

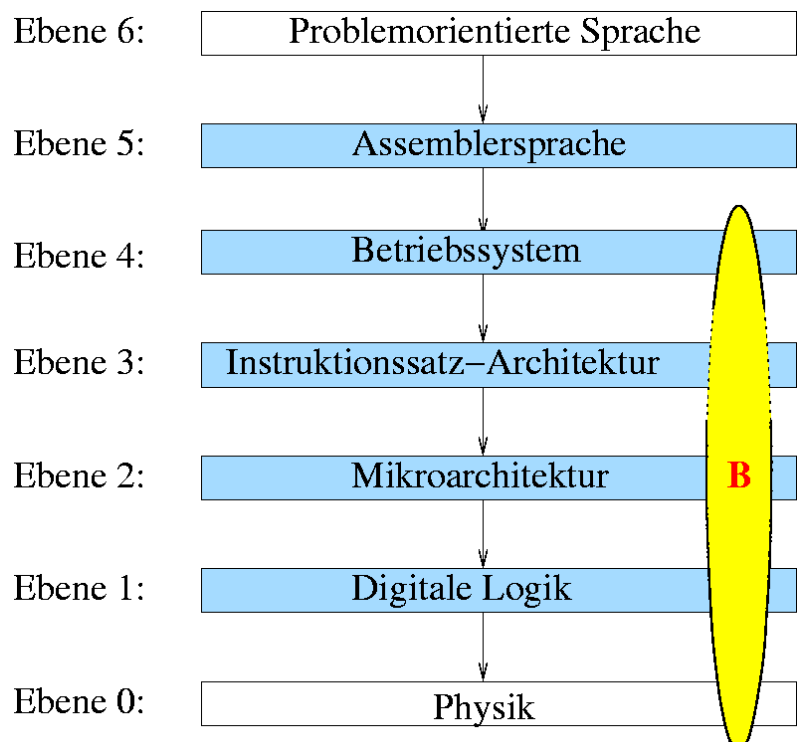
Technische Informatik II

B Hauptspeicher und Cache

1. Begriffe
2. SRAM
3. DRAM
4. DRAM-Varianten: EDO-RAM, SDRAM, DDR-RAM, RAMBUS
5. Festwertspeicher: PROM, EPROM, EEPROM
6. Exkurs: Assoziativspeicher
7. Cache

B Hauptspeicher und Cache

- Einordnung in das Schichtenmodell:

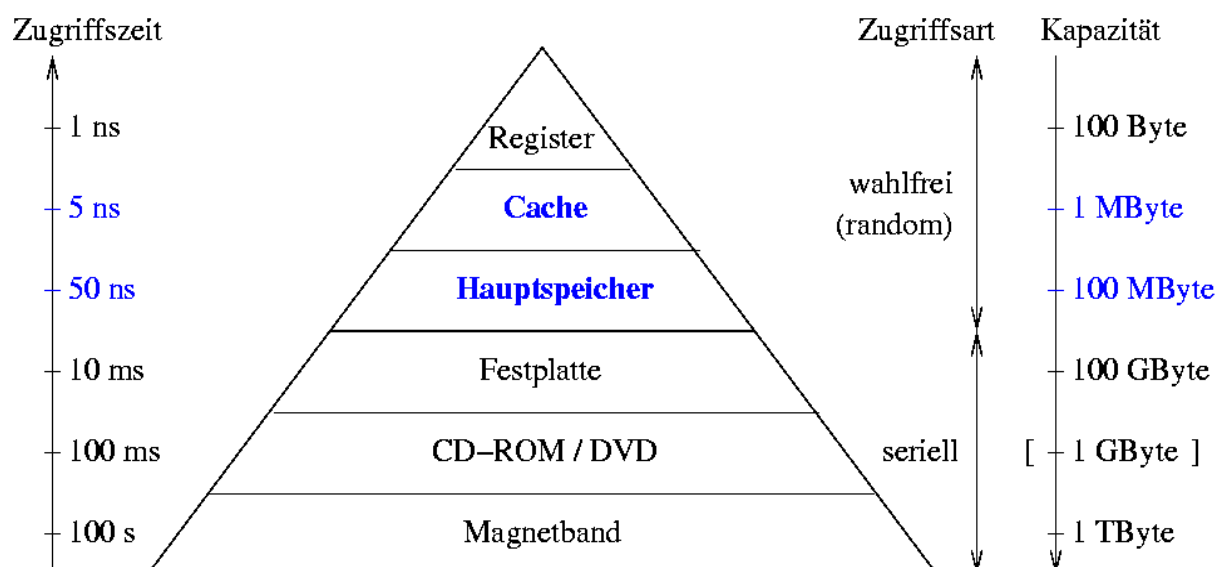


B Hauptspeicher und Cache

- im Laufe der Geschichte wurden verschiedene **Technologien** eingesetzt, um Informationen zu speichern:
 - 1) **Modifikation von Strukturen**: Lochkarte, Schallplatte
 - 2) **Rückkopplung**: Flip-Flops, SRAM
 - 3) **elektrische Ladungen**: Kondensator, DRAM
 - 4) **Magnetismus**: Magnetkernspeicher, Magnetband, Diskette, Festplatte
 - 5) **Optik**: Bar-Codes, CD-ROM, DVD
- Kriterien zum Vergleich von Speichertechnologien:
Geschwindigkeit, Kapazität, Dichte, Energiebedarf, Robustheit, Kosten

B Hauptspeicher und Cache

- in heutigen Rechnersystemen findet man eine bis zu 6-stufige **Speicherhierarchie**:



1 Begriffe

- **RAM** (Random Access Memory)
Speicher mit wahlfreiem Zugriff auf beliebige Adressen
- **SRAM** (Static RAM)
statischer RAM-Speicher
- **DRAM** (Dynamic Random Access Memory)
dynamischer RAM-Speicher
- **SDRAM** (Synchronous DRAM)
synchroner dynamischer RAM-Speicher
- **DDR-SDRAM** (Double Data Rate SDRAM)
synchroner dynamischer RAM-Speicher mit doppelter Datenrate
- **RDRAM** (Rambus DRAM)
spezielle DRAM-Bausteine der Fa. Rambus

1 Begriffe (2)

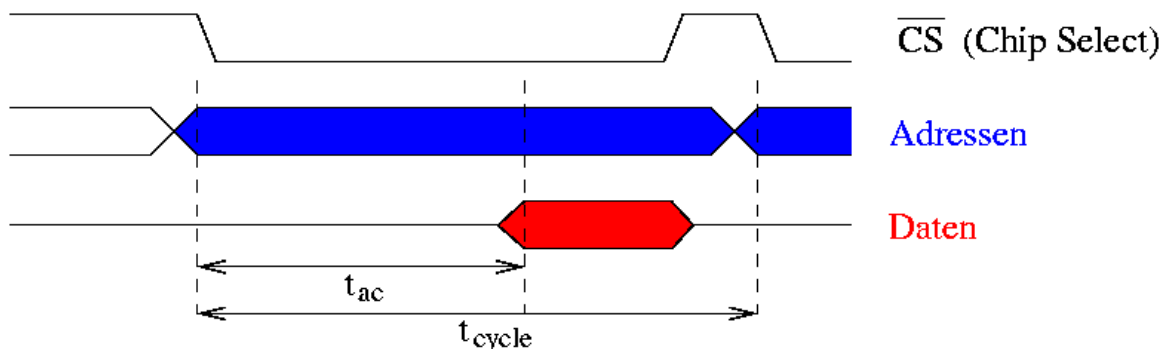
- **ROM** (Read-Only Memory)
- **PROM** (Programmable ROM)
einmalig programmierbarer Speicher, z.B. durch Masken-Programmierung bei der Chip-Herstellung
- **EPROM** (Erasable PROM)
mit UV-Licht löschbares und elektrisch (durch zusätzliche Steuerleitungen und Überspannungen) programmierbares PROM
- **EEPROM** (Electrically Erasable PROM)
elektronisch löschbares und danach wieder programmierbares PROM
- **Flash** (Eigenname), ähnliche Eigenschaften wie EEPROM
- **NV-RAM** (Non-Volatile RAM)
normalerweise SRAM mit eingebauten EEPROM: bei Stromverlust werden die Daten sofort in das EEPROM umkopiert; wenn wieder Strom vorhanden ist, werden Daten vom EEPROM in das SRAM zurückgeschrieben.
- **Batteriegepuffertes RAM**

1 Begriffe (3)

- Speicher werden eingeteilt in
 - 1) **flüchtige** Speicher: SRAM, DRAM, SDRAM, ...
Informationen gehen nach Ausschalten der Versorgungsspannung verloren!
 - 2) **nichtflüchtige** Speicher: PROM, EPROM, EEPROM, ...
Informationen bleiben auch ohne Versorgungsspannung über längere Zeit (typischerweise einige Jahre) erhalten!
- Die **Kapazität** von Speicherbausteinen wird in kByte (bzw. kBit), MByte (bzw. MBit) oder GByte (bzw. MBit) angegeben; es gilt:
 - 1 KByte = 1024 Byte = 2^{10} Byte
 - 1 MByte = 1024 KByte = 1024×1024 Byte = 2^{20} Byte = 1.048.576 Byte
 - 1 GByte = 1024 MByte = $1024 \times 1024 \times 1024$ Byte = 2^{30} Byte = 1.073.741.824 Byte

1 Begriffe (4)

- als **Zugriffszeit** t_{ac} bezeichnet man die Zeitspanne vom Anlegen einer Adresse bis zur Gültigkeit der ausgelesenen Daten
- als **Zykluszeit** t_{cycle} bezeichnet man die Zeitspanne vom Anlegen einer Adresse bis zum möglichen Anlegen der nächsten Adresse
- mögliches **Zeitdiagramm** eines Lesezyklus:



- Zykluszeit ist oft (z.B. bei DRAMs) größer als die Zugriffszeit!

1 Begriffe (5)

- zwei unterschiedliche Arten der Realisierung **flüchtiger** Speicher:

1) Statischer Speicher

- Speicherung durch 4 bis 6 Transistoren je Bit
- kein Refresh notwendig
- Bausteine: SRAM, Dual-ported SRAM
- Zugriffs- und Zykluszeit: ca. 10 ns als externe Bausteine (schneller, wenn sie direkt in den Prozessor integriert werden, z.B. als Cache).

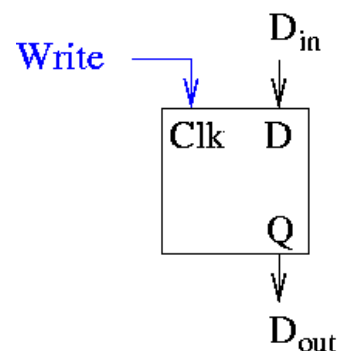
2) Dynamischer Speicher

- Speicherung durch einen Transistor und einen Kondensator je Bit
- Refresh notwendig, da der Kondensator im Verlauf der Zeit (Größenordnung: einige ms) seine Ladung verliert.
- sehr hohe Datendichte (Kapazität je mm² Chipfläche) möglich
- Bausteine: DRAM, SDRAM, DDR-SDRAM, RDRAM
- Zugriffszeit für erstes Datenwort: ca. 40 ns, Zykluszeit: ca. 60 ns

2 SRAM

- **Idee:** auf Flip-Flop Prinzip basierender Speicherbaustein, z.B. ein D-Flip-Flop für ein Bit:

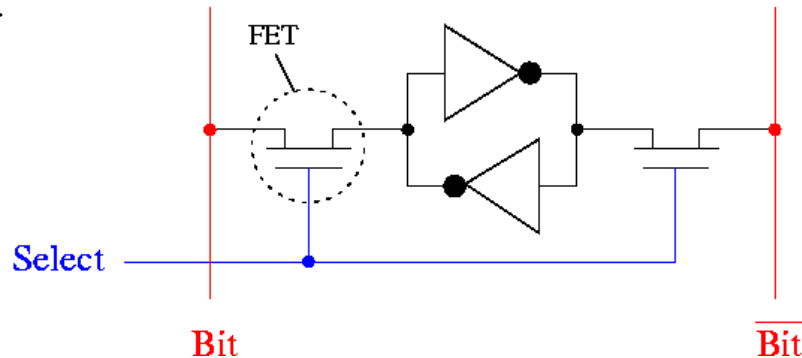
(vgl. Folie F-6 aus TI I)



- bei **Write = 1** wird Information von D_{in} gespeichert und steht am Ausgang D_{out} zur Verfügung
- Information bleibt gespeichert, auch wenn sich D_{in} bei **Write = 0** wieder ändern sollte
(solange Versorgungsspannung anliegt)

2 SRAM (2)

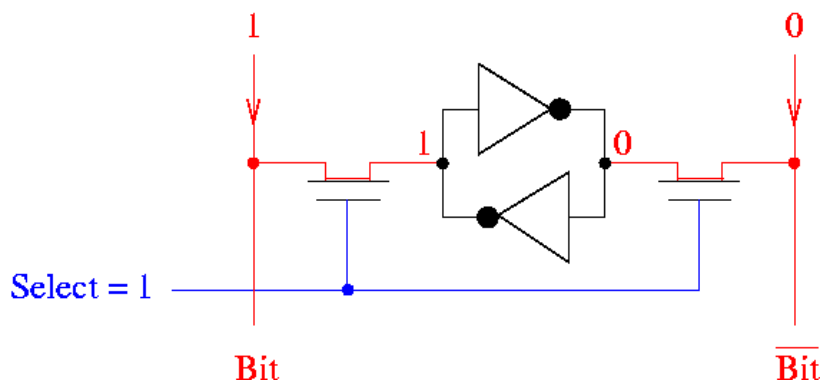
- SRAM-Bausteine basieren jedoch nicht auf vollständigen D-Flip-Flops, sondern auf einfacheren SRAM-Zellen
- vereinfachter Aufbau einer **typischen SRAM-Zelle** mit FET-Transistoren:



- zwei Inverter repräsentieren bistabile Kippstufe
- bei **Select = 1** wird die bistabile Kippstufe mit Leitungen **Bit** und **Bit-bar** verbunden

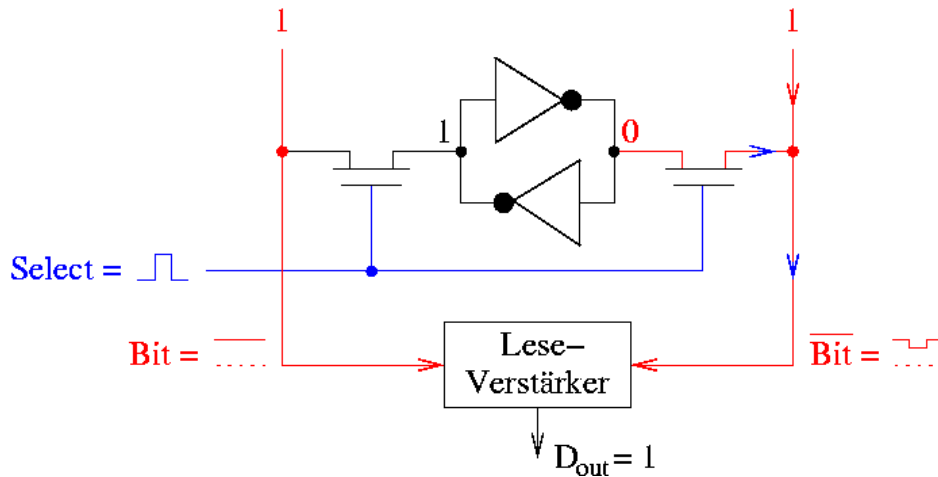
2 SRAM (3)

- **Beschreiben** einer SRAM-Zelle:
 - Bit-Leitungen werden mit **Bit = 1** und **Bit-bar = 0** (zum Speichern einer „1“), bzw. mit **Bit = 0** und **Bit-bar = 1** (zum Speichern einer „0“) geladen
 - Auswahl-Leitung wird auf **Select = 1** gesetzt
 - Kippstufe schwingt ggf. in den anderen stabilen Zustand
 - Rücksetzen der Pegel auf **Select**-Leitung und danach auf **Bit**-Leitungen
 - Zustand bleibt erhalten, solange Versorgungsspannung anliegt



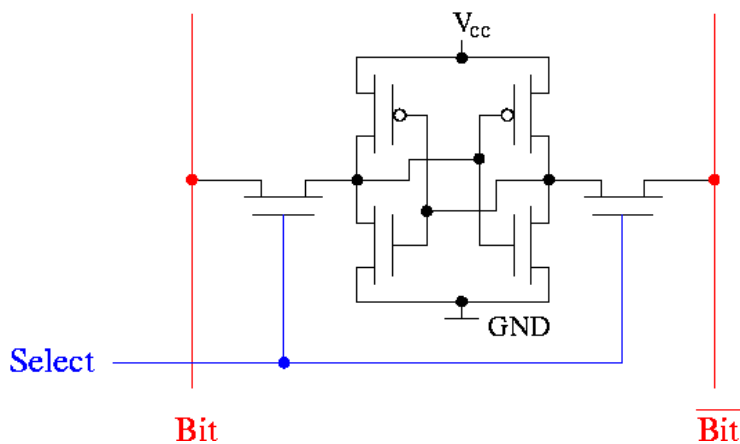
2 SRAM (4)

- **Auslesen** einer SRAM-Zelle:
 - Bit-Leitungen werden mit **Bit = 1** und $\overline{\text{Bit}} = 1$ geladen („Precharging“)
 - auf Auswahl-Leitung wird ein kurzer Impuls gegeben
 - resultierender geringer Spannungsabfall auf **Bit** oder $\overline{\text{Bit}}$ wird durch Verstärkerschaltung erkannt, die eine entsprechende Ausgabe generiert



2 SRAM (5)

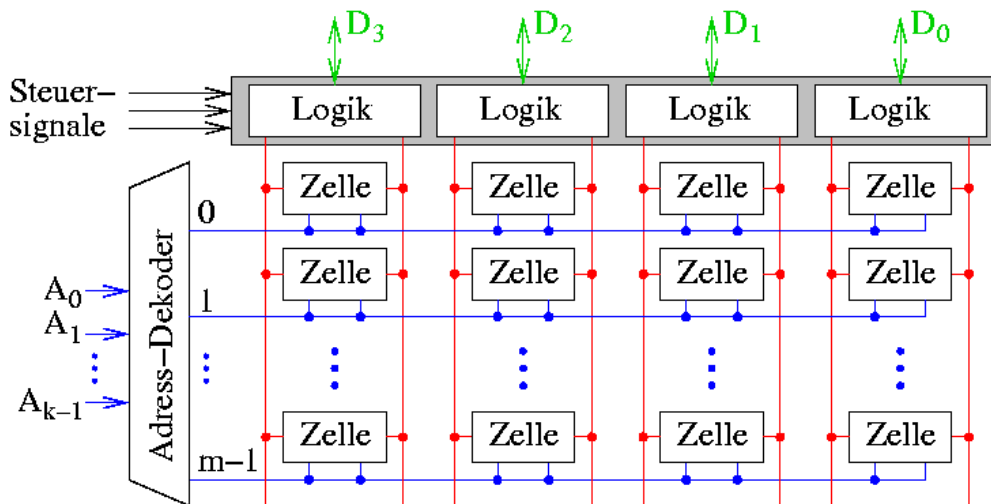
- Realisierung einer SRAM-Zelle mit insgesamt 6 Transistoren (mit 2 zusätzlichen FET-Transistoren je Inverter):



- heute i.a. Realisierung in CMOS-Technologie (Complementary Metal Oxide Semiconductor)

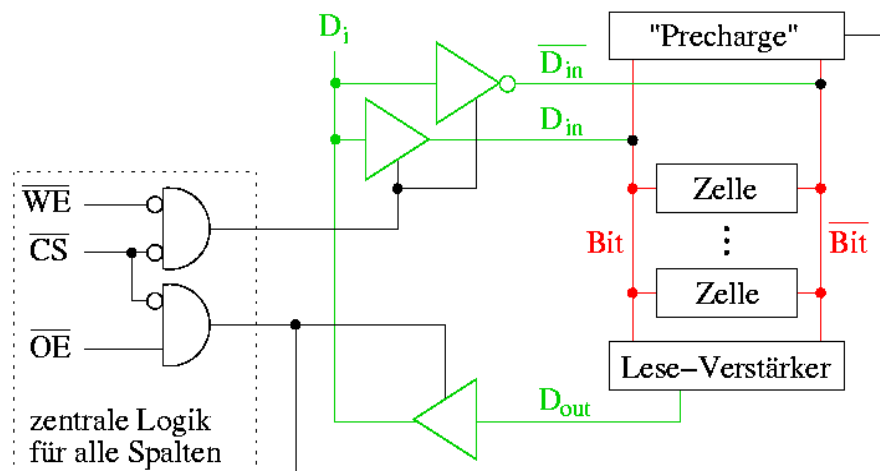
2 SRAM (6)

- **Organisation** eines SRAM-Bausteins: $m \times n$ Speichermatrix (m Speicherzeilen mit je n SRAM-Zellen, Auswahl einer der $m = 2^k$ Zeilen über k Adressleitungen und einen k -zu- m Adress-Dekoder, alle Zellen einer Spalte nutzen die gleichen Bit-Leitungen)
- vereinfachte **Architektur** eines $m \times 4$ SRAM-Bausteins:



2 SRAM (7)

- **Steuerlogik** eines SRAMs mit **Tristate-Puffern** und einigen weiteren Steuerleitungen (i.a. „low active“):
 - $\overline{CS}$ (Chip Select) zur Auswahl und Aktivierung eines SRAM-Bausteins
 - $\overline{WE}$ (Write Enable) zum Speichern eines Wertes
 - $\overline{OE}$ (Output Enable) zum Lesen und Freischalten der Ausgänge
- Realisierung der Steuerlogik für SRAM-Spalte i :



2 SRAM (8)

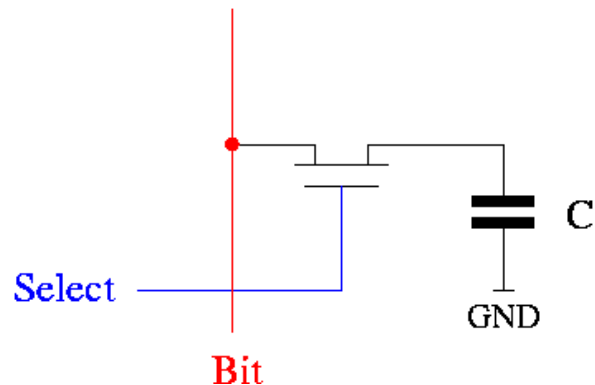
- SRAM-Bausteine gibt es in verschiedenen Organisationsformen, und Kapazitäten, z.B.:
 - **64k × 4** 64k Worte à 4 Bit (16 Adress-, 4 Datenleitungen, 32 KByte)
 - **128k × 8** 128k Worte à 8 Bit (17 Adress-, 8 Datenleitungen, 128 KByte)
 - **512k × 8** 512k Worte à 8 Bit (19 Adress-, 8 Datenleitungen, 512 KByte)
 - **1M × 16** 1M Worte à 16 Bit (20 Adress-, 16 Datenleitungen, 32 MByte)
- oft wahlweise als „*low power*“ oder als „*high speed*“ Version erhältlich
- auch synchrone (d.h. getaktete SRAMs) verfügbar
 - arbeiten synchron mit Prozessortakt
 - sehr kurze Zugriffszeiten (ca. 5 ns)
 - interne Inkrementenschaltung für Adressen erlaubt sehr schnelles getaktetes Schreiben und Lesen von vier Datenworten auf aufeinanderfolgenden Adressen (**Burst-Modus**, vgl. DRAMs)

2 SRAM (9)

- **Vor-/Nachteile** von SRAMs:
 - schneller Zugriff (weniger als 10 ns bei externem Baustein)
 - unempfindlich gegen elektromagnetische Strahlung
 - geringe Datendichte auf dem Chip
⇒ hoher Flächenbedarf (ca. 4-fach im Vergleich zu DRAM)
 - hoher Energiebedarf (bei hoher Geschwindigkeit)
 - hoher Preis
- Typischer **Einsatz** von SRAMs:
 - Netzwerkkomponenten
 - Weltraumgeräte
 - Höchstgeschwindigkeitsrechner (Vektorrechner)
 - L1 und L2 Cachespeicher (mit synchronen SRAMs)

3 DRAM

- benötigt zur Speicherung eines Bits nur einen Transistor und einen Kondensator:
- Aufbau einer **DRAM-Zelle**:



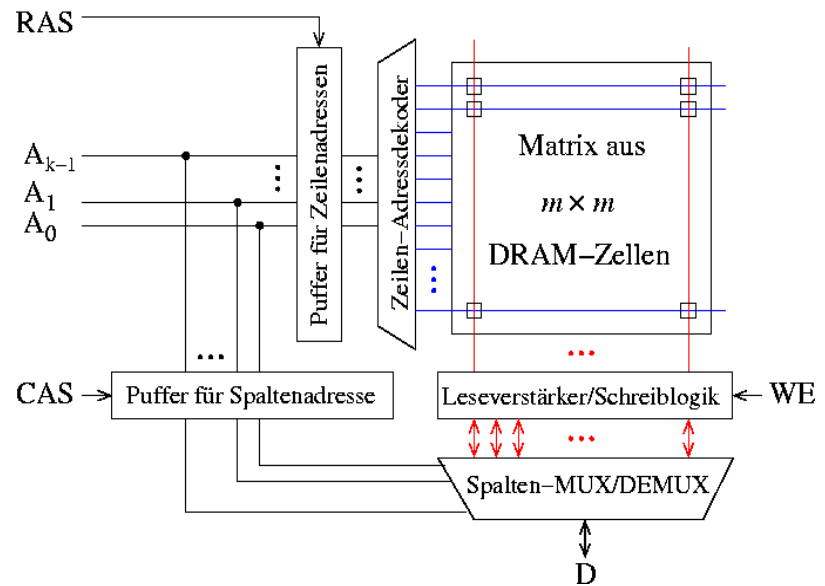
- extrem hohe Speicherkapazität durch geringen Flächenbedarf einer DRAM-Zelle: 128 MBit bis 1 GBit je Chip (Stand 2003)

3 DRAM (2)

- **Schreiben** einer DRAM-Zelle:
 - Bit-Leitung wird auf **Bit = 1** oder **Bit = 0** gesetzt
 - Auswahlleitung **Select = 1** setzen $\Rightarrow$ Transistor leitet und Kondensator wird aufgeladen oder entladen
- **Auslesen** einer DRAM-Zelle:
 - Auswahlleitung **Select = 1** setzen $\Rightarrow$ Transistor leitet
 - falls Kondensator aufgeladen war, wird ein kurzer Impuls auf **Bit**-Leitung erzeugt
 - Impuls kann durch **Leseverstärker** erkannt werden, der eine 1 am Ausgang erzeugt
 - der Kondensator wird jedoch beim Auslesen **entladen**; daher muß die DRAM-Zelle nach jedem Lesen wieder mit dem zuvor gelesenen Wert beschrieben werden!

3 DRAM (3)

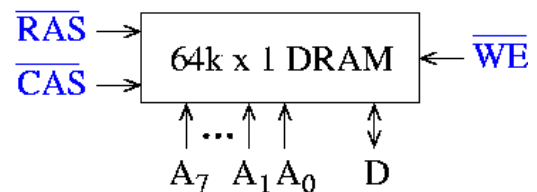
- Organisation eines $m^2 \times 1$ DRAMs (vereinfacht):



- Multiplex-Interface:**
 - jede $2k$ -Bit Adresse (mit $m = 2^k$) besteht aus Zeilen- und Spaltenadresteil
 - um E/A-Pins einzusparen, werden beide Teile sequentiell über k Adressleitungen übertragen ($\Rightarrow$ zusätzliche Steuersignale notwendig)

3 DRAM (4)

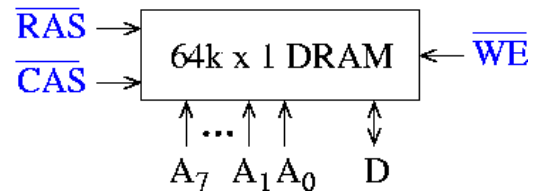
- Lesen eines Wertes aus einem $64k \times 1$ DRAM-Baustein:



- zuerst werden die Adressbits 15 bis 8 an A_7 bis A_0 angelegt
- das Signal **RAS** (*Row Address Strobe*) wird aktiviert und alle 256 Bits Speicherzeile werden von den Leseverstärkern ausgelesen
- dann werden die Adressbits 7 bis 0 an A_7 bis A_0 angelegt
- das Signal **CAS** (*Column Address Strobe*) wird aktiviert, wodurch aus den Ausgängen der 256 Leseverstärker das gewünschte Bit ausgewählt und am Pin D ausgegeben wird
- ein weiteres Bit aus der gleichen Zeile (*Page*) kann ausgelesen werden, indem **CAS** deaktiviert, eine andere Spaltenadresse angelegt wird, und **CAS** wieder aktiviert wird ($\Rightarrow$ **Fast Page Mode, FPM**)
- nach Deaktivierung von **RAS** und **CAS** wird Zeile zurückgeschrieben

3 DRAM (5)

- Schreiben eines Wertes in einen $64k \times 1$ DRAM-Baustein:



- zuerst werden die Adressbits 15 bis 8 an A_7 bis A_0 angelegt
- die Signale **RAS** (*Row Address Strobe*) und **WE** werden aktiviert und alle 256 Bits der Speicherzeile werden von den Leseverstärkern ausgelesen
- dann werden die Adressbits 7 bis 0 an A_7 bis A_0 angelegt
- das Signal **CAS** (*Column Address Strobe*) wird aktiviert und das zu schreibende Bit wird an D angelegt; im Leseverstärker wird das ausgewählte Bit durch das Signal an D überschrieben
- in der ausgewählten Zeile können ggf. weitere Bits geschrieben werden (**Fast Page Mode**, vgl. vorige Folie)
- nach Deaktivierung von **RAS** und **CAS** wird modifizierte Zeile in die Speichermatrix zurückgeschrieben

3 DRAM (6)

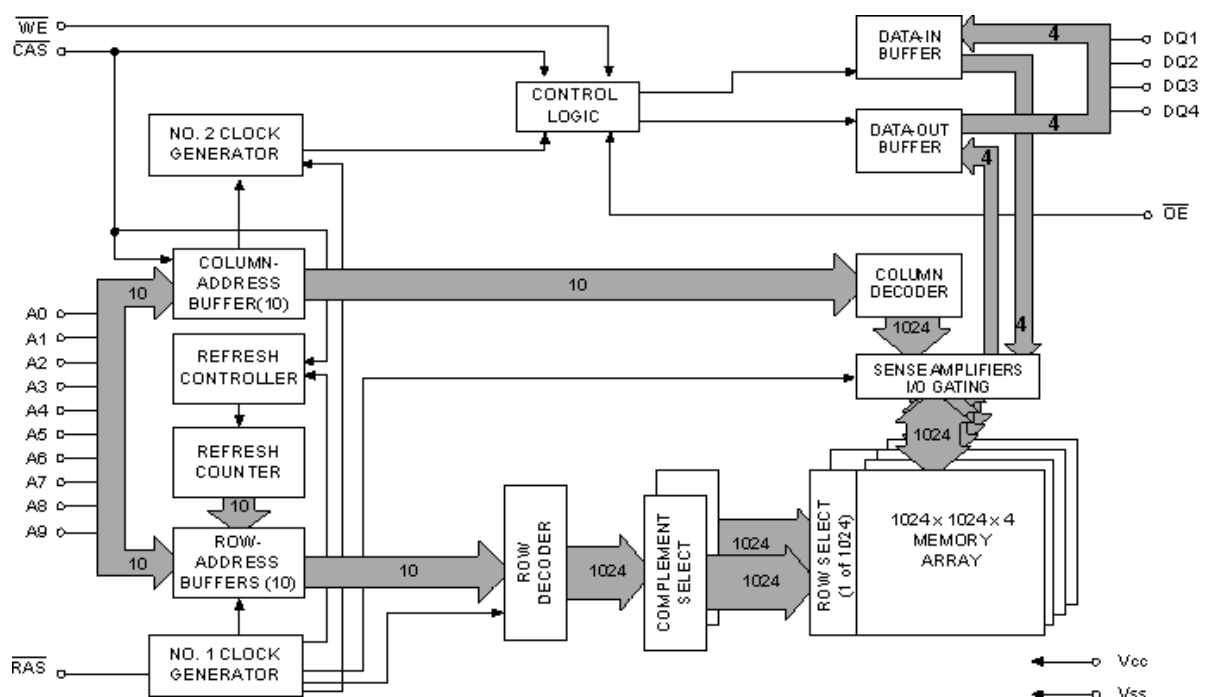
- Kapazität C des Kondensators einer DRAM-Zelle ist sehr gering (einige Femto-Farad, d.h. einige 10^{-15} Farad)
- Ladungsverlust nicht nur bei jedem Lesen, sondern auch
 - langsam mit der Zeit aufgrund eines geringen Leckstroms im Transistor (Größenordnung: einige ms)
 - durch elektromagnetische Strahlung
- periodischer **Refresh** erforderlich, um Zelleninhalt über längere Zeit zu speichern:
 - jede Zeile muss regelmäßig in Abständen von **8** bis **64 ms** gelesen werden, wodurch ihr Inhalt erneut geschrieben wird
 - dies wird implementiert durch einen in das DRAM integrierten **Zähler**, dessen Zählerstand die nächste aufzufrischende Zeilenadresse angibt
 - der Memory-Controller erzeugt periodisch (z.B. alle **15.6 μ s**, $\rightarrow$ BIOS) eine bestimmte Kombination der Steuersignale (z.B. *CAS-before-RAS*), durch die ein **Blindlesezyklus** ausgelöst und Zähler inkrementiert wird

3 DRAM (7)

- der Zugriff auf eine beliebige Speicheradresse benötigt ca. **60 ns** (da Auslesen der sehr geringen Ladung über Leitungen mit hohem Widerstand und das Verstärken eine gewisse Zeit benötigen)
- Folgezugriffe mit anderen Spaltenadressen in der gleichen Speicherzeile benötigen jeweils ca. **30 ns**
- DRAM-Bausteine sind wie SRAMs in unterschiedlichen **Organisationen** verfügbar
(z.B. kann ein 16 MBit Speicher als $1\text{M} \times 16$, $512\text{k} \times 8$, $4\text{M} \times 4$ oder auch als $16\text{M} \times 1$ organisiert sein)
- bei einer Kapazität c und einer Wortbreite von n Bit benötigt man n Speichermatrizen aus $m \times m$ DRAM-Zellen mit $m = \sqrt{c / n}$
- erfüllen einzelne DRAM-Bausteine nicht die Anforderungen bzgl. Wortbreite oder Adreßraum, so können mehrere DRAM-Bausteine geeignet zusammengeschaltet werden

3 DRAM (7)

- detaillierter Aufbau eines $1\text{M} \times 4$ DRAM-Bausteins:

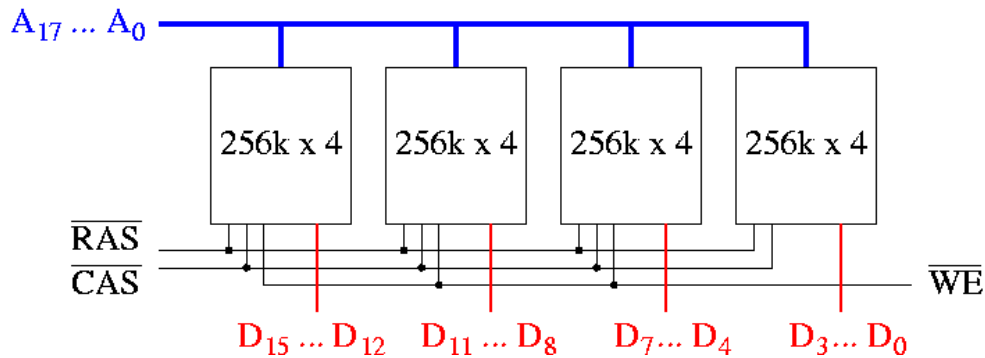


3 DRAM (8)

- Möglichkeiten der Zusammenschaltung von DRAM-Bausteinen:

1) Vergrößern der Wortbreite:

- identische Adress- und Steuer-Leitungen zu allen Speicherbausteinen, Datenleitungen werden aufgeteilt
- Beispiel: Speicher mit 256k Worten à 16 Bit

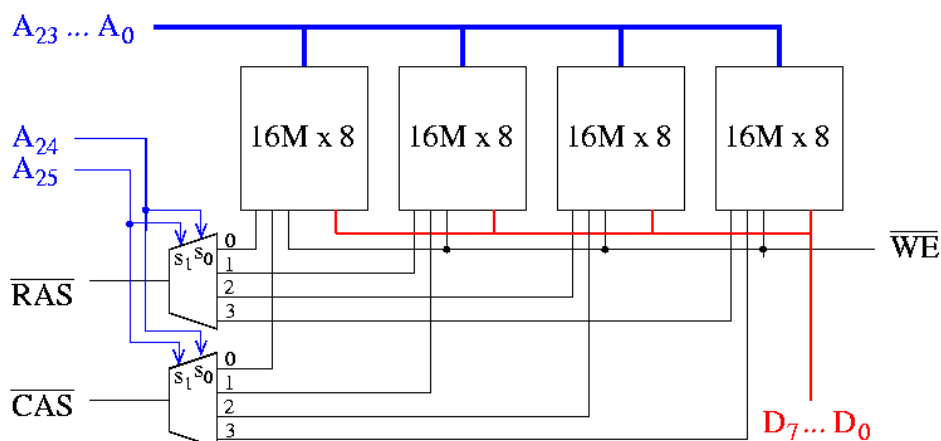


- auch die Speicherbandbreite (Anzahl übertragbarer Bytes je Sekunde zwischen CPU und Speicher) wird hierdurch erhöht!

3 DRAM (9)

2) Vergrößern des Adressraums:

- Datenleitungen und untere Adressleitungen an allen Bausteinen identisch; obere Adressleitungen dienen dem Multiplexen der Steuersignale
- Beispiel: Speicher mit 64M Worten à 8 Bit



- Möglichkeiten 1) und 2) werden i.a. kombiniert !

