0	0	DATA OUT PHASE	(INIT to TARGET)	DATA PHASES
0	0	1	DATA IN PHASE	(INIT from TARGET)	
1	0	0	COMMAND PHASE	(INIT to TARGET)	
1	0	1	STATUS PHASE	(INIT from TARGET)	
0	1	0	*		
0	1	1	*		
1	1	0	MESSAGE OUT PHASE	(INIT to TARGET)	MESS. PHASES
1	1	1	MESSAGE IN PHASE	(INIT from TARGET)	

Notes: 0 = False, 1 = True;

INIT = Initiator; * = Not used;

Signale REQ und ACK: Sie steuern den Datentransfer:

Die Datentransfersignale REQ, ACK können verschiedene Bedeutungen haben.

„Bedeutung 1“: REQ = Aufforderung an Partner "Übernehme Daten"

(Die Daten sind schon auf dem Bus)

ACK = Bestätigung vom Partner "Habe Daten Übernommen"

z.B. Lesedaten von Platte: Target → CPU

„Bedeutung 2“: REQ = Aufforderung an Partner "Sende Daten"

(Der Partner legt die Daten auf den Bus)

ACK = Bestätigung vom Partner "Daten sind auf dem Bus"

z.B. Schreibdaten auf Platte: CPU → Target

Beispiel 1: Target sendet an den Initiator (entspricht „Bedeutung 1“)

asynchroner Transfer Daten		
Target	➔	Initiator
Byte 1		
REQ	→	
		übernehmen
	←	ACK
Byte 2		
REQ	→	
		übernehmen
	←	ACK

synchroner Transfer (nur für Daten, nicht für Befehle)		
Target	➔	Initiator
Byte 1		
REQ	→	übernehmen
Byte 2		
REQ	→	übernehmen
Byte 3		
REQ	→	übernehmen
.....		
	←	ACK
	←	ACK
	←	ACK

$\sum \text{REQ} = \sum \text{ACK}$, sonst Fehler!

Beispiel 2: Target fordert Initiator auf, zu senden (entspricht „Bedeutung 2“):

asynchroner Transfer Daten oder Befehle		
Target	←	Initiator
REQ	→	
		Byte 1
	←	ACK
übernehmen		
REQ	→	
		Byte 2
	←	ACK
übernehmen		

Fehlersuche am SCSI-Bus

Es gibt SCSI-Analysatoren für die Signalaufzeichnung und SCSI-Initiator-Emulatoren und SCSI-Target-Emulatoren für die Nachbildung von Initiator und Target.

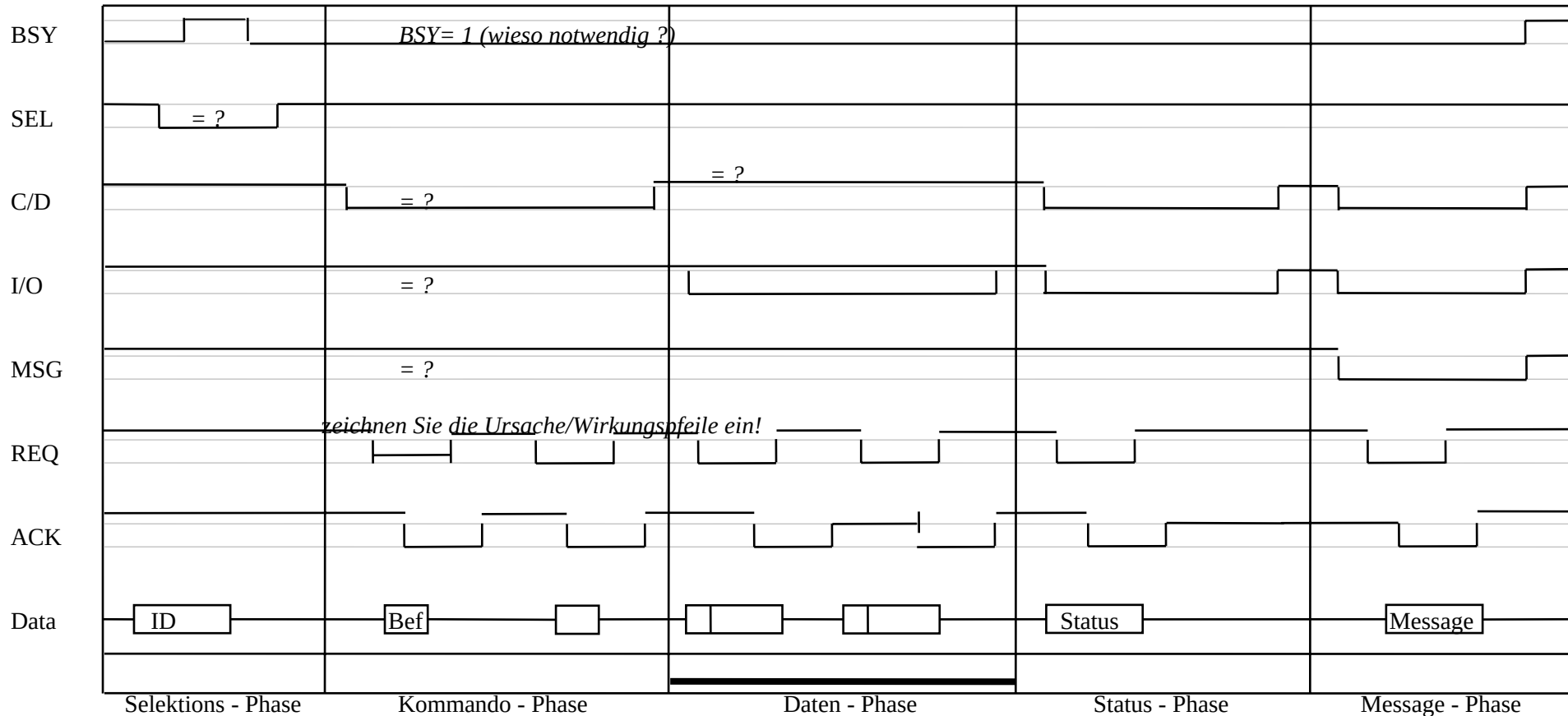
Impulsdiagramm:

siehe nächste Seite

Tabelle „Ihr Wissen über den SCSI-Bus“ am Kapitel-Anfang ergänzen/korrigieren!

SCSI - Bus: Zeitdiagramm einer Befehlsausführung (ohne Arbitrierungs-Phase)

Pegel-Darstellung (low=aktiv)



	Der Initiator:-Adresse (ID) des Targets auf den Datenbus, dann SEL-Sig. Target: BUSY.	Das Target legt C/D und REQ an. Die einzelnen Befehlsbytes können dann übertragen werden. Die Anzahl der übertragenen Bytes hängt vom Kommando ab.	Hier werden ggf. Daten entweder zum oder vom Initiator übertragen. I/O gibt die Richtung an. Die Anzahl der Bytes hängt vom Kommando ab.	Das Target sendet hier ein Status-Byte an den Initiator. Z.B. den Wert 00, wenn kein Fehler aufgetreten ist.	Das Target sendet hier ein Mitteilungs-Byte an den Initiator. Z.B. den Wert 00, für Abschluß des Kommandos.
--	---	--	--	--	---

Kein arbitrierendes System

4.4. Serial Attached SCSI (SAS)



Der Trend „Parallelschnittstellen durch serielle Schnittstellen zu ersetzen“ mit den damit verbundenen Vorteilen (kleine Busbreite, Skalierbarkeit usw.) gilt auch für SCSI.

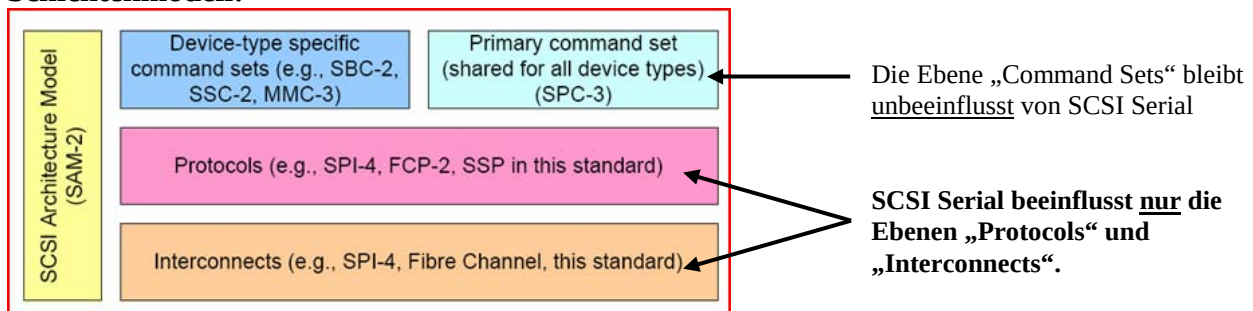
Serial Attached SCSI (SAS) ist eine Punkt-zu-Punkt-Verbindung (point-to-point interface) zwischen SAS Initiator Device (Host) und SAS Target Device (Gerät). Zwischen beiden kann ein Netzwerk (Expander Devices) geschaltet sein (Netzwerke siehe auch Vorlesung „Computernetze“).

Wichtige Unterschiede (Vorteile) gegenüber SCSI Parallel:

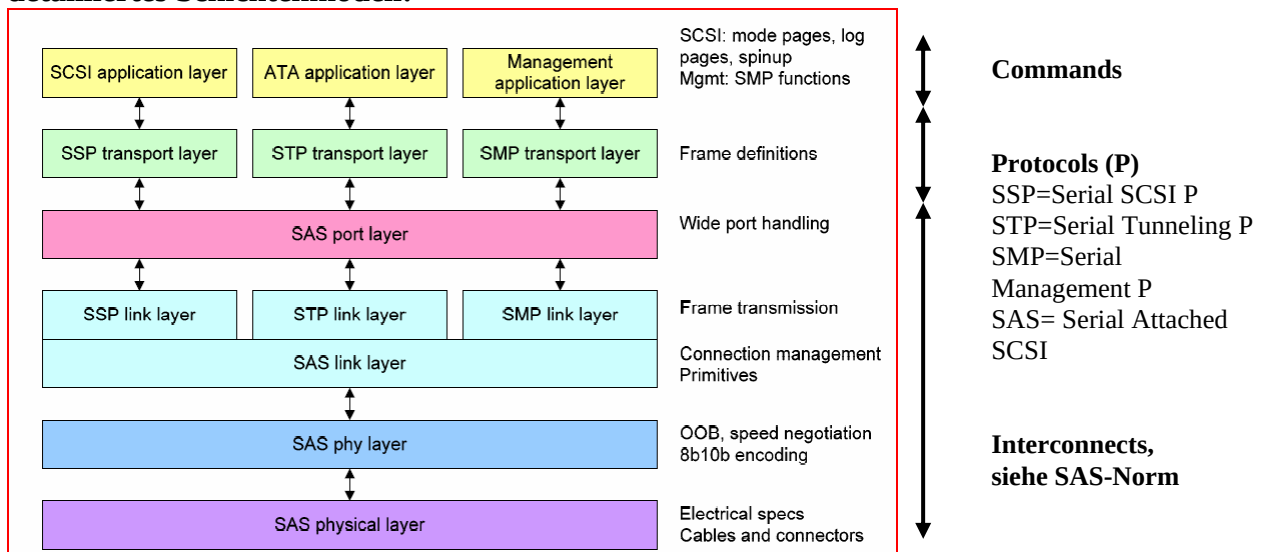
	Anforderung	SCSI Parallel	Serial Attached SCSI (SAS)
Datenrate	hoch, aber skalierbar (wegen Pinzahl) Festplatte, RAID-System, usw.	bei 16 Bit Datenbus: Ultra 320 SCSI = 320 MByte/s Ultra 640 SCSI = 640 MByte/s	300 MByte/s (1x, Narrow Port) bis 1200 MByte/s (4x, Wide Port) auf Basis von 3,0 GBit/s Übertragungsrate
Pinzahl (Kabel)	möglichst klein	68 Pins bei differentieller Übertragung (51 Signale+GNDs)	1x, Narrow Port : 7 Pins (4 Sign. +GNDs+Vcc) Anpassung an Datenrate : Wide Port
Datentransfer in Realzeit	keine Verzögerung	alle SCSI-Devices teilen sich die Busbandbreite → kurzzeitige Verzögerungen	Punkt-zu-Punkt-Verbindung, für Hin- und Rückrichtung getrennt → keine Verzögerungen
Anzahl anschließbare r Geräte		max. 15 (16 Bit Bus)	max. 16 384 Devices, Anzahl abhängig von Verbindungsnetzwerk (Expander Devices)
Konfiguration		Adreßschalter, manuell	Konfiguration automatisch, Plug & play

SAS unterstützt auch die serielle ATA-Schnittstelle (SATA).

Schichtenmodell:



detailliertes Schichtenmodell:



Cables and Connectors:	Kabel und Stecker
Electrical Specs:	elektrische Signal-Pegel
8b10b Encoding:	Codierung, siehe folgende Seiten (ff.)
OOB:	OOB Sequence, siehe ff.
Speed Negotiation:	Vorgang am Beginn, bei dem die maximal mögliche Datenrate (1,5 GBit/s, 3 GBit/s) einer Verbindung (Device--Expander--Device) festgelegt wird.
Conn. M. Primitives:	siehe ff.
Frame Tansmission:	siehe ff.
Wide Port Handling:	Verbindung zwischen Host und Geräten unterschiedlicher Port-Breiten ist möglich
Mode Pages, Log Pages :	Parameter, die der Initiator dem Target über sogenannte Mode Pages (Mode Select Command) übergibt regeln das Verhalten wie z.B. Disconnect und Reconnect des Targets. Über sog. Log Pages (Log Sense und Log Select Command) kann der Initiator Statistik- und Betriebs-Informationen vom Target abholen.

3 Protokolle sind definiert:

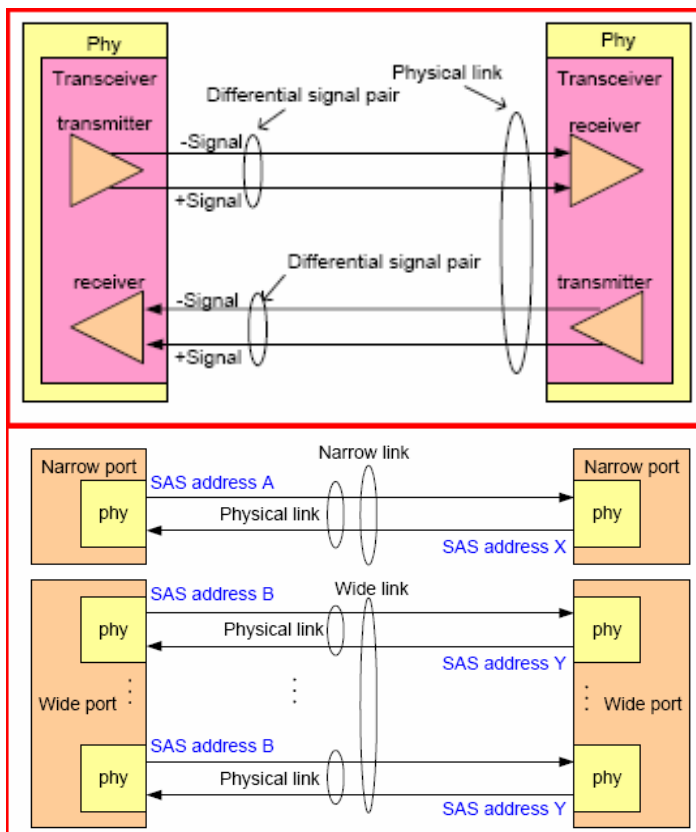
- **Serial SCSI Protocol SSP**
unterstützt serielle SCSI-Geräte wie Festplatten, Bandkassetten, optische Speicher usw.
- **Serial ATA Tunneling Protocol STP**
unterstützt serielle ATA-Geräte wie Festplatten, Bandkassetten, optische Speicher usw.
- **Serial Manangement Protocol SMP**
unterstützt SAS-Expander

4.4.1. Physikalische Ebene (Interconnects layer)

Ein Physical Link besteht aus 2 Signalpaaren (D+, D-). Daten können in beiden Richtungen gleichzeitig übertragen werden.

Codierung 8Bit → 10Bit (keine langen „0“- oder „1“-Sequenzen)

Datenrate = 1,5 GBit/s (150 MByte/s) bzw. 3,0 GBit/s (300 MByte/s) bzw. 6,0 GBit/s pro Richtung.



Ein Physical Link „Phy“ besteht aus 2 Signalpaaren (D+, D-).

Eine Verbindung (SAS-Port, Kabel) besteht aus einem oder aus mehreren Physical Links .

Maximal sind 4 Physical Links pro Verbindung möglich.

Die Datenrate beträgt dementsprechend 300 MByte/s bzw. n 300 MByte/s

Beispiele:

SAS Disk Drive: narrow Port

SAS RAID Controller: wide Port

Initial 8-bit value	Resulting 8b10b encoded value (RD-)	Resulting 8b10b encoded value (RD+)
0000 0000	100111 0100	011000 1011
0000 0001	011101 0100	100010 1011
0000 1111	010111 0100	101000 1011
1111 0000	011011 0001	100100 1110
1111 1111	101011 0001	010100 1110

Codierung 8Bit → 10Bit (einige Bitmuster)

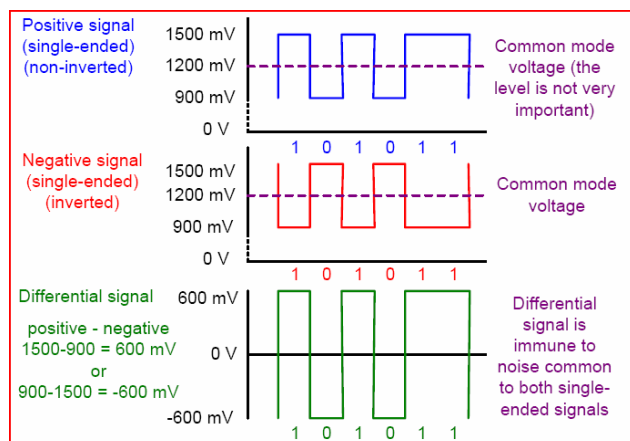
auch bei PCI Express verwendet

--Takt-Synchronisierung

--Geichspannungsausgleich (DC free Code)

--Erzeugung spezieller Bitmuster

--Fehlererkennung



Differentielle Signalisierung D+, D-:

← D+ Signal (positiv)

← D- Signal (negativ)

← Differenz-Signal im Empfänger

OOB Sequence:

Eine OOB-Sequenz (OOB Signals = Out-Of-Band Signals) ist eine definierte Anzahl von „Pause(Idle Time)-Pegelwechsel(Signal Burst)“-Folgen. Die Pegelwechsel erfolgen mit der niedrigsten Frequenz eines Ports. Dies ermöglicht es, dass Devices mit unterschiedlicher maximaler Übertragungsrate miteinander kommunizieren können

Stecker:



4.4.2. Adressierung

Byte\Bit	7	6	5	4	3	2	1	0
0	NAA (5h)				IEEE Company ID (24 bits)			
1								
2								
3								
4					Vendor-Specific Identifier (40 bits)			
5								
6								
7								

Jedes SAS Port Device (Initiator|Target|Expander Device) hat eine weltweit einzigartige 64 Bit-Adresse (SCSI Port Identifier) und einen Device Name.

