PC-Hardware und Linux

Ansteuern der Schnittstellen

Jürgen Plate

14. März 2012

Parallele Schnittstelle

- Peripheriegeräte über mehrere Leitungen gleichzeitig ansprechen

Parallele Schnittstelle

- Peripheriegeräte über mehrere Leitungen gleichzeitig ansprechen
- Daten schnell byte- oder wortweise einlesen oder ausgeben

Parallele Schnittstelle

- Peripheriegeräte über mehrere Leitungen gleichzeitig ansprechen
- Daten schnell byte- oder wortweise einlesen oder ausgeben
- Standardgerät ist der Drucker

Parallele Schnittstelle

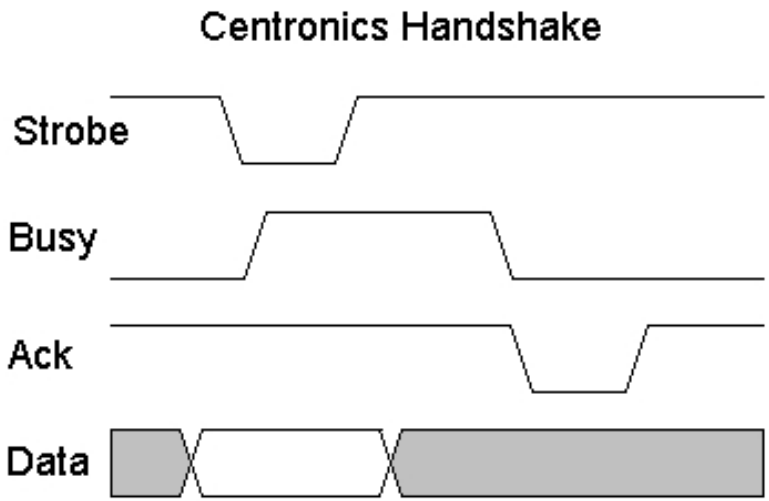
- Peripheriegeräte über mehrere Leitungen gleichzeitig ansprechen
- Daten schnell byte- oder wortweise einlesen oder ausgeben
- Standardgerät ist der Drucker
- „klassische“ Druckerschnittstelle: 8 Datenleitungen, 1 Strobe-Leitung, 1 Busy-Leitung, 1 Acknowledge-Leitung

Parallele Schnittstelle

- Peripheriegeräte über mehrere Leitungen gleichzeitig ansprechen
- Daten schnell byte- oder wortweise einlesen oder ausgeben
- Standardgerät ist der Drucker
- „klassische“ Druckerschnittstelle: 8 Datenleitungen, 1 Strobe-Leitung, 1 Busy-Leitung, 1 Acknowledge-Leitung
- später: bidirektionale Datenleitungen, weitere Leitungen/Funktionen

Centronics-Schnittstelle

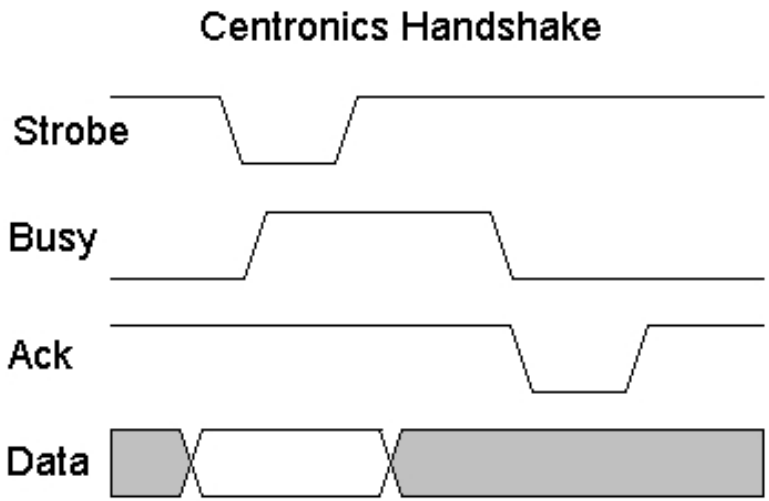
Der normale Ablauf beim Drucken eines Zeichens stellt sich folgendermaßen dar:



- Zeichen wird an den Datenport anlegen und warten, bis Busy-Leitung auf Low-Pegel geht

Centronics-Schnittstelle

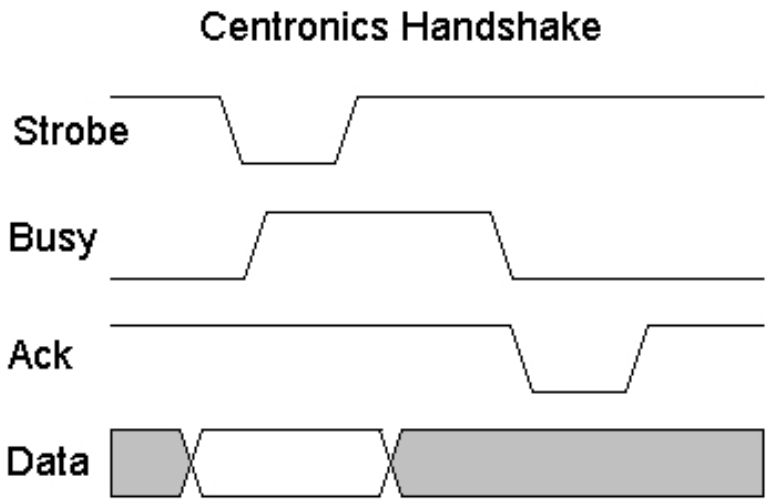
Der normale Ablauf beim Drucken eines Zeichens stellt sich folgendermaßen dar:



- Zeichen wird an den Datenport anlegen und warten, bis Busy-Leitung auf Low-Pegel geht
- Low-Impuls auf die STROBE-Leitung legen – das Zeichen wird vom Drucker übernommen

Centronics-Schnittstelle

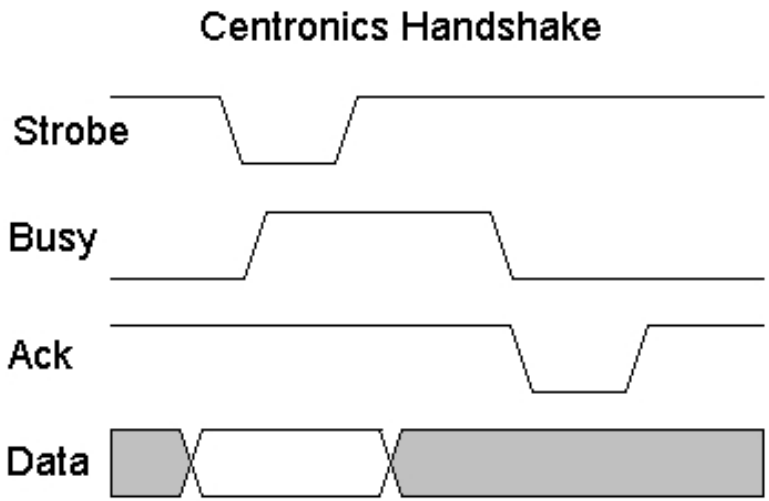
Der normale Ablauf beim Drucken eines Zeichens stellt sich folgendermaßen dar:



- Zeichen wird an den Datenport anlegen und warten, bis Busy-Leitung auf Low-Pegel geht
- Low-Impuls auf die STROBE-Leitung legen – das Zeichen wird vom Drucker übernommen
- warten, bis Busy-Leitung auf Low-Pegel geht und/oder Low-Impuls an Ack auftritt

Centronics-Schnittstelle

Der normale Ablauf beim Drucken eines Zeichens stellt sich folgendermaßen dar:



- Zeichen wird an den Datenport anlegen und warten, bis Busy-Leitung auf Low-Pegel geht
- Low-Impuls auf die STROBE-Leitung legen – das Zeichen wird vom Drucker übernommen
- warten, bis Busy-Leitung auf Low-Pegel geht und/oder Low-Impuls an Ack auftritt

Schnittstelleneigenschaften

- erste Realisierung mit Standard-Logikbausteinen, heute in der South-Bridge implementiert

Schnittstelleneigenschaften

- erste Realisierung mit Standard-Logikbausteinen, heute in der South-Bridge implementiert
- bei modernen Rechnern ist sehr schnelle bidirektionale Datenübertragung möglich

Schnittstelleneigenschaften

- erste Realisierung mit Standard-Logikbausteinen, heute in der South-Bridge implementiert
- bei modernen Rechnern ist sehr schnelle bidirektionale Datenübertragung möglich
- $\rightarrow$ gedacht für Peripherie wie Scanner, CD-ROM-Laufwerke usw.

Schnittstelleneigenschaften

- erste Realisierung mit Standard-Logikbausteinen, heute in der South-Bridge implementiert
- bei modernen Rechnern ist sehr schnelle bidirektionale Datenübertragung möglich
- $\rightarrow$ gedacht für Peripherie wie Scanner, CD-ROM-Laufwerke usw.
- $\rightarrow$ inzwischen sind diese Aufgaben vom USB übernommen worden

Schnittstelleneigenschaften

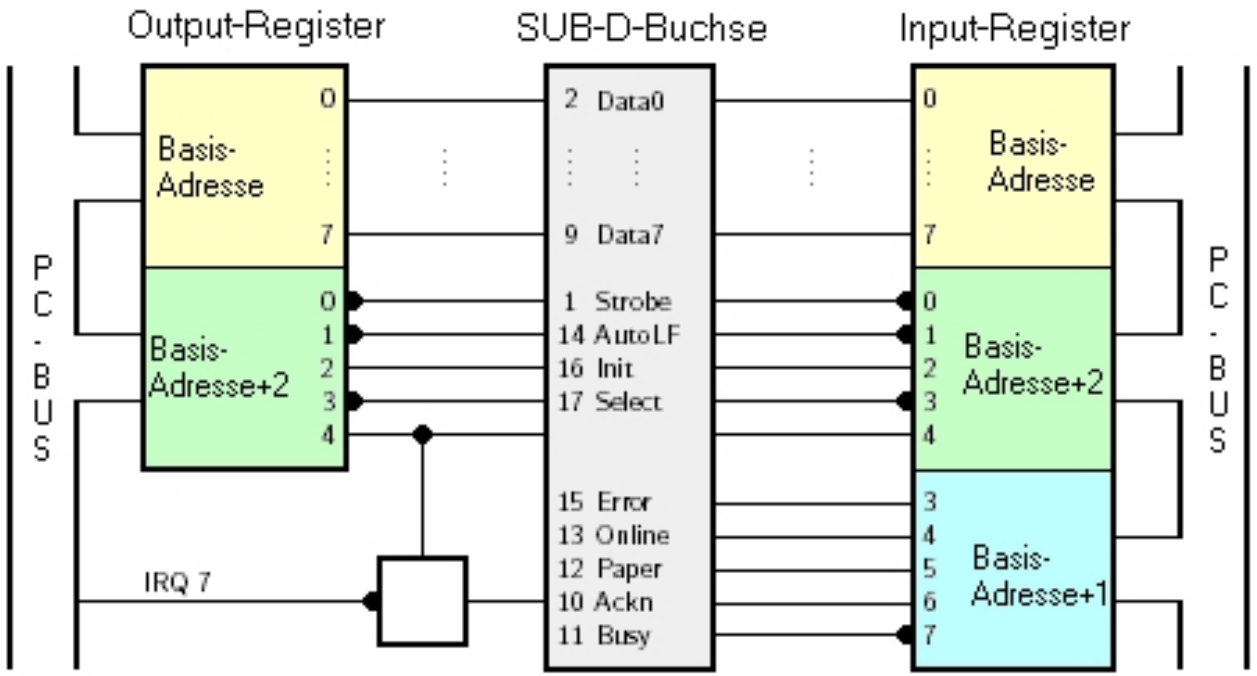
- erste Realisierung mit Standard-Logikbausteinen, heute in der South-Bridge implementiert
- bei modernen Rechnern ist sehr schnelle bidirektionale Datenübertragung möglich
- $\rightarrow$ gedacht für Peripherie wie Scanner, CD-ROM-Laufwerke usw.
- $\rightarrow$ inzwischen sind diese Aufgaben vom USB übernommen worden
- $\rightarrow$ daher hier Beschränkung auf den ursprünglichen Standard (SPP)

Schnittstelleneigenschaften

- erste Realisierung mit Standard-Logikbausteinen, heute in der South-Bridge implementiert
- bei modernen Rechnern ist sehr schnelle bidirektionale Datenübertragung möglich
- $\rightarrow$ gedacht für Peripherie wie Scanner, CD-ROM-Laufwerke usw.
- $\rightarrow$ inzwischen sind diese Aufgaben vom USB übernommen worden
- $\rightarrow$ daher hier Beschränkung auf den ursprünglichen Standard (SPP)
- Für „Bastelarbeiten“ empfiehlt sich der Kauf einer Steckkarte

Schaltungstechnik 1

Die „klassische“ Druckerschnittstelle wird über festverdrahtete, nicht weiter konfigurierbare Ausgabe- und Eingabe-Register betrieben. Überblick der Schaltung und Zuordnung der Register-Bits zur 25-poligen Buchse:



Schaltungstechnik 2

Nutzbare Leitungen:

