

Übung zu Betriebssysteme

Ein- und Ausgabe

24. & 25. Oktober 2019

Andreas Ziegler, Bernhard Heinloth, Christian Eichler

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



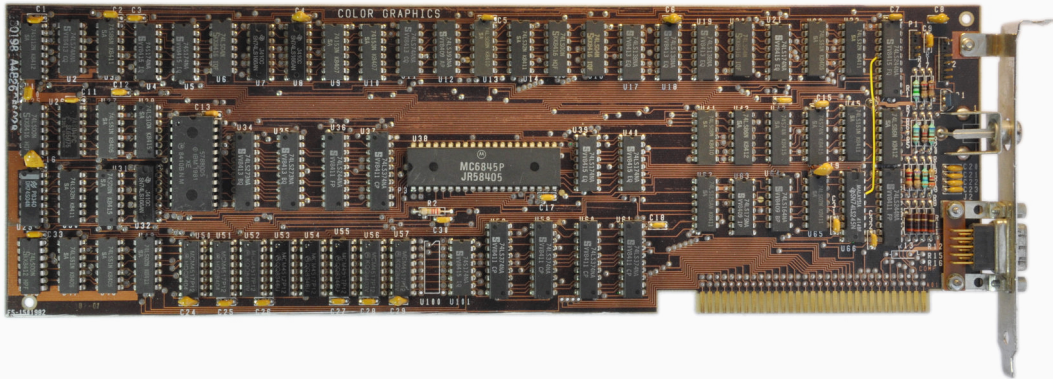
Lehrstuhl für Verteilte Systeme
und Betriebssysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Color Graphics Adapter



Quelle: Wikipedia

Der CGA Bildschirm ...

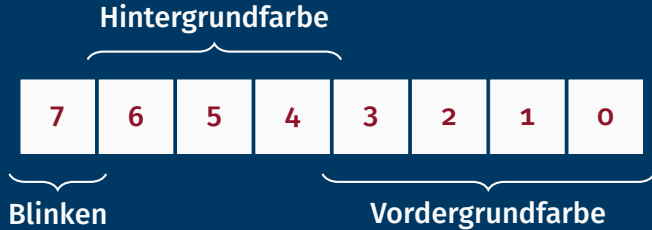


... zwei Bytes pro Zeichen ...

1. Byte: **ASCII Zeichen**
2. Byte: **Darstellungsattribut**

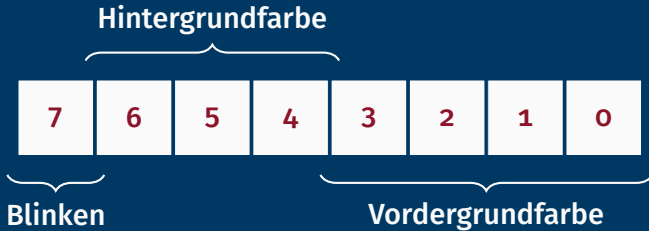
... zwei Bytes pro Zeichen ...

- 1. Byte: ASCII Zeichen
- 2. Byte: Darstellungsattribut



... zwei Bytes pro Zeichen ...

- 1. Byte: ASCII Zeichen
- 2. Byte: Darstellungsattribut



0	schwarz	4	rot	8	dunkelgrau	12	hellrot
1	blau	5	magenta	9	hellblau	13	hellmagenta
2	grün	6	braun	10	hellgrün	14	gelb
3	cyan	7	hellgrau	11	hellcyan	15	weiß

...eingebildet im Arbeitsspeicher ab 0xb8000

⋮	
'B'	← 0xb8000
0xf4	← 0xb8001
'a'	← 0xb8002
0x74	← 0xb8003
'r'	← 0xb8004
0x74	← 0xb8005
⋮	

Bar

ar

Rekapitulation: Bitshiftoperationen

- $\&$ Konjunktion (bitweises UND)
- $|$ Disjunktion (bitweises ODER)
- \wedge exklusives ODER
- \sim Einerkomplement (bitweise Negation)
- \ll verschieben nach links (Multiplikation mit 2)
- \gg verschieben nach rechts (Division durch 2)

Ansprechen von einzelnen Bits

- Bitmasken und logischen Bitoperationen

Ansprechen von einzelnen Bits

- Bitmasken und logischen Bitoperationen
- Bitfelder in C/C++

```
struct cga_attrib {  
    unsigned char fg : 4,  
                 bg : 3,  
                 bl : 1;  
};
```

```
struct cga_attrib a;  
a.fg = 4; // rot  
a.bg = 7; // hellgrau  
a.bl = 1; // blinken
```

Ansprechen von einzelnen Bits

- Bitmasken und logischen Bitoperationen
- Bitfelder in C/C++

```
struct cga_attrib {  
    unsigned char fg : 4,  
                 bg : 3,  
                 bl : 1;  
};
```

```
struct cga_attrib a;  
a.fg = 4; // rot  
a.bg = 7; // hellgrau  
a.bl = 1; // blinken
```

