

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

Sommersemester 2025

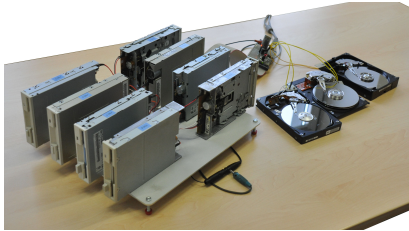
## Wichtiger Hinweis

Diese Vortragsfolien dienen dazu, den Vortrag der/des Lehrenden zu unterstützen. Sie enthalten **nur einen Teil** der Lerninhalte. Wie groß dieser Teil ist, hängt von den konkreten Lerninhalten ab und kann von „praktisch alles“ bis „praktisch gar nichts“ schwanken. Diese Folien alleine sind daher **nicht für ein Selbststudium geeignet!**

Mindestens genauso wichtig wie die Vortragsfolien sind die Beispiel-Programme, Notizen und Tafelbilder, die vor Ihren Augen in den Vorlesungen erarbeitet werden. Diese sind im Git-Repository mit allen Zwischenschritten enthalten (<https://gitlab.cvh-server.de/pgerwinski/es>) und befinden sich in den zu den jeweiligen Kalenderdaten gehörenden Verzeichnissen (z. B. für den 27. 3. 2025 unter <https://gitlab.cvh-server.de/pgerwinski/es/tree/2025ss/20250327/>).

In jedem Fall: *Viel Erfolg!*

Campus  
Velbert/Heiligenhaus

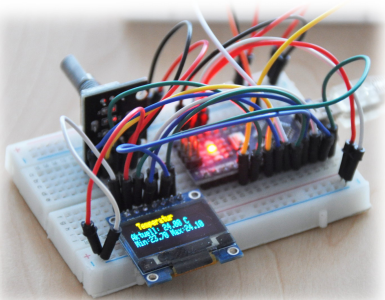


# Eingebettete Systeme



**Hochschule Bochum**  
TECHNIK WIRTSCHAFT GESUNDHEIT

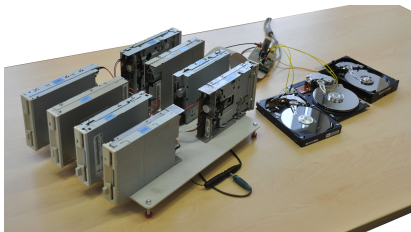
Sommersemester 2025



Prof. Dr. rer. nat. Peter Gerwinski  
27. März 2025

<https://www.peter.gerwinski.de/>

Campus  
Velbert/Heiligenhaus

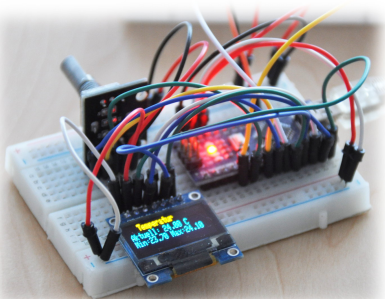


# Eingebettete Systeme



Hochschule Bochum  
TECHNIK WIRTSCHAFT GESUNDHEIT

Sommersemester 2025



Prof. Dr. rer. nat. Peter Gerwinski

27. März 2025

← *rerum naturalium*  
der Naturwissenschaften

<https://www.peter.gerwinski.de/physik/>

# Vorab: Online-Werkzeuge

- Diese Veranstaltung findet **in Präsenz** statt.  
Wir versuchen aber, auch eine Online-Teilnahme zu ermöglichen.
- **Mumble**: Seminarraum 2  
Fragen: Mikrofon einschalten oder über den Chat  
Umfragen: über den Chat – **auch während der Präsenz-Veranstaltung**
- **VNC**: Kanal 6, Passwort: `testcvh`  
Eigenen Bildschirm freigeben: VNC-Software oder Web-Interface *yesVNC*  
Eigenes Kamerabild übertragen: Web-Interface *CVH-Camera*
- Allgemeine Informationen: <https://www.cvh-server.de/online-werkzeuge/>
- Notfall-Schnellzugang: <https://www.cvh-server.de/virtuelle-raeume/>  
Seminarraum 2, VNC-Passwort: `testcvh`
- **Lehrmaterialien**: <https://gitlab.cvh-server.de/pgerwinski/es>

# Was sind eingebettete Systeme?

*Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.*

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

# Was sind eingebettete Systeme?

Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

- keine Aussage über die Größe
- in der Praxis: so klein wie möglich

# Was sind eingebettete Systeme?

Der Ausd  
Rechner  
(eingebe

- keine
- in de



# Was sind eingebettete Systeme?

Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

- keine Aussage über die Größe
- in der Praxis: so klein wie möglich, z. B.:
  - mehrere Rechnerschränke
  - Industrie-PC
  - Einplatinencomputer
  - Mikro-Controller

# Was sind eingebettete Systeme?

Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

- keine Aussage über die Größe
- in der Praxis: so klein wie möglich, z. B.:
  - mehrere Rechnerschränke
  - Industrie-PC
  - Einplatinencomputer
  - Mikro-Controller
- Bedienung über spezielle Ein-/Ausgabegeräte (z. B. PKW, Telefon)

# Was sind eingebettete Systeme?

Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

- keine Aussage über die Größe
- in der Praxis: so klein wie möglich, z. B.:
  - mehrere Rechnerschränke
  - Industrie-PC
  - Einplatinencomputer
  - Mikro-Controller
- Bedienung über spezielle Ein-/Ausgabegeräte (z. B. PKW, Telefon)
- Wartung über speziellen Administrator-Zugang
  - Bus-Schnittstelle (RS-232, CAN-BUS)
  - Netzwerk (TCP/IP, Ethernet oder WLAN)

# Was sind eingebettete Systeme?

Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

- keine Aussage über die Größe
- in der Praxis: so klein wie möglich, z. B.:
  - mehrere Rechnerschränke
  - Industrie-PC
  - Einplatinencomputer
  - Mikro-Controller
- Bedienung über spezielle Ein-/Ausgabegeräte (z. B. PKW, Telefon)
- Wartung über speziellen Administrator-Zugang
  - Bus-Schnittstelle (RS-232, CAN-BUS)
  - Netzwerk (TCP/IP, Ethernet oder WLAN)
- Programmierung oft außerhalb des Systems (Cross-Entwicklungswerkzeuge)

# In dieser Lehrveranstaltung

- Auffrischung: Hardwarenahe Programmierung
- Auffrischung bzw. neu: Einführung in Unix
- Auffrischung bzw. neu: TCP/IP in der Praxis
- Neu: Bus-Systeme
- Neu: Echtzeit-Systeme
- ...
- **Prüfungsleistung:** Projektaufgabe  
Eingebettetes System eigener Wahl zum Laufen bringen
- Statt Klausur: Hausarbeit mit Kolloquium
- Details werden in der Lehrveranstaltung abgesprochen.
- **Praktikum:** kombiniert mit der Prüfungsleistung  
(Details werden noch festgelegt.)
- **Lehrmaterialien:** <https://gitlab.cvh-server.de/pgerwinski/es>

# In dieser Lehrveranstaltung

- Auffrischung: Hardwarenahe Programmierung
- Auffrischung bzw. neu: Einführung in Unix
- Auffrischung bzw. neu: TCP/IP in der Praxis
- Neu: Bus-Systeme
- Neu: Echtzeit-Systeme
- ...
- **Prüfungsleistung:** Projektaufgabe → Projektaufgabe überlegen  
Eingebettetes System eigener Wahl zum Laufen bringen
- Statt Klausur: Hausarbeit mit Kolloquium
- Details werden in der Lehrveranstaltung abgesprochen.
- **Praktikum:** kombiniert mit der Prüfungsleistung  
(Details werden noch festgelegt.)
- **Lehrmaterialien:** <https://gitlab.cvh-server.de/pgerwinski/es>

# Eingebettete Systeme

<https://gitlab.cvh-server.de/pgerwinski/es>

## **1 Einführung**

**1.1** Was sind eingebettete Systeme?

**1.2** In dieser Lehrveranstaltung

## **2 Hardwarenahe Programmierung**

## **3 Einführung in Unix**

## **4 TCP/IP in der Praxis**

...

## 2 Hardwarenahe Programmierung

- Übungsaufgaben 1, 17. 10. 2024:  
Schaltjahr ermitteln, Multiplikationstabelle, Fibonacci-Zahlen, fehlerhaftes Programm
- Übungsaufgaben 5, 21. 11. 2024:  
Zahlensysteme, Mikrocontroller, LED-Blinkmuster
- Übungsaufgabe 8.1, 12. 12. 2024:  
Trickprogrammierung
- Übungsaufgabe 10.1, 9. 1. 2025:  
Personen-Datenbank

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

3. April 2025

# Vorab: Online-Werkzeuge

- Diese Veranstaltung findet **in Präsenz** statt.  
Wir versuchen aber, auch eine Online-Teilnahme zu ermöglichen.
- **Mumble**: Seminarraum 2  
Fragen: Mikrofon einschalten oder über den Chat  
Umfragen: über den Chat – **auch während der Präsenz-Veranstaltung**
- **VNC**: Kanal 6, Passwort: `testcvh`  
Eigenen Bildschirm freigeben: VNC-Software oder Web-Interface *yesVNC*  
Eigenes Kamerabild übertragen: Web-Interface *CVH-Camera*
- Allgemeine Informationen: <https://www.cvh-server.de/online-werkzeuge/>
- Notfall-Schnellzugang: <https://www.cvh-server.de/virtuelle-raeume/>  
Seminarraum 2, VNC-Passwort: `testcvh`
- **Lehrmaterialien**: <https://gitlab.cvh-server.de/pgerwinski/es>

# Was sind eingebettete Systeme?

Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

- keine Aussage über die Größe
- in der Praxis: so klein wie möglich

# Was sind eingebettete Systeme?

Der Ausd  
Rechner  
(eingebe

- keine
- in de



# Was sind eingebettete Systeme?

Der Ausdruck **eingebettetes System** (...) bezeichnet einen elektronischen Rechner oder auch Computer, der in einen technischen Kontext eingebunden (eingebettet) ist.

[https://de.wikipedia.org/wiki/Eingebettetes\\_System](https://de.wikipedia.org/wiki/Eingebettetes_System)

- keine Aussage über die Größe
- in der Praxis: so klein wie möglich, z. B.:
  - mehrere Rechnerschränke
  - Industrie-PC
  - Einplatinencomputer
  - Mikro-Controller
- Bedienung über spezielle Ein-/Ausgabegeräte (z. B. PKW, Telefon)
- Wartung über speziellen Administrator-Zugang
  - Bus-Schnittstelle (RS-232, CAN-BUS)
  - Netzwerk (TCP/IP, Ethernet oder WLAN)
- Programmierung oft außerhalb des Systems (Cross-Entwicklungswerkzeuge)

# In dieser Lehrveranstaltung

- Auffrischung: Hardwarenahe Programmierung
- Auffrischung bzw. neu: Einführung in Unix
- Auffrischung bzw. neu: TCP/IP in der Praxis
- Neu: Bus-Systeme
- Neu: Echtzeit-Systeme
- ...
- **Prüfungsleistung:** Projektaufgabe → Projektaufgabe überlegen  
Eingebettetes System eigener Wahl zum Laufen bringen
- Statt Klausur: Hausarbeit mit Kolloquium
- Details werden in der Lehrveranstaltung abgesprochen.
- **Praktikum:** kombiniert mit der Prüfungsleistung  
(Details werden noch festgelegt.)
- **Lehrmaterialien:** <https://gitlab.cvh-server.de/pgerwinski/es>

# Eingebettete Systeme

<https://gitlab.cvh-server.de/pgerwinski/es>

## **1 Einführung**

**1.1** Was sind eingebettete Systeme?

**1.2** In dieser Lehrveranstaltung

## **2 Hardwarenahe Programmierung**

## **3 Einführung in Unix**

## **4 TCP/IP in der Praxis**

...

## 2 Hardwarenahe Programmierung

- :) Übungsaufgaben 1, 17. 10. 2024:  
Schaltjahr ermitteln, Multiplikationstabelle,  
Fibonacci-Zahlen, fehlerhaftes Programm
- Übungsaufgaben 5, 21. 11. 2024:  
Zahlensysteme, Mikrocontroller, LED-Blinkmuster
- Übungsaufgabe 8.1, 12. 12. 2024:  
Trickprogrammierung
- Übungsaufgabe 10.1, 9. 1. 2025:  
Personen-Datenbank

## Praxisaufgabe: Aufbau eines grundlegenden TCP/IP-Netzwerks

Eingebettete Systeme - Sommersemester 2025 · Prof. Dr. Peter Genwinski

Aufgabe: Vernetzen Sie Ihre Rechner mittels TCP/IP.