(128 SAS Adressen pro Edge Expander).

Jede Logical Unit innerhalb eines SCSI Target Devices besitzt eine 64 Bit Logical Unit Number LUN (unterteilt in 4 Ebenen zu 16 Bit) und einen Logical Unit Name.

(LUN 0 ist obligatorisch)

Erstellen einer Verbindung (bei Netzwerk):

Source Phy sendet einen OPEN Address Frame (Informationsblock), der die Destination SAS Address (Zieladresse) enthält. Expander routen den OPEN Address Frame zum Destination Phy. Der Destination Phy antwortet mit einem OPEN_ACCEPT Primitive, damit ist die Verbindung erstellt.

Zum Beenden tauschen beide Teilnehmer ein CLOSE Primitive aus.

Primitive:

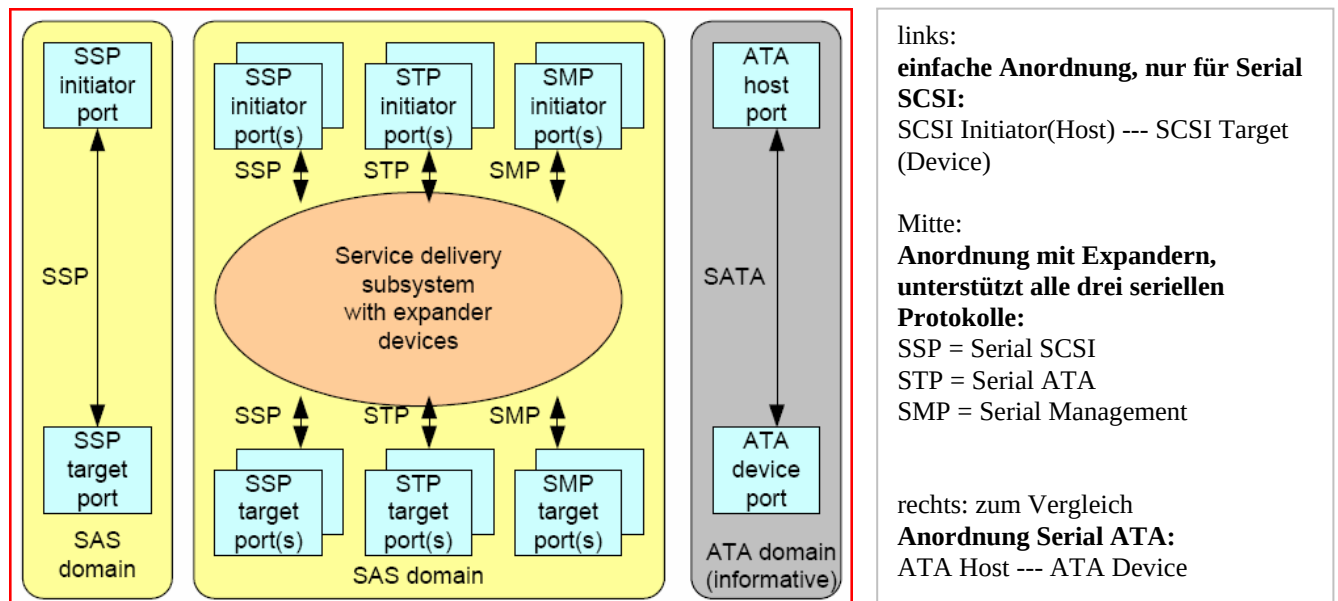
40 Bit-Folge (1 control character + 3 data character, 8/10Bit codiert)

Bei Vorhandensein von Expandern ist eine Verbindung (Connection) eine temporäre Verbindung zwischen Initiator Phy und Target Phy, geroutet über die Expander.

4.4.3. Konfiguration

hot plug-and-play

4.4.4. Architektur und Buskommunikation



Domain = I/O-System (Ein/Ausgabe-System), bestehend aus Initiator (Host) und Target (Gerät, Device) sowie gegebenenfalls einem dazwischen liegenden Netzwerk (Expander).

Der Informationsaustausch zwischen SAS Initiator Device und SAS Target Device erfolgt über „SSP Frames“. Dies sind Informationsblöcke (Information Units). Sie umfassen wie bei „SCSI Parallel“ Befehle, Daten und Status-Meldungen. Zusätzlich müssen sie die Funktion von den Steuersignalen des SCSI Parallel, die ja bei Serial SCSI wegfallen, übernehmen.

Unterschiede bei Buskommunikation zu SCSI Parallel:

Z.B. übernehmen ACK- und NAK-Primitives die Funktion der bei SCSI Parallel vorhandenen ACK- und NAK-Signale.

Der OPEN Address Frame übernimmt die Funktion der bei SCSI Parallel üblichen Adressierung des Partners (Targets) mit Hilfe des SEL-Signals.

Das CLOSE Primitive (Verbindung beenden) ersetzt die bei SCSI Parallel übliche Busfreigabe durch Wegnahme des BUSY-Signals.

Die verschiedenen SSP-Frames ersetzen die bei SCSI Parallel übliche Informationskennzeichnung mit C/D-, I/O- und MSG-Signal usw.....

Command	Information Unit field size	Direction	Description
COMMAND	28 to 284	I to T	Send a command
TASK	28	I to T	Send a task management function
XFER_RDY	12	T to I	Request write data
DATA	1 to 1024	I to T or T to I	Write data (I to T) or read data (T to I)
RESPONSE	24 to 1024	T to I	Send SCSI status (for commands) or task management response (for task management functions)

Command:
Befehl für Gerät

Task:
z.B. Abbrechen Befehl,
Rücksetzen Logical Unit,
usw.

Bild: Auflistung der „SSP Frames“

Byte	Field(s)
0	Frame Type
1 to 3	Hashed Destination SAS address
4	Reserved
5 to 7	Hashed Source SAS address
8 to 9	Reserved
10	Reserved Retransmit Rsvd
11	Reserved Number of Fill Bytes
12 to 15	Reserved
16 to 17	Tag
18 to 19	Target Port Transfer Tag
20 to 23	Data Offset
24 to m	Information Unit
m to (n-3)	Fill bytes, if needed
(n-3) to n	CRC

- One common SSP frame format
 - Frame header: 24 bytes
 - Information Unit: 0 to 1024 bytes
 - Fill bytes: 0 to 2 bytes
 - CRC: 4 bytes
- Frame format is based on Fibre Channel

Jeder Frame beginnt mit einem SOF Primitive (=Start of Frame Bitmuster) und endet mit einem EOF-Primitive

Jeder Fram hat eine fortlaufende Nummer (TAG) zur Identifizierung beim Initiator, Target und im Netzwerk.
Hashed=zerhackt

Information Unit

Bild: „SSP Frame“- Format (Serial SCSI Protocol)

Byte/Bit	7	6	5	4	3	2	1	0
0	LOGICAL UNIT NUMBER ←							
7								
8	Reserved							
9	ENABLE FIRST BURST	TASK PRIORITY				TASK ATTRIBUTE		
10	Reserved							
11	ADDITIONAL CDB LENGTH (n dwords)						Reserved	
12	CDB ←							
27								
28	ADDITIONAL CDB BYTES							
27+n×4								

LUN, erweitert gegenüber SCSI Parallel (dort nur 3 Bit)

CDB = Command Descriptor Block (Inhalt wie SCSI Parallel, Kap. 4.3.2.), enthält Befehl und Befehls-Parameter (16 Byte)

Bild: „Information Unit“ detailliert, Bestandteil eines „SSP Frame“ (Feld „24 to m“)
Die Inhalte hier betreffen den „Command Frame“ (Befehlsblock)

Aufgabe:

Zeigen Sie, ausgehend vom Command Descriptor Block CDB, den „Overhead“, der beim Übergang von SCSI Parallel zu Serial SCSI entsteht.

4.5. Aufgaben

Aufgabe SCSI-1: SCSI-3

Der SCSI-3 Standard wird auch als „Multiple Standard“ bezeichnet. Was versteht man darunter?

Aufgabe SCSI-2: SCSI Parallel

1. Erklären Sie die Begriffe "Initiator" und "Master" allgemein (Haben beide Begriffe dieselbe Bedeutung?)
Erklären Sie detailliert beide Begriffe am Befehlsablauf "CPU liest eine Datei von einer SCSI-Festplatte".
Wer ist Initiator, wer Master ?
2. Was versteht man unter "Reselection" am Bus? Nennen Sie ein Beispiel, wo "Reselection" einen Sinn macht.
3. Die Datenrate bei Ultra160SCSI beträgt 160 MByte/s. Eine Messung ergibt jedoch, dass die effektive Datenrate wesentlich niedriger ist. Nennen Sie durch den Bus bedingte Ursachen dafür! (keine genauen Zahlenwerte nötig)
4. Welche Bedeutung hat die "Device-ID" am Bus ?
5. Ermitteln Sie die Verzögerungszeit von Beginn Busbelegen bis Beginn Command Phase.
6. Vergleichen Sie die Bussteuerungen von PCI Conventional und SCSI Parallel.

Aufgabe SCSI-3: SCSI Parallel/Seriell

1. Nennen Sie die 4 Schichten des SCSI-3-Architekturmodells.
2. Welche Schichten werden beim Übergang „SCSI Parallel“ → „SCSI Seriell“ beeinflusst und welche nicht?

5. USB (Universal Serial Bus)

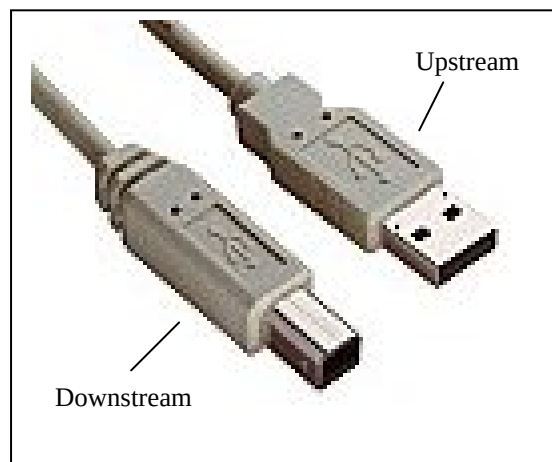
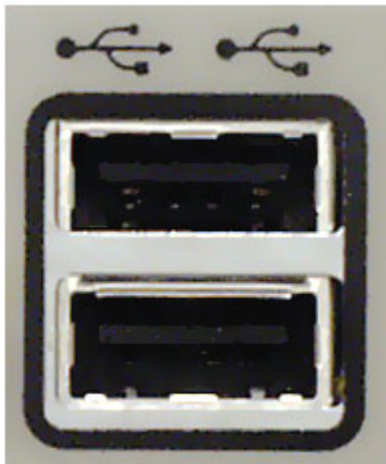
www.usb.org

5.1. Einführung

Wegen der weiten Verbreitung wurde die USB-Schnittstelle ein „**de facto**“-**Industriestandard** für den Anschluss peripherer Geräte am Computer.

In zunehmendem Maße wird eine direkte Kommunikation zwischen zwei Geräten (ohne den Umweg über einen Computer) verlangt. Dies ist mit der „**On-The-Go**“-**Erweiterung** des USB (USB-OTG) möglich.

Natürlich macht der drahtlose Anschluß auch vor dem USB nicht Halt. Dem versucht man mit der Erweiterung **Wireless USB** Rechnung zu tragen.



Merkmale:

	„Kapitel-Beginn“	„Kapitel-Ende“
Bus-Steuerung ? Bus-Zuteilung ?		
Bus-Master ? wenn ja, wie viele		
Prozessor- abhängig ?		
Maximale Teilnehmerzahl ?		
Konfiguration ?		
Datentransferrate: --MByte/s ? --optimiert durch ?		
Interrupts ?		

Der **USB** schließt Geräte wie Tastatur, Maus, Drucker, Scanner, Lautsprecher, Mikrofon, Gamepad, ISDN-Adapter, Telefon mit USB-Schnittstelle usw. am Computer an und ersetzt somit die bisherige serielle Schnittstelle RS232 (COM1, COM2) und die Parallelschnittstelle Centronics /ECP /EPP (LPT1-LPT4).

Darüberhinaus bieten USB2.0-Anschlüsse mit 480MBit/s Datenrate (=60MByte/s) ausreichende Transferraten für den externen Anschluss von Peripheriegeräten wie Festplatten oder Bandkassetten.

Transferraten:

low speed:	Mbit/s
full speed:	Mbit/s
high speed: ...	Mbit/s

Das Kabel verwendet 4-polige Stecker. Alle USB-Geräte dürfen bei laufendem Betrieb entfernt oder angeschlossen werden; die Identifikation und die Zuweisung einer Adresse am Bus erfolgen automatisch durch den USB-Hostadapter (hot plug-unplug). Die USB-Software veranlaßt das Laden der gerätespezifischen Treiber für neu hinzugekommene Geräte. Einstellungen durch den Anwender entfallen.

Maximal sind 127 Geräte (incl. Hubs) anschließbar. Inklusive dem Root Hub (im PC) darf die Baumstruktur des Busses nur sieben Ebenen umfassen (Zwischen Root Hub als oberster Ebene und einem Gerät dürfen maximal 5 Hubs hintereinander geschaltet sein).

Zwei verschiedene Steckertypen (A-, B-Stecker) stellen sicher, dass man USB-Geräte nicht zu einer unerlaubten Schleife zusammenstecken kann.

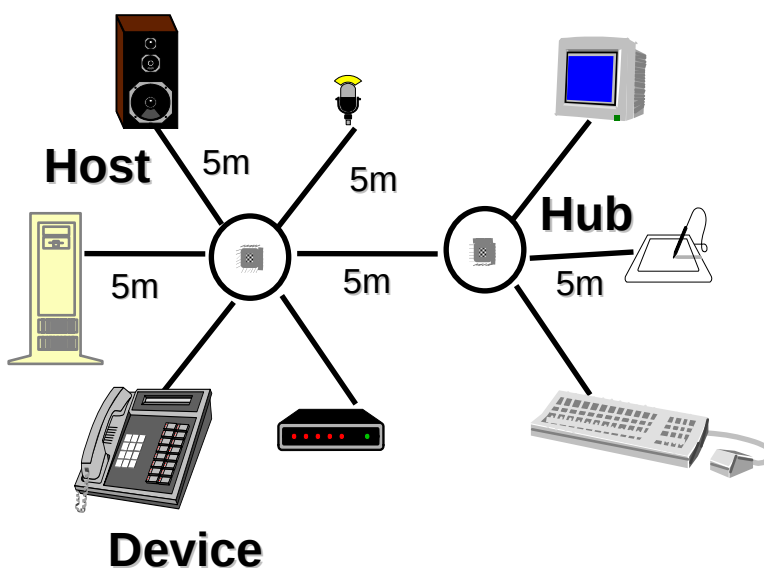


Bild: USB-Topologie

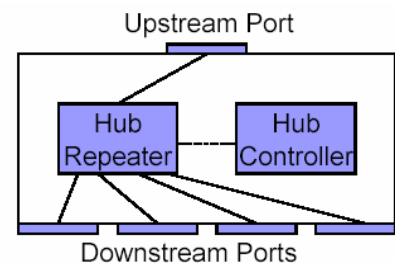


Bild: USB-Hub

Frage:

Nennen Sie notwendige Einstellungen, die der Anwender beim SCSI-Bus und bei der RS232-Schnittstelle vornehmen muß.

.....

5.2. Host Controller und angeschlossene Busteilnehmer

Host:

Der PC (=USB-Host) steuert alle Transaktionen am USB. Der USB-Hostadapter ist im PCI-Chipsatz des PC integriert. An die Ports des USB-Hostadapters sind wahlweise Geräte (Functions) und/oder Hubs anschließbar. Ein Hub ist ein Bus-Verteiler (Schnittstellen-Vervielfacher). Hubs lassen sich beliebig kaskadieren und bilden somit eine kombinierte Stern-Strang-Struktur. Jedes Draht-Segment ist eine Punkt-zu-Punkt-Verbindung Host---Gerät(Function) oder Host---Hub oder Hub---Gerät(Function) oder Hub---Hub. Jedes Gerät kann als Hub ausgeführt sein und versorgt dann weitere Geräte. Z.B. sind an einer Tastatur mit integriertem Hub weitere Geräte wie Maus und Joystick anschließbar.

Der USB-Hostadapter (=USB-Hardware im Host) wird in der Norm noch unterteilt in den "USB-Host-Controller" (Steuerteil) und den "Root Hub" (Ports).

Für den Host gibt es derzeit 3 Realisierungs-Varianten:

-- **UHCI** (Universal Host Controller Interface, Fa. Intel)

-- **OHCI** (Open Host Controller Interface, Fa. Compac, Microsoft, National Semiconductor).

Beide besitzen die gleichen Grundfunktionen. Ein UHCI besitzt zwei, ein OHCI mehrere Anschlüsse (Downstream Ports).

-- **EHCI** (Enhanced Host Controller Interface) für USB2.0-kompatible High Speed Ports.

In einem USB2.0-Host-Controller sind ein oder mehrere UHCI- oder OHCI-Controller für USB1.1 Geschwindigkeitsklassen (Low- und Full-Speed) eingesetzt (**Companion** Host Controller). Eine Routing-Logik bewirkt, dass USB1.1-Geräte mit UHCI/OHCI-Controllern und USB2.0-Geräte mit dem EHCI-Controller verbunden werden können. Erkennt ein USB2.0-Treiber ein angeschlossenes Gerät als Full- oder Low-Speed-Gerät, konfiguriert er die Routing-Logik so, dass dieses Gerät mit den Ports eines Companion USB Host Controllers verbunden wird.