Layout mit GCC auf x86:



Einschub: Füllen von Strukturen

```
struct Test {  
    char foo[3];  
    int bar;  
    bool baz;  
};  
  
cout << sizeof(Test);
```

Einschub: Füllen von Strukturen

```
struct Test {  
    char foo[3]; // 3 Byte  
    int bar;     // 4 Byte  
    bool baz;   // 1 Byte  
};  
  
cout << sizeof(Test); // "12"
```

Einschub: Füllen von Strukturen

```
struct Test {  
    char foo[3]; // 3 Byte  
    int bar;     // 4 Byte  
    bool baz;   // 1 Byte  
} __attribute__((packed));  
  
cout << sizeof(Test); // "8"
```

Einschub: Füllen von Strukturen

```
struct Test {  
    char foo[3]; // 3 Byte  
    int bar;     // 4 Byte  
    bool baz;    // 1 Byte  
} __attribute__((packed));  
  
static_assert(sizeof(Test) == 8, "Test kaputt");
```

Schreibmarke (Cursor)

Schreibmarke (Cursor)_

Entweder Cursorposition in Software merken...

... oder Cursorposition aus Hardware auslesen

... oder Cursorposition aus Hardware auslesen

- CGA hat 18 Steuerregister mit je 8 Bit

0	Horizontal Total	w
1	Horizontal Displayed	w
2	H. Sync Position	w
3	H. Sync Width	w
4	Vertical Total	w
5	V. Total Adjust	w
6	Vertical Displayed	w
7	V. Sync Position	w
8	Interlace Mode	w
9	Max Scan Line Address	w
10	Cursor Start	w
11	Cursor End	w
12	Start Address (high)	w
13	Start Address (low)	w
14	Cursor (high)	rw
15	Cursor (low)	rw
16	Light Pen (high)	r
17	Light Pen (low)	r

... oder Cursorposition aus Hardware auslesen

- CGA hat 18 Steuerregister mit je 8 Bit
- Position in Register 14 (high) und 15 (low)

0	Horizontal Total	w
1	Horizontal Displayed	w
2	H. Sync Position	w
3	H. Sync Width	w
4	Vertical Total	w
5	V. Total Adjust	w
6	Vertical Displayed	w
7	V. Sync Position	w
8	Interlace Mode	w
9	Max Scan Line Address	w
10	Cursor Start	w
11	Cursor End	w
12	Start Address (high)	w
13	Start Address (low)	w
14	Cursor (high)	rw
15	Cursor (low)	rw
16	Light Pen (high)	r
17	Light Pen (low)	r

... oder Cursorposition aus Hardware auslesen

- CGA hat 18 Steuerregister mit je 8 Bit
- Position in Register 14 (high) und 15 (low)
- Nur indirekter Zugriff über Index- (0x3d4) und Datenregister (0x3d5)

0	Horizontal Total	w
1	Horizontal Displayed	w
2	H. Sync Position	w
3	H. Sync Width	w
4	Vertical Total	w
5	V. Total Adjust	w
6	Vertical Displayed	w
7	V. Sync Position	w
8	Interlace Mode	w
9	Max Scan Line Address	w
10	Cursor Start	w
11	Cursor End	w
12	Start Address (high)	w
13	Start Address (low)	w
14	Cursor (high)	rw
15	Cursor (low)	rw
16	Light Pen (high)	r
17	Light Pen (low)	r