- Stellen Sie eine geeignete Hardware-Infrastruktur her.  
(Kabelgebundenes Netz, WLAN, Avian Carriers, ...)
- Wählen Sie geeignete IP-Adressen und prüfen Sie mittels `ping` die gegenseitige Erreichbarkeit.
- Stellen Sie mittels Netcat (`nc`) TCP-Verbindungen her (Ende-zu-Ende-Chat).
- Bieten Sie (z. B. mittels `nc -e`) in diesem Netz Dienste an und testen Sie diese.
- Untersuchen Sie den Netzwerkverkehr mittels `tcpdump` und/oder Wireshark.
- Testen Sie, wie sich externe Dienste (Webseiten, E-Mail) mittels `nc` nutzen lassen.

*Viel Erfolg!*

Stand: 9. April 2025

Copyright © 2025 Peter Genwinski

Lizenz: CC BY-SA (Version 4.0) oder GNU GPL (Version 3 oder höher)

Sie können diese Praxisaufgabe einschließlich  $\LaTeX$ -Quelltext herunterladen unter:

<https://gitlab.cvh-server.de/pgenwinski/es>

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

17. April 2025

# Eingebettete Systeme

<https://gitlab.cvh-server.de/pgerwinski/es>

- 1 Einführung
- 2 Hardwarenahe Programmierung
- 3 Einführung in Unix
- 4 TCP/IP in der Praxis

...

# 1 Hardwarenahe Programmierung

- :) Übungsaufgaben 1, 17. 10. 2024:  
Schaltjahr ermitteln, Multiplikationstabelle,  
Fibonacci-Zahlen, fehlerhaftes Programm
- Übungsaufgaben 5, 21. 11. 2024:  
Zahlensysteme, Mikrocontroller, LED-Blinkmuster
- Übungsaufgabe 8.1, 12. 12. 2024:  
Trickprogrammierung
- Übungsaufgabe 10.1, 9. 1. 2025:  
Personen-Datenbank

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

8. Mai 2025

# Eingebettete Systeme

<https://gitlab.cvh-server.de/pgerwinski/es>

- 1 Einführung
- 2 Hardwarenahe Programmierung
- 3 Einführung in Unix
- 4 TCP/IP in der Praxis
- 5 Bus-Systeme
  - 5.1 Was sind Bus-Systeme?
  - 5.2 RS-232
  - 5.3 I<sup>2</sup>C (TWI)
  - 5.4 SPI
  - 5.5 PWM
  - 5.6 Sonstiges

...

## 2 Hardwarenahe Programmierung

- :) Übungsaufgaben 1, 17. 10. 2024:  
Schaltjahr ermitteln, Multiplikationstabelle,  
Fibonacci-Zahlen, fehlerhaftes Programm
- :) Übungsaufgaben 5, 21. 11. 2024:  
Zahlensysteme, Mikrocontroller, LED-Blinkmuster
- Übungsaufgabe 8.1, 12. 12. 2024:  
Trickprogrammierung
- Übungsaufgabe 10.1, 9. 1. 2025:  
Personen-Datenbank

# 5 Bus-Systeme

## 5.1 Was sind Bus-Systeme?

*Ein Bus ist ein System zur Datenübertragung zwischen mehreren Teilnehmern über einen gemeinsamen Übertragungsweg. Findet eine Datenübertragung zwischen zwei Teilnehmern statt, so müssen die übrigen Teilnehmer schweigen, da sie sich sonst gegenseitig stören würden. Umgangssprachlich werden mitunter – oft aus historischen Gründen – auch Datenübertragungssysteme als „Bus“ bezeichnet, die technisch eigentlich eine andere Topologie besitzen.*

[https://de.wikipedia.org/wiki/Bus\\_\(Datenverarbeitung\)](https://de.wikipedia.org/wiki/Bus_(Datenverarbeitung))

Beispiele:

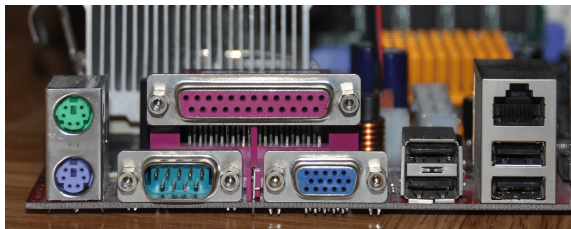
- Computer kommuniziert mit Peripherie
- Computer kommunizieren (direkt) miteinander
- Prozessor kommuniziert mit externem Speicher
- Teile eines Prozessors kommunizieren miteinander

# 5 Bus-Systeme

## 5.1 Was sind Bus-Systeme?

Standard-Personal-Computer:

- Einsteckkarten: PCI (und Vorgänger)
- Festplatten: SATA (und Vorgänger)
- USB, FireWire, ...
- Ethernet, CAN-Bus, ...
- WLAN, BlueTooth, IR, ...
- PS/2, RS-232, Centronics



# 5 Bus-Systeme

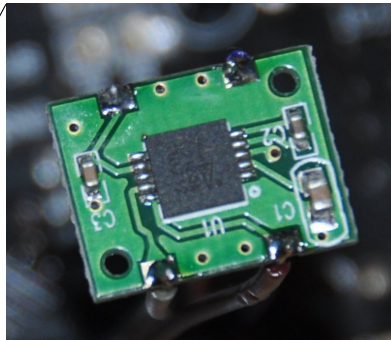
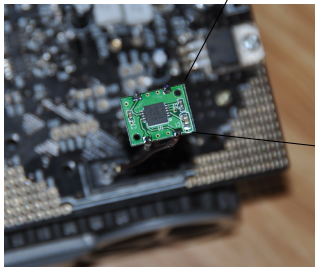
## 5.1 Was sind Bus-Systeme?

Standard-Personal-Computer:

- Einsteckkarten: PCI (und Vorgänger)
- Festplatten: SATA (und Vorgänger)
- USB, FireWire, ...
- Ethernet, CAN-Bus, ...
- WLAN, BlueTooth, IR, ...
- PS/2, RS-232, Centronics

Minimal-Hardware:

- RS-232
- I<sup>2</sup>C (TWI)
- SPI



# 5 Bus-Systeme

## 5.1 Was sind Bus-Systeme?

<i>seriell</i>	jedes Bit einzeln übertragen
<i>parallel</i>	mehrere Bits gleichzeitig
<i>synchron</i>	Abgleich über Steuerleitung: <i>Takt</i>
<i>asynchron</i>	Abgleich über Zeitvereinbarungen
<i>Punkt-zu-Punkt</i>	genau zwei Teilnehmer
<i>busfähig</i>	mehrere Teilnehmer, mit <i>Adressierung</i>

- I<sup>2</sup>C: seriell, synchron, mit Adressierung
- RS-232: seriell, asynchron, Punkt-zu-Punkt
- RS-485, USB, CAN: seriell, asynchron, mit Adressierung
- SPI: seriell, synchron, Punkt-zu-Punkt oder mit Adressierung

# 5 Bus-Systeme

## 5.2 RS-232

seriell

- *TX*: 1 Leitung für Daten
- *RX*: ggf. 1 Leitung für Daten in der anderen Richtung
- *GND*: gemeinsame *Masse*
- evtl. zusätzliche Steuerleitungen

asynchron

- *keine* Taktleitung für Abgleich, wann Daten anliegen
- Stattdessen: Abgleich über Zeitvereinbarungen

—> Jeder Teilnehmer braucht eine eigene Zeitbasis.

Punkt-zu-Punkt

- nur 2 Teilnehmer vorgesehen

## 5.2 RS-232

Synchronisation

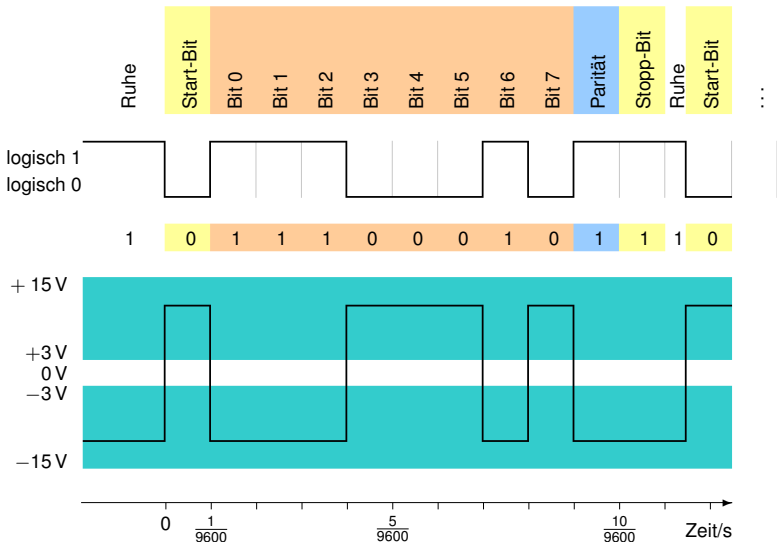
Daten

Check

9600 Baud, 8 Daten-Bits, ungerade Parität, 1 Stopp-Bit

Beispiel-Daten: ASCII „G“ = 71 = 0100 0111 binär

Übertragung der Daten von rechts (Bit 0) nach links (Bit 7)



## 5.3 I<sup>2</sup>C (TWI)

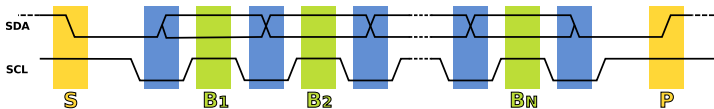
I<sup>2</sup>C = Inter-IC; TWI = Two-Wire-Interface

seriell

- *SDA*: 1 Leitung für Daten (in beiden Richtungen)
- *SCL*: Taktleitung (Clock)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung



busfähig

- *Controller* (früher: *Master*) initiiert Kommunikation und steuert Taktleitung
- *Target* (früher: *Slave*) liest Taktleitung, liest und schreibt Daten
- erstes gesendetes Byte: *Adresse* des Teilnehmers
- 2 Adressen pro Teilnehmer: Lesen/Schreiben

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

15. Mai 2025

# Eingebettete Systeme

<https://gitlab.cvh-server.de/pgerwinski/es>

## 1 Einführung

## 2 Hardwarenahe Programmierung

## 3 Einführung in Unix

## 4 TCP/IP in der Praxis

## 5 Bus-Systeme

### 5.1 Was sind Bus-Systeme?

### 5.2 RS-232

### 5.3 I<sup>2</sup>C (TWI)

### 5.4 SPI

### 5.5 PWM

### 5.6 Sonstiges

## 6 Echtzeit

...

## 2 Hardwarenahe Programmierung

- :) Übungsaufgaben 1, 17. 10. 2024:  
Schaltjahr ermitteln, Multiplikationstabelle,  
Fibonacci-Zahlen, fehlerhaftes Programm
- :) Übungsaufgaben 5, 21. 11. 2024:  
Zahlensysteme, Mikrocontroller, LED-Blinkmuster
- Übungsaufgabe 8.1, 12. 12. 2024:  
Trickprogrammierung
- Übungsaufgabe 10.1, 9. 1. 2025:  
Personen-Datenbank

# 5 Bus-Systeme

## 5.1 Was sind Bus-Systeme?

*Ein Bus ist ein System zur Datenübertragung zwischen mehreren Teilnehmern über einen gemeinsamen Übertragungsweg. Findet eine Datenübertragung zwischen zwei Teilnehmern statt, so müssen die übrigen Teilnehmer schweigen, da sie sich sonst gegenseitig stören würden. Umgangssprachlich werden mitunter – oft aus historischen Gründen – auch Datenübertragungssysteme als „Bus“ bezeichnet, die technisch eigentlich eine andere Topologie besitzen.*

[https://de.wikipedia.org/wiki/Bus\\_\(Datenverarbeitung\)](https://de.wikipedia.org/wiki/Bus_(Datenverarbeitung))

Beispiele:

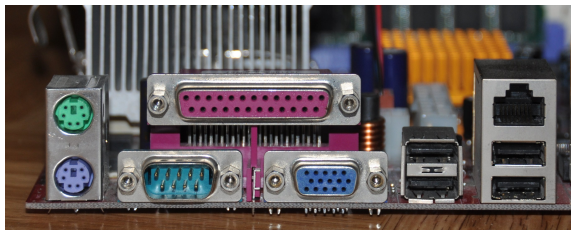
- Computer kommuniziert mit Peripherie
- Computer kommunizieren (direkt) miteinander
- Prozessor kommuniziert mit externem Speicher
- Teile eines Prozessors kommunizieren miteinander

# 5 Bus-Systeme

## 5.1 Was sind Bus-Systeme?

Standard-Personal-Computer:

- Einsteckkarten: PCI (und Vorgänger)
- Festplatten: SATA (und Vorgänger)
- USB, FireWire, ...
- Ethernet, CAN-Bus, ...
- WLAN, BlueTooth, IR, ...
- PS/2, RS-232, Centronics



# 5 Bus-Systeme

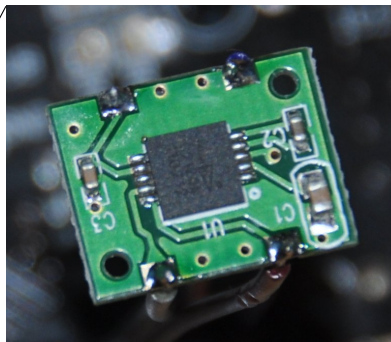
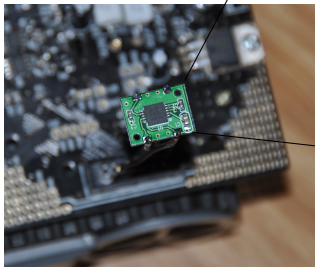
## 5.1 Was sind Bus-Systeme?