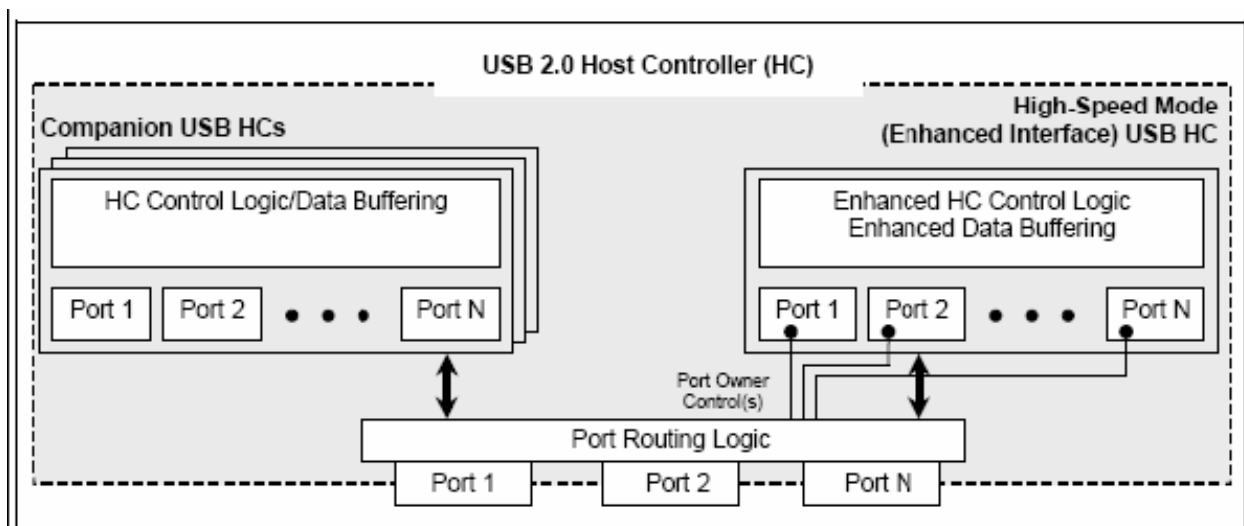
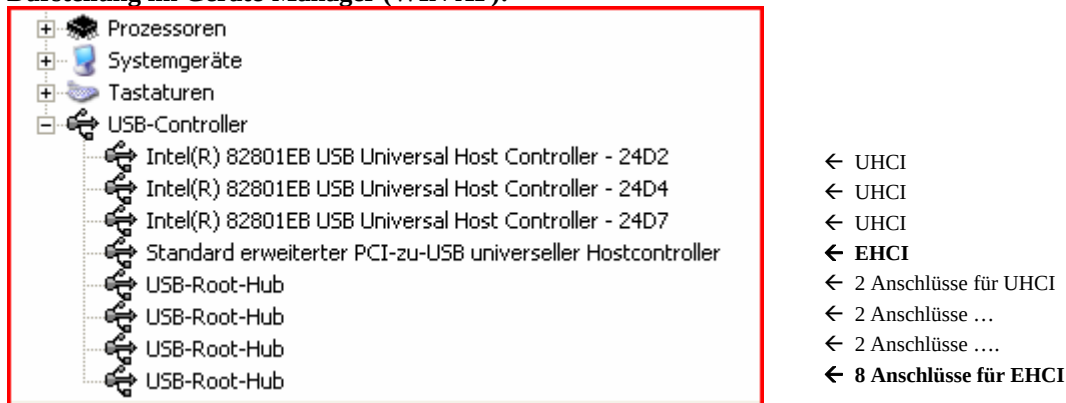


Bild: Host-Controller nach dem EHCI-Standard

Erkennt der USB2.0-Treiber ein angeschlossenes Gerät als Full- oder Low-Speed-Gerät, dann konfiguriert er die Port Routing Logic so, dass das Gerät mit dem Port eines USB1.1-Host Controllers verbunden wird (im Bild: linke Seite Companion USB HC)

Verfügt das System nur USB1.1-Controller, dann werden High-Speed-Geräte an einem Port des USB1.1-Controllers angeschlossen (High-Speed-Geräte müssen auch den Full-Speed-Modus unterstützen).

Darstellung im Geräte-Manager (WIN XP):



Geräte (Functions):

Angeschlossene Geräte (Tastatur, Maus, Drucker usw.) werden Funktionen (Functions) genannt. Nach dem Einschalten besitzt ein Gerät die Default-Adresse 0. Bei der Initialisierung - hier Enumeration genannt - wird jedem Gerät eine eigene Adresse zugewiesen. Jedes Gerät besitzt entweder eine eigene Spannungsversorgung (self powered) oder es wird über das USB-Kabel versorgt (bus powered). Vor der Initialisierung darf die Stromaufnahme aus dem USB-Kabel maximal 100mA und nach der Initialisierung maximal 500mA betragen.

Geräte-Klasse HID (human interface device): einheitlicher Treiber für Maus, Tastatur, Joystick usw.

Hub:

Ein Hub ermöglicht den Anschluß mehrerer Geräte. Jeder Hub besitzt einen Upstream-Port in Richtung Host und mehrere Downstream-Ports in Richtung der Geräte oder weiterer Hubs. Vom Host gesendete Daten leitet der Hub an alle seine aktiven Downstream-Ports weiter. Daten vom Gerät sendet der Hub nur in Richtung Host.

Ein Hub isoliert angeschlossene Full/ Low-Speed-Geräte vom High-Speed-Mode und leitet an diese Geräte gerichtete Daten in der passenden Geschwindigkeit weiter. Außerdem übernimmt der Hub das Power-Management für die angeschlossenen Geräte. Hubs besitzen eine eigene Spannungsversorgung oder werden über das USB-Kabel versorgt.

5.3. Bus-Kommunikation

Der USB ist ein Polling-Bus (Polling=Abfrage). Das Polling wird vom Host-Adapter eigenständig durchgeführt. Zur zeitlichen Synchronisation der Geräte sendet der Host periodisch ein **Paket SOF** (Start-of-Frame-Token) bzw. **uSOF** auf den Bus. Damit wird der Bus in Zeitintervalle (Frames) von **1ms (SOF bei Full Speed)** bzw. **von 0,125ms (uSOF bei High Speed)** eingeteilt. SOF (uSOF)-Tokens entfallen bei Low-Speed-Anschlüssen. Nicht jedes Gerät überträgt Daten in jedem Frame. Der Host-Controller-Treiber verteilt die Datentransfers für die verschiedenen Geräte auf die einzelnen Zeitintervalle (Scheduling der Transaktionen).

Ein USB-Gerät kann von sich aus weder einen Transfer starten noch per Interrupt einen Transfer anfordern. Mit einem Trick lässt sich jedoch für ein Gerät quasi eine Interrupt-Fähigkeit herstellen: Ordnet man einem Gerät einen Interrupt-Endpunkt zu, dann wird es periodisch im Millisekundenabstand vom Host abgefragt und kann so Transferanforderungen dem Host bekanntmachen.

Der Hostadapter selbst ist jedoch interrupt-fähig, d.h., er kann von der CPU Bedienung anfordern.

Das Anwender-Programm startet einen Datentransfer, indem es beim USB-Geräte-Treiber einen Transfer anfordert. Es wird eine Verbindung ("Pipe") etabliert und die Daten werden übertragen. Damit keine Daten „verloren gehen“, muß jedes USB-Gerät einen Datenpuffer haben, dessen Größe sich nach dem typischen Datenaufkommen und nach seiner Datentransferrate richtet. Ebenso sind Pufferspeicher auf der Host-Seite notwendig, um Daten eines Geräts zwischenspeichern zu können für den Fall, dass die Daten nicht schnell genug zum Arbeitsspeicher der CPU übertragen werden (Puffer-Speicher im Host Adapter siehe DEMO PCI-Abfrage).

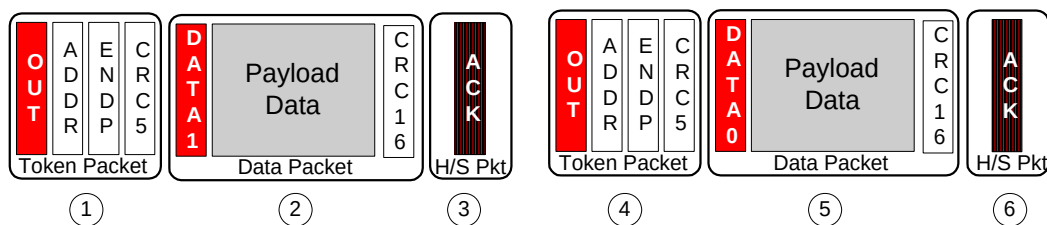
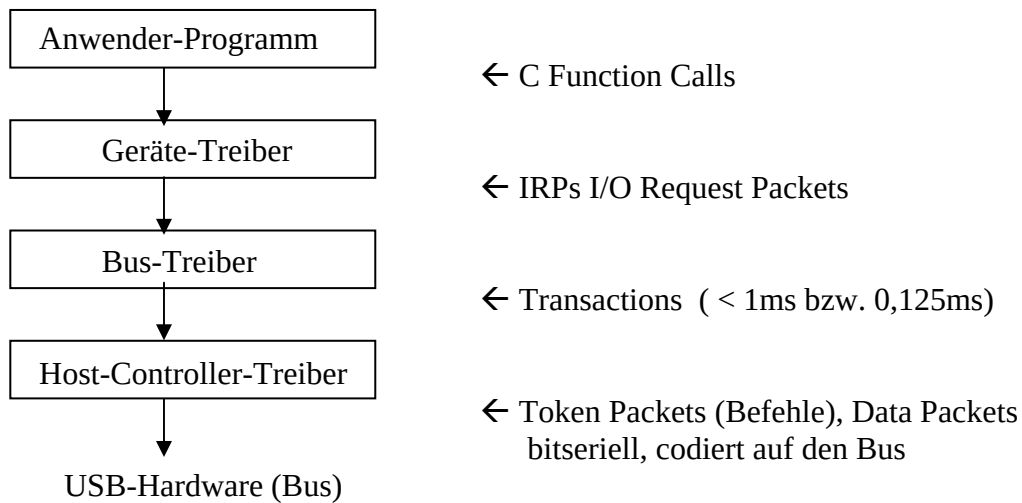


Bild: Informationstransfer am USB: Host sendet Daten an Gerät(Function)

Der Datentransfer vom Host zu den Geräten erfolgt im sog. Broadcast-Modus, d.h., die Daten werden über die Hubs an alle Geräte verteilt. Nur das Gerät, das die eigene Adresse im Token-Paket erkennt, antwortet.

Eine direkte Kommunikation zwischen zwei Geräten ohne Beteiligung des Host ist nicht möglich. Ein Datentransfer zwischen zwei Geräten erfolgt grundsätzlich über den Host.

5.4. Software-Schichten



Anwender-Programm:

Es stößt mit Funktionsaufrufen (z.B. C-Funktionsaufrufe wie Readfile/ Writefile, definiert für die Geräte-Treiber-Schnittstelle) über den Geräte-Treiber einen Datentransfer an.

Geräte-Treiber (Device Driver):

Er setzt die Funktionsaufrufe in USB-Befehle (USB-Requests) um und sendet diese an den Bus-Treiber in Form von I/O-Request-Packets IRPs und stellt einen Pufferspeicher für die Übertragungsdaten bereit. Ein IRP initiiert einen Datentransfer mit einem Gerät.

Device-specific-USB-Requests können z.B. Commands aus einem SCSI Command Set sein (z.B. ein Read Command für Mass Storage Devices).

Geräte einer Klasse können von einem einheitlichen Treiber bedient werden.

Bus-Treiber (Bus Driver):

Er "zerstückelt" die IRPs eines Datentransfers in einzelne Transaktionen (transactions). Dies ist notwendig, da der Busbetrieb in 1ms-Zeitintervalle eingeteilt ist und jeder (Teil)-Transfer daher maximal nur 1 ms bzw. 0,125 ms dauern darf und außerdem der Bus von mehreren Geräten "gleichzeitig" benützt werden kann. Eine Transaktion muß innerhalb eines 1ms-Zeitfensters auf dem Bus durchführbar sein. Bei 12MBit/s sind maximal ca. 1500 Byte in einem 1 ms-Rahmen, bei 480MBit/s sind 7500 Byte in einem 0,125ms-Rahmen übertragbar.

Host-Controller-Treiber (Host Controller Driver):

Der Host-Controller-Treiber greift auf die Hardware zu. Er stößt die Transaktionen über die Host Controller Hardware (Host-Adapter) an. Diese setzt jede Transaktion in einzelne Pakete (Token- und Data-Packets) um und sendet die NRZI-codierten Daten bitseriell auf die Busleitung (diese Token- und Data-Packets sind nicht zu verwechseln mit den I/O-Request-Packets IRPs).

Für jede Transaktion erstellt der Host-Controller-Treiber im Arbeitsspeicher einen Transaktions-Deskriptor (Festlegungen für den Transfer).

Transaktions-Deskriptor:

- Adresse des angesprochenen Geräts (Geräteadresse + Endpunktadresse innerhalb des Geräts)
- Transfer-Art (Control-, Interrupt-, Bulk- oder Isochronous-Transfer)
- Transfer-Richtung (IN, OUT)
- Pufferspeicher-Adresse im Host-Arbeitsspeicher für die Daten

Der Inhalt des Transaktions-Deskriptors entspricht dem Endpunkt-Deskriptor (Endpoint Descriptor, siehe Kap.) des jeweiligen Geräts. Nur die Pufferspeicher-Adresse wird vom Geräte-Treiber hinzugefügt.

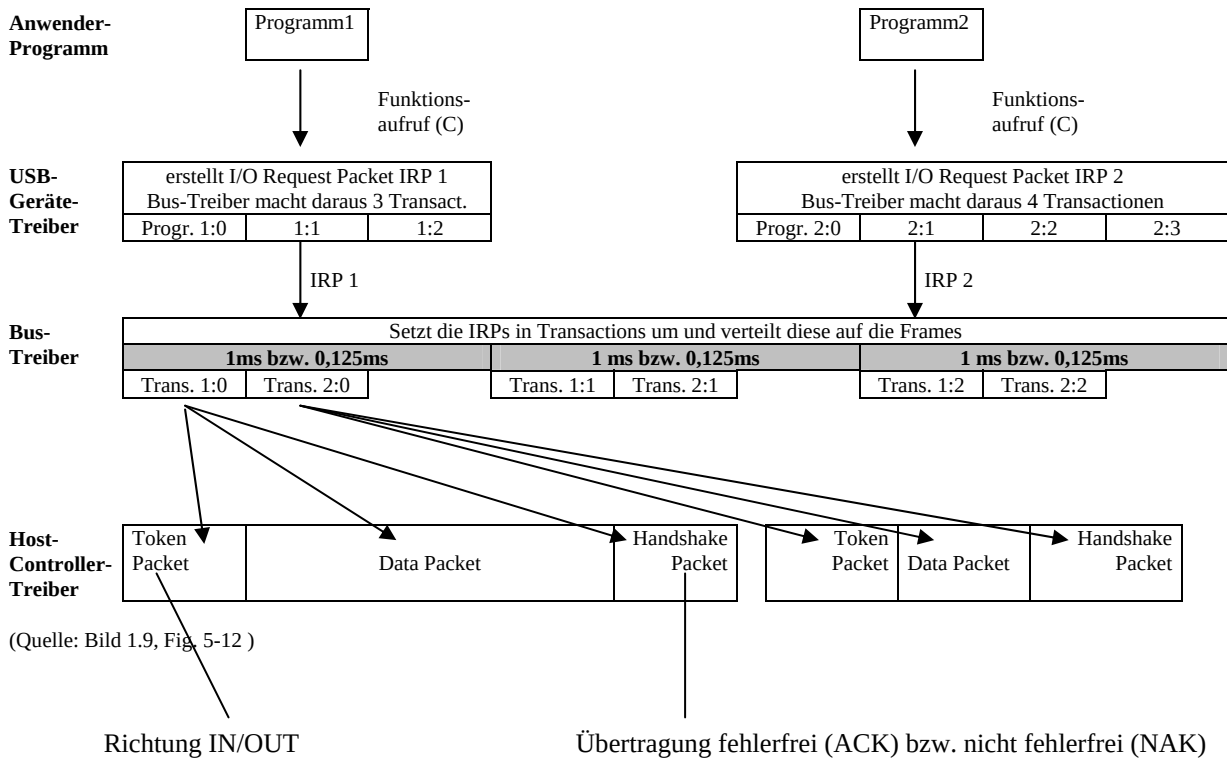
Im Pufferspeicher sind die zu übertragenden Daten für OUT- und die Befehle für SETUP-Transfers (siehe Kap.) bereitgestellt oder es werden dort die beim IN-Transfer erhaltenen Daten abgespeichert.

Der Host-Controller-Treiber organisiert die zeitliche Abfolge der einzelnen Transaktionen auf dem Bus (Scheduling). Dazu erstellt er für jedes 1ms-Zeitfenster eine Transaktions-Liste (Frame-List). Da ein Datentransfer aus mehreren Transaktionen bestehen kann, kann dieser auf mehrere Zeitfenster verteilt sein.

Bild: Abarbeitung zweier Anwender-Funktionsaufrufe am Bus, dargestellt in den verschiedenen Ebenen:

- Transfer-Ebene
- Transaktions-Ebene
- Paket-Ebene

Diese Ebenen sind auch mit Hilfe eines USB-Analysators darstellbar.