3 DRAM (10)

- **Vor-/Nachteile** von DRAM-Bausteinen:
 - periodischer Refresh erforderlich
 - hohe Zugriffszeit von ca. 60 ns für das erste Datenwort, dank FPM kürzere Zugriffszeit von ca. 30 ns für folgende Datenworte der gleichen Zeile
 - ca. 4-fach höhere Datendichte als bei SRAM
 - geringer Preis je MBit
- typischer **Einsatz** von DRAMs:
 - Hauptspeicher in PCs und Workstations
 - Pufferspeicher, z.B. in Druckern
- „einfache“ DRAMs sind heute nicht mehr erhältlich, sondern nur noch die schnelleren **DRAM-Varianten** ...

4 DRAM-Varianten

- Möglichkeiten der Beschleunigung des Zugriffs auf die Daten eines DRAM-Bausteins:
 - 1) **Überlappung**: Auslesen eines Datenwortes erfolgt simultan zum Anlegen der *Spaltenadresse* für den nächsten Speicherzugriff
 - 2) **Burst Mode**: es wird eine festgelegte Anzahl von Daten aus *aufeinander folgenden Spaltenadressen* gelesen oder geschrieben, wobei nur die Startadresse von der CPU bereit gestellt wird
 - 3) **Pipelining**: durch eine mit dem Systemtakt *synchrone Arbeitsweise* kann je Taktzyklus *ein neuer Spaltenzugriff* initialisiert bzw. abgeschlossen werden

4 DRAM-Varianten (2)

- **EDO-RAM** (*Extended Data Out RAM*, 1996)
 - Weiterentwicklung des **Fast Page Mode**
 - Daten stehen am Ausgang D durch zusätzliche interne Pufferung auch nach Rücknahme des CAS-Signals noch für eine definierte Zeit zur Verfügung
 - simultan kann bereits die nächste Spaltenadresse (in der gleichen Zeile) angelegt werden
($\Rightarrow$ **Überlappung** von Auslesen eines Speicherwortes und Anlegen der nächsten Adressinformation)
 - reduzierte Zugriffszeit für weitere Daten aus einer Zeile: ca. 20 ns

4 DRAM-Varianten (3)

- **SDRAM** (*Synchronous DRAM*, 1999)
 - **getakteter** Speicherbus (typisch 100, 133 oder 166 MHz)
 - alle Signale werden nur **bei steigender Taktflanke** übernommen und ausgegeben
 - Kombination von Signalen auf CS, WE, RAS, CAS definiert Buszyklus
 - arbeitet mit **Pipelining**: in jedem Taktzyklus kann *eine neue Spaltenadresse* angelegt werden, wobei die zugehörigen Daten eine bestimmte, einstellbare Anzahl von Takten später am Ausgang bereitstehen
 - kann im **Burst Modus** arbeiten: Lesen oder Schreiben einer bestimmten, einstellbaren Anzahl von Datenworten von benachbarten Spaltenadressen (typisch 2, 4 oder 8 Datenworte oder eine ganze Speicherzeile)
 - Zugriffszeit im Burst-Modus: **ca. 6 bis 10 ns**
(entspricht einer *maximalen* Speicherbandbreite von 400 bis 666 MByte/s bei Einsatz eines 32-Bit Speicherbusses, bzw. von 800 bis 1,33 GByte/s bei einem 64-Bit Speicherbus)

4 DRAM-Varianten (4)

- Nomenklatur von SDRAMs: PC $\mathbf{xxx}$ $\mathbf{a-b-c}$
 - $\mathbf{xxx}$ gibt die maximale Taktfrequenz des Speicherbusses an
 - $\mathbf{a}$ gibt die **CAS-Latenzzeit** (CAS *Latency*) an, d.h. die Zeit t_{CL} (in Taktzyklen) von der fallenden Flanke des CAS-Signals bis zur Ausgabe der Daten
 - $\mathbf{b}$ gibt die **RAS-zu-CAS-Verzögerung** (RAS-to-CAS *Delay*) an, d.h. die minimale Zeit t_{RCD} (in Taktzyklen) zwischen Anlegen von RAS und CAS
 - $\mathbf{c}$ gibt die **RAS-Vorladezeit** (RAS *Precharge Time*) an, d.h. die Zeit t_{RP} (in Taktzyklen) zum Beenden des letzten Zugriffszyklus und Vorbereiten des nächsten Zeilenzugriffes (*Precharging*)
- oft wird nur die CAS-Latenzzeit t_{CL} angegeben (z.B. als CL2, oder CL3)
- heute (2003) werden SDRAMs für 100 und 133 MHz bei Kapazitäten von 64 MBit bis 512 MBit angeboten

4 DRAM-Varianten (5)

- DDR-SDRAM (Double Data Rate SDRAM, 2001)
 - pro Takt werden zwei Datenwörter übertragen: je eins bei der **steigenden und** bei der **fallenden Taktflanke**
($\Rightarrow$ Verdopplung der Datenrate im Burst-Modus möglich)
- Nomenklatur bei DDR-SDRAMs: **DDR $\mathbf{xxx}$** oder **PC $\mathbf{yyyy}$**
 - $\mathbf{xxx}$ bezeichnet die **doppelte** Taktfrequenz
 - $\mathbf{yyyy}$ bezeichnet die **maximale Datenrate** bei einem 64-Bit Speicherbus
 - zusätzliche Angaben in der Form $\mathbf{a-b-c}$ bedeuten $\mathbf{a} = t_{CL}$, $\mathbf{b} = t_{RCD}$ und $\mathbf{c} = t_{RP}$ (vgl. SDRAMs)
- heute (2003) werden DDR-SDRAMs für Taktfrequenzen von 133 und 166 MHz (gelegentlich auch 200 MHz) bei Kapazitäten von 128 MBit bis 1 GBit angeboten
 - theoretische maximale Speicherbandbreiten von 2100 bis 3200 MByte/s, praktisch erreichbare Bandbreiten liegen deutlich **niedriger!**

4 DRAM-Varianten (6)

- RDRAM (*Rambus DRAM*, 1999)
 - Entwicklung der Fa. Rambus in Zusammenarbeit mit Intel
 - interner konventioneller DRAM Speicher, gekoppelt mit sehr schnellem komplexen **16-Bit Interface** (RDRAM Kanal bzw. *Channel*):
mit einem **800 MHz Takt** werden bei steigender und bei fallender Taktflanke sowohl Daten als auch **Adressen** übertragen
(⇒ Übertragungsrate von **1.6 GByte/s** möglich)
 - spezieller RDRAM Memory-Controller erforderlich; Kommunikation zwischen Controller und RDRAM-Bausteinen erfolgt in **Paketen**
 - je RDRAM Memory-Controller sind zwei parallele RDRAM Kanäle möglich, so daß sich eine **32-Bit** Datenbreite ergibt
(⇒ Übertragungsrate von **3.2 GByte/s** möglich)
 - zukünftige Verbreitung in PCs unklar, zumal zwei- bis dreimal so teuer wie DDR-SDRAMS (Stand 2003)
 - jedoch häufiger Einsatz in Geräten der Unterhaltungselektronik

4 DRAM-Varianten (7)

- Zusammenfassung typischer Werte (Stand 2003):

<i>Variante</i>	<i>mit Takt</i>	<i>Zykluszeit</i>	<i>Zugriffszeit</i>	<i>Zugriffszeit Folgezugriffe</i>
DRAM (FPM)	–	80 ns	60 ns	30 ns (25 MWorte/s)
EDO-RAM	–	80 ns	50 ns	20 ns (50 MWorte/s)
SDRAM	133 MHz	60 ns	40 ns	7,5 ns (133 MWorte/s)
DDR-SDRAM	133 MHz	60 ns	40 ns	3,75 ns (266 MWorte/s)
RDRAM	800 MHz	70 ns	45 ns	1,25 ns (800 MWorte/s)

- eine einzelne DRAM-Speicherzelle ist kaum schneller geworden; lediglich das Interface wurde so verbessert, dass Zugriffe auf aufeinander folgende Daten wesentlich beschleunigt werden!
⇒ sinnvoll für Caches, da hier stets komplette Cachezeilen gelesen bzw. geschrieben werden
(Cachezeile beim Pentium III: 32 Byte, beim Pentium 4: 64 Byte)

4 DRAM-Varianten (8)

- weitere DRAM-Entwicklungen:
 - **mehrere** (typisch 2 oder 4) interne **Speicherbänke** je Speicherbaustein: während des Lesens von Daten aus einer Zeile kann bereits eine andere Zeile adressiert werden, sofern diese in einer anderen Speicherbank ist (⇒ höherer Datendurchsatz möglich)
 - **DDR2**-SDRAMs ermöglichen höhere Taktraten als DDR-SDRAMs durch modifizierte Schaltungstechnik und niedrigere Versorgungsspannung
 - Speicherbausteine mit **QBM** (*Quad Band Memory*) Technologie erlauben im Vergleich zu SDRAMs eine vierfache Datenrate durch das Übertragen von vier Datenworten je Taktzyklus
 - **RLDRAM** (*Reduced Latency DRAM*) ermöglicht durch einen Verzicht auf das Multiplexing von Zeilen- und Spaltenadressen kürzere Latenzzeiten bei wahlfreiem Speicherzugriff

5 Speichermodule

- Speicherbausteine werden nicht einzeln verwendet, sondern als **Module**



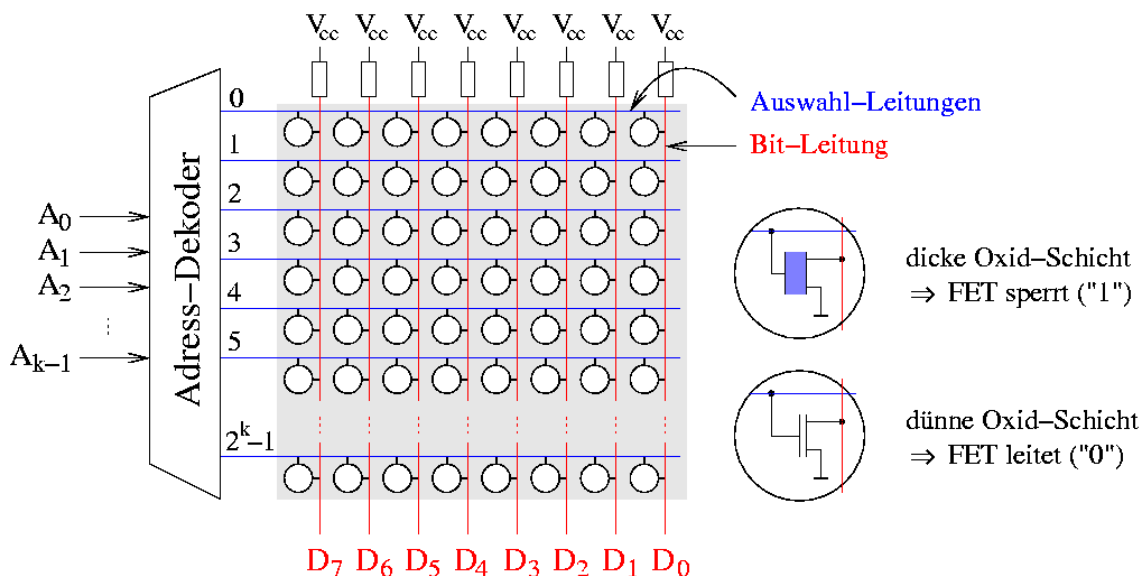
- **SIMM** (*Single Inline Memory Module*) oder **PS/2-Modul**
 - einseitig bestückt, 72 Kontakte nur auf einer Seite der Speicherplatine, 32 Datenbits (⇒ jeweils 2 identische Module für 64-Bit Bus erforderlich)
- **DIMM** (*Dual Inline Memory Module*)
 - Kontakte beidseitig, i.a. zweiseitig bestückt, 64 Datenbits
 - bei Verwendung von SDRAMs: 168 Kontakte, bei Verwendung von DDR-SDRAMs: 184 Kontakte
 - heute (2003) von 128 MByte bis 1 GByte erhältlich

5 Speichermodule (2)

- **RIMM** (*Rambus Inline Memory Module*)
 - 184 Kontakte beidseitig, i.a. zweiseitig bestückt, 16 Datenbits
 - un belegte Sockel sind mit einem Dummy-Modul (C-RIMM) zu belegen
 - heute (2003) von 128 MByte bis 512 MByte erhältlich
- neben den normalen („*unbuffered*“) Speichermodulen gibt es auch solche mit zusätzlichen **Registern** („*registered*“) zum Aufbau von Servern mit sehr großem Arbeitsspeicher
- auch Speichermodule mit **Fehlererkennung** oder **Fehlerkorrektur** verfügbar:
 - zusätzlicher Speicherbaustein für Prüfbits erforderlich
 - **Parity-Module** gestatten eine *Fehlererkennung*, 1 zusätzliches Prüfbit
 - **ECC-Module** (*Error Correcting Code*) gestatten eine *Fehlerkorrektur*, 8 zusätzliche Prüfbits bei 64-Bit Modulen

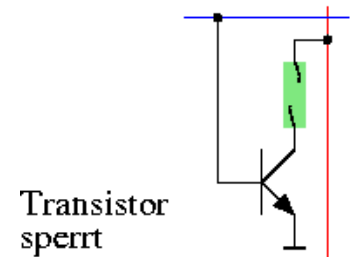
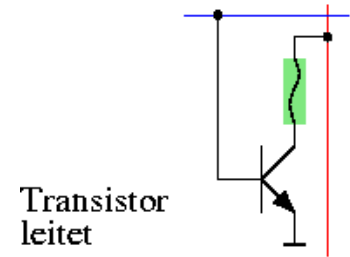
5 Nichtflüchtige Speicher

- ein Festwertspeicher (**ROM** = *Read Only Memory*) hat auch eine matrixartige Architektur; der FET in einer jeden Zelle wird über eine Maske bei der Produktion permanent leitend oder sperrend



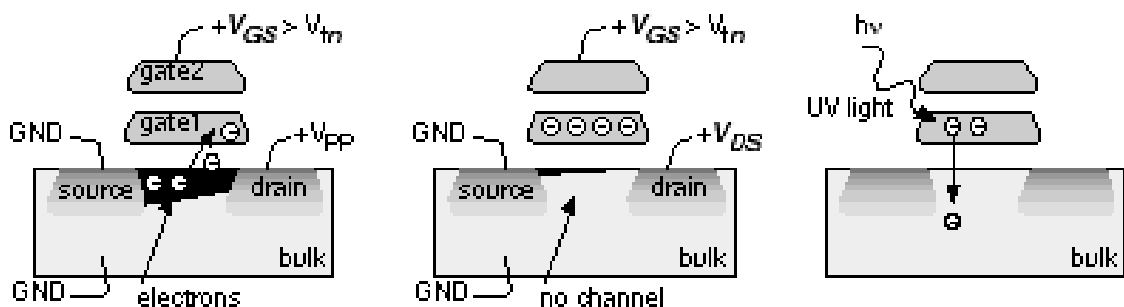
5 Nichtflüchtige Speicher (2)

- bei einem **PROM** (Programmable ROM) sind die Transistoren nur über eine **sehr dünne Metallschicht** (*fuse*) mit der Bitleitung verbunden
 - bipolare Transistoren anstatt FETs
 - alle Transistoren eines PROMs sind zunächst leitend
 - durch einen kurzen Überspannungsimpuls auf der Bitleitung kann die Metallschicht verdampft werden ($\Rightarrow$ Transistor sperrt)
 - Programmierung ist irreversibel
 - sehr kurze Zugriffszeiten (ca. 5 ns) beim Lesen möglich
 - auch zur Realisierung logischer Schaltungen geeignet



5 Nichtflüchtige Speicher (3)

- ein **EPROM** (Erasable PROM) basiert auf Floating Gate FETs:
 - FET ist zunächst leitend
 - bei Anlegen einer hohen Spannung V_{GS} (12V) an gate2 entsteht ein elektrisches Feld, das die Elektronen auf das „Floating Gate“ (gate1) springen lässt ($\Rightarrow$ FET sperrt)
 - Ladung auf „Floating Gate“ bleibt einige Jahre erhalten
 - erst durch UV Licht werden Elektronen wieder freigesetzt ($\Rightarrow$ FET leitet)
 - Programmierung erfordert spezielles Programmiergerät
 - Lebensdauer: einige 100 Lösch-/Brennvorgänge



5 Nichtflüchtige Speicher (4)

- ein **EEPROM** (*Electronically Erasable PROM*) basiert auf Floating Gate Tunnel Oxide FET Transistoren:
 - dünne Oxidschicht, durch die bei Anlegen *negativer* Spannungen V_{GS} die Elektronen zurückspringen können
 - bis zu 10000 Schreibvorgänge möglich
 - Löschen einzelner Werte möglich
 - Zeit für Lesen eines Wertes: 35 ns, für das Schreiben jedoch 5 bis 10 ms
- **Flash**-Speicher arbeiten ähnlich wie EEPROMs, haben jedoch günstigere Eigenschaften
 - 60 bis 150 ns Zugriffszeit für Lesen (entspricht bis zu 16 MWorte/s)
 - Zeit für Schreiben eines Wertes: ca. 5 μ s
 - bis zu 100000 Schreibvorgänge möglich
 - hohe Kapazität, heute (2003) bis zu 64 MByte
 - nur blockweises Löschen möglich

5 Nichtflüchtige Speicher (5)

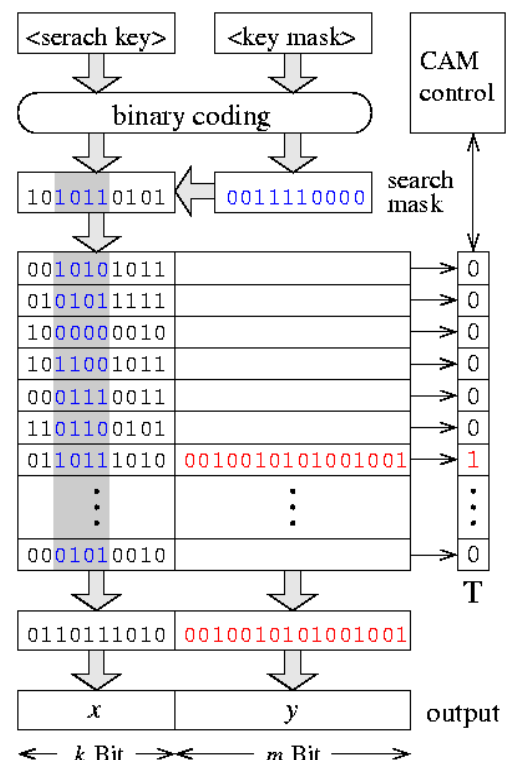
- Möglichkeiten der Ansteuerung von Festwertspeichern:
 - **bitseriell**: serielles Interface, bestehend aus Taktleitung und gemeinsamer Leitung für Daten und Adressen
 - **parallel**: mehrere Daten- und Adressleitungen
- typische Anwendungen von Festwertspeichern:
 - Speicherung des **BIOS** (*Basic Input/Output System*) beim PC
 - Speicherung aller Programme bei kleineren **eingebetteten Systemen**
 - in Geräten der Kommunikationstechnik und Unterhaltungselektronik zur Speicherung von Rufnummern, Senderfrequenzen, ...
 - als Flashkarten zur **Speicherung von Bildern** in digitalen Fotoapparaten, für **Musikdateien** beim MP3-Player, ...
 - in USB **Memory-Sticks** zur Speicherung von Daten oder Programmen
 - Initialisierung von **FPGAs**

6 Exkurs: Assoziativspeicher

- alternative Möglichkeit der Speicherung von Informationen in einem Computer: **Assoziativspeicher (inhaltsadressierbarer Speicher bzw. CAM = Content Addressable Memory)** :
 - bei einem RAM muss die **Adresse** bekannt sein, um ein bestimmtes Datum aus dem Speicher zu holen
 - ein CAM kann alle gespeicherten Daten ermitteln, von denen ein Teil des **Inhalts** bekannt ist
 - Konzept bereits 1943 von **Zuse** vorgestellt, jedoch erst mit Aufkommen der Halbleitertechnik in Hardware realisiert
 - ein CAM ermöglicht die schnelle Realisierung von **Suchverfahren** (z.B. in Datenbanken oder auf dem Gebiet der Künstlichen Intelligenz)
- heute weitgehend ersetzt durch schnelle Algorithmen (z. B. Hashing-Techniken), die auf konventionellem Speicher arbeiten

6 Exkurs: Assoziativspeicher (2)

- allgemeiner Aufbau eines CAM:
 - jede CAM-Speicherzeile besteht aus
 - k -Bit **Schlüssel x (key)**
 - m -Bit **Datenfeld y**
 - k -Bit Suchschlüssel (**search key**)
 - Suchmaske (**key mask**) bestimmt relevanten Teil
 - Suche erfolgt **teil-** oder **vollparallel** im nicht ausmaskierten Teil des Schlüsselfelds mittels zusätzlicher Logik
 - Trefferregister T** zeigt an, in welchen Zeilen Übereinstimmung vorlag
 - sequentielles Auslesen/Verarbeiten der Treffer durch Steuerlogik



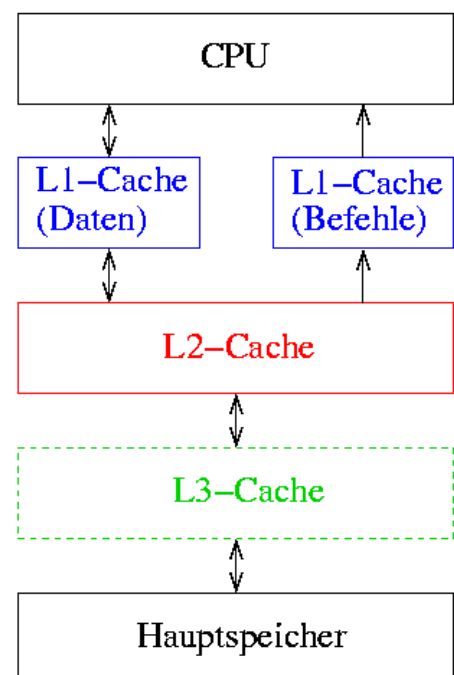
7 Caches

- aufgrund immer höherer Taktfrequenzen, der superskalaren Prozessor-Architektur und „*Out-of-Order*“ Befehlsausführung nehmen die Anforderungen moderner CPUs an den Speicher ständig zu; benötigt werden:
 - **kurze Zugriffszeit**
 - **hohe Transferrate**
 - Speichermodule aus aktuellen DRAM-Varianten erreichen zwar eine relativ hohe Transferrate; die Zugriffszeit ist bei wahlfreier Adressierung ist jedoch völlig unzureichend!
- ⇒ durch Einsatz einer **Speicherhierarchie** (aus gestaffelt schnellen Speichern) soll ein Speicher aus DRAM-Bausteinen ähnlich schnell werden wie ein Speicher aus SRAM-Bausteinen

7.1 Speicherhierarchie

heutige Rechner verfügen über eine mehrstufige Speicherhierarchie:

- **prozessorinterne L1 Caches** für Code und Daten; Zugriff in 1-3 Takten; typische Größe von 8–64 KByte, SRAM
- **prozessorexterner**, aber jedoch auf dem gleichen Chip integrierter (für Code und Daten gemeinsamer) **L2 Cache**; Zugriff in 4-8 Takten; typische Größe von 256 KByte bis 16 MByte, SRAM
- in einigen Systemen zusätzlicher externer **L3 Cache**; typische Größe 4-64 MByte
- **Hauptspeicher** (DRAM) von 64 MByte bis zu einigen GByte; Zugriff in ca. 40 ns

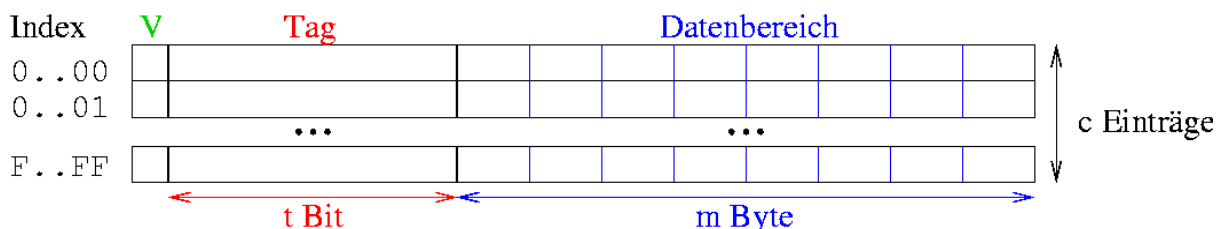


7.1 Speicherhierarchie (2)

- Cache auf Ebene $i+1$ ist größer und langsamer als auf Ebene i
- jeder Cache arbeitet wie ein **Assoziativspeicher** für Speichereinträge des Hauptspeichers:
 - der **Schlüssel** entspricht der Speicheradresse
 - das **Datenfeld** enthält den Speicherinhalt
- bei jedem Speicherzugriff wird beginnend beim Cache auf der Ebene $i = 1$ überprüft, ob die Speicheradresse als Schlüssel gespeichert ist:
 - falls vorhanden (**Cache Hit**), enthält der Cache der Ebene i eine Kopie des Speicherinhaltes
 - falls nicht vorhanden (**Cache Miss**), wird der Cache auf Ebene $i+1$ bzw. der Hauptspeicher konsultiert und die adressierten Daten werden für spätere Zugriffe im Cache der Ebene i gespeichert

7.2 Cache-Aufbau

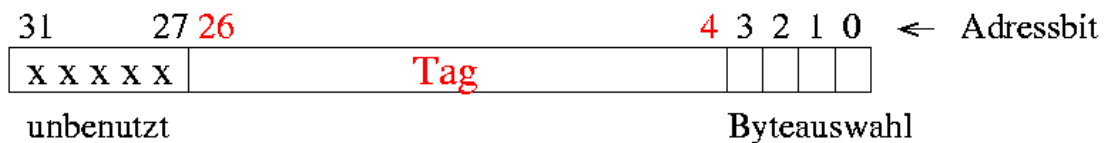
- ein Cache besteht aus $c = 2^k$ Cache-Zeilen
- jede Cache-Zeile (*Cache Line* oder *Cache Block*) besteht aus einem **t -Bit Identifikator (*Tag*)**, einem **Datenbereich**, einem Index und **Gültigkeits-Flag V (*Valid Bit*)**:



- **Datenbereich** besteht aus $m = 2^d$ Bytes (typisch: $m = 8, 16, 32$); m wird Eintragsgröße (*Block Size* oder *Line Size*) genannt
- der **Tag** enthält einen Teil der Speicheradresse
- Größe des Cache-Speichers: $S_c = c \cdot m$

7.3 Vollassoziativer Cache

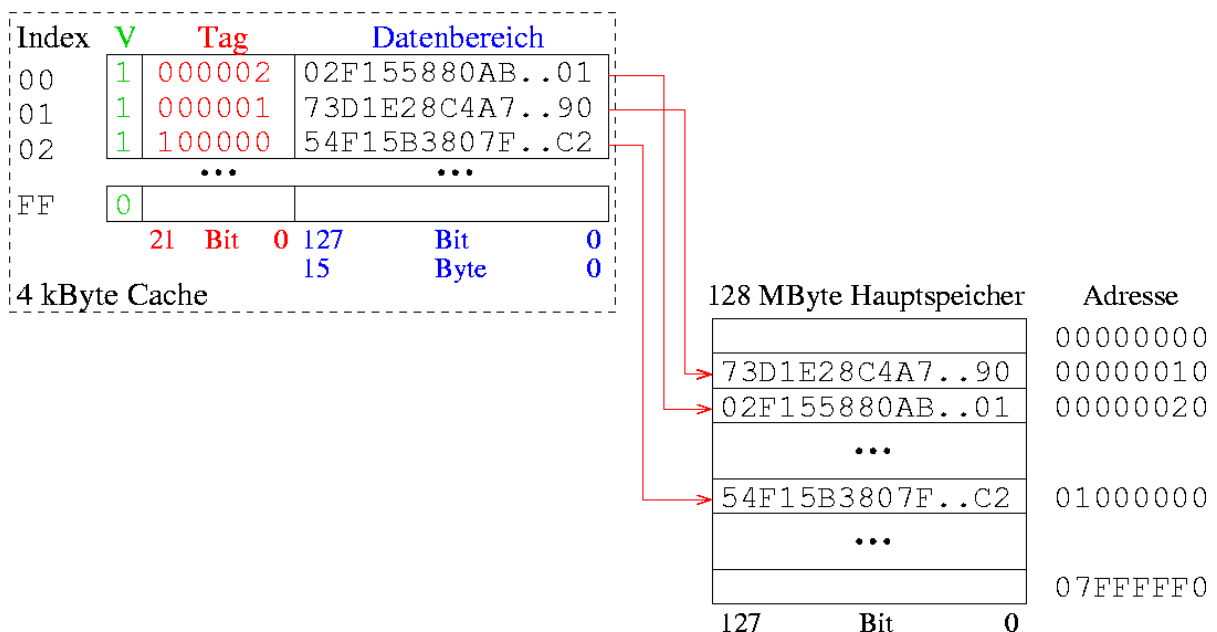
- ein Hauptspeichereintrag mit $m = 2^d$ Bytes ab Adresse a wird in **beliebiger** Cache-Zeile abgelegt
 - es muss gelten: $a \bmod m = 0$
 - bei $a \bmod m \neq 0$ würde Speicherung in zwei Cache-Zeilen erfolgen
- für einen Hauptspeicher der Größe $S_M = 2^w$ Byte ist ein **Tag** von $t = w - d$ Bit erforderlich
 - Beispiel: 32-Bit Adressen, 128 MByte Hauptspeicher ($\Rightarrow w = 27$), Cache der Größe 4 KByte mit Zeilen aus je $m = 16$ Bytes ($\Rightarrow d = 4$)



- niedrige d Bit einer Adresse nicht im **Tag** enthalten; sie dienen der **Auswahl eines Bytes** aus Cache-Zeile

7.3 Vollassoziativer Cache (2)

- Aufbau und Arbeitsweise eines vollassoziativen 4 KByte Caches mit 256 Zeilen à 16 Byte bei einem 128 MByte Hauptspeicher:

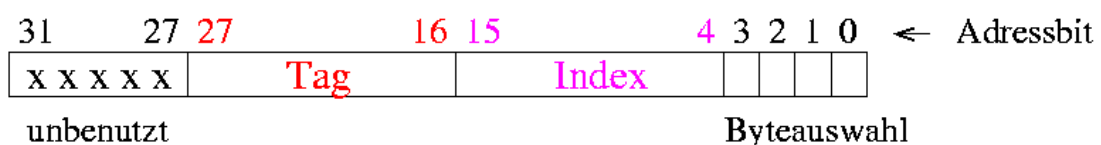


7.3 Vollassoziativer Cache (3)

- bei Zugriff auf Adresse a wird das **Tag** aller gültigen Cache-Zeilen i.a. gleichzeitig (**vollparallel**) mit den entsprechenden Adress-Bits von a verglichen werden
- Aufwand: $(w-d) \cdot c$ Bit-Vergleiche
ein c -Bit Treffer-Register
sowie Baum aus UND- und ODER-Gattern zur Bestimmung eines *Cache Hit* Signals für CPU aus Treffer-Registern
- alternativ kann der Vergleich auch mit geringerem Aufwand **teilparallel** erfolgen (d.h. sequentiell über alle gültigen Zeilen *oder* über alle **Tag**-Spalten); ist jedoch bedeutend langsamer!
- vollassoziativer Cache wird **nur für kleine Caches** verwendet!
(ansonsten ist Aufwand zu hoch und die Trefferbestimmung aufgrund der aufwendigen Logik zu langsam)

7.4 Direkt abbildender Cache

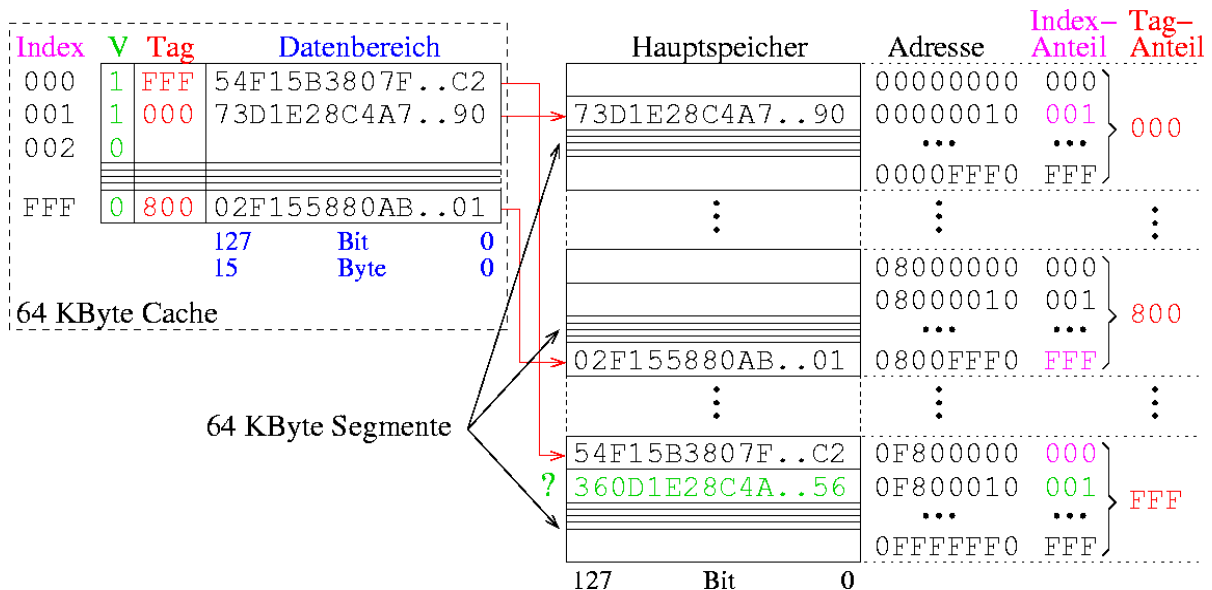
- Hauptspeicher wird in **gleich große Segmente** unterteilt:
 - Größe des Hauptspeichers $S_M = 2^w$ Byte
 - Aufteilung in $s = 2^t$ der Cachegröße S_C entsprechende Segmente
 - es muss gelten: $S_M = s \cdot S_C = s \cdot c \cdot m$
- bei einem Hauptspeicher der Größe $S_M = 2^w$ Byte und einem Cache mit $c = 2^k$ Zeilen aus jeweils $m = 2^d$ Byte ist ein **Tag** von $t = w - k - d$ Bit erforderlich
 - Beispiel: 32-Bit Adressen, 256 MByte Hauptspeicher ($\Rightarrow w = 28$), 64 KByte Cache mit 4096 Zeilen ($\Rightarrow k = 12$) aus je 16 Bytes ($\Rightarrow d = 4$)



- **Index**-Anteil gibt **direkt** die Cache-Zeile an!

7.4 Direkt abbildender Cache (2)

- Aufbau und Arbeitsweise eines direkt abbildenden 64 KByte Caches mit 4096 Zeilen à 16 Byte bei einem 256 MByte großen Hauptspeicher:



7.4 Direkt abbildender Cache (3)

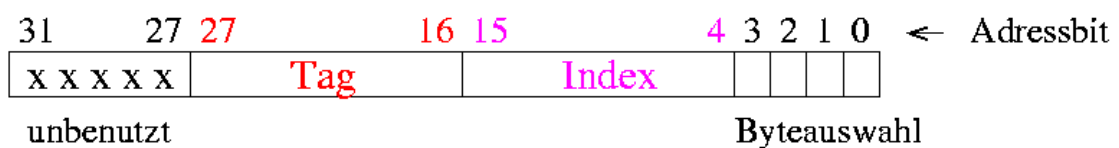
- jede Hauptspeicherzeile kann nur **direkt** auf **eine bestimmte** Cache-Zeile (*direct mapped cache*) abgebildet werden
- s Hauptspeicherzeilen mit gleichem Indexanteil (d.h. mit Adress-Distanz 2^{w-t}) **konkurrieren** um eine einzige Cache-Zeile, in der sie gespeichert werden können!
 - bei einem Konflikt muss entsprechende Zeile erst freigegeben werden, bevor sie erneut belegt wird
- bei Eintrag der i -ten Hauptspeichereile des j -ten Segmentes
 - entspricht i dem **Index**
 - entspricht j dem **Tag**
- bei Cache-Zugriff muss nur ein t -Bit *Tag* mit den zugehörigen t Bit der Adresse verglichen werden
(Aufwand: t Bit-Vergleiche, kein Assoziativspeicher!)