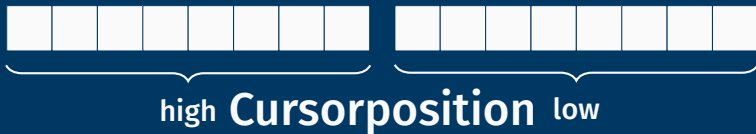
0x3d4

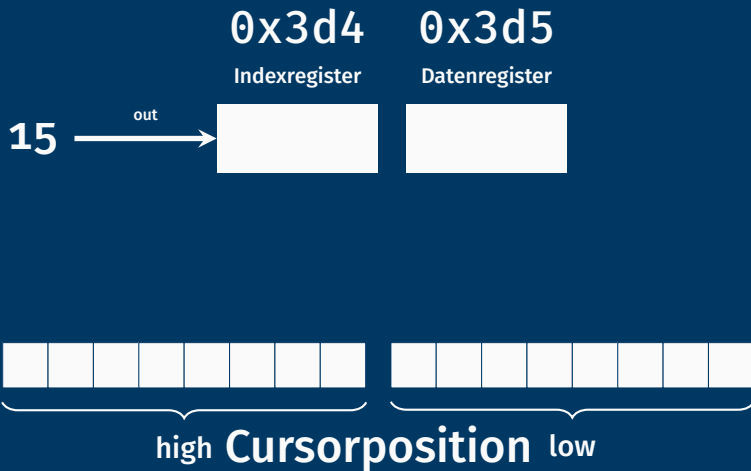
Indexregister



0x3d5

Datenregister





0x3d4

Indexregister

15

0x3d5

Datenregister

0x55

in



high **Cursorposition** low



0x3d4

Indexregister

14

0x3d5

Datenregister

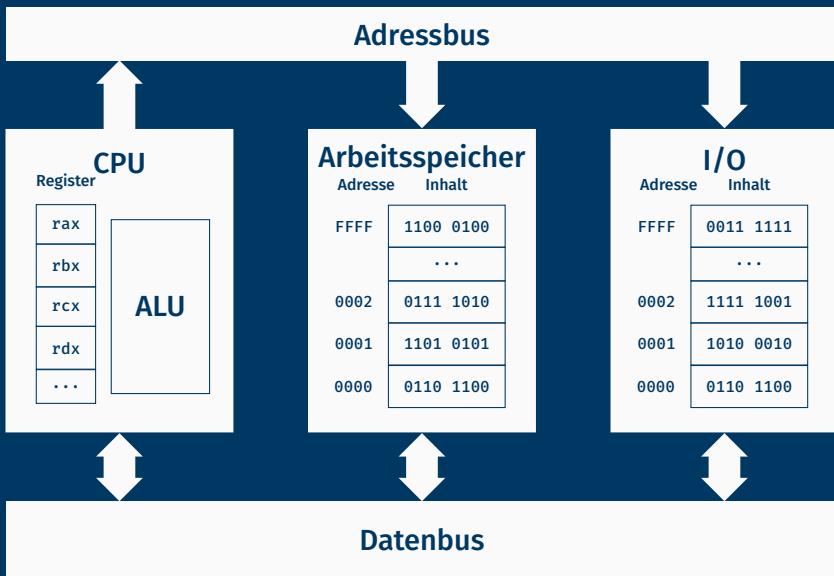
0x0a

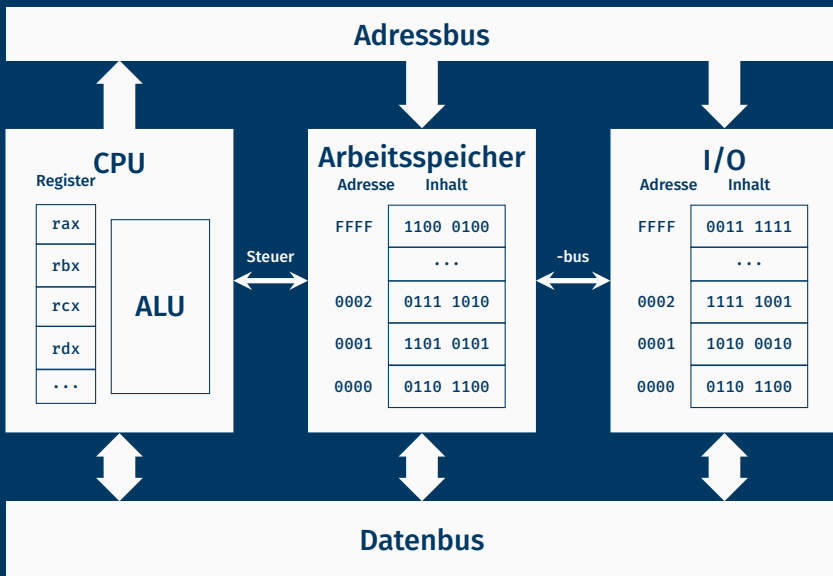
in



high **Cursorposition** low

Kleiner Exkurs:
Zugriff auf Arbeitsspeicher vs. I/O Ports