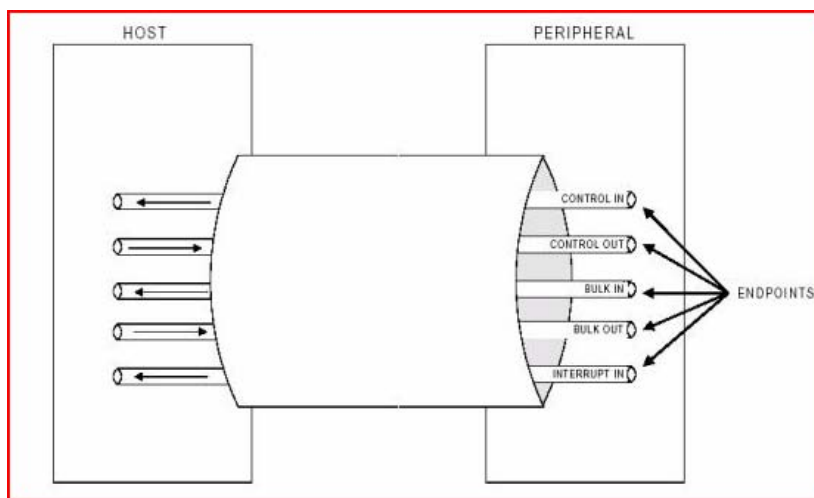
DEMO: „Aufzeichnung der Paketebene mit dem SCSI-Analyzer “
Diskussion

5.5. Informations-Transfer

Die Informationen werden, wie schon beschrieben, in Form von Daten-Paketen übertragen. Der Begriff "Information" umfaßt alle Arten von Informationen, also Daten für die Steuerung und die Initialisierung der Geräte sowie Daten im engeren Sinn wie z.B. Druckerdaten usw.!

Pipes:

Der Datentransfer zu einem USB-Gerät erfolgt physikalisch über zwei Signaladern (D+, D-) im Kabel. Jeder Transfer ist über Transaktions-Deskriptoren beschrieben, man spricht auch von virtuellen Datenkanälen, sogenannten "Pipes", die der Host einrichtet. Jede Pipe beginnt im Pufferbereich des Arbeitsspeichers und endet in einem Endpunkt (Endpoint EP) eines Geräts. Ein Endpunkt im Gerät ist ein Sende/Empfangs-FIFO. Die Eigenschaften eines Endpunkts sind im Endpoint Descriptor festgelegt. Dieser ist in jedem Gerät fest abgespeichert.



Endpunkt einer Pipe:
Pufferbereich im Arbeitsspeicher

Endpunkt einer Pipe:
Datenpuffer (FIFO) im Gerät

Bild:
Virtuelle Datenkanäle (Pipes)
zwischen Host und Gerät
(Peripheral)

im Bild:

- "Control In" Pipe
- "Control Out" Pipe
- "Bulk In" Pipe
- "Bulk Out" Pipe
- „Interrupt In“ Pipe

Die Informationstransfers über die "Pipes" erfolgen zeitlich nacheinander auf dem Kabel (in Form von Datenpaketen).

Der USB unterstützt zwei unterschiedliche Pipe-Konzepte:

- Message Pipes** (definiertes Protokoll)
- Stream Pipes** (freies Protokoll, aber auch Pakete)

Message Pipes übertragen Befehle (Standard-Device-Requests) und Abfragedaten (Deskriptoren) und besitzen eine in der USB-Spezifikation definierte Struktur, bestehend aus 3 Paketen:

- Dem "Request"-Paket (Bus-Befehl) vom Host,
- einer Datenübertragung und
- dem Quittungs-Paket vom USB-Gerät.

Die Firmware im USB-Gerät interpretiert die vom Host gesendeten Daten und quittiert dem Host den Empfang. Message Pipes verwenden sogenannte "Control-Transfers".

Stream Pipes übertragen Datenströme, deren Struktur in der USB-Spezifikation nicht definiert ist. Sie transportieren die Daten ohne festes Format, jedoch in Paketen (Daten-Header, Daten-Ende). Über Stream Pipes laufen "Interrupt-Transfers", "Isochrone Transfers", und "Bulk-Transfers". Ein quasiparalleler Datentransfer mit mehreren Geräten ist möglich, indem die Geräte ihre Datenpakete abwechselnd mit maximaler Datenrate übertragen. Während ein Gerät überträgt sammeln die anderen ihre Daten im Datenpuffer.

Adressierung:

Jedes USB-Gerät kann mehrere Endpunkte und mehrere Pipes unterstützen. Die Adressierung auf dem Bus erfolgt über eine 7Bit-USB-Adresse und im Gerät über eine 4Bit-Endpunkt-Nummer. Am Bus sind dadurch maximal 127 Geräte und Hubs anschließbar. Die Endpunkt-Nummer ermöglicht die Realisierung verschiedener adressierbarer Funktionen innerhalb eines USB-Geräts. Ein Endpunkt ist eindeutig definiert durch seine Nummer und seine Richtung. Pro Gerät sind daher neben dem EP0 noch 15 IN- und 15 OUT-Endpunkte möglich. Bezugspunkt für die Richtung des Datentransfers ist der Host:

--**Out:** vom Host zum Gerät (**downstream**)

--**In:** vom Gerät zum Host (**upstream**)

Alle Geräte müssen einen Control Endpoint EP0 unterstützen. Über EP0 wird ein Gerät initialisiert und gesteuert. Über diesen Endpunkt können Steuerungs- und Abfrage-Daten gesendet und empfangen werden. Alle anderen Endpunkte sind uni-direktional (senden oder empfangen). Die im EP0 übertragenen Daten sind als Standard-Device-Requests definiert. Die zum EP0 aufgebaute Pipe ist eine "Message" Pipe.

Low-Speed-Geräte unterstützen nur Control- und Interrupt-Endpunkte.

Transferarten:

-- **Control-Transfers:**

Transfers für die Konfiguration und Steuerung der USB-Geräte werden als Control-Transfers bezeichnet. Sie können in beiden Übertragungsrichtungen ablaufen und beginnen (bzw. enden) im Gerät immer im Endpunkt EP0. Die mit EP0 aufgebaute Pipe wird als Message Pipe bezeichnet.

Bspl. 1: Geräteadresse zuweisen bei der Konfiguration (Standard Device Request):

SETUP-Token (Befehlspaket) vom Host (enthält Geräteadresse 0 und Endpunkt-Adresse EP0)

Datenpaket vom Host (enthält u.a. die neue Geräteadresse)

ACK-Handshake-Paket an Host (Übertragung fehlerfrei)

IN-Token vom Host (enthält Geräteadresse 0 und Endpunkt-Adresse EP0)

0Byte-Datenpaket an Host („konnte Befehl ausführen“)

ACK-Handshake-Paket vom Host

Bspl. 2: Lesen eines Device Descriptors:

SETUP-Token (Befehlspaket) vom Host (enthält Geräte- und Endpunkt-Adresse)

Datenpaket vom Host (enthält Befehlsparameter „Descriptor abfragen“)

ACK-Handshake-Paket an Host

IN-Token vom Host (enthält Geräteadresse 0 und Endpunkt-Adresse EP0)

Datenpaket an Host mit dem Device Descriptor

ACK-Handshake-Paket vom Host

Die Datenübertragung beim Control-Transfer ist über eine integrierte Fehlerkorrektur und über eine Handshake-Prozedur besonders gesichert.

FIFO-Grösse im Endpunkt: 8 Byte (Low Speed), 8-64 Byte (Full Speed), 64 Byte (High Speed).

Die folgenden 3 Transfers beginnen (bzw. enden) in den Endpunkten EP1 bis EP15 und sind unidirektional. Die dazugehörigen Pipes werden als Stream Pipes bezeichnet:

-- Interrupt-Transfers:

Bei Geräten, bei denen anfallende Daten zeitkritisch sind, d.h., sofort übertragen werden müssen, wäre normalerweise ein Interrupt-Meldverfahren zum Host notwendig. Eine Interrupt-Verarbeitung gibt es jedoch am USB nicht! Man hat dieses Problem dahingehend gelöst, dass solche Geräte periodisch vom Host abgefragt werden. Typische Anwendungen sind Maus, Tastatur, Joystick usw. Das Polling-Intervall (Abfrage-Intervall), das ein Gerät benötigt, ist im Endpoint Descriptor eines Geräts definiert. Ein Polling-Intervall von z.B. 8ms besagt, dass der Host alle 8ms einen Datentransfer mit dem Gerät durchführt. Hat das Gerät keine Daten, dann quittiert es die Anforderung mit einem NAK-Handshake-Paket

Zulässige Polling-Intervalle:

Low/Full Speed Geräte: 1ms bis 255ms

spezifiziert im Endpoint Descriptor
als Vielfache von 1ms

High Speed 0,125ms bis ..??

spezifiziert im Endpoint Descriptor
als Vielfache von 0,125ms

(siehe Unterlage „Basisinformationen“ zu den Praktikumsversuchen)

Bei Übertragungsfehlern versucht der Host eine Wiederholung des Transfers in der nächsten freien Transferperiode.

Paketfolge z.B.

IN-Token (Befehlspaket) an das Gerät (enthält Geräte- und Endpunkt-Adresse)

Datenpaket DATA0/1 vom Gerät

ACK-Handshake-Paket vom Host (Übertragung fehlerfrei)

Max. FIFO-Grösse im Endpunkt: 8 Byte (Low Speed), 64 Byte (Full Speed), 3072 Byte ? (High Speed, $3072 = 3 \times 1024$). Die maximale Datenpaketgrösse innerhalb eines 125us Microframes bei High Speed beträgt Theoretisch 1024 Byte. Grundsätzlich sollen Interrupttransfers aber nur zur Übertragung kleiner Datenmengen verwendet werden!
{480 MBit/s x 125us = 62 914 Bit = 7 864 Byte}

-- Isochrone (=zeitgenaue) Transfers:

Eine weitere Übertragungsart für zeitkritische Daten ist der isochrone Transfer. Er wird z. B. für die Echtzeitübertragung von Sprache und Musik verwendet. Isochron bedeutet, dass die Daten periodisch vom Host transferiert werden. Dies erfolgt wie beim Interrupttransfer in jedem Frame (1ms) bzw. Mikroframe (125us) oder mit einem Polling-Intervall (=Vielfaches der Intervall-Länge), das im Endpoint-Deskriptor spezifiziert ist.

Im Unterschied zum Interrupttransfer findet im Fehlerfall keine Wiederholung des Transfers statt, d.h., kein Handshake durch das Empfangsgerät (verspätete Daten sind unnütze Daten).

Ansonsten würde es Unterbrechungen geben, wenn z.B. der Datenpuffer eines USB-Lautsprechers leer wird oder der Datenpuffer eines USB-Mikrophons überläuft. Bei der Übertragung von z.B. Audio-Daten zu Lautsprechern entsteht kein Qualitätsmangel durch den Verlust einzelner Samples (Abtastwerte).

Max. FIFO-Grösse im Endpunkt: 1023 Byte ? (Full Speed), 3072 Byte ? (High Speed).

Paketfolge z.B.

OUT-Token (Befehlspaket) an das Gerät (enthält Geräte- und Endpunkt-Adresse)

Datenpaket DATA0/1 an das Gerät

Isochrone Transfers sind nicht für Low-Speed-Geräte spezifiziert.

-- **Bulk-Transfers:**

Geräte mit großen Datenmengen benutzen Bulk-Transfers. Die Daten dürfen nicht zeitkritisch sein, da kein Polling-Intervall festgelegt wird.

Bulk-Transfers sind nicht für Low-Speed-Geräte spezifiziert.

Im Fehlerfall wird vom Host die Datenübertragung wiederholt.

Max. FIFO-Grösse im Endpunkt: 64 Byte (Full Speed), 512 Byte (High Speed).

Die maximale Datenpaketgrösse entspricht der FIFO-Grösse.

Paketfolge z.B.

OUT-Token (Befehlspaket) an das Gerät (enthält Geräte- und Endpunkt-Adresse)

Datenpaket DATA0/1 an das Gerät

ACK-Handshake-Paket vom Gerät (Übertragung fehlerfrei)

Die einem Endpoint zugeordnete Transferart ermittelt der Gerätetreiber aus den jeweiligen Deskriptoren (Gerät "sagt", welchen Transfer es benötigt).

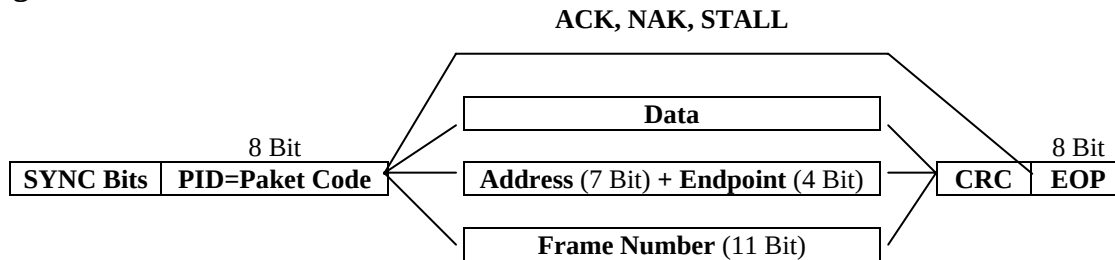
Nicht alle Geräte unterstützen alle 4 Transferarten.

Interrupt-Transfers und isochrone Transfers belegen statisch Busbandbreite. Gegebenenfalls lehnt der Host die Konfiguration weiterer Geräte ab, wenn dadurch der Bus „überbelegt wäre“.

5.6. Paket-Formate

Informationen werden in Form von Daten-Paketen übertragen.

Allgemeines Paket Format:



Synchronisier-Bits:

Jedes Paket beginnt mit einem Synchronisierfeld (7x"0" und 1x"1", bei High Speed: 31x"0" und 1x"1"). Die "0"-Serie liefert wegen der NRZI-Codierung Polaritätswechsel (selbsttaktender Code, siehe Kap.), mit denen der Takt im USB-Gerät synchronisiert wird. Mit diesem Gerätetakt wird der serielle Datenstrom auf den Leitungen D+, D- abgetastet.

Paket Code PID:

Jedes Paket enthält einen 4Bit-Paket-Identifizier PID und den invertierten Wert /PID als Redundanz. Entsprechend dem PID werden die nachfolgenden Daten im Empfänger verschieden interpretiert. Grundsätzlich wird am USB die Information beginnend mit LSB übertragen.

Optionale Elemente:

Folgende Elemente sind je nach Paket-Typ in den Paketen halten:

FRAME#	Fortlaufende 11Bit-FRAME-Nummer. Ist der maximale Wert 7FF erreicht wird wieder mit 000 begonnen. Nur im START-Paket SOF. <u>uSOF</u> : nur jedes achte uSOF inkrementiert die FRAME#. 8 Mikroframes innerhalb eines Frames verwenden dieselbe FRAME#.
Address	7Bit-Geräte-Adresse
Endpoint	4Bit-Endpunkt-Nummer innerhalb eines Geräts
CRC	Redundante Information als Datensicherung des Pakets.
EOP	End of Packet (Low/Full Speed: SE0-Signalisierung, High Speed: 01111111 ohne Bit-Stuffing)

Paket-Name	Funktion	PID(3..0)= Paket Code	PID-Byte	Quelle
Token Packets:				
SOF Start of Frame, uSOF	Start-Paket, markiert den zeitlichen Beginn eines Rahmens (Frame). SOF wird ersetzt durch Keep-Alive -Signalisierung bei Low-Speed-Geräten	0101	A5	Host
SETUP	Befehlspaket transportiert USB Requests (Befehle), siehe Kap. "USB-Befehle"	1101	2D	Host
IN	„ Richtungs -Pakete“ leiten Daten-transfer ein: IN = Gerät → Host	1001	69	Host
OUT		0001	E1	Host
Data Packets:				
DATA0	Datenpakete , enthalten die Nutzdaten, toggeln	0011	C3	Host, Gerät
DATA1		1011	4B	Host, Gerät
DATA2	nur High Speed, spez. Anwendung bei isochronen Endpunkten und Split Transactions			
MDATA				
Handshake Packets:				
ACK	Handshake -Pakete, quittieren den Empfang ()	0010	D2	Host, Gerät
NAK		1010	5A	Gerät
STALL	Gerät fordert Hilfe vom Host	1110	1E	Gerät
NYET	=Not YET, nur für High Speed Geräte Nach dem Empfang eines Datenpakets kann ein Geräteendpunkt ein NYET zurückgeben, was heisst, dass das Gerät die Daten empfangen hat aber für neue Daten noch nicht bereit ist.	0110	96	Gerät
Special Packets:				
SPLIT	High Speed Bus: SPLIT leitet Transfer zu einem Low oder Full Speed Gerät ein. (2.0-Hosts und 2.0-Hubs übertragen Low und Full Speed Datenverkehr im High Speed Modus) SSPLIT : Parameter SC=0 Start SPLIT CSPLIT : Parameter SC=1 Complete SPLIT	1000	78	Host
ERR	Wird nur in Split-Transaktionen verwendet, um dem Host Fehler zu melden.	1100	3C	Gerät, Hub
PRE	Full Speed Bus: PRE leitet Transfer zu einem Low Speed Gerät ein	1100	3C	Host
PING	Prüft bei Bulk OUT Transfers, ob ein Gerät noch beschäftigt ist (geht schneller, als ein NAK nach einem erfolglosen Bulk OUT abzuwarten).	0100	B4	Host

(engl. "Frame" = Rahmen, "Token" = Zeichen, Merkmal, Marke)

Start-Paket:

Der Host sendet periodisch ein Start-of-Frame-Token **SOF** als Zeitmarke auf den USB und erzeugt damit Zeitfenster: **1ms** im Full Speed Mode (**SOF**), **0,125 ms** im High Speed Mode (**uSOF**).

Die Frame Number (11 Bit) wird bei jedem SOF hochgezählt (modulo 2048). Im High Speed Mode erhöht der Host die Frame Number nur nach jedem achten uSOF.

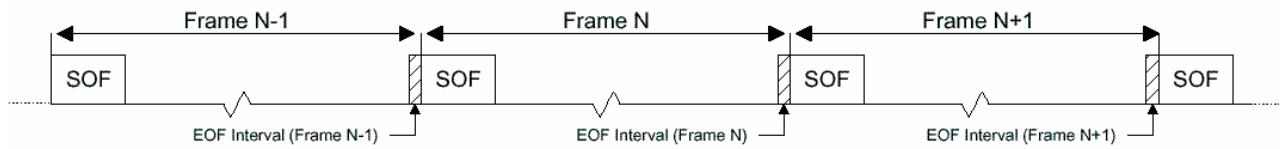
Das SOF wird im Full-Speed-Mode gesendet. Low-Speed-Ports können daher kein SOF empfangen, deshalb wird das SOF vom Hub in eine spezielle Signalisierung EOP umgewandelt (Low-Speed-**Keep-Alive**: Auf die beiden zum Low-Speed-Gerät führenden Signaladern D+/ D- wird für 2 Low-Speed-Taktzyklen Low-Pegel angelegt).

