Industrielle Steuerungstechnik und Echtzeitsysteme

Prof. Dr.-Ing. Bernardo Wagner

Grundlagen Echtzeitsysteme
Speicherprogrammierbare Steuerungen
Programmierung eingebetteter Computer
Echtzeitbetriebssystem, Middleware und Feldbusse

Stand April 2015

Leibniz Universität Hannover
Fachgebiet Echtzeitsysteme
www.rts.uni-hannover.de

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Industrielle Steuerungstechnik und Echtzeitsysteme

Prof. Dr.-Ing. Bernardo Wagner

Einführung in die systematische Entwicklung von Echtzeitsystemen mit einem Schwerpunkt im Bereich der Programmierung speicherprogrammierter Steuerungen (IEC61131) und eingebetteter Mikrocontroller, dem Einsatz von Feldbussen (CAN, Interbus, RTnet), der Vorstellung eines Echtzeitbetriebs-system (Xenomai) und einer Echtzeit-Middleware (RACK).

In den begleitenden Übungen werden kleinere Aufgaben im Umfang und im Niveau von Prüfungsaufgaben behandelt.

Vorkenntnisse werden erwartet in der Digitaltechnik, insbesondere der Funktionsweise von Digitalrechnern, und in mindestens einer höheren Programmiersprache, wie Java oder C.

Die Teilnehmer der Lehrveranstaltung kommen aus unterschiedlichen Studiengängen: Elektrotechnik, Mechatronik, Wirtschaftsingenieurwesen, (Technische) Informatik, Maschinenbau und Lehramt an berufsbildenden Schulen. Entsprechend sind Lücken im Vorwissen und Überlappungen mit anderen Lehrveranstaltungen nicht vermeidbar.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Neue Vorlesungsinhalte seit SS 2012!

Die Vorlesung wurde im Sommersemester 2012 inhaltlich deutlich überarbeitet. Die Prüfungsinhalte sind entsprechend angepasst worden. Bitte beachten Sie dies bei der Prüfungsvorbereitung!

Die Beschreibung verteilter Steuerungen nach IEC61499 wird in dieser Lehrveranstaltung nicht mehr behandelt und wurde stark gekürzt in die Vorlesung „Entwurf diskreter Steuerungen“ integriert.

Die Vorlesung „Industrielle Steuerungstechnik“ wurde um den allgemeinen Aspekt der Echtzeitsysteme erweitert und heißt seither „Industrielle Steuerungstechnik und Echtzeitsysteme“. Neu hinzu gekommen sind, neben allgemeinen Definitionen zum Thema Echtzeitsysteme, die Vorstellung eines konkreten Echtzeit-Betriebssystems für eingebettete Mikrocontroller und einer Echtzeit-Middleware für verteilte Systeme.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Was sind Echtzeitsysteme?

Bei Echtzeitsystemen muss das Ergebnis nicht nur korrekt sein, sondern es muss auch innerhalb einer fest definierten Zeitschranke rechtzeitig erzielt werden. Man unterscheidet harte und weiche Echtzeitsysteme. Bei weichen Echtzeitsystemen sinkt der Nutzen bzw. die Qualität nach Erreichen der Zeitschranke (z.B. Bildstörungen bei der Videodarstellung). Bei harten Echtzeitsystemen sind Ergebnisse, die nach Erreichen der Zeitschranke eintreffen, unbrauchbar und können zu erheblichen Schäden führen (z.B. Flugzeuginstabilitäten oder Unfall nach verspäteter Ansteuerung von Aktoren).

Klassische Beispiele sind Steuer- und Regelgeräte im Auto (wie Airbag, ABS, ESP usw.). Im Fall eines Unfalls muss beispielsweise der Airbag nach der Erkennung des Aufpralles durch einen Sensor innerhalb weniger Millisekunden aufblasen, ansonsten hat das System versagt. *)

Echtzeitsysteme sind auch in anderen Bereichen der Industrie und im Alltag zu finden: Roboter, Medizingeräte, Fahrstühle, Handys, Kraftwerke, Flugzeuge, Produktionssysteme, Waschmaschinen, Videokonferenzsysteme usw.

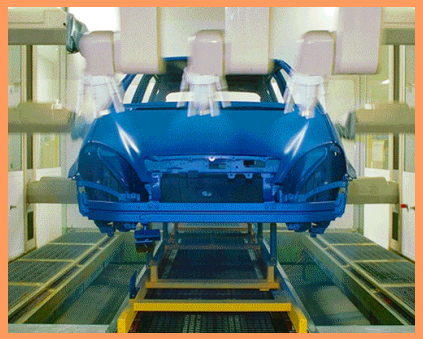
*) Es wäre aber genauso fehlerhaft, wenn der Airbag zu früh oder unbegründet auslöst (z.B. ausgelöst durch ein Schlagloch oder bereits durch starkes Bremsen).

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Echtzeitsysteme in der Anwendung

Produktionstechnik

Fertigungssteuerung



SPS

Elektrogeräte

Ablaufsteuerung

ASIC / FPGA

Echtzeitsysteme

Kfz-Elektronik

Motorsteuerung

Mikrocontroller

Verkehrstechnik

Leitsteuerung

Prozessrechner

. . .

Verfahrenstechnik

Prozesssteuerung

Industrie-PC

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

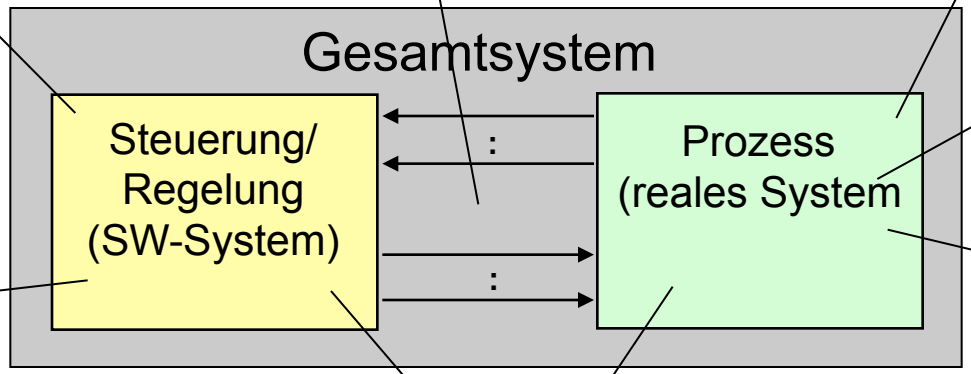
Echtzeitsysteme sind reaktive Systeme

Modellierung: Statecharts, Petri-Netze, IEC 61499, ...

Implementierung: IEC 61131, RT-Java, C/C++ ...

Nicht terminierend und nebenläufig

Kommunikation (intern/extern) mit Zeitbedingungen

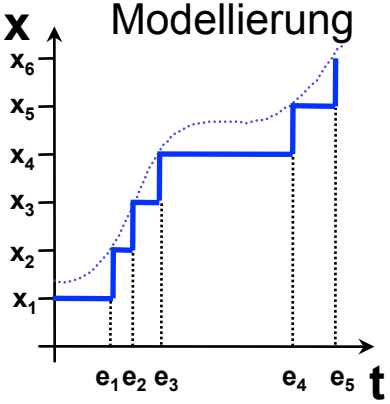


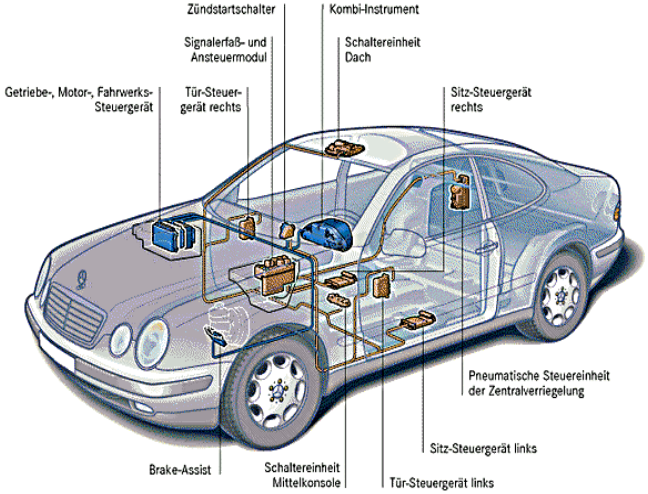
Räumlich verteilte Prozesskomponenten

Kontinuierliche Zustandsgrößen

Zeitgleiche aber vernetzte Vorgänge

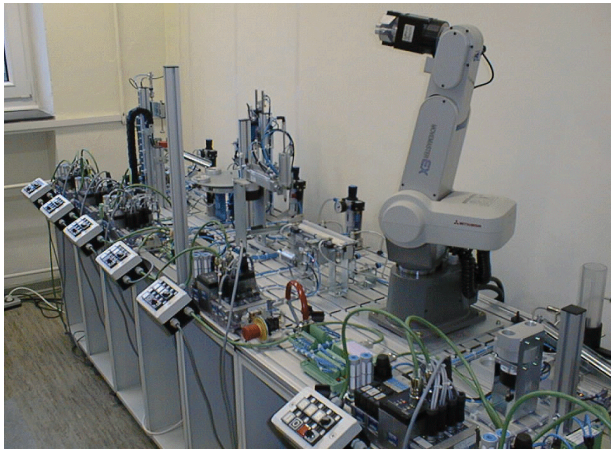
Ereignisdiskrete Modellierung





Eingebettete Echtzeitsysteme

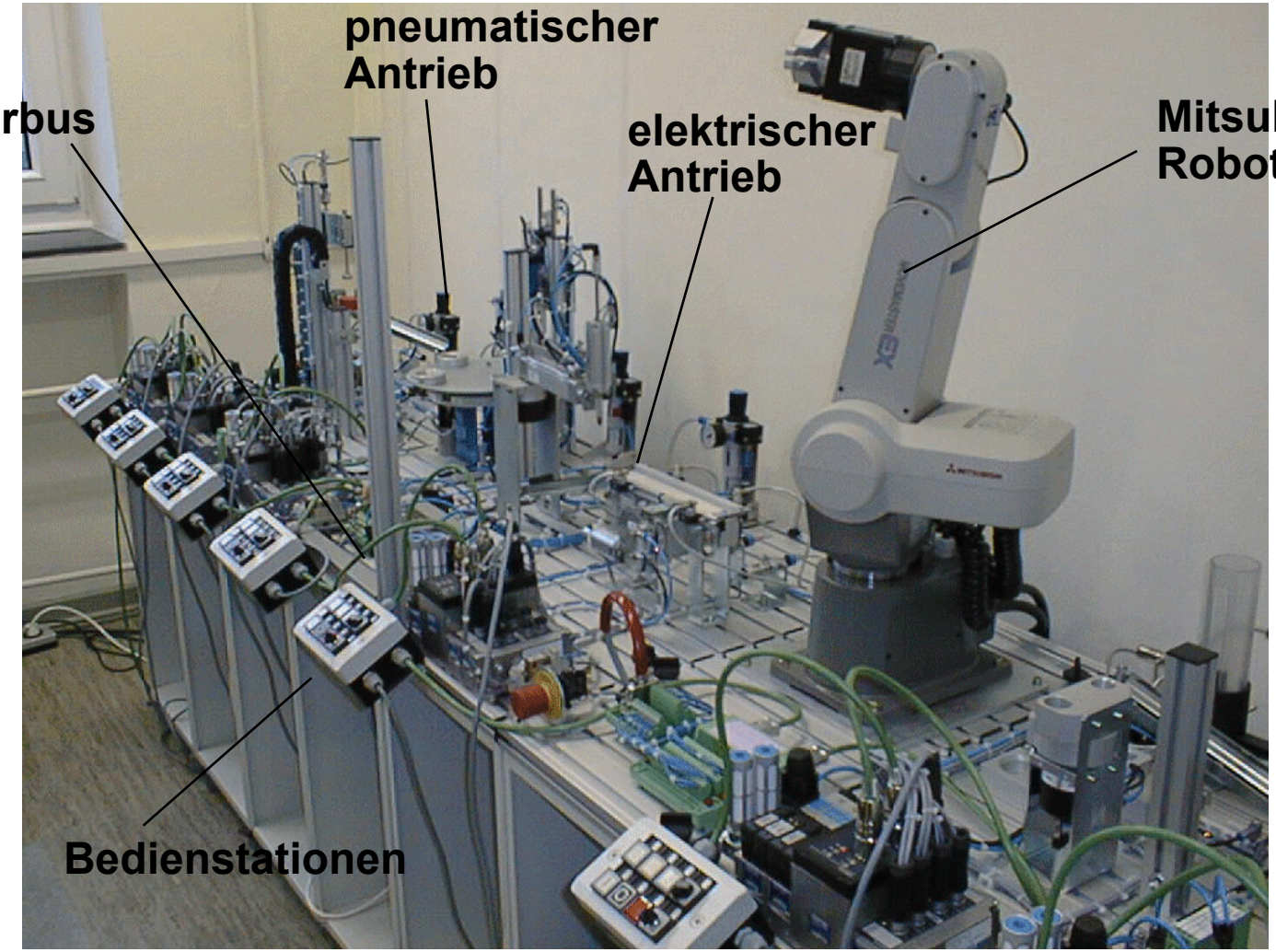
Der Entwurf und die Modellierung ereignisdiskreter Systeme wird in der Vorlesung Entwurf diskreter Steuerungen im Wintersemester behandelt!



Industrielle Echtzeitsysteme

Modellanlage eines Echtzeitsystems

(hybride, industrielle Steuerung)



Zur praktischen Vertiefung der Inhalte dieser Vorlesung wird das Oberstufenlabor Steuerungstechnik im Sommersemester angeboten.

Gliederung der Vorlesung

1. Allgemeine Einführung
2. Grundlagen Echtzeitsysteme
3. Steuerungssysteme (Industrieroboter, NC, SPS ..)
4. Speicherprogrammierbare Steuerungen (IEC61131-3)
5. Eingebettete Computersysteme
6. Echtzeitbetriebssysteme mit Beispiel Linux/Xenomai
7. Kommunikation in Echtzeit am Beispiel von CAN, Interbus, RTnet und der Echtzeit-Middleware RACK

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Vorlesungsmanuskript

Das Folienskript enthält alle Folien der Vorlesung (jeweils die Version zu Beginn des Sommersemesters). In der spezifischen Vorbereitung und Durchführung der Vorlesung können sich Änderungen ergeben. Die jeweils aktuellen Folienversionen werden im Internet bereitgestellt.

Die Folien werden ergänzt durch eingestreute Beispiele, die an der Tafel im Dialog mit den Hörern der Vorlesung entwickelt werden. Hierzu müssen eigene Aufzeichnungen angefertigt werden.

Gegen Ende des Sommersemesters werden in den Übungen alte Prüfungsaufgaben zur individuellen Klausurvorbereitung verteilt.

Aktuelle Informationen, die Folien und die SW-Tools finden sich im Internet in Stud.IP:

http://www.uni-hannover.de/de/studium/elearning/

Buchempfehlungen

Wörn, H. und Brinkschulte, U.: Echtzeitsysteme.
Springer-Verlag Berlin Heidelberg 2005

John, K.-H., Tiegelkamp, M.: SPS Programmierung mit IEC 61131-3.
Springer-Verlag, Berlin, Heidelberg 2009
-> in der TIB als pdf-Dokument verfügbar

Reißenweber, B.: Feldbussysteme zur industriellen Kommunikation.
Oldenbourg Industrieverlag München 2008

Termine Sommersemester 2015

Grobe, unverbindliche Planung

Nr.	Termin	Einheiten	Inhalt	Kommentar
1	14.4.	3	1. Einführung und 2. Echtzeitsysteme	Keine Übungen!
2	21.4.	3	3. Steuerungssysteme	Keine Übungen!
3	28.4.	2	4. SPS	
4	5.5.	2	4. SPS	
5	12.5.	2	4. SPS	
6	19.5.	2	4. SPS	
7	26.5.	2	-	Exkursionswoche Pfingsten
8	2.6.	2	5. Eingebettete Computersysteme	
9	9.6.	-	5. Eingebettete Computersysteme	
10	16.6.	2	6. Echtzeitbetriebssysteme	
11	23.6.	2	6. Echtzeitbetriebssysteme	
12	30.6.	2	7. Kommunikation in Echtzeit	
13	7.7.	2	7. Kommunikation in Echtzeit	
14	14.7.	2	Reserve	
15	21.7.	-		Übungen + Klausurvorbereitung – Frau Schulz (3)

Gesamtsumme Vorlesung SS: 14 Einheiten (2 SWS) + 1 Klausuraufgaben (2 SWS)

Die Übungen zur Vorlesung werden von Frau **M.Sc. Irina Schulz** gehalten.

Kapitel 2: Echtzeitsysteme

2.1 Grundbegriffe

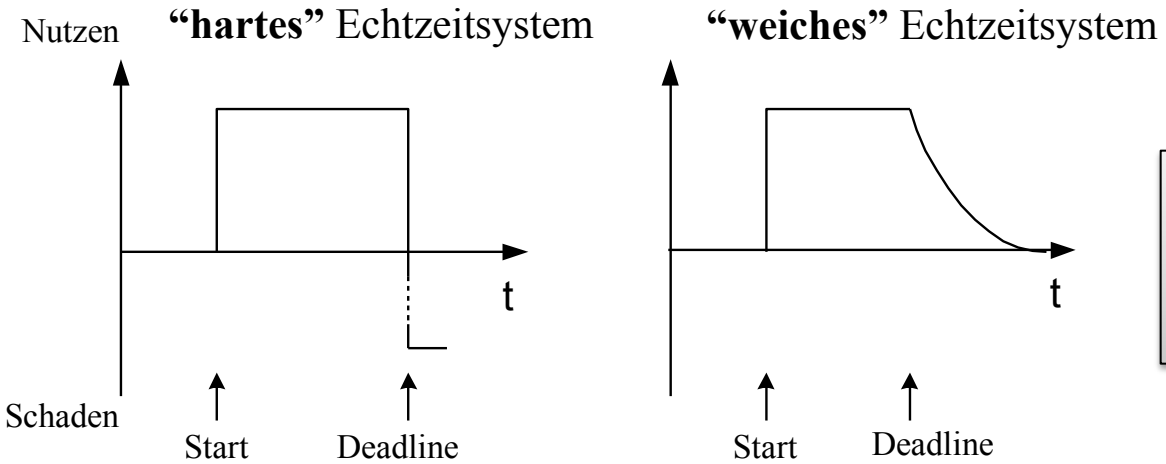
2.2 Programmierung von Echtzeitsystemen

2.3 Organisation von Echtzeitsystemen

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Was ist ein Echtzeitsystem?

- Echtzeitsysteme sind technische Systeme, die korrekte Reaktionen innerhalb eines definierten Zeitlimits produzieren müssen.
- Wenn die Reaktionen diese Zeitlimits überschreiten, dann resultieren daraus Leistungseinbußen oder Fehlfunktionen.
- Ein logisch korrektes Ergebnis zum falschen Zeitpunkt kann die Sicherheit eines Systems gefährden.
- „Echtzeit“ oder „Realzeit“ bedeutet, den echten bzw. realen Zeitabläufen entsprechend. Eine Zeitdehnung oder Raffung ist nicht zulässig.
- Man unterscheidet harte und weiche Echtzeitsysteme.



Echtzeit oder synomym Realzeit (engl. real time) wird häufig mit RT abgekürzt:
RTS = Real Time System
RTOS = Real Time Operating System

Harte vs. weiche Echtzeit

- **Weiche Echtzeitsysteme**
  Es gibt zeitliche Fristen bis zu denen ein Ergebnis berechnet werden muss. Werden diese Fristen nicht eingehalten, hat dies negativen Einfluss auf die Qualität bzw. die Dienstgüte. Es gibt einen „weichen“ Übergang bis zum Totalausfall.
  Beispiel: Video- oder Audio-Übertragung
  Mögliche Folgen: Bild ruckelt, Artefakte entstehen.