- 8 Datenleitungen (Speicher-Register = Latch), rücklesbar

Schaltungstechnik 2

Nutzbare Leitungen:

- 8 Datenleitungen (Speicher-Register = Latch), rücklesbar
- 5 Statusleitungen (Busy invertiert)

Schaltungstechnik 2

Nutzbare Leitungen:

- 8 Datenleitungen (Speicher-Register = Latch), rücklesbar
- 5 Statusleitungen (Busy invertiert)
- 4 Steuerleitungen (Strobe, AutoLF, Select invertiert)

Schaltungstechnik 2

Nutzbare Leitungen:

- 8 Datenleitungen (Speicher-Register = Latch), rücklesbar
- 5 Statusleitungen (Busy invertiert)
- 4 Steuerleitungen (Strobe, AutoLF, Select invertiert)
- TTL-kompatibel ($0 \equiv 0 \ldots 0.4$ V, $1 \equiv 3{,}4 \ldots 5$ V)

Schaltungstechnik 2

Nutzbare Leitungen:

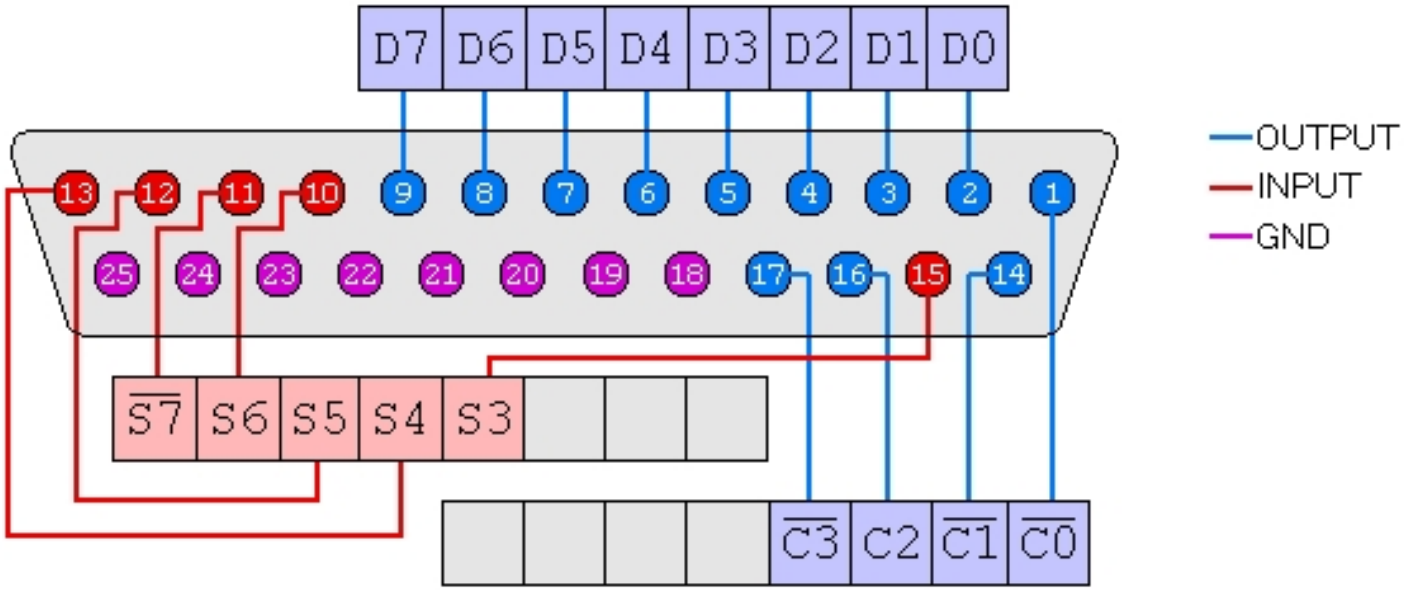
- 8 Datenleitungen (Speicher-Register = Latch), rücklesbar
- 5 Statusleitungen (Busy invertiert)
- 4 Steuerleitungen (Strobe, AutoLF, Select invertiert)
- TTL-kompatibel ($0 \equiv 0 \ldots 0.4$ V, $1 \equiv 3{,}4 \ldots 5$ V)
- Leitungslänge: einige Meter

Schaltungstechnik 2

Nutzbare Leitungen:

- 8 Datenleitungen (Speicher-Register = Latch), rücklesbar
- 5 Statusleitungen (Busy invertiert)
- 4 Steuerleitungen (Strobe, AutoLF, Select invertiert)
- TTL-kompatibel ($0 \equiv 0 \ldots 0.4$ V, $1 \equiv 3{,}4 \ldots 5$ V)
- Leitungslänge: einige Meter
- Datenrate (theoretisch): einige $10 \ldots 100$ kHz

Steckerbelegung



Steckerbelegung

SUB-D	Signal	Richtung	Register	invert.
1	Strobe	Out	Control	Ja
2 - 9	Data 0 - 7	Out	Data	
10	Ack	In	Status	
11	Busy	In	Status	Ja
12	Paper-Out	In	Status	
13	Select	In	Status	
14	Auto-Linefeed	Out	Control	Ja
15	Error	In	Status	
16	Reset	Out	Control	
17	Select-In	Out	Control	Ja

Portadressen

- Standardbelegung:

Data	**Status**	**Control**
278	279	27A
378	379	37A
3BC	3BD	3BE

Portadressen

- Standardbelegung:

Data	**Status**	**Control**
278	279	27A
378	379	37A
3BC	3BD	3BE

- Beim PCI-Bus auch höhere Adressen möglich

Portadressen

- Standardbelegung:

Data	**Status**	**Control**
278	279	27A
378	379	37A
3BC	3BD	3BE

- Beim PCI-Bus auch höhere Adressen möglich
- Adressen herausfinden mit: *cat /proc/ioports — less*

Datenport

Der Datenport ist ein 8-Bit-Ausgangsport, der ursprünglich durch ein Register realisiert wurde. Die ausgegebenen Daten können über einen Leitungs-Treiber zurückgelesen werden. Ein erstes Programmierbeispiel (noch unvollständig):

```
#define DATA 0x378
#define STATUS 0x379
#define CONTROL 0x37A
...
unsigned char byte;
...
/* Ausgabe des Bitmusters 0110 1001 */
byte = 0x69;
outb(byte, DATA);

/* Setzen Bit 5, alle anderen Bits belassen */
outb(inb(DATA) | 0x10, DATA);

/* Ruecksetzen Bit 5, alle anderen Bits belassen */
outb(inb(DATA) & (~0x10), DATA);
```

Steuerport

Bit	Funktion	Pegel
0	Strobe	0
1	Auto-Feed	1
2	Reset	0
3	Select printer	1
4	IRQ7 enable	1
5	Direction	1

- Direction-Bit nur von bidirektionalen Schnittstellen unterstützt

Steuerport

Bit	Funktion	Pegel
0	Strobe	0
1	Auto-Feed	1
2	Reset	0
3	Select printer	1
4	IRQ7 enable	1
5	Direction	1

- Direction-Bit nur von bidirektionalen Schnittstellen unterstützt
- einige Leitungen des Steuerports sind per Hardware invertiert

Steuerport

Bit	Funktion	Pegel
0	Strobe	0
1	Auto-Feed	1
2	Reset	0
3	Select printer	1
4	IRQ7 enable	1
5	Direction	1

- Direction-Bit nur von bidirektionalen Schnittstellen unterstützt
- einige Leitungen des Steuerports sind per Hardware invertiert
- korrigieren durch Exklusiv-Oder-Verknüpfung des Wertes mit 0x0B (binär 0000 1011)

Steuerport

Bit	Funktion	Pegel
0	Strobe	0
1	Auto-Feed	1
2	Reset	0
3	Select printer	1
4	IRQ7 enable	1
5	Direction	1

- Direction-Bit nur von bidirektionalen Schnittstellen unterstützt
- einige Leitungen des Steuerports sind per Hardware invertiert
- korrigieren durch Exklusiv-Oder-Verknüpfung des Wertes mit 0x0B (binär 0000 1011)
- *outb(byte ĵ0x0B, CONTROL)*

Statusport

Bit	Funktion	Pegel
3	Fehler	0
4	Select out	1
5	Paper out	1
6	Acknowledge	0
7	Busy	0

- das Busy-Bit ist hardwaremäßig invertiert

Statusport

Bit	Funktion	Pegel
3	Fehler	0
4	Select out	1
5	Paper out	1
6	Acknowledge	0
7	Busy	0

- das Busy-Bit ist hardwaremäßig invertiert
- korrigieren durch Exor-Verknüpfung mit 0x80 (binär 1000000)

Statusport

Bit	Funktion	Pegel
3	Fehler	0
4	Select out	1
5	Paper out	1
6	Acknowledge	0
7	Busy	0

- das Busy-Bit ist hardwaremäßig invertiert
- korrigieren durch Exor-Verknüpfung mit 0x80 (binär 1000000)
- gleich noch nach unten auf die Bits 0 bis 4 verschieben

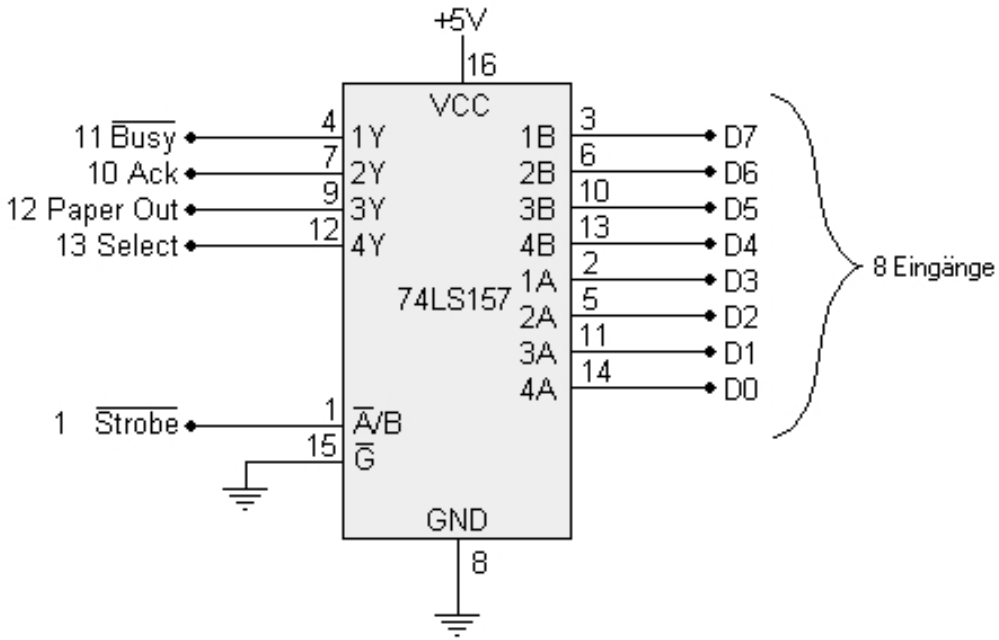
Statusport

Bit	Funktion	Pegel
3	Fehler	0
4	Select out	1
5	Paper out	1
6	Acknowledge	0
7	Busy	0

- das Busy-Bit ist hardwaremäßig invertiert
- korrigieren durch Exor-Verknüpfung mit 0x80 (binär 1000000)
- gleich noch nach unten auf die Bits 0 bis 4 verschieben
- `byte = ((inb(STATUS) ^0x80) >> 3);`

Eingangserweiterung 1

Mit etwas Hardware kann man einen statischen 8-Bit-Eingang realisieren. Es wird ein Multiplexer an den Statusport geschaltet. Über die Strobe-Leitung des Steuerports kann nun das obere oder untere Nibble ausgewählt werden.