Standard-Personal-Computer:

- Einsteckkarten: PCI (und Vorgänger)
- Festplatten: SATA (und Vorgänger)
- USB, FireWire, ...
- Ethernet, CAN-Bus, ...
- WLAN, BlueTooth, IR, ...
- PS/2, RS-232, Centronics

Minimal-Hardware:

- RS-232
- I<sup>2</sup>C (TWI)
- SPI



# 5 Bus-Systeme

## 5.1 Was sind Bus-Systeme?

<i>seriell</i>	jedes Bit einzeln übertragen
<i>parallel</i>	mehrere Bits gleichzeitig
<i>synchron</i>	Abgleich über Steuerleitung: <i>Takt</i>
<i>asynchron</i>	Abgleich über Zeitvereinbarungen
<i>Punkt-zu-Punkt</i>	genau zwei Teilnehmer
<i>busfähig</i>	mehrere Teilnehmer, mit <i>Adressierung</i>

- I<sup>2</sup>C: seriell, synchron, mit Adressierung
- RS-232: seriell, asynchron, Punkt-zu-Punkt
- RS-485, USB, CAN: seriell, asynchron, mit Adressierung
- SPI: seriell, synchron, Punkt-zu-Punkt oder mit Adressierung

# 5 Bus-Systeme

## 5.2 RS-232

seriell

- *TX*: 1 Leitung für Daten
- *RX*: ggf. 1 Leitung für Daten in der anderen Richtung
- *GND*: gemeinsame *Masse*
- evtl. zusätzliche Steuerleitungen

asynchron

- *keine* Taktleitung für Abgleich, wann Daten anliegen
- Stattdessen: Abgleich über Zeitvereinbarungen

—> Jeder Teilnehmer braucht eine eigene Zeitbasis.

Punkt-zu-Punkt

- nur 2 Teilnehmer vorgesehen

## 5.2 RS-232

Synchronisation

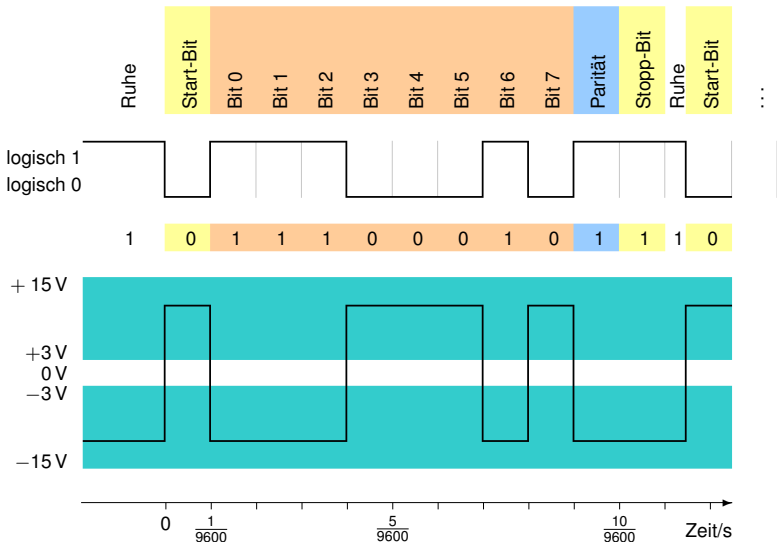
Daten

Check

9600 Baud, 8 Daten-Bits, ungerade Parität, 1 Stopp-Bit

Beispiel-Daten: ASCII „G“ = 71 = 0100 0111 binär

Übertragung der Daten von rechts (Bit 0) nach links (Bit 7)



## 5.3 I<sup>2</sup>C (TWI)

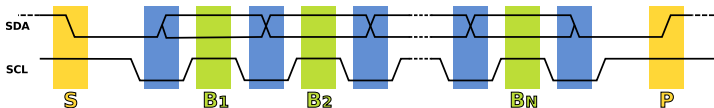
I<sup>2</sup>C = Inter-IC; TWI = Two-Wire-Interface

seriell

- *SDA*: 1 Leitung für Daten (in beiden Richtungen)
- *SCL*: Taktleitung (Clock)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung



busfähig

- *Controller* (früher: *Master*) initiiert Kommunikation und steuert Taktleitung
- *Target* (früher: *Slave*) liest Taktleitung, liest und schreibt Daten
- erstes gesendetes Byte: *Adresse* des Teilnehmers
- 2 Adressen pro Teilnehmer: Lesen/Schreiben

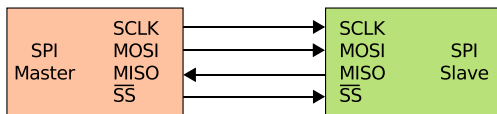
# 5 Bus-Systeme

## 5.4 SPI

### Serial Peripheral Interface

seriell

- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- $\overline{SS}$ : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät



synchron

- Abgleich über Taktleitung

busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- *Slave* wird über *Slave Select* ausgewählt

# 5 Bus-Systeme

## 5.4 SPI

### Serial Peripheral Interface

seriell

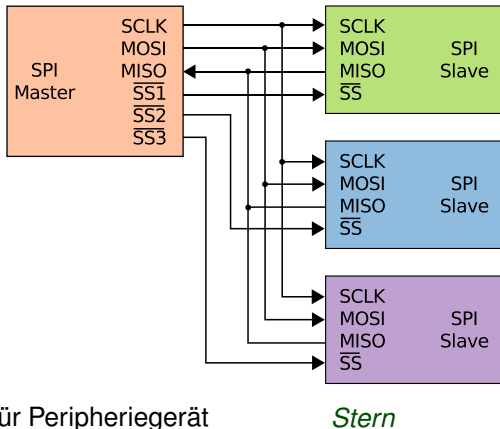
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- $\overline{SS}$ : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- *Slave* wird über *Slave Select* ausgewählt



# 5 Bus-Systeme

## 5.4 SPI

Serial Peripheral Interface

seriell

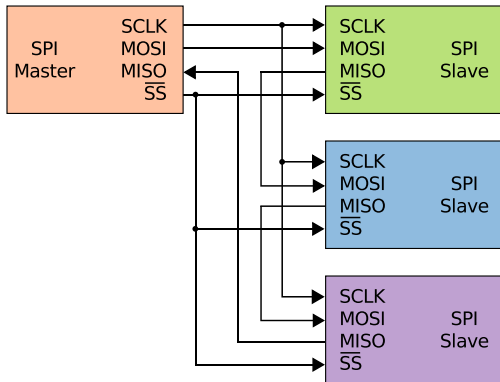
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- $\overline{SS}$ : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- *Slave* wird über *Slave Select* ausgewählt



*Kaskade  
Daisy Chain*

# 5 Bus-Systeme

## 5.4 SPI

Serial Peripheral Interface

seriell

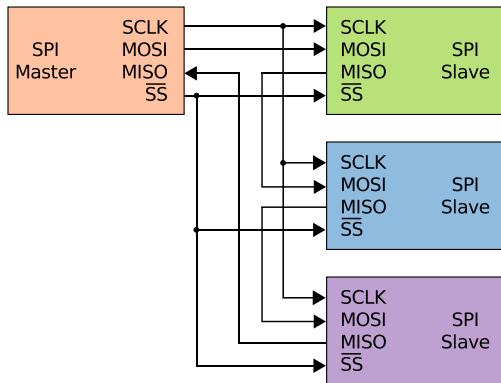
- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- $\overline{SS}$ : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- *Slave* wird über *Slave Select* ausgewählt



*Kaskade  
Daisy Chain*

Slave gibt MOSI-Input um 1 Takt verzögert an MISO aus → Master setzt „im richtigen Moment“  $\overline{SS}$

# 5 Bus-Systeme

## 5.5 PWM

Pulsweitenmodulation – *pulse-width modulation*

- Steuerung von Motoren
- Nutzung als allgemeines Protokoll zur Übertragung analoger Werte

# 5 Bus-Systeme

## 5.6 Sonstiges

### Matrix-Schaltung

- möglichst viele Aktoren/Sensoren  
über möglichst wenige digital Inputs abfragen  
bzw. über möglichst wenige digitale Outputs steuern
- Beispiele: LED-Felder, Tastaturen

# 5 Bus-Systeme

## 5.6 Sonstiges

### Matrix-Schaltung

- möglichst viele Aktoren/Sensoren  
über möglichst wenige digitals Inputs abfragen  
bzw. über möglichst wenige digitale Outputs steuern
- Beispiele: LED-Felder, Tastaturen

### R/2R-Netzwerk

- möglichst viele digitale Inputs  
über einen einzigen analogen Input abfragen
- Beispiele: Tastaturen

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

22. Mai 2025

# Eingebettete Systeme

<https://gitlab.cvh-server.de/pgerwinski/es>

## 1 Einführung

## 2 Hardwarenahe Programmierung

## 3 Einführung in Unix

## 4 TCP/IP in der Praxis

## 5 Bus-Systeme

### 5.1 Was sind Bus-Systeme?

### 5.2 RS-232

### 5.3 I<sup>2</sup>C (TWI)

### 5.4 SPI

### 5.5 PWM

### 5.6 Sonstiges

## 6 Echtzeit

...

## 2 Hardwarenahe Programmierung

- :) Übungsaufgaben 1, 17. 10. 2024:  
Schaltjahr ermitteln, Multiplikationstabelle,  
Fibonacci-Zahlen, fehlerhaftes Programm
- :) Übungsaufgaben 5, 21. 11. 2024:  
Zahlensysteme, Mikrocontroller, LED-Blinkmuster
- Übungsaufgabe 8.1, 12. 12. 2024:  
Trickprogrammierung
- Übungsaufgabe 10.1, 9. 1. 2025:  
Personen-Datenbank

# 5 Bus-Systeme

## 5.2 RS-232

seriell

- *TX*: 1 Leitung für Daten
- *RX*: ggf. 1 Leitung für Daten in der anderen Richtung
- *GND*: gemeinsame *Masse*
- evtl. zusätzliche Steuerleitungen

asynchron

- *keine* Taktleitung für Abgleich, wann Daten anliegen
- Stattdessen: Abgleich über Zeitvereinbarungen

—> Jeder Teilnehmer braucht eine eigene Zeitbasis.