- **Harte Echtzeitsysteme**
  Kann die Berechnungsfrist nicht eingehalten werden, hat dies möglicherweise katastrophale Folgen. Die Grenze ist „hart“.
  Beispiel: Raketen- oder Flugzeugsteuerung
  Mögliche Folgen: Absturz und Tod von Menschen.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Definition von Echtzeit nach DIN 44300

[1985]

„Echtzeitbetrieb ist ein Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig derart betriebsbereit sind, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen.“

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Missverständnisse bei Echtzeitsystemen

- Echtzeitsysteme sind nicht zwangsläufig besonders “schnelle” Systeme. Entscheidend ist die garantierte Einhaltung der Zeitschranken und die sehr hohe Systemzuverlässigkeit. Eingebettete Echtzeitsysteme sind häufig sogar auf Basis vergleichsweise leistungsschwacher Mikrocomputer realisiert.
- Schnellere Computerhardware ist keine Lösung des Echtzeitproblems. Nicht vorhersehbare Verzögerungen in der Ausführung haben Ihre Ursachen in nicht-deterministischen Ausführungskonzepten (z.B. nonpreemptives Scheduling, zufallsbasierter Medienzugriff bei Ethernet) oder mangelnde Synchronisation von Uhren.
- Echtzeitsysteme werden heute in der Regel nicht mehr in Assembler programmiert. Der größte Teil wird in Hochsprachen, vor allem C, oder den SPS-Fachsprachen programmiert.
- Hundertprozentige Garantien, z.B. zur Einhaltung von Reaktionszeiten, kann es wegen möglicher Hardwarefehler und nie vollständig vermeidbarer Programmierfehler grundsätzlich nicht geben! Bei der Entwicklung von Echtzeitsystemen muss deshalb zusätzlich, beispielsweise durch redundante Auslegung der Software und Hardware, eine möglichst geringe Fehlerwahrscheinlichkeit erreicht werden. Echtzeitsysteme benötigen deshalb die Angabe der geforderten und nachweisbaren Fehlerwahrscheinlichkeit.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Realisierungsformen von Echtzeitsystemen

- Ein Echtzeitsystem kann in Hardware (z.B. FPGA oder Hydraulische Steuerung) sowie mit geeigneter Software auf einem Digitalrechner (z.B. Mikrocontroller, SPS) realisiert werden.
- Computer, die zur Automatisierung (Steuerung und Regelung) technischer Einrichtungen (Maschinen, Anlagen, Fahrzeuge usw.) eingesetzt werden, sind praktisch immer Echtzeitsysteme. Die zeitlichen Anforderungen ergeben sich aus dem Ein- und Ausgabeverhalten eines „echten“ technischen Prozess und der Automatisierungsaufgabe und können sehr unterschiedlich sein (< 1 ms bis mehrere Stunden).
- Beispiele für Echtzeitsysteme:
  - Harte Echtzeitsysteme:
    - Airbag, Anti-Blockier-System (ABS), ESP, ACC etc. (Mikrocontroller)
    - Industrielle Fertigungssteuerung (Speicherprogrammierbare Steuerung)
    - Heizungsregler (ASIC/FPGA oder Mikrocontroller)
    - Motorsteuerung (Relaissteuerung)
    - Digitaler Messdatenlogger (Digitaler Signalprozessor)
    - Prozessleitsteuerung (Prozessrechner)
    - Maschinensteuerung (CNC)
  - Weiche Echtzeitsysteme:
    - TV-Sportübertragung
    - Navigationssystem im Auto
    - Telefonanlage

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Eingebettetes System

- Technisches System, das durch ein integriertes, programmiertes Computersystem auf Basis eines Mikroprozessor gesteuert wird.
- Das Computersystem ist nach außen in der Regel nicht direkt sichtbar. Man spricht dann von einem eingebetteten Computersystem.
- Das Computersystem ist über spezielle Schnittstellen in Form von Sensoren und Aktoren mit dem technischen System vielfältig verbunden.
- Häufig werden leistungsärmere, energieeffiziente aber robuste Mikroprozessoren mit Einschränkungen in Bezug auf Rechenleistung und Speicherfähigkeit eingesetzt.
- Praktisch immer werden Anforderung zur harten oder wenigstens weichen Echtzeitfähigkeit gestellt.
- Mehrere verbundene, eingebettete Systeme bilden ein verteiltes System und damit auch ein verteiltes Echtzeitsystem.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Anforderungen an ein Echtzeitsystem

Zentrale Anforderungen sind:

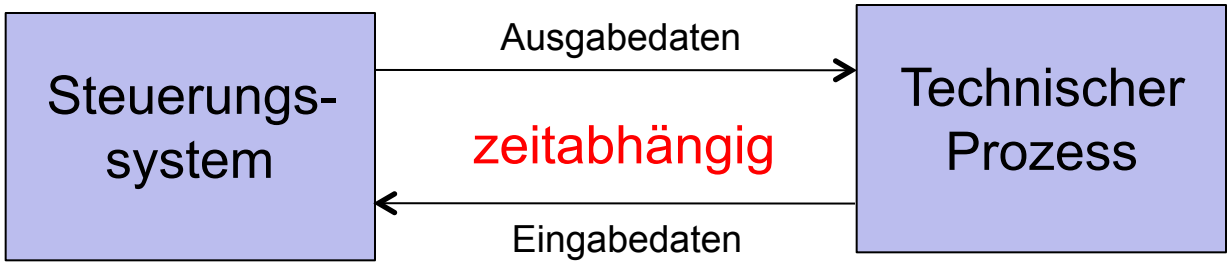
- Rechtzeitigkeit
- Gleichzeitigkeit

Weitere, wichtige Anforderungen sind außerdem:

- Sicherheit (keine Gefährdung von Menschen und Sachen)
- Verfügbarkeit / Zuverlässigkeit
- Robustheit / Adaptivität
- Unterbrechbarkeit
- Korrektheit / Genauigkeit
- Existenz von Datenschnittstellen zum externen Prozess

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Forderung nach Rechtzeitigkeit



- Eingabedaten müssen rechtzeitig abgerufen werden (Kriterien sind Dauer und Frequenz von Signalen und das Abtasttheorem).
- Ausgabedaten müssen rechtzeitig verfügbar sein (max. Reaktionszeit, Regelgüte und Stabilität).
- Zu unterscheiden sind:
  - Absolutzeitbedingungen (bezogen auf eine genaue, globale Uhrzeit T)
  - Relativzeitbedingungen (zeitlich relativ zu bevorstehenden Ereignissen $T_E \pm \delta t$)
  - Feste, äquidistante Zeitpunkte (genau zu diesem Zeitpunkt T bzw. k*T)
  - früheste Zeitpunkte (nicht bevor T oder $T_E - \delta t$)
  - späteste Zeitpunkte (nicht nach T oder $T_E + \delta t$)
  - Toleranzintervall (nicht bevor T_1 und nicht nach T_2 oder zwischen $T_E - \delta t$ und $T_E + \delta t$)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Forderung nach Gleichzeitigkeit

- Die Forderung nach Gleichzeitigkeit ergibt sich aus der Tatsache, dass Echtzeitsysteme auf Vorgänge in ihrer Umgebung reagieren müssen, die dort gleichzeitig ablaufen (unabhängige Zustandsgrößen).
- Lösbar dadurch, dass für jeden zeitlich parallelen Vorgang (Zustandsgröße) ein eigener Computer oder Steuerungssystem eingesetzt wird.
- Die große Rechenleistung heutiger Prozessoren erlaubt aber die quasi gleichzeitige Ausführung mehrerer unabhängiger Programme. Jedes Programm wird dazu beispielsweise von einem entsprechenden Echtzeitbetriebssystem immer nur für eine kurze, begrenzte Zeit ausgeführt, bevor es erneut unterbrochen wird, und das nächste Programm gestartet wird. Sind alle Programme an der Reihe gewesen, beginnt das Ganze zyklisch wieder von vorne (Round Robin). Solche Programme werden meist Task genannt.

Kapitel 2: Echtzeitsysteme

2.1 Grundbegriffe

2.2 Programmierung von Echtzeitsystemen

2.3 Organisation von Echtzeitsystemen

Task

Definition: Der Task ist der zeitliche Vorgang der Abarbeitung eines sequentiellen Programms auf einem Computer. Ein Task existiert nicht nur während der Ausführung der Befehle, sondern auch während geplanter oder erzwungener Wartezeiten. Er beginnt mit dem Eintrag in eine Liste des Echtzeitbetriebssystems und endet mit dem Löschen aus dieser Liste. Tasks können unterschiedliche Zustände einnehmen, wie „geplant“, „aktiviert“, „angehalten“, „bereit“, „ruhend“ u.ä.

Der Begriff „Task“ wird nicht einheitlich verwendet. Er wird auch als Oberbegriff für „Thread“ und „(Rechen-)Prozess“ oder synonym dazu verwendet.

Multitasking bezeichnet dann den Vorgang mehrere Tasks quasi gleichzeitig auf einem Computer auszuführen. Die Organisation der Taskausführung erfolgt durch den Scheduler, der Teil eines Echtzeitbetriebssystems ist. Die Ausführung des Task erfolgt zeit- und prioriätengesteuert. Können laufende Tasks durch das Betriebssystem unterbrochen werden, spricht man von präemptivem Multitasking.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Programmierung von Echtzeitsystemen

- Planung des zeitlichen Ablaufs der Tasks vor ihrer Ausführung (Zeitsteuerung)
  ➔ synchrone Programmierung
  - Der gesamte zeitliche Systemablauf wird zur Übersetzungszeit festgelegt.
  - Voraussetzung ist eine präzise, globale Uhr insbesondere bei verteilten Systemen oder eine feste zyklische Ausführung mit festen Codepfaden.
  - Je Task wird ein festes Zeitfenster reserviert. Dazu ist die Abschätzung der maximalen Laufzeit (WCET) notwendig.
- Organisation des zeitlichen Ablaufs während der Ausführung der Programme (Ereignissteuerung)
  ➔ asynchrone Programmierung
  - Ereignisse (Interrupts) triggern die zugehörigen Funktionen.
  - Der Aufruf der Tasks erfolgt dynamisch zur Laufzeit (Scheduling). Prioritäten regeln mögliche Konflikte und die Dringlichkeit.
  - Mögliche Unterbrechungen und Verschiebungen durch höher priorisierte Tasks müssen bei der garantierten Antwortzeit berücksichtigt werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Synchrone Programmierung

- Die Tasks werden nach einem festen Zeitplan gestartet. Man nennt dies auch Zeitsteuerung.
- Jedem Task wird eine genau definierte Zeitscheibe zugeteilt, in der er seine Berechnungen abschließen muss.
- Eine Überlastung des Echtzeitsystems kann dann ausgeschlossen werden, wenn die vorab bestimmten Laufzeiten die zugeteilten Zeiten sicher nicht überschreiten. Die Laufzeiten müssen dazu verlässlich abgeschätzt bzw. berechnet werden können.
- Die synchrone Programmierung erfordert einen hohen Planungsaufwand vor bzw. während der Implementierung. Jede spätere Änderung muss erneut hinsichtlich Ihrer Echtzeitfähigkeit überprüft werden, gegebenenfalls muss umgeplant werden.
- Wenn die benötigten Ausführungszeiten exakt bekannt sind kann der Prozessor zu nahezu 100% ausgelastet werden.
- Wenn die Tasks jeweils einzeln durch exakte Uhren getriggert werden, kann die zeitliche Unschärfe (Jitter), die sich beispielsweise aus nicht kontrollierbaren Einflüsse aus der Computerhardware oder dem Betriebssystem ergeben, minimiert werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Beispiel für synchrone Programmierung

Pseudocode für Assembler auf Intel-Mikrocontroller

```
 mov a, 10000 ; Grenzwert der Drehzahl
 mov b, 30    ; Grenzwert der Temperatur
 mov O, 1     ; Abschaltsignal

:loop         ; Markierung im Programmfluss (keine Instruktion)
 in t,PORT1   ; einlesen der aktuellen drehzahl-Werte
 in d,PORT2   ; einlesen der aktuellen temp-Werte

:drehcheck
 cmp t,a      ; prüfe die Drehzahl
 jg  tempcheck; wenn Grenzwert nicht erreicht, springe zu :tempcheck
 out PORT3,O  ; Grenzwert erreicht! setze Abschaltsignal

:tempcheck
 cmp d,b      ; prüfe die Temperatur
 jg  loop     ; wenn Grenzwert nicht erreicht, springe zu :loop
 out PORT3,O  ; Grenzwert erreicht! setze Abschaltsignal

 jmp loop     ; unbedingter Sprung zur Marke :loop (Endlosschleife)
```

zeitlich synchrone Erfassung der Sensorwerte

Quelle: Programmcode siehe Wikipedia “Echtzeitsysteme”

Endlos-Programmschleife (ohne Echtzeitbetriebssystem und Tasks) mit der auf das Überschreiten einer Temperatur t oder einer Drehzahl d durch Abschalten reagiert werden soll. Die maximale Reaktionszeit bis zur Ausgabe des Abschaltsignals, kann exakt berechnet werden, da die Ausführungszeit jedes einzelnen Befehls bekannt ist. Sie kann hier grob mit der zweifachen Zykluszeit der Endlosschleife nach oben abgeschätzt werden (WCET << 1ms). Die exakte Reaktionszeit ist im Einzelfall aber nicht vorab bestimmbar, da dies davon abhängt, zu welchem Zeitpunkt relativ zum zeitlichen Status der Ausführung der Endlosschleife, die Abbruchbedingung erstmals erfüllt wird.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Asynchrone Programmierung

- Ziel ist es, auf ein von außen kommendes Ereignis (z.B. Interrupt) schnellstmöglich zu reagieren. Dies nennt man auch Ereignissteuerung.
- Gegebenenfalls wird dazu ein gerade laufender Task unterbrochen.
- Auf ein einzelnes Ereignis kann so mit sehr geringem Zeitverlust reagiert werden.
- Wenn mehrere Ereignisse zeitgleich oder sehr zeitnah auftreten, kommt es zum Konflikt. Hierfür muss es eine Strategie geben, diesen Konflikt deterministisch aufzulösen (Steuerung über Prioritäten).
- Zur Abschätzung bzw. Berechnung der maximalen Reaktionszeit, muss bekannt sein, wie häufig die einzelnen Ereignisse auftreten können und wie viele andere Tasks jeweils höher priorisiert sind und wie lange diese zur vollständigen Ausführung brauchen.
- In einer harten Echtzeitumgebung muss sichergestellt werden, dass selbst der Task mit der niedrigsten Priorität immer noch rechtzeitig sein Ergebnis abliefern kann (Puffer). Die Rechenleistung kann daher bei der asynchronen Programmierung nie zu 100% genutzt werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Beispiel für asynchrone Programmierung

Ausschnitt aus C-Systemfunktion in Xenomai (CAN-Treiber)

```
/* Initialisierungsfunktion beim Treiberstart */
void rtcan_register(struct rtcan_device *dev) {
    /* RTDM (Real-Time Driver Model) ist das Echtzeit-Treibermodell von Xenomai.
     * dev: Gerätestruktur für Verwaltungsinformationen
     * irq_handle: IRQ-spezifische Verwaltungsinformationen
     * rtcan_interrupt_handler: Interrupt-Handler-Routine */
    rtdm_irq_request(&dev->irq_handle, irq_number, rtcan_interrupt_handler, irq_flags, name, dev);
}

/* Treiber beenden */
void rtcan_unregister(struct rtcan_device *dev) {
    rtdm_irq_free(&dev->irq_handle);
}

/* Interrupt-Handler, wird in rtcan_register() registriert (siehe oben). */
int rtcan_interrupt_handler(rtdm_irq_t *irq_handle) {
    /* Zuerst muss festgestellt werden, welche Art von Interrupt aufgetreten ist.
     * Dazu wird ein CAN-Hardware-spezifisches Interrupt-Register (IR) ausgelesen. */
    u8 irq_source = read_reg(dev, IR);

    /* Transmit-Interrupt? */
    if (irq_source & IR_TI)
        /* Sender signalisieren, dass der Transmit einer CAN-Nachricht abgeschlossen ist
                    * und der Programmfluss fortgeführt werden kann. */
        rtdm_sem_up(&dev->tx_sem);
    /* Receive Interrupt? */
    else if (irq_source & IR_RI) {
        /* CAN-Nachricht aus Hardware-Registern auslesen. Dies geschieht in einer
                    * separaten Funktion (hier nicht angegeben). */
        rtcan_rx_interrupt(dev, &can_frame_buffer);
                    /* überreiche die empfangene CAN-Nachricht an die interessierten Tasks. */
        rtcan_rcv(dev, &can_frame_buffer)
    }
    /* Bestätige der CAN-Hardware die Beendigung der Interrupt-Abarbeitung. */
    irq_ack(dev);

    return RTDM_IRQ_HANDLED;
}
```

A) siehe nächste Folie

Dieser Code ist unvollständig und soll nur das Grundprinzip verdeutlichen.

Grundprinzip: Zunächst muss für die spätere Reaktion auf einen möglichen Interrupt die Adresse des Interrupthandlers (Task) in einer Tabelle abgelegt werden. Kommt dann später der Interrupt, wird das laufende Programm gestoppt und der zugehörige Interrupthandler aus dieser Tabelle heraus gestartet. Der Interrupthandler kehrt nach Bearbeitung zum Betriebssystem zurück, das dafür sorgt, dass das ursprüngliche Programm fortgesetzt wird.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

XENOMAI-Anwendungs-programm

das CAN-Nachrichten in Echtzeit verarbeitet (Ausschnitt)

```
#define DREHZAHL_GRENZWERT      100
#define TEMP_GRENZWERT          30
#define ABSCHALTSIGNAL          1

#define DREHZAHL_CAN_ID         1
#define TEMP_CAN_ID             2
#define ABSCHALT_CAN_ID         3

struct can_frame recv_frame;
struct can_frame send_frame;

void can_task()
{
   int sock_recv = rt_dev_socket(PF_CAN, SOCK_RAW, CAN_RAW);
   int sock_send = rt_dev_socket(PF_CAN, SOCK_RAW, CAN_RAW);

   send_frame.can_id = ABSCHALT_CAN_ID;
   send_frame.can_dlc = 1;
   send_frame.data[0] = ABSCHALTSIGNAL;

   while (1) {
      rt_dev_recv(sock_recv, &recv_frame, sizeof(struct can_frame), 0);

      if ((recv_frame.can_id == DREHZAHL_CAN_ID) && (recv_frame.data[0] > DREHZAHL_GRENZWERT))
         rt_dev_send(sock_send, &send_frame, sizeof(struct can_frame), 0);

      if ((recv_frame.can_id == TEMP_CAN_ID) && (recv_frame.data[0] > TEMP_GRENZWERT))
         rt_dev_send(sock_send, &send_frame, sizeof(struct can_frame), 0);
   }
}
```

A) siehe vorhergehende Folie

Deterministische Ausführungszeiten

Um Ausführungszeiten im Betrieb garantieren zu können (WCET worst case execution time), dürfen bestimmte Techniken bei Echtzeit-Computersystemen nicht oder nur sehr eingeschränkt verwendet werden, wie z.B.:

- Dynamische Speicherallokation und automatische Garbage Collection.
- Rekursive Funktionsaufrufe, wenn Rekursionstiefe nicht bekannt.
- Warten auf externe Eingaben oder Bestätigungen in Programmschleifen (z.B. Prozesssignale, Benutzereingaben, Speicherzugriffe o.ä.)
- Zufallsgesteuerte Medienzugriffe (z.B. Netzwerk-Kommunikation mit zufälliger Wartezeit nach Kollision)
- Virtueller Speicher (unkalkulierbare Wartezeiten beim Auslagern und Nachladen von Speicherblöcken)
- Betriebssystemaufrufe (nur wenn max. Ausführungsdauer garantiert)
- u.ä.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kapitel 2: Echtzeitsysteme

2.1 Grundbegriffe

2.2 Programmierung von Echtzeitsystemen

2.3 Organisation von Echtzeitsystemen

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Echtzeitbetriebssystem

Nach DIN 44300 versteht man unter einem Betriebssystem die Programme eines Computers, die zusammen mit den Eigenschaften der Computerhardware die Grundlage der möglichen Betriebsarten des Computersystems bilden und insbesondere die Abwicklung von Programmen steuern und überwachen.

Ein Echtzeitbetriebsystem (engl. Real-Time Operating System) muss zusätzlich die Bedingungen der Gleichzeitigkeit und Rechtzeitigkeit erfüllen. Es muss deshalb innerhalb bestimmter, garantierter Zeiten auf Signale von außen reagieren und deshalb gewährleisten, dass alle Tasks rechtzeitig ausgeführt werden. Zu einem Echtzeitbetriebsystem gehören deshalb zusätzlich die folgenden Komponenten:

- Ein-/Ausgabsteuerung mit Zugriff auf die Schnittstellen zum technischen Prozess über Sensoren und Aktoren (Treiber)
- Interruptsteuerung
- Taskverwaltung (➔Task-Scheduling)

Beispiele: QNX, VxWorks, Linux/Xenomai, RT-Linux, OSEK-OS, T-Kernel, FreeRTOS uvm.

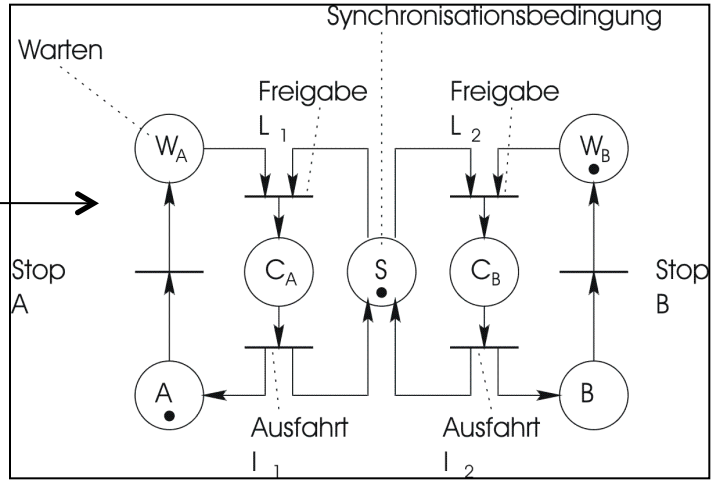
Task-Scheduling

- Ein Programm, das die Zuteilung der Betriebsmittel (CPU, Speicher, E/A usw.) vornimmt, durch das also die Zuteilungsstrategie realisiert wird, wird Scheduler genannt.
- Neben den vom Anwender definierten Tasks muss eine Vielzahl anderer, sogenannter Systemtasks verwaltet werden (Statistiken, Speicherverwaltung, Benutzerinterface usw.).
- Ziel des Schedulers ist es, den effizienten Parallelbetrieb möglichst vieler Betriebsmittel zu erreichen. Dazu gehört z.B., dass wenn ein Task auf eine Eingabe wartet, die CPU sofort für einen anderen Task freigegeben wird. Die gleichzeitige Anforderung eines Betriebsmittels durch mehrere Tasks wird über Warteschlagen verwaltet. Es kommen verschiedene Strategien zur Verwaltung der Warteschlangen zum Einsatz, wie z.B.: EDF (earliest deadline first), FIFO (first in first out), SJN (shortest job next), Round Robin usw.
- Ein besonderer Task ist der Idle-Task (auch Null-Prozess), der immer dann aktiviert wird, wenn kein anderer Task eingeplant und ausführungsbereit ist.