Eingangserweiterung 2

Programmierung mit logischen Verknüpfungen:

```
outb(inb(CONTROL) | 0x01, CONTROL); /* Select Low Nibble (A)*/
byte = inb(STATUS) & 0xF0;          /* Read Low Nibble */
byte = byte >> 4;                   /* Shift Right 4 Bits */
outb(inb(CONTROL) & 0xFE, CONTROL); /* Select High Nibble (B)*/
byte = byte | (inb(STATUS) & 0xF0); /* Read High Nibble */
byte = byte ^ 0x88;                 /* Adjust Value */
```

Programmierung 1

- Programmierung im User-Space (keine Treiber)

Programmierung 1

- Programmierung im User-Space (keine Treiber)
- Programme schreiben/lesen direkt auf/von Hardware-Ports

Programmierung 1

- Programmierung im User-Space (keine Treiber)
- Programme schreiben/lesen direkt auf/von Hardware-Ports
- $\rightarrow$ nur mit *root*-Berechtigung erlaubt

Programmierung 1

- Programmierung im User-Space (keine Treiber)
- Programme schreiben/lesen direkt auf/von Hardware-Ports
- $\rightarrow$ nur mit *root*-Berechtigung erlaubt
- Lesen mit: *unsigned inb(unsigned port)*

Programmierung 1

- Programmierung im User-Space (keine Treiber)
- Programme schreiben/lesen direkt auf/von Hardware-Ports
- $\rightarrow$ nur mit *root*-Berechtigung erlaubt
- Lesen mit: *unsigned inb(unsigned port)*
- Schreiben mit: *void outb(unsigned char byte, unsigned port)*

Programmierung 1

- Programmierung im User-Space (keine Treiber)
- Programme schreiben/lesen direkt auf/von Hardware-Ports
- $\rightarrow$ nur mit *root*-Berechtigung erlaubt
- Lesen mit: *unsigned inb(unsigned port)*
- Schreiben mit: *void outb(unsigned char byte, unsigned port)*
- Bibliothek: *unistd.h* (libc5) oder *sys/io.h* (glibc)

Programmierung 1

- Programmierung im User-Space (keine Treiber)
- Programme schreiben/lesen direkt auf/von Hardware-Ports
- $\rightarrow$ nur mit *root*-Berechtigung erlaubt
- Lesen mit: *unsigned inb(unsigned port)*
- Schreiben mit: *void outb(unsigned char byte, unsigned port)*
- Bibliothek: *unistd.h* (libc5) oder *sys/io.h* (glibc)
- Ports müssen reserviert und wieder frei gegeben werden

Programmierung 1

- Programmierung im User-Space (keine Treiber)
- Programme schreiben/lesen direkt auf/von Hardware-Ports
- $\rightarrow$ nur mit *root*-Berechtigung erlaubt
- Lesen mit: *unsigned inb(unsigned port)*
- Schreiben mit: *void outb(unsigned char byte, unsigned port)*
- Bibliothek: *unistd.h* (libc5) oder *sys/io.h* (glibc)
- Ports müssen reserviert und wieder frei gegeben werden
- Rechte werden nicht von *fork()*, wohl aber von *exec()* vererbt

Programmierung 2

- *inb()* und *outb()* sind eigentlich Makros

Programmierung 2

- *inb()* und *outb()* sind eigentlich Makros
- mit der Option -O (-O2) kompilieren → Expansion der Inline-Funktionen

Programmierung 2

- *inb()* und *outb()* sind eigentlich Makros
- mit der Option -O (-O2) kompilieren → Expansion der Inline-Funktionen
- reservieren mit *ioperm()* (einzelne Ports) bzw. *iopl()* (gesamter Adessraum).

Programmierung 2

- *inb()* und *outb()* sind eigentlich Makros
- mit der Option -O (-O2) kompilieren → Expansion der Inline-Funktionen
- reservieren mit *ioperm()* (einzelne Ports) bzw. *iopl()* (gesamter Adessraum).
- beide Funktionen sind spezifisch für Intel/AMD-Prozessoren

Programmierung 2

- *inb()* und *outb()* sind eigentlich Makros
- mit der Option -O (-O2) kompilieren → Expansion der Inline-Funktionen
- reservieren mit *ioperm()* (einzelne Ports) bzw. *iopl()* (gesamter Adessraum).
- beide Funktionen sind spezifisch für Intel/AMD-Prozessoren
- *int ioperm (unsigned long from, unsigned long num, int turn_on)*
  Portzugriff reservieren für *num* Adressen, beginnend bei *from*.
  Belegen mit $turn\_on = 1$, freigeben mit $turn\_on = 0$.

Programmierung 2

- *inb()* und *outb()* sind eigentlich Makros
- mit der Option -O (-O2) kompilieren → Expansion der Inline-Funktionen
- reservieren mit *ioperm()* (einzelne Ports) bzw. *iopl()* (gesamter Adessraum).
- beide Funktionen sind spezifisch für Intel/AMD-Prozessoren
- *int ioperm (unsigned long from, unsigned long num, int turn_on)*
  Portzugriff reservieren für *num* Adressen, beginnend bei *from*. Belegen mit $turn\_on = 1$, freigeben mit $turn\_on = 0$.
- Bei *iopl()* wird nur ein Level angeben (3 = Portzugriff, 0 = Freigabe)

Programmierung 3

Ein erstes Beispiel:

```
#include <sys/io.h>
int main(int argc, char* argv[])
  {
  int base = atoi(argv[1]);
  int value = atoi(argv[2]);
  ioperm(base,1,1);
  outb(value,base);
  ioperm(base,1,0);
  return 0;
  }
```

Übersetzt wird das Programm mit *gcc -O2 -o iop iop.c* und kann dann beispielsweise mit *./iop 888 170* gestartet werden.

Programmierung 4

Das folgende Programm benötigt zwei Parameter auf der Kommandozeile:

- dataport value: Ausgabewert für den Datenport (0 ... 255)
- control port value: Ausgabewert für den Steuerport (0 ... 15)

Die Portnummer ist fest verdrahtet. Zuerst der „Vorspann“:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/io.h>
#include <unistd.h>

/* Base addresses of parallel port */
#define LPT 0x378

void usage(void)
  {
  printf("Syntax : setport <dataport value> <control port value>\n");
  exit(1);
  }

int main(int argc, char *argv[])
  {
  int data = 0;             /* data port value */
  int control = 0;          /* control port value */
  int BASEPORT = LPT;       /* selected printer port */

  if (argc != 3) usage();         /* falsche Parameteranzahl */
```

Programmierung 6

Eingabedaten prüfen und auf erlaubte Werte begrenzen:

```
data = atoi(argv[1]);
if (data < 0 || data > 255)
  { perror("Error: data value out of range"); exit(1); }
control = atoi(argv[2]);
if (control < 0 || control > 15)
  { perror("Error: control value out of range"); exit(1); }
/* Get access to the ports */
if (ioperm(BASEPORT, 3, 1))
  { perror("Error: cannot access ioport"); exit(1); }
```

Programmierung 7

Die eingentliche Ausgabe

```
/* Set the data signals of the port */
outb(data, BASEPORT);
outb(control ^ 0x0B, BASEPORT + 2);  /* mit Korrektur! */
/* We don't need the ports anymore */
if (ioperm(BASEPORT, 3, 0))
  { perror("Error: cannot access ioport"); exit(1); }
exit(0);
}
```

Programmierung 8

Programm für das Lesen vom Statusport. Der Rückgabewert ist im Fehlerfall 255. Bei erfolgreichem Portzugriff entspricht er den Werten des Statusports (nach rechts geschoben auf die Bits 0 - 4).

```
#include <stdio.h>
#include <sys/io.h>
#include <unistd.h>
#define LPT 0x378  /* Base addresses of parallel port */