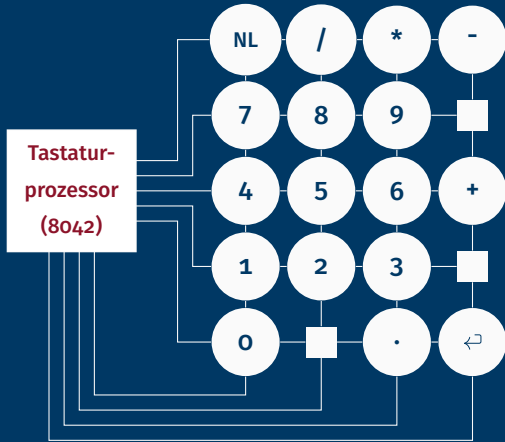


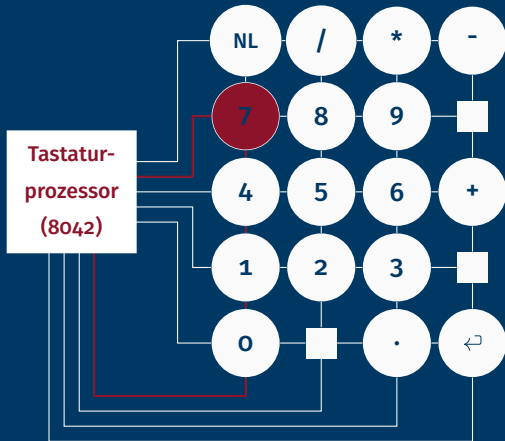
Tastatur

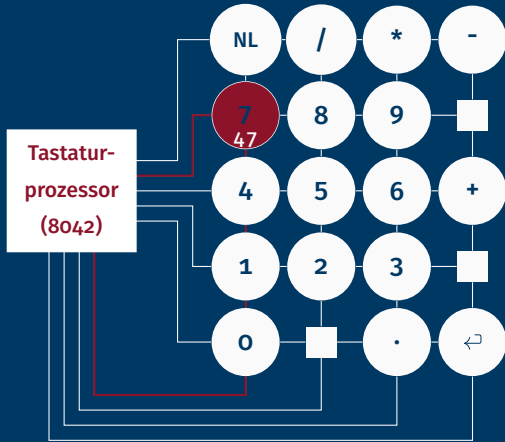


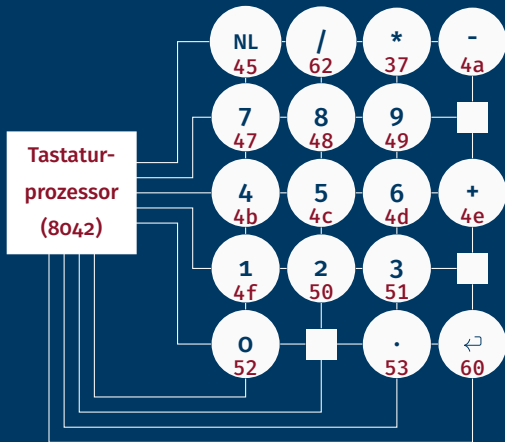
Quelle: Golem

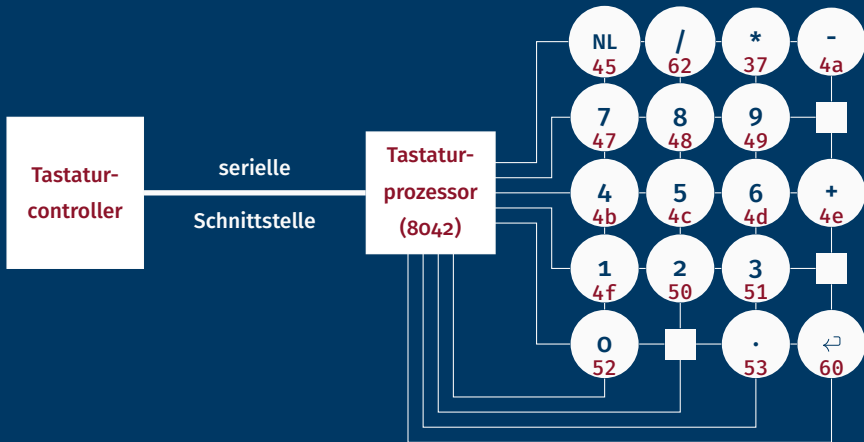


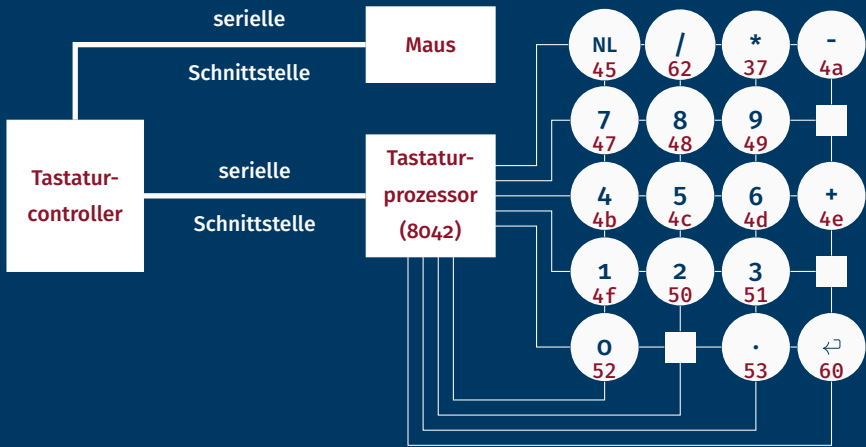












ASCII: Darstellung von Zeichen, 8 Bit

SCANCODE: Tastenkennung, 7 Bit

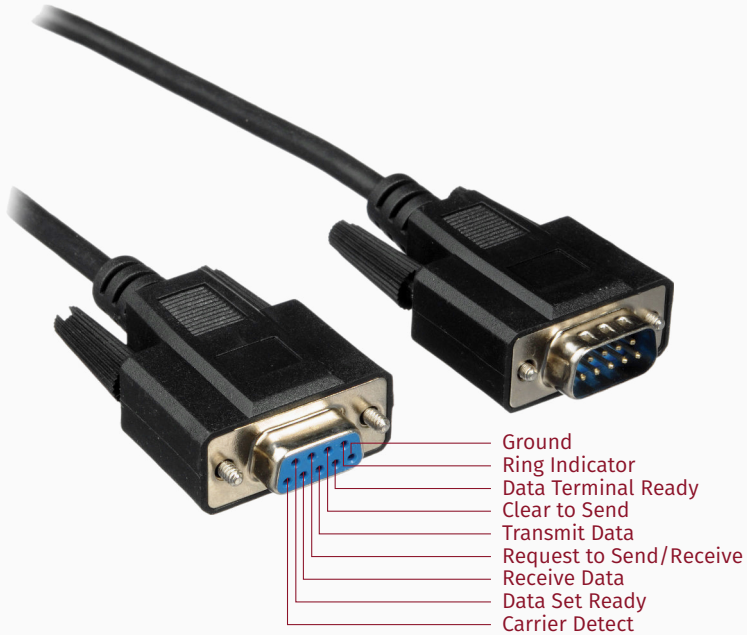
MAKECODE: Tastendruck (Scancode + 0x80)