Punkt-zu-Punkt

- nur 2 Teilnehmer vorgesehen

## 5.2 RS-232

Synchronisation

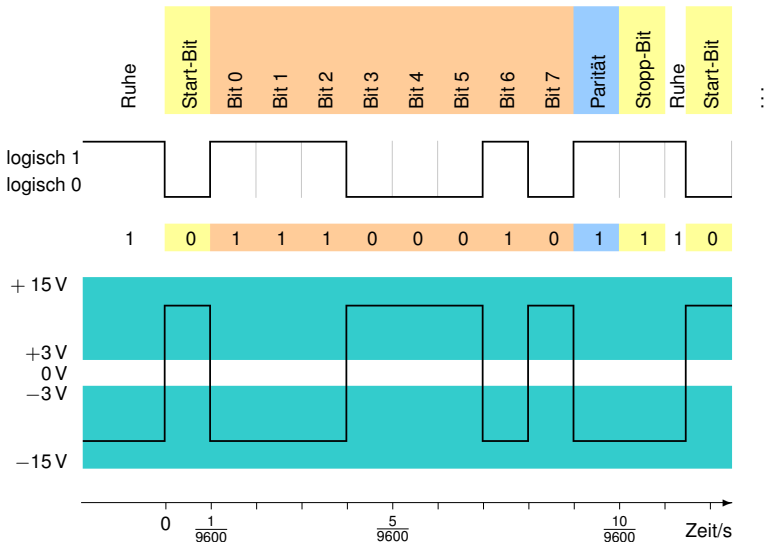
Daten

Check

9600 Baud, 8 Daten-Bits, ungerade Parität, 1 Stopp-Bit

Beispiel-Daten: ASCII „G“ = 71 = 0100 0111 binär

Übertragung der Daten von rechts (Bit 0) nach links (Bit 7)



## 5.3 I<sup>2</sup>C (TWI)

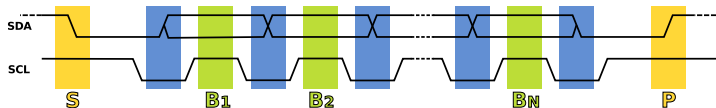
I<sup>2</sup>C = Inter-IC; TWI = Two-Wire-Interface

seriell

- *SDA*: 1 Leitung für Daten (in beiden Richtungen)
- *SCL*: Taktleitung (Clock)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung



busfähig

- *Controller* (früher: *Master*) initiiert Kommunikation und steuert Taktleitung
- *Target* (früher: *Slave*) liest Taktleitung, liest und schreibt Daten
- erstes gesendetes Byte: *Adresse* des Teilnehmers
- 2 Adressen pro Teilnehmer: Lesen/Schreiben

# 5 Bus-Systeme

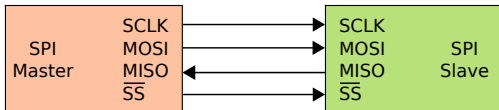
## 5.4 SPI

Serial Peripheral Interface

seriell

- *MOSI*: Master Out, Slave In
- *MISO*: Master In, Slave Out
- *SCLK*: Taktleitung (Clock)
- $\overline{SS}$ : Slave Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

historische  
Bezeichnungen



synchron

- Abgleich über Taktleitung

busfähig

- *Master* initiiert Kommunikation und steuert Taktleitung
- *Slave* wird über *Slave Select* ausgewählt

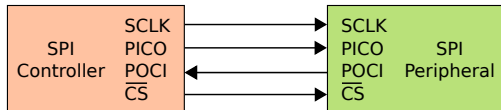
# 5 Bus-Systeme

## 5.4 SPI

### Serial Peripheral Interface

seriell

- *PICO*: Peripheral In, Controller Out
- *POCI*: Peripheral Out, Controller In
- *SCLK*: Taktleitung (Clock)
- $\overline{CS}$ : Chip Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät



synchron

- Abgleich über Taktleitung

busfähig

- *Controller* initiiert Kommunikation und steuert Taktleitung
- *Peripheral* wird über *Chip Select* ausgewählt

# 5 Bus-Systeme

## 5.4 SPI

### Serial Peripheral Interface

seriell

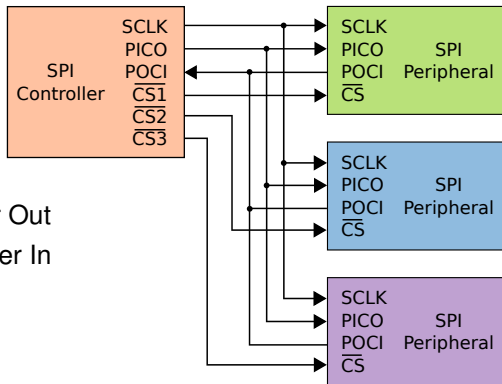
- *PICO*: Peripheral In, Controller Out
- *POCI*: Peripheral Out, Controller In
- *SCLK*: Taktleitung (Clock)
- $\overline{CS}$ : Chip Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

- *Controller* initiiert Kommunikation und steuert Taktleitung
- *Peripheral* wird über *Chip Select* ausgewählt



*Stern*

# 5 Bus-Systeme

## 5.4 SPI

Serial Peripheral Interface

seriell

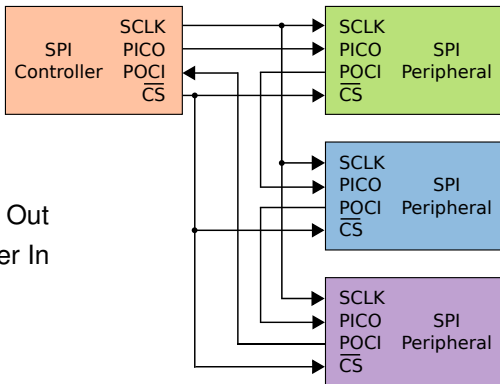
- *PICO*: Peripheral In, Controller Out
- *POCI*: Peripheral Out, Controller In
- *SCLK*: Taktleitung (Clock)
- $\overline{CS}$ : Chip Select (invertiert)
- *GND*: gemeinsame Masse
- evtl. *VCC*: Stromversorgung für Peripheriegerät

synchron

- Abgleich über Taktleitung

busfähig

- *Controller* initiiert Kommunikation und steuert Taktleitung
- *Peripheral* wird über *Chip Select* ausgewählt



*Kaskade  
Daisy Chain*

Peripheral gibt PICO-Input um 1 Takt verzögert an POCI aus → Controller setzt „im richtigen Moment“  $\overline{CS}$

# 5 Bus-Systeme

## 5.5 PWM

Pulsweitenmodulation – *pulse-width modulation*

- Steuerung von Motoren
- Nutzung als allgemeines Protokoll zur Übertragung analoger Werte

# 5 Bus-Systeme

## 5.6 Sonstiges

### Matrix-Schaltung

- möglichst viele Aktoren/Sensoren  
über möglichst wenige digitals Inputs abfragen  
bzw. über möglichst wenige digitale Outputs steuern
- Beispiele: LED-Felder, Tastaturen

### R/2R-Netzwerk

- möglichst viele digitale Inputs  
über einen einzigen analogen Input abfragen
- Beispiele: Tastaturen

## 6 Echtzeit

### 6.1 Was ist Echtzeit?

## 6 Echtzeit

### 6.1 Was ist Echtzeit?

- Animation in Echtzeit:  
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung

## 6 Echtzeit

### 6.1 Was ist Echtzeit?

- Animation in Echtzeit:  
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung
- Fantasy-Rollenspiel in Echtzeit:  
Der Zeitverlauf der Spielwelt entspricht dem der realen Welt.

## 6 Echtzeit

### 6.1 Was ist Echtzeit?

- Animation in Echtzeit:  
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung
- Fantasy-Rollenspiel in Echtzeit:  
Der Zeitverlauf der Spielwelt entspricht dem der realen Welt.
- Datenverarbeitung in Echtzeit:  
Die Daten werden so schnell verarbeitet, wie sie anfallen.

# 6 Echtzeit

## 6.1 Was ist Echtzeit?

- Animation in Echtzeit:  
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung
- Fantasy-Rollenspiel in Echtzeit:  
Der Zeitverlauf der Spielwelt entspricht dem der realen Welt.
- Datenverarbeitung in Echtzeit:  
Die Daten werden so schnell verarbeitet, wie sie anfallen.
- speziell: Echtzeit-Steuerung von Maschinen:  
Die Berechnung kann mit den physikalischen Vorgängen schritthalten.

# 6 Echtzeit

## 6.1 Was ist Echtzeit?

- Animation in Echtzeit:  
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung
- Fantasy-Rollenspiel in Echtzeit:  
Der Zeitverlauf der Spielwelt entspricht dem der realen Welt.
- Datenverarbeitung in Echtzeit:  
Die Daten werden so schnell verarbeitet, wie sie anfallen.
- speziell: Echtzeit-Steuerung von Maschinen:  
Die Berechnung kann mit den physikalischen Vorgängen schritthalten.

→ „Schnell genug.“

## 6.1 Was ist Echtzeit?

„Schnell genug.“

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

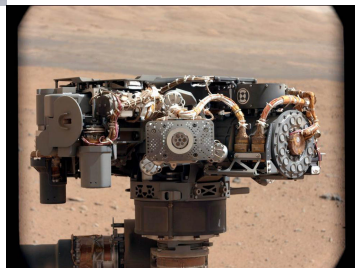
- „Ganz schlecht.“ → *harte Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*  
verspätetes Ergebnis qualitätsmindernd
  - verwenden und Verzögerung in Kauf nehmen
  - verwerfen und Ausfall in Kauf nehmen

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*  
verspätetes Ergebnis qualitätsmindernd
  - verwenden und Verzögerung in Kauf nehmen
  - verwerfen und Ausfall in Kauf nehmen
- „Es gibt keinen festen Termin. Möglichst schnell halt.“

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*  
verspätetes Ergebnis qualitätsmindernd
  - verwenden und Verzögerung in Kauf nehmen
  - verwerfen und Ausfall in Kauf nehmen
- „Es gibt keinen festen Termin. Möglichst schnell halt.“  
→ *keine Echtzeit*

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

Beispiel:

- Eine Motorsteuerung benötigt alle  $2000\ \mu\text{s}$  einen Steuerimpuls, dessen Berechnung maximal  $10\ \mu\text{s}$  dauert.

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

Beispiel:

- Eine Motorsteuerung benötigt alle  $2000\ \mu\text{s}$  einen Steuerimpuls, dessen Berechnung maximal  $10\ \mu\text{s}$  dauert.
- Entweder: Der Steuer-Computer macht noch andere Dinge.  
→ Risiko der Zeitüberschreitung

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

Beispiel:

- Eine Motorsteuerung benötigt alle  $2000\ \mu\text{s}$  einen Steuerimpuls, dessen Berechnung maximal  $10\ \mu\text{s}$  dauert.
- Entweder: Der Steuer-Computer macht noch andere Dinge.  
→ Risiko der Zeitüberschreitung
- Oder: Der Steuer-Computer macht nichts anderes.  
→ Verschwendung von Rechenzeit

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

Beispiel:

- Eine Motorsteuerung benötigt alle  $2000\ \mu\text{s}$  einen Steuerimpuls, dessen Berechnung maximal  $10\ \mu\text{s}$  dauert.
- Entweder: Der Steuer-Computer macht noch andere Dinge.  
→ Risiko der Zeitüberschreitung
- Oder: Der Steuer-Computer macht nichts anderes.  
→ Verschwendung von Rechenzeit  
→ Na und?

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

**„Verschwendung von Rechenzeit – na und?“**

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

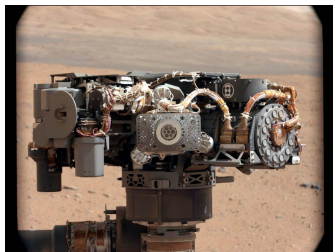
### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren



## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren
- Implantat: Platz- und Stromverbrauch minimieren



## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren
- Implantat: Platz- und Stromverbrauch minimieren

→ **Echtzeitprogrammierung**

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware

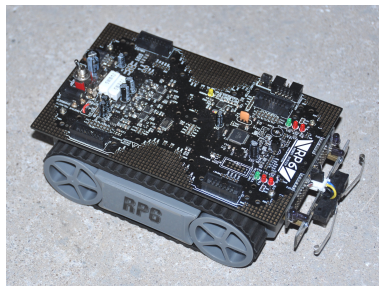


## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software



## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software
- Quadrocopter:  
Motorsteuerung vs. Sensoren-Abfrage  
vs. Funk-Fernsteuerung ...  
→ spezielle Software



## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Beispiele für Lösungen:

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software
- Quadrocopter:  
Motorsteuerung vs. Sensoren-Abfrage  
vs. Funk-Fernsteuerung ...  
→ spezielle Software
- Flugzeugkabinensimulatortür:  
Türsteuerung vs. Bedienung  
→ Echtzeitbetriebssystem



## 6.2 Echtzeitprogrammierung

Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

5. Juni 2025

# Eingebettete Systeme

<https://gitlab.cvh-server.de/pgerwinski/es>

- 1 Einführung**
- 2 Hardwarenahe Programmierung**
- 3 Einführung in Unix**
- 4 TCP/IP in der Praxis**
- 5 Bus-Systeme**
- 6 Echtzeit**
  - 6.1 Was ist Echtzeit?
  - 6.2 Echtzeitprogrammierung
  - 6.3 Multitasking
  - 6.4 Ressourcen
  - 6.5 Prioritäten

# 6 Echtzeit

## 6.1 Was ist Echtzeit?

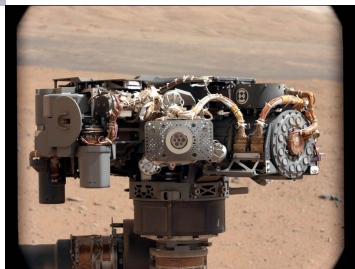
- Animation in Echtzeit:  
schnelle Berechnung anstatt Wiedergabe einer Aufzeichnung
- Fantasy-Rollenspiel in Echtzeit:  
Der Zeitverlauf der Spielwelt entspricht dem der realen Welt.
- Datenverarbeitung in Echtzeit:  
Die Daten werden so schnell verarbeitet, wie sie anfallen.
- speziell: Echtzeit-Steuerung von Maschinen:  
Die Berechnung kann mit den physikalischen Vorgängen schritthalten.