int main(void)
  {
  int status = 0, BASEPORT = LPT;
  /* Get access to the ports */
  if (ioperm(BASEPORT, 3, 1))
    { perror("Error: cannot access ioport"); exit(255); }
  /* Get status port */
  status = ((inb(BASEPORT + 1) ^ 0x80) >> 3);
  /* We don't need the ports anymore */
  if (ioperm(BASEPORT, 3, 0))
    { perror("Error: cannot access ioport"); exit(255); }
  exit(status);
  }
```

Mehr unter

`http://www.netzmafia.de/skripten/hardware/Parallel/`

PC-Hardware und Linux

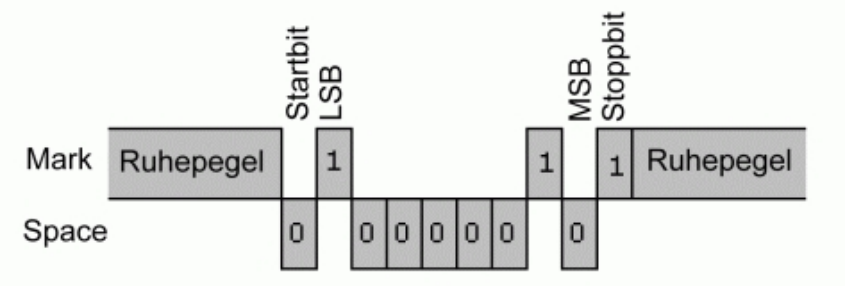
Ansteuern der Schnittstellen

Jürgen Plate

14. März 2012

Serielle Übertragung

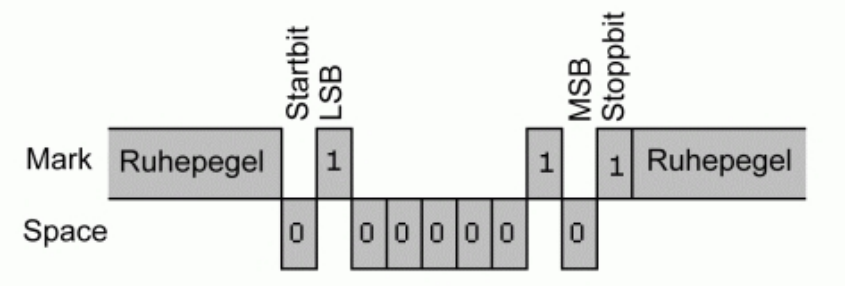
Bei einem seriellen, asynchronen Datentransfer werden die einzelnen Bits eines Datenbytes nacheinander über eine Leitung übertragen.



- Byte als Folge von acht Datenbits, die von Start- und Stoppbit eingerahmt werden.

Serielle Übertragung

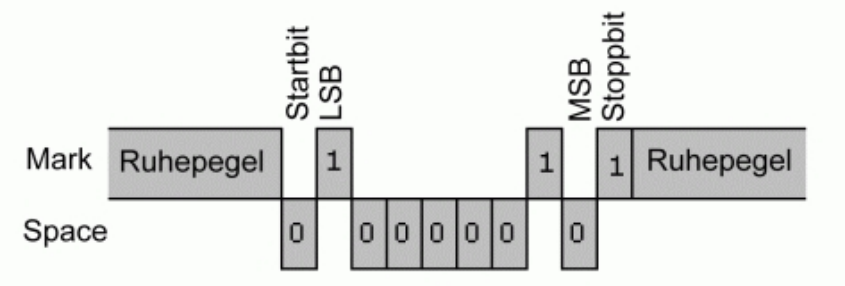
Bei einem seriellen, asynchronen Datentransfer werden die einzelnen Bits eines Datenbytes nacheinander über eine Leitung übertragen.



- Byte als Folge von acht Datenbits, die von Start- und Stoppbit eingerahmt werden.
- Beliebig lange Pausen zwischen zwei Bytes

Serielle Übertragung

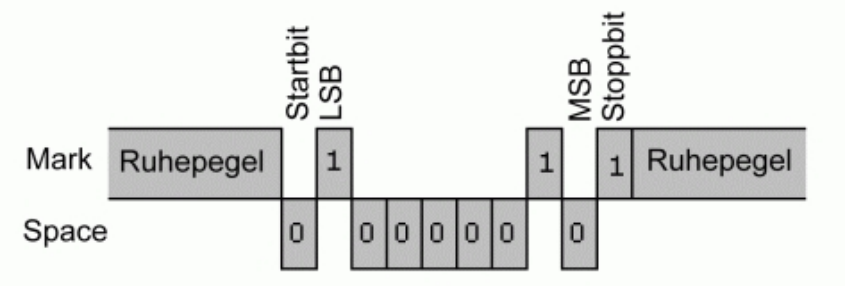
Bei einem seriellen, asynchronen Datentransfer werden die einzelnen Bits eines Datenbytes nacheinander über eine Leitung übertragen.



- Byte als Folge von acht Datenbits, die von Start- und Stoppbit eingerahmt werden.
- Beliebig lange Pausen zwischen zwei Bytes
- Beginn eines Zeichens wird am Startbit eindeutig erkannt

Serielle Übertragung

Bei einem seriellen, asynchronen Datentransfer werden die einzelnen Bits eines Datenbytes nacheinander über eine Leitung übertragen.



- Byte als Folge von acht Datenbits, die von Start- und Stoppbit eingerahmt werden.
- Beliebig lange Pausen zwischen zwei Bytes
- Beginn eines Zeichens wird am Startbit eindeutig erkannt
- asynchrone Übertragung

Handshake

Zur Vermeidung von Datenverlusten muss der Empfänger die Datenübertragung anhalten können, wenn keine weiteren Daten mehr verarbeitet werden können. Dieses sogenannte Handshake kann auf zwei Arten realisiert werden:

- **Hardware-Handshake:** Der Empfänger steuert über entsprechende Leitungen die Handshake-Eingänge des Senders (CTS, DSR) mit seinen Handshake-Ausgängen (DTR, RTS).

Handshake

Zur Vermeidung von Datenverlusten muss der Empfänger die Datenübertragung anhalten können, wenn keine weiteren Daten mehr verarbeitet werden können. Dieses sogenannte Handshake kann auf zwei Arten realisiert werden:

- **Hardware-Handshake:** Der Empfänger steuert über entsprechende Leitungen die Handshake-Eingänge des Senders (CTS, DSR) mit seinen Handshake-Ausgängen (DTR, RTS).
- **Software-Handshake:** Der Empfänger sendet zur Steuerung des Datenflusses spezielle Zeichen an den Sender (z. B. XON, XOFF).

Die Serielle PC-Schnittstelle

Herzstück ist der serielle Baustein UART 8250 (Universal Asynchronous Receiver and Transmitter) bzw. kompatible Schaltungen.

- Gerätenamen: /dev/ttyS0 . . . /dev/ttySx

Die Serielle PC-Schnittstelle

Herzstück ist der serielle Baustein UART 8250 (Universal Asynchronous Receiver and Transmitter) bzw. kompatible Schaltungen.

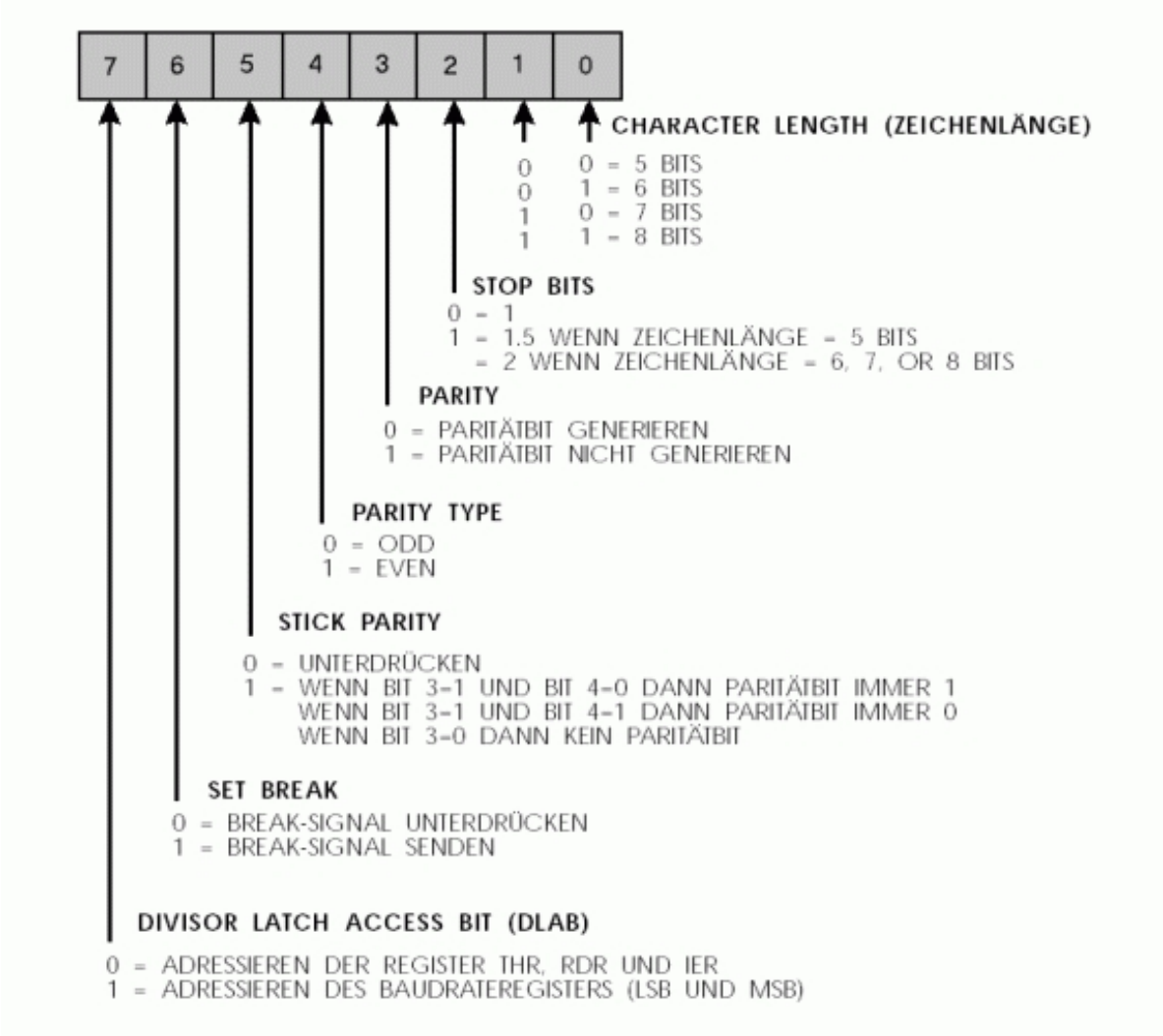
- Gerätenamen: /dev/ttyS0 . . . /dev/ttySx
- Adressen:

Port	**Funktion**
Base	Sendedaten (TDR) Empfangsdaten (RDR) [1]
Base	Baudrate (niederwertiges Bit) [2]
Base+1	Interrupt Enable Register (IER) [1]
Base+1	Baudrate (höherwertiges Bit) [2]
Base+2	Interrupt ID (IID)
Base+3	Line Control
Base+4	Modem Control
Base+5	Line Status
Base+6	Modem Status

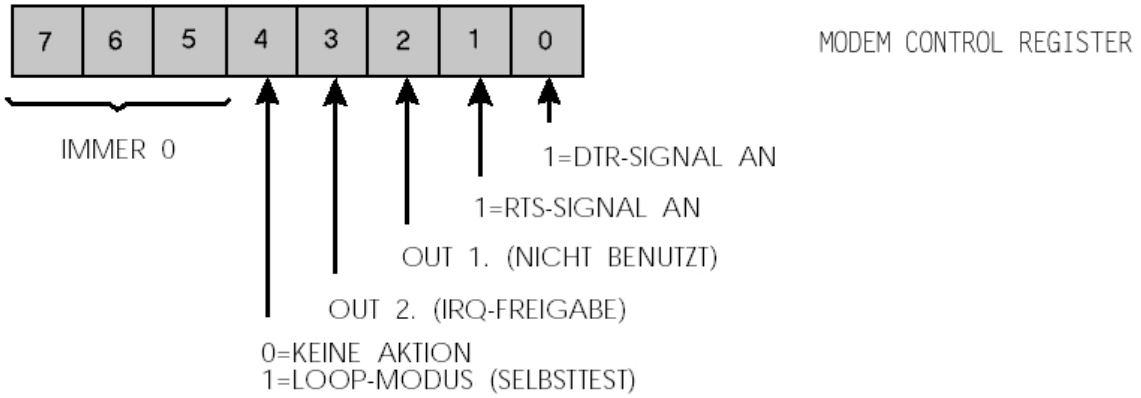
[1] Bit 7 im Line Control Register = 0

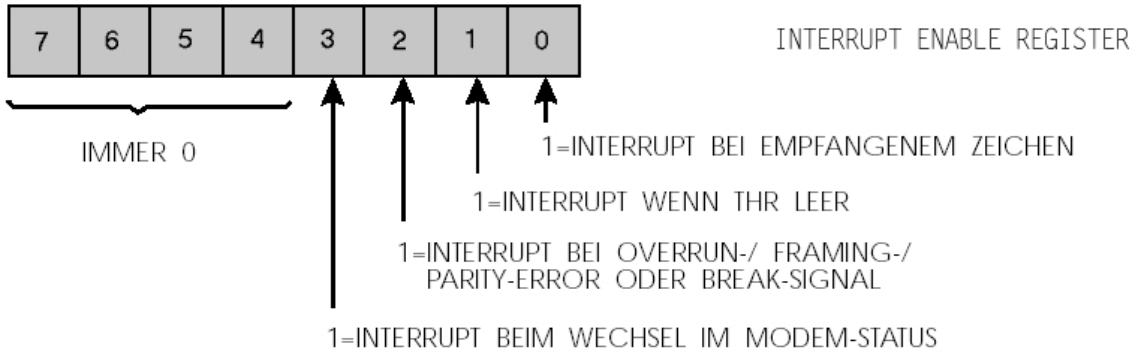
[2] Bit 7 im Line Control Register = 1

Line-Control-Register

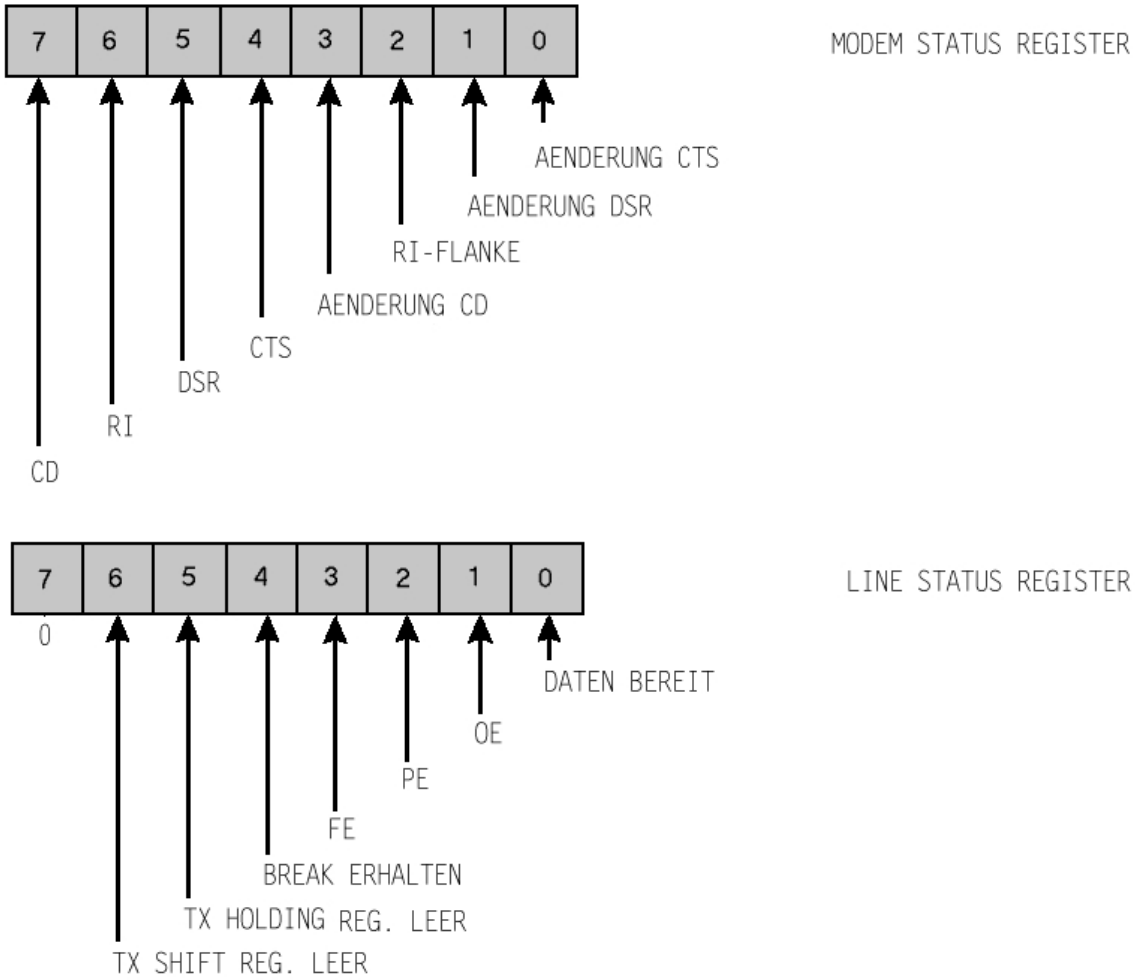


Modem-Control und Interrupt-Register





Modem- und Line-Status-Register



Die RS232-Schnittstelle

RS232C ist in Europa mit fast identischen Eigenschaften unter V.24 genormt. Die Signale der RS232 sind bipolar ausgelegt.

- logische 0 bei Datenleitungen: +3 bis +15 Volt

Die RS232-Schnittstelle

RS232C ist in Europa mit fast identischen Eigenschaften unter V.24 genormt. Die Signale der RS232 sind bipolar ausgelegt.

- logische 0 bei Datenleitungen: +3 bis +15 Volt
- logische 1 bei Datenleitungen: -3 bis -15 Volt

Die RS232-Schnittstelle

RS232C ist in Europa mit fast identischen Eigenschaften unter V.24 genormt. Die Signale der RS232 sind bipolar ausgelegt.

- logische 0 bei Datenleitungen: +3 bis +15 Volt
- logische 1 bei Datenleitungen: -3 bis -15 Volt
- Bei den Steuerleitungen sind die Pegel genau umgekehrt

Die RS232-Schnittstelle

RS232C ist in Europa mit fast identischen Eigenschaften unter V.24 genormt. Die Signale der RS232 sind bipolar ausgelegt.

- logische 0 bei Datenleitungen: +3 bis +15 Volt
- logische 1 bei Datenleitungen: -3 bis -15 Volt
- Bei den Steuerleitungen sind die Pegel genau umgekehrt
- Laut EIA-Norm maximal 15 Meter lang

Die RS232-Schnittstelle

RS232C ist in Europa mit fast identischen Eigenschaften unter V.24 genormt. Die Signale der RS232 sind bipolar ausgelegt.

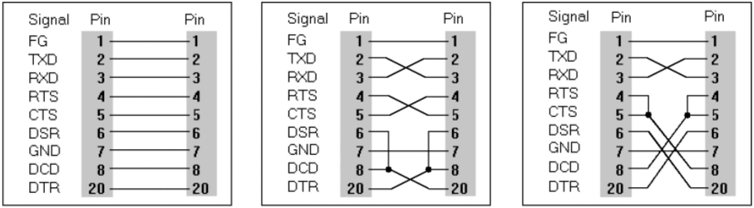
- logische 0 bei Datenleitungen: +3 bis +15 Volt
- logische 1 bei Datenleitungen: -3 bis -15 Volt
- Bei den Steuerleitungen sind die Pegel genau umgekehrt
- Laut EIA-Norm maximal 15 Meter lang
- bei kapazitätsarmen Kabeln bis zu 50 Metern

RS232-Signale

Die Richtungsangabe bezieht sich auf die Peripherie. Werden Peripherie/Modem (DÜE = Datenübertragungseinrichtung) und Computer (DEE = Datenendeinrichtung) miteinander verbunden, verwendet man ein Kabel mit einer 1:1-Verbindung der wichtigsten Leitungen.

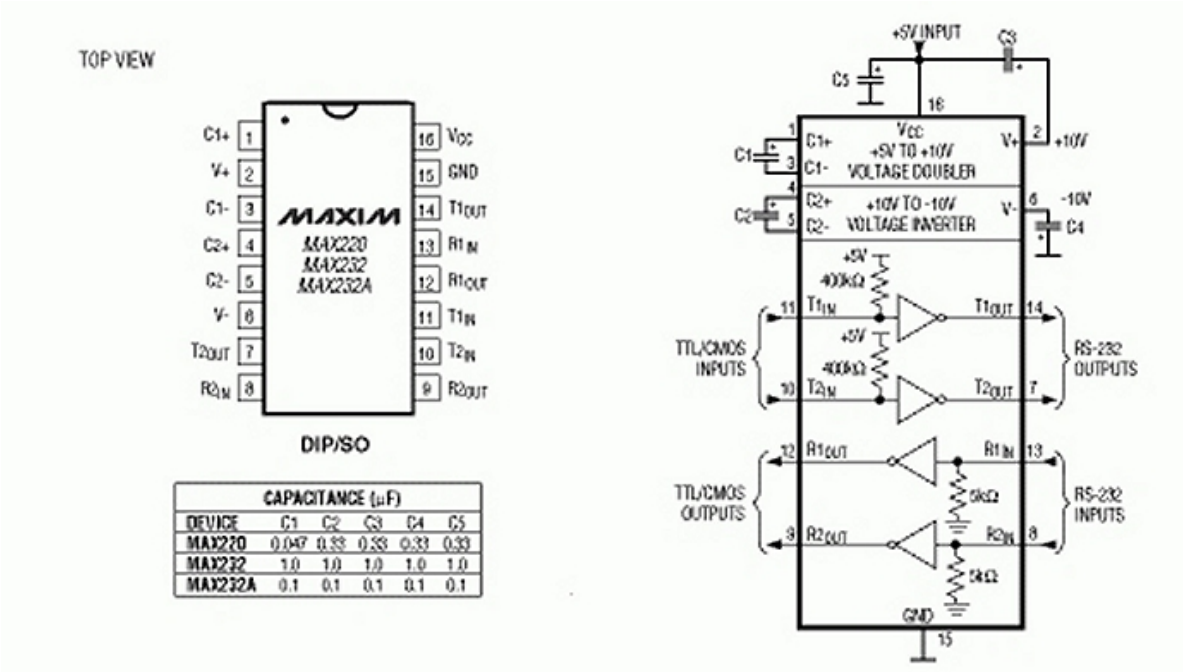
ITU	DIN	US	25-pol.	9-pol.	Beschreibung	Richt.
101	E 1	-	1	-	Schutzerde	-
102	E 2	GND	7	5	Signalerde (Ground)	-
103	D 1	TXD	2	3	Sendedaten (Transmit Data)	←
104	D 2	RXD	3	2	Empfangsdaten (Receive Data)	→
105	S 2	RTS	4	7	Sendeteil einschalten (Request To Send)	←
106	M 2	CTS	5	8	Sendebereitschaft (Clear To Send)	→
107	M 1	DSR	6	6	Betriebsbereitschaft (Data Set Ready)	→
108.2	S 1.2	DTR	20	4	Terminal ist bereit (Data Terminal Ready)	←
109	M 5	DCD	8	1	Empfangspegel (Data Carrier Detect)	→
125	M 3	RI	22	9	Ankommender Ruf (Ring Indicator)	→

Sollen zwei Computer direkt, also ohne Modem miteinander verbunden werden, müssen die Leitungen gekreuzt werden. Werden die Steuerleitungen gleich im Stecker zurückgeführt (RTS auf CTS, DTR auf DSR), kommt man mit einer dreiadrigen Verbindung aus.
So eine direkte Verbindung zwischen zwei Computern wird allgemein auch als „Nullmodem“ bezeichnet.



RS232-Pegelwandler

Bei der Pegelwandlung von TTL nach RS232 können wir auf einen bewährten integrierten Baustein zurückgreifen, den MAX232 von Maxim.



Ähnlich erfolgt die Konvertierung auf Industriestandards wie RS422 oder RS485.

USB-RS232-Adapter

Inzwischen gibt es zahlreiche Adapter für den Anschluss von RS232-Peripherie an die USB-Schnittstelle. Für fast alle Modelle liegen die Treiber bereits als Kernel-Module vor und werden bei den meisten Distributionen automatisch eingebunden.

Programmierung 1

- glücklicherweise keine direkte Hardwareprogrammierung nötig

Programmierung 1

- glücklicherweise keine direkte Hardwareprogrammierung nötig
- POSIX-standardisierte Funktionen

Programmierung 1

- glücklicherweise keine direkte Hardwareprogrammierung nötig
- POSIX-standardisierte Funktionen
- Infos: Linux Serial HOWTO, Linux Serial Programming HOWTO, GNU C-Library Reference Manual

Programmierung 1

- glücklicherweise keine direkte Hardwareprogrammierung nötig
- POSIX-standardisierte Funktionen
- Infos: Linux Serial HOWTO, Linux Serial Programming HOWTO, GNU C-Library Reference Manual
- User muss Schreib- und Leseberechtigung für das Device haben

Programmierung 1

- glücklicherweise keine direkte Hardwareprogrammierung nötig
- POSIX-standardisierte Funktionen
- Infos: Linux Serial HOWTO, Linux Serial Programming HOWTO, GNU C-Library Reference Manual
- User muss Schreib- und Leseberechtigung für das Device haben
- Gegebenenfalls in die Gruppe uucp aufnehmen oder Pseudo-User anlegen

Programmierung 1

- glücklicherweise keine direkte Hardwareprogrammierung nötig
- POSIX-standardisierte Funktionen
- Infos: Linux Serial HOWTO, Linux Serial Programming HOWTO, GNU C-Library Reference Manual
- User muss Schreib- und Leseberechtigung für das Device haben
- Gegebenenfalls in die Gruppe uucp aufnehmen oder Pseudo-User anlegen
- Hier nur die wichtigsten Dinge

Programmierung 2

- alle Parameter stehen in der Stuktur *termios* $\rightarrow$

Programmierung 2

- alle Parameter stehen in der Stuktur *termios* $\rightarrow$
  - 32-Bit-Maske für die Eingabe-Flags

Programmierung 2

- alle Parameter stehen in der Stuktur *termios* $\rightarrow$
  - 32-Bit-Maske für die Eingabe-Flags
  - 32-Bit-Maske für die Ausgabe-Flags

Programmierung 2

- alle Parameter stehen in der Stuktur *termios* $\rightarrow$
  - 32-Bit-Maske für die Eingabe-Flags
  - 32-Bit-Maske für die Ausgabe-Flags
  - 32-Bit-Maske für die Control-Flags

Programmierung 2

- alle Parameter stehen in der Stuktur *termios* $\rightarrow$
  - 32-Bit-Maske für die Eingabe-Flags
  - 32-Bit-Maske für die Ausgabe-Flags
  - 32-Bit-Maske für die Control-Flags
  - 32-Bit-Maske für lokale Einstellungen

Programmierung 2

- alle Parameter stehen in der Stuktur *termios* $\rightarrow$
  - 32-Bit-Maske für die Eingabe-Flags
  - 32-Bit-Maske für die Ausgabe-Flags
  - 32-Bit-Maske für die Control-Flags