Ein Gerät, das kein SOF oder Low-Speed-Keep-Alive EOP empfängt, geht in den Schlaf-Modus (Suspend Mode) mit verringerter Stromaufnahme über:

Maximale Stromaufnahme eines Geräts am Bus:	500mA konfiguriert
	100mA unkonfiguriert
	500uA im Suspend Mode

Das **SOF**-Token geht an alle Geräte.

Zeitraumen (1ms Frames) am USB: (fig. 10-3)



Befehls-Pakete (SETUP):

SETUP -Token enthalten eine Geräteadresse und eine Endpunktadresse. Nur Geräte, die ihre Adresse in einem Paket erkennen, dürfen reagieren. Dasselbe gilt für **IN**- und **OUT**-Token.

Daten-Pakete:

Die Nutzdaten werden in **DATA0**- und **DATA1**-Paketen übertragen. Sind es größere Datenmengen, werden – aus Gründen einer eventuellen Fehlerbehandlung - immer abwechselnd DATA0- und DATA1-Pakete verwendet. Maximal sind pro Paket 1024 Byte Nutzdaten (Full Speed) bzw. xxxx Byte (High Speed) übertragbar. Die Nutzdaten sind über eine 16Bit-CRC-Information gesichert.

Der Empfang von Nutzdaten wird über ein **ACK**-Paket (erfolgreich) oder eine **NAK**-Packet (nicht erfolgreich) quittiert. Über das **STALL**-Paket fordert ein Gerät die Hilfe des Host an wenn es z.B. weder Daten empfangen noch senden kann.

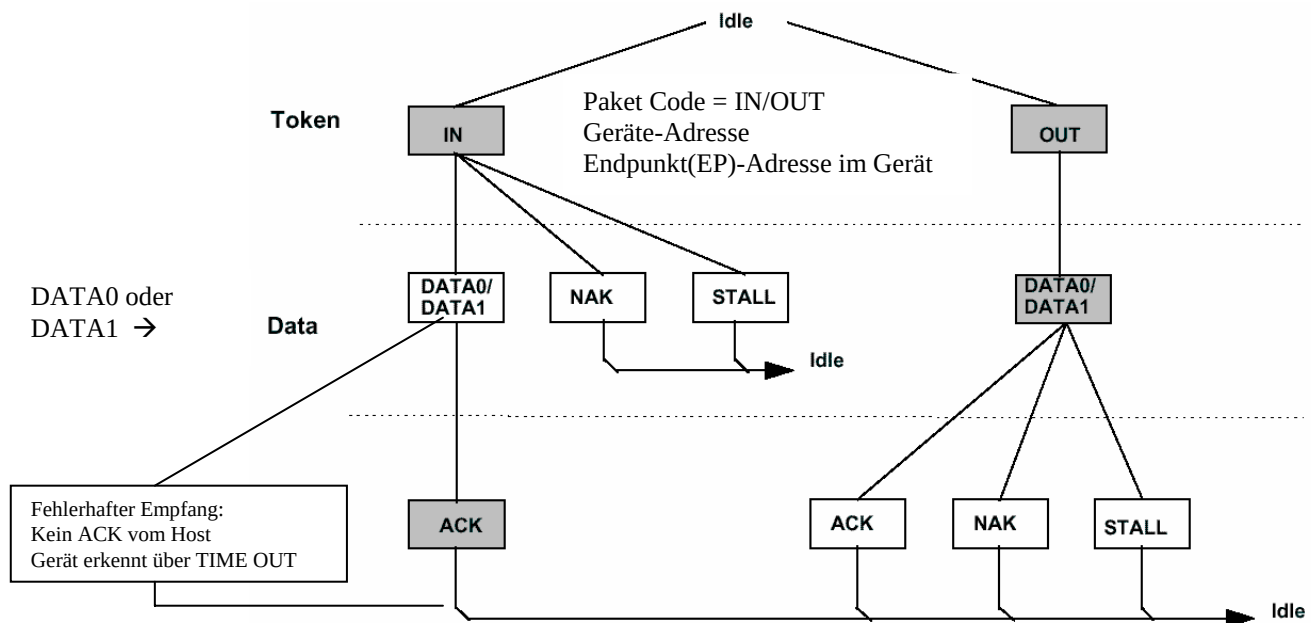
5.7. Phasen einer USB-Transaktion (Transaktions-Formate)

Jede Transaktion von Daten am Bus besteht aus sogenannten „Phasen“. In der ersten Phase (Token-Phase) wird ein Token-Paket vom Host gesendet, da nur er einen Transfer beginnen kann. Nach dem Daten-Paket (Daten-Phase) wird in der Handshake-Phase der Empfang des Daten-Pakets mit einem ACK-Paket quittiert, z.B.:

OUT-Paket	HOST → Gerät (Gerät weiß, "jetzt kommen Daten")
DATA0/DATA1-Paket	Daten Host → Gerät (vorheriges "Out" hat Richtung festgelegt)
ACK-Paket	"Empfangsbestätigung", Host ← Gerät

Datentransfer:

Beispiel **Bulk Transaction** (für Übertragung großer Datenmengen):



IN-Token:

Das IN-TOKEN (enthält Geräteadresse und EP-Adresse) veranlasst das Gerät, ein Datenpaket sofort nach dem EOP des IN-TOKENs zu senden. Die Übertragung der Daten beginnt immer mit einem DATA0-Paket. Jedes einzelne Datenpaket wird mit einem IN-TOKEN angefordert.

Fehlerfreier Empfang: Host quittiert mit einem ACK-Paket.

Fehler: Host sendet keine Quittierung. Das Gerät erkennt dies über einen Time-Out-Fehler. Für eine nochmalige Übertragung muss der Host erneut ein IN-TOKEN senden.

NAK: Gerät „habe keine Daten mehr“.

STALL: „Hilferuf“ an Host

Beispiel Maus: wird periodisch (8ms) abgefragt. Falls keine neuen Positionsdaten → Maus antwortet mit NAK.

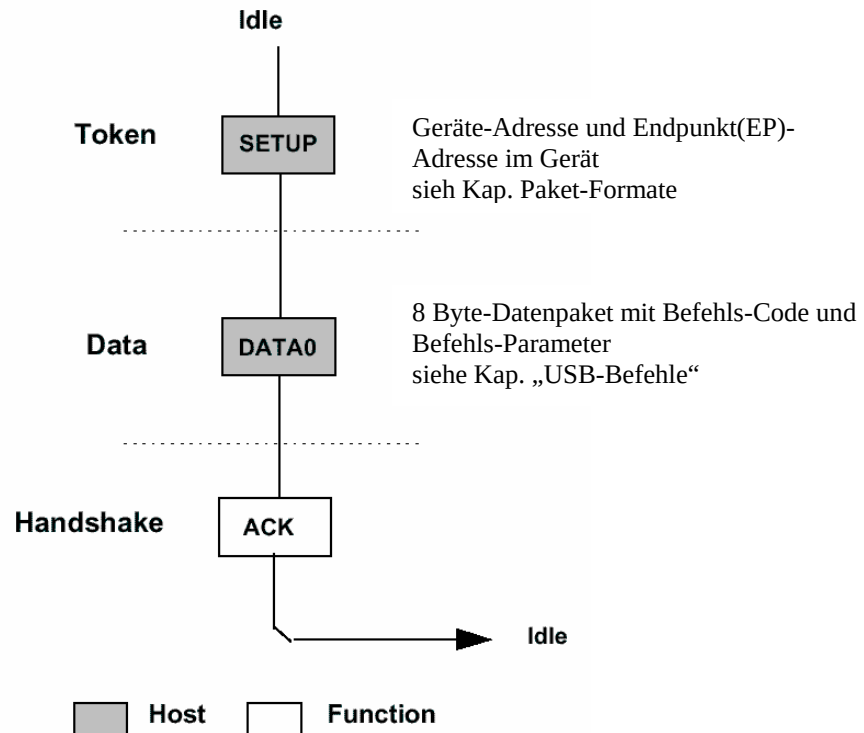
OUT-Token:

ACK: wie bei IN

NAK: Fehlerhafter Empfang

STALL: wie bei IN

Befehlstransfer: Control Transaction:



DEMO: USB-Transfer, aufgezeichnet mit einem USB-Analyzer „USB Agent“
Befehlstransfer

3) bulk-Transfer: bei IN und OUT: immer Toggeln, beginnend mit DATA1 , siehe Protokoll 1

5.8. Low- und Full-Speed-Geräte am High-Speed-Bus bzw. Hub

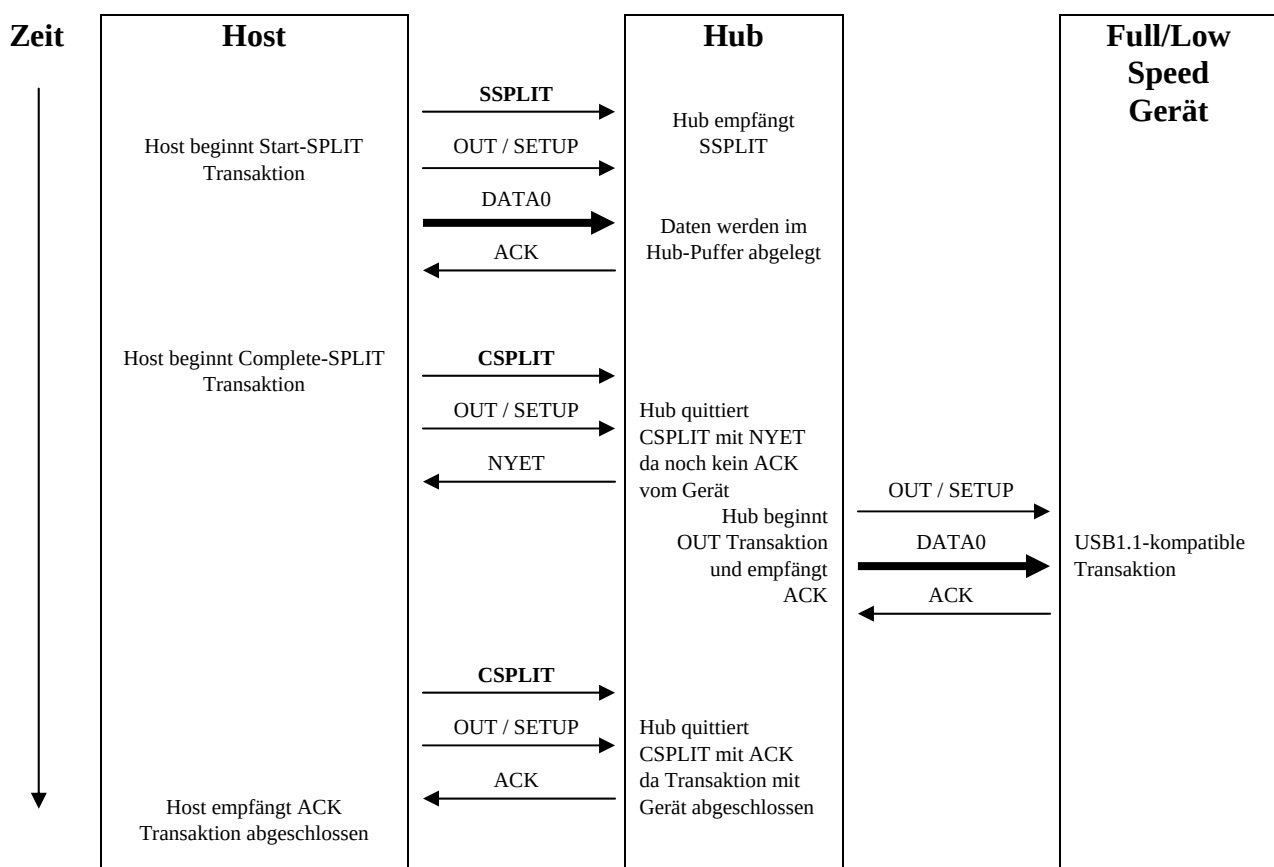
a) Low/Full Speed Geräte am High Speed Hub (=USB 2.0-Hub)

Wenn ein Low- oder Full-Speed-Gerät am High-Speed-Hub angeschlossen ist, dann wandelt der Hub die Geschwindigkeiten um. Die Datenrate bei High Speed ist 40 mal höher als bei Full Speed und 320 mal höher als bei Low Speed. Es wäre nicht sinnvoll den ganzen Bus warten zu lassen, weil ein Hub Daten mit einem langsamen Gerät austauscht.

Der Host kommuniziert im High-Speed-Modus mit dem USB2.0-Hub und der Hub im Full-Speed- oder Low-Speed-Modus mit dem Gerät.

Dies erfordert zusätzlich sogenannte Split-Transaktionen (SPLIT Packets), insgesamt wird aber wegen der wesentlich höheren Datenrate im High-Speed-Modus Zeit eingespart.

Beispiel: Ablauf bei Bulk OUT- und bei Control-Transfer



Format SPLIT Packet:

SYNC Bits	PID=78hex	Hub-Adr	SC=0/1	Port	S	E	ET	CRC5
-----------	-----------	---------	--------	------	---	---	----	------

Hub-Adr: Hub, an dem das Gerät hängt

SSPLIT: SC=0, **CSPLIT:** SC=1

Port: Port-Nummer des Downstream-Ports, an dem das Gerät hängt

S,E: gibt Lage des Datenpackets im Datentransfer an,

SE: 00 = Mitte, 01 = Ende, 10 = Anfang, 11 komplette FS-Daten.

ET = Endpunkt-Typ,

ET : 00 = Control, 01 = Isochron, 10 = Bulk, 11 = Interrupt

Anmerkung zu Bulk-IN-Transfer: Host gibt kein ACK bei Empfang DATA0/1 vom Hub (USB2.0 Norm)

b) Low-Speed-Geräte am Full-Speed-Hub

Dieser Fall ist im Praktikum wichtig, da bei den Versuchen V1 und V2 mit USB1.1-Analysatoren gearbeitet wird und u.a. Protokolle von Low-Speed-Geräten (z.B. Maus) am Full-Speed-Hub zu interpretieren sind. USB2.0 stellt hinsichtlich der Buskommunikation nur eine Ergänzung des USB1.1-Protokolls dar. Aktuell ausgelieferte Rechner (Desktops, Laptops usw.) enthalten USB1.1-Controller (UHCI oder OHCI), die mit einem USB2.0-Controller (EHCI) ergänzt sind, siehe Kapitel „Host-Controller ...“.

Low-Speed-Geräte werden durch den Hub vor Full-Speed-Datenverkehr geschützt. Will der Host ein Low-Speed-Gerät ansprechen wie z.B. die Maus, dann muss er dem Hub den Low-Speed-Verkehr ankündigen. Das erfolgt über das **PRE-Token**. Dieses wird im Full-Speed-Mode an den Hub gesendet. Erkennt ein Hub ein PRE-Token, dann aktiviert er die Verbindung von seinem Upstream-Port zu seinen Downstream-Ports, an denen Low-Speed-Geräte hängen. Das nachfolgende Paket (z.B. OUT-Token oder Datenpaket usw.), das für ein Low-Speed-Gerät bestimmt ist, wird vom Host im Low-Speed-Mode an den Hub gesendet und von diesem an die Low-Speed-Downstream-Ports durchgereicht. Mit dem Erkennen des Low-Speed-EOPs deaktiviert der Hub wieder seine Low-Speed-Ports.

PRE	1005	00000001	0x3C				1.33 us
SETUP	1006	00000001	0x2D	ADDR/EP	0x00/0x00	0x08	22.67 us
PRE	1007	00000001	0x3C				1.33 us
DATA0	1008	00000001	0xC3	DATA	(8) 80 06 00 01 00 00 40 00	0xBB29	65.33 us
ACK	1009	00000001	0xD2				12.00 us
SOF	1010	00000001	0xA5	FRAME#	0x3DB	0x0A	897.33 us
PRE	1011	00000001	0x3C				1.33 us
IN	1012	00000001	0x69	ADDR/EP	0x00/0x00	0x08	22.67 us
DATA1	1013	00000001	0x4B	DATA	(8) 12 01 00 01 00 00 00 08	0xC8E7	65.33 us
PRE	1014	00000001	0x3C				1.33 us
ACK	1015	00000001	0xD2				12.00 us
SOF	1016	00000001	0xA5	FRAME#	0x3DC	0x14	897.33 us
SOF	1017	00000001	0xA5	FRAME#	0x3DD	0x0B	1.000 ms
PRE	1018	00000001	0x3C				1.33 us
OUT	1019	00000001	0xE1	ADDR/EP	0x00/0x00	0x08	22.67 us
PRE	1020	00000001	0x3C				1.33 us
DATA1	1021	00000001	0x4B	DATA	(0)	0x00	22.67 us
ACK	1022	00000001	0xD2				12.00 us
SOF	1023	00000001	0xA5	FRAME#	0x3DE	0x09	940.00 us
SOF	1024	00000001	0xA5	FRAME#	0x3DF	0x16	1.000 ms
SETUP	1025	00000001	0x2D	ADDR/EP	0x02/0x00	0x15	2.83 us

Bild: Protokoll mit Low-Speed-Gerät am Full-Speed-Hub.

Das erste Setup-Token ist Low-Speed, das zweite Setup-Token ist Full-Speed; siehe unterschiedliche „Duration Times“ (Faktor 8!)

5.9. Übertragungsreihenfolge innerhalb eines Zeitrahmens

Der Host-Controller-Treiber legt die Reihenfolge der Datentransfers und die Aufteilung auf die verschiedenen Transferarten innerhalb eines 1ms-Zeitrahmens (bzw. 0,125ms) fest.

Derzeit gibt es drei Realisierungsvarianten des Host-Controllers:

USB 2.0: Enhanced Host Controller Interface **EHCI** (Fa. Intel, ..)

USB 1.1: Universal Host Controller Interface **UHCI** (Fa. Intel, ..)

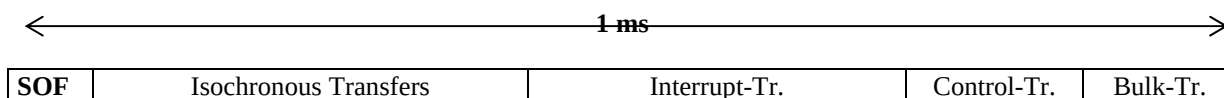
Open Host Controller Interface **OHCI** (Fa. Microsoft, Compac, ..)

a) Enhanced Host Controller Interface EHCI (Fa. Intel, ..)