→ „Schnell genug.“

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*



## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*  
verspätetes Ergebnis qualitätsmindernd
  - verwenden und Verzögerung in Kauf nehmen
  - verwerfen und Ausfall in Kauf nehmen

## 6.1 Was ist Echtzeit?

„Schnell genug.“ – „Und wenn nicht?“

- „Ganz schlecht.“ → *harte Echtzeit*  
rechtzeitiges Ergebnis funktionsentscheidend
- „Unschön.“ → *weiche Echtzeit*  
verspätetes Ergebnis qualitätsmindernd
  - verwenden und Verzögerung in Kauf nehmen
  - verwerfen und Ausfall in Kauf nehmen
- „Es gibt keinen festen Termin. Möglichst schnell halt.“  
→ *keine Echtzeit*

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

Beispiel:

- Eine Motorsteuerung benötigt alle  $2000\ \mu\text{s}$  einen Steuerimpuls, dessen Berechnung maximal  $10\ \mu\text{s}$  dauert.
- Entweder: Der Steuer-Computer macht noch andere Dinge.  
→ Risiko der Zeitüberschreitung
- Oder: Der Steuer-Computer macht nichts anderes.  
→ Verschwendung von Rechenzeit  
→ Na und?

## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

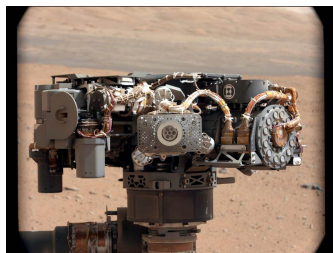
### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren



## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren
- Implantat: Platz- und Stromverbrauch minimieren



## 6.1 Was ist Echtzeit?

Das Problem:

Harte Echtzeitanforderungen



Ressourcen optimal nutzen

### „Verschwendung von Rechenzeit – na und?“

Große Stückzahlen

- 138 000 Toyota Prius V von Mai 2011 bis April 2012

Wertvolle Ressourcen

- Fähigkeiten einer Raumsonde optimieren
- Implantat: Platz- und Stromverbrauch minimieren

→ **Echtzeitprogrammierung**

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware

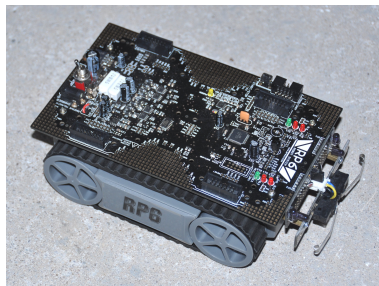


## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software



## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software
- Quadrocopter:  
Motorsteuerung vs. Sensoren-Abfrage  
vs. Funk-Fernsteuerung ...  
→ spezielle Software



## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software
- Quadrocopter:  
Motorsteuerung vs. Sensoren-Abfrage  
vs. Funk-Fernsteuerung ...  
→ spezielle Software
- Flugzeugkabinensimulatortür:  
Türsteuerung vs. Bedienung  
→ Echtzeitbetriebssystem



## 6.2 Echtzeitprogrammierung

Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

## 6.2 Echtzeitprogrammierung

### Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

### RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- LED-Ansteuerung ...

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

12. Juni 2025

# Eingebettete Systeme

<https://gitlab.cvh-server.de/pgerwinski/es>

- 1 Einführung**
- 2 Hardwarenahe Programmierung**
- 3 Einführung in Unix**
- 4 TCP/IP in der Praxis**
- 5 Bus-Systeme**
- 6 Echtzeit**
  - 6.1** Was ist Echtzeit?
  - 6.2** Echtzeitprogrammierung
  - 6.3** Multitasking
  - 6.4** Ressourcen
  - 6.5** Prioritäten

## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware

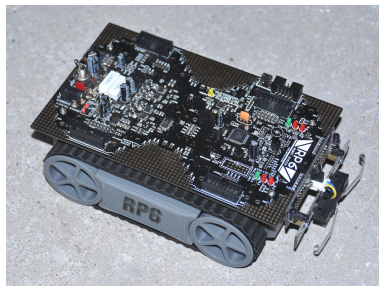


## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software



## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software
- Quadrocopter:  
Motorsteuerung vs. Sensoren-Abfrage  
vs. Funk-Fernsteuerung ...  
→ spezielle Software



## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software
- Quadrocopter:  
Motorsteuerung vs. Sensoren-Abfrage  
vs. Funk-Fernsteuerung ...  
→ spezielle Software
- Flugzeugkabinensimulatortür:  
Türsteuerung vs. Bedienung  
→ Echtzeitbetriebssystem



## 6.2 Echtzeitprogrammierung

Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

## 6.2 Echtzeitprogrammierung

### Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

### RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
Durch Zähler im Interrupt-Handler: verschiedene Taktraten  
1000mal pro Sekunde: Stopwatches  
5mal pro Sekunde: Blinkende Power-On-LED  
1000mal pro Sekunde: Bumper, ACS, PWM zur Motorsteuerung  
Geschwindigkeitsmessung durch Zählen der Ticks in 0.2 s  
Anpassung der Motorkraft in  $\pm 1$ -Schritten

## 6.2 Echtzeitprogrammierung

### Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

### RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
Durch Zähler im Interrupt-Handler: verschiedene Taktraten  
1000mal pro Sekunde: Stopwatches  
5mal pro Sekunde: Blinkende Power-On-LED  
1000mal pro Sekunde: Bumper, ACS, PWM zur Motorsteuerung  
Geschwindigkeitsmessung durch Zählen der Ticks in 0.2 s  
Anpassung der Motorkraft in  $\pm 1$ -Schritten
- Nebenbei: Benutzerprogramm

## 6.2 Echtzeitprogrammierung

### Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

### RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
Durch Zähler im Interrupt-Handler: verschiedene Taktraten  
1000mal pro Sekunde: Stopwatches  
5mal pro Sekunde: Blinkende Power-On-LED  
1000mal pro Sekunde: Bumper, ACS, PWM zur Motorsteuerung  
Geschwindigkeitsmessung durch Zählen der Ticks in 0.2 s  
Anpassung der Motorkraft in  $\pm 1$ -Schritten
- Nebenbei: 1 Benutzerprogramm

# Eingebettete Systeme

Prof. Dr. rer. nat. Peter Gerwinski

26. Juni 2025

# Eingebettete Systeme

<https://gitlab.cvh-server.de/pgerwinski/es>

- 1 Einführung**
- 2 Hardwarenahe Programmierung**
- 3 Einführung in Unix**
- 4 TCP/IP in der Praxis**
- 5 Bus-Systeme**
- 6 Echtzeit**
  - 6.1** Was ist Echtzeit?
  - 6.2** Echtzeitprogrammierung
  - 6.3** Multitasking
  - 6.4** Ressourcen
  - 6.5** Prioritäten

## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware

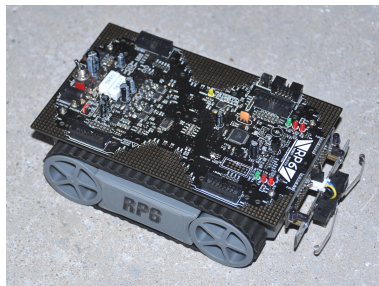


## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software



## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software
- Quadrocopter:  
Motorsteuerung vs. Sensoren-Abfrage  
vs. Funk-Fernsteuerung ...  
→ spezielle Software



## 6.2 Echtzeitprogrammierung

Echtzeitanforderungen erfüllen, ohne Ressourcen zu verschwenden

Aber wie?

- ZigBee-Modul:  
Funk- vs. UART-Protokoll  
→ dedizierte Hardware
- RP6:  
Motorsteuerung vs. Anwender-Software  
→ spezielle Software
- Quadrocopter:  
Motorsteuerung vs. Sensoren-Abfrage  
vs. Funk-Fernsteuerung ...  
→ spezielle Software
- Flugzeugkabinensimulatortür:  
Türsteuerung vs. Bedienung  
→ Echtzeitbetriebssystem



## 6.2 Echtzeitprogrammierung

Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

## 6.2 Echtzeitprogrammierung

### Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

### RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
Durch Zähler im Interrupt-Handler: verschiedene Taktraten  
1000mal pro Sekunde: Stopwatches  
5mal pro Sekunde: Blinkende Power-On-LED  
1000mal pro Sekunde: Bumper, ACS, PWM zur Motorsteuerung  
Geschwindigkeitsmessung durch Zählen der Ticks in 0.2 s  
Anpassung der Motorkraft in  $\pm 1$ -Schritten

## 6.2 Echtzeitprogrammierung

### Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

### RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
Durch Zähler im Interrupt-Handler: verschiedene Taktraten  
1000mal pro Sekunde: Stopwatches  
5mal pro Sekunde: Blinkende Power-On-LED  
1000mal pro Sekunde: Bumper, ACS, PWM zur Motorsteuerung  
Geschwindigkeitsmessung durch Zählen der Ticks in 0.2 s  
Anpassung der Motorkraft in  $\pm 1$ -Schritten
- Nebenbei: Benutzerprogramm

## 6.2 Echtzeitprogrammierung

### Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

### RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
Durch Zähler im Interrupt-Handler: verschiedene Taktraten  
1000mal pro Sekunde: Stopwatches  
5mal pro Sekunde: Blinkende Power-On-LED  
1000mal pro Sekunde: Bumper, ACS, PWM zur Motorsteuerung  
Geschwindigkeitsmessung durch Zählen der Ticks in 0.2 s  
Anpassung der Motorkraft in  $\pm 1$ -Schritten
- Nebenbei: 1 Benutzerprogramm

## 6.3 Multitasking

- *Kooperatives Multitasking*  
Prozesse geben freiwillig Rechenzeit ab
- *Präemptives Multitasking*  
Das Betriebssystem unterbricht laufende Prozesse  
(englisch: *to pre-empt* – jemandem zuvorkommen)

## 6.3 Multitasking

- *Kooperatives Multitasking*  
Prozesse geben freiwillig Rechenzeit ab
- *Präemptives Multitasking*  
Das Betriebssystem unterbricht laufende Prozesse  
(englisch: *to pre-empt* – jemandem zuvorkommen)
- *Scheduler*  
Steuerprogramm, das Prozessen Rechenzeit zuteilt
- *Kontextwechsel*  
Umschalten zwischen zwei Prozessen
- *Round-Robin-Verfahren (Rundlauf)*  
Zuteilung von *Zeitschlitz*en auf einer *Zeitscheibe* an die Prozesse

## 6.3 Multitasking

- *Kooperatives Multitasking*  
Prozesse geben freiwillig Rechenzeit ab
- *Präemptives Multitasking*  
Das Betriebssystem unterbricht laufende Prozesse  
(englisch: *to pre-empt* – jemandem zuvorkommen)
- *Scheduler*  
Steuerprogramm, das Prozessen Rechenzeit zuteilt
- *Kontextwechsel*  
Umschalten zwischen zwei Prozessen
- *Round-Robin-Verfahren (Rundlauf)*  
Zuteilung von *Zeitschlitz*en auf einer *Zeitscheibe* an die Prozesse
- **Ausblick:** Zuteilung von Rechenzeit = wichtiger Spezialfall  
allgemein: Zuteilung von Ressourcen

# Beispiele für Multitasking

## Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

## RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
verschiedene Tasks werden unterschiedlich häufig aufgerufen
- Nebenbei: 1 Benutzerprogramm

# Beispiele für Multitasking

## Quadrocopter-Steuerung *MultiWii*

- Konfiguration durch bedingte Compilierung (Präprozessor)
- In der Hauptschleife wird 50mal pro Sekunde der RC-Task aufgerufen, ansonsten zyklisch einer von bis zu 5 weiteren Tasks.