Task-Synchronisierung

- Bei der zeitlich parallelen Ausführung von Tasks kann es vorkommen, dass ein Task einen anderen Task „logisch“ überholt oder dass zwei Tasks gleichzeitig auf das selbe Betriebsmittel zugreifen wollen. Dies muss synchronisiert werden. Man unterscheidet:
  ➔ Logische Synchronisation: Erzwingen von definierten zeitlichen Folgen von Aktionen, d.h. die Synchronisation zwischen den Abläufen im technischen Prozess und den Tasks.
  ➔ Betriebsmittelorientierte Synchronisation: Einhalten von Bedingungen bzgl. der Verwendung gemeinsam benutzter Betriebsmittel (Hardware oder Software).

- Zur Synchronisation solcher Prozessabläufe gibt es verschiedene Verfahren, wie:
  - Semaphore,
  - kritische Regionen,
  - Rendez-vous-Konzept,
  - usw.



Echtzeitkommunikation

In einem verteilten Echtzeitsystem ist es erforderlich, dass auch der Datenaustausch zwischen Tasks oder der Datenaustausch mit dem technischen Prozess in zeitlich deterministischer Weise erfolgt.

Für diesen Zweck wurden Netzwerke mit entsprechenden Protokollen entwickelt, die einen Datenaustausch unter Echtzeitbedingungen ermöglichen. Diese Netzwerke werden Feldbusse genannt. Je nach Nähe zum technischen Prozess unterscheidet man: Sensor/Aktorbus, Feldbus oder Prozessbus.

Kollisionen beim Medienzugriff müssen wegen der sich daraus ergebenden, unkalkulierbaren Verzögerungen verhindert werden. Dies wird erreicht durch Verfahren, wie zyklische Ringbusse, Master/Slave mit Zeitscheiben, Nachrichten mit Prioritäten u.ä.

Beispiele: ASI, Interbus, CAN, Profibus, RTnet, EtherCAT usw.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Echtzeit-Middleware

Unter Middleware versteht man eine Vermittlungssoftware, die den Informationsaustausch zwischen Anwendungen so unterstützt, dass die gesamte Komplexität der Kommunikation zwischen den Programmen für den Programmierer weitgehend verborgen bleibt. Die Programme können auf der gleichen Computerhardware oder auch auf über ein Netzwerkwerk verbundenen aber getrennten Computern ausgeführt werden. Welches Netzwerk mit welchem Protokoll genutzt wird, ist für den Programmierer unsichtbar. Die Kommunikationstechnologie kann entsprechend leicht ausgetauscht werden, ohne dass dies Auswirkungen auf die Anwendungsprogramme hat. Für die Kommunikation kommen abstrakte Konzepte zum Einsatz, wie z.B. Client/Server, Publish/Subscribe oder Mailboxen.

Eine Middleware, die darüber hinaus zeitliche Garantien beim Informationsaustausch gibt, kann als Echtzeit-Middleware bezeichnet werden. Dazu gehört auch die Synchronisation von Uhren im Netzwerk sowie die Verknüpfung absoluter Zeitstempel mit Informationen, wie beispielsweise Messwerten. In einem verteilten Systeme muss die Kommunikation entsprechend über Feldbusse erfolgen.

Beispiele: RT-CORBA, OSA+, OROCOS RTT, SensorNet, RACK usw.

Zusammenfassung

Es wurde eine Einführung in Echtzeitsysteme gegeben. Dabei wurden die folgenden Begriffe überblicksartig vorgestellt und erläutert:

- Echtzeitsystem
- Eingebettetes System
- Harte und weiche Echtzeit
- Rechtzeitigkeit und Gleichzeitigkeit
- Synchrone und asynchrone Programmierung
- Task bzw. (Rechen-)Prozess
- Echtzeitbetriebssystem
- Task-Scheduling
- Task-Synchronisation
- Echtzeitkommunikation
- Echtzeit-Middleware

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kapitel 3: Steuerungssysteme

3.1 Grundbegriffe

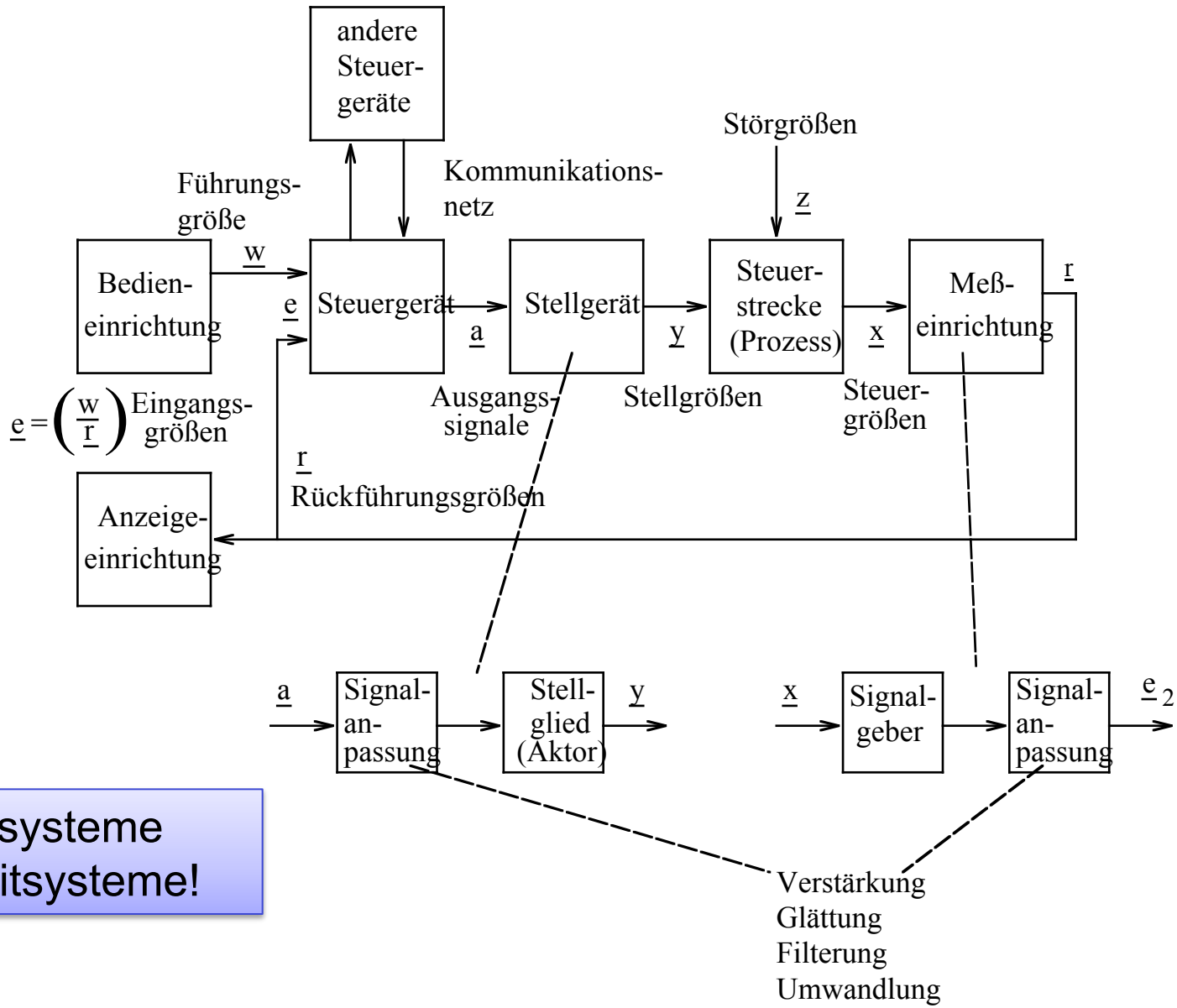
3.2 Formen von Steuergeräten

3.3 Systematische Entwicklung von Steuerungen

3.4 Ein erstes Beispiel: Steuerung Werktor

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

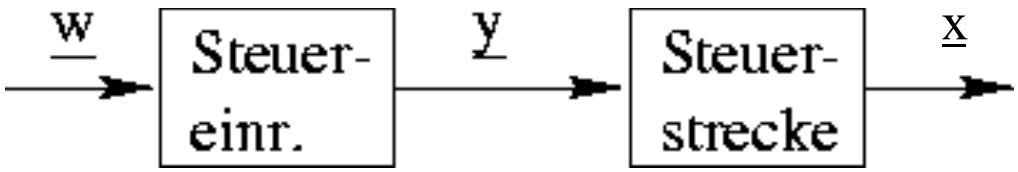
Elemente eines Steuerungssystems



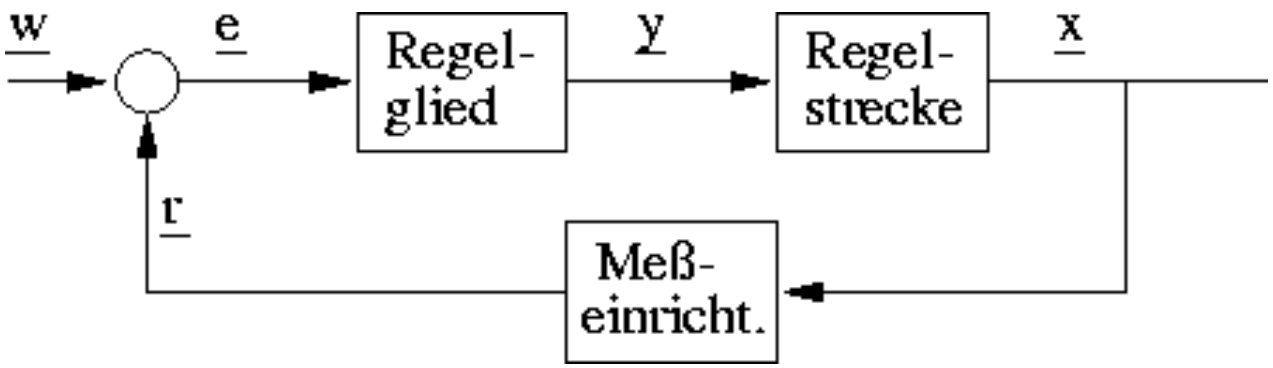
Steuerungssysteme sind Echtzeitsysteme!

Steuerung und Regelung

Offener und geschlossener Wirkungsablauf nach DIN19221 und 19226.



Die Rückführung der Steuergrößen zum Steuergerät, wie sie z.B. für Ablaufsteuerungen notwendig ist, bezeichnet man als geschlossene Steuerung.



Von Regelung spricht man nur, wenn der Wert der kontrollierten Größe fortlaufend mit dem Wert der zugeordneten Führungsgröße verglichen wird, mit dem Ziel, Abweichungen sofort auszugleichen.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Klassifizierung von Steuerungen

Unterscheidung nach Art der Signalverarbeitung:

- synchrone Steuerung (Taktsignal $\underline{a}(kt)$, $\underline{e}(kt)$)
- asynchrone Steuerung (ereignisdiskretes System)
- Verknüpfungssteuerung (kombinatorisches System $\underline{a}=f(\underline{e})$)
- Ablaufsteuerung (sequentielle Automaten $\underline{a}=f(\underline{e},\underline{s})$)

Vergleich synchrone und asynchrone Programmierung bei Echtzeitsystemen.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Unterscheidung nach Informationsdarstellung

analoge Steuerung:

stetig wirkende Funktionsglieder
Beispiel: Steuerung der Gitterspannung einer Verstärkerröhre durch den Anodenstrom.

binäre Steuerung:

keine Zahlen oder Codes, Verwendung von Verknüpfungs-, Zeit- und Speichergliedern
Beispiel: Elektromechanische Steuerung mit Schalter, Relais, Lampen usw.

digitale Steuerung:

Zahlen und Codes, Verwendung von Zählern, Rechenwerken, Registern usw.
Beispiel: Steuerung mit digitalisierten Signalwerten, Zugriff über Geräteadressen, Benutzerdialoge usw.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Unterscheidung der zu steuernden Prozesse

Fließprozesse:

zeit- und ortsabhängige, kontinuierliche Prozessgrößen wie Temperatur, Druck, usw.
→ Differentialgleichungen
Beispiele: Heizungsanlagen, Energieversorgung in Kraftwerken

Folgeprozesse:

diskrete Prozessgrößen und Folge von Ereignissen, wie Schalterstellung
→ Boolesche Funktionen, Automaten
Beispiele: Fahrstuhl, Fertigungsmaschine, Waschmaschine

Stückprozesse:

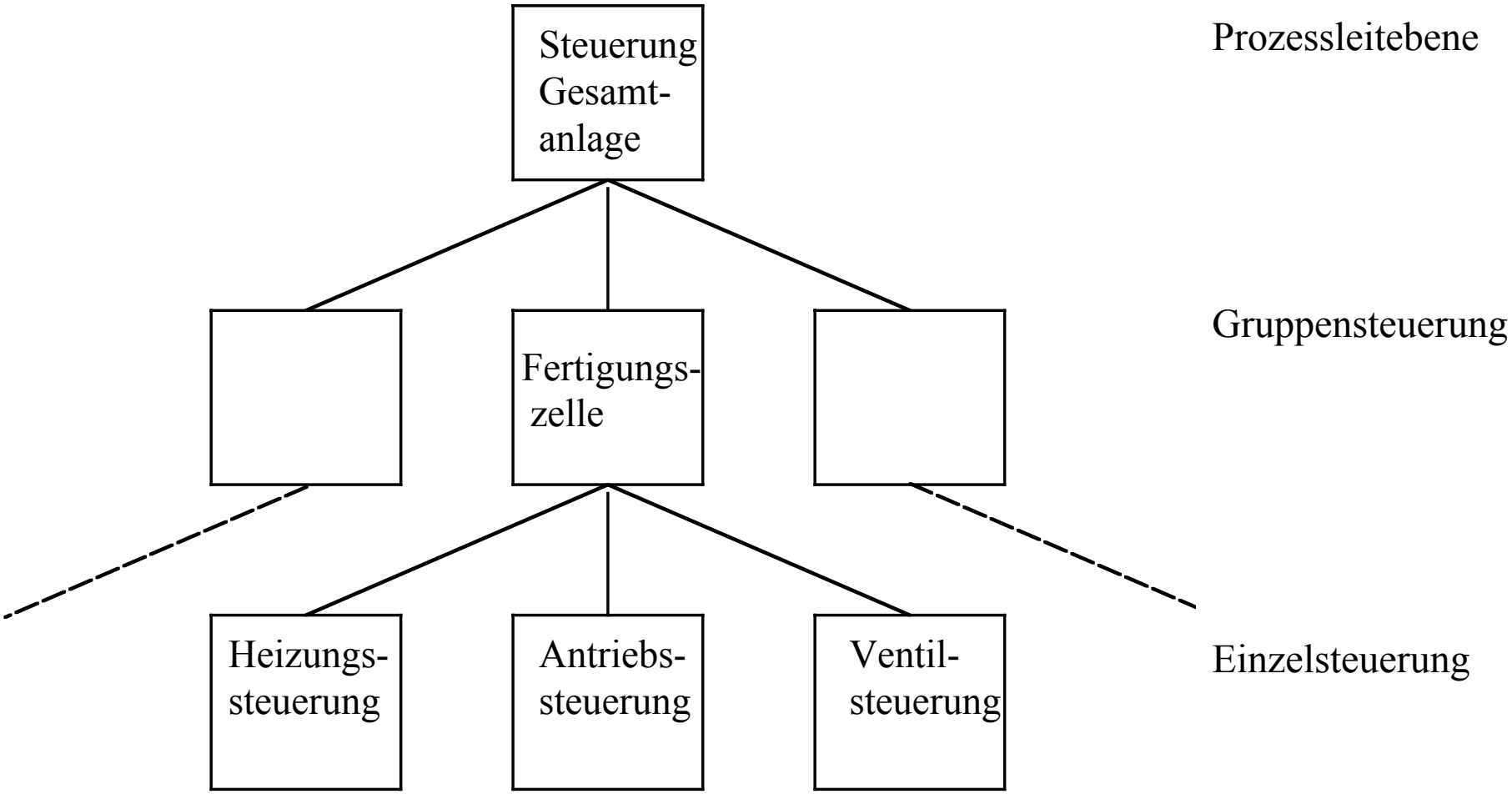
einzelne, identifizierbare Objekte (Stücke), Veränderungen im Ort und Zustand
→ Stellen-/Transitionsnetze
Beispiele: Lagerhaltung, Paketverteilanlagen, Montage

Chargenprozesse:

kontinuierliche Vorgänge, die in diskreten Zeitabständen ablaufen
→ Automaten und Differentialgleichungen
Beispiele: Brauerei, Gießerei, Produktion von Fertiggerichten, Kläranlagen

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Hierarchisch organisierte Steuerung



Kapitel 3: Steuerungssysteme

3.1 Grundbegriffe

3.2 Formen von Steuergeräten

3.3 Systematische Entwicklung von Steuerungen

3.4 Ein erstes Beispiel: Steuerung Werktor

Steuergeräte sind eine Geräteform von Echtzeitsystemen.

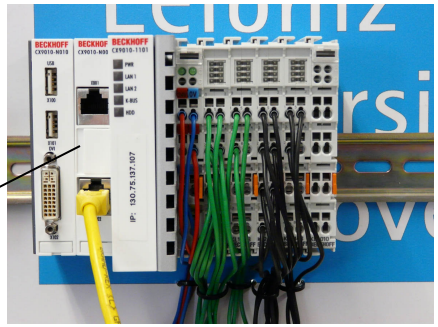
SPS-Automatisierungsgerätes

Modularer Aufbau:
- Netzgerät (Stromversorgung)
- Verarbeitungseinheit (CPU)
- Digital I/O
- Analog I/O
- Feldbus-Schnittstelle
- Programmiergerät (heute: PC)

CPU mit:
Steuer- und Rechenwerk, Akku, Programmspeicher, Merkern, Zeitgliedern, Zählern, Prozessabbild und Bussystem

Programmierung nach IEC 61131:
Programmierung in verschiedenen Sprachen möglich: AWL, ST, KOP, AS, FBS

Standardisierte Betriebsarten:
- Neustart bei Initialisierung
- Warmstart bzw. Wiederanlauf bei Störung
- Automatikbetrieb
- Einzelschritt bzw. Tippbetrieb



SPS mit Windows XP Embedded & Echtzeit Laufzeitumgebung

Eingebettete Steuergeräte

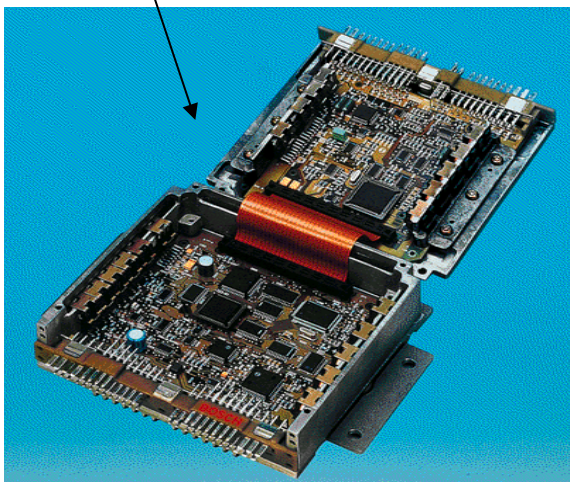
Kleine Automatisierungsgeräte, die in das zu steuernde System baulich integriert sind. Wesentlicher Kern sind leistungsfähige Mikrocontroller (z.B. Intel 8051 oder Siemens 80167) mit denen die Steuerungs-funktionen softwaremäßig in einer höheren Programmiersprache, meist C oder seltener Assembler, realisiert werden. Es bestehen große Anforderungen an die Zuverlässigkeit.

Anwendungsbeispiele:

- KFZ-Steuergeräte für Airbag, ABS oder Motormanagement
  Die Steuerungsfunktionen werden zunehmend auf viele Controller verteilt.
- Waschmaschinen, DVD-Player, Hausheizung o.ä.

Experimental-roboter mit eingebettetem Steuergerät

Wegen ihrer sehr großen Stückzahlen sind Komponenten eingebetteter Steuergeräte sehr preiswert bei gleichzeitig hoher Zuverlässigkeit. Sie gewinnen daher im industriellen Bereich zunehmend an Bedeutung und verdrängen traditionelle Technologien.