7.5 n -Wege teilassoziativer Cache

- **Mischform** aus einem **vollassoziativen** und einem **direkt abbildenden** Cache:
 - Hauptspeicher ist wie beim direkt abbildendem Cache in $s = 2^t$ Segmente der Größe S_M / s unterteilt
 - auch der Cache wird in n Partitionen (*Sets*) der Größe S_C / n unterteilt (*n-Way Set-Associative Cache*)
 - jede Partition ist ein direkt abbildender Cache; assoziativer Zugriff auf entsprechende Zeilen aller n Partitionen
- jede Hauptspeicherzeile kann in einer **beliebigen** Partition, dort aber nur in einer **bestimmten** Cache-Zeile abgespeichert werden
 - bei einer Kollision kann eine andere Partition ausgewählt werden, sofern dort in der entsprechenden Zeile ein Eintrag kollisionsfrei möglich ist

7.5 n -Wege teilassoziativer Cache (2)

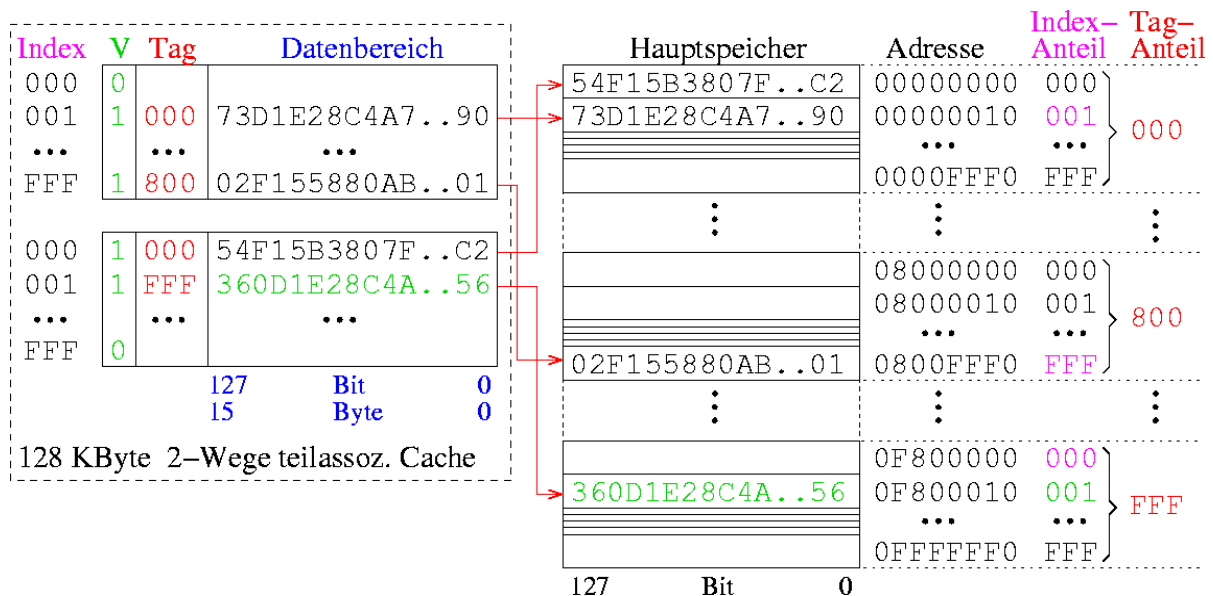
- bei einem Hauptspeicher der Größe $S_M = 2^w$ Byte und einem n -Wege teilassoziativem Cache mit $c = 2^k$ Zeilen je Partition und Cache-Zeilen aus $m = 2^d$ Byte ist ein **Tag** von $t = w - k - d$ Bit erforderlich
 - resultierende Cache-Gesamtgröße: $S_C = n \cdot c \cdot m$ Byte $= n \cdot 2^k \cdot 2^m$ Byte
 - Beispiel: 32-Bit Adressen, 256 MByte Hauptspeicher ($\Rightarrow w = 28$), 2-Wege teilassoziativer Cache der Größe 128 KByte, d.h. 2 Partitionen à 64 KByte mit jeweils $c = 4096$ Zeilen ($\Rightarrow k = 12$) und Cache-Zeilen aus $m = 16$ Bytes ($\Rightarrow d = 4$)



- Adressbildung erfolgt wie bei einem direkt abbildendem Cache der Größe S_C / n

7.5 n -Wege teilassoziativer Cache (3)

- Aufbau und Arbeitsweise eines 2-Wege teilassoziativen Caches der Größe 128 KByte mit Zeilen à 16 Byte bei einem 256 MByte Hauptspeicher:



7.5 n -Wege teilassoziativer Cache (4)

- sämtliche n Zeilen mit gleichem Index stellen jeweils einen **Assoziativspeicher** dar
- s Hauptspeicherzeilen mit gleichem Indexanteil (d.h. mit Adress-Distanz 2^{w-t}) **konkurrieren** um n Cache-Zeilen!
(Allgemein gilt $n \ll s$, d.h. Konflikte sind möglich, aber seltener als beim direkt abbildenden Cache)
- beim Eintrag der i -ten Hauptspeichereile des j -ten Segmentes
 - entspricht i dem **Index**
 - wird eine Partition ermittelt, in der eine Cache-Zeile mit **Index** i frei ist
 - wird dort j als **Tag** eingetragen
- bei einem Cache-Zugriff müssen die t -Bit *Tags* aller n Zeilen mit den zugehörigen t Bit der Adresse verglichen werden
(n parallele Vergleiche, Aufwand: $n \cdot t$ Bit-Vergleiche)

7.6 Lese- und Schreibzugriffe

- Folgende Situationen können bei einem **Lesezugriff** auf eine Cache-Zeile auftreten:
 - **Read Hit**: bei einem erfolgreichen Lesezugriff (d.h. *Tag* und die zugehörigen Adress-Bits stimmen überein)
 - wird ein Datum aus typisch 1, 2, 4, 8 oder m Byte aus Cache-Zeile in den Befehlspuffer bzw. in ein Register der CPU geladen
 - **Read Miss**: bei einem erfolglosen Lesezugriff auf Adresse a
 - wird stets eine **komplette Cache-Zeile** mit m Bytes (Burst-Modus !) ab der Adresse $a - a \bmod m$ aus dem Hauptspeicher bzw. aus dem Cache der nächsten Hierarchie-Stufe eingelesen
 - muss ggf. eine Cache-Zeile ersetzt werden, z.B. gemäß der **LRU-Strategie** (*Least Recently Used*)

7.6 Lese- und Schreibzugriffe (2)

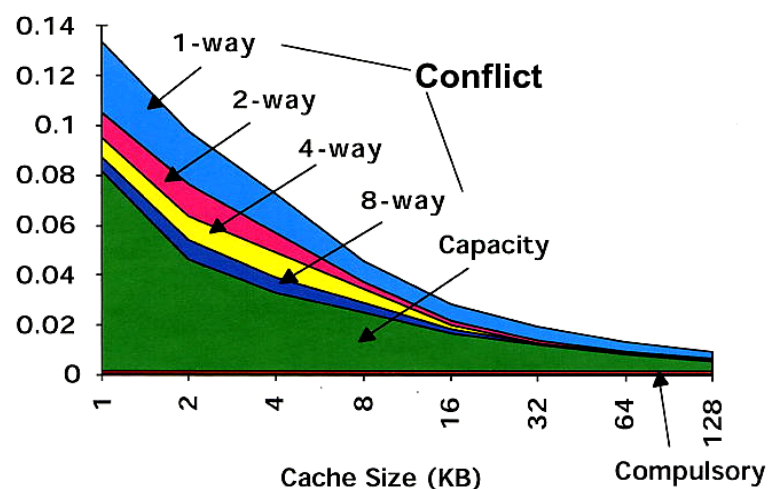
- Folgende Situationen können bei einem **Schreibzugriff** auf eine Cache-Zeile auftreten:
 - **Write Hit**: bei erfolgreichem Schreibzugriff auf Adresse a wird ein Datum aus typisch 1, 2, 4 oder 8 Byte im Cache aktualisiert und die **komplette Cache-Zeile** wird entweder
 - unmittelbar in den Hauptspeicher zurück geschrieben (**Write Through**)
 - oder durch ein zusätzliches Flag (*Dirty Bit*) markiert und erst später bei Verdrängung in den Hauptspeicher zurück geschrieben (**Write Back**)
 - **Write Miss**: bei erfolglosem Schreibzugriff wird entweder
 - der Eintrag zunächst aus dem Speicher geholt und dann wie bei einem *Write Hit* aktualisiert (**Fetch on Write**)
 - oder der Eintrag wird nur im Hauptspeicher ohne Modifikation des Caches aktualisiert (**Write Around**)

7.7 Verhalten von Caches

- Faustregeln:
 - ein **2-Wege teilassoziativer** Cache hat typischerweise eine *Miss Rate* wie ein **doppelt** großer **direkt abbildender** Cache!
 - ein **8-Wege teilassoziativer** Cache weist für die meisten Anwendungen ungefähr eine *Miss Rate* wie ein **vollassoziativer** Cache auf!
- mittlere Speicher-Zugriffszeit:
 - mit L1-Cache: $t_{\text{access}} = t_{\text{accessL1}} + \mu_{\text{L1}} \cdot t_{\text{accessMemory}}$
 - mit L1- und L2-Cache: $t_{\text{access}} = t_{\text{accessL1}} + \mu_{\text{L1}} \cdot (t_{\text{accessL2}} + \mu_{\text{L2}} \cdot t_{\text{accessMemory}})$
(wobei μ_{L1} und μ_{L2} die jeweiligen Cache-Fehlzugriffsraten darstellen)
- 3 Ursachen für Cache-Fehlzugriffen (3 „C“s):
 - bei **Erstbelegung** nach Programmstart (*Compulsory*)
 - wenn wegen zu geringer **Kapazität** Verdrängungen benötigter Zeilen auftreten (*Capacity*)
 - wenn benötigte Zeilen wegen **Konflikten** verdrängt werden (*Conflict*)

7.7 Verhalten von Caches (2)

- typische Fehlzugriffsraten in Abhängigkeit von Ursache, Cache-Größe und -Typ
(ermittelt für Spec92 Benchmark, Patterson 1998)



- Fehlzugriffsraten sinkt deutlich bei Vergrößerung des Caches!
- Fehlzugriffe durch Erstbelegung sind für viele Anwendungen vernachlässigbar!

7.7 Verhalten von Caches (3)

- **Fazit:** eine gut dimensionierte Cache-Hierarchie kann durch Ausnutzung
 - von **zeitlicher** und **räumlicher Lokalität** in Programmen
 - des **Burst-Modus** bei ausreichend hoher Eintragsgrößedie Nachteile von langsamen DRAM-Bausteinen weitgehend verdecken!

7.8 Cache-Optimierungen im Programm

- Einfügen von **Prefetch-Instruktionen**, d.h. benötigte Daten werden schon vorab in Cache geholt (automatisch vom Compiler oder manuell im Assemblerprogramm)
- **Erhöhung der Lokalität** beim Zugriff auf Daten:

Beispiel 1: *Merging*

```
/* vorher: */  
String[] Name;  
int[] PersonalNummer;  
  
/* nachher: */  
class Person {  
    String Name;  
    int PersonalNummer;  
}  
Person[] Personal;
```

Beispiel 2: *Loop Interchange*

```
/* vorher: */  
for (j=0;j<100;j=j+1)  
    for (i=0;i<5000;i=i+1)  
        x[i][j] = 2*x[i][j]  
  
/* nachher: */  
for (i=0;i<5000;i=i+1)  
    for (j=0;j<100;j=j+1)  
        x[i][j] = 2*x[i][j]
```

7.8 Cache-Optimierungen im Programm (2)

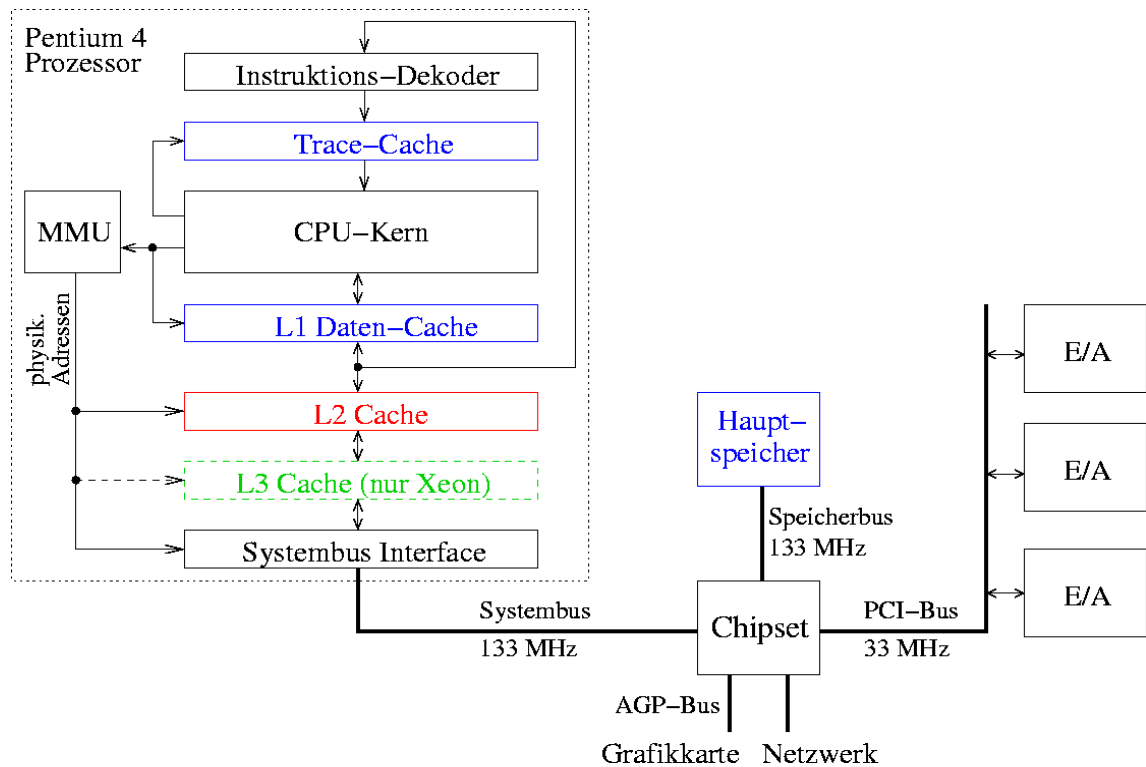
- **Vermeidung von Cache-Konflikten**

- die Wahrscheinlichkeit für Verdrängungen durch Cache-Konflikte kann z.B. steigen, wenn
 - die Größe einer Dimension eines mehrdimensionalen Feldes einer **Zweierpotenz** entspricht
 - die Elemente mehrerer Felder, deren Größe jeweils einer **Zweierpotenz** entspricht, verknüpft werden
- Lösung: Einfügen von **Füllworten** (*Padding* bzw. *Array Padding*), z.B.
 - durch Erhöhen der Feldgröße auf einen Wert, der keine Zweierpotenz darstellt
 - durch Einfügen zusätzlicher Variablen zwischen der Deklaration von Feldern
- einige Compiler für Höchstleistungscomputer können ein *Padding* auch automatisch durchführen

7.8 Implementierung von Caches

- Cache-Zugriff erfolgt i.a. mit internem **Pipelining**
 - je Takt kann ein neuer Zugriff gestartet werden
 - Dauer eines Zugriffs beträgt jedoch mehrere Takte
- Caches können **logisch/virtuell** oder **physikalisch** adressiert werden
 - L1-Caches werden zumeist logisch/virtuell adressiert
 - **Vorteil:** sehr schnell, da keine Abbildung der Adressen bei Zugriff erforderlich
 - **Nachteil:** bei Prozesswechsel muss L1-Cache gelöscht werden
 - L2- oder L3-Cache werden zumeist physikalisch adressiert
 - **Vorteil:** keine Veränderung bei Prozesswechsel
 - **Nachteil:** Adressen müssen vor Zugriff erst abgebildet werden

7.9 Fallstudie: Speicherarchitektur eines PC



7.9 Fallstudie: Speicherarchitektur eines PC (2)

- ein **Trace-Cache**
 - enthält nur die tatsächlich **ausgeführten** und bereits **dekodierten** Befehle (μ -Befehle) in der **zeitlichen Reihenfolge** des Auftretens im Programm
 - ersetzt hier einen internen L1 Code-Cache
- Cache-Hierarchie des **Pentium 4** Prozessors:

	Trace-Cache	L1 Daten	L2
Cache-Typ	?	4-Wege teilasso.	8-Wege teilasso.
Cache-Größe	12 K μ -Befehle	8 KByte	256 KByte
Eintragsgröße	6 μ -Befehle	64 Bytes	128 Bytes
Schreibstrategie	—	Write Through	Write Back
Zugriffszeit / Durchsatz	2 Takte / 3 μ -Befehle je Takt	2 Takte / 1 je Takt	7 Takte / 1 je zwei Takte

8 Lernziele

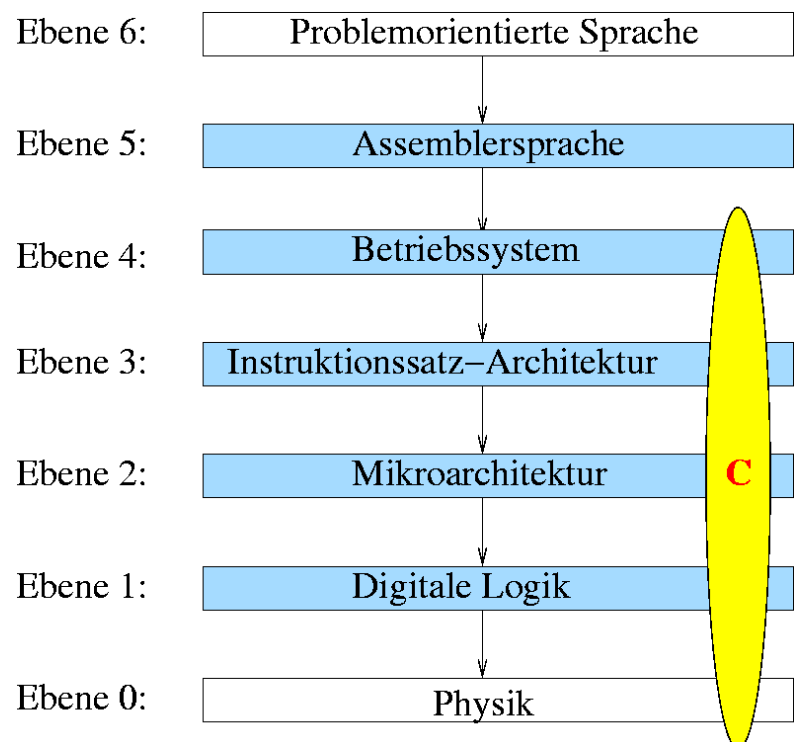
- **Begriffe:** SRAM, DRAM, SDRAM, DDR-SDRAM, RDRAM, EPROM, EEPROM, Flash-Speicher, ...
- Verständnis der Arbeitsweise einer **SRAM-Zelle**, einer **DRAM-Zelle** sowie einer **EPROM-Zelle**
- Möglichkeiten der Beschleunigung bei DRAMs
- **Organisation** von Speicherbausteinen; Aufbau eines Speichers aus vorgegebenen Bausteinen
- Unterschiede in **Aufbau**, **Adressierung** und **Arbeitsweise** eines vollassoziativen, eines direkt abbildenden sowie eines n -Wege teilassoziativen Cache
- **Verhalten** von Caches und Auswirkungen von Caches auf die Programmlaufzeit; Cache-Optimierungen im Programm

C Ein-/Ausgabekonzepte und Bussysteme

1. Ein-/Ausgabe
2. Strategien der Ein-/Ausgabe
3. Techniken der Datenübertragung
4. Punkt-zu-Punkt Schnittstellen
5. Direct Memory Access (DMA)
6. Bussysteme
7. Gerätetreiber

C Ein-/Ausgabekonzepte und Bussysteme

- Einordnung in das Schichtenmodell:



1 Ein-/Ausgabe

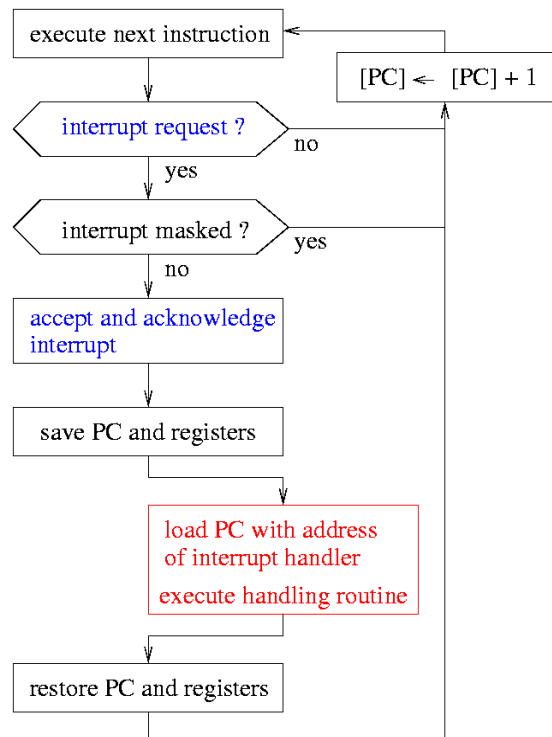
- zur Realisierung einer Ein-/Ausgabe werden benötigt:
 - eine **Strategie** der Ein-/Ausgabe
 - eine nach Möglichkeit **standardisierte Schnittstelle** (*Interface*) zur Ein-/Ausgabe, z.B. seriell (RS-232) oder parallel (EPP):
 - Definition eines *Übertragungsprotokoll*
 - Definition von Steckern/Buchsen und Kabeln
 - Hardwareunterstützung durch entsprechenden E/A-Baustein (*I/O Controller*), ggf. separate *Schnittstellenkarte*
 - **E/A-Geräte** bzw. **Peripheriegeräte**
 - zur Umwandlung von elektrischen Signalen in eine verwertbare physikalische Form ... (z.B. Bildschirm, Drucker)
 - ... und umgekehrt (z.B. Maus, Tastatur, Scanner)

2 Strategien der Ein-/Ausgabe

- **programmierte Ein-/Ausgabe:**
 - ein laufendes Programm (z.B. ein Anwendungsprogramm) legt explizit fest, *wann* eine Ein-/Ausgabe erfolgt
 - Falls der Ausgabebaustein noch nicht bereit ist, kann CPU in einer Warteschleife auf dessen Bereitschaft warten (*Busy Waiting*)
 - Eingabebausteine können zu definierten Zeitpunkten abgefragt werden, ob neue Eingabedaten anliegen (*Polling*)
- **Unterbrechungen** (*Interrupts*):
 - bei Eintreffen neuer Daten kann ein Eingabebaustein eine Unterbrechung anfordern (*Interrupt Request*)
 - Ausgabebaustein kann durch eine Unterbrechungsanforderung der CPU die erneute Bereitschaft signalisieren
 - sobald möglich, bestätigt CPU den Interrupt (*Interrupt Acknowledge*) und startet eine zugehörige Behandlungsroutine

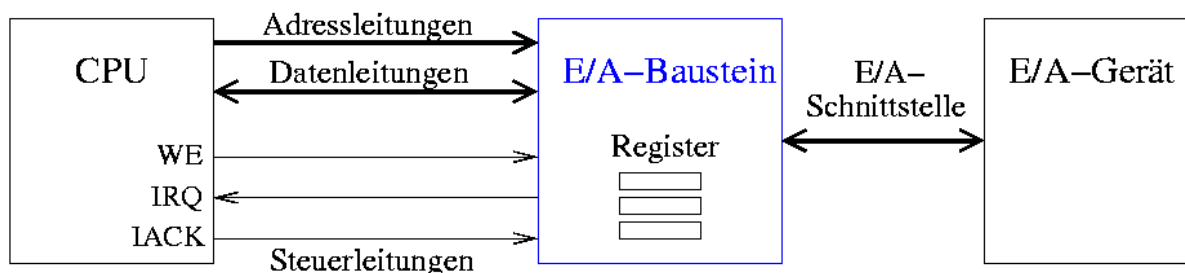
2 Strategien der Ein-/Ausgabe (2)

- Unterbrechungen erlauben eine schnelle Reaktion der CPU auf E/A-Ereignisse
 - allgemeiner **Ablauf** bei einer Unterbrechung:
 - Unterbrechung nur **nach** der Ausführung einer Instruktion möglich
 - fast alle Prozessoren bieten die Möglichkeit, Unterbrechungen zu verbieten (d.h. zu **maskieren**) bzw. wieder zu gestatten
 - verschiedene **Prioritätsstufen** bei mehreren E/A-Geräten sinnvoll



2 Strategien der Ein-/Ausgabe (3)

- Kopplung von CPU und E/A-Geräten erfolgt über spezielle **E/A-Bausteine** am Systembus:
 - Auswahl eines E/A-Bausteins über Adressleitungen
 - Datentransfer von/zu E/A-Baustein über Datenleitungen
 - Steuerleitungen z.B. für Richtungsauswahl, Unterbrechungsanforderung und Bestätigung einer Unterbrechung
 - zur Kommunikation mit CPU bietet E/A-Bausteine einige interne Register an



2 Strategien der Ein-/Ausgabe (4)

- ein E/A-Baustein verfügt über drei Arten interner Register:
 - **Kontrollregister**
 - zur Initialisierung und Parameterwahl durch CPU
 - **Datenregister**
 - zur Zwischenpufferung von einzulesenden oder auszugebenden Daten (nötig, da E/A-Geräte zumeist langsamer als CPU sind und zudem asynchron zur CPU arbeiten)
 - **Statusregister**
 - zum Austausch von Statusinformationen zwischen E/A-Baustein und CPU (z.B. Verfügbarkeit eines neuen Eingabewertes, oder Ausgabegerät hat Zeichen aus Datenregister gelesen)
 - E/A-Baustein setzt/löscht entsprechende Bits im Statusregister selbständig
 - bei Verwendung der Strategie *Polling* wird Statusregister in einer Schleife abgefragt

2 Strategien der Ein-/Ausgabe (5)

- zwei Möglichkeiten für den Zugriff der CPU auf die internen Register der E/A-Bausteine:
 - **speicherbezogene Adressierung** (*Memory-Mapped I/O*):
 - Register sind an bestimmte Speicheradressen in den physikalischen Adressraum der CPU eingeblendet
 - E/A-Adressen müssen vom Caching ausgenommen werden!
 - Zugriff mit normalen `load`- und `store`-Befehlen, Zugriffsschutz nur über Speicherverwaltung
 - **separate Ein-/Ausgabeadressen**
 - separater, oft kleiner E/A-Adressraum
 - spezielle, i.a. privilegierte Befehle (z.B. `in`, `out`) für Lesen und Schreiben im E/A-Adressraum
 - zusätzliche Steuerleitung $\overline{\text{IO/Memory}}$ zur Selektion von Speicher- oder E/A-Adressraum

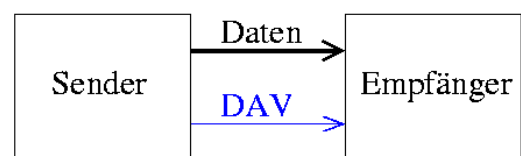
3 Techniken der Datenübertragung

- zwischen Sender (z.B. Rechner) und Empfänger (z.B. Peripheriegerät) existieren Leitungen für die Übertragung von **Daten** und **Steuersignalen**
- drei verschiedene Übertragungsprotokolle:
 - 1) **Open-Loop** Datenübertragung
 - keine Rückmeldung bei der Übertragung
 - 2) **Closed-Loop** Datenübertragung
 - Quittierung der Datenannahme über Steuersignal
 - auch als *Handshaking* bezeichnet
 - 3) **Fully-Interlocked** Datenübertragung
 - Quittierung sowohl der Datenannahme als auch aller Steuersignale
 - auch als *Fully-Interlocked Handshaking* bezeichnet

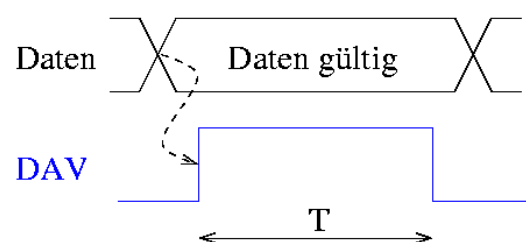
3.1 Open-Loop Datenübertragung

- eine Steuerleitung:
DAV = Data Valid
- Sender legt ein Datum auf Datenleitungen und setzt für eine Zeitdauer T das Signal **DAV** = 1
- Empfänger muss die Daten übernehmen, solange **DAV** = 1 ist
- Sender und Empfänger müssen Zeitdauer T vereinbart haben

Signale:



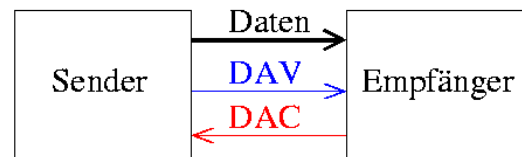
Zeitdiagramm:



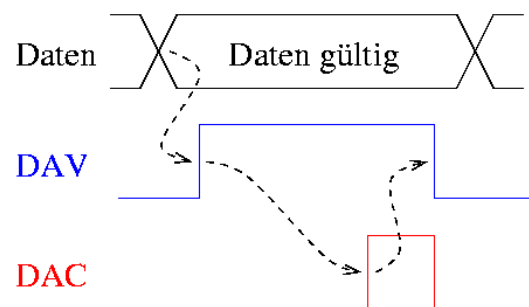
3.2 Closed-Loop Datenübertragung

- zwei Steuerleitungen:
DAV = *Data Valid*
DAC = *Data Accepted*
- nach Empfang eines Datums bestätigt Empfänger dies mit **DAC = 1**
- Sender nimmt Daten von Datenleitungen zurück, sobald er **DAC = 1** empfangen hat
- **einfache Flusskontrolle**
(langsamer Empfänger kann Sender anhalten)
- ggf. Abbruch nach Wartezeit (**Timeout**)

Signale:



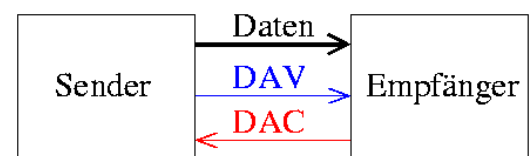
Zeitdiagramm:



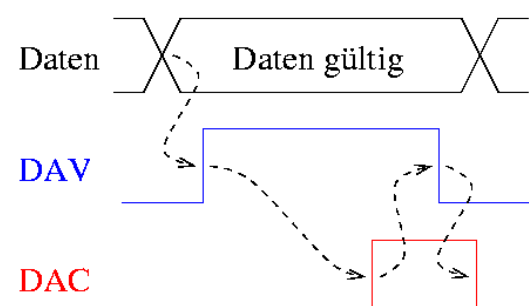
3.3 Fully-Interlocked Datenübertragung

- zwei Steuerleitungen, wie bei Closed-Loop
- Unterschiede zu Closed-Loop:
 - Empfänger hält **DAC = 1**, bis der Sender **DAV = 0** setzt
 - Empfänger bestätigt somit, die Deaktivierung des Signals DAV gesehen zu haben
 - Sender gibt erst neue Daten aus, wenn **DAC = 0** vorliegt
- **verbesserte Flußkontrolle**
(Empfänger kann auch nach Empfang von Daten den Sender aufhalten)

Signale:



Zeitdiagramm:



4 Punkt-zu-Punkt Schnittstellen

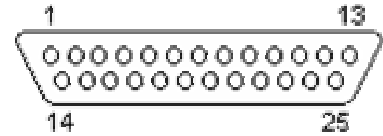
- Bei Verwendung einer **Punkt-zu-Punkt Schnittstelle** lässt sich nur ein einzelnes Peripheriegerät an eine Schnittstelle anschließen
- Gegenteil: **Bussysteme** (→ Abschnitt 6)
- Punkt-zu-Punkt Schnittstelle
 - **unidirektional**
(es gibt einen Sender und einen Empfänger)
 - **bidirektional**
(beide Seiten können als Sender oder Empfänger arbeiten)
- Beispiele für standardisierte Punkt-zu-Punkt Schnittstellen
 - 1) parallele Schnittstelle (IEEE 1284)
 - 2) serielle Schnittstelle (RS-232)

4.1 Parallele Schnittstelle

- **8-Bit** Datenworte werden **parallel** vom Sender zum Empfänger übertragen
- früher: *Centronics* Schnittstelle
 - nur unidirektionaler Betrieb: Ausgabe von Daten an ein Peripheriegerät
 - Transferrate bis zu 150 KByte/s
- heute: IEEE 1284 Schnittstelle, unterstützt u.a. 3 Modi:
 - *Enhanced Parallel Port* (**EPP**) gestattet einen bidirektionalen Betrieb: bei Anschluß eines Druckers kann dieser z.B. auch Statusmeldungen (kein Papier, Toner leer, ...) übertragen
 - *Extended Capability Port* (**ECP**) leistet zusätzlich Datenkomprimierung und Auswahl mehrerer logischer Geräte
 - *Standard Parallel Port* (**SPP**) definiert einen unidirektionalen Betrieb (kompatibel zur *Centronics* Schnittstelle)
 - mit EPP ist eine Transferrate bis zu ca. 2 MByte/s möglich

4.1 Parallele Schnittstelle (2)

- **25-polige** Steckerverbindung (Sub-D):
- Bedeutung der Steckerpins bei Einsatz im **SPP-Modus**:

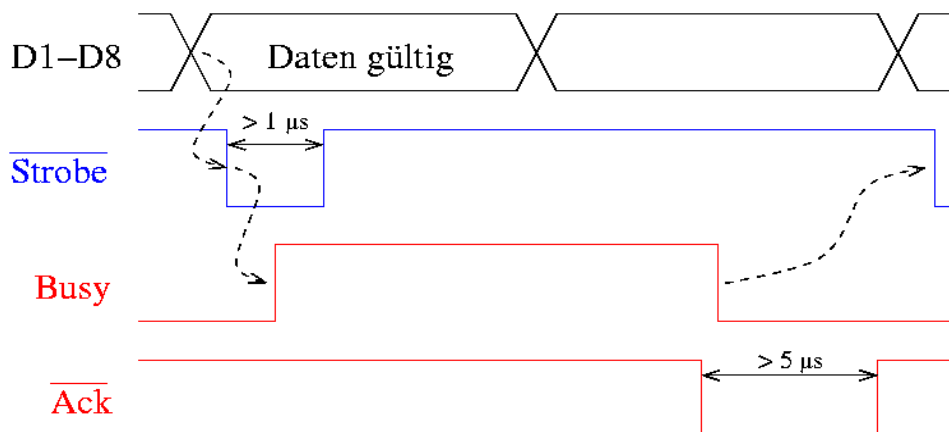


1 /Strobe	A	Daten gültig	10 /Ack	E	Bestätigung
2 D1 (LSB)	A	Datenbit 0	11 Busy	E	Endgerät beschäftigt
3 D2	A	Datenbit 1	12 PE	E	Papierende
4 D3	A	...	13 Select	E	Endgerät offline/online
5 D4	A		14 /Auto Feed	A	autom. Zeilenvorschub
6 D5	A		15 /Error	E	Fehler
7 D6	A		16 /Reset	A	Endgerät zurücksetzen
8 D7	A	Datenbit 6	17 /Select In	A	Auswahlleitung
9 D8 (MSB)	A	Datenbit 7	18-25 GND	–	Abschirmung

- einige Steuersignale (mit / markiert) mit sind „*low active*“
- heute z.T. auch 36-polige Steckerverbindung üblich

4.1 Parallele Schnittstelle (3)

- Zeitdiagramm für die Ausgabe eines Bytes zum Peripheriegerät:



- TTL-Pegel, d.h. +5V für logisch 1, 0V für logisch 0
- Closed-Loop Datenübertragung mit Steuersignalen **/Strobe** und **Busy**; das Signal **/ACK** liefert eine zusätzliche Bestätigung und kann zur Unterbrechungsanforderung verwendet werden

4.1 Parallele Schnittstelle (4)

- Bedeutung der Steckerpins bei Einsatz im **EPP-Modus**:

1 /Write	A	Richtung	10 /Intr	E	Interrupt
2 D1 (LSB)	E/A	Datenbit 0	11 /Wait	E	Endgerät bereit
3 D2	E/A	Datenbit 1	12 UserDef0	E	frei belegbar
4 D3	E/A	...	13 UserDef1	E	frei belegbar
5 D4	E/A		14 /DStrobe	A	Daten gültig
6 D5	E/A		15 UserDef2	E	frei belegbar
7 D6	E/A		16 /Reset	A	Endgerät zurücksetzen
8 D7	E/A	Datenbit 6	17 /AStrobe	A	Adressen gültig
9 D8 (MSB)	E/A	Datenbit 7	18-25 GND	–	Abschirmung

- Datenleitungen nun **bidirektional**, zusätzliche Steuerleitung zur **Richtungsangabe** (/Write = 0 : Ausgabe, /Write = 1: Eingabe)
- EPP stellt eine Multifunktionsschnittstelle dar
(⇒ Adressübertragung, benutzerdefinierbare statt druckerspezifische Signale)

4.1 Parallele Schnittstelle (5)

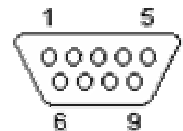
- Übertragungsprotokoll bei **SPP** in Software:
 - CPU muss Status des Empfängers (**Busy**) mittels Statusregister auswerten,
 - ein Byte in das Datenregister des E/A-Bausteins schreiben,
 - Strobe-Signal mittels bestimmter Bits im Kontrollregister aktivieren und ca. 1 µs später wieder deaktivieren, und
 - Bestätigung des Empfängers (**Ack**) im Statusregister abwarten
- Übertragungsprotokoll bei **EPP** in Hardware:
 - CPU schreibt/liest lediglich ein Byte in/aus Datenregister
- EPP** stellt heute eine einfache, preiswerte Schnittstelle für Geräte mit relativ hoher Datenübertragungsrate dar:
 - Drucker und Scanner
 - Zip-Laufwerk
 - ...
- wird jedoch von anderen Schnittstellen (z.B. **USB**) verdrängt

4.2 Serielle Schnittstelle

- serielle, **asynchrone** Punkt-zu-Punkt Verbindung
 - jedes zu übertragene Zeichen wird mit Start-, Stop- und ggf. Paritätsbit in ein Paket verpackt, das asynchron (ungetaktet) und bitseriell über eine Leitung übertragen wird
 - mit minimal 3 Leitungen bereits funktionsfähig (⇒ geringe Kabelkosten)
 - verbindet **DTE** (*Data Terminal Equipment*, z.B. PC) **mit DCE** (*Data Communication Equipment*, z.B. Modem),
 - kann auch **DTE mit DTE** verbinden
 - Sender und Empfänger müssen sich für die Übertragung eines jeden Pakets erneut synchronisieren
- Standard: **RS-232** (*Electronic Industries Association*, 1969)
 - auch als V.24 bezeichnet (CCITT)

4.2 Serielle Schnittstelle (2)

- **9-polige** Steckerverbindung (Sub-D):
- Pinbelegung aus Sicht eines DTE:
(mit E = Eingangsleitung, A = Ausgangsleitung)

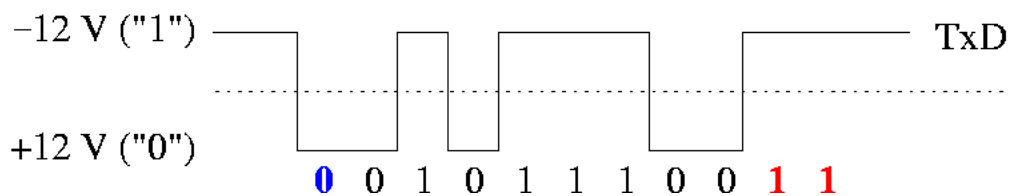


1	DCD	E	Trägererkennung (<i>Data Carrier Detect</i>)
2	RxD	E	Empfangsleitung (<i>Receive Data</i>)
3	TxD	A	Sendeleitung (<i>Transmit Data</i>)
4	DTR	A	DTE ist betriebsbereit (<i>Data Terminal Ready</i>)
5	GND	–	Masse, Abschirmung (<i>Ground</i>)
6	DSR	E	DCE ist betriebsbereit (<i>Data Set Ready</i>)
7	RTS	A	DTE ist bereit zum Datenaustausch (<i>Request to Send</i>)
8	CTS	E	DCE ist bereit zum Datenaustausch (<i>Clear to Send</i>)
9	RI	E	ankommender Anruf (<i>Ring Indicator</i>)

- auch **25-polige** Steckerverbindung (zusätzliche Steuersignale)

4.2 Serielle Schnittstelle (3)

- **Spannungspegel** auf den Leitungen:
 - Logisch 1: $< -3V$ (typisch $-12V$, maximal $-15V$)
 - Logisch 0: $> +3V$ (typisch $+12V$, maximal $+15V$)
 - hoher Störabstand $\Rightarrow$ maximale Leitungslänge: ca. 20m
- serielle Datenübertragung
 - Ruhezustand: $TxD = 1$
 - **1 Startbit** ($TxD = 0$)
 - 5,6,7 oder 8 Datenbits (niedrigstwertiges Datenbit zuerst)
 - **1** oder **2 Stopbits** ($TxD = 1$)
- Beispiel: serielle Übertragung des Byte $3A_{16} = 00111010_2$



4.2 Serielle Schnittstelle (4)

- optional zusätzliches **Paritätsbit** :
 - **gerade Parität** (*Even Parity*): Summe der 1-Datenbits ist gerade
 - **ungerade Parität** (*Odd Parity*): Summe der 1-Datenbits ist ungerade
- typische Parameterwahl:
 - 8 Datenbits, keine Parität (*None Parity*), 1 Stopbit $\Rightarrow$ Kurzform: 8N1
- Übertragungsgeschwindigkeit wird in **Baud** angegeben:
 - Baud-Rate = Anzahl Bit je Sekunde
 - typische Baud-Raten: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
 - bei einer Baudrate von 115200 dauert jedes Bit $8.6 \mu s$
- Auf Sende- und Empfangsseite sind Baud-Rate sowie Parameter (Parität, Anzahl Daten- und Stop-Bits) **identisch** einzustellen

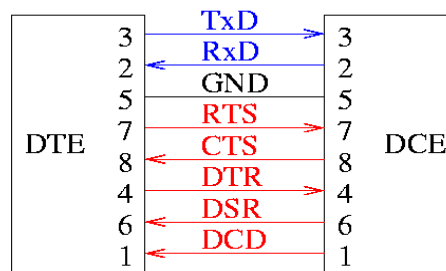
4.2 Serielle Schnittstelle (5)

- keine Empfangsbestätigung $\Rightarrow$ Open-Loop Datenübertragung
- Anpassung unterschiedlicher Geschwindigkeiten von DTE und DCE mittels **Flußkontrolle**:
 - entweder mittels **Hardware** (*RTS/CTS protocol*):
 - wenn **DCE** keine Daten mehr aufnehmen kann, erhält DTE auf der Eingangsleitung **CTS** den Pegel 0
 - wenn **DTE** keine Daten mehr aufnehmen kann, wechselt DTE den Pegel auf der Ausgangsleitung **RTS** von 1 auf 0
 - oder mittels **Software** (*Xon/Xoff protocol*):
 - Senden des Steuerzeichens *Xoff* (Ctrl-S, ASCII 19h) stoppt den Datentransport in die entgegengesetzte Richtung
 - Senden des Steuerzeichens *Xon* (Ctrl-Q, ASCII 17h) hebt den Stop wieder auf
 - beide Steuerzeichen dürfen nicht im Datenstrom vorkommen!