BREAKCODE: Taste loslassen

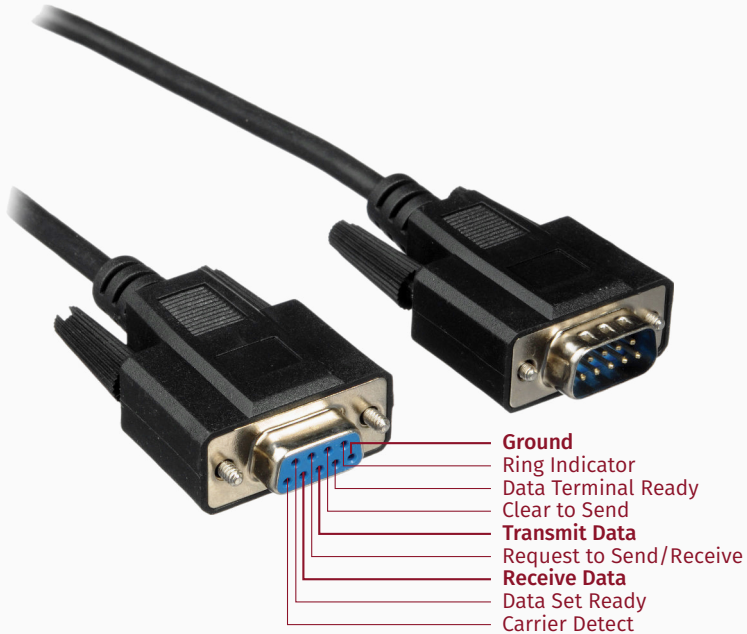
Serielle Schnittstelle



Quelle: B&H Photo Video

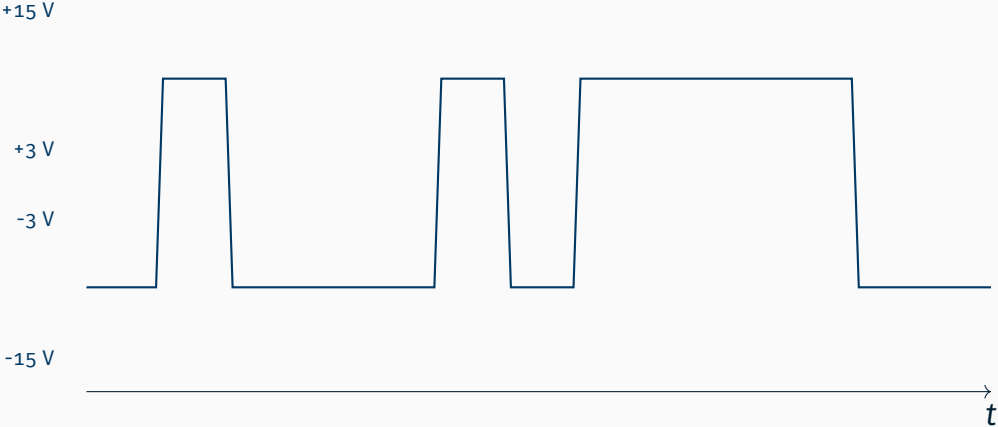


Quelle: B&H Photo Video

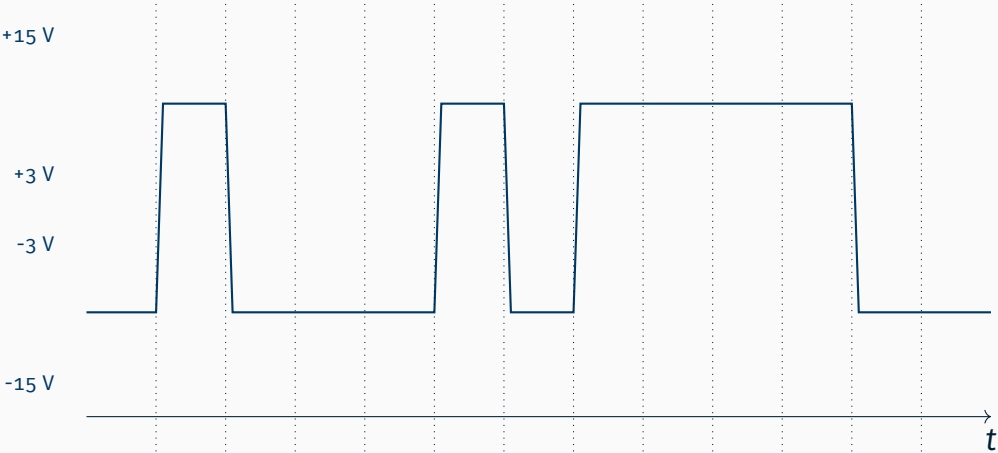


Quelle: B&H Photo Video

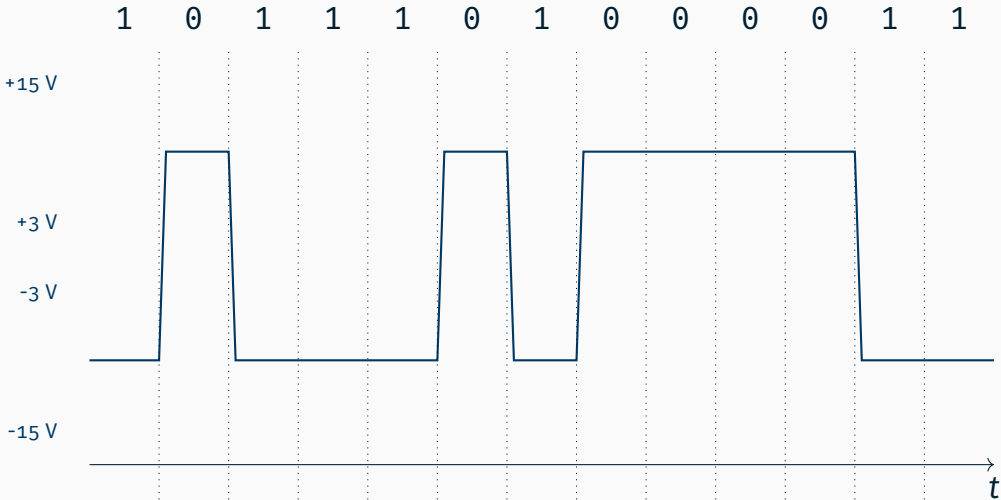
RS232



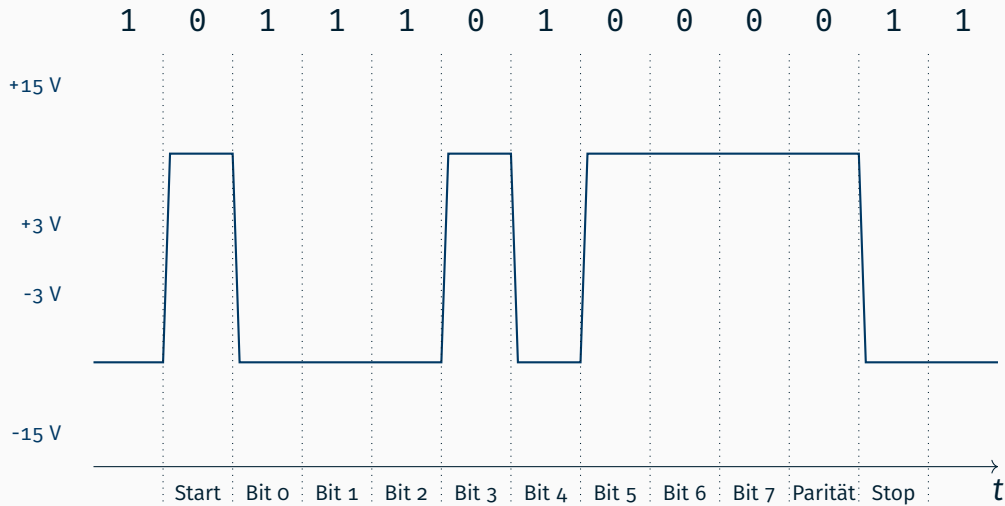
RS232



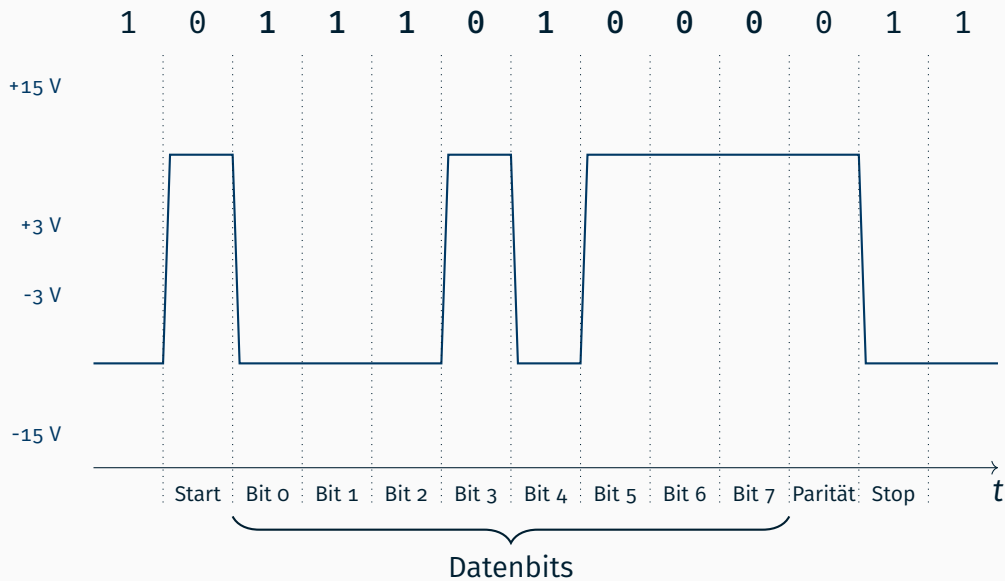
RS232



RS232



RS232



Serielle Schnittstelle

- Übertragungsrate (Baud rate) ist Teiler von 115 200 Hz
- Kommunikation 8-N-1 mit 9 600 Baud oft Standardeinstellung
 - 8** Anzahl der Datenbits
 - N** kein Paritätsbit
 - 1** Stopbit
- Aktuelle PCs haben derzeit meist maximal eine Hardwareschnittstelle (COM1)
- Controller wird über I/O-Ports programmiert

Serielle Schnittstelle (I/O-Ports)

- Übliche Basisadressen

0x3f8 COM1

0x2f8 COM2

0x3e8 COM3

0x2e8 COM4

Serielle Schnittstelle (I/O-Ports)

- Übliche Basisadressen

 - 0x3f8 COM1

 - 0x2f8 COM2

 - 0x3e8 COM3

 - 0x2e8 COM4

- 12 Register über 8 Offsetadressen