EHCI-Controller sind nur für die High-Speed-Kommunikation zuständig. Um alle drei Geschwindigkeiten unterstützen zu können, muss ein Host über einen EHCI-Controller sowie über einen begleitenden UHCI- oder OHCI-Controller verfügen.

b) Universal Host Controller Interface UHCI (Fa. Intel, ..)

Für Isochronous- und Interrupt-Transfers (Transaktionen) sind maximal 90% der Zeit vorgesehen, für Control-Transfers sind 10% der Zeit garantiert. Verbleibenden Restzeiten werden für die zeitunkritischen Bulk-Transfers benutzt.



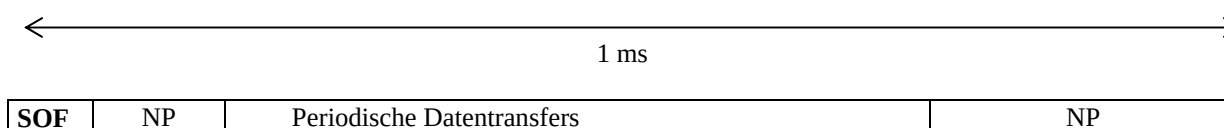
Die Kontrollregister für den UHCI-Host-Controller liegen im PCI-I/O-Adreßbereich. Die Basisadresse BAdr wird während des PCI-Konfigurationsprozesses festgelegt.

Kontrollregister:

USB-Command-Register	2 Byte	
USB-Status-Register	2 Byte	
USB-Interrupt-Enable	2 Byte	
Frame-Nummer	2 Byte	
Frame-List-Base-Address	4 Byte	
Start-of-Frame-Modify	1 Byte	Wert, der zum Startwert addiert wird. Damit ist die Länge des Frame-Intervalls variierbar.
Port1-Status/Control	2 Byte	
Port2-Status/Control	2 Byte	

c) Open Host Controller Interface OHCI (Fa. Microsoft, Compac, ..)

Nach dem SOF-Token beginnt der Host-Controller mit nicht-periodischen Transferarten. Bei der Initialisierung des Host-Controllers kann der Host-Controller-Treiber die maximale Zeit für dieses erste NP-Intervall festlegen (typischer Wert: 10% des Frame-Intervalls). Danach kommen die periodischen Datentransfers. Verbleibende Restzeiten sind wieder für nicht-periodische Datentransfers.



NP = nicht-periodische Datentransfers =

Periodische Datentransfers =

Frage:

was sind periodische und was sind nicht-periodische Datentransfers ? Bitte oben einfügen!

Die Kontrollregister des OHCI-Host-Controllers befinden sich im PCI-I/O-
Adreßbereich.

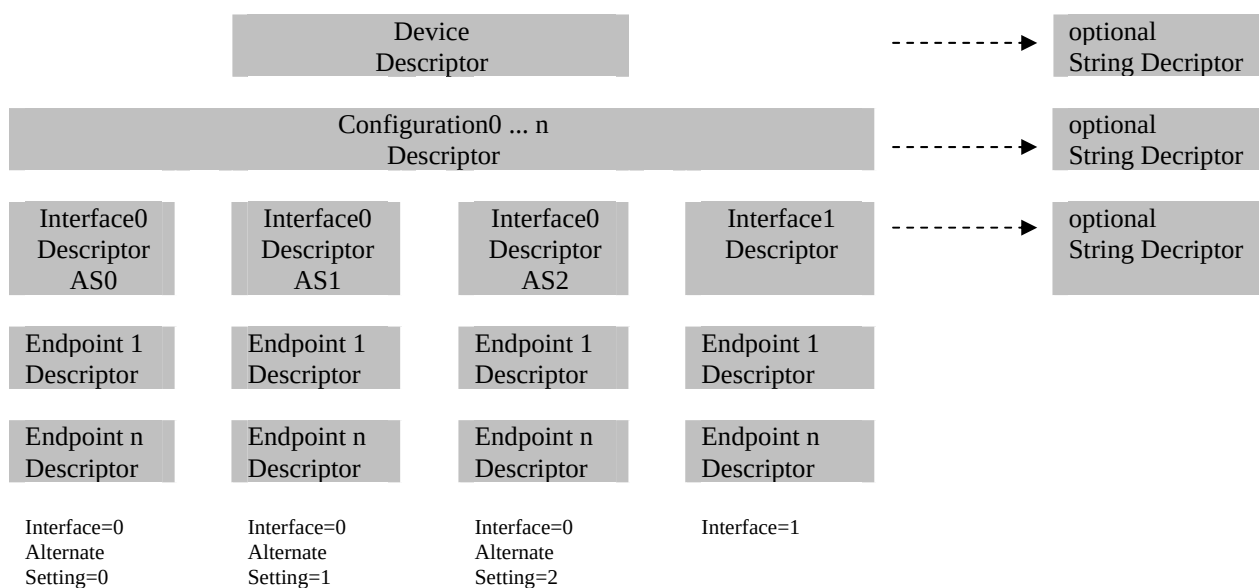
5.10. Deskriptoren

Um ein sogenanntes "Plug-and-Play" zu verwirklichen, muß der Host alle für die Inbetriebnahme eines Geräts am Bus notwendigen Informationen vom Gerät abrufen können. Die Eigenschaften eines Geräts sind in den Deskriptoren enthalten. Der Host fragt diese bei der Konfiguration (Enumeration) ab (Standard-Request "GetDescriptor").

Standard-Deskriptoren sind:

--Device Descriptor	Code =01
--Configuration Descriptor	=02
--String Descriptor (optional)	=03
--Interface Descriptor	=04
--Endpoint Descriptor	=05

Descriptor-Hierarchie: Device → Configuration → Interface → Endpoint.



Zu einem Device können mehrere Konfigurationen (Configuration Descriptors) gehören. Pro Konfiguration sind mehrere Interfaces (Interface Descriptors) und pro Interface mehrere Endpunkte (Endpoint Descriptors) möglich. Eine ausführliche Diskussion mit Anwendungen findet im Praktikumsversuch 2 statt.

Zu einem Interface können verschiedene Interface-Deskriptoren gehören, unterscheidbar durch den Parameter AS (Alternate Setting). Defaultmäßig ist Interface/AS = 0/0 aktiv.

Detailinformationen zu den Deskriptor-Inhalten siehe Unterlage „Basisinformationen“ zu den Praktikumsversuchen.

DEMO: USB-Transfer, aufgezeichnet mit einem USB-Analyzer „USB Agent“
Deskriptoren

beschreibt Grösse und Aufbau der vom Gerät gesendeten Reports

Device Descriptor (18 Byte): **Geräteadresse wo abgespeichert?**

Version USB-Spec./ Tiefe EP0-FIFO/ Vendor ID/ Product ID/ Zahl möglicher Konfigurationen usw. ..

Jedes Gerät hat einen einzigen Device Descriptor.

Er muß in jedem Gerät vorhanden sein. Jeder Hersteller, der Mitglied im USB-Implementers-Forum IF ist, erhält eine Vendor ID. Nichtmitglieder müssen sie beim IF beantragen (ca. 200US\$).

Anmerkung: Host liest Device Descriptor über EP0. Beim ersten Zugriff nur die ersten 8 Byte. Darin ist die FIFO-Tiefe (bMaxPacketSize0 in Byte 7) enthalten. Damit weiss der Host, wie viele IN-Token notwendig sind (pro IN-Token wird das FIFO komplett ausgelesen).

Configuration Descriptor (9 Byte):

Konfigurations-Nummer/ Zahl Interfaces/ Attribute wie "Bus-powered"/ Stromaufnahme in dieser Konfiguration/ usw. ..

Z.B. kann ein Gerät mit hohem Strombedarf (Configuration 1) einen Low-Power-Mode besitzen (Configuration 2), der ausgewählt wird, wenn das Gerät über den Bus mit Strom versorgt wird.

String Descriptor, optional:

Für textuelle Beschreibungen vom Hersteller, Gerät, Seriennummer, usw.

Interface Descriptor (9 Byte):

Interface-Nummer/ veränderbare Einstellungen (mehrere Interfaces einer Konfiguration sind unterscheidbar über den Parameter „AlternateSettings“)/ Zahl Endpoints ohne EP0/ Klassen-Code/ usw. ..

Zu einer Configuration können mehrere Interfaces (logische Schnittstellen) gehören. ISDN-Adapter können eine Telefonie- und eine Datenkommunikations-Schnittstelle anbieten. CD-ROMs können Massenspeicher und Audio/Video-Geräte (Klassen-Code) sein. Zu jedem Interface gehört ein entsprechender Geräte-Treiber im Host.

Endpoint Descriptor (7 Byte):

Endpoint-Adresse, z.B. IN, EP1/ Attribute wie Transferart, z.B. Bulk-Transfer-

Endpoint/ FIFO-Größe des Endpoints, z.B. 8 Byte/ Polling-Intervall, z.B. 16ms/ usw.

Der Endpoint (FIFO) ist Anfang bzw. Ende eines Datenkanals (pipe).

Die Endpoint-Adresse besteht aus Endpoint-Nummer und Richtung.

Der Endpoint Descriptor enthält die Eigenschaften des virtuellen Datenkanals (Pipe).

Durch Ändern von FIFO-Tiefe im Endpoint Descriptor und von AlternateSetting im Interface Descriptor kann die Bus-Belastung, die z.B. Geräte mit isochronem Transfer verursachen, verändert werden. Eine eingeschaltete Kamera muß z.B. periodisch abgefragt werden, was ca. 2 Mbit/s Busbelastung entspricht. Bei ausgeschalteter Kamera wird über FIFO-Tiefe =0 die periodische Geräte-Abfrage abgeschaltet (Busbelastung=0).

Neben den Standard-Deskriptoren gibt es noch **klassenspezifische Deskriptoren:**

z.B.:

HID-Descriptor	:	Code=21	(Human Interface Descriptor: für Maus, Tastatur, Joystick)
Hub-Descriptor		Code=29	

Spezifisch für die HID-Klasse:

Report-Descriptor	Code=22
Physical Descriptor	Code=23

5.11. USB-Befehle (USB Device Requests)

Die Steuerung/Abfrage von USB-Geräten erfolgt über "USB Device Requests", auf die jedes Gerät reagieren muß. Eine Untermenge davon sind die "Standard Device Requests".

Device Requests übergibt der Geräte-Treiber in Form von I/O-Request-Packets IRPs an den Bus-Treiber. Alle Requests werden mit SETUP-Token über den bidirektionalen Control-Endpunkt EP0 abgewickelt.

Aktivität am Bus :

SETUP-Token

DATA0 (Byte 0..7)

**Befehlsparameter, werden direkt nach dem SETUP-Token auf den Bus gelegt.
siehe Bild „Befehlstransfer“ Kap. 5.7**

Befehlsparameter in DATA0 (Byte 0..7):

Im ersten Byte (Feld "**bmRequestType**") eines jeden USB-Requests sind enthalten:

Byte 0 von DATA0:

Bit 7	nachfolgende Datentransferrichtung	0=OUT, 1=IN
Bit 6,5	Request Typ	00=Standard Request 01=Klassenspezif. Req. 10=Vendorspezif. Req. 11=Reserviert
Bit 4 .. 0	Empfänger des Request	00000=Device 00001=Interface 00010=Endpoint 00011=Andere

Bspl..

OUT/ Standard Request/ für Device: DATA0, BYTE0 = 00hex

Standard Device Requests:

Byte 1 von DATA0:

	Byte1 in DATA0	
GetStatus		Lesen Device Status, Interface Status, Endpoint Status
SetFeature/ ClearFeature		Ein/Ausschalten von Geräteeigenschaften, z.B. wake-up
SetAddress	05	Nach dem Einschalten hat ein Gerät die Adresse 0. Bei der Enumeration wird mit SetAddress eine neue Adresse zugewiesen (Position 2)
GetDescriptor	06	Lesen der Deskriptoren (Descriptor length = Position 6)
SetDescriptor (optional)		Änderung von Deskriptoren.
GetConfiguration		Abfrage der Konfiguration
SetConfiguration	09	Zuweisen (auswählen) einer Konfiguration
GetInterface		Abfrage des Interface (falls mehreren Interfaces definiert)
SetInterface		Zuweisen (auswählen) eines Interfaces
SynchFrame		Einstellen einer spez. Synchronisation

Beim "GetStatus"-Request wird z.B. der Device-Status vom Host abgeholt (Datentransferrichtung=IN).

Frage:

Der Host will mit einem Request "GetStatus" den Geräte-Status ermitteln. Wie lautet das erste Byte des Request-Feldes

Zusätzlich zu den Standard Device Requests unterstützt jedes Gerät noch **gerätespezifische Requests**. In vielen Fällen sind dies Befehle aus den **SCSI-Befehlssätzen der SCSI-3 Norm**.

Beispiel: Requests der Geräteklasse "Printer":

GetDeviceID	Drucker sendet Geräte-ID in Form einer Zeichenkette
GetPortStatus	Drucker sendet Status-Byte. Aufbau entspricht dem Statusregister bei der Parallelschnittstelle (Paper Empty, Select, ..)
SoftReset	Löschen aller Pufferspeicher im Drucker

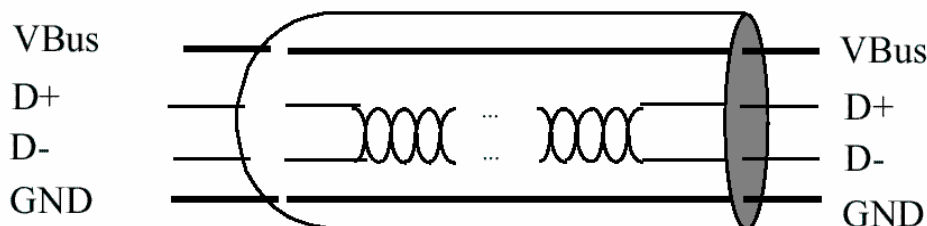
5.12. Elektrische Eigenschaften und Datencodierung

Der USB ist ein serieller Bus mit Punkt-zu-Punkt Verkabelung, der die Daten über ein bidirektionales Leitungspaar überträgt (Halbduplex-Übertragung, Daten jeweils nur in Hin- oder in Rückrichtung.). Die Datenübertragung ist differentiell (D+, D-).

J-Pegel: Differenz zwischen D+ und D- ist positiv (differentielle „1“)

K-Pegel: Differenz zwischen D+ und D- ist negativ (differentielle „0“)

Zusätzliche Adern: Masse GND und Versorgungsspannung VBus(5Volt).



Leitung	Pin am Stecker	Adern-Farbe
Vcc	1	Rot
D-	2	Weiss
D+	3	Grün
GND	4	Schwarz

Es gibt weder Handshake-Signale (wie beim SCSI-Bus oder bei der ECP/EPP-Schnittstelle) noch eine eingestellte Baudrate (wie bei RS232) noch eine separate Taktleitung. Der Takt für die Abtastung des übertragenen Datenstroms und das Synchronisierungssignal für den Takt werden aus einem Synchronisier-Bitmuster und aus dem Datenstrom selbst generiert. Jede "0" im Bitstrom liefert einen Pegelwechsel und damit ein Signal für die Taktgewinnung. Nach einer Folge von sechs Einsen ("1") wird automatisch in den Datenstrom eine "0" eingefügt (Bit Stuffing). Damit kommt spätestens nach der sechsten Taktperiode ein Signal für die Takt-Synchronisierung. Jedes Gerät hat spätestens nach sieben Taktperioden Gelegenheit, seinen internen Takt mit dem Bustakt zu synchronisieren.