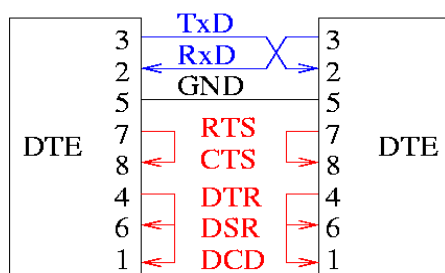
4.2 Serielle Schnittstelle (6)

- verschiedene Kabel zur Verbindung von Geräten mittels RS-232:

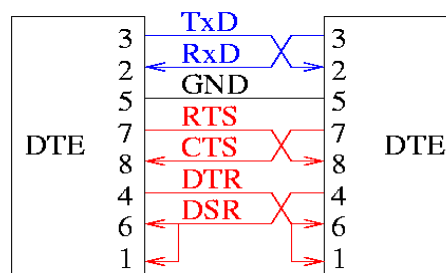
Modem-Kabel (*straight cable*):
(für RTS/CTS Flußkontrolle)



Nullmodem-Kabel:
(für Xon/Xoff Flußkontrolle)



Nullmodem-Kabel:
(für RTS/CTS Flußkontrolle)

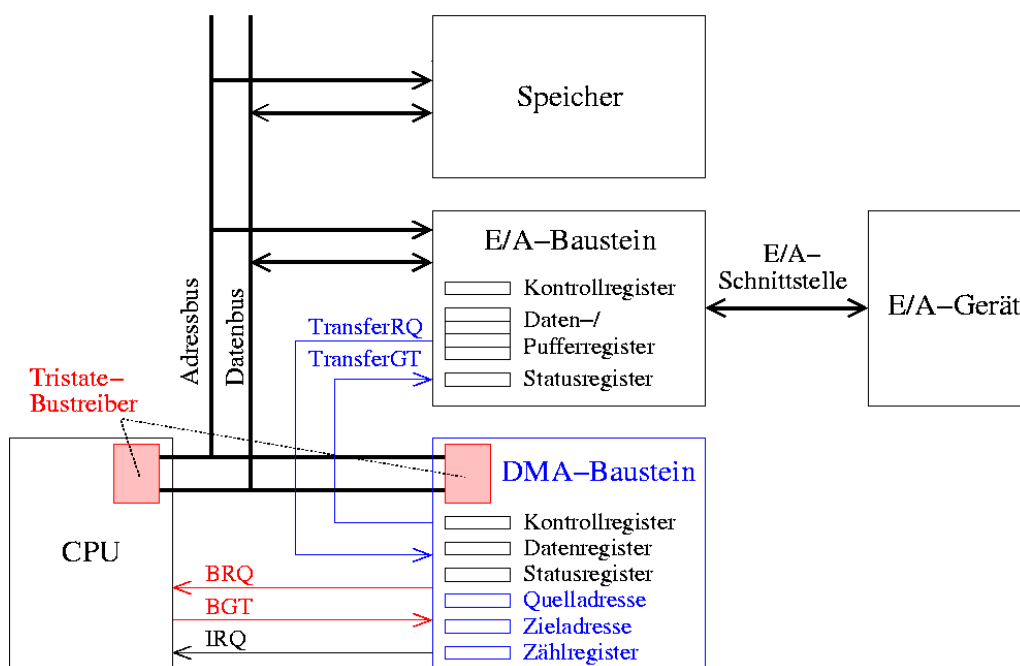


5 Direct Memory Access

- oft werden lange Datenströme aus dem Speicher zur Peripherie ausgegeben, bzw. von der Peripherie in den Speicher eingelesen
($\Rightarrow$ unnötige Belastung der CPU mit trivialen Aufgaben: Inkrementieren der Adresse, Zählen der Datenworte, Statusabfrage des E/A-Bausteins)
- Idee: ein zusätzlicher **DMA-Baustein** (*Direct Memory Access*) führt nach Initialisierung durch die CPU den Speichertransfer selbständig durch
($\Rightarrow$ CPU kann sich anspruchsvolleren Tätigkeiten widmen!)
- ein DMA-Baustein enthält
 - ein **Quelladressregister** und ein **Zieladressregister**, in dem die Start- und Zieladresse des zu transferierenden Datenblocks eingetragen werden
 - ein **Zählregister**, das mit der Anzahl der zu transferierenden Bytes bzw. Datenworten initialisiert werden muss
 - ein **Kontrollregister**, um z.B. Richtung oder Arbeitsmodus festzulegen

5 Direct Memory Access (2)

- prinzipieller Aufbau eines Systems mit DMA-Bausteins:



5 Direct Memory Access (3)

- prinzipieller Ablauf eines DMA-Transfers vom E/A-Baustein in den Speicher:
 - 1) CPU initialisiert E/A-Baustein (Kontrollregister) und DMA-Baustein (mit Startadresse, Zieladresse und Anzahl an Datenworten)
 - 2) E/A-Baustein setzt das Signal **TransferRQ** (*Transfer Request*), sobald Daten vorhanden sind
 - 3) DMA-Baustein fordert mit dem Signal **BRQ** (*Bus Request*) den Bus von der CPU an
 - 4) CPU deaktiviert eigene Bustreiber und setzt Signal **BGT** (*Bus Grant*)
 - 5) DMA-Baustein zeigt dem E/A-Baustein den Beginn des Datentransfers durch das Signal **TransferGT** (*Transfer Grant*) an
 - 6) DMA-Baustein transferiert Daten vom E/A-Baustein in den Speicher
 - 7) DMA-Baustein gibt durch Rücknahme von **BRQ** den Bus wieder frei
 - 8) DMA-Baustein kann der CPU durch Interrupt das Ende des Transfers signalisieren

5 Direct Memory Access (4)

- Arbeitsmodi eines DMA-Bausteins:
 - **Burst Modus**: DMA-Baustein erhält den Systembus für den kompletten Transfer eines Blocks aus vielen Datenworten
 - E/A-Baustein benötigt internen Pufferspeicher
 - **Vorteil**: sehr hohe Transferrate
 - **Nachteil**: CPU wird für lange Zeit am Speicher-/Buszugriff gehindert
 - **Cycle Stealing**: DMA-Buszyklen und CPU-Buszyklen werden gemischt
 - ein fester Anteil an Buszyklen (z.B. jeder zweite Buszyklus) wird vom DMA-Baustein der CPU „gestohlen“
 - DMA kann zusätzlich alle von der CPU nicht benötigten Buszyklen nutzen (z.B. während der Ausführungsphase von Register/Register-Befehlen)
 - **Vorteil**: CPU wird nur geringfügig behindert
 - **Nachteil**: geringere Transferrate

5 Direct Memory Access (5)

- Anmerkungen:
 - im Kontrollregister des DMA-Bausteins kann die **Übertragungsart** gewählt werden:
 - beim **direkten** DMA-Transfer erfolgt die Datenübertragung direkt zwischen E/A-Baustein und Speicher ($\Rightarrow$ 1 Buszyklus je Wort)
 - beim **indirekten** DMA-Transfer wird jedes Wort im Datenregister des DMA-Bausteins zwischengespeichert ($\Rightarrow$ 2 Buszyklen je Wort)
 - Statusregister des DMA-Bausteins enthält Informationen über Zustand (frei/belegt), Blockende und Fehler
 - auch der Datentransfer zwischen zwei E/A-Bausteinen oder von Speicher zu Speicher kann über DMA realisiert werden
 - oft mehrere **DMA-Kanäle** je DMA-Baustein ($\Rightarrow$ mehrere Transfers gleichzeitig in Bearbeitung)
 - DMA-Baustein ist bei heutigen PCs im **Chipset** integriert

6 Bussysteme

- Verbindung von **mehreren** Komponenten über identische Transport- und Steuerleitungen
(Vorteil: bedeutend kleinerer Verdrahtungsaufwand als bei Punkt-zu-Punkt Verbindungen)
- Bussysteme waren früher hersteller- und systemspezifisch, sind heute jedoch weitgehend **standardisiert**
 - E/A-Geräte bzw. E/A-Karten können unabhängig von Hersteller und Rechnersystem entwickelt werden:
 - größerer Absatzmarkt, geringere Kosten
 - Beispiele: Steckkarten für PCI-Bus, Festplatten
 - Standardisierung umfasst
 - Signale und Spannungspegel
 - zeitliches und elektrisches Verhalten der Bussignale
 - Steckverbinder und Pinbelegungen

6 Bussysteme (2)

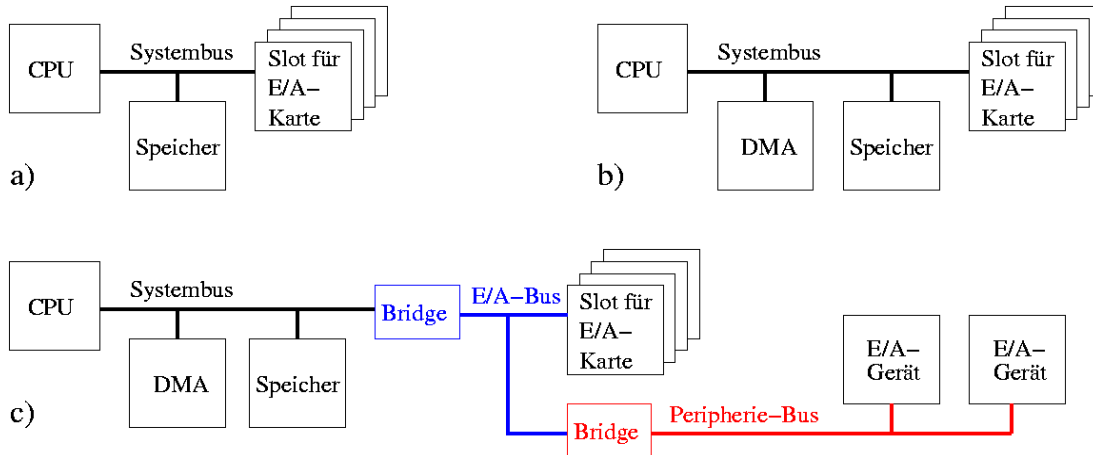
- verschiedene Arten von Bussystemen in einem Rechner:
 - **prozessorinterne Busse**
 - arbeiten i.a. mit CPU-Taktfrequenz
 - verbinden z.B. Registersatz, Arithmetik-Einheiten und L1-Datencache
 - **Systembus**
 - verbindet CPU mit schnellen Systemkomponenten
 - auch als *Front-Side-Bus* bezeichnet
 - Taktfrequenz typisch 100 bis 200 MHz
 - **Ein-Ausgabebus** (intern, auch *Local Bus*)
 - Bus für E/A-Erweiterungen der Hauptplatine, z.B. PCI-Bus
 - **Peripherie-Bus** (extern)
 - zum Anschluss mehrerer Peripheriegeräte an eine Busschnittstelle, z.B. USB, SCSI-Bus

6 Bussysteme (3)

- unterscheidende Merkmale von Bussystemen:
 - **Breite** von **Datenbus** und **Adressbus**
 - **Multiplexing** von Adress- und Datenleitungen
 - maximale **Datentransferrate** (in MByte/s)
 - **Taktung** (synchron/asynchron) und ggf. **Bustaktfrequenz**
 - **maximale Anzahl** von möglichen Buskomponenten bzw. Bussteckplätzen (*Slots*)
 - Art der **Busarbitrierung**
 - Art der möglichen **Buszyklen**
 - Realisierung von **Interrupts**
 - Unterstützung von **DMA**
 - **physikalische Größen**: Spannungspegel, Steckverbinder

6 Bussysteme (4)

- zwei verschiedene Bussysteme können mittels einer **Bridge** gekoppelt werden:
 - Anpassung z.B. von Datenbusbreite (z.B. 32 → 16 Bit), Taktfrequenz, Buszyklen
- einige mögliche **Systembus-Architekturen**:



6 Bussysteme (5)

- Bus besteht aus Datenleitungen (**Datenbus**), Adressleitungen (**Adressbus**), Steuerleitungen (**Steuerbus**) sowie weiteren Leitungen zur Stromversorgung
- oft **Multiplexing** von Daten und Adressen zur Einsparung von Busleitungen
- ein **Master** ist eine aktive Buskomponente, die einen Buszyklus auslösen kann (z.B. CPU, DMA-Baustein)
- ein **Slave** ist stets passiv (z.B. Speichermodul, E/A-Baustein)
- Busleitungen sind
 - zumeist *Tristate*-Leitungen, die je Buszyklus von einem ausgewählten Master getrieben werden
 - ggf. auch *Open-Collector*-Leitungen zur Realisierung einer globalen Oder-Verknüpfung (*Wired-OR*), z.B. für Interrupt- oder Busanforderung
 - i.a. „*low active*“ (aktiv, wenn das Bussignal = 0 ist)

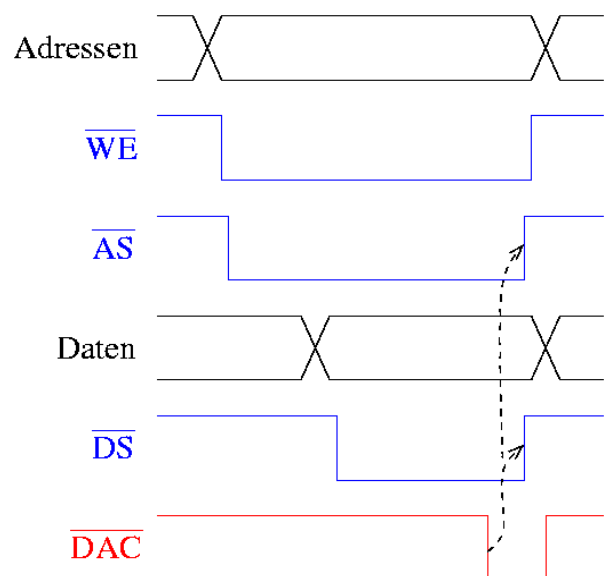
6.1 Busprotokolle

- standardisiertes **Busprotokoll** legt Zeitverhalten für verschiedene Arten von Buszyklen fest, z.B.:
 - Einzelwort-Transfer (Adresse, 1 Datenwort)
 - Block- oder Burst-Transfer (Wortanzahl n , Startadresse, n Datenworte)
- Busprotokoll auf einem **asynchronen Bus**:
 - Master adressiert den Slave und wählt die Richtung (Lesen/Schreiben)
 - Synchronisation über Handshaking z.B. mittels der Signale **/AS** (*Address Strobe*), **/DS** (*Data Strobe*) und **/DAC** (*Data Acknowledge*)
 - Ausbleiben der Bestätigung **/DAC** wird durch einen Timer überwacht; ggf. wird ein *Bus Error* ausgelöst
- Busprotokoll auf einem **synchronen Bus**:
 - Synchronisation erfolgt über ein gemeinsames Bustaktsignal (**BClk**)
 - es gibt feste Zeitpunkte für die Gültigkeit von Adressen und Daten und für die Übernahme von Daten

6.1 Busprotokolle (2)

- Beispiel 1: Schreiben eines Datenworts auf **asynchronem** Bus

- 1) Master gibt Adresse aus und setzt Signal **/WE** (*Write Enable*)
- 2) Master zeigt Gültigkeit der Adresse durch Signal **/AS**
- 3) Master gibt Datenwort aus
- 4) Master zeigt Gültigkeit des Datenworts durch **/DS** an
- 5) Slave quittiert Datenübernahme durch Signal **/DAC**
- 6) Master nimmt **/AS** und **/DS** zurück
- 7) Slave nimmt **/DAC** zurück

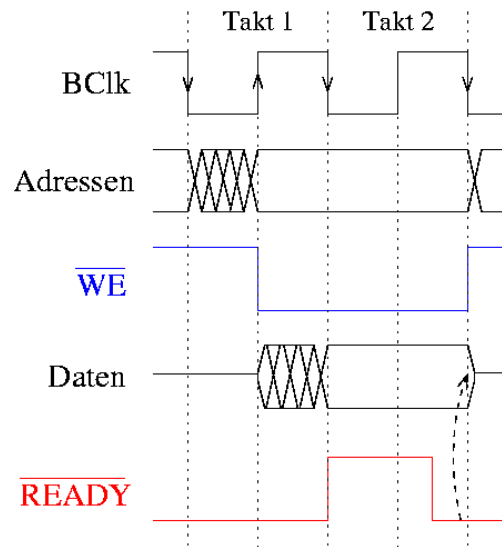


- Slave bestimmt die Länge eines Buszyklus durch Signal **/DAC** !

6.1 Busprotokolle (3)

- Beispiel 2: Schreiben eines Datenworts auf **synchronem** Bus

- 1) Master beginnt bei ↓ einen neuen Buszyklus und gibt eine Adresse aus
- 2) bei folgender Flanke ↑ ist Adresse gültig; gleichzeitig setzt Master das Signal **/WE** und gibt Datenwort aus
- 3) bei folgender Flanke ↓ sind Daten gültig; Slave setzt **/READY=1**
- 4) ist bei ↓ am Ende des zweiten Taktes wieder **/READY=0**, so hat Slave die Daten übernommen
- 5) Master beginnt einen neuen Buszyklus



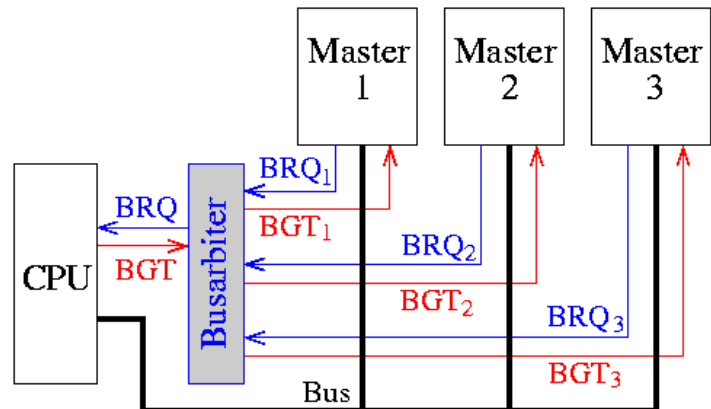
- Slave kann über Signal **/READY=1** den Buszyklus um einen oder mehrere Bustakte (*Wait Cycles*) verlängern!

6.2 Busarbitrierung

- können **mehrere** Buskomponenten **Master** werden, so ist eine **Busarbitrierung** erforderlich:
 - wenn ein Master den Bus benötigt, setzt er Signal **/BRQ** (*Bus Request*)
 - Zuteilung des Busses durch Signal **/BGT** (*Bus Grant*) an Master
 - nach Beendigung des Buszyklus deaktiviert Master das Signal **/BRQ**
 - **/BGT** wird zurückgesetzt
- eine einfache Busarbitrierung für *einen* zusätzlichen Master für den Systembus ist i.a. im Mikroprozessor integriert
- bei mehreren Mastern ist eine weitere externe Busarbitrierung erforderlich; zwei Varianten:
 - **zentrale** Busarbitrierung (zentrale Logik, implementiert in einem zusätzlichen Bussteuerbaustein)
 - **dezentrale** Busarbitrierung (Logik ist über alle Buskomponenten verteilt)

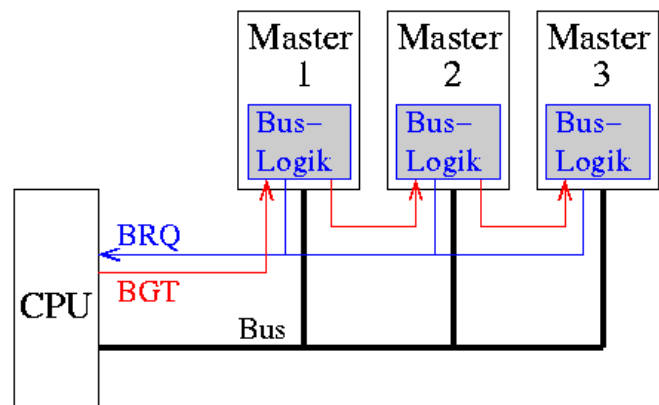
6.2 Busarbitrierung (2)

- Realisierung einer **zentralen** Busarbitrierung:
 - separate Anforderungsleitungen **/BRQ_i** und Zuteilungsleitungen **/BGT_i** für jeden Master *i*
 - beliebige Algorithmen zur Busarbitrierung in Hardware implementierbar
 - unterschiedliche Prioritäten können ggf. berücksichtigt werden
 - schnelle** Buszuteilung
 - Beispiel: PCI-Bus



6.2 Busarbitrierung (3)

- Realisierung einer **dezentralen** Busarbitrierung:
 - eine gemeinsame **/BRQ**-Leitung, realisiert in *Wired-OR* Technik
 - Signal **/BGT** wird in einer Kette (*Daisy Chain*) weitergegeben: Master *i* gibt Signal **/BGT** an Master *i* + 1 weiter, wenn er Bus nicht angefordert hat
 - Master mit kleinstem Index in der Kette hat höchste Priorität
 - unfaire** Busvergabe
(⇒ Aushungerungsproblem!)
 - langsame** Buszuteilung

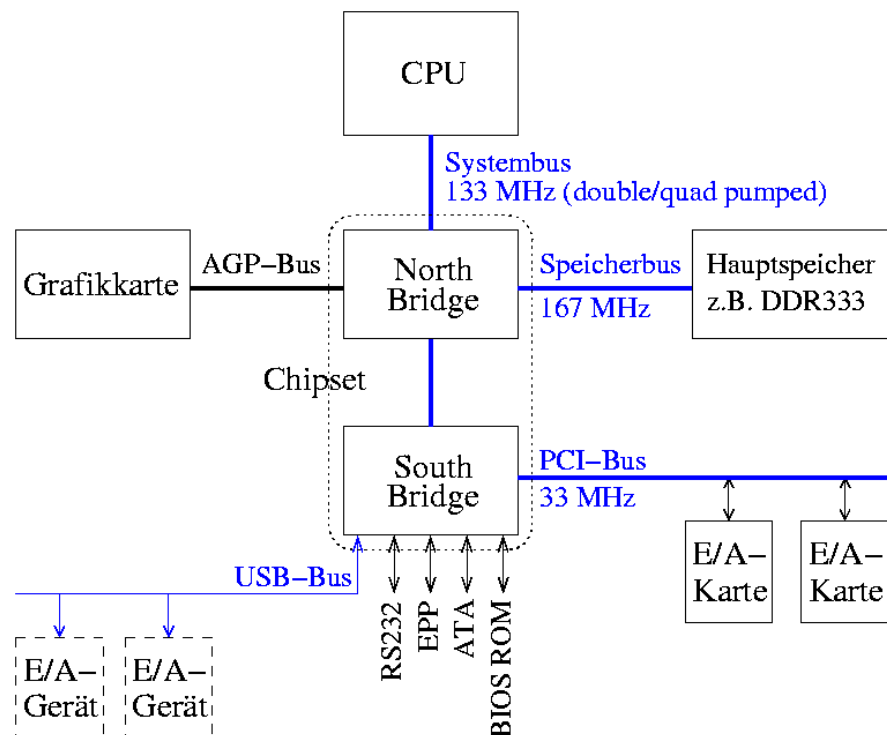


6.4 Fallstudie: PC-Bussysteme

- Bussysteme im PC:
 - Hauptplatine enthält x86 CPU und Chipset, gekoppelt über **Systembus** (64 Bit Daten, 32 Bit Adressen, synchron, typisch 100 bis 200 MHz Taktfrequenz)
 - Datenübertragungsrate höher, da je Taktzyklus oft mehrere Datenworte übertragen werden („*double pumped*“ oder „*quad pumped*“)
 - Chipset steuert **Speicherbus** (i.a mit einer vom Systembus abweichenden Taktfrequenz)
 - Chipset enthält serielle und parallele Schnittstelle, DMA-Baustein, Plattenkontroller, Bridge für PCI-Bus, Bridge für USB, ...
 - Chipset kann mehrere CPU-Zugriffe puffern und ggf. zusammenfassen
 - schneller **AGP-Bus** (*Accelerated Graphics Port*) für Grafikkarte
 - **PCI-Bus** mit mehreren PCI-Steckplätze (*PCI-Slots*) für Soundkarte, 10/100 MBit-Netzwerkkarte, ...

6.4 Fallstudie: PC-Bussysteme (2)

- Bus-Architektur eines PC:
(Stand 2003)



6.4 Fallstudie: PC-Bussysteme (3)

- früher: **ISA-Bus** (*Industry Standard Architecture*, 1984)
 - zuerst 8-Bit Daten, später **16-Bit Daten** und **24 Bit Adressen**
 - synchroner/asynchroner Bus mit **8 MHz** Taktfrequenz, max. 8 MByte/s
 - konzipiert als prozessornaher Systembus für 286-basierte AT PCs, Bussignale überwiegend identisch zum Prozessorbus
 - ISA-Steckverbinder:
(62+36 Pins) 
 - E/A-Adressen und Interrupts auf ISA-Buskarte über Jumper einzustellen
 - bis vor kurzem als weiterer Ein-/Ausgabebus für langsame, preiswerte E/A-Karten in PCs eingesetzt
 - die Übertragungsrate war Ende der 80er Jahre für PCs nicht mehr ausreichend; viele Alternativen wurden entwickelt:
 - **MCA** (*Microchannel Architecture*, IBM, 1987): 32-Bit Daten, 10 MHz
 - **EISA** (*Enhanced ISA*, 1989): 32-Bit Daten und Adressen, 8 MHz
 - **VLB** (*VESA Local Bus*, 1992): 32-Bit Daten, 40 MHz

6.4 Fallstudie: PC-Bussysteme (4)

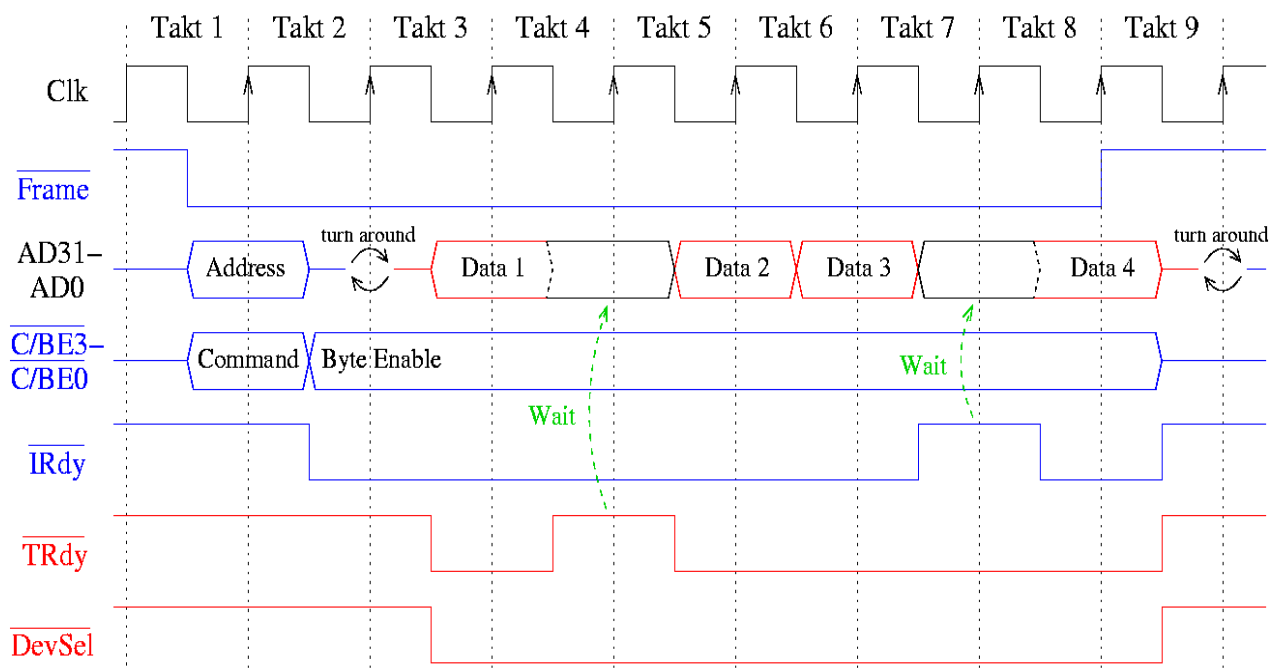
- heute: **PCI-Bus** (*Peripheral Component Interconnect*, 1993)
 - synchroner Bus, von Intel entwickelt
 - 12 Arten von Buszyklen, u.a. auch Einzelwort- und Burst-Transfer mit beliebiger Blocklänge
 - PCI 1.0 (heute typisch): **32-Bit Daten**, bis zu **33 MHz** Bustaktfrequenz
 - PCI 2.1: auch **64-Bit Daten**, bis zu **66 MHz** Bustaktfrequenz möglich
 - theor. max. Übertragungsraten von **133 MByte/s** (32-Bit Bus, 33 MHz) bis zu **533 MByte/s** (64-Bit Bus, 66 MHz) bei Burst-Transfer
 - **Multiplexing** von Daten und **32-Bit Adressen**
 - bis zu **4 masterfähige Slots** mit zentraler Bus-Arbitrierung
 - PCI-Steckverbinder:
(124 Pins bei 32-Bit Daten) 
 - prozessorunabhängiger Bus (nicht als Systembus einsetzbar!), auch in anderen Architekturen (z.B. Ultra-Sparc, PowerPC, ...) verbreitet
 - Weiterentwicklung: **PCI-X** mit bis zu 133 MHz Taktfrequenz

6.4 Fallstudie: PC-Bussysteme (5)

- **PCI-Busleitungen** (Auswahl) für **Master** und **Slave**:
 - **Clk**: Bustakt, auf dem alle Signale synchronisiert sind
 - **AD31 bis AD0**: gemultiplexer 32-Bit Adress-/Datenbus
 - **C/BE3 bis C/BE0** (*Command / Byte Enable*): enthält entweder ein Bus-Kommando (zur Auswahl einer Buszyklusart) oder eine Byteauswahl (aus 32-Bit Datenwort)
 - **/REQ_i**: Busanforderung von Karte in Slot i
 - **/GNT_i**: Buszuteilung an Karte in Slot i
 - **/INTA bis /INTD**: vier Unterbrechungs-Leitungen
 - **/Frame**: signalisiert Beginn und Ende eines Buszyklus
 - **/IRdy** (*Initiator Ready*): Master ist bereit zum Datentransfer
 - **/TRdy** (*Target Ready*): Slave ist bereit zum Datentransfer
 - **/DevSel**: Target bestätigt die Dekodierung seiner Adresse
 - **/IdSel_i**: Auswahl von Karte in Slot i zur Konfiguration

6.4 Fallstudie: PC-Bussysteme (6)

- Beispiel eines PCI-Buszyklus (Lesen eines Blocks aus 4 Worten):



6.4 Fallstudie: PC-Bussysteme (7)

- **bidirektionale Flußkontrolle:**
 - sowohl **Master** (*Initiator*) als auch **Slave** (*Target*) können den Transfer eines Datenworts durch Aktivierung von **/IRDY** bzw. **/TRDY** um einen oder mehrere Bustakte verzögern
- max. **Transferrate:** ein 32-Bit Wort je Bustakt
 - kann nur bei einem sehr langen Burst-Transfer erreicht werden
- **Richtungsumschaltung** der Bustreiber (*turn around*)
 - für gemultiplexte Adress-/Datenleitungen
 - benötigt einen zusätzlichen Bustakt
 - nur beim Lesen erforderlich ($\Rightarrow$ langsamer als Schreiben!)
- über PCI-PCI-Bridges **hierarchisch** erweiterbares Bus-System mit maximal 255 PCI-Bussen

6.4 Fallstudie: PC-Bussysteme (8)

- **3 Adressräume:**
 - 32 Bit **Speicher-Adressraum** (max. 4GByte)
 - 32 Bit **E/A-Adressraum** (I/O-Ports)
 - **Konfigurations-Adressraum** (256 Byte je PCI-Karte)
 - Buskommando legt für jeden Buszyklus den Adressraum fest, z.B.:

0010 <i>I/O Read</i>	0110 <i>Memory Read</i>	1010 <i>Configuration Read</i>
0011 <i>I/O Write</i>	0111 <i>Memory Write</i>	1011 <i>Configuration Write</i>
 - bei Zugriff auf den Speicher-Adressraum sind **Burst-Transfers** möglich, z.B. mit den Buskommandos

1110 <i>Memory Read Line</i>	(Lesen einer kompletten Cachezeile)
1100 <i>Memory Read Multiple</i>	(Lesen mehrerer Cachezeilen in einem Buszyklus)

6.4 Fallstudie: PC-Bussysteme (9)

- **Plug & Play:**
 - beim Booten liest BIOS den 256-Byte **Konfigurationsblock** jeder PCI-Karte mit:
 - Code für **Hersteller** und **Gerätenummer**
 - Code für **Gerätetyp**
 - **Zeitparameter**
 - BIOS konfiguriert automatisch
 - **Startadressen** für E/A-Register (im Speicher- oder E/A-Raum) und Erweiterungs-ROM
 - **Interrupt-Leitungen**
 - BIOS löst ggf. alle Konflikte auf
 - keine manuelle Konfiguration der PCI-Karten erforderlich!