 - 0 Daten (empfangen / senden)

 - 1 Interrupt aktiviert / Teiler niederwertig

 - 2 Interruptregistration / FIFO-Control / Teiler höchstwertig

 - 3 Line-Control

 - 4 Modem-Control

 - 5 Line-Status

 - 6 Modem-Status

 - 7 Scratch

Serielle Schnittstelle (I/O-Ports)

- Übliche Basisadressen

 - 0x3f8 COM1

 - 0x2f8 COM2

 - 0x3e8 COM3

 - 0x2e8 COM4

- 12 Register über 8 Offsetadressen

 - 0 Daten (empfangen / senden)

 - 1 Interrupt aktiviert / Teiler niederwertig

 - 2 Interruptregistration / FIFO-Control / Teiler höchstwertig

 - 3 Line-Control

 - 4 Modem-Control

 - 5 Line-Status

 - 6 Modem-Status

 - 7 Scratch

- Details auf osdev.org und lowlevel.eu

Terminal



ANSI-Escape-Sequenzen

- SteuerCodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')

ANSI-Escape-Sequenzen

- SteuerCodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle

\e[Am Attribute

- 0 keine
- 1 fett
- 2 matt
- 3 kursiv *
- 4 unterstrichen
- 5 blinkend
- 6 blinkend (schnell) *
- 7 invertiert
- 8 unsichtbar *

* wird nur selten unterstützt