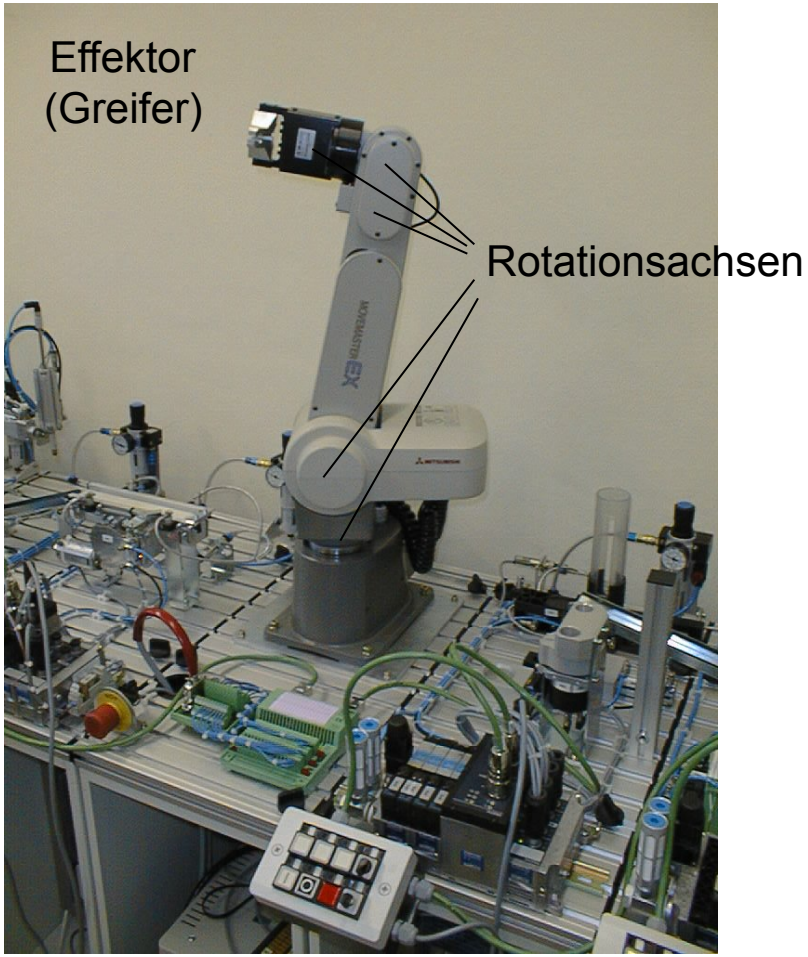
Steuerung von Industrierobotern

für Montage, Handhabung oder Werkstückbearbeitung

Roboterprogrammierung:

1. Teach-In Verfahren: Interaktive Programmierung von Raumpositionen, Bewegungskurven und Effektoraktionen mit einem mobilen Handprogrammiergerät (Teach-Box).
2. Textuelle Programmierung: Verwendung einer für Robotersteuerung entwickelten höheren Programmiersprache (z.B. MRL oder IRL)
3. Kombinierte Verfahren: Programmierung von Positionen und Teilbewegungen mit der Teach-Box, Anfahren und Ausführung aus einer höheren Programmiersprache heraus.

Man unterscheidet:
PTP Point to Point: **Punktsteuerung** - Die Bewegungsbahn wird unkontrolliert und ohne Funktionszusammenhang durchfahren.
CP Continuos Point: **Bahnsteuerung** - kontrollierte Bewegung entlang einer definierten Bahnkurve.
MP Multipoint: **Vielpunktsteuerung** - Annäherung des Bahnverhaltens durch mehrere Zwischenpunkte.



Mitsubishi Roboter (5 Freiheitsgrade)

Eigenschaften von Industrierobotern

1. Beweglichkeit

Rotationsachsen oder Translationsachsen (Zahl der Freiheitsgrade bestimmt die Beweglichkeit). Die Achsen bestimmen den Arbeitsraum und das zweckmäßige Koordinatensystem :

- kartesisch (z.B. Portalroboter)
- zylindrisch (z.B. Kranroboter)
- kugelförmig (z.B. bei drei Rotationsachsen)

Weitere Einflussfaktoren:
- Traglast
- Auflösung Messsystem
- Gelenkspiel
- Materialkrümmung

2. Effektor

Die Einsatzmöglichkeiten werden durch den Effektor (Greifer, Werkzeug) bestimmt. Die Effektoren werden um aufgaben-spezifische Sensoren ergänzt, wie z.B. optische oder taktile Sensoren zur Objekterkennung bzw. zur Bestimmung der am Greifer wirkenden Kräfte.

3. Wiederholgenauigkeit

Zu unterscheiden von Positioniergenauigkeit

Präzision der Positionierung bei Wiederholung der Bewegung (z.B. 0,5 mm bei 1000 Wiederholungen).

4. Geschwindigkeit

Schnelles Anlaufen und Bremsen, z.B. bewirkt durch Gleichstromservomotoren oder Schrittmotoren usw.

5. Tragkraft, Beschleunigung, Antriebe, Positioniergenauigkeit usw.

Maschinensteuerung

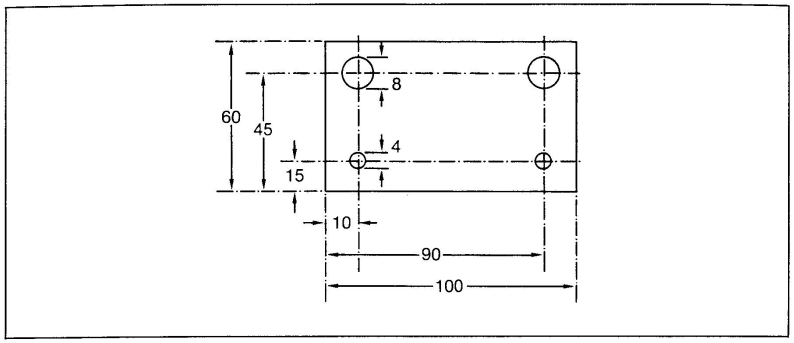
NC - Numerical Control (Lochstreifen)
CNC - Computer Numerical Control (Computer)
DNC – Direct Numerical Control (Netzwerk)

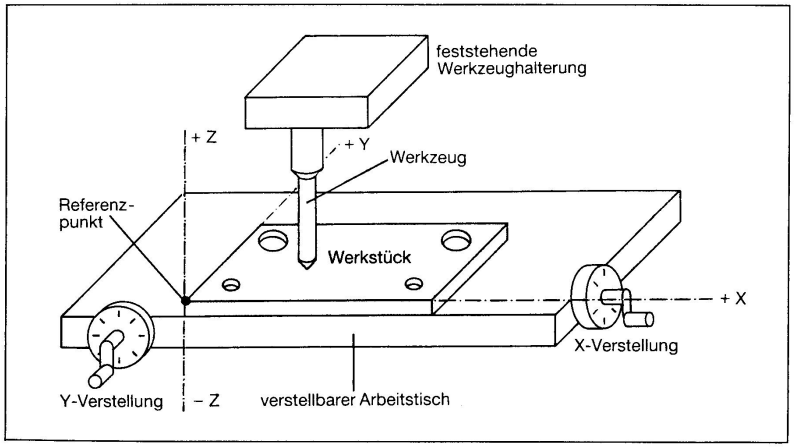
Steuerung von Werkzeugmaschinen zur spanabhebenden Umformung von Rohlingen (Fräsen, Drehen, Bohren), auch in der Holzbearbeitung:

Der NC Programmierer muss die prinzipielle Funktionsweise der Werkstückbearbeitung und das Zusammenspiel zwischen Steuerung und Maschine kennen.

Programmierung in speziellen NC-Programmiersprachen (EXAPT, ADAPT u.a.).

Beispiele:
- Geometriebefehle: P1 = POINT / 15.7, 25, 13
- Bewegungsbefehle: GOTO / P1
- Hilfsbefehle: COOLNT (*Kühlmittel einschalten*)

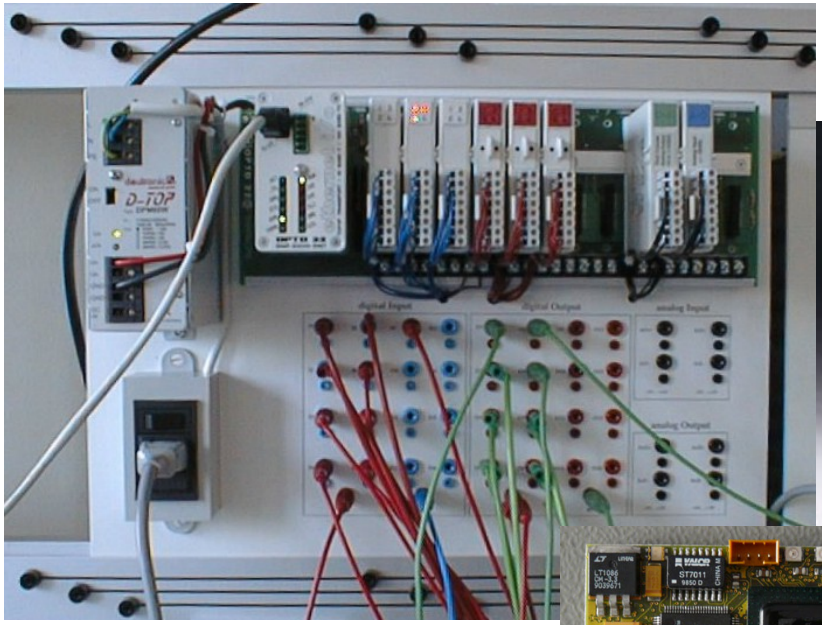




Satznr.	Wegbe-dingung	Weginformation X	Y	Z	Schaltinformationen Vorschub	Drehzahl	Werkzeug	Zusatz
N 001	G 00	X 10	Y 15	Z 2	F 1000	S 55	T 01	M 09
N 002				Z 0				
N 003		X 90		Z 2				
N 004				Z 0				
N 005			Y 45	Z 2	F 500	S 30	T 02	
N 006				Z 0				
N 007		X 10		Z 2				
N 008				Z 0				
N 009		X 0	Y 0					M 30

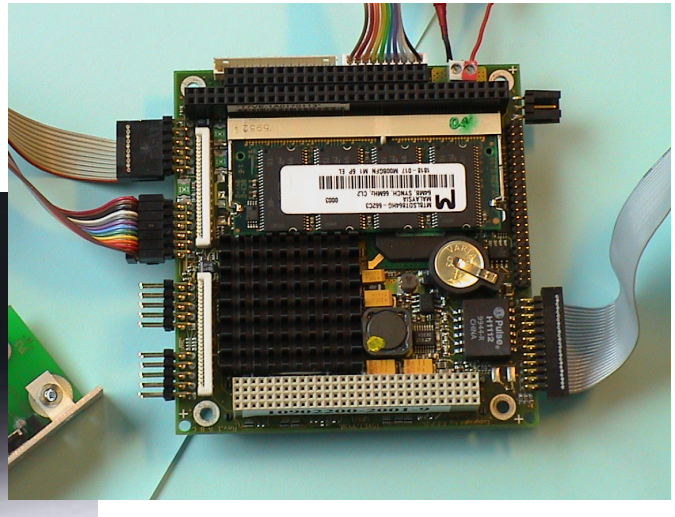
© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Remote Controller



Opto22

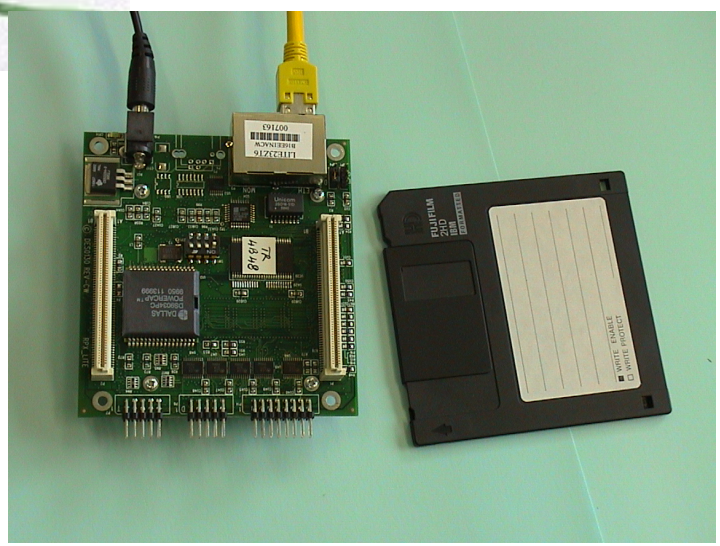
RFC



Pentium mit Linux/RTAI

Zugriff auf Steuerungen aus der Ferne für Wartung, Programmierung, Bedienung usw.

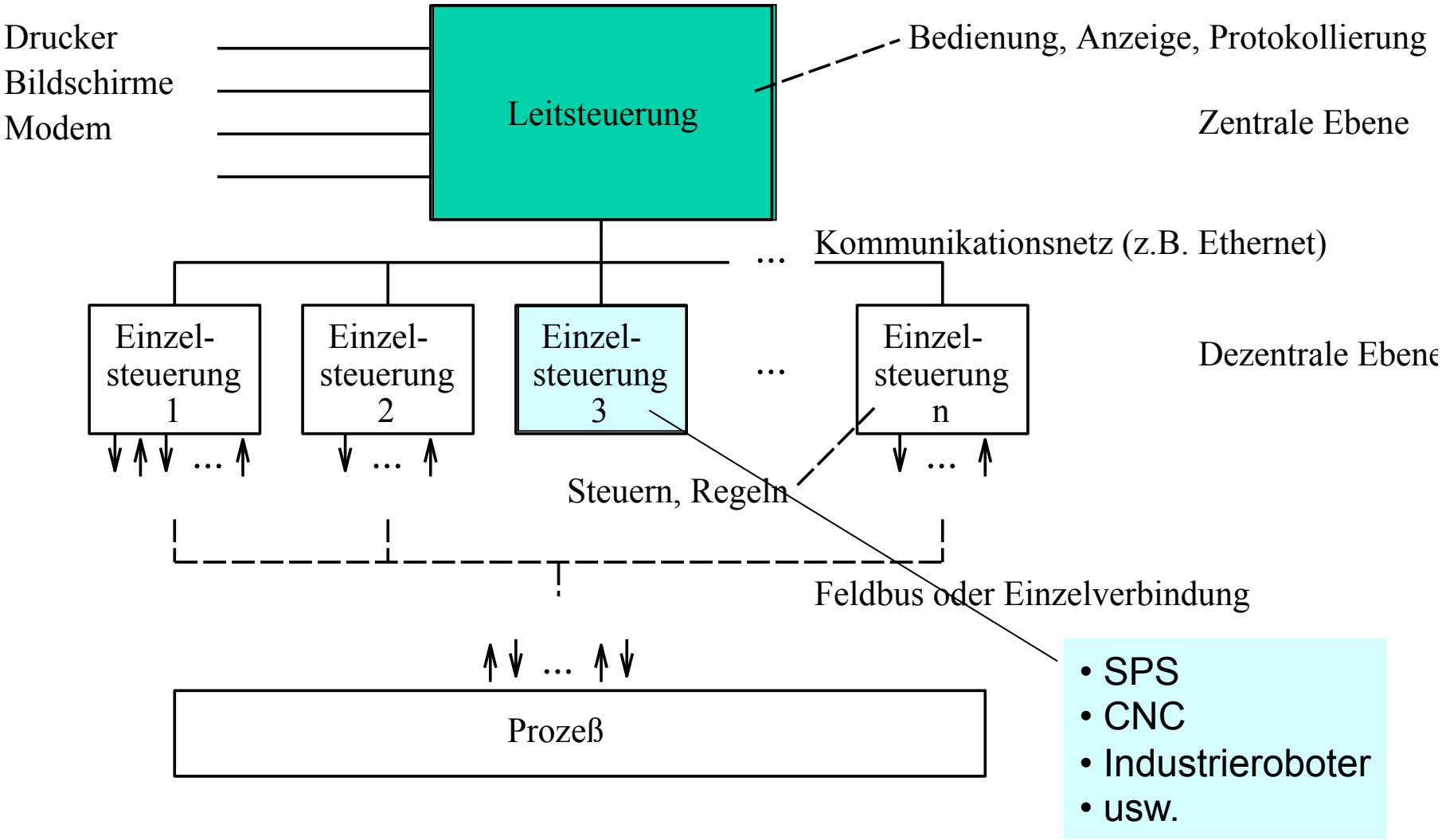
DIMM-PC



Power-PC mit Jbed

Prozessleitsteuerung

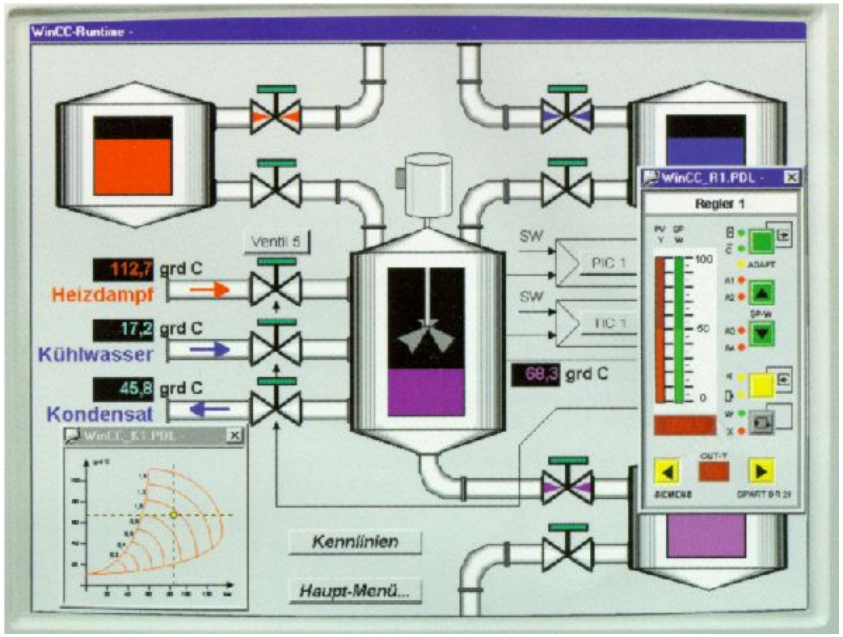
(auch: Prozessleitsystem)



Aufgaben eines Prozessleitsystems (PLS)

Funktionen in Echtzeit

1. Prozessvisualisierung auf Bildschirmen
2. Datenauswertung und Archivierung
3. Systemdiagnose
4. Bedienung
5. Systemdokumentation
   (Hilfetexte, Reparaturanweisungen ...)
6. Alarmbehandlung



Prozessvisualisierung eines verfahrens-technischen Prozesses

Alternative Steuergerätetechnologien

Verbindungsprogrammierte Steuerung nach DIN 19237: Steuerung deren Programm durch die Art der Funktionsglieder und durch deren Verbindungen vorgegeben wird.

- Mechanische Steuerung
- Relaissteuerung
- Pneumatische Steuerung
- Hydraulische Steuerung

Heute werden diese Gerätetechnologien häufig mit Speicherprogrammierten Steuerungen oder Mikrocontrollern kombiniert, so dass hybride Steuerungen (Echtzeitsysteme) entstehen.