## RP6-Steuerung

- Konfiguration durch bedingte Compilierung (Präprozessor)
- Lichtschranken an Encoder-Scheiben lösen bei Bewegung Interrupts aus. Die Interrupt-Handler zählen Variable hoch.
- 10000mal pro Sekunde: Timer-Interrupt  
verschiedene Tasks werden unterschiedlich häufig aufgerufen
- Nebenbei: 1 Benutzerprogramm

## Linux 0.01

- Timer-Interrupt: Zähler des aktuellen Tasks wird dekrementiert; Task mit höchstem Zähler bekommt Rechenzeit.
- Wenn es keinen lafbereiten Task mit positivem Zähler gibt, bekommen alle Tasks gemäß ihrer Priorität neue Zähler zugewiesen.
- *keine* harte Echtzeit

# Zombies

# Zombies

Wikipedia:

*Als Zombie wird die fiktive Figur eines zum Leben erweckten Toten (Untoter) oder eines seiner Seele beraubten, willenlosen Wesens bezeichnet. Der Begriff leitet sich von dem Wort nzùmbe aus der zentral-afrikanischen Sprache Kimbundu ab und bezeichnet dort ursprünglich einen Totengeist.*

# Zombies

Wikipedia:

*Als Zombie wird die fiktive Figur eines zum Leben erweckten Toten (Untoter) oder eines seiner Seele beraubten, willenlosen Wesens bezeichnet. Der Begriff leitet sich von dem Wort nzùmbe aus der zentral-afrikanischen Sprache Kimbundu ab und bezeichnet dort ursprünglich einen Totengeist.*

Ein Zombie-Prozeß ist bereits beendet („tot“),  
bekommt keine Rechenzeit mehr („seiner Seele beraubt“),  
hat alle belegten Ressourcen wieder freigegeben („willenlos“),  
wird aber noch in der Prozeßliste geführt („untot“).

# Zombies

Wikipedia:

*Als Zombie wird die fiktive Figur eines zum Leben erweckten Toten (Untoter) oder eines seiner Seele beraubten, willenlosen Wesens bezeichnet. Der Begriff leitet sich von dem Wort nzùmbe aus der zentral-afrikanischen Sprache Kimbundu ab und bezeichnet dort ursprünglich einen Totengeist.*

Ein Zombie-Prozeß ist bereits beendet („tot“),  
bekommt keine Rechenzeit mehr („seiner Seele beraubt“),  
hat alle belegten Ressourcen wieder freigegeben („willenlos“),  
wird aber noch in der Prozeßliste geführt („untot“).

- Warum? —> Ein anderer Prozeß (Elternprozeß) wartet noch auf den Rückgabewert des beendeten Prozesses.

# Zombies

Wikipedia:

*Als Zombie wird die fiktive Figur eines zum Leben erweckten Toten (Untoter) oder eines seiner Seele beraubten, willenlosen Wesens bezeichnet. Der Begriff leitet sich von dem Wort nzùmbe aus der zentral-afrikanischen Sprache Kimbundu ab und bezeichnet dort ursprünglich einen Totengeist.*

Ein Zombie-Prozeß ist bereits beendet („tot“),  
bekommt keine Rechenzeit mehr („seiner Seele beraubt“),  
hat alle belegten Ressourcen wieder freigegeben („willenlos“),  
wird aber noch in der Prozeßliste geführt („untot“).

- Warum? → Ein anderer Prozeß (Elternprozeß) wartet noch auf den Rückgabewert des beendeten Prozesses.
- Schadet das? → Nein.

# Zombies

Wikipedia:

*Als Zombie wird die fiktive Figur eines zum Leben erweckten Toten (Untoter) oder eines seiner Seele beraubten, willenlosen Wesens bezeichnet. Der Begriff leitet sich von dem Wort nzùmbe aus der zentral-afrikanischen Sprache Kimbundu ab und bezeichnet dort ursprünglich einen Totengeist.*

Ein Zombie-Prozeß ist bereits beendet („tot“),  
bekommt keine Rechenzeit mehr („seiner Seele beraubt“),  
hat alle belegten Ressourcen wieder freigegeben („willenlos“),  
wird aber noch in der Prozeßliste geführt („untot“).

- Warum? → Ein anderer Prozeß (Elternprozeß) wartet noch auf den Rückgabewert des beendeten Prozesses.
- Schadet das? → Nein.
- Aber? → Wenn sich Zombie-Prozesse anhäufen, deutet dies auf einen Prozeß hin, der andere Prozesse erzeugt und anschließend „hängt“.

# Zombies

Wikipedia:

*Als Zombie wird die fiktive Figur eines zum Leben erweckten Toten (Untoter) oder eines seiner Seele beraubten, willenlosen Wesens bezeichnet. Der Begriff leitet sich von dem Wort nzùmbe aus der zentral-afrikanischen Sprache Kimbundu ab und bezeichnet dort ursprünglich einen Totengeist.*

Ein Zombie-Prozeß ist bereits beendet („tot“), bekommt keine Rechenzeit mehr („seiner Seele beraubt“), hat alle belegten Ressourcen wieder freigegeben („willenlos“), wird aber noch in der Prozeßliste geführt („untot“).

- Warum? → Ein anderer Prozeß (Elternprozeß) wartet noch auf den Rückgabewert des beendeten Prozesses.
- Schadet das? → Nein.
- Aber? → Wenn sich Zombie-Prozesse anhäufen, deutet dies auf einen Prozeß hin, der andere Prozesse erzeugt und anschließend „hängt“.
- Beispiel: Datenträger entfernt, zugreifender Prozeß „hängt“.  
→ Tochterprozesse werden zu Zombies.

## 6 Echtzeit

### 6.3 Multitasking

Qualitätsaspekte beim Multitasking

# 6 Echtzeit

## 6.3 Multitasking

Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*

# 6 Echtzeit

## 6.3 Multitasking

Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
  - *Latenz*: interaktive Anwendungen
  - *Jitter*: Echtzeitanwendungen
  - *Durchsatz*: Stapelverarbeitung

# 6 Echtzeit

## 6.3 Multitasking

Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe*

# 6 Echtzeit

## 6.3 Multitasking

Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe* (= spezielle *Semaphore*)

# 6 Echtzeit

## 6.3 Multitasking

Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe* (= spezielle *Semaphore*)  
→ kommt gleich

# 6 Echtzeit

## 6.3 Multitasking

### Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe* (= spezielle *Semaphore*)  
→ kommt gleich
- Verschiedene Methoden  
der Priorisierung  
→ später

# 6 Echtzeit

## 6.3 Multitasking

### Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe* (= spezielle *Semaphore*)  
→ kommt gleich
- Verschiedene Methoden  
der Priorisierung  
→ später
- Umgehung der Probleme durch  
speziell geschriebene Software  
(MultiWii, RP6, ...)

# 6 Echtzeit

## 6.3 Multitasking

### Qualitätsaspekte beim Multitasking

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*Mutexe* (= spezielle *Semaphore*)  
→ kommt gleich
- Verschiedene Methoden  
der Priorisierung  
→ später
- Umgehung der Probleme durch  
speziell geschriebene Software  
(MultiWii, RP6, ...)

### Qualitätsaspekte für Netzwerke:

- Verschiedene Anforderungen:  
*Latenz* vs. *Jitter* vs. *Verluste*  
vs. *Durchsatz*
- Ressourcen reservieren:  
*IntServ* mit *Resource Reservation Protocol (RSVP)*
- Klassifizierung und Priorisierung:  
*DiffServ* mit Type-of-Service-Bits  
(IPv4) bzw. Traffic-Class-Bits (IPv6)  
im IP-Header
- Eigenes Protokoll (Telefondienste):  
*Asynchronous Transfer Mode (ATM)*

# 6 Echtzeit

## 6.4 Ressourcen

Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt → Kontextwechsel  
griechisch: *sema* – Zeichen, *pherein* – tragen  
„Eisenbahnsignal“

# 6 Echtzeit

## 6.4 Ressourcen

Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt  $\longrightarrow$  Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann  
  
englisch: *mutual exclusion* – wechselseitiger Ausschluß  
spezieller binärer Semaphor: nur „Besitzer“ darf freigeben

# 6 Echtzeit

## 6.4 Ressourcen

### Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt  $\rightarrow$  Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann
- *Spinlock (busy waiting)*  
leichtgewichtige Alternative zu Kontextwechsel  
englisch: *spin* – rotieren, *lock* Sperre  
*busy waiting* auf etwas Schnelles, z. B. auf einen Semaphor  
Hardware-Unterstützung: Prüfen, ob Variable bestimmten Wert hat;  
wenn ja, auf anderen Wert setzen; andere Prozessoren solange anhalten

# 6 Echtzeit

## 6.4 Ressourcen

### Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt → Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann
- *Spinlock (busy waiting)*  
leichtgewichtige Alternative zu Kontextwechsel
- *Kritischer Abschnitt – critical section*  
Programmabschnitt zwischen Reservierung  
und Freigabe einer Ressource  
→ sollte immer so kurz wie möglich sein

# 6 Echtzeit

## 6.4 Ressourcen

Ressourcen reservieren – Beispiel: `linux-3.7rc1`

- *Semaphor*  
kernel/semaphor.c  
drivers/usb/core/file.c
- *Mutex*  
kernel/mutex.c  
drivers/usb/serial/usb-serial.c
- *Spinlock*  
kernel/spinlock.c  
kernel/semaphor.c, kernel/mutex.c

**Beispiel:** `usb_serial_get_by_index()` – serielle Schnittstelle reservieren  
Datei `linux-3.7-rc1/drivers/usb/serial/usb-serial.c`, ab Zeile 62

```
struct usb_serial *usb_serial_get_by_index (unsigned index)
{
    struct usb_serial *serial;
    mutex_lock (&table_lock); ← exklusiven Zugriff auf Tabelle sichern
    serial = serial_table[index];
    if (serial)
    {
        mutex_lock (&serial->disc_mutex);
        if (serial->disconnected)
        {
            mutex_unlock (&serial->disc_mutex);
            serial = NULL;
        }
        else
            kref_get (&serial->kref);
    }
    mutex_unlock (&table_lock); ← exklusiven Zugriff auf Tabelle wieder freigeben
    return serial;
}
```

```

mutex_lock() - Ressource beanspruchen, nichts warten
Datei linux-3.7-rc1/drivers/tty/serial/usb-serial.c, ab Zeile 62
void __sched mutex_lock (struct mutex *lock)
{
    might_sleep ();
    __mutex_lockpath_lock (&lock->count, __mutex_lock_sleeppath);
    mutex_set_owner (lock);
}

Datei linux-3.7-rc1/arch/arm/include/asm/mutex_32.h, ab Zeile 24
Macro-Definition für __mutex_lockpath_lock (separiert)
Assembler:
    lock dec (lock->count)
    jne 1
    call __mutex_lock_sleeppath

Datei linux-3.7-rc1/kernel/mutex.c, ab Zeile 398
static __used noinline void __sched
__mutex_lock_sleeppath (atomic_t *lock_count)
{
    struct mutex *lock = container_of (lock_count, struct mutex, count);
    __mutex_lock_common (lock, TASK_UNINTERRUPTIBLE, 0,
        NULL, _RET_IP_);
}

Datei linux-3.7-rc1/kernel/mutex.c, ab Zeile 132
static inline int __sched
__mutex_lock_common (struct mutex *lock, long state, unsigned int subclass,
    struct lockdep_map *nest_lock, unsigned long ip)
{
    struct task_struct *task = current;
    struct mutex_wailer wailer;
    unsigned long flags;

    preempt_disable ();
    mutex_acquire_nest (&lock->dep_map, subclass, 0, nest_lock, ip);
    /* -- */
    spin_lock_mutex (&lock->wait_lock, flags);
    debug_mutex_lock_common (lock, &wailer);
    debug_mutex_add_wailer (lock, &wailer, task, thread_info (task));
    /* add waiting tasks to the end of the waitqueue (FIFO): */
    list_add_tail (&wailer.task, &lock->wait_list);
    wailer.task = task;

    if (atomic_cmpxchg (&lock->count, -1) == 1)
        goto done;

    lock_contended (&lock->dep_map, ip);

    for (;;)
    {
        /*
         * Lets try to take the lock again -- this is needed even if
         * we get here for the first time (shortly after failing to
         * acquire the lock, to make sure that we get a wakeup once
         * it's unlocked. Later on, if we sleep, this is the
         * condition that gives us the lock. We sleep if it == -1, so
         * that when we release the lock, we properly wake up the
         * other waiters:
         */
        if (atomic_cmpxchg (&lock->count, -1) == 1)
            break;

        /*
         * got a signal? (This code gets eliminated in the
         * TASK_UNINTERRUPTIBLE case.)
         */
        if (unlikely (signal_pending_state (state, task)))
        {
            mutex_remove_wailer (lock, &wailer, task, thread_info (task));
            mutex_release (&lock->dep_map, 1, ip);
            spin_unlock_mutex (&lock->wait_lock, flags);
            debug_mutex_free_wailer (&wailer);
            preempt_enable ();
            return -EINTR;
        }
        __set_task_state (task, state);

        /* didn't get the lock, go to sleep: */
        spin_unlock_mutex (&lock->wait_lock, flags);
        schedule_preempt_disabled ();
        spin_lock_mutex (&lock->wait_lock, flags);
    }

done:
    lock_acquired (&lock->dep_map, ip);
    /* got the lock - major! */
    mutex_remove_wailer (lock, &wailer, current, thread_info ());
    mutex_set_owner (lock);

    /* set it to 0 if there are no waiters left: */
    if (likely (list_empty (&lock->wait_list)))
        atomic_set (&lock->count, 0);

    spin_unlock_mutex (&lock->wait_lock, flags);
    debug_mutex_free_wailer (&wailer);
    preempt_enable ();

    return 0;
}