ANSI-Escape-Sequenzen

- SteuerCodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle

\e[Am Attribute

\e[3Cm Vordergrundfarbe

\e[4Cm Hintergrundfarbe

- 0 schwarz
- 1 rot
- 2 grün
- 3 gelb
- 4 blau
- 5 magenta
- 6 cyan
- 7 weiß
- 9 *Standardfarbe*

ANSI-Escape-Sequenzen

- SteuerCodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle
 - \e[A m Attribute
 - \e[3C m Vordergrundfarbe
 - \e[4C m Hintergrundfarbe
 - \e[Y;X H Cursorposition setzen

ANSI-Escape-Sequenzen

- Steuercodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle
 - \e[A**m** Attribute
 - \e[3**C**m Vordergrundfarbe
 - \e[4**C**m Hintergrundfarbe
 - \e[**Y**;**X**H Cursorposition setzen
 - \e[6n Cursorposition lesen
 - \e[**Y**;**X**R Antwort

ANSI-Escape-Sequenzen

- SteuerCodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle
 - \e[A**m** Attribute
 - \e[3**C**m Vordergrundfarbe
 - \e[4**C**m Hintergrundfarbe
 - \e[**Y**;**X**H Cursorposition setzen
 - \e[6n Cursorposition lesen
 - \e[**Y**;**X**R Antwort
 - \ec Reset