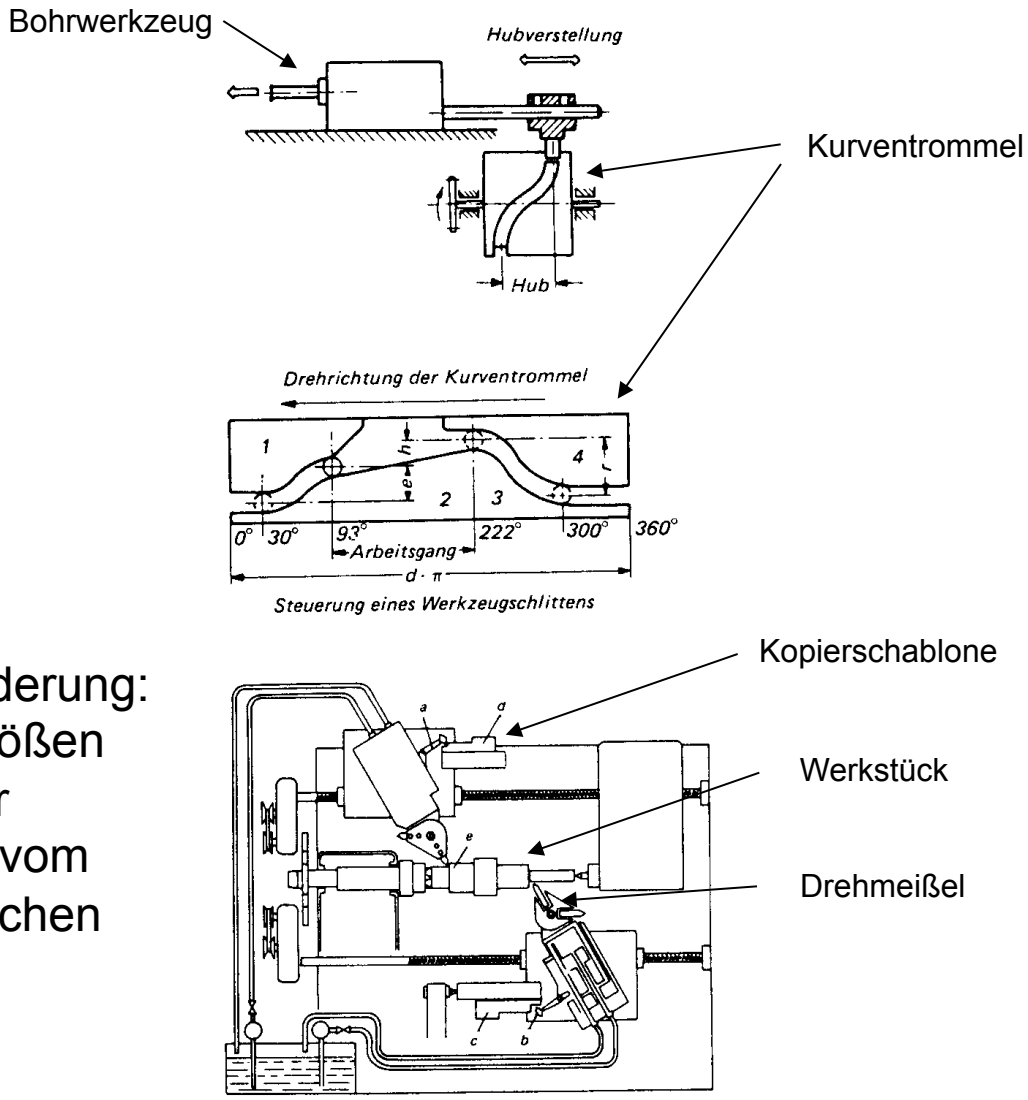
Verbindungsprogrammierte Steuerungen arbeiten nahezu verzögerungsfrei und sind damit hinsichtlich Ihrer Reaktionsverzögerung Echtzeitsysteme.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Mechanische Steuerungen

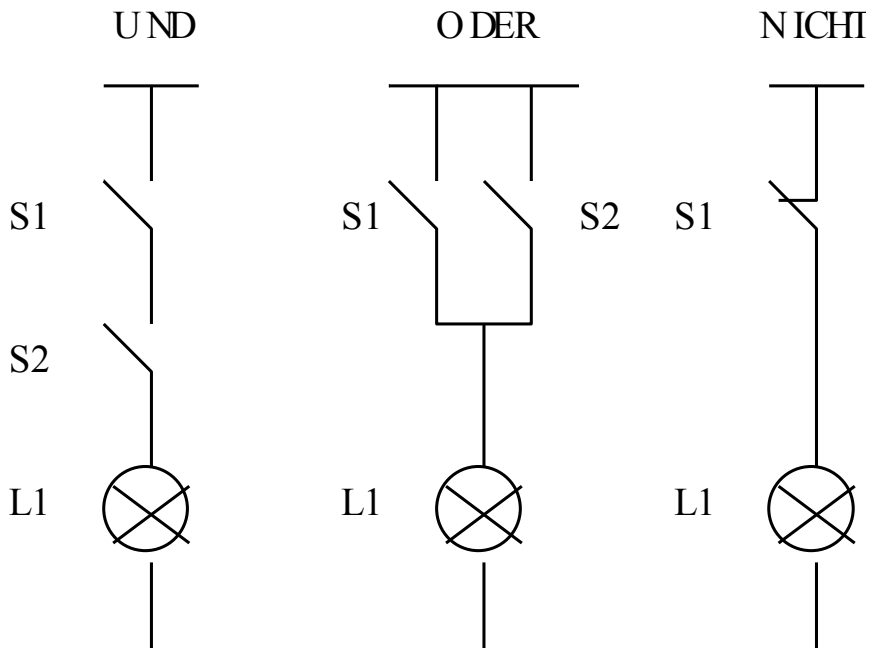
Beispiele:
- Nockenwelle (Taktsteuerung von Verbrennungsmotoren)
- Kurvenscheiben (Steuerung von Vorschüben an Werkzeugschlitten)
- Kopierschablone (Steuerung von Drehwerkzeugen)

Mechanischer Programmspeicher: Steuersignale werden gebildet von Kräften und Wegen bzw. deren Änderung:
➔ Wegplansteuerung: Führungsgrößen werden von einem Programmgeber geliefert, dessen Ausgangsgrößen vom zurückgelegten Weg eines beweglichen Teils der gesteuerten Anordnung abhängen.



Elektromechanische Steuerung

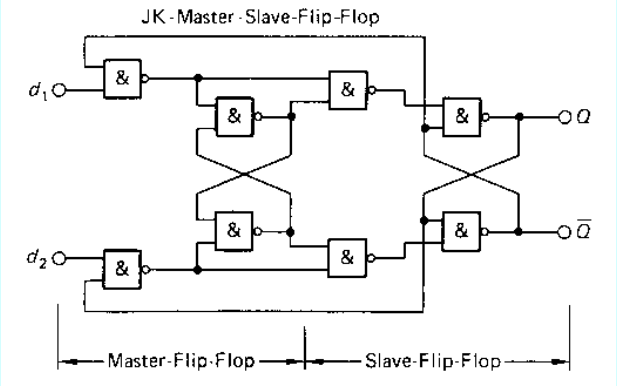
Kontaktbehaftete Geräteelemente (Schütze und Relais), die Steuersignale werden durch Spannungs- bzw. Stromänderungen im offenen bzw. im geschlossenen Stromkreis dargestellt



Stromlaufpläne logischer Grundfunktionen

Weiterentwicklung in Form **„Elektronischer Steuerungen“:**

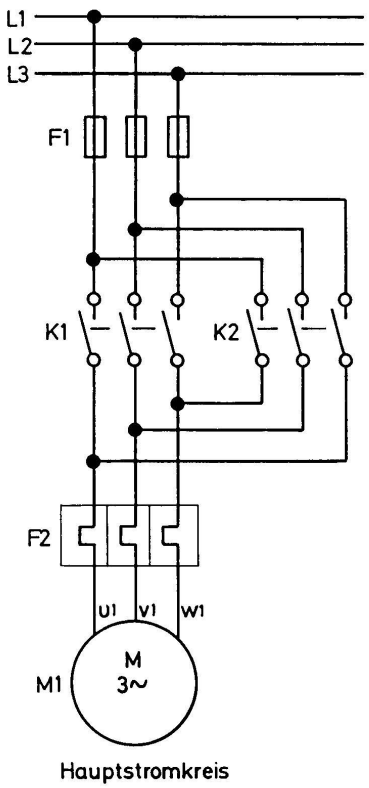
Kontaktloser Aufbau mit elektronischen Schaltkreisen, sogenannte Transistorsteuerungen. Strom- und Spannungsänderungen zwischen definierten Pegeln. Realisierungsformen: TTL-Schaltkreise, ASIC, PAL

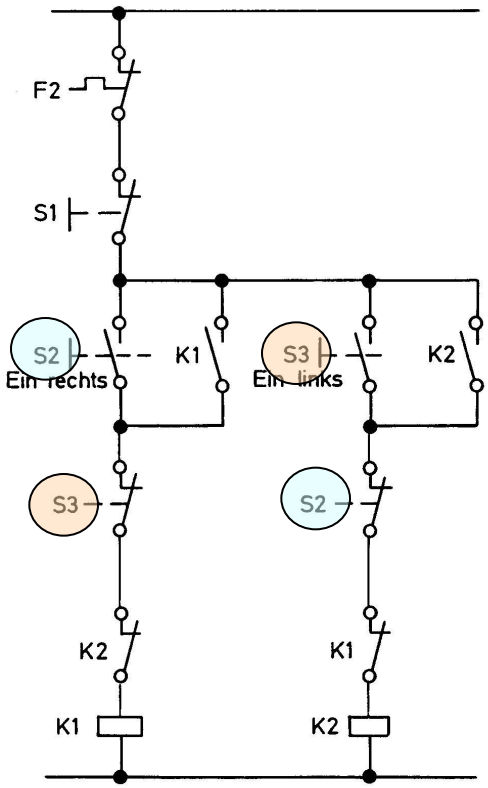


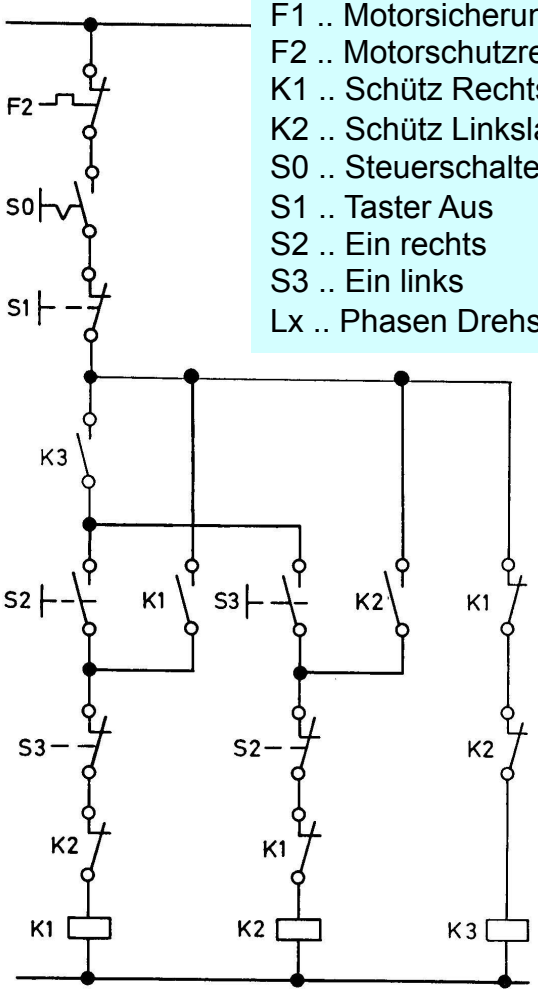
Beispiel: Drehstrom-Asynchronmaschine

Drehrichtungsumkehr

Quelle: Böhm, W.: Elektrische Steuerungen. Vogel 1991







F1 .. Motorsicherung
F2 .. Motorschutzrelais
K1 .. Schütz Rechtslauf
K2 .. Schütz Linkslauf
S0 .. Steuerschalter Ein
S1 .. Taster Aus
S2 .. Ein rechts
S3 .. Ein links
Lx .. Phasen Drehstrom

- Übersichtsschaltplan und Stromlaufplan (Haupt- und Steuerstromkreis)
- andere Phasenreihenfolge: 1-2-3 und 3-2-1
- Steuerstromkreis mit gegenseitiger Verriegelung. Kurzschlussgefahr, wenn: Einschalt- < Ausschaltverzögerung
- Steuerstromkreis mit Umschaltverzögerung: kein Kurzschluss durch Addition von Einschalt- und Ausschaltverzögerung.

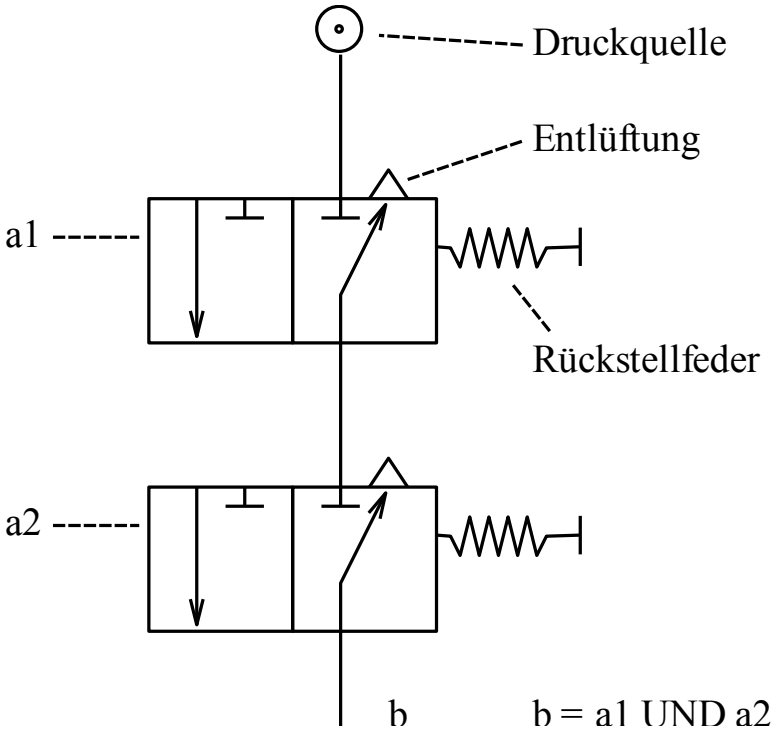
© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Pneumatische und hydraulische Steuerung

Gerätetechnik mit Ventilen und Zylindern. Das Medium der Signalübertragung ist Druckluft oder Hydrauliköl.

Eine hochentwickelte pneumatische Steuerungstechnik ist in den Fluidik-Steuerungen entwickelt.

Aufgrund des möglichen, höheren Drucks (> 100 bar) kann mit Hydraulikzylindern (Linearantrieb) eine sehr große Stellkraft direkt gesteuert werden (Hydromotoren).



Pneumatische UND-Stufe

Kapitel 3: Steuerungssysteme

3.1 Grundbegriffe

3.2 Formen von Steuergeräten

3.3 Systematische Entwicklung von Steuerungen

3.4 Ein erstes Beispiel: Steuerung Werktor

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Entwicklung von Steuerungen

Zentrales Ziel bei der Realisierung industrieller Steuerungen ist der zufriedene Auftraggeber bzw. Betreiber der automatisierten Anlage.

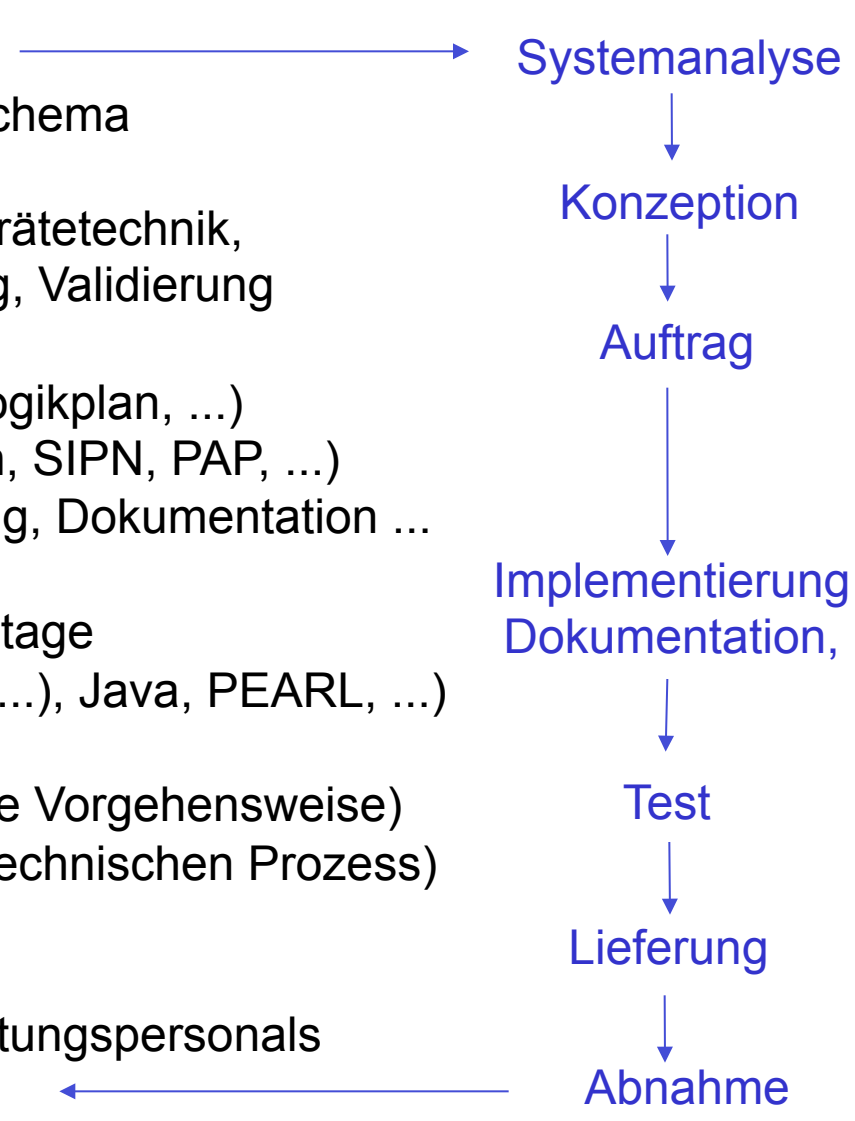
Dies wird erreicht durch:

- Erfüllung der funktionalen, qualitativen und zeitlichen Anforderung. Diese müssen präzise, d.h. am besten formal, in Spezifikation beschrieben werden.
- Kostengünstige Realisierung.
- Geringe Folgekosten (Zuverlässigkeit und Wartbarkeit).
- Sicherheit (keine Gefahr für Leben und Sache).
- Vollständige und korrekte Dokumentation für Bedienung, Pflege, Fehleranalyse und Erweiterung.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Projektphasen

1. Aufgabenstellung
   (a) Lastenheft (Wunsch, Idee)
   verbale Beschreibung, Technologieschema
   (b) Pflichtenheft (Lösung, Vertrag)
   Grobstruktur, Zuordnungstabelle, Gerätetechnik, Betriebsarten, Prototypenentwicklung, Validierung
2. Entwurf
   (a) Hardware-Entwurf (Schaltplan, Logikplan, ...)
   (b) Software-Entwurf (Zustandsgraph, SIPN, PAP, ...)
   (c) Entwurf: Bedienung, Visualisierung, Dokumentation ...
3. Realisierung
   (a) Bestellung, Verdrahtung und Montage
   (b) Programmierung (SPS (ST, AWL,...), Java, PEARL, ...)
4. Test
   (a) Testfallentwicklung (systematische Vorgehensweise)
   (b) Testdurchführung (mit und ohne technischen Prozess)
5. Inbetriebnahme
   (a) Aufbau vor Ort, inkl. Test
   (b) Einweisung des Bedien- und Wartungspersonals
6. Nutzung (längste und teuerste Phase)



© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kapitel 3: Steuerungssysteme

3.1 Grundbegriffe

3.2 Formen von Steuergeräten

3.3 Systematische Entwicklung von Steuerungen

3.4 Ein erstes Beispiel: Steuerung Werktor

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

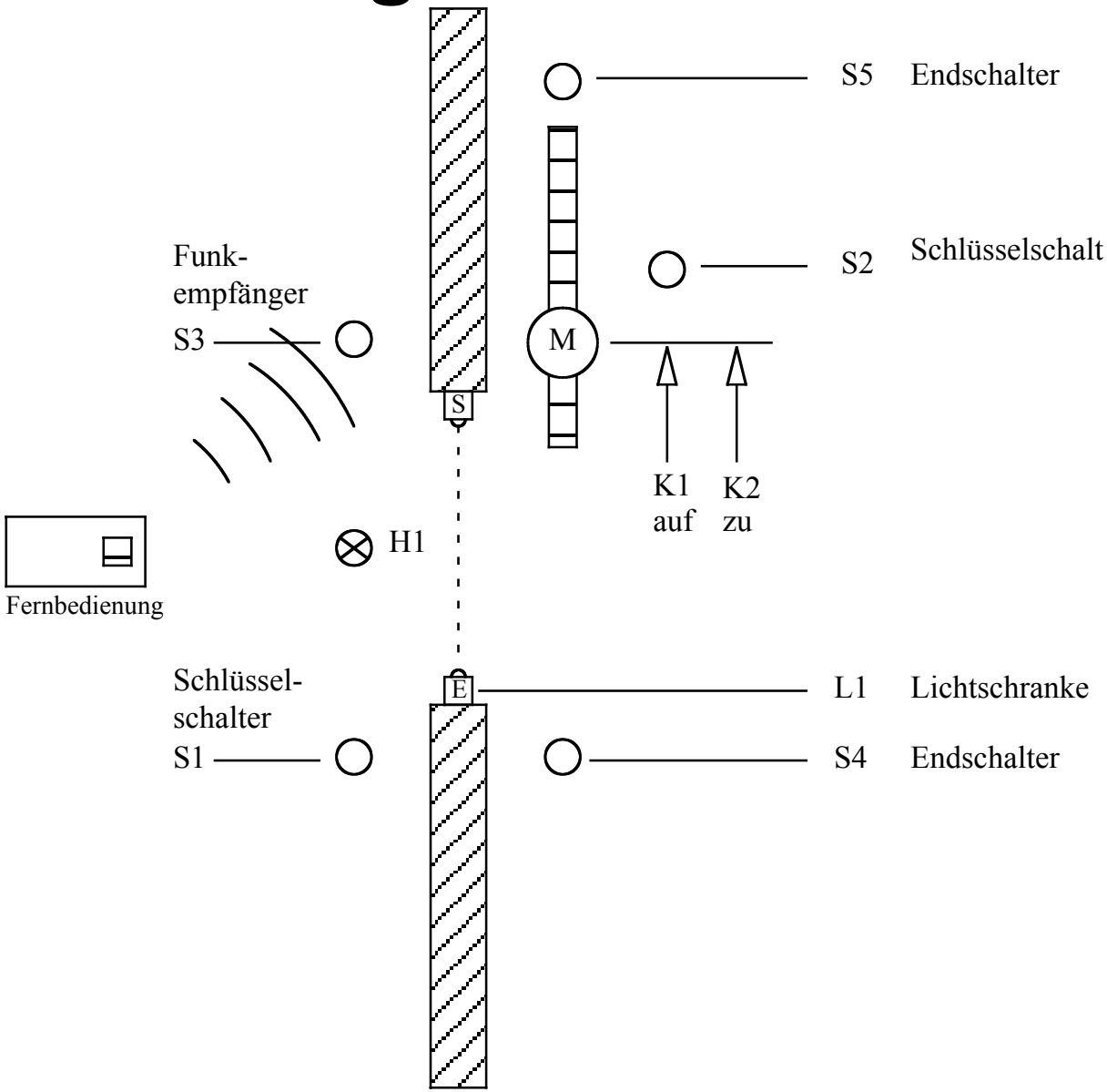
Aufgabenstellung Werktor

Technische Anforderungen::

1. Ein Werktor soll mit einem Elektromotor auf und zu gesteuert werden können.
2. Das Werktor kann von außen und innen mit einem Schlüsselschalter geöffnet werden. Zusätzlich steht eine Fernbedienung zur Verfügung, mit der das Tor von außen und innen geöffnet werden kann.
3. Das vollständig geöffnete Tor soll nach Ablauf der Wartezeit von 20 Sekunden automatisch geschlossen werden.
4. Ein seitlich angebrachtes rotes Blinklicht signalisiert nach beiden Seiten das Öffnen bzw. Schließen des Rollentors.
5. Aus Sicherheitsgründen soll eine Lichtschranke montiert werden. Ist die Lichtschranke unterbrochen, darf das Tor nicht geschlossen werden.
6. Der Elektromotor wird über zwei Schütze für den Links- bzw. Rechtslauf angesteuert.
7. Zwei Endschalter signalisieren „Tor offen“ und „Tor geschlossen“.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Technologieschema Werktor



© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Zuordnungstabelle Werktor

Signal	E/A	Kennz.	logische Zuordnung
Schlüsselschalter außen	E	S1	betätigt S1=1
Schlüsselschalter innen	E	S2	betätigt S2=1
Funkempfänger	E	S3	Code empfangen S3=1
Endschalter Tor zu	E	S4	Tor zu S4=1 *)
Endschalter Tor auf	E	S5	Tor auf S5=1 *)
Lichtschranke	E	L1	unterbrochen L1=1 *)
Blinklicht	A	H1	Licht an H1=1
Motorschütz Tor auf	A	K1	Schütz angezogen K1=1
Motorschütz Tor zu	A	K2	Schütz angezogen K2=1

*) Die logische Zuordnung wird üblicherweise so gewählt, dass im Störungsfall in Verbindung mit der Logik des Steuerprogramms keine gefährlichen Zustände entstehen (z.B. durch Leitungsbruch)! Im nachfolgenden Programmbeispiel wurde dies nicht vollständig umgesetzt.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Verknüpfungssteuerung

Eine Verknüpfungssteuerung ist eine Steuerung, die den Signalzuständen der Eingangssignale bestimmte Signalzustände der Ausgangssignale im Sinne boolescher Funktionen zuordnet (DIN 19226).