31	16	16	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Latency Timer	Cache Line Size	0Ch
Base Address Registers				10h
				14h
				18h
				1Ch
				20h
				24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM Base Address				30h
Reserved			Capabilities Pointer	34h
Reserved				38h
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	3Ch

6.5 Fallstudie: USB

- **Universal Serial Bus**, spezifiziert vom USB Implementers Forum (www.usb.org)
- Ziel: preiswerter, einheitlicher und einfacher Anschluß diverser E/A-Geräte
- **serieller, asynchroner Peripherie-Bus**
 - 4-adriges Kabel: V_{cc} (Stromversorgung 5V, max. 0.5A), GND, D+, D- (Pegel 3.3V)
 - unterschiedliche Stecker für Host (USB-A) und E/A-Gerät (USB-B)
- **USB1.1** (1995)
 - Transferraten: 1.5 MBit/s (*low speed*) oder 12 MBit/s (*full speed*)
- **USB2.0** (2001)
 - weitere Transferrate: 480 MBit/s (*high speed*)



USB-A

USB-B

Beschriftung:



6.5 Fallstudie: USB (2)

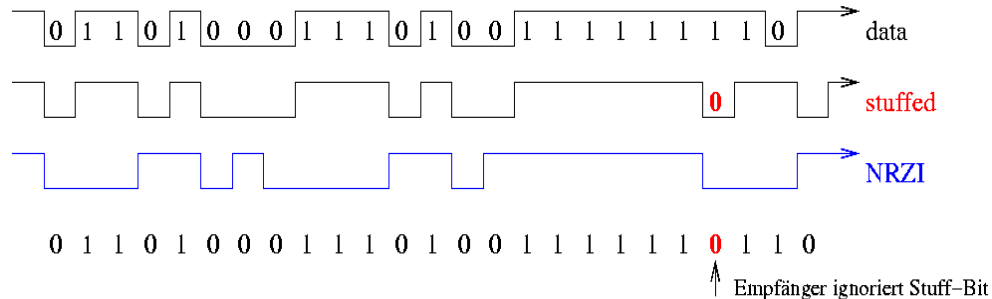
- **hierarchischer Aufbau** eines USB-Bussystems:
 - ausschließlich Punkt-zu-Punkt Verbindungen
 - **USB Host** (auch **Root Hub**, mit 2 bis 4 USB Anschlüssen) ist einziger Master, fragt alle USB Geräte durch *Polling* ab
 - **USB Hub** (Verstärker, ggf. mit Anpassung der Transferrate) verteilt Signale auf mehrere USB-Anschlüsse und ermöglicht den Aufbau eines pyramidenartigen Bussystems
 - maximal **7 physikalische Ebenen** in Pyramide, logisch jedoch eine Ebene
 - insgesamt maximal **127 Buskomponenten**
 - Länge eines Kabels max. **5m** $\Rightarrow$ insgesamt max. **35m** bei 7 Ebenen (USB-A Stecker stets zum Host, USB-B Stecker stets zum E/A-Gerät gerichtet)
 - **Autokonfiguration**: E/A-Geräte identifizieren sich selbst beim Host und erhalten eine Adresse zwischen 1 und 127 (Host hat Adresse 0)
 - Geräteanschluss im laufendem Betrieb möglich (*Hot Plugging*)
- Kopplung zweier USB Hosts ist nicht möglich!

6.5 Fallstudie: USB (3)

- Busprotokoll gestattet vier verschiedene Übertragungsarten:
 - 1) **Kontroll-Transfer**: Initialisierung und Konfiguration eines Gerätes durch USB Host
 - 2) **Interrupt-Transfer**: USB Host fragt alle E/A-Geräte ab, ob Interrupts angefordert wurden
 - 3) **Bulk-Transfer**: Senden langer Datenströme (nur bei *full / high speed*, falls ausreichende Bandbreite verfügbar)
 - 4) **Isochroner Transfer**: Übertragung von Daten mit einer garantierten Bandbreite (d.h. in Echtzeit), z.B. für Sprach- oder Videodaten (nur bei *full / high speed*)
- jeder Transfer wird vom USB Host initiiert!
- periodische Transfers (Interrupt-/Isochroner Transfer) dürfen nicht mehr als 80-90% der Busbandbreite verwenden!
- Hin- und Rückrichtung über die gleichen Leitungen!

6.5 Fallstudie: USB (4)

- **NRZI-Kodierung** (*Non Return to Zero Inverted*):
 - Wechsel des Leitungspegels **nur bei Übertragung eines Null-Bit**
 - **Bit-Stuffing**: Einfügen eines Null-Bits nach jeweils 6 Eins-Bits
 - Beispiel: NRZI-Kodierung der drei Bytes 68_{16} , $E9_{16}$ und FE_{16}



- **Differentielle Signale**
 - zur Übertragung des NRZI-Signals über verdrehtes Kabelpaar D+, D-:
 - Sender: $(D+) - (D-) > 1V$ (Eins-Bit) bzw. $< -1V$ (Null-Bit)
 - Empfänger: $(D+) - (D-) > 0.2V$ (Eins-Bit) bzw. $< -0.2V$ (Null-Bit)

6.5 Fallstudie: USB (5)

- **paketorientierte Übertragung**:
 - Einteilung in Zeitabschnitte von **1 ms** Dauer, auch als *Frame* bezeichnet ($\Rightarrow 12000$ Bit/Frame im *High Speed* Modus)
 - Adressierung der Endgeräte durch **7 Adress-Bits** (für 127 Geräte) und **4 EP-Bits** (für 16 verschiedene Endpunkte je Gerät, z.B. EP0 = *Control*, EP1 = *Bulk*, EP3 = *Interrupt*)
 - Kommunikation zwischen Host und Endpunkten von E/A-Geräten über logische Kanäle (*Pipes*), die einen Teil der Busbandbreite belegen
 - jede Datenübertragung innerhalb eines Frames besteht aus **drei Paketen**, wobei jedes Paket mit einer 8-Bit Typkennung beginnt:
 - 1) **Paket** mit **Richtung** und **Zieladresse** (11 Bit + 5 Bit Prüfsumme)
 - 2) **Datenpaket** (variable Länge + 16 Bit Prüfsumme)
 - 3) **Bestätigungspaket** (Handshaking) des Empfängers



6.5 Fallstudie: USB (6)

- Vergleich von USB1.1 und **USB 2.0**:

	<i>low speed</i>	<i>full speed</i>	<i>high speed</i>
Bit-Transferate	1.5 MBit/s	12 MBit/s	480 MBit/s
max. Bulk-Datenpaketgröße	—	64 Byte	512 Byte
max. Transferrate	16 KByte/s	1.2 MByte/s	54 MByte/s

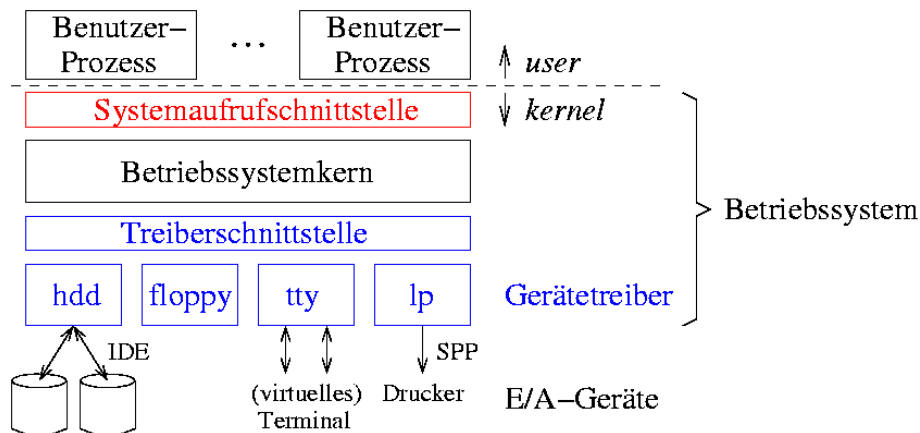
- USB stellt heute die Standardschnittstelle für weit über 50% aller (mittelschnellen) Peripheriegeräte dar
- **Firewire** (IEEE1394) ist sehr ähnlich zu USB:
 - entwickelt und lizenziert von Apple
 - unterstützt mehrere Busmaster (d.h. auch die direkte Kommunikation zwischen zwei E/A-Geräten ist möglich)
 - bis zu 400 MBit/s
 - erfordert aufwendigere Logik auf Host- und Peripherieseite

7 Gerätetreiber

- ein **Gerätetreiber** stellt eine Softwareschicht zwischen dem Betriebssystemkern und dem E/A-Gerät dar:
 - ein Gerätetreiber ist ein **Softwaremodul**, das geräteabhängigen Code zur Steuerung von E/A-Geräten eines Typs enthält
(⇒ Betriebssystemkern bleibt unabhängig von E/A-Geräten!)
 - es sind mehrere Gerätetreiber erforderlich (z.B. für Festplatten, RS232, EPP, USB), die zum Kern hinzu gebunden werden
 - einheitliche **Treiberschnittstelle** zum Betriebssystem (Implementierungsdetails bleiben verborgen, z.B. Adressen und Inhalt der E/A-Register)
 - Betriebssystem bietet eine einheitliche (d.h. geräteunabhängige) **Systemaufrufchnittstelle** zum Benutzerprozess
 - **Beispiel**:
Der Unix-Systemaufruf `read(fd, buf, n)` liest `n` Bytes von einem beliebigen E/A-Gerät `fd` in einen Puffer `buf`, wobei das ausführende Programm keine Kenntnis von der Art des E/A-Gerätes haben muss.

7 Gerätetreiber (2)

- Schichten eines Betriebssystems zwischen Benutzerprozess und E/A-Gerät (vereinfacht):



- Benutzerprozess und Gerätetreiber arbeiten in unterschiedlichen Speicherbereichen (*user / kernel space*)

7 Gerätetreiber (3)

- Aufgaben eines Gerätetreibers (Auswahl):
 - **Initialisierung** und **Überwachung** der E/A-Gerätes durch geeignete Programmierung der E/A-Register
 - Bereitstellen einer Schnittstelle zur Annahme **abstrakter** Anfragen an ein E/A-Gerät und Umsetzen der abstrakten Anfrage in eine **konkrete** geräteabhängige Form
 - Übernahme/Übergabe und Pufferung von Daten
 - **Zuteilung** von E/A-Geräten an Benutzerprozesse (zur exklusiven oder gemeinsamen Nutzung)
 - Implementierung von Warteschlangen für E/A-Geräte
 - Verwaltung von **Zugriffsrechten**
 - Behandlung von **Unterbrechungsanforderungen**
 - Behandlung von **Fehlermeldungen** des E/A-Gerätes
 - Wahl von Parametern/Strategien zur optimalen Nutzung eines Gerätes

7 Gerätetreiber (4)

- Betriebsarten eines Gerätetreibers:

1) **Polling:**

- CPU wartet aktiv, bis E/A-Gerät bereit ist
- nach Ausführung jedes Teiltransfers vom/zum E/A-Gerät wartet CPU aktiv, bis der Transfer beendet ist

2) **Interrupt:**

- wenn E/A-Gerät noch nicht bereit ist, schläft Prozess (→ Prozesswechsel durch Betriebssystem)
- E/A-Gerät kann bei Eintritt der Bereitschaft durch Senden einer Unterbrechungsanforderung den schlafenden Prozess aktivieren
- nach Initiierung jedes Teiltransfers vom/zum E/A-Gerät schläft Prozess (→ Prozesswechsel durch Betriebssystem), bis der Transfer beendet ist.
- **Vorteil:** keine Blockierung anderer Prozesse, höhere Auslastung der CPU
- **Nachteil:** höherer Aufwand (Interruptroutinen, Sicherung der Register, ...)

7 Gerätetreiber (5)

- Arten der Ein-/Ausgabe:

1) **synchron:**

- Systemaufruf zur Ein-/Ausgabe terminiert erst, wenn die E/A-Operation vollständig abgeschlossen ist
- bei Interrupt-Betrieb kann ggf. zwischenzeitlicher Prozesswechsel durch Betriebssystem erfolgen

2) **asynchron:**

- Systemaufruf initiiert lediglich die Ein-/Ausgabe und gibt Kontrolle an den aufrufenden Prozess zurück
- sinnvoll vor allem bei Ausgabeoperationen!
- durch zusätzlichen Systemaufruf kann sich Benutzerprozess nachträglich mit Ende der E/A-Operation synchronisieren
- **Vorteil:** Benutzerprozess kann CPU während der E/A-Operation nutzen

7.1 Gerätetreiber unter Linux

- E/A-Geräte werden als **Spezialdateien** repräsentiert:
 - Öffnen der Spezialdatei ermöglicht Zugriff auf E/A-Gerät, implementiert durch Gerätetreiber
 - Geräte können wie Dateien gelesen und geschrieben werden
- zwei Gerätearten:
 - ein **zeichenorientiertes** Gerät (*char device*) gestattet den sequentiellen Zugriff auf einzelne Bytes oder auf einen Bytestrom (Beispiele: Tastatur, Maus, RS232-Schnittstelle)
 - ein **blockorientiertes** Gerät (*block device*) gestattet den wahlfreien Zugriff auf Block fester Größe (z.B. 1KByte) oder Vielfaches davon (Beispiele: Festplatte, Floppy-Disk, CD-ROM)
- jedes Gerät wird eindeutig beschrieben durch
 - Geräteart: **c** = *char device*, **b** = *block device*
 - **Major-Nummer**: eindeutiger Index für jeden Gerätetreiber
 - **Minor-Nummer**: Auswahl eines Gerätes innerhalb eines Gerätetreibers

7.1 Gerätetreiber unter Linux (2)

- Geräte in einem Linux-System (Auszug aus **/dev**):

```
crw----- 1 alfred  audio  14,  4 /dev/audio
lrwxrwxrwx 1 root    root    3   /dev/cdrom -> hdd
crw----- 1 root    root    5,  1 /dev/console
brw----- 1 alfred  disk    2,  0 /dev/fd0
brw-rw---- 1 root    disk    3,  0 /dev/hda      (Harddisk)
brw-rw---- 1 root    disk    3,  1 /dev/hda1
brw-rw---- 1 root    disk    3,  2 /dev/hda2
brw----- 1 alfred  disk   22, 64 /dev/hdd
crw-rw---- 1 root    lp      6,  0 /dev/lp0      (SPP)
lrwxrwxrwx 1 root    root   10   /dev/mouse -> /dev/psaux
crw-rw---- 1 root    root   10,  1 /dev/psaux    (PS/2)
crw-rw---- 1 root    disk    9,  0 /dev/st0      (SCSI Tape)
crw-rw-rw- 1 root    root    5,  0 /dev/tty      (Terminal)
crw--w--w- 1 alfred  tty     4,  0 /dev/tty0
crw-rw---- 1 root    tty     4,  1 /dev/tty1
crw-rw---- 1 root    uucp    4, 64 /dev/ttyS0    (RS232)
crw-rw---- 1 root    uucp    4, 65 /dev/ttyS1
drwxr-xr-x 2 root    root  4096 /dev/usb
crw-rw---- 1 root    lp   180,  1 /dev/usb/lp1
crw----- 1 alfred  root  180, 48 /dev/usb/scanner0
```

7.1 Gerätetreiber unter Linux (3)

- Hinzufügen eines zusätzlichen Gerätes im Dateisystem durch privilegierten **mknod**-Befehl
- Gerätetreiber können entweder **statisch** (d.h. beim Übersetzen des Betriebssystemkerns) oder **dynamisch** (d.h. zur Laufzeit) als **Modul** dem Betriebssystem hinzugefügt werden
- dynamisches Laden und Entladen von Modulen erfolgt mit den privilegierten Befehlen **insmod** und **rmod**

Beispiel: `mknod /dev/xx1 c 211 1`

- erzeugt zeichenorientiertes Gerät mit Namen **xx1**
- gesteuert durch den Gerätetreiber mit Index **211** (**Major-Nummer**)
- repräsentiert Gerät **1** dieses Treibers (**Minor-Nummer**)

Beispiel: `insmod xx.o` bzw. `rmod xx.o`

- lädt/entlädt den Treiber **xx** zum/vom Linux Kern
- erzeugt/entfernt einen entsprechenden Eintrag in `/proc/modules`

7.1 Gerätetreiber unter Linux (4)

- Befehle **insmod** oder **rmod** bewirken den Aufruf der Routinen **init_module** bzw. **cleanup_module** des Gerätetreibers

Beispiel: Auszug aus Quelltext **xx.c** eines Gerätetreibers **xx**

```
init_module () {  
    register_chrdev(211, "xx", &xx_fops)  
}  
  
cleanup_module () {  
    unregister_chrdev(211, "xx");  
}
```

- Funktion **register_chrdev(211, ...)** registriert den Treiber mit Major-Nummer **211** für ein zeichenorientiertes Gerät beim Kern
(⇒ diese Nummer darf zuvor noch nicht vergeben worden sein; alternativ kann Funktion bei Angabe von **0** auch die nächste freie Major-Nummer zurückliefern)
- Gerätetreiber bekommt Namen **"xx"** (z.B. für System-Fehlermeldungen)
- **xx_fops** ist eine Struktur vom Typ **file_operations**
- Funktion **unregister_chrdev(...)** entfernt Treibereintrag aus Kern

7.1 Gerätetreiber unter Linux (5)

- die Struktur `file_operations` enthält Funktionszeiger auf alle Funktionen des Treibers für ein **zeichenorientiertes** Gerät (Struktur ist definiert in `/usr/include/linux/fs.h`)

Beispiel: Definition eines Funktionszeigerfeldes für Gerätetreiber `xx` in der Datei `xx.c`:

```
static struct file_operations xx_fops = {
    NULL,                /* Zeiger für Treiberfunktion llseek */
    &xx_read,             /* Zeiger für Treiberfunktion read */
    &xx_write,            /* Zeiger für Treiberfunktion write */
    NULL,                /* Zeiger für Treiberfunktion readdir */
    NULL,                /* Zeiger für Treiberfunktion poll */
    &xx_ioctl,           /* Zeiger für Treiberfunktion ioctl */
    NULL,                /* Zeiger für Treiberfunktion mmap */
    &xx_open,            /* Zeiger für Treiberfunktion open */
    NULL,                /* Zeiger für Treiberfunktion flush */
    &xx_release,         /* Zeiger für Treiberfunktion release */
    ...
};
```

7.1 Gerätetreiber unter Linux (6)

- im Quelltext des Gerätetreiber sind die Treiberfunktionen aus dem Funktionszeigerfeld zu implementieren

Beispiel: einige Treiberfunktionen für Treiber `xx` in der Datei `xx.c`

- <code>xx_read(...)</code>	Lesen eines Byte-Stroms
- <code>xx_write(...)</code>	Schreiben eines Byte-Stroms
- <code>xx_open(...)</code>	Öffnen des E/A-Gerätes
- <code>xx_release(...)</code>	Freigeben des E/A-Gerätes
- <code>xx_ioctl(...)</code>	Absetzen gerätespezifischer Befehle und Einstellen von geräteabhängigen Parametern
- <code>xx_poll(...)</code>	Abfragen des Zustandes eines Gerätes
- <code>xx_mmap(...)</code>	Abbilden von Gerätespeicher in den Adressraum des Prozessors

7.1 Gerätetreiber unter Linux (7)

- Benutzerprozess kann **korrespondierende Systemfunktionen** aufrufen; die Zuordnung zu den Treiberfunktionen erfolgt durch das Betriebssystem
 - `int open(char* pathname, int openflag, int mode)`
Öffnen des E/A-Gerätes und Rückgabe des *File Descriptors* `fd`
 - `int read(int fd, void* buf, int count)`
Lesen von `count` Bytes vom E/A-Gerät `fd` in den Puffer `buf`
 - `int write(int fd, void* buf, int count)`
Schreiben von `count` Bytes aus Puffer `buf` zum E/A-Gerät `fd`
 - `int close(int fd)`
Schließen des E/A-Gerätes `fd` und Freigabe für andere Prozesse
 - `int ioctl(int fd, int request, ...)`
Kontrolle des E/A-Gerätes `fd` durch eine geräteabhängige Anforderung `request`; alle weiteren Parameter sind ebenso von Anforderung und E/A-Gerät abhängig!

7.1 Gerätetreiber unter Linux (8)

- **Beispiel:** Steuerung eines Gerätes über Leitung 0 des Ports SPP (in C unter Verwendung von `/dev/lp`):

```
#include <stdio.h>
#include <linux/lp.h>

#define OFF 0x00
#define ON 0x01

char data;
fd = open("/dev/lp", O_WRONLY);
if (fd == -1) {
    fprintf(stderr, "Error: cannot open device lp!");
    exit(1);
}
ioctl(fd, LPRESET);
data = ON; write(fd, &data, 1);
sleep(5);
data = OFF; write(fd, &data, 1);
close(fd);
```

7.1 Gerätetreiber unter Linux (9)

- Implementierung eines Treibers für **blockorientiertes** Gerät **yy** ist sehr aufwendig
- einige Unterschiede:
 - Hinzufügen eines Treibers erfolgt durch Funktion **register_blkdev**, Entfernen des Treibers aus Kern durch **unregister_blkdev**
 - der Aufruf von **register_blkdev** erfordert als Parameter einen Zeiger auf eine Struktur (definiert in **/usr/include/linux/fs.h**)

```
struct block_device_operations {
    int (*open) (...);
    int (*release) (...);
    int (*ioctl) (...);
    int (*check_media_change) (...);
    int (*revalidate) (...);
};
```
 - Installation eines Interrupt-Handlers **yy_handler** für Interrupt-Nr. **irq** über Systemaufruf

```
int request_irq (int irq, void (*yy_handler) (...), ...);
```

7.1 Gerätetreiber unter Linux (10)

- der blockorientierte Gerätetreiber besitzt im Ggs. zum zeichenorientierten Gerätetreiber keine direkten E/A-Funktionen
- Ein-/Ausgabe erfolgt über eine Warteschlange ***queue** mit E/A-Aufträgen, die vom Gerätetreiber initialisiert werden muss:

```
#include <linux/blkdev.h>
blk_init_queue(request_queue_t *queue,
               request_fn_proc *request);
```
- die **request**-Funktion des Block-Gerätetreibers wird vom Betriebssystem aufgerufen, wenn es den Transfer eines Blockes vom/zum E/A-Gerät anfordert

z.B. aufgrund eines zugehörigen Systemaufrufs eines Benutzerprozesses an ein blockorientiertes Gerät: **write(fd,buf,8192)**
- die **request**-Funktion setzt sämtliche Parameter des Transfers (z.B. Richtung, Pufferadresse, Blockgröße, Anzahl Blöcke, ...)

8 Lernziele

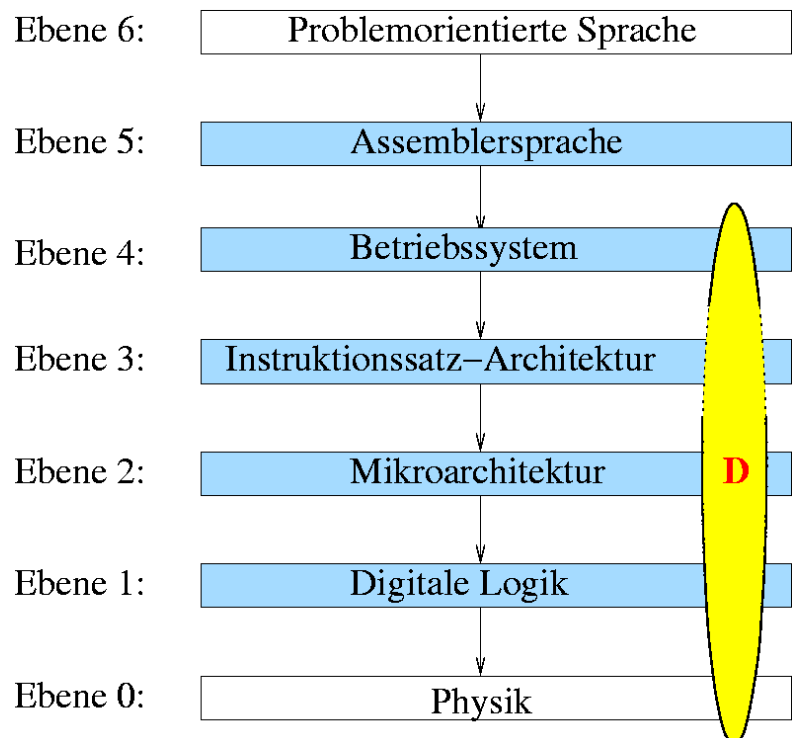
- **Konzepte der Ein-/Ausgabe:**
 - *Busy Waiting, Polling* und *Interrupts*
 - Techniken der Datenübertragung (*open loop, closed loop, fully interlocked*)
 - Adressierung und Programmierung von E/A-Bausteinen
 - DMA
 - synchrone/asynchrone Ein-/Ausgabe
 - serieller Datentransfer (z.B. Datenkodierung bei RS232 und USB)
 - E/A-Systemarchitektur eines heutigen PC
 - Aufgaben und prinzipielle Arbeitsweise eines Gerätetreibers
- **Konzepte von Bussystemen:**
 - Arten und Architektur von Bussystemen
 - Busprotokolle auf synchronem/asynchronem Bus
 - Verfahren der Busarbitrierung
 - wichtige Eigenschaften heute eingesetzter Bussysteme (PCI, USB)

D Festplatten und Dateisysteme

1. Geschichte
2. Festplattenaufbau
3. Festplattenbusse: IDE /ATA und SCSI
4. CD-ROM und Floppy
5. RAID-Systeme
6. Dateisysteme

D Ein-/Ausgabekonzepte und Bussysteme

- Einordnung in das Schichtenmodell:

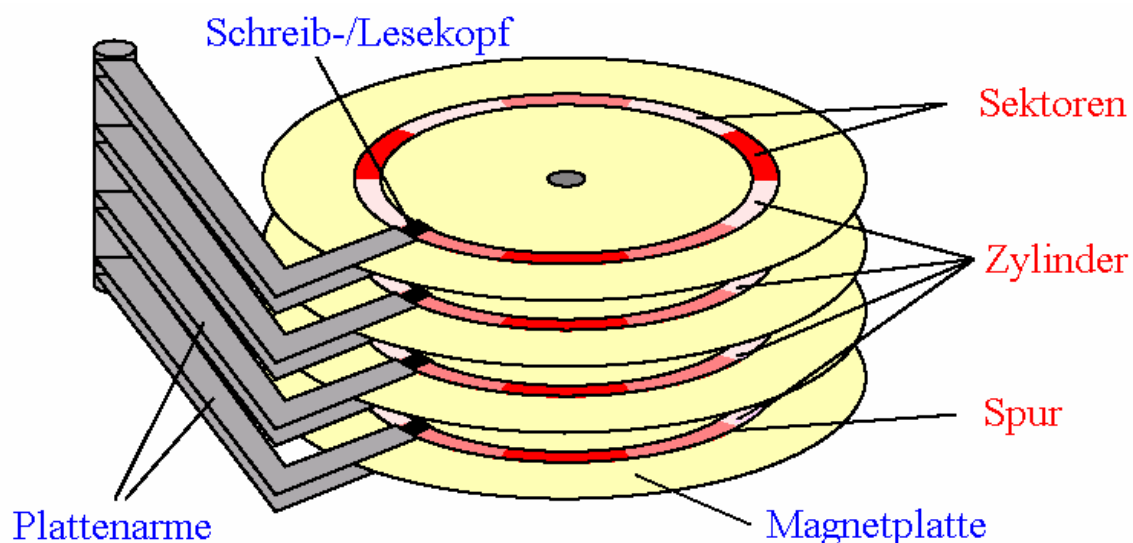


1 Geschichte

- kurzer Abriss der historischen Entwicklung:
 - 1956:** IBM stellt erste Festplatte vor (50 Scheiben mit je **24 Zoll**, insgesamt ca. 5 MByte Kapazität)
 - 1973:** 3340 Winchester Festplatte von IBM (Vorbild aller heutiger Platten)
 - 1979:** erste **5¼ Zoll** Festplatte von Seagate
 - 1981:** Einführung des SCSI-Peripheriebusses zum externen Anschluss von Festplatten
 - 1982:** Seagate stellt die ST506-Schnittstelle vor, aus der sich später IDE, ATA und ATAPI entwickelten
 - 1988:** erste **3½ Zoll** Festplatte von Conner Peripherals
 - 1992:** erste **2½ Zoll** Festplatte von PrairieTek, heute Standard für Laptops
 - 1996:** Seagate stellt Festplatten mit 10000 U/Min. vor
 - 1999:** IBM kündigt **1 Zoll** Festplatten ([Microdrive](#)) für portable Multimedia-Geräte an
- Speicherdichte stieg von ca. **300 Bit/cm²** im Jahre 1956 bis auf ca. **10 GBit/cm²** im Jahre [2003](#)

2 Festplattenaufbau

- Aufbau einer Festplatte und Organisation der Daten (hier für eine Festplatte mit 4 Magnetplatten) :

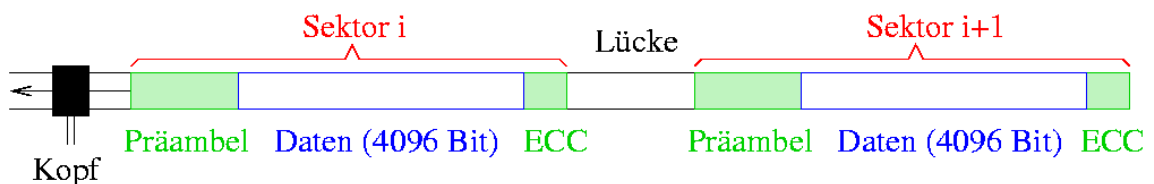


2 Festplattenaufbau (2)

- Aufbau einer Festplatte:
 - eine oder mehrere rotierende an einer gemeinsamen Achse (Spindel) befestigte **Magnetplatten** (unten und oben magnetisch beschichtet)
 - **Plattenarme** mit je zwei **Schreib-/Leseköpfen**, die
 - radial zwischen dem innersten und dem äußersten nutzbaren Radius bewegt werden können
 - mit sehr geringem Abstand ($< 0.1 \mu\text{m}$) über der Magnetplatte schweben
 - Lesen von Information erfolgt durch die **Induktion** eines Stromes im Schreib-/Lesekopf aufgrund der Rotation
 - Schreiben von Information erfolgt durch Magnetisierung der Oberfläche
 - Magnetplatten und Plattenarme mit Köpfen befinden sich im **hermetisch abgeschlossenen Gehäuse** ($\Rightarrow$ Schutz vor Feuchtigkeit, Schmutz)
 - zwei kleine **Elektromotoren**:
 - Antriebsmotor zur Rotation der Platte (typisch 5400 bis 15000 U/Min.)
 - Schrittmotor zur Bewegung des Plattenarmes (als **Aktuator** bezeichnet)

2 Festplattenaufbau (3)

- **Organisation** der Daten auf einer Festplatte:
 - eine **Spur** stellt einen konzentrischen Kreis auf einer Magnetplatte dar; Breite einer Spur: $\leq 1 \mu\text{m}$, Spuren je Zentimeter: 1000 bis 10000
 - jede Spur ist in **Sektoren** (zumeist mit **512 Byte Nutzdaten**) unterteilt:



- jeder Sektor enthält noch 40 bis 100 Bytes für Verwaltungsinformationen, (**Präambel**) und Fehlererkennung/-korrektur (**ECC**)
- es können nur ganze Sektoren gelesen oder geschrieben werden!
- bei Einsatz mehrerer Magnetplatten bilden alle übereinander liegenden Spuren einen **Zylinder** ($\Rightarrow$ mehr Daten ohne neue Kopffositionierung!)
- jeder Sektor ist somit durch **Kopfnummer**, **Zylindernummer** und **Sektornummer** eindeutig identifiziert

2 Festplattenaufbau (4)

- heutige Festplatten verwenden das **Zone Bit Recording**:
 - da die äußere Spur wesentlich länger als die innere Spur ist, können bei gleicher Datendichte hier mehr Daten gespeichert werden
 - Einteilung der Plattenoberfläche in mehrere Zonen, so dass die **äußeren Spuren mehr Sektoren** erhalten **als die inneren Spuren**
 - höhere Datenrate im äußeren Bereich möglich, da hier je Umdrehung mehr Daten gelesen bzw. geschrieben werden können
(⇒ Zylinder werden daher von außen nach innen nummeriert und beschrieben)
 - innerste Spur wird oft als **Parkposition** verwendet (Autopark)
- Signale vom Lesekopf sind sehr klein:
 - richtige Erkennung einzelner Bits ist wegen **Rauschen** (Störungen durch Nachbarspur) nicht möglich
 - statt dessen werden Bitströme trickreich kodiert, so dass Fehler korrigiert werden können (**PRML**-Kodierung)

2 Festplattenaufbau (5)

- beim **Formatieren** einer Festplatte
 - werden Sektoren angelegt und mit Präambel beschrieben (enthält z.B. die Sektor-, Kopf- und Zylinder Nummer)
 - werden fehlerhafte Sektoren (*bad sectors*) erkannt
- einige **Leistungsmerkmale** von Festplatten:
 - **Kapazität**: Speicherkapazität für Nutzdaten (in GByte)
 - **Größe des Pufferspeichers** (typisch 512 kByte bis 16 MByte): eine oder mehrere Spuren werden zwischengespeichert und von hier nur teilweise oder in anderer Reihenfolge ausgelesen
 - **Latenzzeit**: mittlere Wartezeit bei Zugriff in der gleichen Spur (entspricht Zeit für eine halbe Umdrehung)
 - **Zugriffszeit** (typisch 3.5 bis 10 ms): Summe aus mittlerer Zeit für neue Kopfpositionierung und mittlerer Wartezeit
 - **Transferrate** (typisch 500 bis 800 Mbit/s): interne Transferrate zwischen Platte und Controller

2 Festplattenaufbau (6)

- Daten zweier aktueller 3½ Zoll Festplatten (Stand 2003):

	Seagate Barracuda 7200 ST 3160021	Seagate Cheetah 15K ST 373543LW
Kapazität	160 GByte	73.4 GByte
Magnetplatten / Köpfe	2 / 4	4 / 8
Umdrehungen / Minute	7200	15000
Latenzzeit	4.2 ms	2 ms
Zugriffszeit	8.5 ms	3.6 ms
Transferrate	683 MBit/s	891 MBit/s
Bytes je Sektor	512	512
Schnittstelle	UltraATA/100	Ultra320 SCSI
Pufferspeicher	2 MByte	8 MByte
Fehlerrate	1 aus 10^{14} Bits	1 aus 10^{15} Bits

(Quelle: www.seagate.com)

2 Festplattenaufbau (7)

- die Steuerelektronik einer Festplatte (Controller) besteht aus:
 - **Kodierer / Dekodierer:**
 - Transformation von analogen Kopfsignalen in serielle digitale Signale und umgekehrt gemäß der gewählten Kodierung (z.B. PRML)
 - **Datenformatierer:**
 - trennt Nutzdaten von Präambel und erkennt/korrigiert Fehler mittels ECC (beim Lesen) oder berechnet ECC (beim Schreiben)
 - wandelt serielle digitale Daten in parallele Daten um und umgekehrt
 - **Hostadapter:**
 - Schnittstelle der Festplatte zum Systembus bzw. Ein-/Ausgabebus mit Implementierung eines Übertragungsprotokolls zum Host
 - Annahme von Kommandos und ggf. überlappte Ausführung mehrerer Kommandos
 - Steuerung und ggf. Optimierung der Kopfbewegung
 - Pufferspeicher für Inhalt mehrerer Sektoren

3.1 Festplattenbusse: IDE/ATA

- **IDE** (*Integrated Drive Electronics*, 1986):
 - nur eine Bezeichnung von Compaq und Western Digital für die **Idee**, die aufwendige Ansteuerung von der Hauptplatine auf eine Steuerplatine in der Festplatte zu verlagern
 - auf Hauptplatine des Rechners ist nur einfache Dekodierung erforderlich
 - leistet eine einfache Anbindung von Laufwerken mit Steuerplatine an den ISA-Bus
- **ATA** (*Advanced Technology Attachment*, 1989):
 - Spezifikation eines ANSI **Standards** für IDE
 - legt physikalische und elektrische Eigenschaften der Schnittstelle fest
 - definiert Protokoll zwischen Host und Plattenlaufwerk
 - Unterstützung von DMA
 - gestattet maximale Transferrate von 4,2 MByte/s
 - ein Kanal mit Anschlussmöglichkeit von zwei Laufwerken

3.1 Festplattenbusse: IDE/ATA (2)

- Kommunikation zwischen Computer und Festplatte erfolgt über den IDE/ATA-Registersatz:
 - Steuerung über **8-Bit Kontrollregister** (Auswahl):
 - 1) **Fehlerregister**: enthält ggf. einen Fehlercode des letzten Zugriffs
 - 2) **Sektorzahl-Register**: Anzahl der zu lesenden/schreibenden Sektoren
 - 3) **Sektornummer-Register**: Auswahl eines Sektors im Zylinder
 - 4) **Zylindernummer-Register** (low Byte): Auswahl eines Zylinders
 - 5) **Zylindernummer-Register** (high Byte): dito
 - 6) **Laufwerk-Register**: Auswahl eines von zwei Laufwerken (Bit 4) und Kopfnummer (Bits 3 bis 0)
 - 7) **Kommando- und Statusregister**:
 - beim Schreibzugriff: Übermittlung eines von 50 möglichen Befehlen (z.B. *Identify Drive*, *Seek*, *Read Sector*, *Write Sector*, *Format Track*)
 - beim Lesezugriff: Statusregister, enthält Signale wie *drive busy*, *drive ready*
 - Datentransfer über ein **16-Bit Datenregister**

3.1 Festplattenbusse: IDE/ATA (3)

- zwei Verfahren zur Adressierung eines Sektors:
 - 1) **CHS-Modus** (*Cylinder/Head/Sector*):
 - 4-Bit Kopfnummer im Laufwerkregister: max. 8 Platten, 16 Köpfe
 - 16-Bit Zylindernummer: max. 65536 Zylinder
 - 8-Bit Sektornummer: max. 256 Sektoren je Spur
 - bei einer Nutzdatengröße von 512 Byte je Sektor beträgt die maximale Festplattenkapazität $16 \times 65536 \times 256 \times 512 \text{ Byte} = 128 \text{ GByte}$
 - wegen beschränkter Sektor- und Zylinderanzahl und Zone Bit Recording werden heute CHS-Adressen einer logischen Geometrie angegeben und vom Controller in die physikalischen CHS-Adressen umgerechnet
 - 2) **LBA-Modus** (*Logical Block Address*):
 - Sektoren werden über logische 28-Bit Blocknummern angesprochen
 - ausgewählt, wenn Bit 6 im Laufwerk-Register auf 1 gesetzt wird
 - gleiche Einschränkung bei Plattenkapazität wie bei CHS-Modus