  - 32-Bit-Maske für lokale Einstellungen
  - char line discipline

Programmierung 2

- alle Parameter stehen in der Stuktur *termios* $\rightarrow$
  - 32-Bit-Maske für die Eingabe-Flags
  - 32-Bit-Maske für die Ausgabe-Flags
  - 32-Bit-Maske für die Control-Flags
  - 32-Bit-Maske für lokale Einstellungen
  - char line discipline
  - Array c_cc[NCCS]

Programmierung 2

- alle Parameter stehen in der Stuktur *termios* $\rightarrow$
  - 32-Bit-Maske für die Eingabe-Flags
  - 32-Bit-Maske für die Ausgabe-Flags
  - 32-Bit-Maske für die Control-Flags
  - 32-Bit-Maske für lokale Einstellungen
  - char line discipline
  - Array c_cc[NCCS]
- Zugriff mit dem Kommando *stty*, z. B. *stty -a ¡/dev/ttyS0*

Programmierung 2

- alle Parameter stehen in der Stuktur *termios* $\rightarrow$
  - 32-Bit-Maske für die Eingabe-Flags
  - 32-Bit-Maske für die Ausgabe-Flags
  - 32-Bit-Maske für die Control-Flags
  - 32-Bit-Maske für lokale Einstellungen
  - char line discipline
  - Array c_cc[NCCS]
- Zugriff mit dem Kommando *stty*, z. B. *stty -a ¡/dev/ttyS0*
- Zugriff mit dem Kommando *setserial*, z. B. *setserial /dev/ttyS0 -a*

Programmierung 3

- Zugriff im Programm über Bibliotheksfunktionen

Programmierung 3

- Zugriff im Programm über Bibliotheksfunktionen
- Settings lesen: *int tcgetattr(int filedes, struct termios *termios_p)*

Programmierung 3

- Zugriff im Programm über Bibliotheksfunktionen
- Settings lesen: *int tcgetattr(int filedes, struct termios *termios_p)*
- Settings schreiben: *int tcsetattr(int filedes, int when, const struct termios *termios_p)*

Programmierung 3

- Zugriff im Programm über Bibliotheksfunktionen
- Settings lesen: *int tcgetattr(int filedes, struct termios *termios_p)*
- Settings schreiben: *int tcsetattr(int filedes, int when, const struct termios *termios_p)*
- Zeitpunkt-Parameter *when* : TCSANOW = sofort, TCSADRAIN = nach dem Senden aller Daten, TCSAFLUSH = Eingabepuffer löschen, sofort

Programmierung 3

- Zugriff im Programm über Bibliotheksfunktionen
- Settings lesen: *int tcgetattr(int filedes, struct termios *termios_p)*
- Settings schreiben: *int tcsetattr(int filedes, int when, const struct termios *termios_p)*
- Zeitpunkt-Parameter *when* : TCSANOW = sofort, TCSADRAIN = nach dem Senden aller Daten, TCSAFLUSH = Eingabepuffer löschen, sofort
- Datenrate setzen: *int cfsetspeed(struct termios *termios_p, speed_t speed)*

Programmierung 3

- Zugriff im Programm über Bibliotheksfunktionen
- Settings lesen: *int tcgetattr(int filedes, struct termios *termios_p)*
- Settings schreiben: *int tcsetattr(int filedes, int when, const struct termios *termios_p)*
- Zeitpunkt-Parameter *when* : TCSANOW = sofort, TCSADRAIN = nach dem Senden aller Daten, TCSAFLUSH = Eingabepuffer löschen, sofort
- Datenrate setzen: *int cfsetspeed(struct termios *termios_p, speed_t speed)*
- Funktionen liefern im Erfolgsfall den Wert 0 zurück und im Fehlerfall -1

Eingabeflags

c_iflag (Auswahl)

IGNBRK	Ignoriere Breaks
BRKINT	beachte Breaks
IGNPAR	ignoriere Parität
INLCR	ersetze NL durch CR
IGNCR	ignoriere CR
ICRNL	ersetze CR durch NL
IUCLC	Großbuchstaben in Kleinbuchstaben umwandeln
IXON	XON/XOFF-Flusssteuerung einschalten
IXANY	Ausgabe fortsetzen mit einem beliebigen Zeichen
IXOFF	XON/XOFF-Flusssteuerung ausschalten
IMAXBEL	Ton, wenn der Puffer voll ist (Zeilenende)

Ausgabeflags

c_oflag (Auswahl)

ONLCR	ersetze NL durch CR
OCRNL	ersetze CR durch NL
ONOCR	Unterdrücken von CR in Spalte 0
ONLRET	kein CR senden
OFILL	Füllzeichen NUL senden anstelle einer Pause
OFDEL	Füllzeichen ist DEL statt NUL

Formatflags

c_cflag, Teil 1 (Auswahl)

CREAD	Empfangsteil aktivieren
PARENB	Paritätsbit erzeugen
PARODD	ungerade Parität statt gerader
HUPCL	Verbindungsabbruch bei Ende des letzten Prozesses
CLOCAL	Terminal lokal angeschlossen (ignoriere CD)
CRTSCTS	Hardware-Handshake einschalten
CIGNORE	Controlflags ignorieren
CS5	5 Bit
CS6	6 Bit
CS7	7 Bit
CS8	8 Bit
CSTOPB	2 Stoppbits statt einem

Übertragungsrate

c_cflag, Teil 2 (Auswahl)

B0	hang up	B50	50 bps
B75	75 bps	B110	110 bps
B150	150 bps	B300	300 bps
B600	600 bps	B1200	1200 bps
B1800	1800 bps	B2400	2400 bps
B4800	4800 bps	B9600	9600 bps
B19200	19200 bps	B38400	38400 bps
B57600	57600 bps	B115200	115200 bps

lokale Einstellungen

c_lflag (Auswahl)

ECHO	Einschalten der ECHO-Funktion
ICANON	beilenorientierter Eingabemodus (kanonischer Modus)
ISIG	bestimmte Sonderzeichen lösen ein Signal aus (z. B. Ctrl-C)
XCASE	umwandeln von eingegebenen Groß- in Kleinbuchstaben

Terminal-IO-Modi

- **kanonischer Modus** (cooked mode):

Terminal-IO-Modi

- **kanonischer Modus** (cooked mode):
  - Lesen und Schreiben auf das Device zeilenorientiert

Terminal-IO-Modi

- **kanonischer Modus** (cooked mode):
  - Lesen und Schreiben auf das Device zeilenorientiert
  - Weitergabe der Eingabe erst nach dem Linefeed, NL

Terminal-IO-Modi

- **kanonischer Modus** (cooked mode):
  - Lesen und Schreiben auf das Device zeilenorientiert
  - Weitergabe der Eingabe erst nach dem Linefeed, NL
  - Vertwendung der Steuerzeichen des c_cc-Arrays

Terminal-IO-Modi

- **kanonischer Modus** (cooked mode):
  - Lesen und Schreiben auf das Device zeilenorientiert
  - Weitergabe der Eingabe erst nach dem Linefeed, NL
  - Vertwendung der Steuerzeichen des c_cc-Arrays
  - Programm wartet bis NL kommt

Terminal-IO-Modi

- **kanonischer Modus** (cooked mode):
  - Lesen und Schreiben auf das Device zeilenorientiert
  - Weitergabe der Eingabe erst nach dem Linefeed, NL
  - Vertwendung der Steuerzeichen des c_cc-Arrays
  - Programm wartet bis NL kommt
- **nichtkanonischer Modus** (raw mode):

Terminal-IO-Modi

- **kanonischer Modus** (cooked mode):
  - Lesen und Schreiben auf das Device zeilenorientiert
  - Weitergabe der Eingabe erst nach dem Linefeed, NL
  - Vertwendung der Steuerzeichen des c_cc-Arrays
  - Programm wartet bis NL kommt
- **nichtkanonischer Modus** (raw mode):
  - entweder auf eine bestimmte Anzahl von Bytes warten

Terminal-IO-Modi

- **kanonischer Modus** (cooked mode):
  - Lesen und Schreiben auf das Device zeilenorientiert
  - Weitergabe der Eingabe erst nach dem Linefeed, NL
  - Vertwendung der Steuerzeichen des c_cc-Arrays
  - Programm wartet bis NL kommt
- **nichtkanonischer Modus** (raw mode):
  - entweder auf eine bestimmte Anzahl von Bytes warten
  - oder nach einer gewissen Zeit das Lesen beenden

Terminal-IO-Modi

- **kanonischer Modus** (cooked mode):
  - Lesen und Schreiben auf das Device zeilenorientiert
  - Weitergabe der Eingabe erst nach dem Linefeed, NL
  - Vertwendung der Steuerzeichen des c_cc-Arrays
  - Programm wartet bis NL kommt
- **nichtkanonischer Modus** (raw mode):
  - entweder auf eine bestimmte Anzahl von Bytes warten
  - oder nach einer gewissen Zeit das Lesen beenden
  - gesteuert durch *c_cc[VTIME]* und *c_cc[VMIN]*

nichtkanonischer Modus 1

1. Fall: c_cc[VTIME] $>$ 0 und c_cc[VMIN] $>$ 0
read() liefert MIN Bytes, bevor die Zeit TIME abläuft oder *read()* liefert weniger als MIN Bytes, weil die Zeit TIME abgelaufen ist. Sind noch keine Daten empfangen worden, wartet *read()* auf min. ein Byte. Wenn das erste Byte gelesen wurde, läuft der Timer los, wobei jedes ankommende Byte den Timer wieder neu startet. Diese Methode ist günstig, wenn man große Datenmengen lesen, aber auch auf einzelne Zeichen reagieren muss. Aber es kann eine Blockierung stattfinden.

nichtkanonischer Modus 2

2. Fall: c_cc[VTIME] = 0 und c_cc[VMIN] > 0
read() liefert mindestens MIN Bytes, sobald diese eingetroffen sind. Dieser Modus ist günstig, wenn möglichst viele Bytes mit einem *read()* gelesen werden sollen. Andererseits kann man auch auf ein einziges Byte reagieren (MIN = 1). Ist MIN größer als die Anzahl der bei *read()* angegebenen Zeichen, wird gewartet, bis MIN Bytes gelesen, aber nur n Bytes an *read()* geliefert wurden; ein zweites *read()* liefert dann den Rest. Auch hier kommt es zur Blockierung, wenn nicht genügend Bytes eintreffen.

nichtkanonischer Modus 3

3. Fall: c_cc[VTIME] $>$ 0 und c_cc[VMIN] $=$ 0
Diese Einstellung erlaubt es, das Lesen mit Timeout zu programmieren. Sobald ein Byte eintrifft, liefert *read()* dieses ab. Wenn die Zeit TIME seit dem Aufruf von *read()* verstrichen ist, liefert *read()* 0 (gelesene Bytes) zurück.

nichtkanonischer Modus 4

4. Fall: c_cc[VTIME] = 0 und c_cc[VMIN] = 0
read liefert die Anzahl Bytes, die anliegen. Sind keine Daten vorhanden, wird sofort 0 (gelesene Bytes) zurückgegeben. Der Treiber wartet also niemals auf Daten, sondern kehrt immer sofort zurück.

Gerätedatei öffnen 1

- beim open()-Aufruf können dateiuntypische Fehler auftreten

Gerätedatei öffnen 1

- beim open()-Aufruf können dateiuntypische Fehler auftreten
- z. B. meldet der Treiber den Fehlercode EBUSY zurück (Gerät anderweit belegt)

Gerätedatei öffnen 1

- beim open()-Aufruf können dateiuntypische Fehler auftreten
- z. B. meldet der Treiber den Fehlercode EBUSY zurück (Gerät anderweit belegt)
- „blocking open“, bis die Carrier-Leitung des Modems aktiv wird

Gerätedatei öffnen 1

- beim open()-Aufruf können dateiuntypische Fehler auftreten
- z. B. meldet der Treiber den Fehlercode EBUSY zurück (Gerät anderweit belegt)
- „blocking open“, bis die Carrier-Leitung des Modems aktiv wird
- usw.

Gerätedatei öffnen 1

- beim open()-Aufruf können dateiuntypische Fehler auftreten
- z. B. meldet der Treiber den Fehlercode EBUSY zurück (Gerät anderweit belegt)
- „blocking open“, bis die Carrier-Leitung des Modems aktiv wird
- usw.
- Blockieren umgehen: beim open()-Aufruf das Flag *O_NDELAY* mitgeben

Gerätedatei öffnen 2