Wertetafel bzw. Funktionstabelle „Öffnen des Werktores“

Eingänge				Ausgänge	
S1	S2	S3	S5	K1	H1
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	1
:	:	:	:	:	:
1	1	1	1	0	1

Unklare Aufgabenstellung; Präzisierung in formaler Spezifikation erforderlich!

Boolesche Funktion bzw. Schaltausdruck:

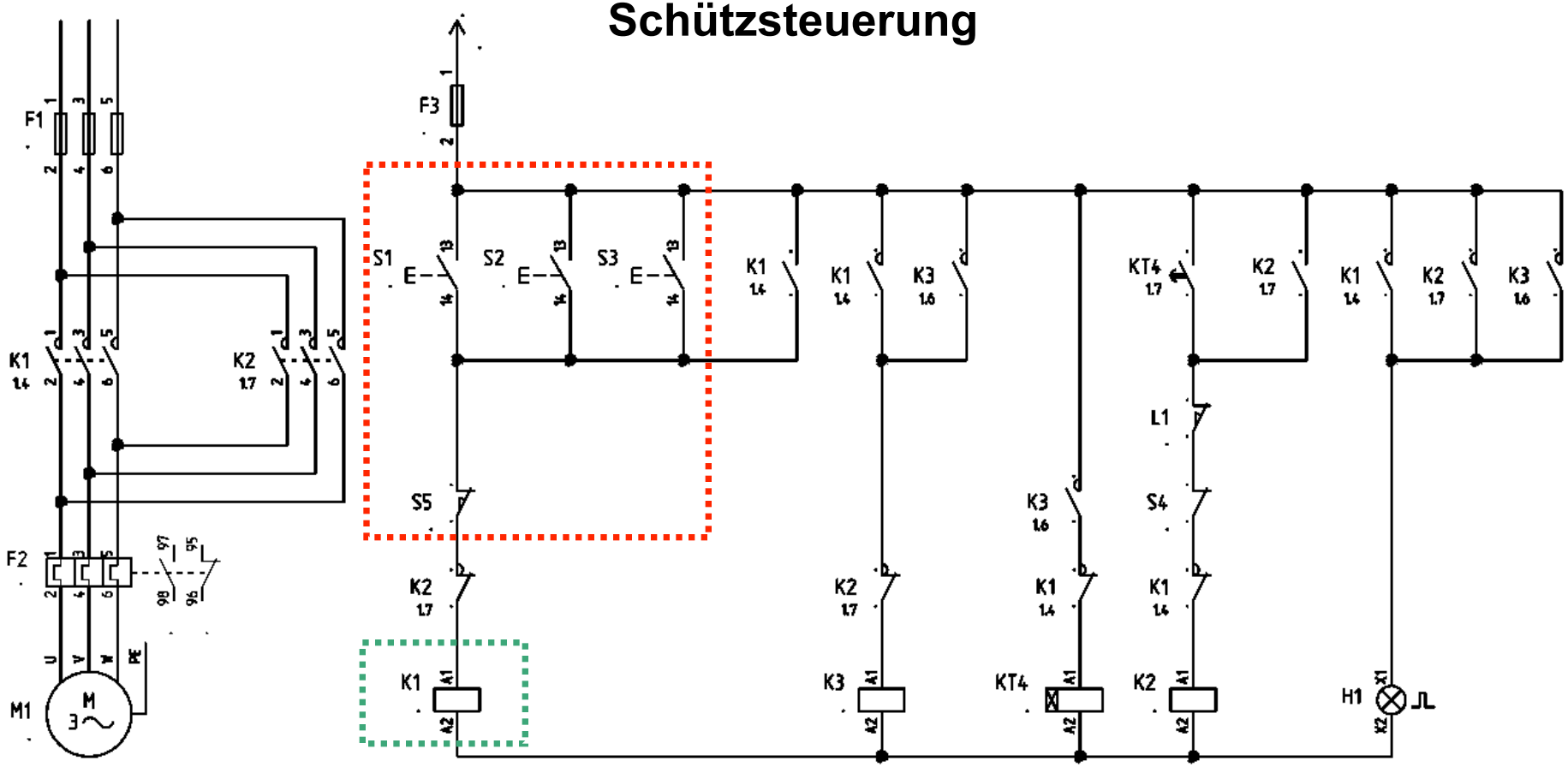
$$K1 = (S1 \vee S2 \vee S3) \wedge \bar{S}5$$
$$H1 = ((S1 \vee S2 \vee S3) \wedge \bar{S}5) \vee (\ldots)$$

Öffnen

Schließen

Stromlaufplan Werktor

Schützsteuerung



$$K1 = (S1 \vee S2 \vee S3) \wedge \overline{S5}$$

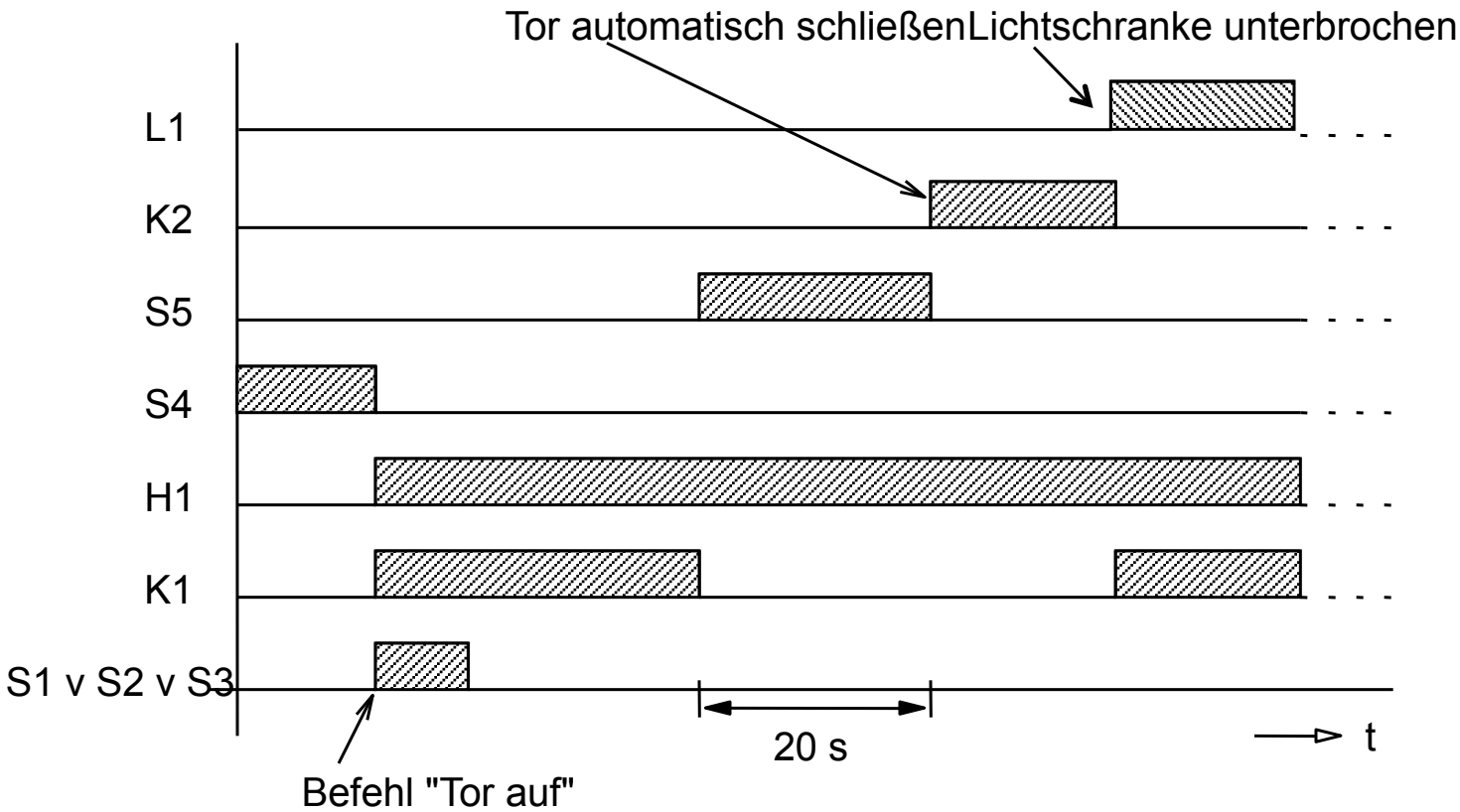
SPS-Programm (ST)

```
PROGRAM OB1

VAR (* Eingangssignale *)
        Tor_Auf AT %IX0.0 : BOOL;
        Tor_Zu AT %IX0.1 : BOOL;
        Schl_Innen AT %IX0.2 : BOOL;
        Schl_Aussen AT %IX0.3 : BOOL;
        Funk AT %IX0.4 : BOOL;
        Lichtschranke AT %IX0.5 : BOOL;
END_VAR
VAR (* Ausgangssignale *)
        Blinklicht AT %QX1.0 : BOOL;
        Motor_Auf AT %QX1.1 : BOOL;
        Motor_Zu AT %QX1.2 : BOOL;
END_VAR
VAR (* Speicherglieder: RS-Flip-Flop *)
        Oeffnen : RS;
        Schliessen : RS;
END_VAR
VAR (* Zeitglied: Einschaltverzögerung *)
        Wartezeit : TON;
END_VAR

(* Beginn Anweisungsteil *)

(* Das Tor öffnet, wenn einer der beiden Schlüssel-
schalter oder die Funkfernbedienung betätigt wird.
Außerdem wird das weder vollständig geschlossene
noch vollständig geöffnete Tor wieder geöffnet,
wenn die Lichtschranke unterbrochen wurde. Der Motor
schaltet ab, sobald der Endschalter (Tor_Auf) an-
spricht. Die drei Tastersignale müssen für die Be-
arbeitung gespeichert werden. *)

Oeffnen ( SET := ((Schl_Innen OR Schl_Aussen OR Funk)
                   AND NOT Tor_Auf)
                  OR
                  (Lichtschranke AND NOT (Tor_Auf
                                          OR Tor_Zu)),
          RESET1 := Tor_Auf);
Motor_Auf := Oeffnen.Q1;

(* Das Tor schließt automatisch nach Ablauf einer Warte-
zeit von 20 Sekunden. Die Wartezeit läuft immer ab dem
Augenblick, in dem das Tor vollständig geöffnet war.
Sobald die Lichtschranke unterbrochen wird, wird das
Tor wieder vollständig geöffnet. *)

Wartezeit(IN := Tor_Auf, PT := t#20s);
Schliessen( SET := NOT Motor_Auf
                   AND NOT (Tor_Zu OR Lichtschranke)
                   AND Wartezeit.Q,
            RESET1 := Tor_Zu OR Lichtschranke);
Motor_Zu := Schliessen.Q1;

(* Das Blinklicht signalisiert Gefahr und muss daher
immer eingeschaltet sein, wenn das Tor sich bewegt und
wenn das Tor geöffnet ist, da es sich jederzeit wieder in
Bewegung setzen kann. *)

Blinklicht := Oeffnen.Q1 OR Schliessen.Q1 OR Tor_Auf;

END_PROGRAM
```

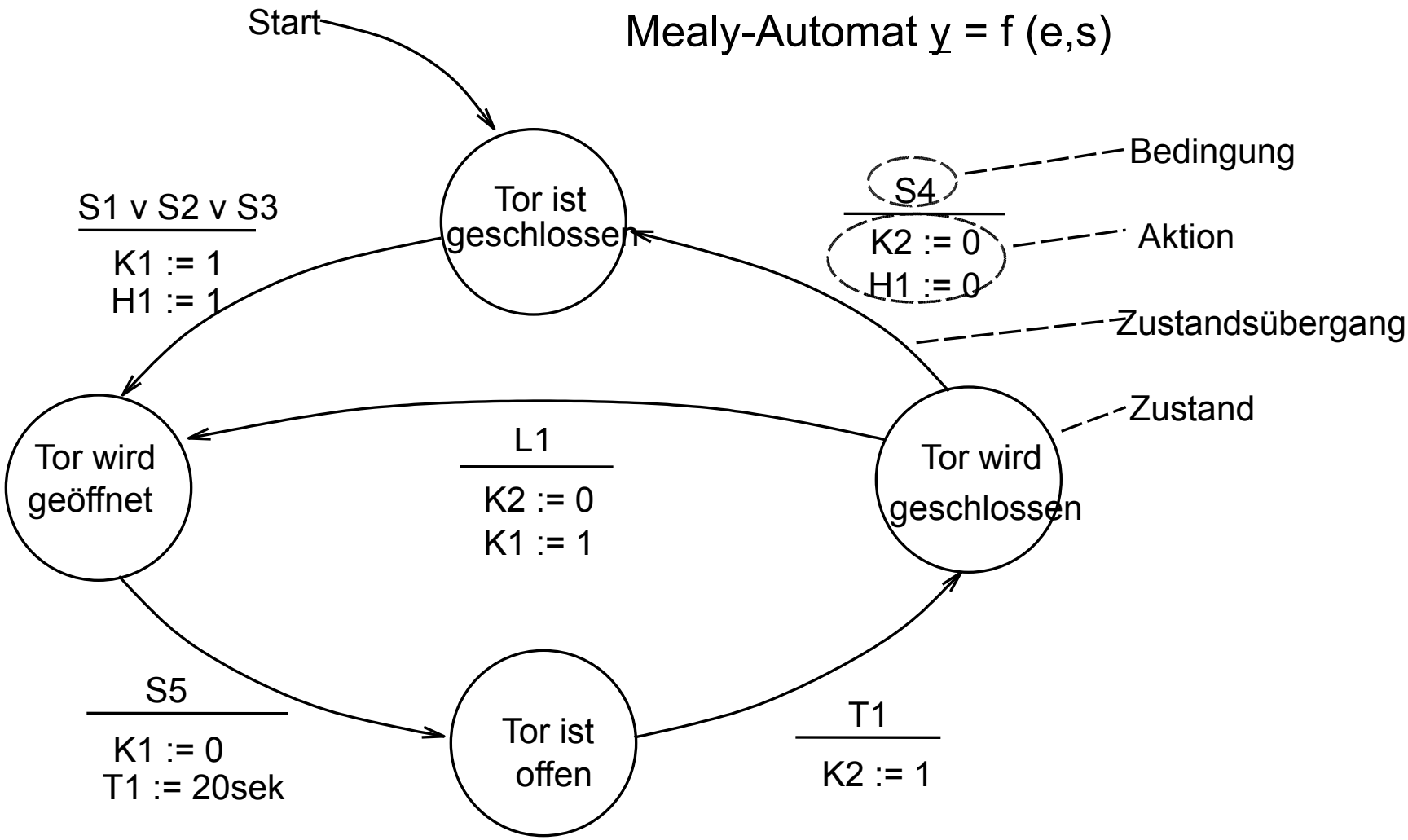
Ablaufsteuerung

Eine Steuerung mit zwangsläufig schrittweisem Ablauf, bei der das Weiterschalten von einem Schritt auf den programmgemäß folgenden abhängig von Weiterschaltbedingungen erfolgt. (DIN 19226).

Beschreibung und Analyse mit dem Zeitdiagramm:



Torsteuerung als Zustandsgraph



Aufgabe

Behauptung: Die vorgestellte Lösung des SPS-Programms für das Werktor ist nicht fehlerfrei!

Hinweis: Der Zustandsgraph enthält diesen Fehler nicht.

Dazu müssen Sie zunächst klären, was „korrekte Funktion“ in diesem Zusammenhang überhaupt bedeutet. Überlegen Sie sich dann, wie Sie auf systematische Art und Weise die korrekte Funktion überprüfen können und führen Sie anschließend die Prüfung und gegebenenfalls Korrektur durch.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Zusammenfassung

Es wurde ein Überblick über die Steuerungstechnik gegeben und dabei die folgenden Fachbegriffe eingeführt:

- Regelung vs. Steuerung, Steuerungssystem, Steuergerät usw.
- Technologieschema, Zuordnungstabelle, Wertetafel bzw. Funktionstabelle, Boolesche Funktion, Schaltausdruck, Zeitdiagramm, Zustandsgraph
- Verknüpfungs- und Ablaufsteuerung
- Analoge, binäre und digitale Steuerung, synchrone und asynchrone Steuerung
- Fließ-, Folge-, Stück- und Chargenprozesse
- Projektphasen: Lasten- und Pflichtenheft, Test usw.