```

exklusiven Zugriff  
auf Mutex sichern

exklusiven Zugriff auf Mutex  
wieder freigeben

```

spin_lock_mutex(); // Mutex beanspruchen, falls busy waiting
Datei linux-3.7-rc1/kernel/mutex.c, ab Zeile 12
#define spin_lock_mutex(lock, flags) \
do { \
    { \
        spin_lock(lock); \
        (void) (flags); \
    } \
} while (0)

Datei linux-3.7-rc1/kernel/spinlock.h, ab Zeile 283
static inline void spin_lock(spinlock_t *lock) \
{ \
    raw_spin_lock(&lock->lock); \
}

Datei linux-3.7-rc1/kernel/spinlock.h, Zeile 170
#define raw_spin_lock(lock) _raw_spin_lock(lock)

Datei linux-3.7-rc1/include/linux/spinlock_api_smp.h, Zeile 47
#define _raw_spin_lock(lock) __raw_spin_lock(lock)

Datei linux-3.7-rc1/kernel/spinlock.c, ab Zeile 46 (expandiert):
void __lockfunc __raw_spin_lock(spinlock_t *lock) \
{ \
    for (;;) \
    { \
        preempt_disable(); \
        if (likely(!do_raw_spin_trylock(lock))) \
            break; \
        preempt_enable(); \
        if (!lock)->break; lock \
        (lock)->break; lock = 1; \
        while (!raw_spin_can_lock(lock) && (lock)->break; lock) \
            arch_spin_relax(&lock->raw_lock); \
    } \
    (lock)->break; lock = 0; \
}

Datei linux-3.7-rc1/include/linux/spinlock.h, ab Zeile 150:
static inline int do_raw_spin_trylock(raw_spinlock_t *lock) \
{ \
    return arch_spin_trylock(&(lock)->raw_lock); \
}

Datei arch/x86/include/asm/spinlock.h, ab Zeile 116:
static __always_inline int arch_spin_trylock(arch_spinlock_t *lock) \
{ \
    return __ticket_spin_trylock(lock); \
}

Datei arch/x86/include/asm/spinlock.h, ab Zeile 65:
static __always_inline int __ticket_spin_trylock(arch_spinlock_t *lock) \
{ \
    arch_spinlock_t old, new; \
 \
    old.tickets = ACCESS_ONCE(lock->tickets); \
    if (old.tickets.head != old.tickets.tail) \
        return 0; \
 \
    new.head_tail = old.head_tail + (1 << TICKET_SHIFT); \
 \
    /* cmpxchg is a full barrier, so nothing can move before it */ \
    return cmpxchg(&(lock)->head_tail, old.head_tail, new.head_tail) == old.head_tail; \
}

Datei arch/x86/include/asm/cmpxchg.h, ab Zeile 147:
#define cmpxchg(ptr, old, new) \
__cmpxchg(ptr, old, new, sizeof(-ptr))

Datei arch/x86/include/asm/cmpxchg.h, ab Zeile 131:
#define __cmpxchg(ptr, old, new, size) \
__raw_cmpxchg((ptr), (old), (new), (size), LOCK_PREFIX)

Datei arch/x86/include/asm/cmpxchg.h, ab Zeile 110:
asm volatile (lock "cmpxchgl %2,%1" \
: "=a"(_ret), "=m"(*_ptr) \
: "Y"(_new), "0"(_old) \
:"memory");

```

atomarer und exklusiver  
Zugriff auf Spinlock  
durch Hardware-Unterstützung

# 6 Echtzeit

## 6.4 Ressourcen

### Ressourcen reservieren

- *Semaphor*  
gemeinsame Variable mehrerer Prozesse  
zur Regelung des Zugriffs auf eine Ressource  
Ressource belegt → Kontextwechsel
- *Mutex*  
Mechanismus, damit immer nur ein Prozeß gleichzeitig  
auf eine Ressource zugreifen kann
- *Spinlock (busy waiting)*  
leichtgewichtige Alternative zu Kontextwechsel
- *Kritischer Abschnitt – critical section*  
Programmabschnitt zwischen Reservierung  
und Freigabe einer Ressource  
→ sollte immer so kurz wie möglich sein

## 6 Echtzeit

### 6.4 Ressourcen

*Verklemmungen*: Gegenseitiges Blockieren von Ressourcen

- *Deadlock*: Prozeß wartet
- *Livelock*: Prozeß macht andere Dinge  
(z. B. *busy waiting*)

# 6 Echtzeit

## 6.4 Ressourcen

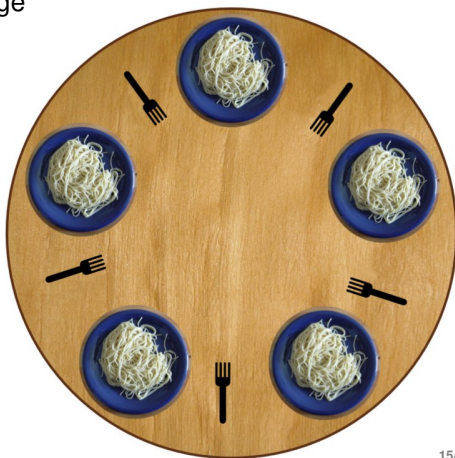
**Verklemmungen:** Gegenseitiges Blockieren von Ressourcen

- **Deadlock:** Prozeß wartet
- **Livelock:** Prozeß macht andere Dinge  
(z. B. *busy waiting*)

Beispiel: Philosophenproblem

- 5 Philosophen, 5 Gabeln
- 2 Gabeln zum Essen notwendig
- Wer essen will, nimmt eine Gabel und wartet notfalls auf die zweite.
- Keiner legt eine einzelne Gabel wieder zurück.

Jeder hält 1 Gabel → **Verklemmung**



# 6 Echtzeit

## 6.4 Ressourcen

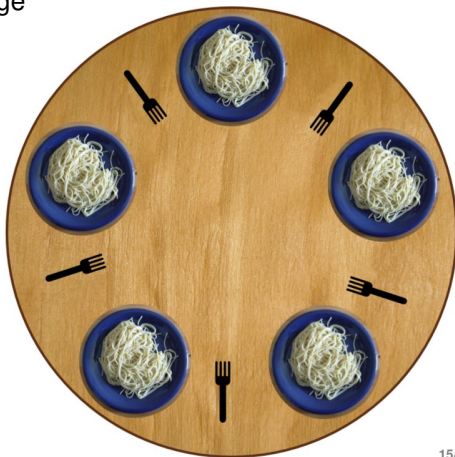
**Verklemmungen:** Gegenseitiges Blockieren von Ressourcen

- **Deadlock:** Prozeß wartet
- **Livelock:** Prozeß macht andere Dinge  
(z. B. *busy waiting*)

Beispiel: Philosophenproblem

- 5 Philosophen, 5 Gabeln
- 2 Gabeln zum Essen notwendig
- Wer essen will, nimmt eine Gabel und wartet notfalls auf die zweite.
- Keiner legt eine einzelne Gabel wieder zurück.

Jeder hält 1 Gabel → **Verklemmung**  
schweigen → **Deadlock**  
philosophieren weiter → **Livelock**



# 6 Echtzeit

## 6.4 Ressourcen

*Verklemmungen*: Gegenseitiges Blockieren von Ressourcen

- *Deadlock*: Prozeß wartet
- *Livelock*: Prozeß macht andere Dinge  
(z. B. *busy waiting*)

Beispiel: Philosophenproblem

Bedingungen für Verklemmungen:

- Exklusivität
- *hold and wait*
- Entzug nicht möglich
- zirkuläre Blockade

# 6 Echtzeit

## 6.4 Ressourcen

*Verklemmungen*: Gegenseitiges Blockieren von Ressourcen

- *Deadlock*: Prozeß wartet
- *Livelock*: Prozeß macht andere Dinge  
(z. B. *busy waiting*)

Beispiel: Philosophenproblem

Bedingungen für Verklemmungen:

- |                        |   |
|------------------------|---|
| • Exklusivität         | → Spooling  |
| • <i>hold and wait</i> | → simultane Zuteilung                             |
| • Entzug nicht möglich | → Prozesse suspendieren, beenden, <i>Rollback</i> |
| • zirkuläre Blockade   | → Reihenfolge abhängig von Ressourcen             |

# 6 Echtzeit

## 6.5 Prioritäten

Linux 0.01

- Timer-Interrupt: Zähler des aktuellen Prozesses wird dekrementiert; Prozeß mit höchstem Zähler bekommt Rechenzeit.
- Wenn es keinen lafbereiten Prozeß mit positivem Zähler gibt, bekommen alle Prozesse gemäß ihrer *Priorität* neue Zähler zugewiesen.
- *keine* harte Echtzeit

→ *dynamische Prioritätenvergabe*:  
Rechenzeit hängt vom Verhalten des Prozesses ab

Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.

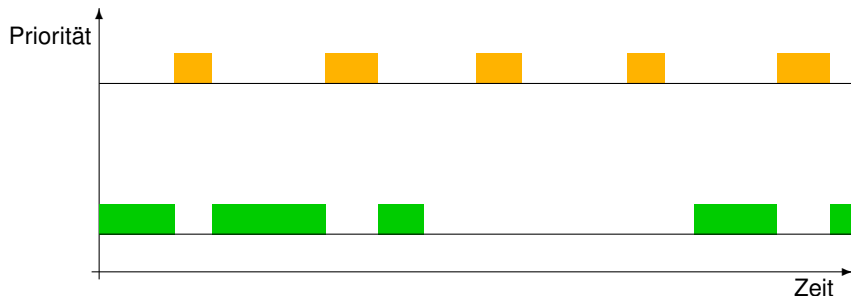
→ *statische Prioritätenvergabe*

# 6 Echtzeit

## 6.5 Prioritäten

### Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.

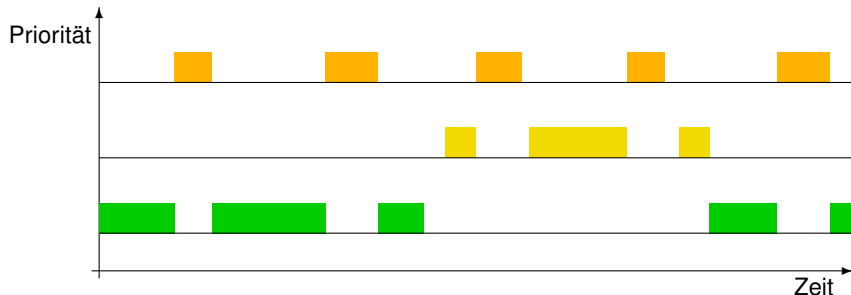


# 6 Echtzeit

## 6.5 Prioritäten

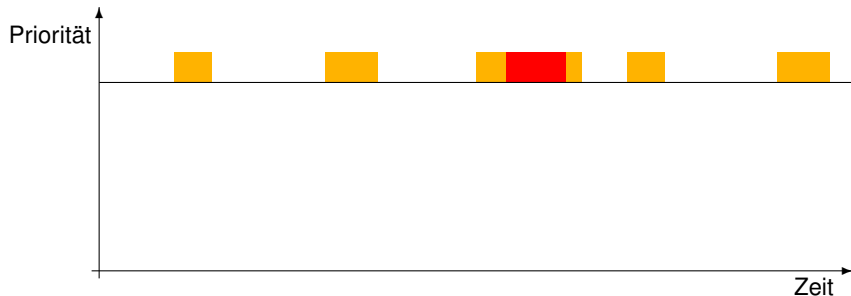
### Echtzeitbetriebssysteme

- Prozesse können einen festen Anteil an Rechenzeit bekommen.
- Bei Ereignissen können Prozesse hoher Priorität Prozesse niedriger Priorität unterbrechen, aber nicht umgekehrt.