```
file_descr = open("/dev/ttyS0", O_RDWR | O_NDELAY | O_NOCTTY);
/*                    Modus:    read     nicht     kein con-      */
/*                              write    warten    trolling entity */
```

Nach dem Öffnen *O_NDELAY* wieder abstellen, da sonst read()-Kommandos nicht auf Daten warten, und damit ein lastintensives „busy waiting“ durchführen: *fcntl(filedescriptor, F_SETFL, O_RDWR)*.

Gerätedatei öffnen 3

```
int open_port(int port)
  {
  /*
   * Oeffnet seriellen Port
   * Gibt das Filehandle zurueck oder -1 bei Fehler
   * der Parameter port grenzt auf erlaubte Portnummern ein
   *
   * RS232-Parameter: - 19200 baud
   *                  - 8 bits/byte
   *                  - 1 stop bit
   *                  - no parity, no handshake
   */
   int fd;
   struct termios options;
   switch (port)
     {
     case 0: fd = open("/dev/ttyS0", O_RDWR | O_NOCTTY | O_NDELAY); break;
     case 1: fd = open("/dev/ttyS1", O_RDWR | O_NOCTTY | O_NDELAY); break;
     /** gegebenenfalls weitere Ports */
     default: fd = -1;
     }
```

Gerätedatei öffnen 4

```
if (fd >= 0)
  {
  /* get current options */
  fcntl(fd, F_SETFL, 0);
  if (tcgetattr(fd, &options) != 0) return(-1);

  /* Structure loeschen, ggf. vorher sichern */
  /* wichtig bei USB-Adaptern - manche "spinnen" sonst */
  bzero(&options, sizeof(options));

  cfsetspeed(&options, B19200);        /* setze 19200 bps */
```

Gerätedatei öffnen 5

```
    /* setze Optionen */
    options.c_cflag &= ~PARENB;          /* kein Paritybit */
    options.c_cflag &= ~CSTOPB;          /* 1 Stoppbit */
    options.c_cflag &= ~CSIZE;           /* 8 Datenbits */
    options.c_cflag |= CS8;
    options.c_cflag |= (CLOCAL | CREAD);/* CD-Signal ignorieren */
    /* Kein Echo, keine Steuerzeichen, keine Interrupts */
    options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
    options.c_oflag &= ~OPOST;           /* setze "raw" Input */
    options.c_cc[VMIN]  = 0;             /* warten auf min. 0 Zeichen */
    options.c_cc[VTIME] = 10;            /* Timeout 1 Sekunde */
    tcflush(fd,TCIOFLUSH);
    if (tcsetattr(fd, TCSAFLUSH, &options) != 0) return(-1);
    }
return(fd);
}
```

Gerätedatei öffnen 6

Benötigte Headerdateien (auch für die folgenden Anwendungen):

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <termio.h>
#include <termios.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

Steuerleitungen 1

Die serielle Schnittstelle besitzt neben den Datenleitungen ja noch Steuerleitungen, die sich als Ein- und Ausgabe nutzen lassen:

Pin	Bitmaske	Konstante	I/O
DTR	0002	TIOCM_DTR	→
RTS	0004	TIOCM_RTS	→
CTS	0020	TIOCM_CTS	←
CD	0040	TIOCM_CAR	←
RI	0080	TIOCM_RNG	←
DSR	0100	TIOCM_DSR	←

Steuerleitungen 2

In manchen Fällen wollen Anwendungen einzelne Kontrollleitungen gezielt auf bestimmte Pegel setzen oder einzelne Pegel abfragen.

- Tasten einlesen

Steuerleitungen 2

In manchen Fällen wollen Anwendungen einzelne Kontrollleitungen gezielt auf bestimmte Pegel setzen oder einzelne Pegel abfragen.

- Tasten einlesen
- ein Relais schalten

Steuerleitungen 2

In manchen Fällen wollen Anwendungen einzelne Kontrollleitungen gezielt auf bestimmte Pegel setzen oder einzelne Pegel abfragen.

- Tasten einlesen
- ein Relais schalten
- Abfragen/Setzen mittels ioctl()-Aufruf

Steuerleitungen 2

In manchen Fällen wollen Anwendungen einzelne Kontrollleitungen gezielt auf bestimmte Pegel setzen oder einzelne Pegel abfragen.

- Tasten einlesen
- ein Relais schalten
- Abfragen/Setzen mittels ioctl()-Aufruf
- Abfragen: ioctl(fd, TIOCMGET, &currstat)

Steuerleitungen 2

In manchen Fällen wollen Anwendungen einzelne Kontrollleitungen gezielt auf bestimmte Pegel setzen oder einzelne Pegel abfragen.

- Tasten einlesen
- ein Relais schalten
- Abfragen/Setzen mittels ioctl()-Aufruf
- Abfragen: ioctl(fd, TIOCMGET, &currstat)
- Setzen: ioctl(fd, TIOCMSET, &currstat)

Steuerleitungen 2

In manchen Fällen wollen Anwendungen einzelne Kontrollleitungen gezielt auf bestimmte Pegel setzen oder einzelne Pegel abfragen.

- Tasten einlesen
- ein Relais schalten
- Abfragen/Setzen mittels ioctl()-Aufruf
- Abfragen: ioctl(fd, TIOCMGET, &currstat)
- Setzen: ioctl(fd, TIOCMSET, &currstat)
- Konstante für Eingänge: TIOCM_RNG, TIOCM_CAR, TIOCM_DSR, TIOCM_CTS

Steuerleitungen 2

In manchen Fällen wollen Anwendungen einzelne Kontrollleitungen gezielt auf bestimmte Pegel setzen oder einzelne Pegel abfragen.

- Tasten einlesen
- ein Relais schalten
- Abfragen/Setzen mittels ioctl()-Aufruf
- Abfragen: ioctl(fd, TIOCMGET, &currstat)
- Setzen: ioctl(fd, TIOCMSET, &currstat)
- Konstante für Eingänge: TIOCM_RNG, TIOCM_CAR, TIOCM_DSR, TIOCM_CTS
- Konstante für Ausgänge: TIOCM_RTS, TIOCM_DTR

Steuerleitungen 3

Setzen/Rücksetzen von RTS (DTR analog):