3.1 Festplattenbusse: IDE/ATA (4)

- IDE/ATA-Festplatten werden durch 40-polige Flachbandkabel und 40-polige Steckverbindungen mit Hauptplatine verbunden
- je Kabel 3 Pfostenstecker (zum Anschluß von 2 Festplatten)
- Signale auf IDE/ATA-Bus (Auswahl):
 - Reset: Laufwerke zurücksetzen
 - D0-D15: 16 Datenleitungen
 - A0-A2: 3 Adressleitungen (zur Registerauswahl)
 - DMACK, DMARQ: DMA Leitungen
 - IRQ: Interrupt Request
 - I/O Read, I/O Write: Lese-/Schreibzugriff
 - IORDY: Bereitschaftsmeldung, ggf. Anforderung von Wartezyklen
 - PDIAG, DASP, SPSYNC: Festplattenspezifische Signale
- auch 80-polige Flachbandkabel (mit 40 zusätzlichen Masseleitungen zur Abschirmung), nötig bei höheren Transferraten

3.1 Festplattenbusse: IDE/ATA (5)

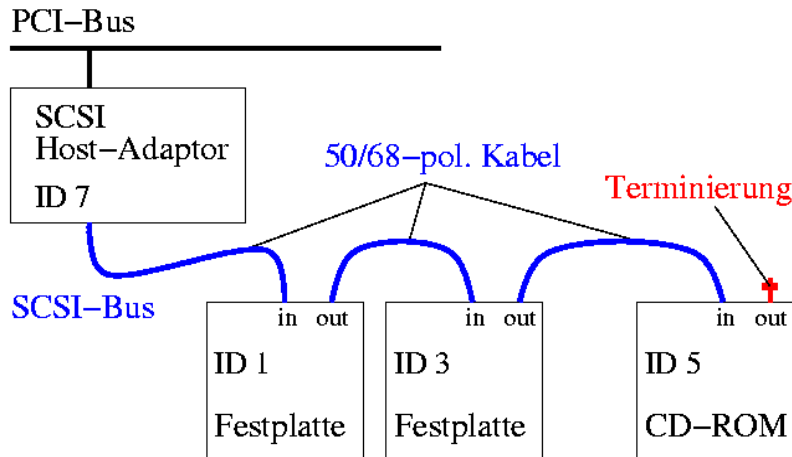
- Erweiterungen von **ATA** (Auswahl):
 - 1) **FastATA** oder **ATA-2** (1996):
 - Verdopplung der Transferrate auf 8.4 MByte/s
 - 2) **ATAPI** (ATA *Packet Interface*, auch in **ATA-4** enthalten):
 - Erweiterung des Standards zum Anschluss von CD-ROMs
 - 3) **UltraATA/66 (ATA-5)**, **UltraATA/100 (ATA-6)**, **UltraATA/133**
 - Transferraten von max. 66, 100 bzw, 133MByte/s
 - 4) **SerialATA**:
 - serielle Datenübertragung
 - einfachere Kabel, kleinere, preiswertere Steckverbinder
 - Transferrate bis zu 150 MByte/s
- **EIDE** (*Enhanced IDE*) ist kein Standard!
 - Oberbegriff von Western Digital für einige ATA-Erweiterungen; zusätzlich Definition von zwei Kanälen mit je zwei Festplatten

3.2 Festplattenbusse: SCSI

- **SCSI** (*Small Computer System Interface*) wurde 1979 von der Firma Shugart entwickelt (ursprünglich als SASI bezeichnet) und 1986 von ANSI als SCSI-1 standardisiert
(viele Anbieter haben sich zur SCSITA = SCSI *Trade Association* zusammengeschlossen ⇒ www.scsita.org)
- paralleles **Bussystem** zum Anschluss **diverser Peripheriegeräte** (vor allem Platten, CD-ROMs, Scanner und Bandgeräte):
 - es gibt **8-Bit** und **16-Bit** SCSI-Busse
 - Unterstützung von 8 Geräten (bei 8 Bit) bzw. 16 Geräten (bei 16 Bit) je Bus, die alle **gleichberechtigt** sind
 - Hostrechner (PC, Workstation) repräsentiert ein Gerät am SCSI-Bus; auch mehrere Rechner an einem SCSI-Bus prinzipiell möglich
 - direkter Datentransport zwischen den Geräten möglich
 - jedes SCSI-Gerät kann einen Bustransfer initiieren

3.2 Festplattenbusse: SCSI (2)

- Aufbau und Verkabelung eines SCSI-Busses:



- jedes SCSI-Gerät hat i.a. zwei gleichwertige Stecker zum Aufbau einer Verkettung
- **Terminierung**: das letzte Gerät am SCSI-Bus benötigt aus elektrischen Gründen einen Abschlusswiderstand

3.2 Festplattenbusse: SCSI (3)

- jedem SCSI-Gerät wird über einen Schalter (z.B. Jumper) eine **eindeutige Nummer** (SCSI ID) zugewiesen
(ein SCSI-Gerät kann zudem aus mehreren logischen Geräten bestehen, die durch LUNs = *Logical Unit Numbers* unterschieden werden, z.B. mehrere Partitionen einer Festplatte)
- Busarbitrierung** auf SCSI-Bus:
 - möchte SCSI-Gerät mit ID i Initiator werden, so aktiviert es zunächst die i -te Datenleitung (sofern Bus frei ist)
 - wird innerhalb einer festen Zeitspanne ($2.4 \mu\text{s}$) keine andere Datenleitung aktiviert, so wird SCSI-Gerät mit ID i Initiator und **setzt** Signal **BSY** (bei einem Konflikt gewinnt SCSI Gerät mit höchster ID)
 - Bustransfer kann nicht unterbrochen werden; erst durch **Rücknahme** von **BSY** wird Bus explizit freigegeben

3.2 Festplattenbusse: SCSI (4)

- ein SCSI-Bustransfer erfolgt in mehreren Phasen:
 - 1) **Bus Free**: SCSI-Bus ist von keinem Gerät belegt
 - 2) **Arbitration**: Busarbitrierung, vgl. letzte Folie
 - 3) **Selection**: *Initiator i* ruft *Target j* durch Aktivierung der *j*-ten Datenleitung
 - 4) **Message Out**: *Initiator i* kann Mitteilung an *Target j* senden
 - 5) **Command**: *Initiator i* sendet ein Buskommando an *Target j*, das aus einem 1-Byte Befehlscode (z.B. Inquiry oder Read) und gerätespezifischen Parametern (z.B. Zylinder-, Kopf- und Sektornummer) besteht
 - 6) **Data In** oder **Data Out**: eigentlicher Datentransfer
 - 7) **Status**: *Target* informiert *Initiator* über Status des Transfers
 - 8) **Message In**: *Target j* kann weitere Mitteilungen an *Initiator i* senden
- Geräte können **asynchronen** Bustransfer (Quittierung nach jedem Wort) oder **synchronen** Bustransfer (Quittierung nach vereinbartem Zeitintervall) vereinbaren

3.2 Festplattenbusse: SCSI (5)

- ein **8-Bit SCSI-Bus** benötigt einen **50-poligen** Stecker:
 - 8 Datenleitungen und 1 Leitung für Paritätsbit
 - 9 Steuerleitungen, z.B.:
 - **BSY**: SCSI-Bus belegt (*Busy*)
 - **SEL**: Anzeige der **Selection** Phase
 - **MSG**: Anzeige einer **Message** Phase
 - **ACK**: Bestätigung eines Transfers
 - **RST**: Rücksetzen *aller* SCSI-Geräte am Bus (*Reset*)
 - **ATN**: Initiator bittet um Aufmerksamkeit (*Attention*)
 - jede Signalleitung ist doppelt vorhanden:
 - Masseleitung zur Abschirmung für Betriebsart SE (*Single Ended*)
 - für differentielle Kodierung bei Betriebsart LVD (*Low Voltage Differential*)
 - +5V Spannungsleitung (nötig für Terminierung !)
- ein **16-Bit SCSI-Bus** benötigt **68-poligen** Stecker:
 - 8 weitere Datenleitungen
 - 1 weitere Leitung für Paritätsbit

3.2 Festplattenbusse: SCSI (6)

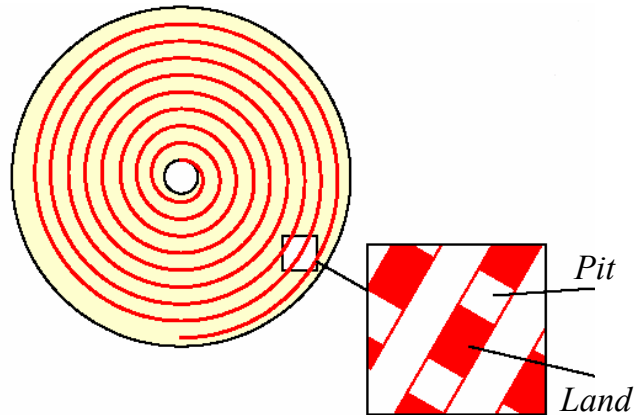
- verschiedene SCSI-Standards:
 - **SCSI-1** (1986): **5 MHz Bustakt**, 8-Bit Bus, 5 MByte/s, SE
 - **SCSI-2** (1990, Fast SCSI): **10 MHz Bustakt**, 8- oder 16-Bit Bus, SE
 - 10 MByte/s bei 8-Bit (*Fast narrow* SCSI)
 - 20 MByte/s bei 16-Bit (*Fast wide* SCSI)
 - erweiterter Befehlssatz z.B. zur Steuerung von CD-ROMs
 - **SCSI-3** (1996) ist eine Sammlung mehrerer Standards:
 - **Ultra**: **20 MHz Bustakt**, 20 MByte/s bei 8-Bit (*Ultra narrow* SCSI) und 40 MByte/s bei 16-Bit (*Ultra wide* SCSI), SE
 - **Ultra2**: **40 MHz Bustakt**, bis zu 80 MByte/s bei 16-Bit (*Ultra2 wide* SCSI), längere Leitungen (bis zu 25m) durch LVD
 - **Ultra160**: 40 MHz Bustakt, Datenübertragung bei **steigender** und **fallender** Taktflanke, LVD, max. 160 MByte/s bei 16-Bit
 - **Ultra320**: **80 MHz Bustakt**, LVD, bis zu 320 MByte/s
 - **Serial SCSI**: serielles Kabel (bis zu 10 km Länge); max. 400 MByte/s

3.3 Festplattenbusse: SCSI und IDE/ATA

- Vergleich von SCSI und IDE/ATA:
 - + SCSI erlaubt den Anschluss von Platten **beliebiger Kapazität**
 - + SCSI erlaubt zur Zeit (2003) **höhere Übertragungsraten** als IDE (sinnvoll jedoch i.a. nur beim Betrieb mehrerer Platten)
 - + SCSI erlaubt **8 bis 16 Geräte je Bus**, IDE nur zwei Geräte je Kanal
 - + SCSI erlaubt je Gerät eine **Warteschlange** für bis zu 256 Kommandos, die überlappt und ggf. in anderer Reihenfolge abgearbeitet werden können (*Command Queueing and Reordering*)
 - SCSI benötigt **zusätzlichen Host-Adaptor** auf Hauptplatine oder im PCI-Slot zur Implementierung des SCSI-Protokolls
 - IDE ist einfacher zu implementieren und hat einen größeren Absatzmarkt (Geräte mit IDE/ATA Interface sind **preiswerter** als mit SCSI)
 - unterschiedliche Steckervarianten
- ⇒ SCSI ist führendes Bussystem im High-End Bereich (z.B. für Server oder Hochleistungs-Workstations)

4.1 CD-ROM

- Entwicklung von Philips und Sony (1984, *Yellow Book*)
- Aufbau einer CD (Audio +CD-ROM):
 - 5 ¼ Zoll Plastikscheibe mit Substrat, überzogen mit reflektierender Silberschicht und transparenter Schutzschicht
 - Substrat erhält durch Herstellungsprozess in der Spur eine Serie von Löchern (*Pit*) und Plateaus (*Land*)
 - **spiralförmige Spur**, innen beginnend, 22200 Windungen, 1.6 µm Breite, 5.68 km Länge
 - Abtastung mit Laserstrahl:
Land: gute Reflexion
Pit: schlechte Reflexion
 - Kodierung: jeder Wechsel
Land → *Pit* und *Pit* → *Land* repräsentiert eine „1“



4.1 CD-ROM (2)

- Organisation der Daten auf CD-ROM:
 - **Symbol**: jedes Byte wird mit 14 Bit kodiert
(Grund: *Pit* bzw. *Land* müssen eine Mindestlänge aufweisen)
 - **Frame** enthält 588 Bit
(24 Byte Nutzdaten, 8 Byte für Fehlerkorrektur, Rest für Synchronisation)
 - **Sektor** besteht aus 98 Frames
(16 Byte Präambel, 2048 Byte Nutzdaten, 288 zusätzliche Byte für Fehlerkorrektur)
⇒ für 2048 Byte Nutzdaten sind insgesamt 7023 Byte erforderlich!
- Abtastung mit linearer Geschwindigkeit von 1.2 m/s
- Transferrate:
 - 1x-Laufwerk: 75 Sektoren/s (153.6kBytes/s)
 - 52x-Laufwerk: 3000 Sektoren/s (7.98 MByte/s)
- Speicherkapazität beträgt ca. **650 MByte**

4.1 CD-ROM (3)

- Varianten der CD-ROM (Philips, *Orange Book*, 1988)
 - **CD-R** (*Recordable*)
 - reflektierende Farbschicht auf Substrat
 - leistungsstarker Laserstrahl kann Reflexivität der Farbschicht irreversibel verändern und somit nicht reflektierende *Pits* in der Spur erzeugen
 - i.a. mit normalen CD-ROM Laufwerken lesbar
 - Schreibmodi: *Disk-at-Once* oder *Multi-Session* (iterativ beschreibbar)
 - **CD-RW** (*ReWritable*)
 - Schicht mit zwei stabilen Zuständen (polykristallin mit hoher Reflexivität und amorph mit geringer Reflexivität) auf Substrat
 - Zustandswechsel durch Erhitzen auf verschiedene Temperaturen (200°C für polykristallin und 500 °C für amorph) mittels Laserstrahl
 - vielfach beschreibbar!
 - aufgrund geringerer Reflexivität nicht kompatibel zu CD-ROMs
 - Rohlinge sind geringfügig teurer als bei CD-R

4.1 CD-ROM (4)

- **DVD** (*Digital Versatile Disk*, 1996)
 - Weiterentwicklung der CD-ROM:
 - schmalere Spur (Breite 0.74 µm statt 1.6 µm)
 - kleinere *Pits* (Länge nur 0.4 µm statt 0.8 µm), möglich durch bessere Optik und Einsatz eines Laserstrahls kürzerer Wellenlänge (635 nm statt 780 nm)
 - Lesegeschwindigkeit 3.5 m/s statt 1.2 m/s (1x)
 - jedoch höhere Empfindlichkeit gegenüber Fingerabdrücken und Kratzer
 - Kapazität: **4.7 GByte** (einseitig), 9.4 GByte (zweiseitig)
- wiederbeschreibbare Varianten der DVD:
 - **DVD-R** (Pioneer, 1997): nur einmal beschreibbar
 - **DVD-RAM** (Toshiba, Hitachi, 1998): ca. 100000-fach beschreibbar
 - **DVD-RW** (Pioneer, 2001): ca. 1000-fach beschreibbar
 - **DVD+RW** (Sony, Philips, ... , 2001): ca. 1000-fach beschreibbar
 - kaum Kompatibilität!

4.2 Floppy

- eine Diskette hat einen ähnlichen Aufbau wie eine Festplatte
 - flexible magnetisierbare Scheibe in einer Plastikschatzhülle
 - maximal zwei Köpfe (Ober- und Unterseite)
 - Köpfe berühren die Diskettenoberfläche (nicht bei ZIP)
- typische Daten einiger Diskettenformate:

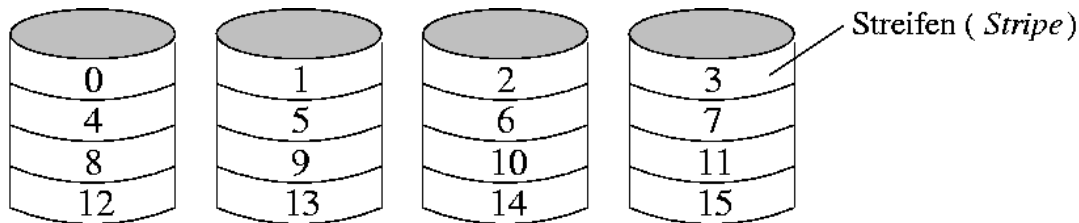
Bezeichnung	3 ½ Zoll, HD	3 ½ Zoll, ED	ZIP
Kapazität (formatiert)	1.44 MByte	2.88 MByte	100 MByte
Köpfe	2	2	2
Spurbreite	187.5 µm	187.5 µm	12 µm
Spuren je Seite	80	80	1817
Sektoren je Spur	18	36	ZBR
Transferrate	62.5 kByte/s	125 kByte/s	1 MByte/s
Umdrehungen / Minute	360	360	3000

5 RAID-Systeme

- Idee von Patterson et al. (University of Berkeley, 1988): Einsatz mehrerer unabhängiger Platten (**RAID** = *Redundant Array of Independent Disks*) zur
 - Erhöhung der Transferleistung durch **verteilte** Speicherung
 - Erhöhung der Ausfallsicherheit durch **redundante** Speicherung
- Steuerung eines RAID entweder in Software (Betriebssystem) oder in Hardware (eigener Controller) implementierbar
- heute (2003) existieren 8 verschiedene RAID-Varianten
 - als RAID 0 bis RAID 7 bezeichnet
 - jedoch werden nur **RAID 0**, **RAID 1**, **RAID 4** und **RAID 5** in der Praxis eingesetzt und hier behandelt
- Voraussetzung: mehrere Platten gleicher Größe, schneller Festplattenbus

5.1 RAID 0

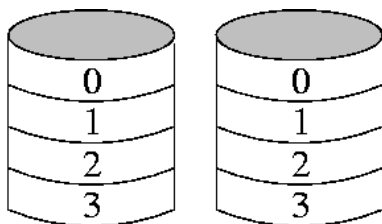
- auch als **gestreifte Platten** (*Striping*) bezeichnet
- eine logische Platte wird in kleine Streifen (*Stripes*) zerschnitten, die zyklisch über mehrere physikalische Platten verteilt werden



- **Vorteile:**
 - schnellere Datentransfers, da bei Zugriff auf eine große Datei mehrere Platten gleichzeitig angesprochen werden
- **Nachteile:**
 - keine Redundanz, d.h. bei Ausfall einer Platte fällt RAID-System aus

5.1 RAID 1

- auch als **gespiegelte Platten** (*Mirroring*) bezeichnet
- Dateien werden gleichzeitig auf zwei Platten gespeichert

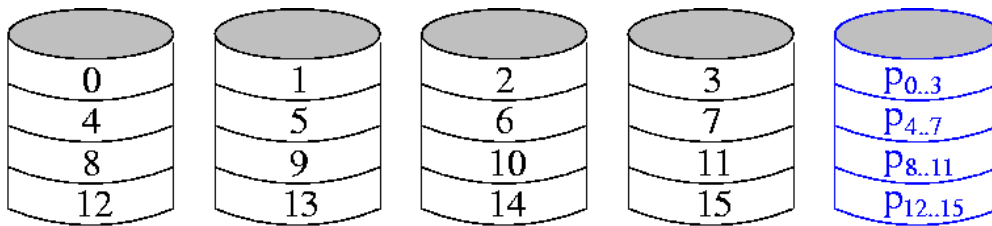


(auch mit RAID 0 kombinierbar)

- **Vorteile:**
 - schnelleres Lesen großer Dateien, da Zugriffe über zwei Platten verteilt werden können
 - auch bei Ausfall einer Platte noch betriebsbereit
- **Nachteile:**
 - geringfügig langsames Schreiben durch Warten auf zwei Plattenaufträge
 - doppelter Speicherbedarf und somit doppelte Kosten für Festplatten

5.1 RAID 4

- Dateien werden wie bei RAID 0 über mehrere Platten verteilt; eine zusätzliche **Paritätsplatte** speichert Parität



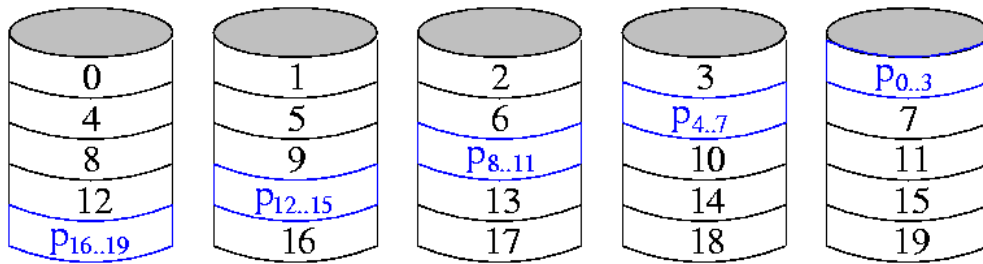
- jeder Paritätsblock enthält die bitweise Parität p von allen zugehörigen Datenblöcken
z.B. für Bit i in den Streifen 0 bis 3 gilt: $p_{0..3}(i) = x_0(i) \oplus x_1(i) \oplus x_2(i) \oplus x_3(i)$
- **Vorteile:**
 - schnelleres Lesen, da Zugriffe über mehrere Platten verteilt werden
 - auch bei Ausfall einer Platte noch betriebsbereit, z.B. bei Ausfall der Platte 1 kann $x_1(i) = x_0(i) \oplus p_{0..3}(i) \oplus x_2(i) \oplus x_3(i)$ rekonstruiert werden

5.1 RAID 4 (2)

- **Nachteile:**
 - jeder Schreibvorgang eines einzelnen Datenblocks j erfordert die Aktualisierung des zugehörigen Paritätsblocks:
$$p^{\text{neu}}(i) = p^{\text{alt}}(i) \oplus x_j^{\text{neu}}(i) \oplus x_j^{\text{alt}}(i)$$
 - erfordert Lesen der alten Inhalte von Paritätsblock und Datenblock j , d.h. vier Zugriffe je Schreibvorgang!
 - Paritätsplatte ist hoch belastet

5.1 RAID 5

- wie RAID 4, jedoch mit **verteilten Paritätsblöcken**



- **Vorteile** und **Nachteile**:
 - wie RAID 4, jedoch wird die zusätzliche Belastung durch Schreiben des Paritätsblocks auf alle Platten verteilt
- RAID 5 ist die heute gängigste RAID-Variante!

6 Dateisysteme

- ein Dateisystem ist eine Abstraktion des Betriebssystems zur **geräteunabhängigen** Verwaltung von Dateien
 - einheitliche Sicht auf verschiedene Arten von Sekundärspeicher, z.B. Festplatten, Floppys, CD-ROMs, DVDs und Bandlaufwerke
 - Benutzer muss sich nicht um physikalische Datenformate auf den verschiedenen Arten der Sekundärspeicher kümmern
 - jede Datei wird als eine Menge von **Blöcken** fester Größe repräsentiert (wobei ein Block einem oder mehrerer Sektoren entsprechen kann)
- Dateisysteme bestehen aus
 - **Dateien** (*Files*) zur Speicherung von ausführbaren Programmen und Daten (Quelltexte, Dokumente, Bilder, ...) und
 - **Verzeichnissen** (*Directories*) zur hierarchischen Strukturierung von Dateien
 - **Verknüpfungen** (*Links*)
 - **Spezialdateien** (*Special Files*)

6 Dateisysteme (2)

- eine **Partition** ist eine Menge von Verzeichnissen und deren Dateien auf einem Teil eines Datenträgers
 - dient zum Trennen von Teilbereichen einer Festplatte oder CD
($\Rightarrow$ die Zylinder einer Festplatte können in mehrere unabhängige Partitionen eingeteilt werden, z.B. für Betriebssystem und Benutzerdateien)
 - jede Partition erhält ihr eigenes Dateisystem
 - einige Dateisysteme gestatten das **Montieren** (Einhängen) weiterer Partitionen (z.B. von anderen Geräten)
- **Anforderungen** an ein Dateisystem:
 - persistente Speicherung von Daten in Dateien
 - schneller Zugriff auf Dateien
 - gleichzeitiger Zugriff durch mehrere Prozesse
 - Implementierung von Schutzrechten

6 Dateisysteme (3)

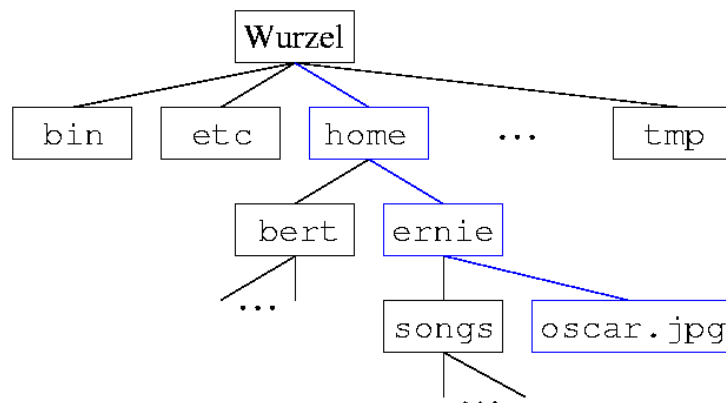
- typische **Attribute** von Dateien:
 - **Symbolischer Name**, vom Benutzer les- und interpretierbar
(i.a. bestehend aus Bezeichnung + Erweiterung, z.B. `loesung1.pdf`)
 - **Typ**, z.B. zeichenorientierte Datei (für sequentiellen Zugriff), satzorientierte Datei (für wahlfreien Zugriff), ausführbare Binärdatei, Spezialdatei (für E/A-Gerät), ...
 - **Größe** (in Bytes oder Dateiblöcken)
 - Identifikation des **Eigentümers** (*User ID*)
 - **Zugriffsrechte**
 - **Zeitstempel**, z.B. der Erstellung und der letzten Modifikation
(wichtig für inkrementelles Backup und Entwicklungswerkzeuge)
 - **Ortsinformation**, d.h. Ort der physikalischen Speicherung
(z.B. Sektoren auf einer Festplatte)

6 Dateisysteme (4)

- typische **Operationen** auf Dateien:
 - **Erzeugen** (*create*): Eintrag für leere Datei im Verzeichnis erstellen und Attribute setzen
 - **Löschen** (*delete*): Eintrag für Datei im Verzeichnis löschen und Freigeben des Speicherplatzes auf Datenträger
 - **Öffnen** (*open*): Zugriff durch Prozess und Laden der Attribute
 - **Schließen** (*close*): Freigabe durch Prozess
 - **Lesen** (*read*): Lesen an aktueller Position eines Dateizeigers
 - **Schreiben** (*write*): dito, für Schreiben
 - **Anfügen** (*append*): Schreiben neuer Daten am Ende einer Datei
 - **Positionieren** (*seek*): Dateizeiger auf bestimmte Stelle setzen

6 Dateisysteme (5)

- ein **Verzeichnis** enthält Namen von Dateien und ggf. Namen von Unterverzeichnissen
 - auch als Ordner (*Folder*) bezeichnet
 - Benutzer kann Dateien in logischen Gruppen arrangieren
 - Aufbau einer hierarchischen Verzeichnis-Struktur unter Berücksichtigung mehrerer Benutzer:



- ermöglicht einfaches Auffinden von Dateien durch Benutzer

6 Dateisysteme (6)

- Zugriff i.a. über **Pfadnamen**
 - Namen aller Verzeichnisse auf dem Pfad von der Wurzel bis zur Datei, durch spezielles Trennzeichen separiert
 - Unix/Linux: `/home/ernie/oscar.jpg`
 - Windows: `C:\home\ernie\oscar.jpg`
- typische **Operationen** auf Verzeichnissen:
 - **Erzeugen** (`mkdir`)
 - **Löschen** (`rmdir`), i.a. nur von leeren Verzeichnissen möglich
 - **Öffnen** (`opendir`) und **Schließen** (`closedir`)
 - **Lesen** eines **Verzeichniseintrags** (`readdir`)
 - Schreiben eines neuen Verzeichniseintrags erfolgt i.a. implizit beim Erzeugen einer neuen Datei
 - Löschen eines Verzeichniseintrags erfolgt i.a. implizit beim Löschen einer Datei bzw. eines Unterverzeichnisses

6.1 Linux-Dateisystem EXT2

- EXT2 ist seit 1992 das Standard-Dateisystem für Linux (heute i.a. ersetzt durch Nachfolger EXT3 mit Journaling)
- arbeitet auf Blöcken der Größe 1 kByte, 2 kByte oder 4 kByte (beim Anlegen des Dateisystems durch `mke2fs` als Parameter wählbar)
- **Datei** in EXT2:
 - unstrukturierte Bytefolge mit beliebigem Inhalt
 - Zugriffsrechte: lesbar (r), schreibbar (w), ausführbar (x), separat für Eigentümer, Gruppe und alle anderen Nutzer setzbar
- **Verzeichnis** in EXT2:
 - jedem Prozess ist ein aktuelles Verzeichnis zugeordnet (*cwd* = *current working directory*) zugeordnet
 - Zugriffsrechte: lesbar (r), schreibbar (w), durchsuchbar (x), separat für Eigentümer, Gruppe und alle anderen Nutzer setzbar

6.1 Linux-Dateisystem EXT2 (2)

- **Eigentümer** und **Zugriffsrechte**:
 - jeder Benutzer wird durch eine eindeutige *User ID* (UID) repräsentiert
 - UID 0 ist reserviert für Supervisor („root“)
 - jeder Benutzer gehört einer oder mehrerer Gruppen an, die durch eine eindeutige Nummer (GID, *Group ID*) repräsentiert werden
 - jede Datei und jedes Verzeichnis ist genau einer UID und einer GID zugeordnet
 - nur Eigentümer darf Rechte zum Lesen (r), Schreiben (w) und Ausführen/Durchsuchen (x) von Dateien/Verzeichnissen ändern
 - Rechte für Eigentümer, Gruppenangehörige, und alle anderen Nutzer können separat gewählt werden
 - Ändern der Zugriffsrechte erfolgt mit Unix-Kommando `chmod`
Beispiel: `chmod g+w,o+r myfile`

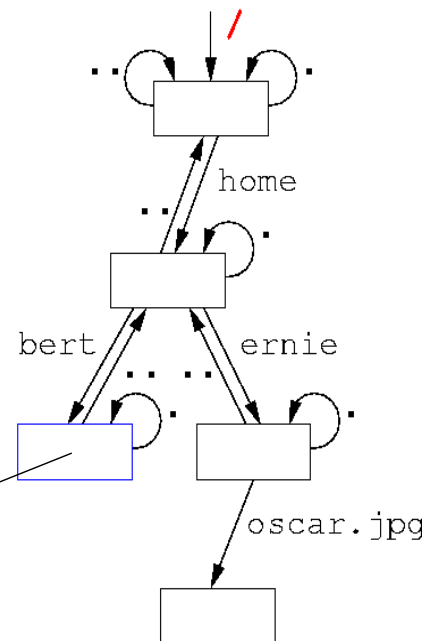
6.1 Linux-Dateisystem EXT2 (3)

- alle Attribute einer **Datei** werden in einem **Inode** (*Index node*) gespeichert:
 - jeder Inode hat eine eindeutige **Inode-Nummer** und belegt 128 Byte
 - Inodes sind für jede Partition getrennt durchnummeriert
 - Inhalt eines Inodes:
 - Dateityp (*mode*), z.B. einfache Datei, Verzeichnis, Spezialdatei
 - Eigentümer (UID) und Gruppenzugehörigkeit (GID)
 - Zugriffsrechte
 - Zugriffszeiten: letzte Änderung (*mtime*), letzter Zugriff (*atime*), letzte Änderung des Inodes (*ctime*)
 - Anzahl der *Hard Links* auf diesen Inode
 - Dateigröße in Bytes
 - Adressen der zugehörigen Dateiblöcke:
12 direkte Adressen und 3 indirekte Adressen (einfach, doppelt und dreifach)
mit jeweils 32 Bit pro Blockadresse
- (vgl. `struct ext3_inode.h` in `/usr/include/linux/ext3_fs.h`)

6.1 Linux-Dateisystem EXT2 (4)

- **Dateibaum** in EXT2:

- Wurzelverzeichnis ist „/“
- nicht Dateien und Verzeichnisse, sondern die Verbindungen zwischen ihnen sind benannt
- jedes Verzeichnis hat einen Verweis auf sich selbst („.“) und auf das übergeordnete Verzeichnis („..“)
- Pfadnamen mit Trennzeichen „/“ (*Slash*)
 - **absoluter** Pfad von **Wurzel** aus, z.B. /home/ernie/oscar.jpg
 - **relativer** Pfad vom **aktuellen Verzeichnis** aus, z.B. ../ernie/oscar.jpg
- Dateien und Verzeichnisse sind somit über mehrere Pfadnamen ansprechbar

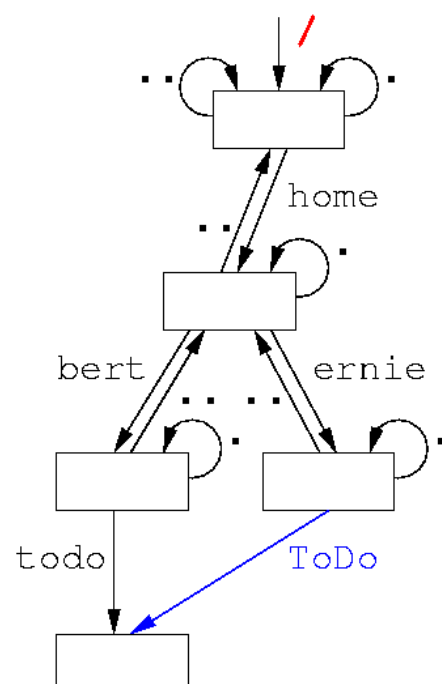


6.1 Linux-Dateisystem EXT2 (5)

- zwei Arten von **Verknüpfungen** (*Links*):

- 1) Feste Verknüpfung (*Hard Link*)

- Dateien können mehrere auf sich zeigende Verweise haben (auch aus verschiedenen Verzeichnissen)
- erzeugbar durch Unix-Kommando `ln`
Beispiel (*cwd* sei /home/ernie):
`ln ../bert/todo ToDo`
- Datei wird erst gelöscht, wenn die letzte feste Verknüpfung gelöscht wird
- nur bei Dateien erlaubt!
- nur innerhalb einer Partition möglich!



6.1 Linux-Dateisystem EXT2 (6)

2) Symbolische Verknüpfung (*Symbolic Link*)

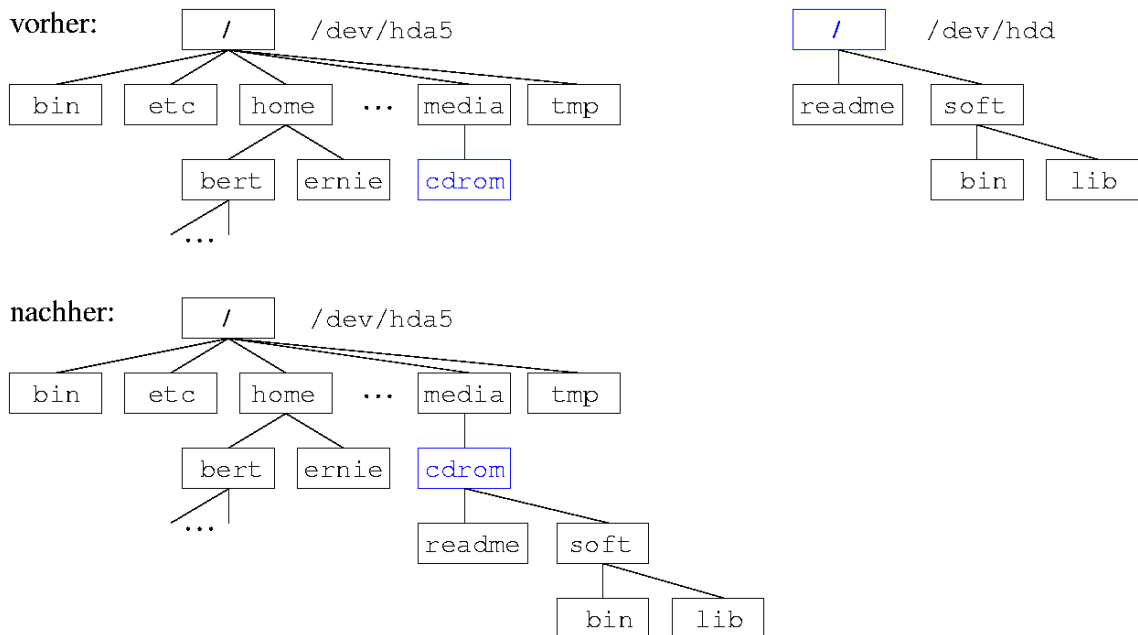
- statt eines festen Verweises auf den Inode der Zielfeile wird hier dessen **symbolischer Pfadname** als Verknüpfung gespeichert
- benötigt jeweils einen Inode
- erzeugbar durch Unix-Kommando `ln -s`
Beispiel: `ln -s /home/bert/todo /home/ernie/ToDo`
- bei kurzen Pfadnamen (bis zu 60 Zeichen) kann dieser direkt im Inode des gespeichert werden (*Fast Symbolic Link*), ansonsten ist zusätzlicher Plattenblock erforderlich
- geringe Geschwindigkeit, da Pfadnamen erst interpretiert werden müssen
- höhere Transparenz als bei festen Verknüpfungen
- auch zwischen Partitionen möglich!
- auch bei Verzeichnissen erlaubt!