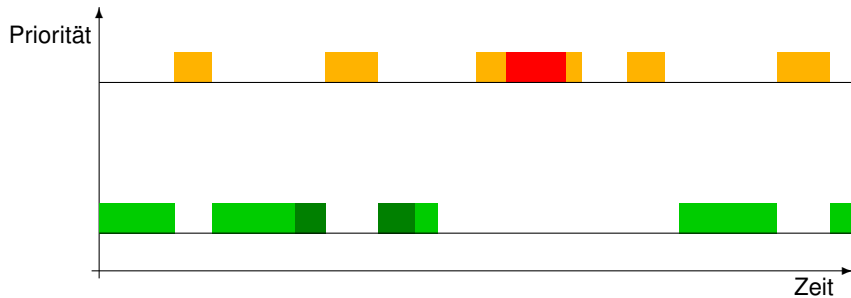
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen



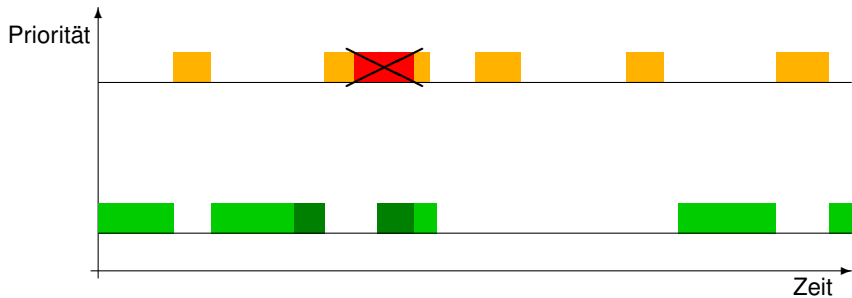
## 6 Echtzeit

## 6.5 Prioritäten und Ressourcen



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen



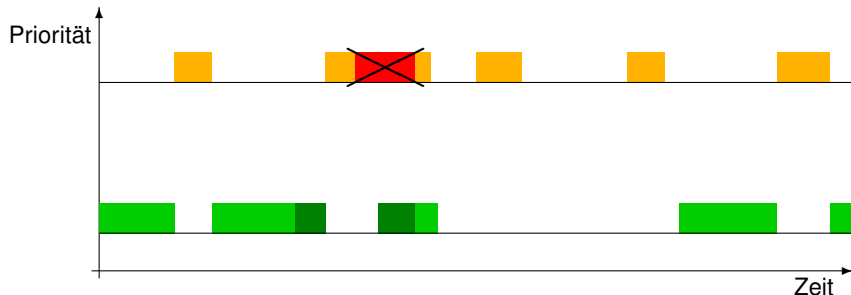
# 6 Echtzeit

## 6.5 Prioritäten und Ressourcen

Der höher priorisierte Prozeß bewirkt selbst, daß er eine Ressource verspätet bekommt.

→ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



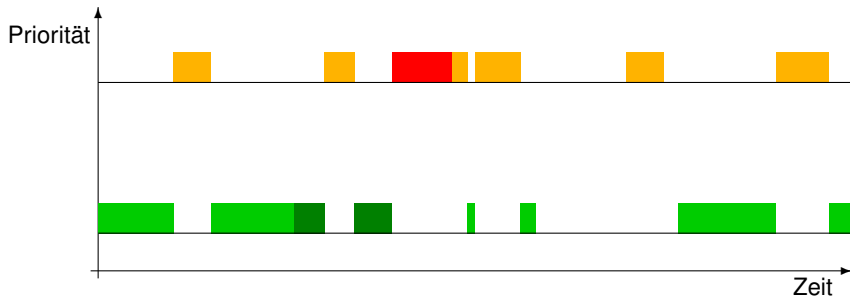
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Der höher priorisierte Prozeß bewirkt selbst, daß er eine Ressource verspätet bekommt.

→ *begrenzte Prioritätsinversion*

maximale Verzögerung: Länge des kritischen Bereichs



## 6 Echtzeit

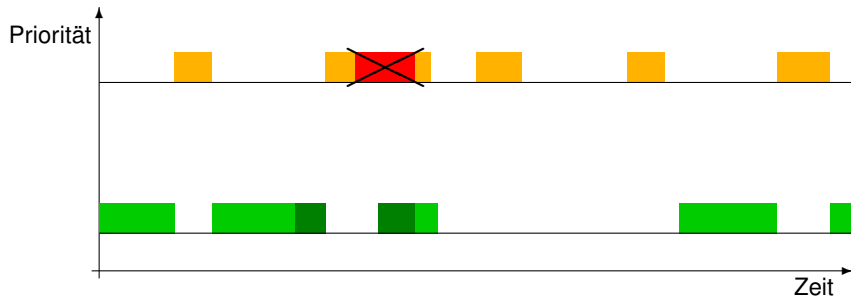
### 6.5 Prioritäten und Ressourcen

*unbegrenzte Prioritätsinversion*

## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

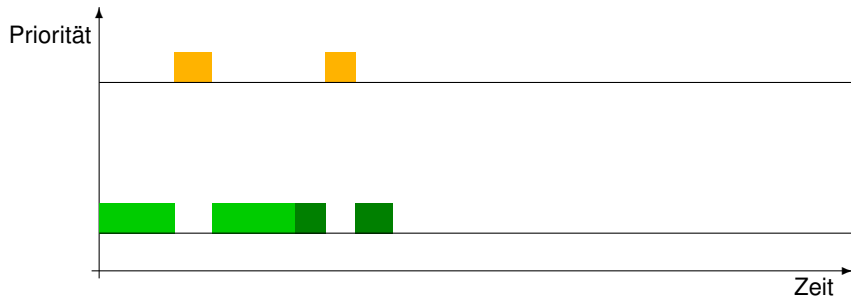
*unbegrenzte Prioritätsinversion*



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

*unbegrenzte Prioritätsinversion*

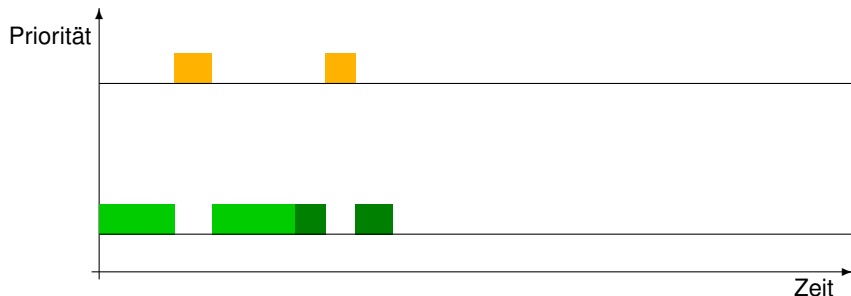


## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

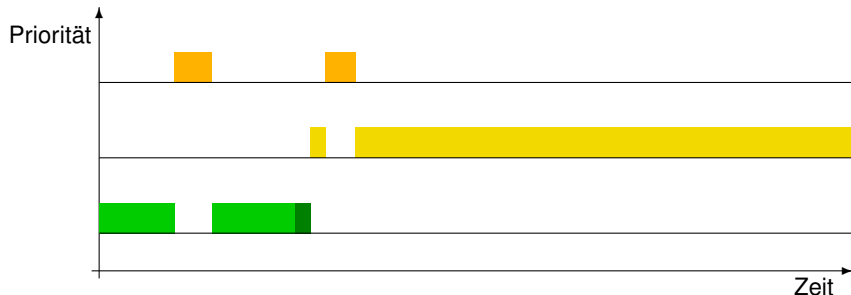


# 6 Echtzeit

## 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*



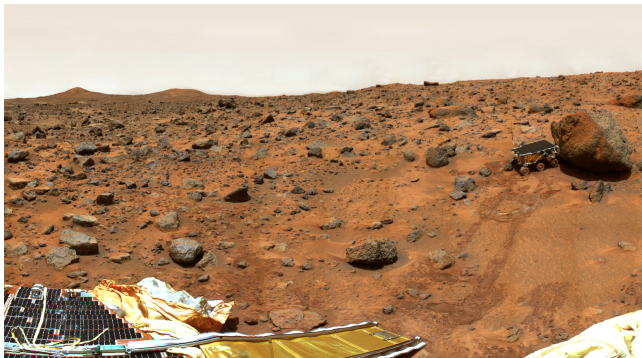
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

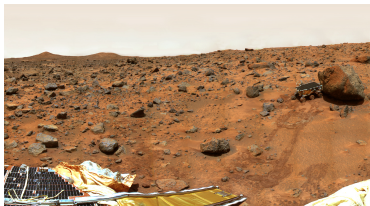
—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.

[http://research.microsoft.com/en-us/um/people/mbj/Mars\\_Pathfinder/](http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/)



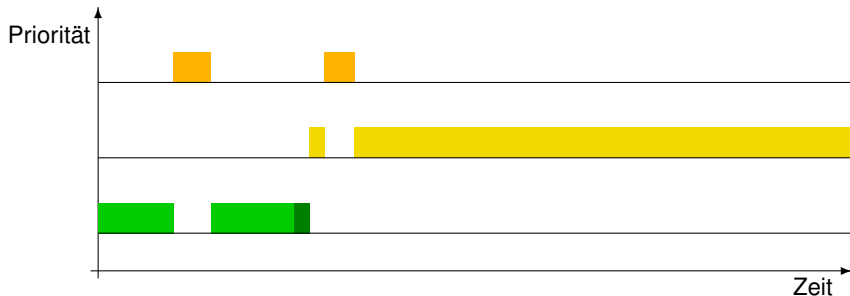
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Inheritance – Prioritätsvererbung*



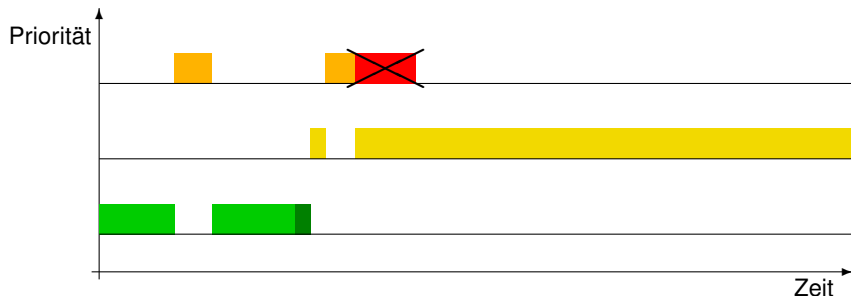
# 6 Echtzeit

## 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Inheritance – Prioritätsvererbung*





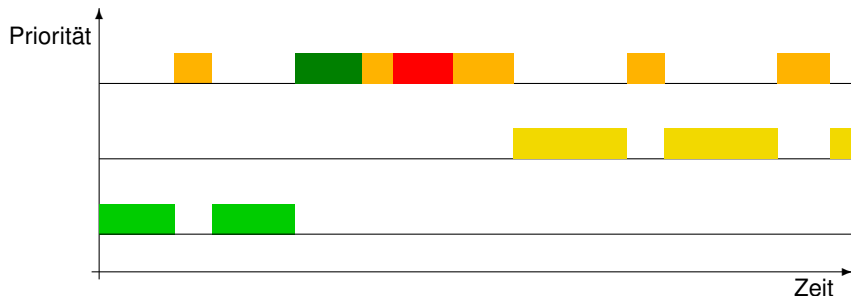
## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

→ *unbegrenzte Prioritätsinversion*

Gegenmaßnahme: *Priority Ceiling – Prioritätsobergrenze*



## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
- *Priority Ceiling – Prioritätsobergrenze*  
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.

## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
- *Priority Ceiling – Prioritätsobergrenze*  
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.

} nur möglich, wenn  
Mutexe im Spiel sind

## 6 Echtzeit

### 6.5 Prioritäten und Ressourcen

Ein Prozeß mit mittlerer Priorität bewirkt, daß ein Prozeß mit hoher Priorität eine Ressource überhaupt nicht bekommt.

—→ *unbegrenzte Prioritätsinversion*

Beispiel: Beinahe-Verlust der Marssonde *Pathfinder* im Juli 1997

Gegenmaßnahmen

- *Priority Inheritance – Prioritätsvererbung*  
Der Besitzer des Mutex erbt die Priorität des Prozesses, der auf den Mutex wartet.
  - *Priority Ceiling – Prioritätsobergrenze*  
Der Besitzer des Mutex bekommt sofort die Priorität des höchstmöglichen Prozesses, der evtl. den Mutex benötigen könnte.
  - *Priority Aging*  
Die Priorität wächst mit der Wartezeit.
- } nur möglich, wenn  
Mutexe im Spiel sind