ANSI-Escape-Sequenzen

- SteuerCodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle
 - \e[A**m** Attribute
 - \e[3**C**m Vordergrundfarbe
 - \e[4**C**m Hintergrundfarbe
 - \e[**Y**;**X**H Cursorposition setzen
 - \e[6n Cursorposition lesen
 - \e[**Y**;**X**R Antwort
 - \ec Reset
- Testen auf der Kommandozeile

```
01 heinloth:~$ echo -e "\ec\e[47m\e[1mF\e[31moo\e[0m"
```

ANSI-Escape-Sequenzen

- SteuerCodes für Cursorposition und Textattribute
- Starten mit ESCAPE-Zeichen (27. ASCII-Zeichen, '\e')
- Wichtige Befehle
 - \e[A**m** Attribute
 - \e[3**C**m Vordergrundfarbe
 - \e[4**C**m Hintergrundfarbe
 - \e[**Y**;**X**H Cursorposition setzen
 - \e[6n Cursorposition lesen
 - \e[**Y**;**X**R Antwort
 - \ec Reset
- Testen auf der Kommandozeile

```
01 heinloth:~$ echo -e "\ec\e[47m\e[1mF\e[31moo\e[0m"
```

Siehe auch <http://www.termssystem.co.uk/vtansi.htm>

Entwicklungsumgebung

Testumgebung

- Virtualisiert

 - QEMU** Softwareemulation

 - KVM** Hardware-Virtualisierung

- Virtualisiert

 - **QEMU** Softwareemulation

 - **KVM** Hardware-Virtualisierung

- Nackte Hardware (*bare-metal*)

 - vier (identische) Testrechner

 - Intel® Core® i5 CPU (2 Kerne @ 3.2 GHz / 4 Threads (HT))

 - 8 GB Arbeitsspeicher

 - PS/2 Tastatur + Maus

 - COM1 verbunden mit faui01r (/dev/ttyBSX)

 - Boot via Netzwerk (PXE)

 - Zugriff

 - direkt im WinCIP mittels KVM-Switch

 - entfernt mittels VNC – auch via Web: <https://i4stubs.cs.fau.de>

 - *privater PC (theoretisch) auch möglich*

- Virtualisiert

 - QEMU** Softwareemulation

 - KVM** Hardware-Virtualisierung

- Nackte Hardware (*bare-metal*)

 - vier (identische) Testrechner

 - Intel® Core® i5 CPU (2 Kerne @ 3.2 GHz / 4 Threads (HT))
 - 8 GB Arbeitsspeicher
 - PS/2 Tastatur + Maus
 - COM1 verbunden mit faui01r (/dev/ttyBSX)

 - Boot via Netzwerk (PXE)

 - Zugriff

 - direkt im WinCIP mittels KVM-Switch
 - entfernt mittels VNC – auch via Web: <https://i4stubs.cs.fau.de>

 - *privater PC (theoretisch) auch möglich*

Abgabe der Aufgaben auf unseren Testrechnern

Wichtige Makefile Targets

make qemu QEMU **ohne** Hardware-Virtualisierung

make kvm QEMU **mit** Hardware-Virtualisierung

Wichtige Makefile Targets

make qemu QEMU **ohne** Hardware-Virtualisierung

make kvm QEMU **mit** Hardware-Virtualisierung

make qemu-gdb starte in QEMU und verbinde zu integrierten GDB-Stub
→ Fehlersuche mit gdb.

make kvm-gdb selbiges mit Hardware-Virtualisierung.

Wichtige Makefile Targets

make qemu QEMU **ohne** Hardware-Virtualisierung

make kvm QEMU **mit** Hardware-Virtualisierung

make qemu-gdb starte in QEMU und verbinde zu integrierten GDB-Stub
→ Fehlersuche mit gdb.

make kvm-gdb selbiges mit Hardware-Virtualisierung.

make netboot für Boot am Test-Rechner ins NFS kopieren

Suffix **-noopt** um Optimierungen auszuschalten (sonst -O3)

Suffix **-verbose** aktiviert zusätzliche Ausgaben (DBG_VERBOSE)

Weitere Makefile Targets

make help zeige eine Beschreibung der verfügbaren Targets an

make iso erstelle ein bootfähiges ISO-Abbild

make qemu-iso boote das Abbild in QEMU

make usb-dev bootfähigen USB-Stick auf **dev** (z.B. *sdb* - **aber Vorsicht!**)
erstellen (als Superuser)

Weitere Makefile Targets

make help zeige eine Beschreibung der verfügbaren Targets an

make iso erstelle ein bootfähiges ISO-Abbild

make qemu-iso boote das Abbild in QEMU

make usb-dev bootfähigen USB-Stick auf **dev** (z.B. *sdb* - **aber Vorsicht!**)
erstellen (als Superuser)

make solution-# Musterlösung zur Aufgabe **#** mit
Hardware-Virtualisierung starten
(auf Testrechner bereits installiert)

Fragen?

Zur Erinnerung: Nächsten Donnerstag (31. Oktober) ist das GDB-Seminar