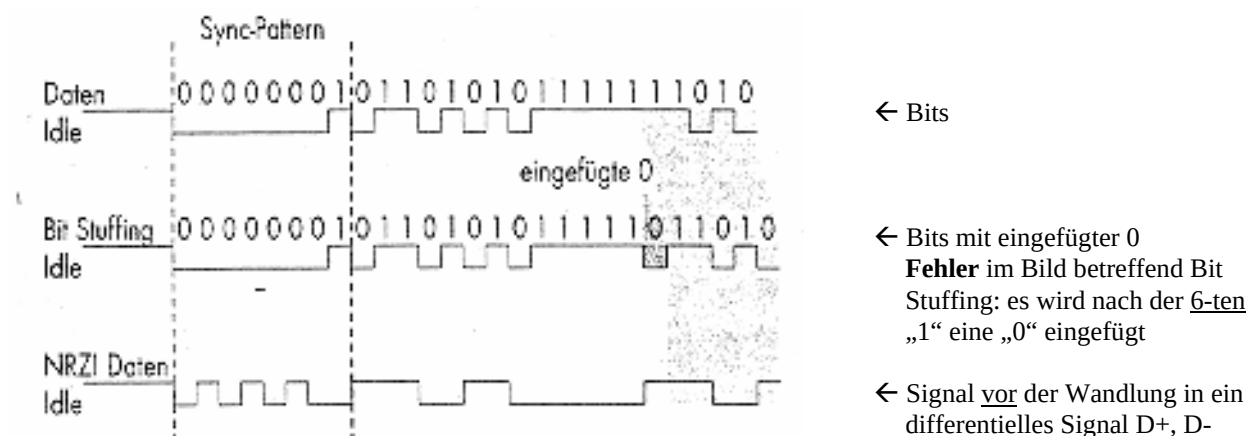


Bild: Daten-Codierung und resultierendes NRZI-Signal

a) Kabel Low Speed und Full Speed

Full-Speed-Kabel: max. 5m (max. 30ns Verzögerung für ein Kabelsegment)

Low-Speed-Kabel: max. 3m (Datenleitungen nicht verdreht, keine Abschirmung)

Kabel-Abschluß bei Full-Speed-Gerät: D+ über $R_{pu}=1,5k$ gegen 3.3Volt

Kabel-Abschluß bei Low-Speed-Gerät: D- über $R_{pu}=1,5k$ gegen gegen 3.3Volt

$R_{pu} = 1,5k$ Pull up Widerstand,

$R_{pd} = 15k$ Pull down Widerstand an D+ und D- (gegen Masse)

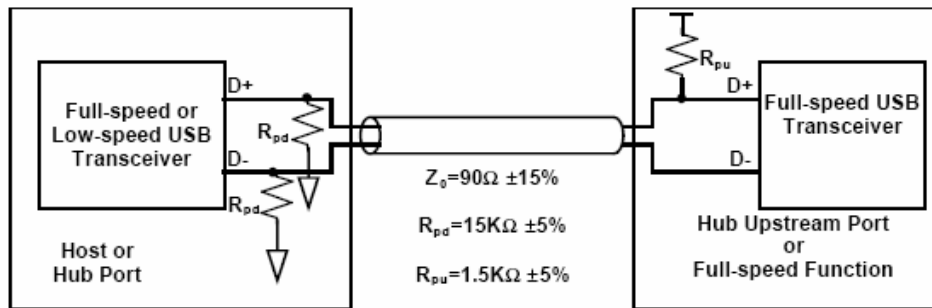


Figure 7-20. Full-speed Device Cable and Resistor Connections

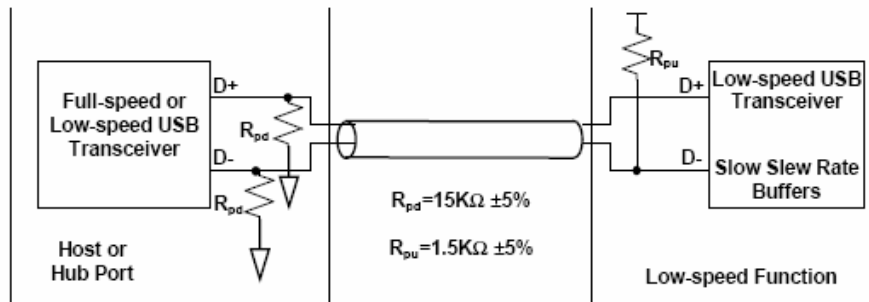


Figure 7-21. Low-speed Device Cable and Resistor Connections

b) Identifizierung der Geschwindigkeitsklasse (Connect, Disconnect)

Der Host muss das Anstecken eines Geräts bemerken und er muss zusätzlich die Geschwindigkeitsklasse identifizieren können (Hot Plug-and-Play). Bei Low- und Full-Speed-Geräten ist dies einfach; das Gerät zieht den D- -Pegel bzw. den D+ -Pegel mit einem Widerstand Rpu auf high, siehe Bild. Für High-Speed-Geräte steht keine weitere Datenleitung zur Verfügung, ausserdem muss sich ein High-Speed-Gerät an einem USB1.1-Controller wie ein Full-Speed-Gerät verhalten (High-Speed-Geräte sollen auch an einen Full-Speed-Controller anschließbar sein). Deswegen wurde für High-Speed-Geräte die weiter unten beschriebene Anmeldeprozedur spezifiziert.

kein Gerät: D+, D- = low ($>2,5\mu s$) = **SE0-Zustand (Single Ended Zero)**

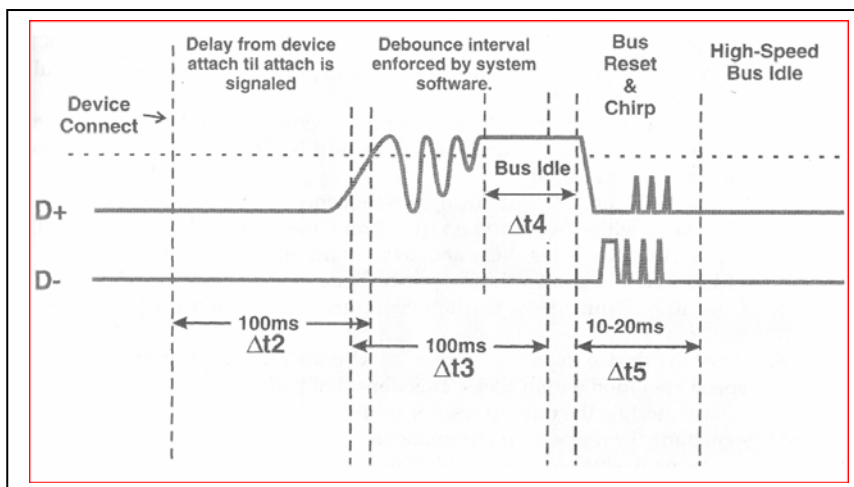
Low Speed: D- = high ($>2,5\mu s$)

Full Speed: D+ = high ($>2,5\mu s$)

High Speed: verhält sich zunächst wie Full Speed Gerät (D+ = high), damit von USB1.1-Controllern/Hubs als Full Speed Gerät identifizierbar (High Speed Geräte sind auch im Full Speed Modus voll funktionsfähig).

Danach identifiziert sich das High Speed Gerät durch Abschalten des Pull-Up-Widerstands und aktiviert die 45 Ohm Abschlusswiderstände an D+ und D-.

(Erkennung der aktuellen Geschwindigkeit eines Full/High Speed Geräts z.B. über Bulk-Endpunkt: Feld MaxPacketSize=512 im Endpunkt-Deskriptor bei High Speed)



Chirp (engl.) = zwitschern

Bild: Identifizierung eines High-Speed-Geräts beim An-stecken an einen USB2.0-Hub

„Bus Idle“ in Bildmitte:
D+=VBus, D-=GND,
entspricht Full Speed-Idle
(Gerät).

Hub legt Reset an D+, D-.
HS-Gerät erkennt Reset und
sendet D-=Chirp K (=1).

Hub antwortet abwechselnd mit
Chirp Ks (D-=1) und Chirp Js
(D+=1).
Gerät schaltet Pull-Up

Tabelle „Ihr Wissen über den USB-Bus“ am Kapitel-Anfang ergänzen/korrigieren

c) Kabel High-Speed

- Länge = 5m
- kein Pull-Up-Widerstand
- beidseitiger Abschluss von D+, D- mit Z=45 Ohm gegen Masse

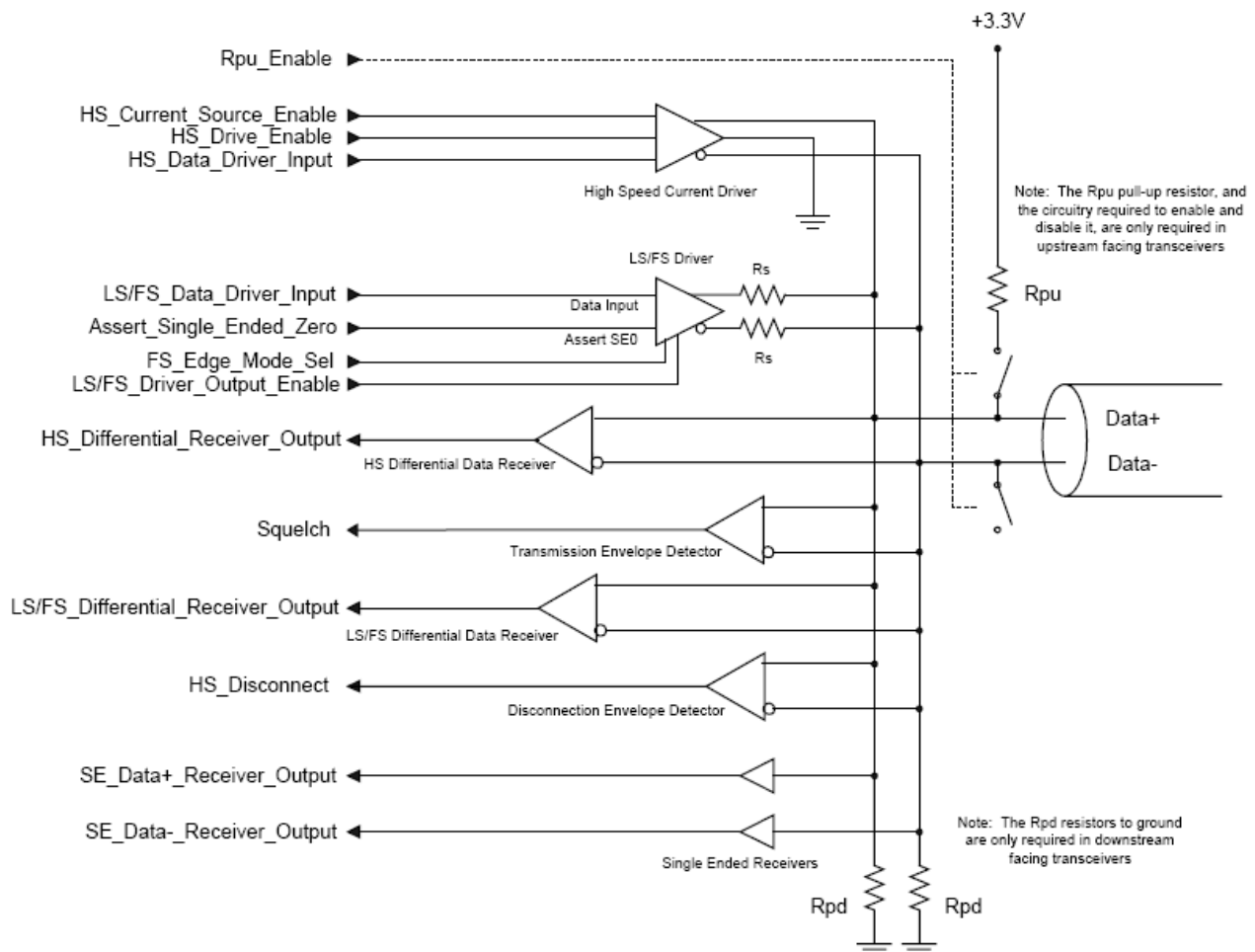


Bild: High Speed Kabel Transceiver (USB20-Spec, Bild 7-1)

High Speed Betrieb:

- LS/FS-Driver (Treiber) liefert Massepunkte (Assert_Singel_Ended_Zero) für die beiden Rs=45 Ω - Widerstände an den Datenleitungen D+, D-. Das ergibt einen differentiellen Widerstand von 90 Ω zwischen D+ und D- .
- der High Speed Current Driver liefert die Datensignale D+, D- (HS_Data_Driver_Input)
- Rpu=1,5 k Ω ist nach der Identifizierung der Geschwindigkeitsklasse (low/full/high Speed) abgeschaltet
- Die beiden Rpd-Widerstände (15 k Ω) sind vernachlässigbar gegen die 45 Ω - Widerstände.
- Die Signalpegel werden jeweils von einer 17,8 mA Stromquelle (High Speed Current Driver) erzeugt:
 - kein Gerät angesteckt: $\pm 17,8 \text{ mA} \cdot 45 \Omega = \pm 800 \text{ mV}$
 - Gerät angesteckt: $\pm 17,8 \text{ mA} \cdot 22,5 \Omega = \pm 400 \text{ mV}$
(45 Ω Widerstände von Sender und Empfänger parallel)

Abziehen eines High-Speed-Geräts (Disconnect):

Der Hub erkennt am Spannungssprung 400mV → 800 mV das Abziehen des Geräts.

5.13. Initialisierung (Bus Enumeration)

Enumeration eines Geräts:

Das Initialisieren oder Konfigurieren eines Geräts am USB wird mit Enumeration bezeichnet. Ein Gerät durchläuft dabei vier der sechs Gerätezustände:

Powered→Default→Addressed→Configured.

Die beiden anderen Zustände sind: **Attached** (Hub versorgt Gerät nicht mit Strom) und **Suspended** (Gerät erkennt mindestens 3ms auf dem Bus keine Aktivität und reduziert die Stromaufnahme).

Das Anstecken oder Abziehen eines Geräts am Bus wird durch den Hub erkannt (siehe Kap.) und dem Host gemeldet. Der Host fragt den Hub ab (Full-Speed- oder Low-Speed-Gerät, Port-Adresse) und führt dann die Enumeration in folgenden Schritten durch:

- Anstecken des Geräts am Hub bzw. Einschalten des Systems
- Port Enable durch den Hub, Hub liefert max. 100mA an Gerät („Powered“)
- Hub erkennt Geschwindigkeitsklasse des Geräts über die Signalleitungen D+, D-
- Hub informiert Host über Interrupt-Kanal
- Hub setzt Gerät zurück: D+ und D- auf logisch Null, danach ist Gerät im Zustand „Default“. Gerät ist jetzt für Control-Transfers zum EP0 bereit.
- Host ermittelt über GetDescriptor die maximale Paketgröße des Kanals (Adr=0, EP0)
- Host weist Geräteadresse zu über SetAddress („Addressed“)
- Host liest Deskriptoren über GetDescriptor
- Host ermittelt aus den Deskriptoren die Ressourcen-Auslastung durch das Gerät (FIFO-Größe, Polling-Intervalle, Stromaufnahme)
- Host lädt einen Gerätetreiber
- Host weist über den Gerätetreiber eine Konfiguration zu mit SetConfiguration („Configured“)

Nach dem Ende der Enumeration werden Interrupt-Endpunkte zyklisch abgefragt und Isochrone Endpunkte jede Millisekunde angesprochen.

Enumeration des gesamten Busses nach dem Einschalten:

Nach einem "Power-on-Reset" arbeiten alle USB-Geräte auf Adresse 0 (Default). Es sind alle Downstream-Ports deaktiviert, die folgenden Standard-Device-Requests sieht nur der Hub1. Zuerst wird der Hub1 enumeriert, dann fragt der Host den Interrupt-Endpunkt des Hub1 ab und erkennt, daß weitere Geräte an den Downstream-Ports von Hub1 angeschlossen sind. Sukzessive schaltet der Host die Ports des Hub1 frei und enumeriert die Geräte.

Praktikum, Versuch1: Interpretation verschiedener Busprotokolle (Paket-Ebene)

Praktikum, Versuch2: Deskriptoren-Layout, Development Board EZ-USB (Controller Chip AN 2131), Composite Devices und Human Interface Devices HID

5.14. On-The-Go USB (OTG-USB)

a) Einführung

In zunehmendem Maße wird eine direkte Kommunikation zwischen zwei Geräten (ohne den Umweg über einen Computer) verlangt. Dies ist mit der „**On-The-Go**“-Erweiterung des USB (USB-OTG) möglich.

Welche USB-Geräte sind OTG-„verdächtig“?



Drucker, PDA (Personal Digital Assistant), Lautsprecher, Hdy, Foto, Tastatur, Hub, Controller Spielkonsole, Festplatte, Kamera, MP3-Player usw.

Anwendungen:

a) PDA + Tastatur

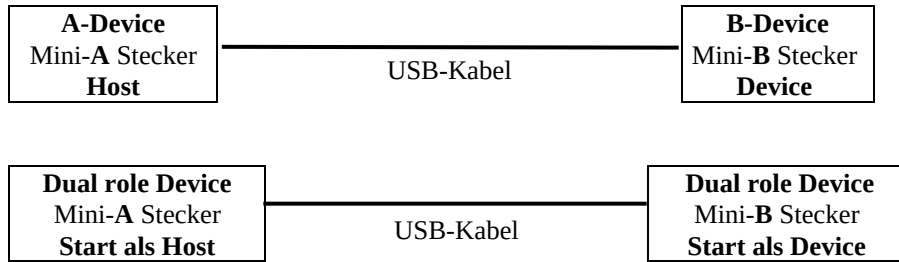


b) Foto + Handy (Transfer Bild → Handy)



DEMO:
Diskussion

b) Merkmale von USB OTG



- Punkt-zu-Punkt (USB)-Verbindung ohne Computer
- 12 MBit/s, 1,5 MBit/s, optional: 480 MBit/s
- verkleinerte Buchsen und Stecker (Mini-A und Mini-B Stecker)
- 4-adriges Kabel, aber Stecker mit 5 Pins (zusätzlich: ID-Pin)
- Geräte sind begrenzt Master-fähig
- OTG-Spezifikation: Ergänzung zur USB2.0-Spezifikation

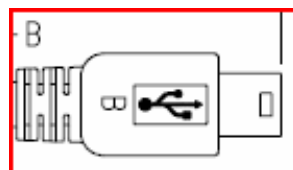
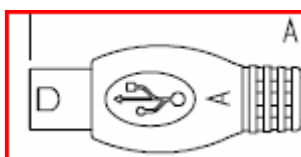
USB OTG-Geräte sind einteilbar in A-Devices, B-Devices oder Dual-Role-Devices:

A-Device = Host (Master)	B-Device = Device (Slave)	Dual-Role-Device
Default Host	Default Device	Start als A-Device oder B-Device, abh. von Stecker (Mini-A oder Mini-B) Host $\leftrightarrow$ Device Rollentausch möglich über HNP (Host Negotiation Protokoll)
Mini-A Stecker ID-Pin des Steckers mit Masse-Pin verbunden (bedeutet im Zustandsdiagramm: ID=False)	Mini-B Stecker ID-Pin „floating“	Mini-AB Buchse für Mini-A oder Mini-B Stecker
liefert Spg. Vbus auf Bus, $8\text{mA} < I_{cc} < 500\text{mA}$	über Bus versorgt (Vbus) oder Batterie-versorgt	liefert Spg Vbus auf Bus, $8\text{mA} < I_{cc} < 500\text{mA}$
		Peripheral List = Verzeichnis der als B-Devices anschließbaren Geräte

Stecker-Belegung für Mini-A Stecker ($\equiv \text{ID=False}$):

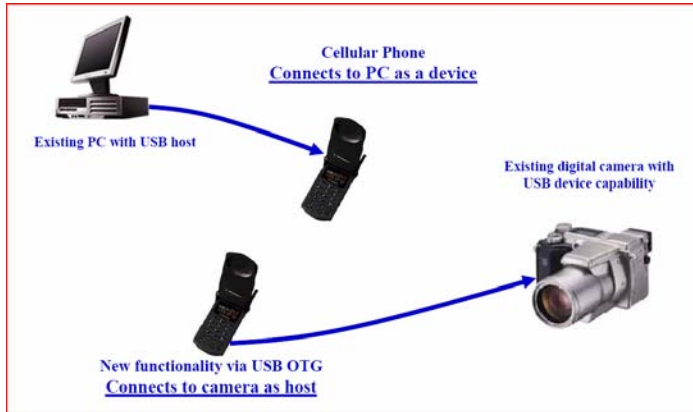
Contact Number	Signal Name	Typical Wiring Assignment
1	VBUS	Red
2	D-	White
3	D+	Green
4	ID	$< 10 \Omega$ to GND
5	GND	Black
Shell	Shield	Drain Wire

Mini-A und Mini-B-Stecker:



Kabel-Kombinationen:

Stecker	Stecker
Mini-A Stecker	Mini-B Stecker
Mini-A	Standard-B
Standard-A	Mini-B



Application Examples

Host	Peripheral	Application
Mobile Phone	Mobile Phone	Exchange contact information
	Still Image Camera	Email pictures, upload pictures to web
	MP3 Player	Upload/download/broadcast music
	Mass Storage	Upload/download files
Still Image Camera	Scanner	Scan business cards
	Still Image Camera	Exchange pictures
	Mobile Phone	Email pictures, upload pictures to web
	Printer	Print pictures
Printer	Mass Storage	Store pictures
	Still Image Camera	Print pictures
	Scanner	Print scanned image
MP3 Player	Mass Storage	Print files stored on device
	MP3 Player	Exchange songs
	Mass Storage	Upload/download songs

Application Examples - more

Host	Peripheral	Application
AutoPC	MP3 player	Play audio in car
	Mobile Phone	Hands free over stereo
PDA	PDA	Exchange files
	Printer	Print files
	Mobile Phone	Upload/download files
	MP3 Player	Upload/download songs
	Scanner	Scan pictures
	Mass Storage	Upload/download files
	GPS	Obtain directions, mapping information
	Still Image Camera	Upload pictures

c) OTG-Protokolle

Enumeration:

Während der Enumeration liest ein A-Device den OTG-Descriptor des B-Device (GetDescriptor Command).