Es wurden die wesentlichen Familien von Steuerungssystemen vorgestellt und kurz das Funktionsprinzip erläutert:

- Mechanische, elektronische und elektromechanische Steuerungen, pneumatische und hydraulische Steuerungen, Speicherprogrammierte Steuerungen (SPS), Roboter- und Maschinensteuerungen (CNC), eingebettete Steuerungen, Remote Controller
- Leitsteuerung bzw. Prozessleitsysteme

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kapitel 4: Speicherprogrammierte Steuerungen

4.1 Aufbau und Funktion einer SPS

4.2 Kontaktplan (KOP)

4.3 Funktionsbausteinsprache (FBS)

4.4 Strukturierter Text (ST)

4.5 Anweisungsliste (AWL)

4.6 Ablaufsprache (AS)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Speicherprogrammierte Steuerung (SPS)

SPS = Spezialrechner für Steuerungszwecke:

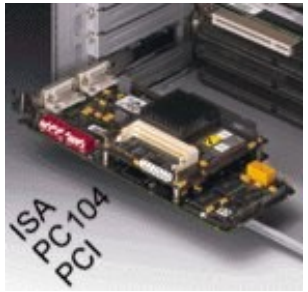
- Steuerprogramm im Programmspeicher als Folge von Anweisungen realisiert
- Ein- und Ausgänge für binäre und analoge Signale
- angepasster Befehlsvorrat: logische Verknüpfungen, Speicherung von Binärwerten, Zeitbildung, Zählen, externe Signale, Verarbeitung von Digitalwerten, Lesen und Schreiben externer Signale
- zyklischer Programmablauf
- synchroner Datenabgleich über Prozessabbild
- robuster Aufbau für Schaltschrankmontage (Hutschiene)
- Aufbau aus Einzelkomponenten (CPU, Netzteil, E/A-Baugruppen, Feldbusanschluss, Programmieranschluss)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Alternativen zur SPS-Programmierung

Industriestandard	Internationale Norm
Siemens S7 (seit 1994) mit Step7 (alt S5 – 1979 bis teilw.2015)	IEC 61131-3 bzw. DIN EN 61131 Teil 3
(AWL, KOP, FUP, Graph 5)	(AWL, KOP, ST, FBS, AS)
sehr hoher Marktanteil	Betreuung und Zertifizierung durch PLC OPEN: http://www.plcopen.org

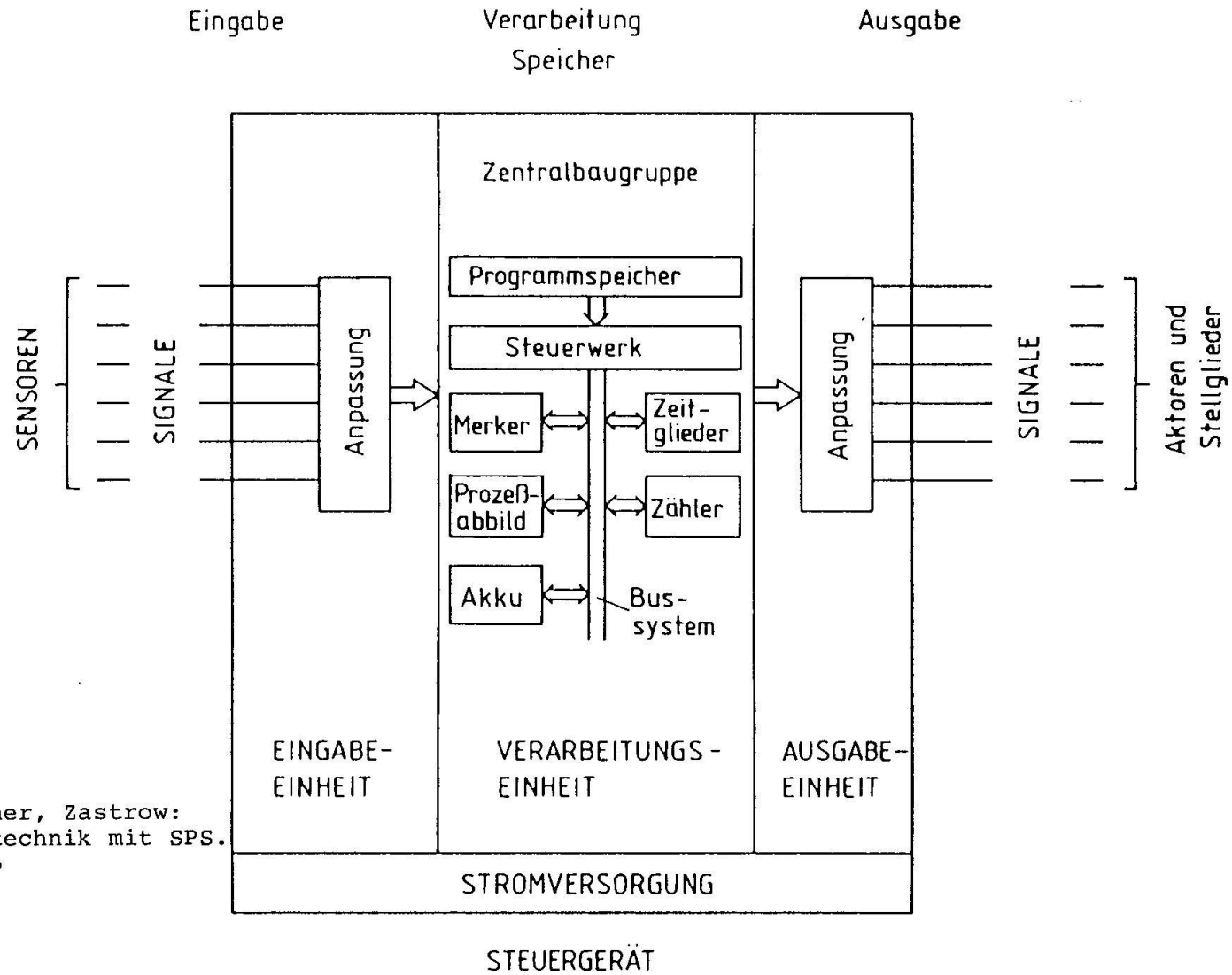
PLC = Programmable Logic Controler



SPS als PC-Erweiterungskarte

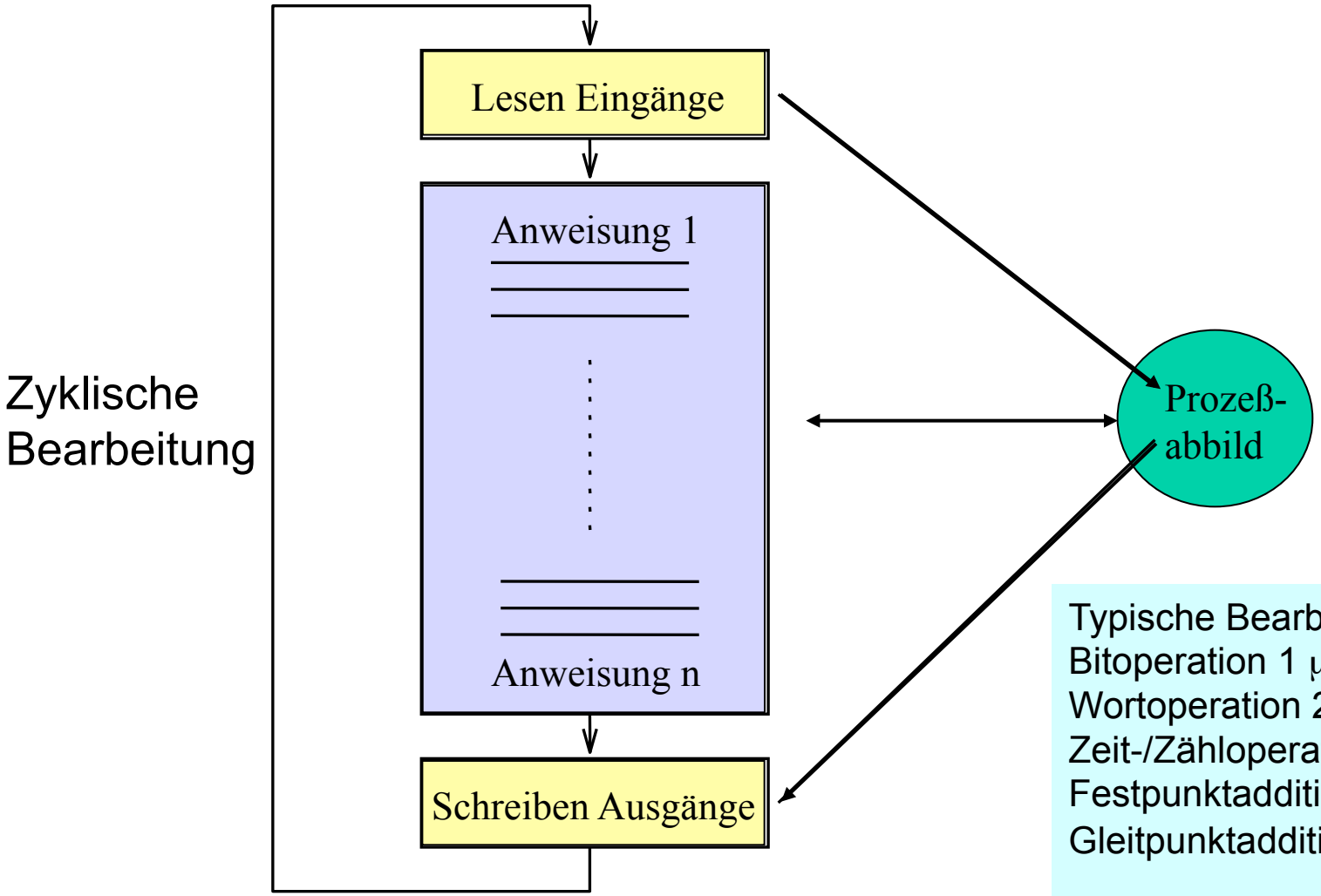
AEG Schneider Automation Concept

Aufbau einer SPS



aus:
Wellenreuther, Zastrow:
Steuerungstechnik mit SPS.
Vieweg 1995

Programmzyklus einer SPS



Typische Bearbeitungszeiten:
Bitoperation 1 µs
Wortoperation 2 µs
Zeit-/Zähloperation 12 µs
Festpunktaddition 3 µs
Gleitpunktaddition 50 µs

SPS-Zyklus 10 … 50 ms

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

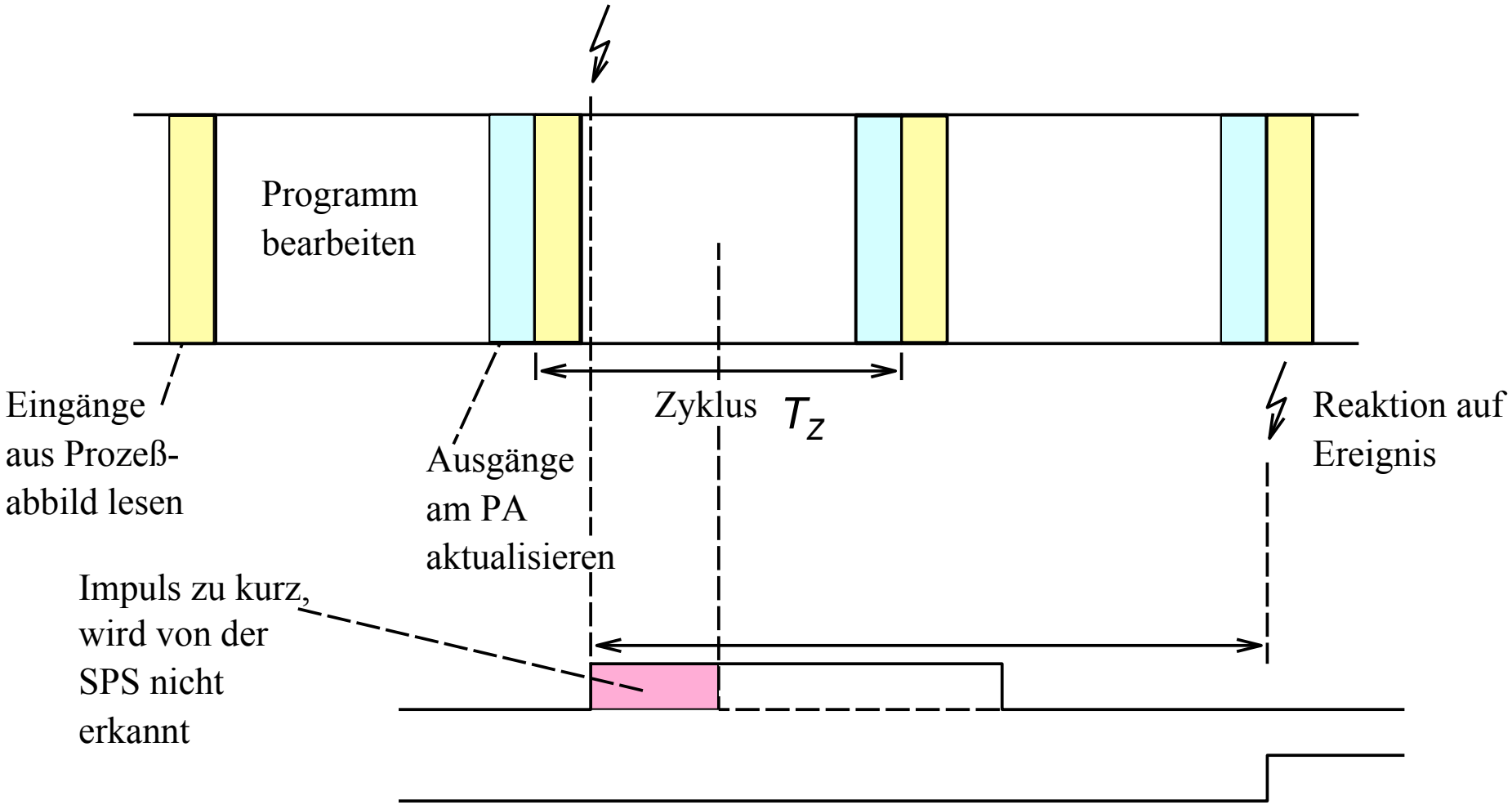
Prozessabbild

Speicherbereich zur Aufnahme der Signalwerte der Ein- und Ausgänge:

- Konstante Eingangssignale während eines Programmzyklus.
- Ausgabewerte werden zwischengespeichert, d.h. kurzzeitige Änderungen während des Zyklus haben nach außen keine Wirkung.
- maximale Reaktionszeit ist etwa doppelte Zyklusdauer T_Z
  d.h. bei gegebener maximaler Reaktionszeit ergibt sich eine Begrenzung der Programmlänge (typischer Wert: 1000 Anweisungen/ms)
- minimale Dauer eines Eingangssignals ist etwa Zyklusdauer T_z
- aufgrund von Programmverzweigungen ist die Zyklusdauer nicht konstant
  ➔ Überwachung der Zyklusdauer
  ➔ Zyklendurchlauf in regelmäßigen Abständen $T_A > T_{Zmax}$
- spezielle SPS Geräte erlauben den direkten Zugriff auf die Prozesssignale (ohne Prozessabbild)
  ➔ langsamer

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Maximale Reaktionszeit

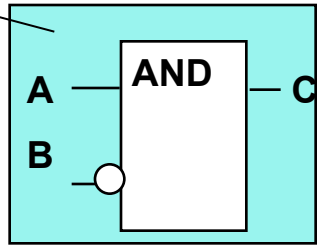


Programmiersprachen nach IEC 61131-3

KOP **Kontaktplan**
(engl. Ladder Diagram, LD)

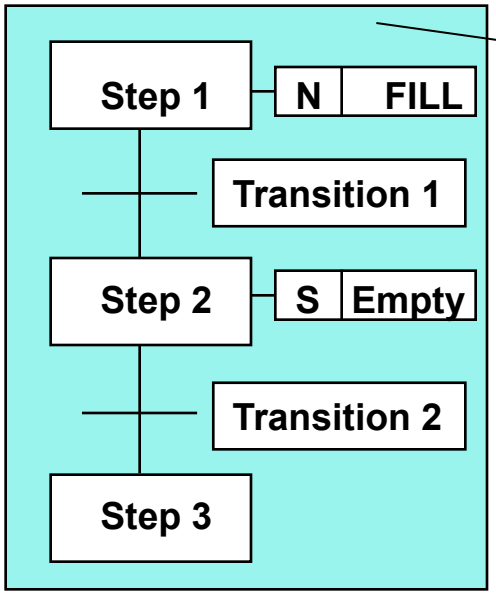
```
A   B              C
-| |--|/|----------------( )
```

FBS **Funktionsbausteinsprache**
(engl. Function Block Diagram, FBD)




```
C:= A   AND  NOT B
```

ST **Strukturierter Text**
(engl. Structured Text, ST)



AS **Ablaufsprache**
(engl. Sequential Function Chart, SFC)

AWL **Anweisungsliste**
(engl. Instruction List, IL)

```
LD      A
ANDN    B
ST      C
```

Standardisierung und Zertifizierung durch PLCopen.

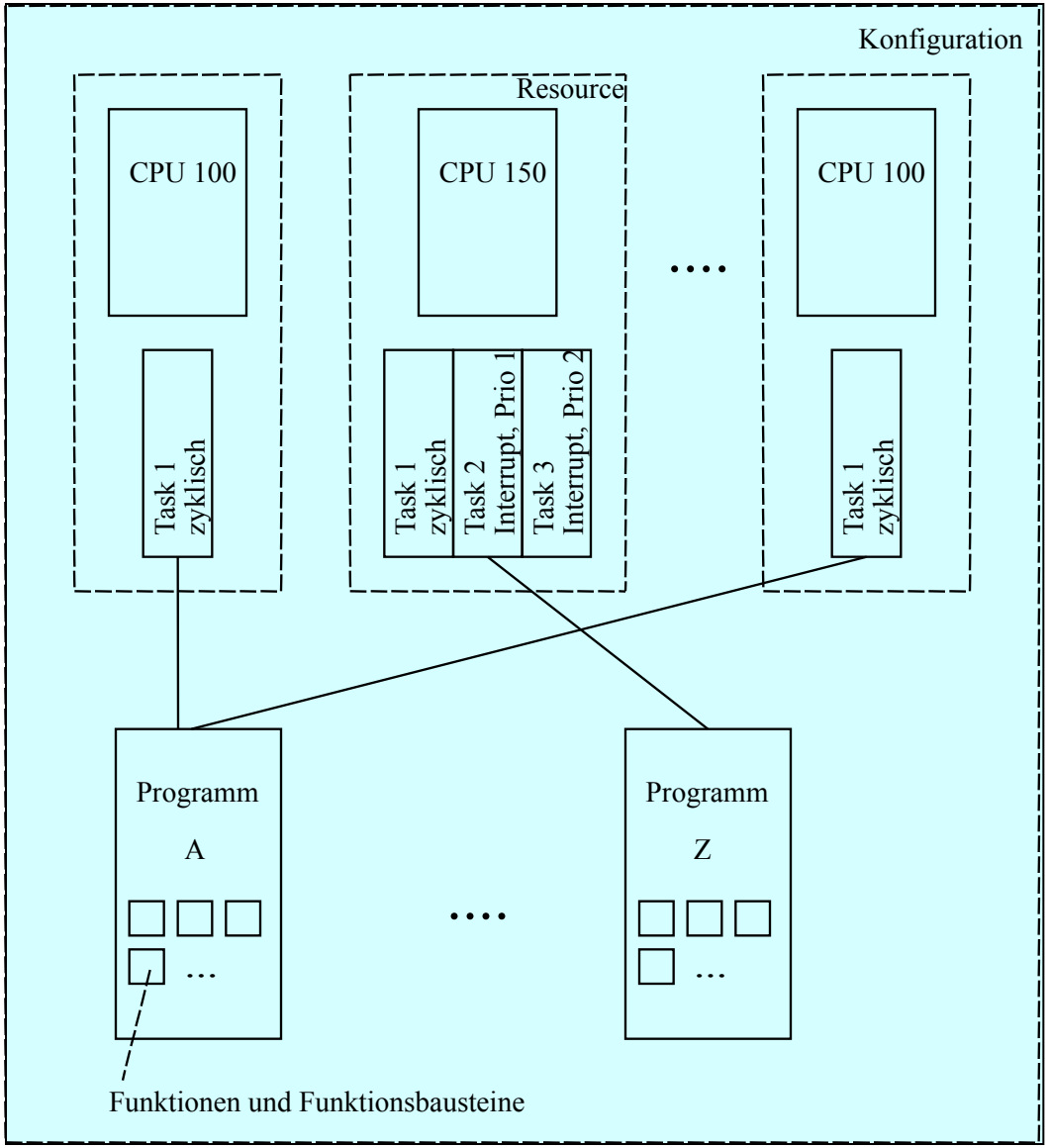
Konfiguration

In der IEC 61131-3 wird unterschieden zwischen:

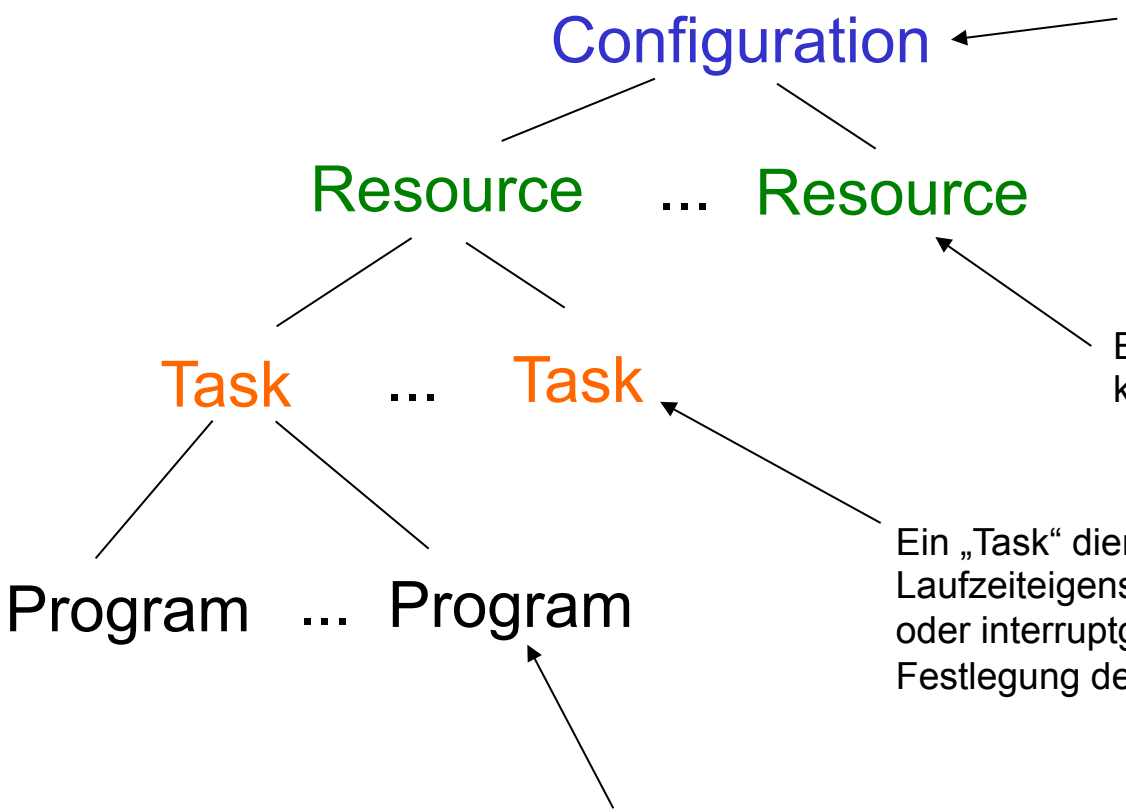
1. Konfigurations-Organisationseinheit (KOE)
   - Beschreibung der SPS Hardware (Ressource)
   - Zuordnung Software auf Hardware

2. Programm-Organisationseinheit (POE) (keine Rekursion!)
   - Programme
   - Funktionsbausteine
   - Funktionen

Programme können einer oder mehreren Ressourcen zugeordnet werden (Instanz)



Software (Struktur) Modell



Eine „Configuration“ fasst ein vollständiges Steuerungssystem bestehend aus einem oder mehreren (kommunizierenden) CPUs unter einem Projekt zusammen.

Eine „Resource“ bezeichnet ein konkretes Steuergerät (CPU).