```
void SetRTS(int fd)
/* Setzt RTS auf ON */
  {
  int currstat;
  ioctl(fd, TIOCMGET, &currstat);
  currstat |= TIOCM_RTS;
  ioctl(fd, TIOCMSET, &currstat);
  }

void ResetRTS(int fd)
/* Setzt RTS auf OFF */
  {
  int currstat;
  ioctl(fd, TIOCMGET, &currstat);
  currstat &= ~TIOCM_RTS;
  ioctl(fd, TIOCMSET, &currstat);
  }
```

Steuerleitungen 4

Lesen von RI (CD, DSR, CTS analog):

```
int GetRI(int fd)
/* Gibt 1 zurueck, wenn RI gesetzt ist, sonst 0 */
  {
  int  currstat;
  ioctl(fd, TIOCMGET, &currstat);
  return((currstat & TIOCM_RNG)?1:0);
  }
```

Lesen Status gesamt:

```
int GetSerStat(int fd)
/* Gibt den kompletten seriellen Status zurueck *
 * ggf. RTS und DTR ausblenden:                 *
 * currstat &=  (TIOCM_DTR | TOICM_RTS)         */
  {
  int  currstat;
  ioctl(fd, TIOCMGET, &currstat);
  return(currstat);
  }
```

Mehr unter

`http://www.netzmafia.de/skripten/hardware/Seriell/`

PC-Hardware und Linux

Ansteuern der Schnittstellen

Jürgen Plate

14. März 2012

Prolog 1

- zu wenige Eingangsleitungen am Parallelport?

Prolog 1

- zu wenige Eingangsleitungen am Parallelport?
- dann „schlachten“ Sie doch einfach eine Tastatur

Prolog 1

- zu wenige Eingangsleitungen am Parallelport?
- dann „schlachten“ Sie doch einfach eine Tastatur
- am besten eine USB-Zusatztastatur, die parallel anschließbar ist

Prolog 1

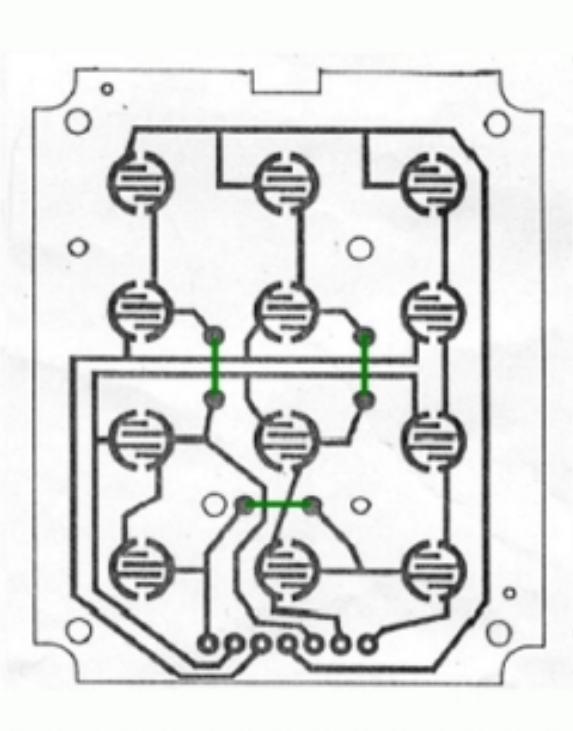
- zu wenige Eingangsleitungen am Parallelport?
- dann „schlachten“ Sie doch einfach eine Tastatur
- am besten eine USB-Zusatztastatur, die parallel anschließbar ist
- Kosten am 5 Euro (z. B. von Pollin)

Prolog 1

- zu wenige Eingangsleitungen am Parallelport?
- dann „schlachten“ Sie doch einfach eine Tastatur
- am besten eine USB-Zusatztastatur, die parallel anschließbar ist
- Kosten am 5 Euro (z. B. von Pollin)
- es geht aber auch eine alte PS2-Tastatur mit USB-Adapter

Prolog 2

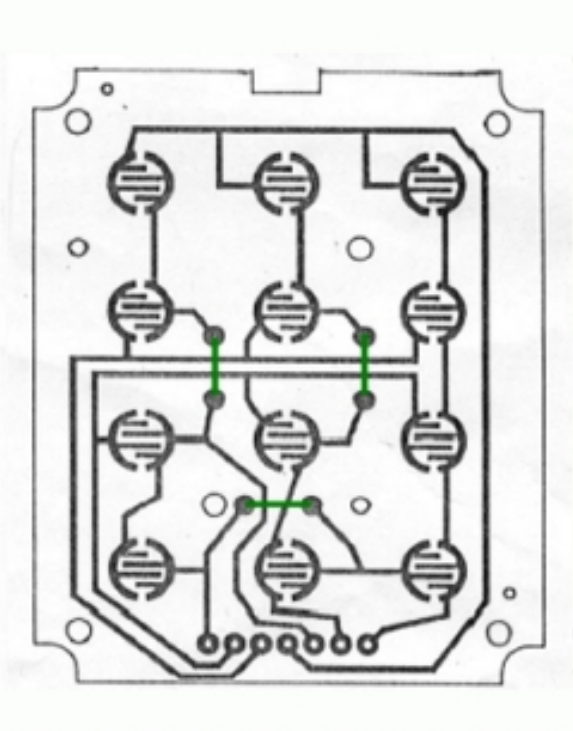
Eine typische Matrix sieht folgendermaßen aus:



- einziger Nachteil: die Tasten sind als Matrix angeordnet

Prolog 2

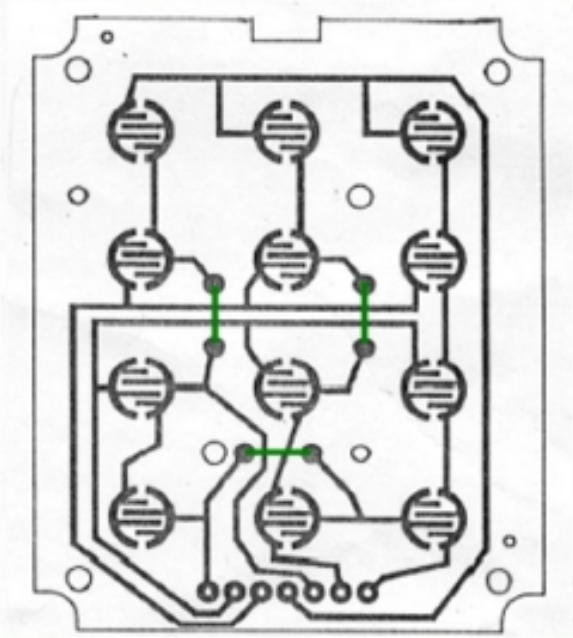
Eine typische Matrix sieht folgendermaßen aus:



- einziger Nachteil: die Tasten sind als Matrix angeordnet
- $\rightarrow$ immer zwei Leitungen am Kreuzungspunkt zu verbinden

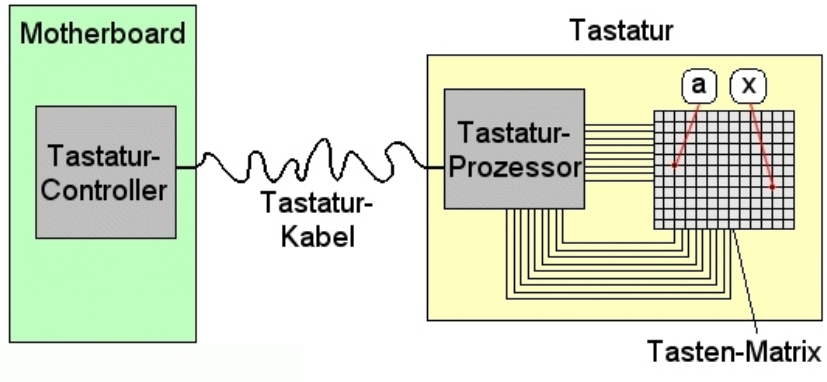
Prolog 2

Eine typische Matrix sieht folgendermaßen aus:



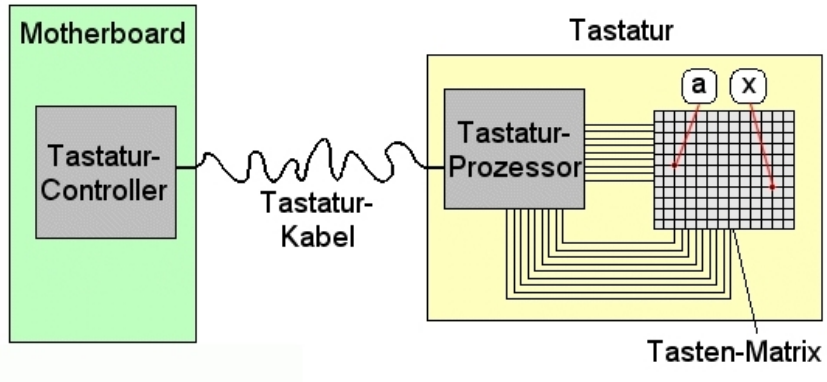
- einziger Nachteil: die Tasten sind als Matrix angeordnet
- $\rightarrow$ immer zwei Leitungen am Kreuzungspunkt zu verbinden
- $\rightarrow$ kein gemeinsamer Massepegel

Die PC-Tastatur



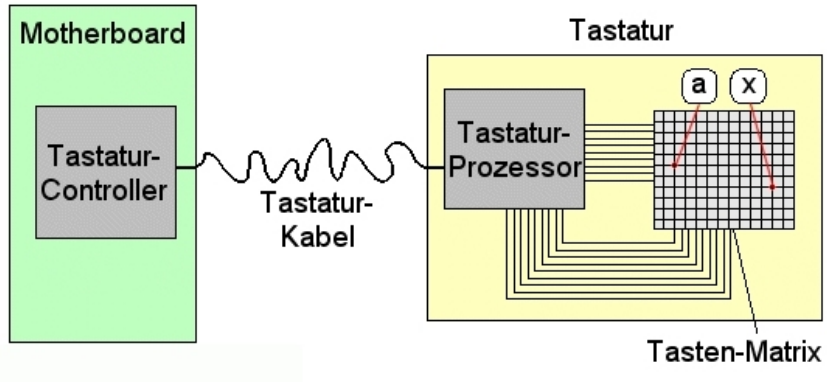
- auf dem Motherboard befindet sich ein Tastaturcontroller

Die PC-Tastatur



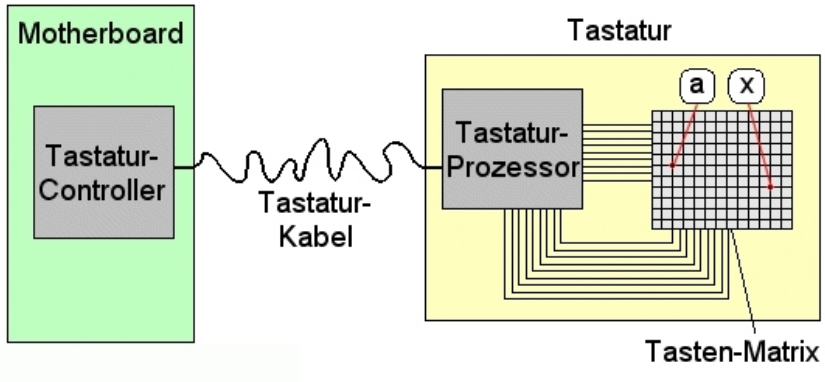
- auf dem Motherboard befindet sich ein Tastaturcontroller
- in der Tastatur befindet sich ein Tastatur-Prozessor und die Tastenmatrix

Die PC-Tastatur



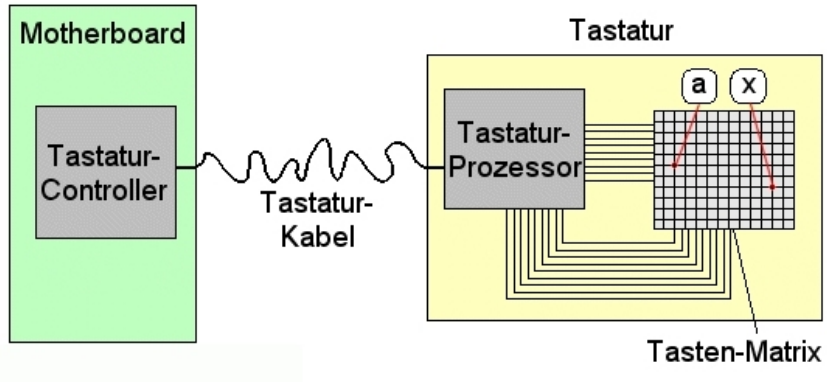
- auf dem Motherboard befindet sich ein Tastaturcontroller
- in der Tastatur befindet sich ein Tastatur-Prozessor und die Tastenmatrix
- beide kommunizieren seriell (PS2-Schnittstelle oder USB)

Die PC-Tastatur



- auf dem Motherboard befindet sich ein Tastaturcontroller
- in der Tastatur befindet sich ein Tastatur-Prozessor und die Tastenmatrix
- beide kommunizieren seriell (PS2-Schnittstelle oder USB)
- Tastatur sendet Codes beim Drücken und Loslassen einer Taste

Die PC-Tastatur



- auf dem Motherboard befindet sich ein Tastaturcontroller
- in der Tastatur befindet sich ein Tastatur-Prozessor und die Tastenmatrix
- beide kommunizieren seriell (PS2-Schnittstelle oder USB)
- Tastatur sendet Codes beim Drücken und Loslassen einer Taste
- Bei PS/2-PCs hängt die Maus auch am Tastaturcontroller

Programmierung 1

- Tastaturcontroller programmierbar über das Senden von Befehlscodes

Programmierung 1

- Tastaturcontroller programmierbar über das Senden von Befehlscodes
- Danach warten auf Quittungsbyte (oxFA)

Programmierung 1

- Tastaturcontroller programmierbar über das Senden von Befehlscodes
- Danach warten auf Quittungsbyte (oxFA)
- Anwendung hier: Setzen/Rücksetzen der Tastatur-LEDs

Programmierung 1

- Tastaturcontroller programmierbar über das Senden von Befehlscodes
- Danach warten auf Quittungsbyte (oxFA)
- Anwendung hier: Setzen/Rücksetzen der Tastatur-LEDs
- weitere Parameter programmierbar (z. B. Wiederholrate

Programmierung 1

- Tastaturcontroller programmierbar über das Senden von Befehlscodes
- Danach warten auf Quittungsbyte (oxFA)
- Anwendung hier: Setzen/Rücksetzen der Tastatur-LEDs
- weitere Parameter programmierbar (z. B. Wiederholrate
- unter Linux unter Linux mit Kernel-Unterstützung per *ioctl()*

Programmierung 2

- Befehlscode: 0xED (set_led)

Programmierung 2

- Befehlscode: 0xED (set_led)
- Daten: 1 Byte, wobei nur die Bist 0 – 2 belegt sind

Programmierung 2

- Befehlscode: 0xED (set_led)
- Daten: 1 Byte, wobei nur die Bist 0 – 2 belegt sind
- Bit 0: Scroll Lock, Bit 1: Caps Lock, Bit 2: Num Lock

Programmierung 2

- Befehlscode: 0xED (set_led)
- Daten: 1 Byte, wobei nur die Bist 0 – 2 belegt sind
- Bit 0: Scroll Lock, Bit 1: Caps Lock, Bit 2: Num Lock
- die eigentlichen Tastaturfunktionen werden **nicht** beeinflusst

Programmierung 2

- Befehlscode: 0xED (set_led)
- Daten: 1 Byte, wobei nur die Bist 0 – 2 belegt sind
- Bit 0: Scroll Lock, Bit 1: Caps Lock, Bit 2: Num Lock
- die eigentlichen Tastaturfunktionen werden **nicht** beeinflusst
- Zugriff auf */dev/console* bzw. */dev/tty0*

Programmierung 2

- Befehlscode: 0xED (set_led)
- Daten: 1 Byte, wobei nur die Bist 0 – 2 belegt sind
- Bit 0: Scroll Lock, Bit 1: Caps Lock, Bit 2: Num Lock
- die eigentlichen Tastaturfunktionen werden **nicht** beeinflusst
- Zugriff auf */dev/console* bzw. */dev/tty0*
- nur mit Root-Rechten

Programmierung 3

Headerdateien:

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <linux/kd.h>
#include <unistd.h>
#include <stdlib.h>
```