6.1 Linux-Dateisystem EXT2 (7)

- Linux-Dateibaum kann aus mehreren **Partitionen** zusammenmontiert werden
 - jede Partition hat ihr eigenes Dateisystem
 - eine Partition wird durch blockorientierte Spezialdatei repräsentiert
Beispiel: `/dev/hda7` oder `/dev/fd0`
 - Einhängen (Montieren) einer neuen Partition erfolgt durch privilegierten Linux-Befehl `mount`
Beispiel: `mount /dev/hda7 /data`
 - Entfernen erfolgt durch privilegierten Linux-Befehl `umount`
 - das *Root File System* (mit Wurzel „/“) stellt das Wurzelverzeichnis des Gesamtsystems dar

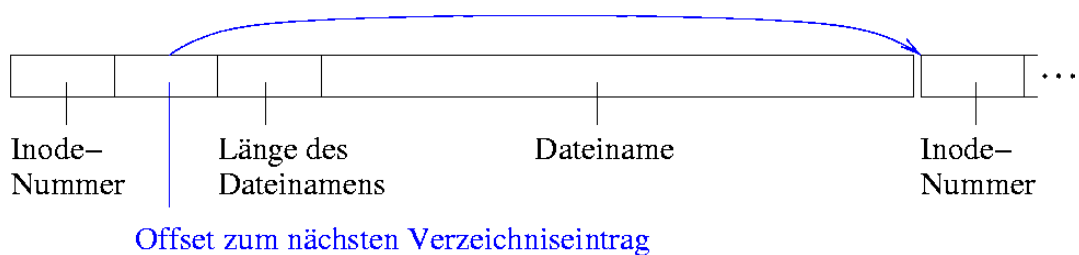
6.1 Linux-Dateisystem EXT2 (8)

- Beispiel: Montieren einer CD-ROM in ein Dateisystem mittels des Unix-Kommandos `mount /dev/hdd /media/cdrom`



6.1 Linux-Dateisystem EXT2 (9)

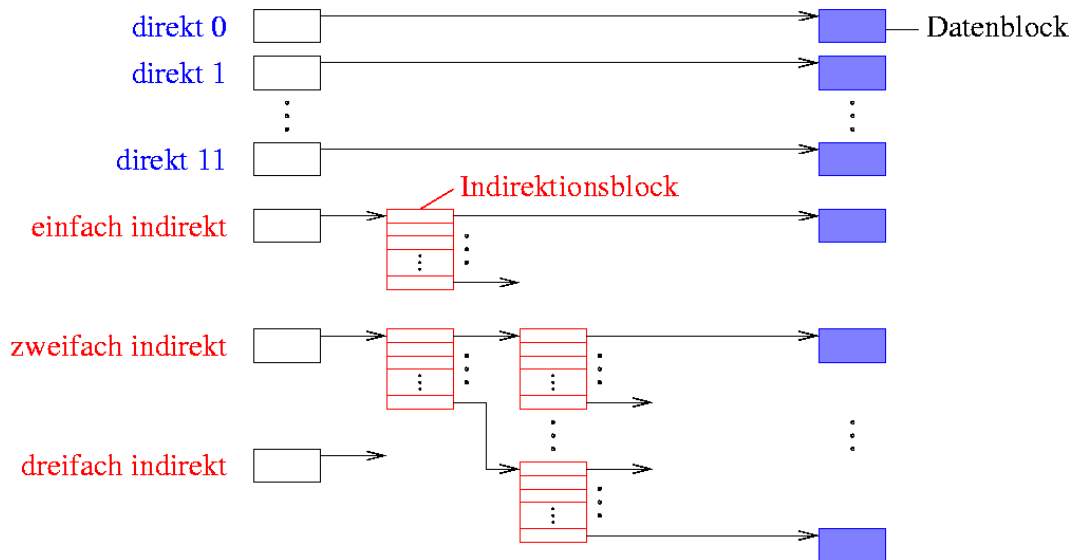
- Aufbau eines **Verzeichnisses** in EXT2:
 - Speicherung von (unsortierten) verketteten Einträgen in einer speziellen Datei, die aus einem oder mehreren Plattenblöcken bestehen kann
 - jeder Eintrag variabler Länge enthält:
 - Inode-Nummer
 - Offset zum nächsten Verzeichniseintrag (in Byte)
 - Länge des Dateinamen (in Byte)
 - Dateinamen (bis zu 255 Zeichen, abgeschlossen mit einem Nullbyte)



- Wurzelverzeichnis hat i.a. die Inode-Nummer 2

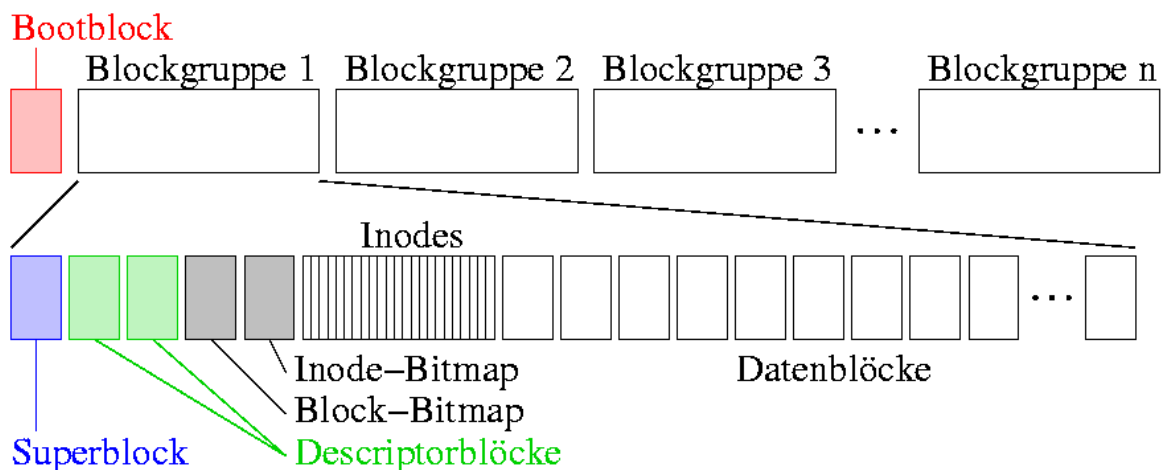
6.1 Linux-Dateisystem EXT2 (10)

- Möglichkeiten zur **Adressierung** der Dateiblöcke in EXT2:
 - 12 **direkte** Blockadressen in jedem Inode
 - je 1 einfach, zweifach und dreifach **indirekte** Blockadresse in jedem Inode



6.1 Linux-Dateisystem EXT2 (11)

- Blockorganisation in einer Partition:



- Aufteilung in mehrere gleich große **Blockgruppen**
(zur Reduktion der Distanz zwischen Inodes und zugehörigen Datenblöcken
⇒ schnellerer Plattenzugriff)

6.1 Linux-Dateisystem EXT2 (12)

- **Bootblock** enthält den *Bootloader* (zum Start des Betriebssystems)
- **Superblock** enthält wichtige Angaben über Layout des Dateisystems, z.B. Anzahl der Inodes, Anzahl der Plattenblöcke, Blockgröße (ist zur Sicherheit in jeder Blockgruppe als Kopie vorhanden!)
- **Gruppendeskriptoren** enthalten u.a. Informationen über Position der Bitmaps für Blöcke und Inodes sowie über Anzahl freier Blöcke und freier Inodes in jeder Blockgruppe (Anzahl der Inodes wird beim Erzeugen des Dateisystems festgelegt)
- das **Block-Bitmap** belegt einen Block und kennzeichnet die freien Blöcke in einer Blockgruppe (max. 8192 Einträge bei 1 kByte Blöcken)
- das **Inode-Bitmap** belegt einen Block und kennzeichnet die freien Inodes in einer Blockgruppe (max. 8192 Einträge bei 1 kByte Blöcken)
- sehr große Dateien sind über mehrere Blockgruppen verteilt!

6.1 Linux-Dateisystem EXT2 (13)

- einige weitere Eigenschaften:
 - eine wählbare Anzahl von Dateiblöcken (i.a. 5 %) bleibt für Superuser reserviert (⇒ zur Erhaltung der Systemstabilität bei einer voll geschriebenen Platte)
 - **Konsistenz** des Dateisystems kann mittels `fsck` überprüft werden; in vielen Fällen ist bei Inkonsistenz eine automatische Reparatur möglich
 - Flag im Superblock gibt **Status** (*clean* / *not clean*) des Dateisystems an (⇒ erspart extrem zeitaufwendigen Konsistenzcheck beim Booten)
 - EXT3 besitzt zusätzlich ein **Journaling**:
 - in einer *Journal*-Datei werden sämtliche Änderungen am Dateisystem **vor** ihrer Durchführung protokolliert (und nach erfolgreicher Durchführung wieder entfernt)
 - nach einem Systemabsturz müssen nur die in der *Journal*-Datei enthaltenen Änderungen nachvollzogen und ggf. die Konsistenz der betroffenen Dateien überprüft werden

6.2 FAT32 Dateisystem

- Dateisystem für **Windows 98**
- einige Unterschiede zum Linux-Dateisystem EXT2:
 - **keine Benutzeridentifikation** für Dateien und Verzeichnisse!
 - Partitionen werden durch Laufwerke repräsentiert, die durch Buchstaben dargestellt werden, z.B.: A: (Floppy), C: (Platte), D: (DVD)
 - jedem Windows-Programm ist ein **aktuelles Laufwerk** und ein **aktuelles Verzeichnis** aus Dateibaum zugeordnet
 - es gibt keine Inodes: die Speicherung aller Attribute einer Datei erfolgt im Verzeichnis
 - es gibt keine *Hard Links*
 - die kleinste adressierbare Einheit heißt **Cluster** und ist ein Block mit einer Zweierpotenz von 1 bis 128 Sektoren (bei Formatierung wählbar)
 - die Blockadressierung erfolgt über eine Tabelle (**FAT** = *File Allocation Table*), in der die Verkettung der Cluster aller Dateien gespeichert ist

6.2 FAT32 Dateisystem (2)

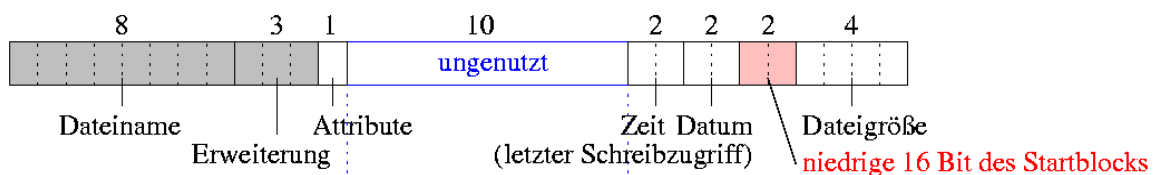
- Attribute einer **FAT32-Datei**:
 - **Name**
 - im MS-DOS Modus: 8 Zeichen Name + 3 Zeichen Erweiterung (z.B.: „AUTOEXEC.BAT“)
 - im Windows 98 Modus: 255 Zeichen inklusive Sonderzeichen (z.B.: „Eigene Dateien“)
 - **Dateilänge**
 - **Typ**: Verzeichnis, versteckte Datei (*hidden*), Systemdatei (*system*), zu archivierende Datei (*archive*)
 - nur **zwei Zugriffsrechte**: „nur lesbar“ und „schreib- und lesbar“
 - **Ortsinformation**: Nummer des ersten Clusters einer Datei
 - **Zeitstempel**:
 - zunächst nur Datum und Uhrzeit des letzten Schreibzugriffs
 - bei Windows 98 zusätzlich Datum und Uhrzeit der Erstellung sowie Datum des letzten Lesezugriffs

6.2 FAT32 Dateisystem (3)

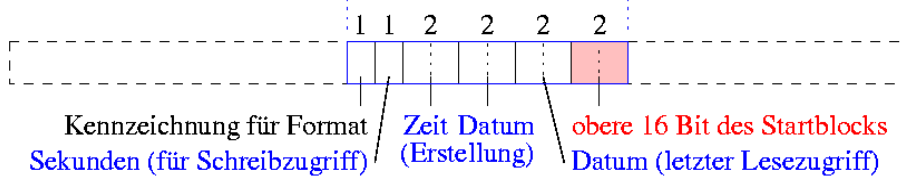
- Aufbau eines **FAT32-Verzeichnis**:

- unsortierte 32-Byte Einträge werden hintereinander in Liste gespeichert

ursprünglicher Eintrag (MS-DOS mit FAT16):



erweiterter Eintrag (Windows 98 mit FAT32):



- aus langen Dateinamen (bis zu 255 Zeichen) wird ein neuer eindeutiger Name aus 8+3 Zeichen generiert und eingetragen; der vollständige Name wird in zusätzlichen vorangestellten 32-Byte Feldern gespeichert

6.2 FAT32 Dateisystem (4)

- die **Verkettung** der Cluster wird in der **FAT** festgehalten:

- enthält Eintrag für jedes Cluster auf der Festplatte
- für jedes Cluster einer Datei ist die Nummer des nachfolgenden Clusters als 32-Bit Zahl eingetragen
 - die Nummer des Startblocks kann dem Verzeichnis entnommen werden
 - Dateiende wird durch Eintrag -1 markiert
- freie Cluster werden durch Eintrag 0 markiert

Auszug aus einer FAT:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	...
-	-	0	8	5	6	20	0	9	15	11	17	0	0	0	16	18	4	-1	0	-1	0	...

Plattenblöcke (Cluster) für Datei A:

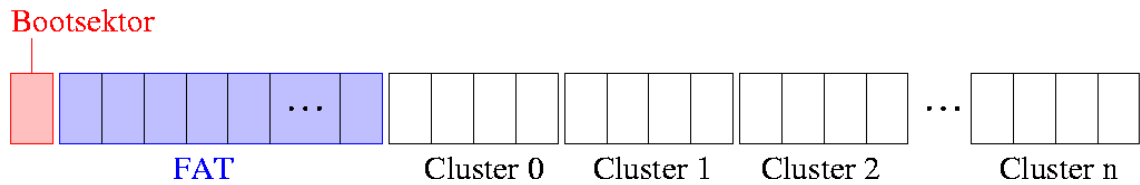
10	11	17	4	5	6	20
----	----	----	---	---	---	----

Plattenblöcke (Cluster) für Datei B:

3	8	9	15	16	18
---	---	---	----	----	----

6.2 FAT32 Dateisystem (5)

- Blockorganisation einer Partition mit FAT32:



- **Bootsektor** enthält neben dem *Bootloader* noch einige Angaben über das Dateisystem, z.B.:
 - Gesamtanzahl der Sektoren (4 Byte)
 - Bytes je Sektor (2 Byte)
 - Sektoren je Cluster (1 Byte, nur Zweierpotenzen von 1 bis 128 erlaubt)
 - Startposition des Hauptverzeichnisses (4 Byte)
 - Label (10 Byte) und Serien-Nummer (4 Byte)
 - Anzahl FATs (1 Byte) und Sektoren je FAT (4 Byte)
- **FAT** kann zur Erhöhung der Sicherheit auch mehrfach auf Festplatte gespeichert sein

6.2 FAT32 Dateisystem (6)

- einige **Nachteile** von FAT32:
 - **umständliche** Datenstrukturen (wegen Kompatibilität zu MS-DOS)
 - **sehr große FAT** bei modernen Festplatten hoher Kapazität
 - Positionieren eines Dateizeigers bei großen Dateien sehr zeitaufwendig
 - für jeden Dateizugriff muss mindestens ein Plattenblock mit einem Teil der FAT von der Festplatte geladen werden
 - FAT enthält Verkettungen für alle Dateien $\Rightarrow$ es werden stets auch viele nicht benötigte Verkettungsinformationen geladen
 - langsame Suche nach freien Clustern
 - **sehr viele Kopfbewegungen**, wenn Cluster einer Datei verstreut sind ($\Rightarrow$ regelmäßiger Aufruf eines Defragmentierungsprogramms sinnvoll; es versucht die Cluster jeder Datei zusammenhängend anzuordnen)
- FAT32 wird nicht mehr weiterentwickelt!

6.3 NTFS Dateisystem

- Dateisystem für **Windows NT**, optional auch für **Windows XP**
- einige Unterschiede zum FAT32 Dateisystem:
 - Unterstützung mehrerer **Benutzer** und **Gruppen** mit umfangreichen Zugriffsrechten
 - jede Partition wird als **Volume** bezeichnet und besteht aus einer linearen Sequenz von **Clustern** (mit z. Zt. 512, 1024, 2048 oder 4096 Byte)
 - Adressierung eines Clusters erfolgt über **64-Bit Cluster-Nummern** (⇒ sehr große Dateien möglich)
 - zentrales Element der Dateioorganisation ist die *Master File Table* (**MFT**), die für jede Datei einen Eintrag enthält
 - Unterstützung von *Hard Links*
 - Dateien können automatisch komprimiert abgespeichert werden
 - Konsistenzüberprüfung mittels Journal-Datei

6.3 NTFS Dateisystem (2)

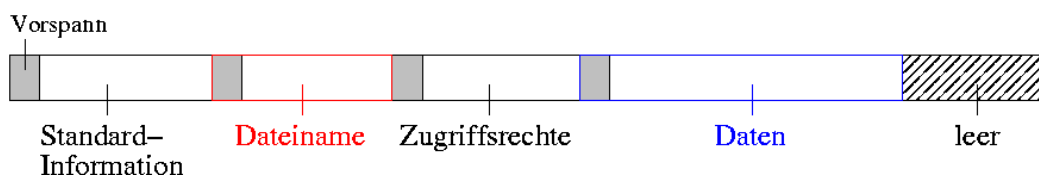
- Zugriffsrechte einer **NTFS-Datei**:
 - *no access* : kein Zugriff (---)
 - *list* : Anzeige von Verzeichnisinhalt erlaubt (r--)
 - *read* : Lesen und Ausführen von Dateien erlaubt (rw-)
 - *add* : Hinzufügen von Einträgen in einem Verzeichnis erlaubt (-wx)
 - *change* : Ändern und Löschen von Dateien erlaubt (rwx)
 - *full* : zusätzlich Ändern von Eigentümer und Zugriffsrechten erlaubt
- jede Datei wird eindeutig durch eine **64-Bit Dateireferenz** (*File reference*) bezeichnet; sie besteht aus:
 - **48-Bit Dateinummer** (*File ID*), die einen eindeutigen Index in der MFT darstellt
 - **16-Bit Folgenummer** (*Sequence ID*), die bei jeder Wiederverwendung der Dateinummer hochgezählt wird

6.3 NTFS Dateisystem (3)

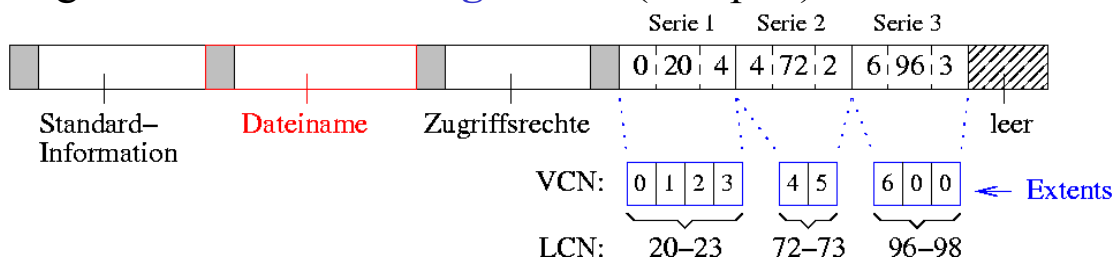
- jede Datei besteht aus mehreren **Strömen**, z.B.:
 - **Standard-Information** (zu MS-DOS kompatibler Dateiname sowie klassische MS-DOS Attribute wie Dateilänge, Zeitstempel, Typ, ...)
 - **Dateiname** (in Unicode mit 16-Bit Zeichen)
 - **Dateireferenz** (64-Bit Wert)
 - **Sicherheits-Beschreibung** (enthält **Eigentümer** und **Zugriffsrechte**)
 - **eigentliche Daten**
- Dateiorganisation erfolgt mittels *Master File Table* (**MFT**):
 - enthält **für jede Datei genau einen Eintrag**
 - Größe jedes Eintrags entspricht der Cluster-Größe
 - **Index** in Tabelle wird durch die **Datei-Nummer** festgelegt
 - Eintrag in Bootsektor verweist auf Beginn der MFT
 - ein Eintrag besteht aus **Hintereinanderreihung mehrerer Ströme**, die jeweils durch einen kurzen Vorspann (mit Länge, ...) eingeleitet werden

6.3 NTFS Dateisystem (4)

- Eintrag in MFT für eine **kurze Datei**:



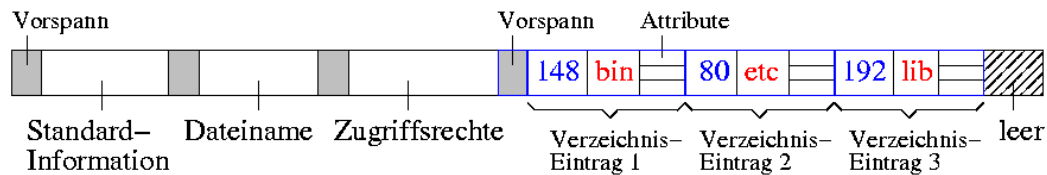
- Eintrag in MFT für eine **lange Datei** (Beispiel):



- Daten befinden sich in Serien aus zusammengehörigen Clustern (**Extents**)
- Zuordnung von virtuellen Cluster-Nummern (VCN) zu logischen Cluster-Nummern (LCN) wird als weiterer Strom gespeichert

6.3 NTFS Dateisystem (5)

- Eintrag in MFT für ein **kurzes Verzeichnis** (Beispiel):



- Inhalt eines Verzeichnisses wird als eigener Strom gespeichert
 - jeder Verzeichniseintrag enthält **Dateireferenz**, **Dateiname** und einige ausgewählte **Attribute** (z.B. Dateilänge, Datum der letzten Modifikation)
 - Sortierung in lexikographischer Reihenfolge
- Eintrag in MFT für ein **langes Verzeichnis**:
 - Verzeichniseinträge werden nicht in MFT, sondern in separaten **Extents** gespeichert
 - Organisation als **B⁺-Baum** ermöglicht eine schnelle Suche in großen Verzeichnissen

6.3 NTFS Dateisystem (6)

- die ersten 16 Dateien in der MFT sind **Metadateien**, die für das System reserviert sind:

Index	Bedeutung
0	MFT
1	Kopie der MFT
2	Journal-Datei (protokolliert die Änderungen am Dateisystem)
3	Volume-Informationen (z.B. Name, Größe des Volumes)
4	Attribut-Tabelle (definiert erlaubte Ströme in den Einträgen)
5	Wurzelverzeichnis
6	Cluster-Bitmap (kennzeichnet alle freien und belegten Cluster)
7	<i>Bootloader</i>
8	<i>Bad Cluster List</i> (enthält die Indizes aller fehlerhaften Cluster)
9 bis 15	... (reserviert für weitere Systemdateien)
16	erste Benutzerdatei

7 Lernziele

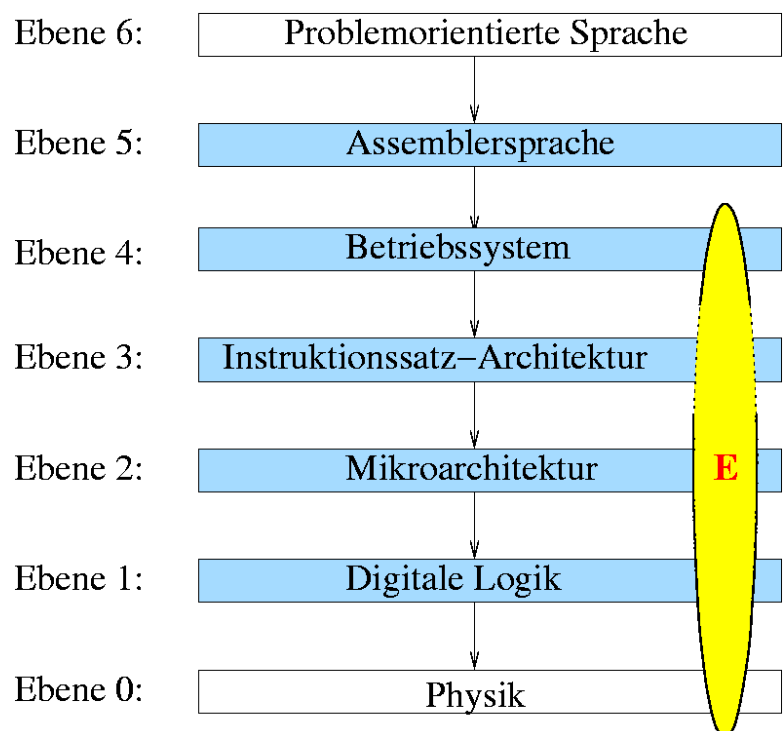
- **Begriffe:** Zone Bit Recording, Hostadapter, Inode, Bitmap, Symbolic/Hard Link, Journaling, FAT, MFT, ...
- **Aufbau einer Festplatte:**
 - Organisation der Daten in Zylinder, Spuren und Sektoren
 - physikalische/logische Adressierung von Sektoren
 - Leistungsmerkmale und ihre Auswirkung beim Plattenzugriff
 - Unterschiede zu CD-ROM, DVD und Floppy
- Verständnis der Abläufe auf einem **Festplattenbus** (IDE, SCSI)
- Arbeitsweise und Vorteile von RAID-Systemen
- prinzipielle Arbeitsweise von **Dateisystemen** am Beispiel von **EXT2, FAT32** und **NTFS**, z.B.:
 - wie werden die Blöcke einer Datei adressiert?
 - wie erfolgt der Zugriff auf eine Datei mit gegebenem Pfadnamen?
 - wie werden freie Blöcke ermittelt?

E Datenkommunikation

1. Netzwerke
2. ISO/OSI-Schichtenmodell
3. Kopplung von Netzwerken
4. Ethernet
5. TCP/IP

E Datenkommunikation

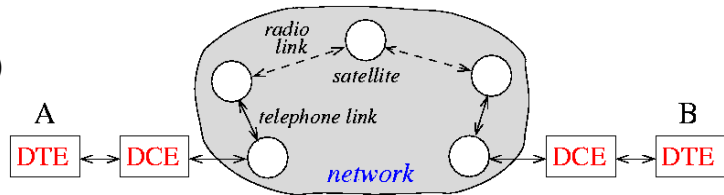
- Einordnung in das Schichtenmodell:



1 Netzwerke

- um Daten von Computer A im Ort x zu Computer B in Ort y zu übertragen, benötigt man:

- Rechner mit E/A-Schnittstellen (**DTE**)
- **Netzwerk**
(Kabel oder Funkstrecken)
- Datenübertragungseinrichtung (**DCE**)



- zudem muß in **standardisierten Protokollen** festgelegt werden, wie die Rechner miteinander kommunizieren, z.B.:
 - Adressierung des Zielrechners
 - Definition physikalischer Größen (Spannungspegel, Bitreihenfolge, ...)
 - Aufbau, Größe und Inhalt von Paketen
 - Fehlererkennung und -behandlung

1 Netzwerke (2)

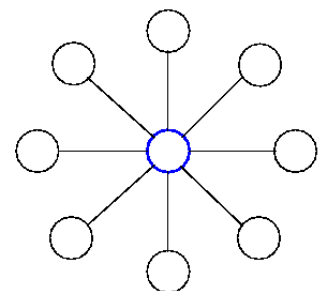
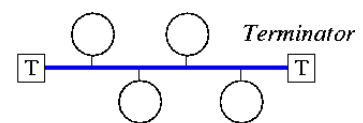
- Arten der Kommunikation über ein Netzwerk:
 - **Leitungsvermittlung** (*Circuit Switching*):
 - vor jeder Datenübertragung wird eine durchgehende physikalische Verbindung explizit aufgebaut
 - Zwischenknoten arbeiten als einfache Schaltelemente.
 - alle zu übermittelnden Daten nehmen den gleichen Weg.
 - **Nachrichtenvermittlung** (*Message Switching*):
 - jede Nachricht wird vollständig zum nächsten Zwischenknoten transportiert, dort gespeichert und erst weitergesendet, wenn nächste Übertragungsstrecke frei ist (*Store and Forward*)
 - **Paketvermittlung** (*Packet Switching*):
 - jede Nachricht wird in mehreren Paketen übertragen, die zusätzliche Adress- und Steuerinformationen enthalten
 - Pakete können unterschiedliche Wege nehmen und somit auch unterschiedliche Laufzeiten aufweisen

1.1 Arten von Netzwerken

- **LAN** (*Local Area Network*): Abstand der Rechner untereinander max. einige hundert Meter; in privater Hand
Beispiele für verschiedene LAN-Technologien: Ethernet (IEEE 802.3), Token Ring (IEEE 802.5), Wireless LAN (IEEE 802.11)
- **MAN** (*Metropolitan Area Network*): auf eine bestimmte Region (z.B. eine Stadt) beschränkt
- **WAN** (*Wide Area Network*): verbindet mehrere LANs und MANs; kann sich über mehrere Länder und Kontinente erstrecken; Benutzung öffentlicher Datenwege
Beispiele: Telefonnetzwerk (*Public Switched Telephone Network*, PSTN), ARPANET (*Advanced Research Projects Agency Network*, 1969-1990), Internet, Datex-P (X.25), BelWü, DFN Gigabit Wissenschaftsnetz (G-WiN)

1.2 Netzwerktopologien

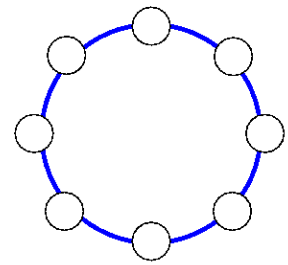
- **Bus**
 - gemeinsam genutzte Verbindungsleitungen
 - dezentrale Steuerung
 - **Vorteile**: einfache Realisierung und Erweiterbarkeit
 - **Nachteile**: Leitungslänge begrenzt, Zugriffskonflikte
- **Stern**
 - nur Punkt-zu-Punkt Verbindungen zu einem **zentralen Vermittlungsknoten**
 - zentrale Steuerung
 - **Vorteile**: einfache Realisierung, Ausfall einer Leitung hat nur lokale Folgen
 - **Nachteile**: hohe Leistungs- und Zuverlässigkeitsanforderungen an Zentralknoten



1.2 Netzwerktopologien (2)

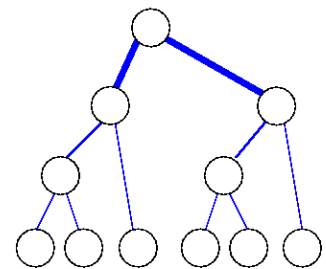
- **Ring**

- jeder Knoten ist mit genau zwei Nachbarknoten verbunden
- **Vorteil:** einfache dezentrale Steuerung mittels zyklischer Weitergabe eines „Token“ (nur der Knoten darf senden, der gerade das *Token* hat)
- **Nachteile:** Knoten- oder Leitungsausfall bewirkt Totalausfall des Netzwerks



- **Baum**

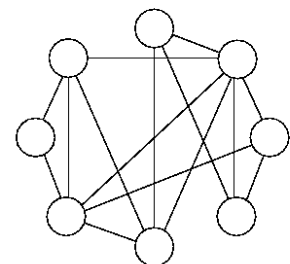
- genau ein Pfad zwischen Sender und Empfänger
- **Vorteile:** mehrere Datentransporte gleichzeitig möglich
- **Nachteile:** Anforderung an Zuverlässigkeit und Leistung der Knoten und Leitungen wächst im Baum von unten nach oben



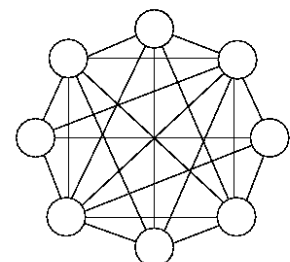
1.2 Netzwerktopologien (3)

- **Maschennetz**

- jeder Knoten ist mit mindestens zwei, i.a. jedoch mit mehreren (oder allen) anderen Knoten direkt verbunden
- **Vorteile:**
 - hohe Ausfallsicherheit bei hohem Vermaschungsgrad
 - hohe Leistung bei paralleler Nutzung mehrerer Verbindungen
- **Nachteile:**
 - hoher Aufwand und somit hohe Kosten
 - aufwendige Steuerung



teilweise Vermaschung

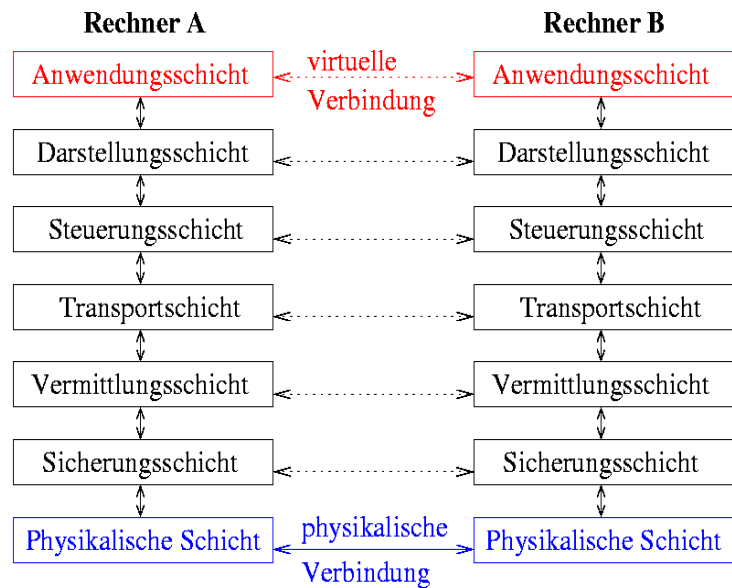


vollständige Vermaschung

2 ISO/OSI Schichtenmodell

- **ISO Referenzmodell für OSI (1983)**

(Intern. Standardization Organization reference model for Open Systems Interconnection)



- verschiedene Stufen der **Abstraktion** und verschiedene **Funktionalität** auf jeder Schicht, Kommunikation nur zwischen benachbarten Schichten
- Hilfsmittel für Protokollentwurf, legt aber keine Protokolle explizit fest

2.1 Physikalische Schicht (*Physical Layer*)

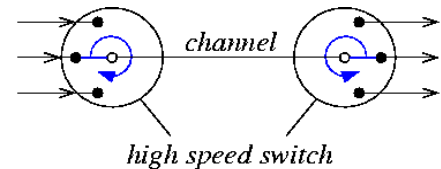
- **ungesicherte** Übertragung von **Bit-Sequenzen** von Knoten A eines Netzwerks zu Knoten B über ein Übertragungsmedium
Übertragungsfehler z.B. durch Rauschen (*Noise*), Übersprechen (*Cross-Talk*), Echo, Störimpulse
- Definition von **physikalischen Größen**, z.B.:
Spannungspegel, Steuersignale, Taktfrequenz, Steckerbelegungen, Kennung von Anfang und Ende einer Bit-Sequenz
- diverse Übertragungsmedien:

	max. Distanz	max. Bitrate
verdrilltes, zweiadriges Kabel („ <i>Twisted Pair</i> “)	100 m	100 MBit/s
Koaxialkabel	3 km	800 MBit/s
Glasfaserkabel	30 km	2 GBit/s
Infrarot (IrDA)	1 m	4 MBit/s
Wireless LAN (WLAN)	100 m	11 MBit/s
Richtfunk	10 km	622 MBit/s
Funkverbindung via Satellit	10000 km	2 GBit/s

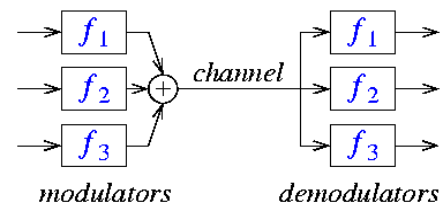
2.1 Physikalische Schicht (2)

- verschiedene Betriebsarten eines Übertragungskanals:
 - **simplex**: Signale werden nur in eine Richtung übertragen
 - **halb-duplex**: Signale können zu einem Zeitpunkt alternativ in einer von beiden Richtungen übertragen werden
 - **voll-duplex**: Signale werden gleichzeitig in beide Richtungen übertragen
- zur Erhöhung der Auslastung eines Kanals können eingesetzt werden:

- **Zeitmultiplex**-Verfahren
(Time Division Multiplexing, TDM)



- **Frequenzmultiplex**-Verfahren
(Frequency Division Multiplexing, FDM)



2.1 Physikalische Schicht (3)

- **Signal-/Rauschverhältnis S/N** gibt das Verhältnis von Signalleistung S zu Rauschleistung N an
(wird i.a. logarithmisch ausgedrückt in dB: $10 \log_{10} S/N$)
- **max. Bitrate r** auf Übertragungskanal mit Bandbreite b :
 - bei L verschiedenen Signalpegeln ohne Rauschen: $r = 2 \cdot b \cdot \log_2 L$
 - bei beliebig vielen Signalpegeln und einem Signal-/Rauschverhältnis S/N (theoretische Kanalkapazität nach Shannon): $r = b \cdot \log_2 (1 + S/N)$
- Beispiel: Datenübertragung über Telefonnetz
 - analoge Übertragung im Frequenzbereich 300 bis 3300 Hz
 $\Rightarrow$ Bandbreite 3000 Hz
 - max. Bitrate bei Übertragung eines reinen Binärsignals: **6000 Bit/s**
 - max. Bitrate bei Verwendung vieler Signalpegel und einem typischen Signal-/Rauschverhältnis von 30 dB (d.h. $S/N = 1000$): **29900 Bit/s**

2.1 Physikalische Schicht (4)

- Beispiel: Datenübertragung über Telefonnetz (Forts.):
 - ein analoger Telefonkanal kann nur Sinusfrequenzen innerhalb der Kanalbandbreite, aber keine digitalen Pulse übertragen
 - Transformation der digitalen Signale mittels **Modem** (Modulator / Demodulator) in analoge Signale erforderlich; verschiedene Verfahren:
 - **Amplitudenmodulation** (AM):
feste Frequenz f , Amplituden a_0 und a_1 zum Senden von 0 / 1
 - **Frequenzmodulation** (FM):
Frequenzen f_0 und f_1 zum Senden von 0 / 1, feste Amplitude a
 - **Phasenmodulation** (PM):
Phasen φ_0 und φ_1 zum Senden von 0 / 1, Frequenz und Amplitude fest
 - **Phasendifferenzmodulation** (PSK):
 2^k Phasen φ_i zum Senden von 2^k möglichen k -Bit Sequenzen
 - **Quadraturamplitudenmodulation** (QAM):
Kombination von PSK mit 2^k Phasen und AM mit q Amplituden

2.1 Physikalische Schicht (5)

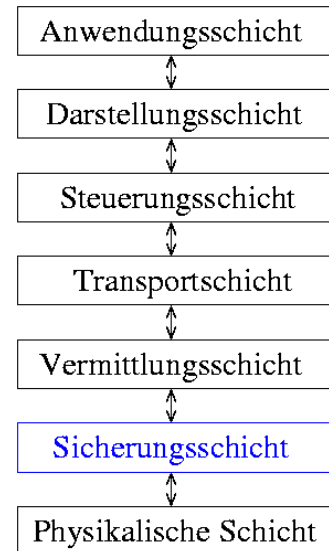
- Standards zur Datenübertragung über das Telefonnetz:

	max. Bitrate	Verfahren
V.21	300	FM ($f_0 = 1180$ Hz, $f_1 = 980$ Hz)
V.22	1200	PSK ($k = 2$)
V.32	9600	QAM ($k = 4$, $q = 2$)
V.34	33.6k	QAM (960 Phasen/Amplituden-Kombinationen)
V.90	33.6k $\uparrow$ / 56k $\downarrow$	wie V.34 $\uparrow$ / quasi-digital $\downarrow$
ISDN	je Kanal 64k	2 Kanäle, digital
ADSL	640k $\uparrow$ / 8M $\downarrow$	Nutzung einer Bandbreite von 1.1 MHz

- **ADSL** (*Asymmetric Digital Subscriber Line*) verwendet auf der Leitung vom Endbenutzer zum 1. Netzknoten (< 5 km) Bandbreiten von 100 kHz ($\uparrow$, *upload*) und 1 MHz ($\downarrow$, *download*) oberhalb des für Sprache verwendeten Frequenzbereichs

2.2 Sicherungsschicht (*Data Link Layer*)

- **fehlerfreie** Übertragung einzelner **Pakete** zwischen zwei direkt verbundenen Netzknoten
- Paketsynchronisation durch **Protokolle**:
 - Definition spezieller Anfangs- und Ende-Zeichen für Pakete
 - Bestätigungen für empfangene Paket
- **Fehlererkennung** für jedes Paket
- **Fehlerkorrektur** für jedes Paket (z.B. durch wiederholtes Senden von fehlerhaft oder nicht empfangenen Paketen)



2.2 Sicherungsschicht (2)

- Beispiel 1: **BiSync**, ein byte-orientiertes Protokoll
 - **ASCII** (*American Standard Code for Information Interchange*) enthält in einer 7-Bit Kodierung neben Buchstaben, Ziffern und Symbolen auch einige **Sonderzeichen** zur Steuerung einer Kommunikation:

		Bits 6–4 →							
		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
Bits 3–0 ↓	0 0000	NULL	DCL	SP	0	@	P	'	p
	1 0001	SOH	DC1	!	1	A	Q	a	q
	2 0010	STX	DC2	"	2	B	R	b	r
	3 0011	ETX	DC3	#	3	C	S	c	s
	4 0100	EOT	DC4	\$	4	D	T	d	t
	5 0101	ENQ	NAK	%	5	E	U	e	u
	6 0110	ACK	SYN	&	6	F	V	f	v
	7 0111	BEL	ETB	'	7	G	W	g	w
	8 1000	BS	CAN	(	8	H	X	h	x
	9 1001	HT	EM	)	9	I	Y	i	y
	A 1010	LF	SUB	*	:	J	Z	j	z
	B 1011	VT	ESC	+	;	K	[	k	}
	C 1100	FF	FS	,	<	L	\	l	
	D 1101	CR	GS	-	=	M	]	m	}
	E 1110	SO	RS	.	>	N	^	n	~
	F 1111	SI	US	/	?	O	_	o	DEL

2.2 Sicherungsschicht (3)

- Beispiel 1 (Forts.):

- BiSync benötigt folgende Sonderzeichen:

SYN (Synchronize)

SOH (Start of Header)

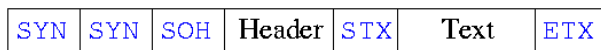
STX/ETX (Start / End of Text)

ACK/NAK (Acknowledge / No Acknowledge)

ENQ (Enquiry)

EOT (End of Transmission)

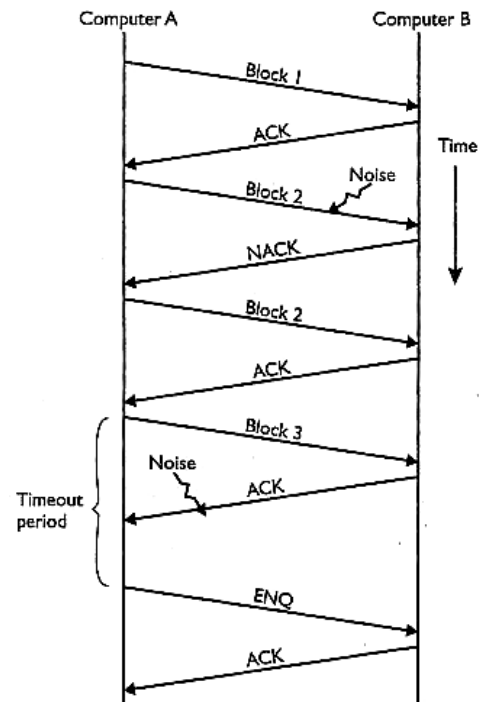
- allgemeiner Aufbau eines Paketes:



z.B.:

address	sequence number	control
---------	-----------------	---------

- ungeeignet zur Übertragung von Bitströmen beliebiger Art !