OTG-Descriptor:

Offset	Field	Size	Value	Description
0	<i>bLength</i>	1	Number (3)	Size of Descriptor
1	<i>bDescriptorType</i>	1	Constant	OTG type = 9
2	<i>bmAttributes</i>	1	Bitmap	Attribute Fields D7...2: Reserved (reset to zero) D1: HNP support D0: SRP support

SetFeature Command:

Ein A-Device kann über das SetFeature Command das B-Device steuern und/oder dem B-Device die eigenen „Merkmale“ melden.

Befehlsparameter:

- Freigabe des HNP Protokolls im B-Device (Command *b_hnp_enable*)
- Meldung: ich(A-Device) bin HNP-fähig (Meldung *a_hnp_support*)
- Meldung: mein anderer Port (A-Device) ist HNP-fähig (Meldung *a_alt_hnp_support*)

Session Request Protocol SRP:

Die OTG-Spezifikation erlaubt einem (batterieversorgten) A-Device, die Busspannung Vbus auszuschalten wenn keine Busaktivität (Session) stattfindet. Wenn ein B-Device den Bus benötigt obwohl Vbus ausgeschaltet ist, muss es das A-Device veranlassen können, Vbus einzuschalten und eine Session zu starten (Vbus eingeschaltet → Session läuft). Dies erfolgt über das SRP (Session-Anforderungsprotokoll).

Es gibt 2 Methoden, mit denen ein B-Device ein A-Device veranlassen kann, eine Session zu beginnen:

- „data-line pulsing“: B-Device gibt Puls-Folge auf D+-Leitung über Pull-up-Widerstand (ein/aus)
- „Vbus-pulsing“: B-Device gibt Puls-Folge auf Vbus-Leitung

Ein B-Device muss beide Methoden beherrschen, ein A-Device nur eine von beiden.

Host Negotiation Protocol HNP:

Das HNP erlaubt es, ohne Umstecken des Kabels die Host-Funktion zwischen zwei Dual-Role-Devices zu übergeben (dynamische Übergabe der Host-Funktion). HNP wird typischerweise ausgeführt als Reaktion auf eine Anforderung durch das Dual-Role-B-Device.

Das A-Device schickt das Command SetFeature (*b_hnp_enable*), schaltet Vbus ab und stoppt alle Busaktivitäten. Das B-Device schaltet Vbus ein und übernimmt den Bus.

5.15. Wireless USB

Momentan besetzen zwei Initiativen den Begriff "WirelessUSB".

Die ältere der beiden wurde von der Firma Cypress initiiert, mittlerweile ist die Firma Atmel als zweiter Chiphersteller auf den Zug aufgesprungen. Das Cypress WirelessUSB ist eigentlich kein drahtloses USB, sondern eine Technik um drahtlose Endgeräte zu bauen, die dann über einen am USB angeschlossenen Empfänger/Sender (Transceiver) mit dem Computer verbunden sind. Dazu wird eine Übertragungstechnik im lizenzfreien 2,4-GHz-Band benutzt, die Datenrate beträgt bis zu 62,5 kbit/s und ist damit für Eingabegeräte völlig ausreichend, für andere Anwendungen aber oft zu knapp bemessen.

Das zweite (aktuelle) WirelessUSB Projekt ist wesentlich anspruchsvoller, das dazugehörige Konsortium wird von Intel angeführt. Ziel ist es, eine Technik zu schaffen, mit der die vollen 480 MBit/s des High-Speed-Übertragungsmodus drahtlos übertragen werden kann. Dabei ist eine kurze Reichweite unter 10 m vorgesehen; die Übertragung soll auf einer Ultra-Wideband Technik basieren. Erste Demonstrationen dieser Technik sind jedoch nicht vor der zweiten Hälfte 2005 zu erwarten, wann dann Produkte kommen werden, ist noch unklar.

Eine erste Spezifikation Rev. 1.0 vom 12.5.2005 liegt vor, unterstützt von Firmen wie Intel, Microsoft, HP, Philips, NEC usw.

Als Datenraten sind vorgeschlagen:

obligatorisch :	53,3, 106,7 und 200 MBit/s
optional :	80, 160, 320, 400 und 480 MBit/s

„Bus powered“ ist dann natürlich nicht mehr möglich und muss durch “Battery powered” ersetzt werden.

5.16. Aufgaben

Aufgabe USB-1: Bus-Zuteilung bei PCI Conventional und USB

1. Welche Funktionen (Master, Slave) können die Busteilnehmer bei beiden Bussen haben?
2. Wie erfolgt die Bus-Zuteilung für einen Teilnehmer?
3. Wie würden Sie die Bus-Zuteilung beim PCI-Bus bezeichnen ? (zentral, dezentral). Begründung:
4. Kennen Sie einen englischen Begriff für Bus-Zuteilung?

Aufgabe USB-2: USB, Begriffe

Der Endpunkt (Endpoint) ist ein zentraler Begriff beim USB. Erläutern Sie ausführlich!
Welche wichtigen Eigenschaften hat er und wo sind diese hinterlegt?

Aufgabe USB-3: USB mit Speicher-Stick und Maus

Am USB sind eine Maus und ein Speicher-Stick angeschlossen. Für beide liegt eine Anforderung für einen Transfer vor. Wer hat „Vorfahrt“? Begründung.

Aufgabe USB-4: USB-Protokoll

PID type	Idle	packet#	SYNCH	PID value	function	function data	crc5/16	time
SETUP		115	00000001	0x2D	ADDR/EP	0x01/0x00	0x17	124.020 ms
DATA0		116	00000001	0xC3	DATA	(8) 80 06 00 01 00 00 12 00	0x72F	124.028 ms
ACK		117	00000001	0xD2				124.029 ms
IN		119	00000001	0x69	ADDR/EP	0x01/0x00	0x17	125.020 ms
DATA1		120	00000001	0x4B	DATA	(8) 12 01 10 01 09 00 00 08	0x48D7	125.028 ms
ACK		121	00000001	0xD2				125.029 ms
IN		123	00000001	0x69	ADDR/EP	0x01/0x00	0x17	126.020 ms
DATA0		124	00000001	0xC3	DATA	(8) 8F 05 54 92 12 03 01 02	0xC62F	126.028 ms
ACK		125	00000001	0xD2				126.029 ms
IN		127	00000001	0x69	ADDR/EP	0x01/0x00	0x17	127.020 ms
DATA1		128	00000001	0x4B	DATA	(2) 00 01	0xFCF1	127.024 ms
ACK		129	00000001	0xD2				127.025 ms
OUT		131	00000001	0xE1	ADDR/EP	0x01/0x00	0x17	128.020 ms
DATA1		132	00000001	0x4B	DATA	(0)	0x00	128.023 ms
ACK		133	00000001	0xD2				128.024 ms
SETUP		136	00000001	0x2D	ADDR/EP	0x01/0x00	0x17	130.020 ms
DATA0		137	00000001	0xC3	DATA	(8) 80 06 00 02 00 00 09 00	0x7520	130.028 ms
ACK		138	00000001	0xD2				130.029 ms
IN		140	00000001	0x69	ADDR/EP	0x01/0x00	0x17	131.020 ms
DATA1		141	00000001	0x4B	DATA	(8) 09 02 19 00 01 01 00 E0	0xF0C7	131.028 ms
ACK		142	00000001	0xD2				131.029 ms
IN		144	00000001	0x69	ADDR/EP	0x01/0x00	0x17	132.020 ms
DATA0		145	00000001	0xC3	DATA	(1) 32	0x8356	132.023 ms
ACK		146	00000001	0xD2				132.025 ms
OUT		148	00000001	0xE1	ADDR/EP	0x01/0x00	0x17	133.020 ms
DATA1		149	00000001	0x4B	DATA	(0)	0x00	133.023 ms
ACK		150	00000001	0xD2				133.024 ms
SETUP		153	00000001	0x2D	ADDR/EP	0x01/0x00	0x17	135.020 ms

beachten Sie die Detail-Infos auf der nächsten Seite!

- Betrachten Sie die Pakete von #115 bis #133:
 - welche Adresse hat der adressierte Busteilnehmer?
 - welcher Deskriptor wird gelesen?
 - wie viele Bytes umfasst dieser Deskriptor?
 - was ist die FIFO-Größe (Byte) im EP0?
 - um welchen Typ des Busteilnehmers handelt es sich?
 - erklären Sie stichpunktartig die folgenden Pakete:
 - data0 #116:
 - data1 #120:
 - data0 #124:
 - data1 #128:
 - data1 #132:
- Annahme: Obiges Gerät ist eine Maus, die an einem Full-Speed-Hub angeschlossen ist. Das Protokoll zeigt die Pakete am Full-Speed-Bus.

Welche wichtige Änderung (Pakete) im Protokoll würden Sie erwarten?

Betrachten Sie die 3 Pakete von #115 bis #117 und geben Sie an, wo sich im Ablauf der Pakete was ändern würde.
- Annahme: Obiges Gerät ist eine Maus, die an einem High-Speed-Hub angeschlossen ist. Das Protokoll zeigt die Pakete am High-Speed-Bus.

Welche wichtige Änderung (Pakete) im Protokoll würden Sie erwarten?

Betrachten Sie die Pakete von #115 bis #117 und geben Sie an, wo sich im Ablauf der Pakete was ändern würde.

Detailinformationen zu Aufgabe 11c:

SETUP → DATA0 →

Byte 0 von DATA0:

Bit 7	Nachfolgender Informationsfluss	0=OUT, 1=IN
Bit 6,5	Request Typ	00=Standard Request 01=Klassenspezif. Req. 10=Vendorspezif. Req. 11=Reserviert
Bit 4 .. 0	Empfänger des Request	00000=Device 00001=Interface 00010=Endpoint 00011=Andere

Byte 1..7 von DATA0:

Byte1		Erklärung bzw. Byte2..7
00	GetStatus	Lesen Device Status/ Interface Status/ Endpoint Status, spezifiziert in Byte0/ Bit4..0 (=Empfänger des Requests, siehe oben)
01/03	ClearFeature/ SetFeature	Aus/Einschalten von Geräteeigenschaften, z.B. wake-up
05	SetAddress	Nach dem Einschalten hat ein Gerät die Adresse 0. Bei der Enumeration wird mit SetAddress eine neue Adresse zugewiesen Belegung der Bytes 2 ..7: Byte 2,3: Device Address Byte 4,5: Index, Byte 6,7: Länge der folgenden Datenübertragung
06	GetDescriptor	Lesen der Deskriptoren Belegung der Bytes 2 ..7: Byte 2: Descriptor Index Byte 3: Descriptor Type 01 = device descriptor 02 = configuration descr. 03 = string descr. 04 = interface descr. (enthält u.U. endpoint descr.) 05 = endpoint descr. Byte 4,5: Index Byte 6,7: Descriptor Length

Device-Deskriptor:

Offset	Feldbezeichnung	Länge(Byte)	Beschreibung	Werte
0	bLength	1	Größe dieses Deskriptors in Byte	0x01 Device Descriptor 0x02 Configuration Descriptor 0x03 String Descriptor (optional) 0x04 Interface Descriptor 0x05 Endpoint Descriptor, z.B. 0x0110 bedeutet Version 1.1 0x00 HI (Human Interfaces): Maus, .. 0x08 Mass Storage 0x09 Hub
1	bDescriptorType	1	Deskriptortyp = Device-Descriptor	
2	bcdUSB	2	Version der USB Spec. (z.B. 1.1)	
4	bDeviceClass	1	Klassen-Code	
5	bDeviceSubClass	1	Subklassen-Code	
6	bDeviceProtocoll	1	Protokoll-Code	
7	bMaxPacketSize0	1	Tiefe EP0-FIFO in Byte	
17	bNumConfigurations	1	Zahl möglicher Konfigurationen	

Configuration-Deskriptor:

Offset	Feldbezeichnung	Länge(Byte)	Beschreibung											
0	bLength	1	Größe dieses Deskriptors in Byte											
1	bDescriptorType	1	Deskriptortyp = Configuration-Descriptor											
2	wTotalLength	2	Länge aller zu dieser Konfiguration gehörenden Deskriptoren											
4	bNumInterfaces	1	Anzahl zu dieser Konfiguration gehörenden Interfaces											
5	bConfigurationValue	1	Nummer dieser Konfiguration											
6	iConfiguration	1	String-Index für „Konfigurationen“											
7	bmAttributes	1	Attribute für diese Konfiguration											
			<table><tr><th>Bitposition</th><th>Bedeutung</th></tr><tr><td>7</td><td>Bus-powered Device</td></tr><tr><td>6</td><td>Self-powered Device</td></tr><tr><td>5</td><td>Remote-Wakeup</td></tr><tr><td>4..0</td><td>Reserviert</td></tr></table>	Bitposition	Bedeutung	7	Bus-powered Device	6	Self-powered Device	5	Remote-Wakeup	4..0	Reserviert	
Bitposition	Bedeutung													
7	Bus-powered Device													
6	Self-powered Device													
5	Remote-Wakeup													
4..0	Reserviert													
8	maxPower	1	Stromaufnahme für diese Konfiguration in 2 mA-Einheiten											

6. Serielle Schnittstelle RS-232

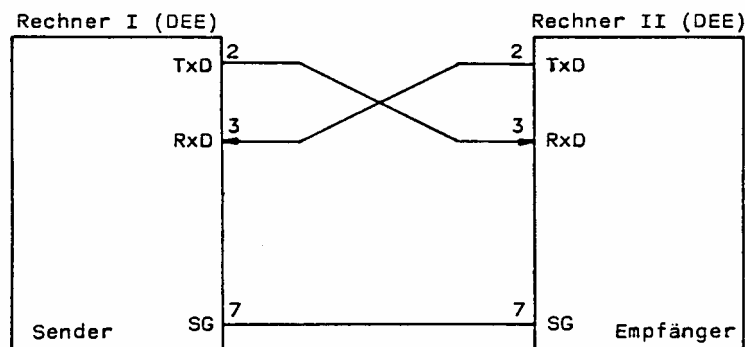
Stark gekürzt! Komplettes Kapitel siehe Anhang

Software-Handshaking

Die Kommunikation erfolgt über spezielle Steuerzeichen die in den Datenstrom eingefügt und von der Software der Geräte erkannt werden. Es sind keine Steuerleitungen notwendig.

XON/XOFF-Protokoll:

Verbindungsleitungen TxD, RxD und SG



Steuerzeichen

XON = Strg.Q = ASCII-Code DC1 = 11 Hex
XOFF = Strg.S = ASCII-Code DC3 = 13 Hex

Protokollablauf

- Empfangsgerät steuert Sender über XON, XOFF (Puffer voll)
- Sender beginnt mit der Datenübertragung
- Empfänger kann Datenstrom durch Übertragen von XOFF unterbrechen
- Sender wartet auf einen erneuten XON-Befehl

Die Nachfolge-Schnittstelle ist der Universal Serial Bus USB.

Aufgaben

Aufgabe RS232-1: Serielle Schnittstelle RS232

1. Wieso sind beim Software-Handshaking keine Steuerleitungen notwendig ?
2. Welches Protokoll wird beim Software-Handshaking verwendet ?
3. Zeichnen Sie die für das Software-Handshaking notwendigen Verbindungen ein:

PC1 (DEE)

- 1 o
- 2 o
- 3 o
- 4 o
- 5 o
- 6 o
- 7 o

PC2 (DEE)

- o 1 G
- o 2 TD
- o 3 RD
- o 4.. RTS
- o 5 CTS
- o 6 DSR
- o 7 G

4. Was wird mit "Null-Modem" bezeichnet und wie wird es mit einem Kabel realisiert?

7. Parallele Schnittstelle

Stark gekürzt! Komplettes Kapitel siehe Anhang

Der Druckerhersteller Centronics entwickelte in den siebziger Jahren ein Protokoll für die parallele Datenübertragung, das sich zu einem nicht genormten Industriestandard entwickelte. Damit ist eine acht Bit breite, unidirektionale Datenübertragung zwischen PC und Drucker möglich. Die Datenübertragung von einem Peripheriegerät zum PC ist im Centronicsstandard nicht definiert, jedoch durch entsprechende Erweiterungen und Umkonfigurierungen erreichbar. Um die hohe Datenrate des Parallelschnittstellenports in beiden Übertragungsrichtungen besser nutzen zu können, war eine Normierung und Verbesserung notwendig. Diese wurde mit dem **IEEE Std.1284-1994, "Standard Signaling Method for a Bi-directional Parallel Peripheral Interface for Personal Computers"**, erreicht.

Für die Parallelschnittstelle gibt es zur Zeit fünf verschiedene Betriebsarten. Einige der älteren Betriebsarten wurden dabei aus Kompatibilitätsgründen in den neueren integriert. Die wichtigsten sind **Enhanced Parallel Port (EPP)** und **Extended Capability Port (ECP)**.

Die Nachfolge-Schnittstelle ist der Universal Serial Bus USB.