Definitionen:

```
#define SCRLED 1
#define NUMLED 2
#define CAPLED 4
```

Programmierung 4

Programm: LEDs für vorgegebene Zeit setzen/rücksetzen

```
int main(int argc, char *argv[])
  {
  int fd, time;
  char leds, oldleds;
  fd = open("/dev/console",O_RDONLY);
  leds = atoi(argv[1]);
  time = atoi(argv[2]);
  leds = leds & 7;                /* zur Sicherheit */
  time = time % 10000;            /* max. 10 s */
  ioctl(fd, KDGETLED, &oldleds); /* alten Wert merken */
  ioctl(fd, KDSETLED, leds);      /* Ausgabe */
  usleep(1000*time);
  ioctl(fd, KDSETLED, oldleds);  /* alten Wert restaurieren */
  close(fd);
  return(0);
  }
```

Programmierung 5

Programm: LEDs als Binärzähler

```
int main(void)
  {
  int fd, i;
  int leds[] = {0,1,4,5,2,3,6,7};  /* binary counter */
  fd = open("/dev/console",O_RDONLY);

  while (1)
    {
    i = (i + 1) % 8;
    ioctl(fd, KDSETLED, leds[i]);
    sleep(1);
    }
  close(fd);
  return(0);
  }
```

Mehr unter `http://www.netzmafia.de/skripten/hardware/Keyboard/tastatur-leds.html`

PC-Hardware und Linux

Ansteuern der Schnittstellen

Jürgen Plate

14. März 2012

Prolog

- der interne Lautsprecher wird selten beachtet

Prolog

- der interne Lautsprecher wird selten beachtet
- denn es gibt ja die Soundkarte

Prolog

- der interne Lautsprecher wird selten beachtet
- denn es gibt ja die Soundkarte
- Server haben normalerweise keine Soundkarte

Prolog

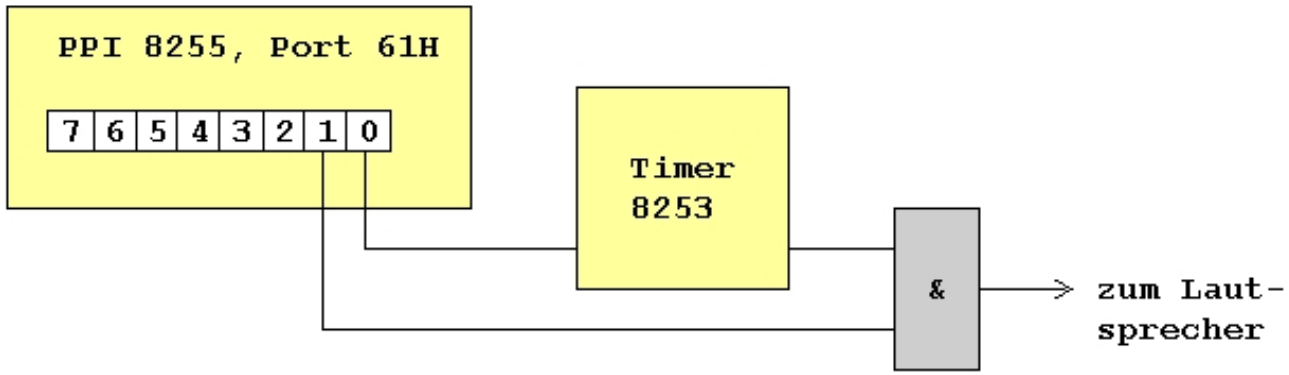
- der interne Lautsprecher wird selten beachtet
- denn es gibt ja die Soundkarte
- Server haben normalerweise keine Soundkarte
- und oft auch keinen Monitor und keine Tastatur

Prolog

- der interne Lautsprecher wird selten beachtet
- denn es gibt ja die Soundkarte
- Server haben normalerweise keine Soundkarte
- und oft auch keinen Monitor und keine Tastatur
- sie können oft nur kommunizieren wie R2-D2

Wie funktioniert's?

Der Lautsprecher wird durch zwei Leitungen des PPI 8255 gesteuert. Dessen Ausgangsbits 0 und 1 lassen sich über die Portadresse 61H ansprechen. Bit 1 steuert den Timer, Bit 0 die Freigabe des Timerausgangs.



Die Ausgangsfrequenz

- der Timer wird über die Ports 43H und 42H angesprochen

Die Ausgangsfrequenz

- der Timer wird über die Ports 43H und 42H angesprochen
- die Frequenz wird durch einen 16-Bit-Teilfaktor bestimmt

Die Ausgangsfrequenz

- der Timer wird über die Ports 43H und 42H angesprochen
- die Frequenz wird durch einen 16-Bit-Teilfaktor bestimmt
- Teilfaktor = 133100/Frequenz

Die Ausgangsfrequenz

- der Timer wird über die Ports 43H und 42H angesprochen
- die Frequenz wird durch einen 16-Bit-Teilfaktor bestimmt
- Teilfaktor = 133100/Frequenz
- Zugriff von Linux aus per *ioctl()* auf */dev/console*

ioctl()-Requests

- KDMKTONE generiert einen Ton für eine vordefinierte Zeit

ioctl()-Requests

- KDMKTONE generiert einen Ton für eine vordefinierte Zeit
- $\rightarrow$ *ioctl (fd, KDMKTONE,(long) argument)*

ioctl()-Requests

- KDMKTONE generiert einen Ton für eine vordefinierte Zeit
- $\rightarrow$ *ioctl (fd, KDMKTONE,(long) argument)*
- KIOCSOUND generiert einen Dauerton oder stoppt den Sound

ioctl()-Requests

- KDMKTONE generiert einen Ton für eine vordefinierte Zeit
- → *ioctl (fd, KDMKTONE,(long) argument)*
- KIOCSOUND generiert einen Dauerton oder stoppt den Sound
- → *ioctl(fd,KIOCSOUND,(int) tone)*

ioctl()-Requests

- KDMKTONE generiert einen Ton für eine vordefinierte Zeit
- $\rightarrow$ *ioctl (fd, KDMKTONE,(long) argument)*
- KIOCSOUND generiert einen Dauerton oder stoppt den Sound
- $\rightarrow$ *ioctl(fd,KIOCSOUND,(int) tone)*
- *tone*: MSB = Dauer (in Timer-Ticks), LSB = Tonhöhe

ioctl()-Requests

- KDMKTONE generiert einen Ton für eine vordefinierte Zeit
- → *ioctl (fd, KDMKTONE,(long) argument)*
- KIOCSOUND generiert einen Dauerton oder stoppt den Sound
- → *ioctl(fd,KIOCSOUND,(int) tone)*
- *tone*: MSB = Dauer (in Timer-Ticks), LSB = Tonhöhe
- Tonhöhe 0 = Stille

ioctl()-Requests

- KDMKTONE generiert einen Ton für eine vordefinierte Zeit
- → *ioctl (fd, KDMKTONE,(long) argument)*
- KIOCSOUND generiert einen Dauerton oder stoppt den Sound
- → *ioctl(fd,KIOCSOUND,(int) tone)*
- *tone*: MSB = Dauer (in Timer-Ticks), LSB = Tonhöhe
- Tonhöhe 0 = Stille
- Tonerzeugung ist nicht blockierend

Tonerzeugung

- Timer wird mit 1,19 MHz getaktet

Tonerzeugung

- Timer wird mit 1,19 MHz getaktet
- $\rightarrow$ Tonhöhe entspricht nicht der Frequenz

Tonerzeugung

- Timer wird mit 1,19 MHz getaktet
- $\rightarrow$ Tonhöhe entspricht nicht der Frequenz
- $\rightarrow$ Tonhöhe 1190000/Frequenz

Tonerzeugung

- Timer wird mit 1,19 MHz getaktet
- $\rightarrow$ Tonhöhe entspricht nicht der Frequenz
- $\rightarrow$ Tonhöhe 1190000/Frequenz
- KDMKTONE eignet sich für Warnsignale

Tonerzeugung

- Timer wird mit 1,19 MHz getaktet
- $\rightarrow$ Tonhöhe entspricht nicht der Frequenz
- $\rightarrow$ Tonhöhe 1190000/Frequenz
- KDMKTONE eignet sich für Warnsignale
- KIOCSOUND eignet sich für Melodien

Tonerzeugung

- Timer wird mit 1,19 MHz getaktet
- $\rightarrow$ Tonhöhe entspricht nicht der Frequenz
- $\rightarrow$ Tonhöhe 1190000/Frequenz
- KDMKTONE eignet sich für Warnsignale
- KIOCSOUND eignet sich für Melodien
- nur mit Root-Rechten

Töne und Noten

Die folgende Frequenztabelle gibt die Frequenzen der Noten für 7 Oktaven an:

C	C#	D	D#	E	F	F#	G	G#	A	A#	B
65,	69,	73,	78,	82,	87,	92,	98,	104,	110,	116,	123,
131,	139,	147,	156,	165,	175,	185,	196,	208,	220,	233,	247,
262,	278,	294,	312,	330,	350,	370,	392,	416,	440,	466,	494,
524,	556,	588,	624,	660,	700,	740,	784,	832,	880,	932,	988,
1048,	1112,	1176,	1248,	1320,	1400,	1480,	1568,	1664,	1760,	1864,	1976,
2096,	2224,	2352,	2496,	2640,	2800,	2960,	3136,	3328,	3520,	3728,	3952,
4192,	4448,	4704,	4992,	5280,	5600,	5920,	6272,	6656,	7040,	7456,	7904

Programmierung 1

Headerdateien:

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <linux/kd.h>
#include <unistd.h>
#include <stdlib.h>
```

Programmierung 2

Das Programm *beep(int freq, int duration)* erzeugt auf dem internen Lautsprecher einen Ton der Frequenz freq [Hz] und der Dauer duration [ms]

```
int main (int argc, char *argv[])
  {
  int fd;
  long duration, freq, res;
  fd = open("/dev/console",O_RDONLY);
  if (argc > 2)
    {
    freq = atoi(argv[1]);
    duration = atoi(argv[2]);
    if (freq > 0)
      {
      res = (duration << 16) + (1193180/freq);
      ioctl(fd,KDMKTONE,res);
      }
    usleep(duration*1000);
    }
  close(fd);
  return(0);
  }
```

Mehr unter `http://www.netzmafia.de/skripten/hardware/Speaker/simplesound.html`