2.2 Sicherungsschicht (4)

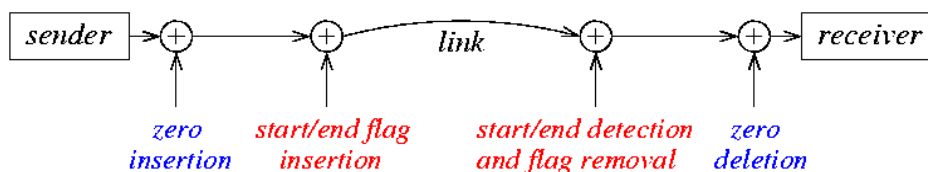
- Beispiel 2: **HDLC** (High-Level Data Link Control), ein bit-orientiertes Protokoll

- Übertragung von Bitströmen erfolgt in Paketen, die mit der 8-Bit Kennung **01111110** beginnen und aufhören.

- folgen in den Daten fünf 1-Bits aufeinander, wird vom Sender ein **0-Bit** eingefügt (*Bit Stuffing*)

Beispiel: Übertragung der Bitfolge 011011111101111100 erfolgt im HDLC-Paket **01111110**0110111111**0**1011111**0**00**01111110**

- Empfänger löscht stets ein 0-Bit nach Erkennen von 11111

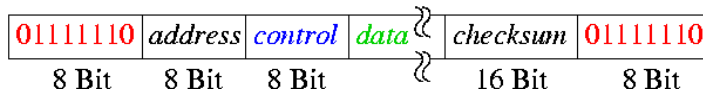


- strikte Trennung von Sender (initiiert Kommunikation) und Empfänger (darf nur auf Anforderung Antwortpakete senden)

2.2 Sicherungsschicht (5)

- Beispiel 2 (Forts.):

- allgemeiner Aufbau eines HDLC-Paketes:



- Ablauf wird gesteuert durch den Inhalt des Feldes *control* :

type	N (S)	N (R)	P/F
1 Bit	3 Bit	3 Bit	1 Bit

type : Paket enthält in N (s) eine Meldung, z.B.:

REJ = Zurückweisen aller Pakete seit N (R)

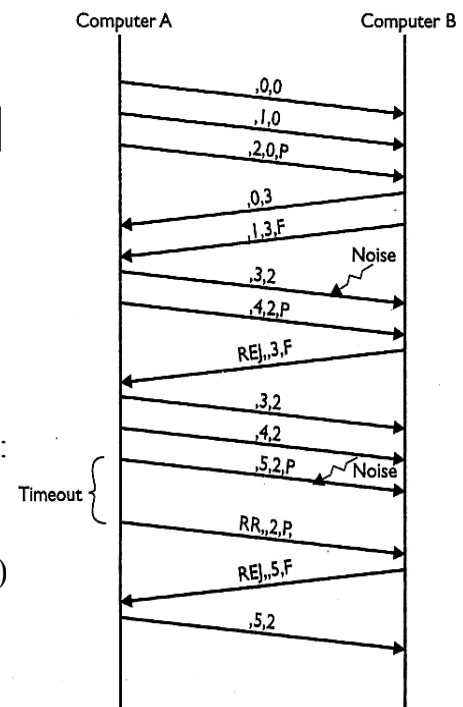
RR = empfangsbereit (*Receive Ready*)

N (S) : Nummer des gesendeten Paketes (aus 0–7)

N (R) : Nummer des nächsten erwarteten Paketes

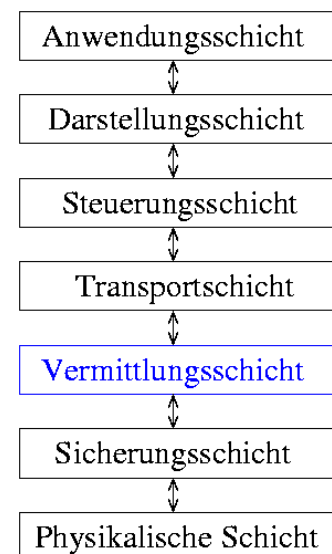
P/F : *Poll* = Sender bittet um Antwort

Final = Empfänger sendet letztes Paket



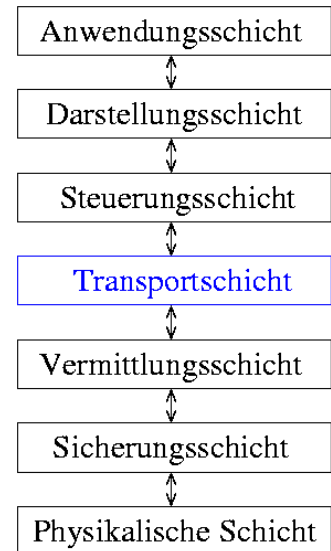
2.3 Vermittlungsschicht (Network Layer)

- Auswahl eines **Transportweges** (*Routing*) für die folgende Übertragungsstrecke
 - **statisch**: fest für alle Pakete einer Sitzung
 - **dynamisch**: für jedes Paket neu ermittelt
- Kopplung** von Subnetzen mit unterschiedlicher Technologie
(⇒ Flußsteuerung bei unterschiedlichen Bitraten)
- Erkennen und Beseitigen von **Überlast-situationen**
- ebenso wie physikalische und Sicherungsschicht i.a. direkt in **Hardware** realisiert
- Beispiele: Paketvermittlungsprotokoll **X.25**, Internet Protokoll **IP**



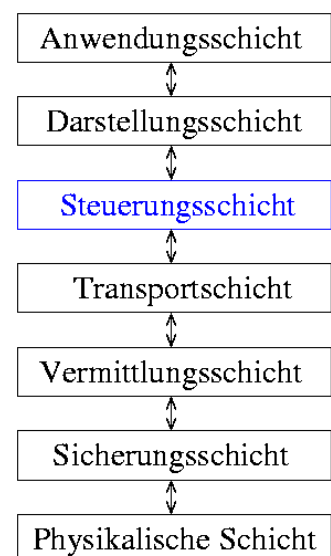
2.4 Transportschicht (*Transport Layer*)

- Aufbau einer vollständigen (virtuellen) **Verbindung zwischen zwei Endsystemen** für die Dauer einer Sitzung
- **Adressierung** des Endteilnehmers
- Sicherstellung, dass **gesamte Nachricht fehlerfrei** (als Strom oder in Paketen) übertragen wurde
- Definition von **Qualitätsparametern**
- Beispiel: TCP



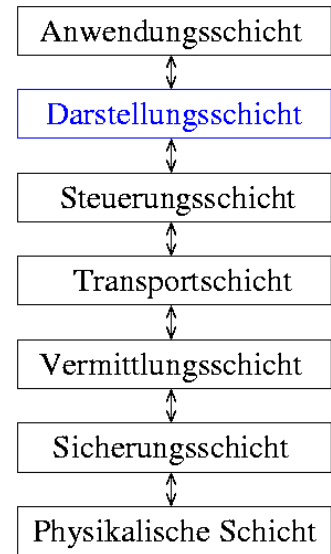
2.5 Steuerungsschicht (*Session Layer*)

- steuert **Dialog** zwischen 2 Endteilnehmern für eine Sitzung (z.B. das Rederecht bei halb-duplex Verbindungen)
- Festlegen von **Synchronisationspunkten** (*Check Points*) zum Wiederaufsetzen nach Zusammenbruch einer Verbindung
- automatischer **Wiederaufbau** einer Verbindung nach Zusammenbruch
- Aufteilung des Datenstroms auf mehrere Transportverbindungen zur Erhöhung des Durchsatzes
- **Kopplung** mehrerer Verbindungen (z.B. Audio/Video) in einer Sitzung



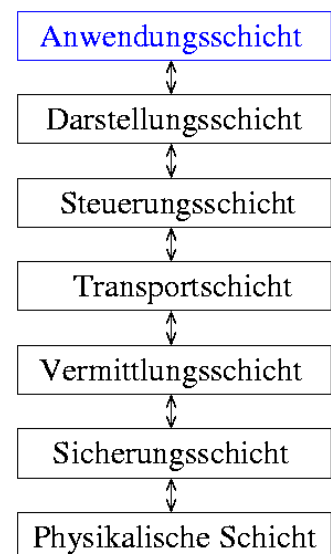
2.6 Darstellungsschicht (*Presentation Layer*)

- Wahl eines **standardisierten Datenformats** für die Übertragung von Daten (z.B. ISO-8859-x, ASCII, Unicode, XDR)
- Transformation aller Anwenderdaten in das gewählte Format auf Senderseite und Rücktransformation auf Empfängerseite
- ggf. **Verschlüsselung** der Daten bei sicherheitskritischen Anwendungen
- ggf. automatische **Komprimierung** der Daten und **Dekomprimierung** auf der Empfängerseite



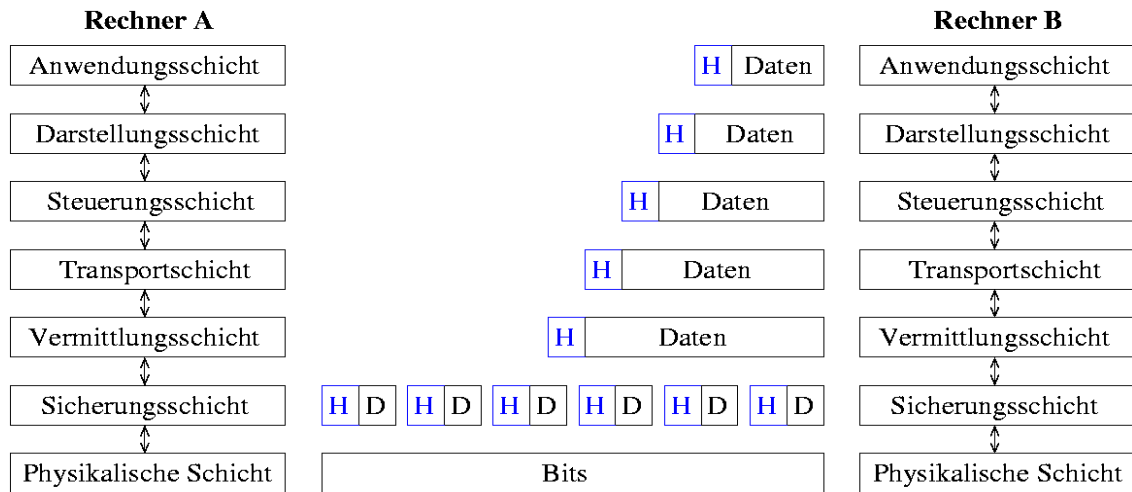
2.7 Anwendungsschicht (*Application Layer*)

- Schnittstelle zwischen Anwendungsprogramm und Kommunikationssystem
- definiert einige Protokolle für typische **Kommunikationsdienste**, z.B.
 - *Email* (SMTP, POP, IMAP)
 - *File Transfer* (FTAM, FTP)
 - *Virtual Terminal* (VT, Telnet)
 - *Remote Procedure Call* (RPC)
 - *Remote Database Access* (RDA)
 - *Hypertext Transfer Protocol* (HTTP)
 - *Domain Name Service* (DNS)
- dAnwendungsprogramme selbst gehören nicht zur Anwendungsschicht!



2.8 Datenübertragung im ISO/OSI Modell

- jede Schicht fügt den von ihr zu übertragenden Daten weitere **Steuerinformationen** (*Header*) hinzu, z.B. über Zieladresse, gewähltes Format, Paketgröße, ... :



- für einzelne Schichten kann der *Header* auch leer sein!

2.9 Bewertung ISO/OSI Schichtenmodell

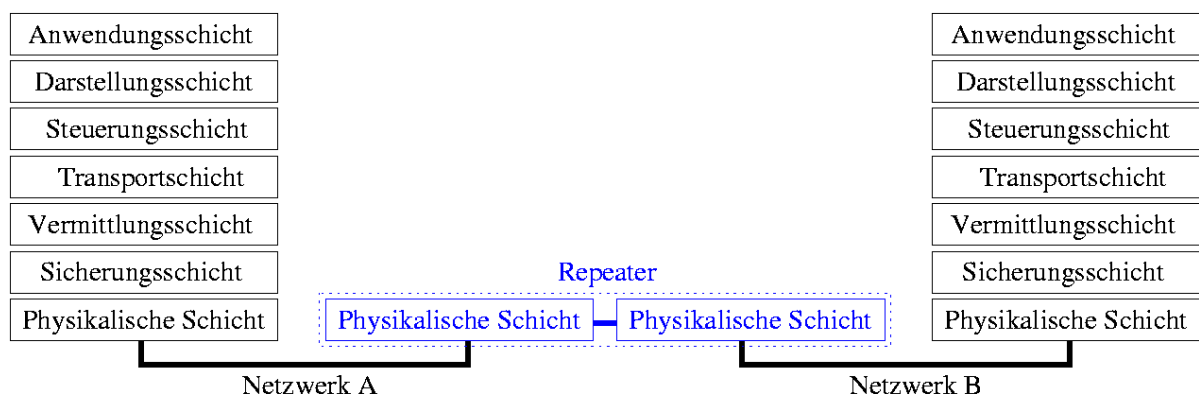
- i.a. nicht alle 7 Schichten in einer Implementierung vorhanden (eine vollständige Implementierung für alle Schichten wäre ineffizient)
- ISO/OSI Referenzmodell ist ein gutes **Hilfsmittel** zur
 - **Strukturierung** einer Datenkommunikation
 - **Einordnung** und **Bewertung** von Kommunikationsprotokollen(durch hierarchischen Aufbau und die klare Trennung der verschiedenen Aufgabenbereiche)
- bereits vor der ISO Standardisierung etablierte Quasi-Standards (vor allem **TCP/IP**) passen nicht gut in dieses Modell

3 Kopplung von Netzwerken

- **Kopplungseinheiten** dienen der Verbindung von Netzwerken
- verschiedene **Ziele** bei der Kopplung von Netzwerken, z.B.:
 - Überwindung der physikalisch bedingten maximalen Ausdehnung eines Netzwerks (z.B. aufgrund der Leitungslänge)
 - Erhöhung der Knotenzahl
 - logische Trennung von Netzwerksegmenten ($\Rightarrow$ Reduzierung der Netzlast)
- mögliche **Aufgaben** einer Kopplungseinheit sind:
 - Adreßumwandlung
 - Wegewahl (Routing)
 - Zusammensetzen von Paketen und Aufteilung in Pakete anderer Größe
 - Fehlerkontrolle
- Die Schicht im ISO/OSI-Referenzmodell, auf der eine Kopplungseinheit operiert, bestimmt den Funktionsumfang

3 Kopplung von Netzwerken (2)

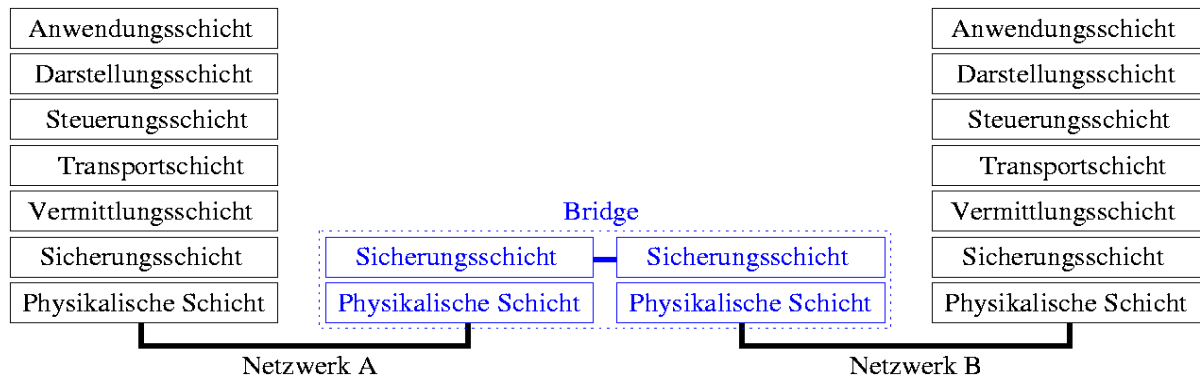
- ein **Repeater**
 - dient der reinen Signalverstärkung auf der **physikalischen Schicht**
 - leistet eine einfache Verbindung von zwei Netzwerksegmenten



- Protokolle aller darüberliegenden Schichten müssen gleich sein!
- ein **Hub** ist ein Repeater für mehr als zwei Netzwerkanschlüsse

3 Kopplung von Netzwerken (3)

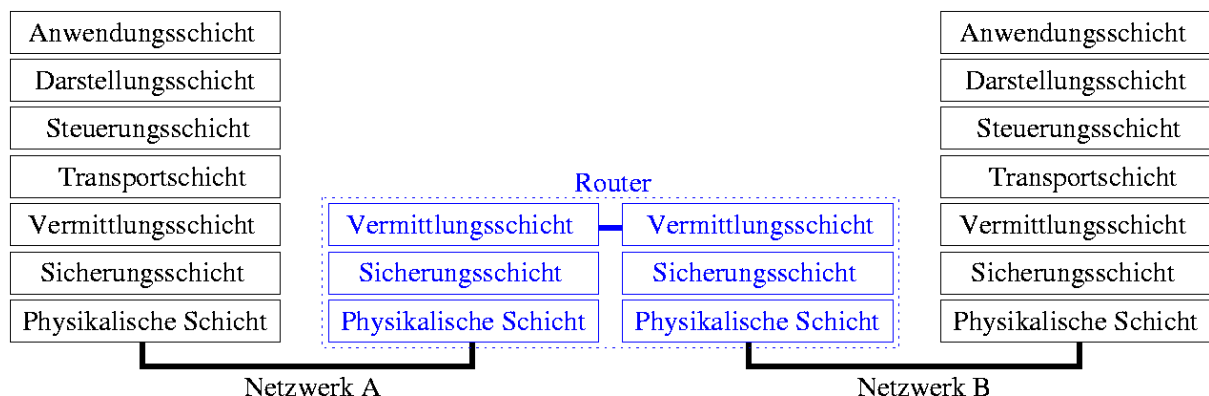
- eine **Brücke** (*Bridge*)
 - verbindet i.a. zwei Subnetzwerke auf der **Sicherungsschicht** ($\Rightarrow$ mit Fehlerbehandlung!)
 - leistet eine logische Trennung von Netzen: nur Pakete mit nichtlokalen Zieladressen werden in das angekoppelte Subnetzwerk übertragen



- ein **Switch** ist eine Brücke für mehr als zwei Netzwerkanschlüsse

3 Kopplung von Netzwerken (4)

- ein **Router**
 - verbindet Netzwerke mit unterschiedlichen Netzwerktechnologien und Netzwerkprotokollen
 - operiert auf **Vermittlungsschicht**: bei mehreren alternativen Pfaden kann Router einen Weg zum Zielknoten auswählen
 - wertet die in Paketen enthaltenen Adressangaben aus und legt interne **Routing-Tabellen** zur Bestimmung optimaler Wege an



4 Ethernet

- weltweit sehr verbreitete **LAN-Technologie**
- historische Entwicklung:
 - 1976 im Xerox Palo Alto Research Center entwickelt
 - 1980 erster Standard von Xerox, DEC und Intel
 - 1983 erster IEEE Standard 802.3
- leistet **ungesicherte Paketübertragung** (ohne Quittierung!) auf verschiedenen Übertragungsmedien
- verschiedene Standards:

Bezeichnung	Standard	Übertragungsmedium	Bitrate	Topologie
„ <i>Thickwire</i> “ Ethernet	10Base5	10 mm Ø Koaxialkabel	10 MBit/s	Bus (< 500m)
„ <i>Thinwire</i> “ Ethernet	10Base2	5 mm Ø Koaxialkabel	10 MBit/s	Bus (< 185m)
Ethernet	10BaseT	2 TP-Kabel (UTP)	10 MBit/s	Stern
Fast Ethernet	100BaseT	2 TP-Kabel (UTP)	100 MBit/s	Stern
Gigabit Ethernet	1000BaseT	4 abgeschirmte TP-Kabel (STP)	1000 MBit/s	Stern

4.1 Ethernet Protokoll

- Standard-Protokoll für Halb-Duplex Betrieb: **CSMA/CD**
 - 1) wenn ein Rechner A Daten senden möchte, prüft er zunächst, ob der Übertragungskanal frei ist (**CS** = *C*arrier *S*ense)
 - 2) sobald Kanal frei ist, beginnt Rechner A mit der Datenübertragung
 - 3) gleichzeitig hört Rechner den Datenkanal ab und vergleicht gesendetes Signal mit abgehörtem Signal
 - 4) hat ein anderer Rechner B fast gleichzeitig auch mit einer Übertragung begonnen (**MA** = *M*ultiple *A*ccess), kann Rechner A eine Kollision feststellen (**CD** = *C*ollision *D*etect)
- Verhalten im Falle einer **Kollision**:
 - 5) Rechner A bricht Übertragung ab und sendet spezielles „*Jamming*“-Signal, um die Ungültigkeit der Daten zu kennzeichnen
 - 6) Rechner A wartet eine zufällig gewählte Zeitspanne und beginnt Übertragung erneut mit Punkt 1)

4.1 Ethernet Protokoll (2)

- Aufbau eines **Ethernet-Paketes** (Größe: 64-1526 Bytes):

<i>preamble</i> (7 Byte)	10101011 (1 Start Byte)	<i>destination addr.</i> (6 Byte)	<i>source addr.</i> (6 Byte)	<i>length</i> (2 Byte)	<i>data</i> (38–1500 Byte)	<i>checksum</i> (4 Byte)
-----------------------------	----------------------------	--------------------------------------	---------------------------------	---------------------------	-------------------------------	-----------------------------

- *preamble* : 56-Bit Muster zur Synchronisation
(gibt dem Empfänger Zeit, ein Signal zu erkennen und Lesevorgang zu starten)
- *source address* : eindeutige physikalische 6-Byte MAC-Adresse des sendenden Rechners; in den ersten 3 Byte ist Hersteller/Modell kodiert
- *destination address* : eindeutige physikalische 6-Byte MAC-Adresse des Zielrechners oder Broadcast (alle 48 Bit auf 1 gesetzt)
- *length* : Länge des Datenfeldes (in Byte)
- *data* : zu übertragende Daten (max. 1500 Byte); ggf. werden Null-Bytes eingefügt, um eine minimale Paketlänge von 64 Byte zu erreichen
- *checksum* : 32-Bit Prüfsumme
($\Rightarrow$ mehrere Bitfehler können mit hoher Wahrscheinlichkeit erkannt und zum Teil auch korrigiert werden!)

4.2 Ethernet Timing

- **Signallaufzeit τ** (*propagation delay*)
 - maximale Zeit τ , die ein Paket benötigen darf, um von einem Ende eines Netzwerks zum anderen Ende transportiert zu werden
 - legt maximale Entfernung im Ethernet Netzwerk fest
- **Rundlaufzeit 2τ** (*round trip time*)
 - minimale Zeit für die Erkennung einer Kollision mit dem am weitesten entfernten Rechners im Netzwerk
 - wurde für 10/100 MBit/s Ethernet festgelegt als die Übertragungszeit für 512 Bit, also 51,2 μ s bzw. 5,12 μ s
 - hieraus resultiert eine minimale Paketlänge von 64 Byte
- **Wartezeit** (*interpacket gap*)
 - minimale Zeit, die ein Knoten vor dem Versenden eines weiteren Paketes warten muss
 - Übertragungszeit von 96 Bit, bei 10/100 MBit/s also 9,6 μ s bzw. 0,96 μ s
 - ermöglicht auch anderen Knoten, ein Paket zu senden ($\Rightarrow$ Fairness!)

5 Fallstudie: TCP/IP

- TCP/IP = Transmission Control Protocol / Internet Protocol
- ursprünglich vor über 30 Jahren im Rahmen des ARPANET entwickelt, heute **Standard-Protokolle** des **Internets**
- hohe Popularität durch **offene** Protokolle
(zuerst in jeder Unix-Implementierung, heute auf jeder Plattform verfügbar)
- Funktionalität entspricht in etwa den ISO/OSI-Schichten 3 und 4

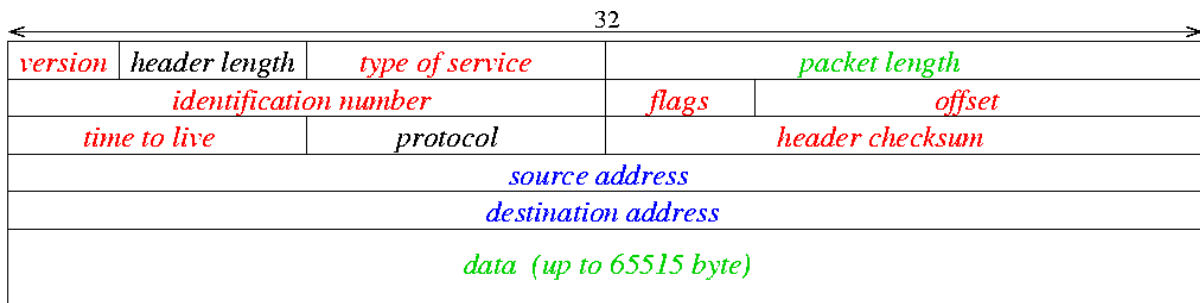
ISO/OSI	Anwendungsschicht	7	<i>application layer</i> email/smtp ftp http DNS telnet ...	Internet
	Darstellungsschicht	6		
	Steuerungsschicht	5		
	Transportschicht	4	TCPUDP	
	Vermittlungsschicht	3	IPARP	
	Sicherungsschicht	2	<i>network layer</i> Ethernet FDDI SLIP /PPP ...	
	Physikalische Schicht	1		

5.1 IP Protokoll

- Protokoll der Vermittlungsschicht für eine **paketvermittelte** Datenübertragung zwischen zwei Netzwerkknoten A und B
- von der Transportschicht übernommene Nachrichten werden ggf. in kleinere Pakete unterteilt und **unbestätigt** über ein Subnetzwerk übertragen
(⇒ keine gesicherte Übertragung, Verlust von Paketen ist möglich!)
- Pakete können über Netzwerke **unterschiedlicher Technologie** transportiert werden (i.a. mit unterschiedlichen Paketgrößen)
- **dynamisches Routing** mittels Routing-Tabellen
(Pakete einer Datenübertragung können unterschiedliche Wege nehmen)
- jeder Host und jeder Router im Internet verfügt über eine ihm zugeteilte **globale logische Adresse** (IP-Adresse)
(32-Bit Adressen in IP Version 4, 128-Bit Adressen in IP Version 6)

5.1 IP Protokoll (2)

- Aufbau eines **IPv4-Paketes**:



- *version* : IP Versionsnummer (z. Zt. IPv4)
- *type of service* : angeforderte Service-Qualität (z.B. *normal*, *priority*)
- *packet length* : Länge des Gesamt-Paketes (in Bytes, max. 65535)
- *identification number* : Datenübertragung, zu der ein Paket gehört
- *flags* und *offset* : Angaben zum Paket (Position, noch weitere Pakete ?)
- *time-to-live* : max. Zeit, die ein Paket im Netzwerk verbleiben darf
- *header checksum* : Prüfsumme, nur für Header!

5.2 IPv4 Adressen

- 5 verschiedene **Netzwerkklassen** bei IPv4-Adressen:

A	0	<i>network ID</i>		<i>host ID</i>		1.0.0.0 bis 126.255.255.255
B	1	0	<i>network ID</i>		<i>host ID</i>	128.0.0.0 bis 191.255.255.255
C	1	1	0	<i>network ID</i>	<i>host ID</i>	192.0.0.0 bis 223.255.255.255
D	1	1	1	0	<i>multicast address</i>	224.0.0.0 bis 239.255.255.255
E	1	1	1	1	0	<i>reserved for future use</i> 240.0.0.0 bis 255.255.255.255

- jedes Netzwerk der Klassen A, B oder C kann in **Subnetzwerke** aufgeteilt werden, z.B. für Klasse B:

B	1	0	<i>network ID</i>	<i>subnetwork ID</i>	<i>host ID</i>
---	---	---	-------------------	----------------------	----------------

- Unterteilung wird jedem Knoten eines Subnetzwerks durch eine binäre **Subnetzmaske** (0 für alle *host ID* Bits) angezeigt, z.B. für ein Subnetzwerk mit 64 Adressen: 255.255.255.196

5.2 IPv4 Adressen (2)

- spezielle IP Adressen:

00	this host		
00	host ID	host in the local network	
network ID		11111111	network broadcast
11			local broadcast
127	(anything)		loopback test

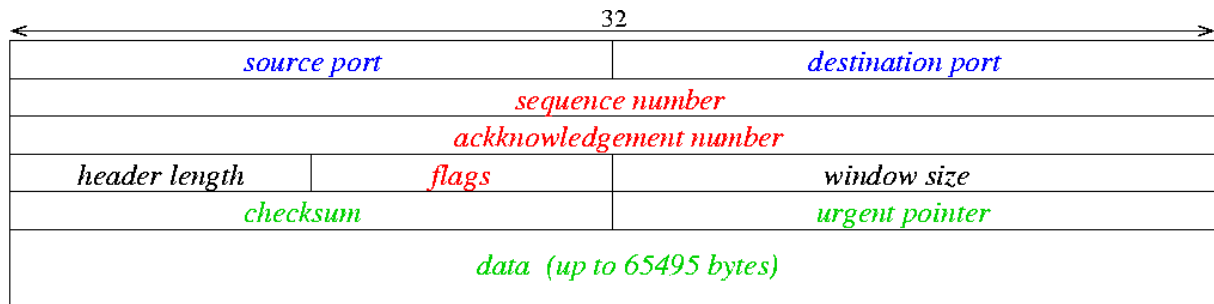
- den IP-Adressen werden **symbolische Namen** zugeordnet (z.B.: 134.60.73.1 = neuro.informatik.uni-ulm.de)
- mittels des **DNS-Dienstes** (*Domain Name Service*) kann die IP-Adresse eines Rechners anhand des symbolischen Namens ermittelt werden
- **ARP-Protokoll** dient der Abbildung von logischen IP-Adressen auf physikalische 48-Bit Ethernet-Adressen

5.3 TCP Protokoll

- Protokoll der Transportschicht **zwischen zwei Endknoten** einer Datenübertragung
- ermöglicht Übertragung eines byte-orientierten **Datenstroms** vom Sender zum Empfänger
- leistet **sichere Übertragung**: TCP erkennt und fordert fehlende Pakete erneut an, sortiert Pakete ggf. in korrekter Reihenfolge
- Schnittstellen zum TCP bilden von der Anwendung vergebene Kommunikationspunkte (**Ports**), die durch 16-Bit Portnummern angegeben werden
- Portnummern für Standard-Anwendungsdienste sind festgelegt (z.B.: 7 für echo, 21 für ftp, 23 für telnet, 25 für smtp, 80 für http)
- die Kombination aus IP-Adresse und Portnummer bezeichnet man als **TSAP** (*Transport Service Access Point*)

5.3 TCP Protokoll (2)

- Aufbau eines **TCP-Paketes**:



- *source/destination port* : End-Kommunikationspunkte auf beiden Seiten
- *sequence number* : Nummer des ersten im Paket gesendeten Bytes
- *acknowledgement number* : Nummer des nächsten erwarteten Bytes
- *flags* : Kennzeichnung spezieller Nachrichten, z.B. „*acknowledgement*“, „*urgent*“, „*final*“ oder „*connection reset*“
- *urgent pointer* : Zeiger (Offset) auf wichtige Information in den Daten
- *checksum* : 16-Bit Prüfsumme, für Header und Daten!

5.4 Sockets

- auf TCP/IP aufbauende „*stream*“-basierte **Kommunikations-schnittstelle**, mit der eine Punkt-zu-Punkt Verbindung zwischen zwei Prozessen implementiert werden kann
- Unterscheidung zwischen **Server** (Dienstanbieter) und **Klient**
- Sockets in Java: Klassen `Socket` und `ServerSocket` im Package `java.net`
- Ablauf einer Socket-basierten Kommunikation in Java:

auf Server-Seite:

```
ServerSocket s = new ServerSocket(1234);
while (true) {
    Socket c = sock.accept();
    in = c.getInputStream();
    out = c.getOutputStream();
    query = in.read();
    out.write(...);
    c.close(); in.close(); out.close();
}
```

auf Klienten-Seite:

```
...
Socket s = new Socket("a.b.de",1234);
out = s.getOutputStream();
in = s.getInputStream();
out.write(...);
answer = in.read();
s.close(); in.close(); out.close();
...
```

6 Lernziele

- **Begriffe:** halb-/voll-duplex, CSMA/CD, Ethernet, TSAP, ...
- Vor- und Nachteile verschiedener **Netztopologien**
- Funktionalität der 7 Schichten des **ISO/OSI-Referenzmodells**
 - Einsatz zur Unterscheidung von Hub, Switch, Router
 - Einsatz zur Einordnung von Kommunikationsprotokollen
- Arbeitsweise einer **paketorientierten Datenübertragung**
 - **Aufbau** von Paketen (Bsp.: TCP/IP)
 - **Adressierung:**
 - physikalische Adressen (Bsp.: Ethernet)
 - logische Adressen für Hosts und (Sub-)Netzwerke (Bsp.: IPv4)
 - **Sicherung:**
 - Paritätsbits und Prüfsummen (Bsp.: HDLC, Ethernet)
 - Sequenz- und Quittierungsnummern (Bsp.: HDLC, TCP)