Ein „Task“ dient der Beschreibung von Laufzeiteigenschaften, wie zyklischer oder interruptgesteuerter Aufruf oder Festlegung der Priorität.

Ein „Program“ setzt sich zusammen aus aus Netzwerken, FUNCTIONs und FUNCTION-BLOCKs implementiert in ST, FBS, AWL oder KOP.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Implementierungen

Die IEC 61131-3 enthält Sprachmittel zur präzisen Beschreibung einer realen SPS-Konfiguration. Die Implementierungen in den einzelnen Programmiersystemen entsprechen der Norm derzeit nur in Grundzügen (z.B. Siemens S7 oder AEG Modicon). Die Beschreibung der Konfiguration ist meist in die interaktiven Programmierumgebungen integriert (vgl. Siemens S7).

Die Norm verlangt keine vollständige Implementierung sondern nur die korrekte Realisierung der implementierten Sprachmittel. Es muss daher bei allen Programmiersystemen auf die Herstellerunterlagen verwiesen werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kapitel 4: Speicherprogrammierte Steuerungen

Engl.: Ladder Diagram

4.1 Aufbau und Funktion einer SPS

4.2 Kontaktplan (KOP)

4.3 Funktionsbausteinsprache (FBS)

4.4 Strukturierter Text (ST)

4.5 Anweisungsliste (AWL)

4.6 Ablaufsprache (AS)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kontaktplan (KOP) - Grundelemente

Historisch begründete, grafische Form der SPS Programmierung (Verbindungsprogrammierte Relaissteuerung für einfache VKS)

Signalgeber

Ausgang, Merker

e_1 `--]  [--`	Kontakt (vgl. Schließer)
e_1 `--]/[--`	negierter Kontakt (vgl. Öffner)
a_1 `--(  )--`	Spule (vgl. Relais)

Der Wirkungszusammenhang wird durch den fiktiven Stromfluss von links nach rechts beschrieben. Bei mehreren Netzwerken erfolgt die Bearbeitung von oben nach unten.

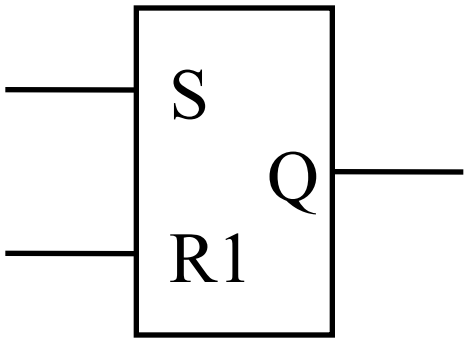
- UND-Verknüpfung durch *Reihenschaltung*
- ODER-Verknüpfung durch *Parallelschaltung*

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Weitere Elemente

Weitere Elemente der Kontaktplandarstellung:

- Sprünge, Sprungmarken und Rücksprünge
- Flankenerkennung
- Speicherelemente (Merker, Flip-Flop)
- Funktionen (kein Speicher, EIN Funktionswert)
- Funktionsbausteine (mit Speicher, mehrere Rückgabewerte)



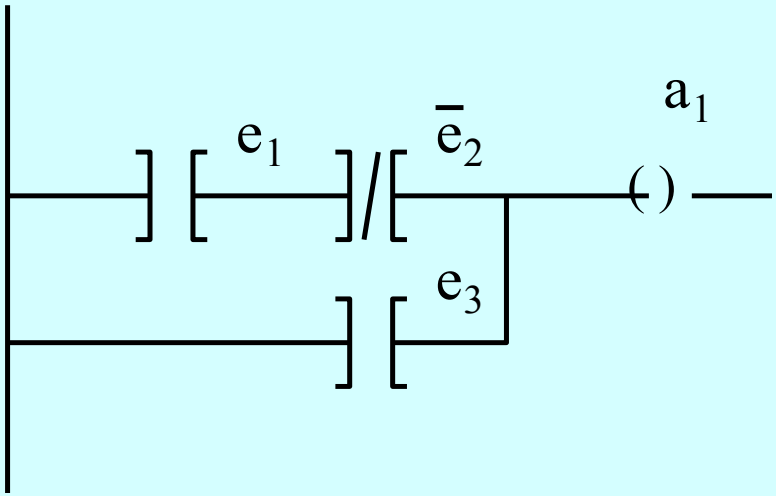
RS-Flip-Flop

- Priorität hat Rücksetzen (R**1**)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

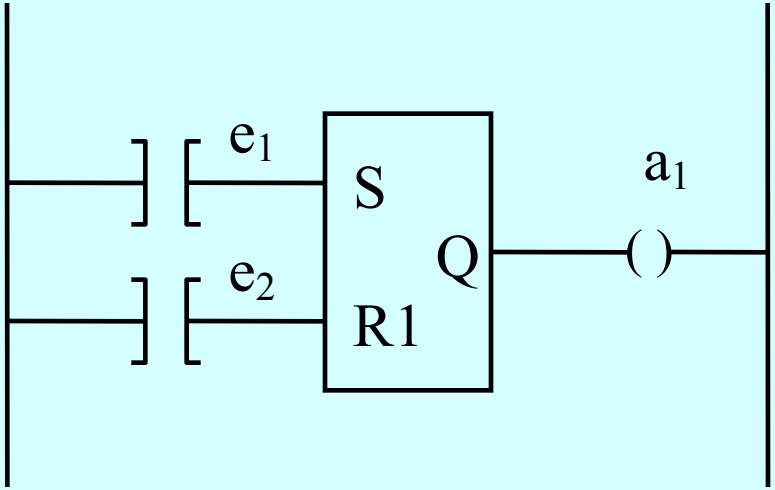
KOP - Beispiele

$$a_1 = e_3 \vee (e_1 \wedge \overline{e}_2)$$



Beispiel 1

$$a_1 = (a_1 \vee e_1) \wedge \overline{e}_2)$$



Beispiel 2

Aufgabe

In der Einführung wurde eine einfache Steuerung für ein Werktor vorgestellt. Beim Öffnen und Schließen des Werktores sollte ein Blinklicht die Gefahr signalisieren. Versuchen Sie eine Lösung in Kontaktplandarstellung für die logische Steuerung dieses Signals anzugeben. Das Blinken muss hier nicht explizit programmiert werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kapitel 4: Speicherprogrammierte Steuerungen

4.1 Aufbau und Funktion einer SPS

4.2 Kontaktplan (KOP)

4.3 Funktionsbausteinsprache (FBS)

4.4 Strukturierter Text (ST)

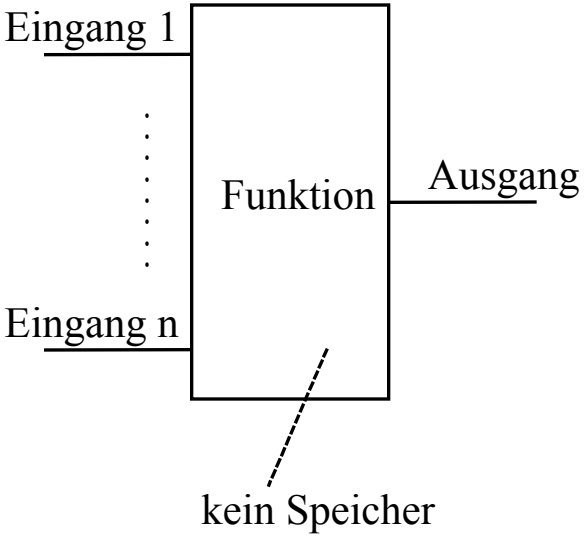
4.5 Anweisungsliste (AWL)

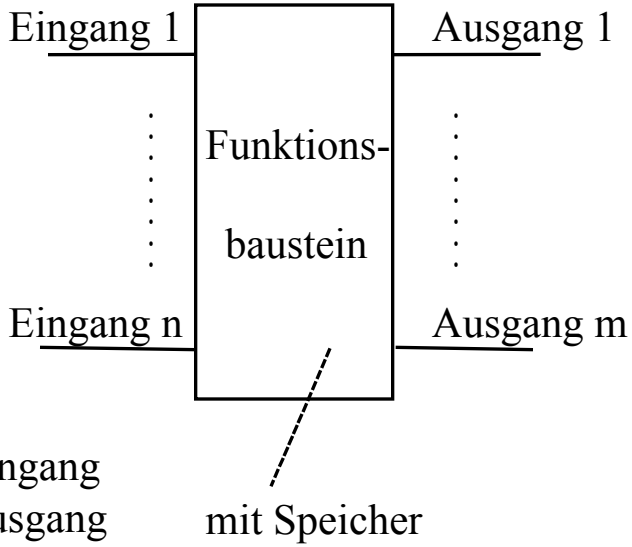
4.6 Ablaufsprache (AS)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Funktionsbausteinsprache (FBS)

- Graphische Programmiersprache für Verknüpfungssteuerungen in Form einzelner Netzwerke. Die Wirkungsrichtung (Auswertereihenfolge) entsteht durch Verkettung der Funktionen und Funktionsbausteine, wobei links immer die Eingänge und rechts immer die Ausgänge gezeichnet werden.
- Ein- und Ausgänge können miteinander beliebig verbunden werden. Insbesondere sind Rückkopplungen möglich.
- Auch hier sind Sprünge, Marken und Rücksprünge möglich (unübersichtlich).





invertierter Eingang

invertierter Ausgang

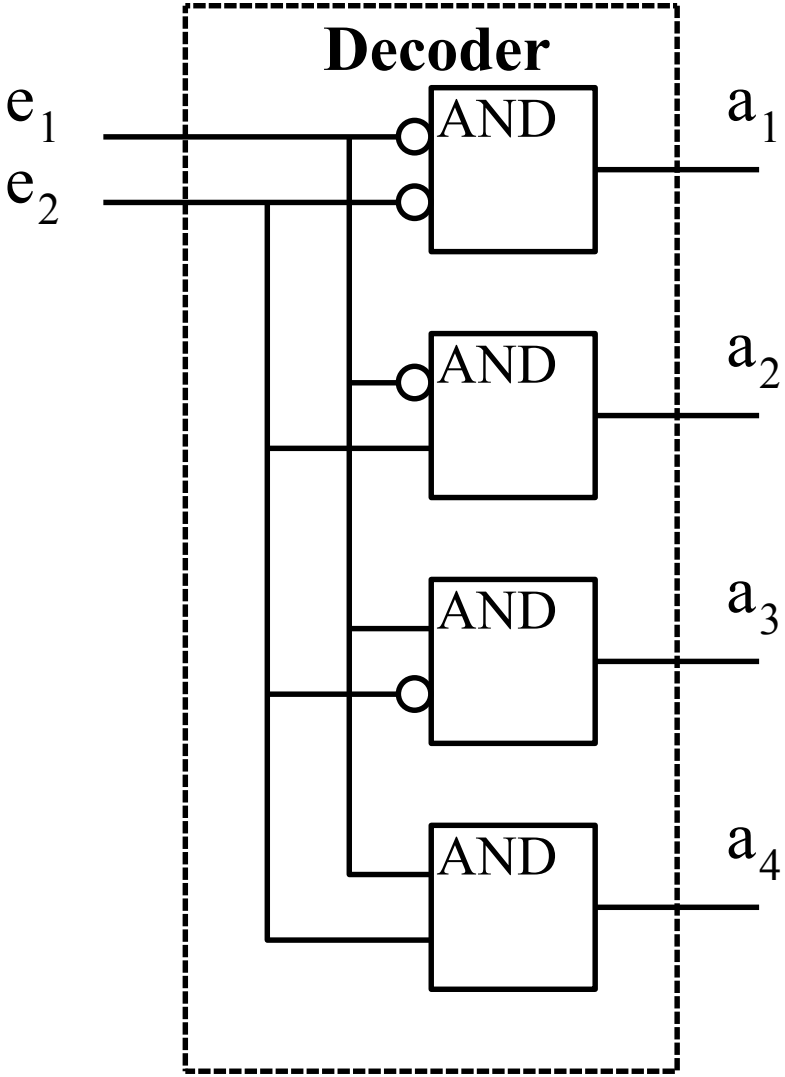
Beispiel: Decoder

Wertetafel

e_1	e_2	a_1	a_2	a_3	a_4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

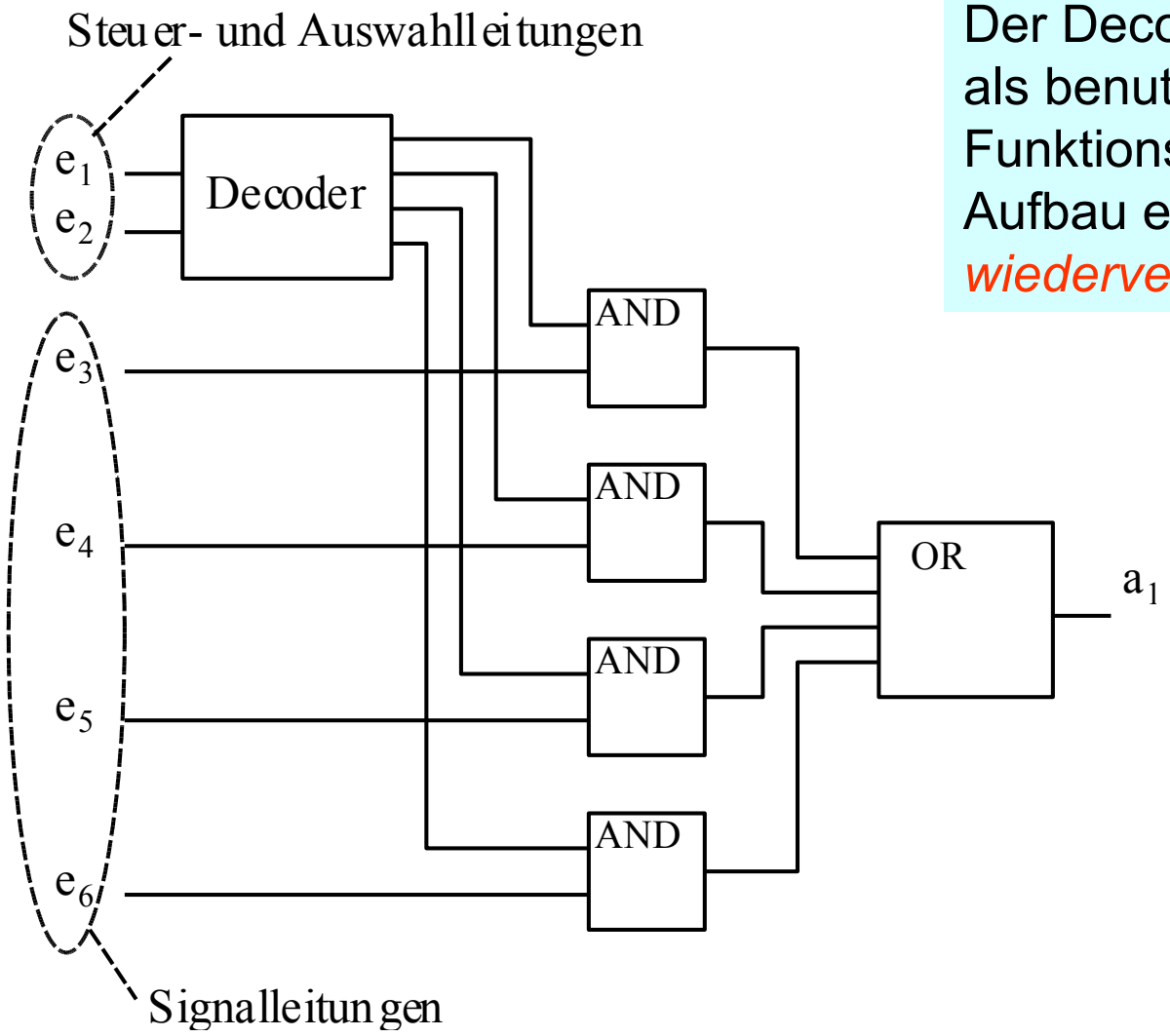
© 2015 B. Wagner - Leibniz Universität Hannover - RTS

FBS Netzwerk Decoder



© 2015 B. Wagner - Leibniz Universität Hannover - RTS

FBS Netzwerk Multiplexer



Der Decoder selbst kann als benutzerdefinierter Funktionsbaustein zum Aufbau eines Multiplexers *wiederverwendet* werden.

Speicher, Zähler und Zeitglieder

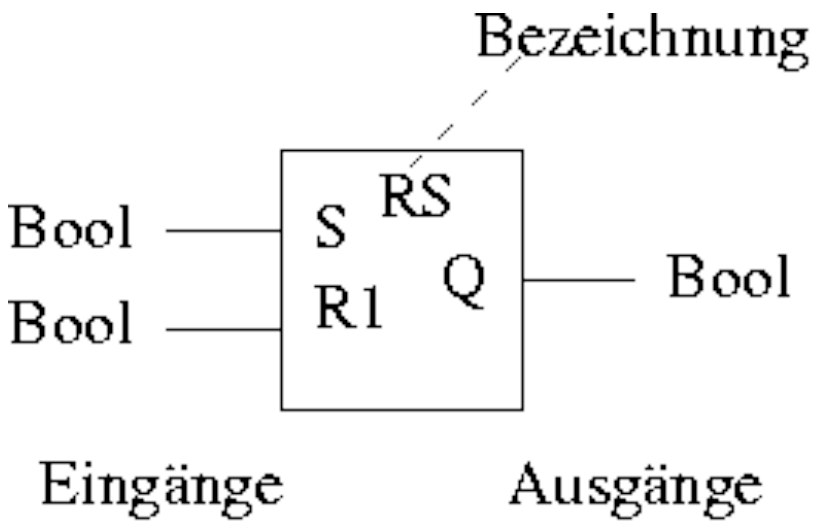
In vielen Steuerungsaufgaben sind häufig Teilprobleme der folgenden Art zu lösen:

- ein Taster soll zum Ein-, Aus- oder Umschalten verwendet werden;
  ➔ Speicherglieder Flankenerkennung
- die Anzahl der Werkstücke in einer Pufferstrecke soll kontrolliert werden
  ➔ Auf- und Abwärtszähler
- das Einschalten zweier Maschinen soll zeitversetzt erfolgen
  ➔ Ein- und Ausschaltverzögerung

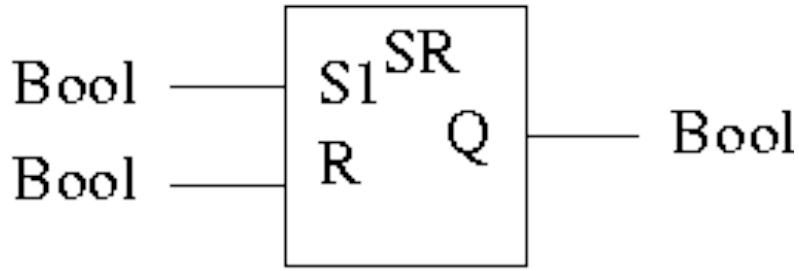
Zur Realisierung sind in der IEC 61131-3 Standardfunktionsbausteine vorgesehen. Die Standard-FBs können in Programmiersystemen aus vorbereiteten Bibliotheken ausgewählt werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

RS: Bistabiler Funktionsbaustein



Rücksetzen dominant

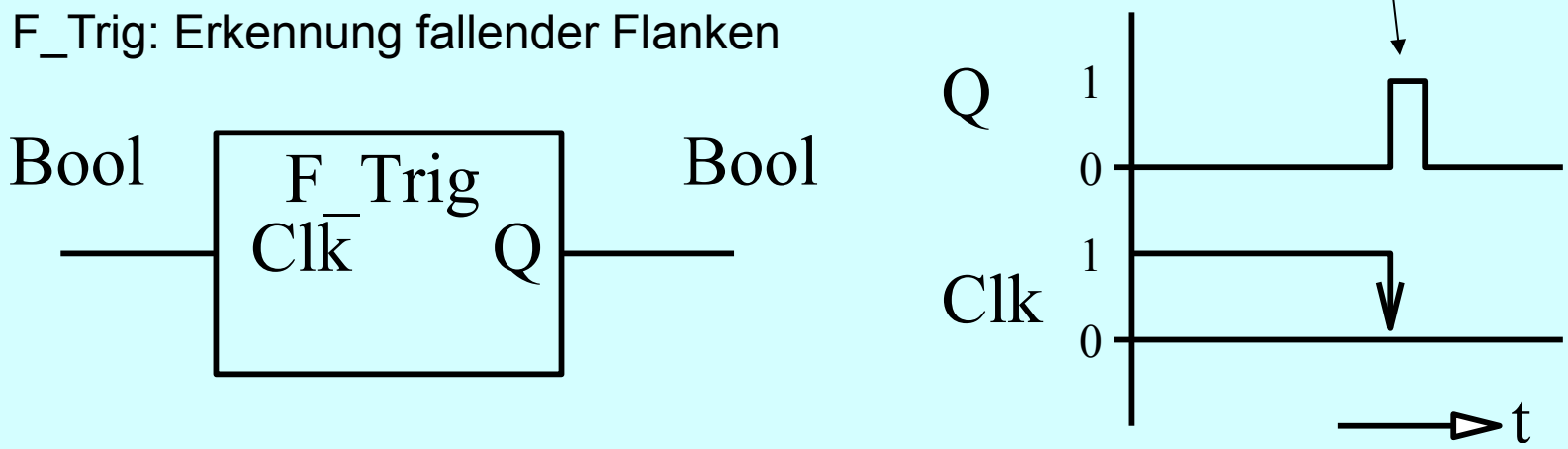


Setzen dominant

Flankenerkennung

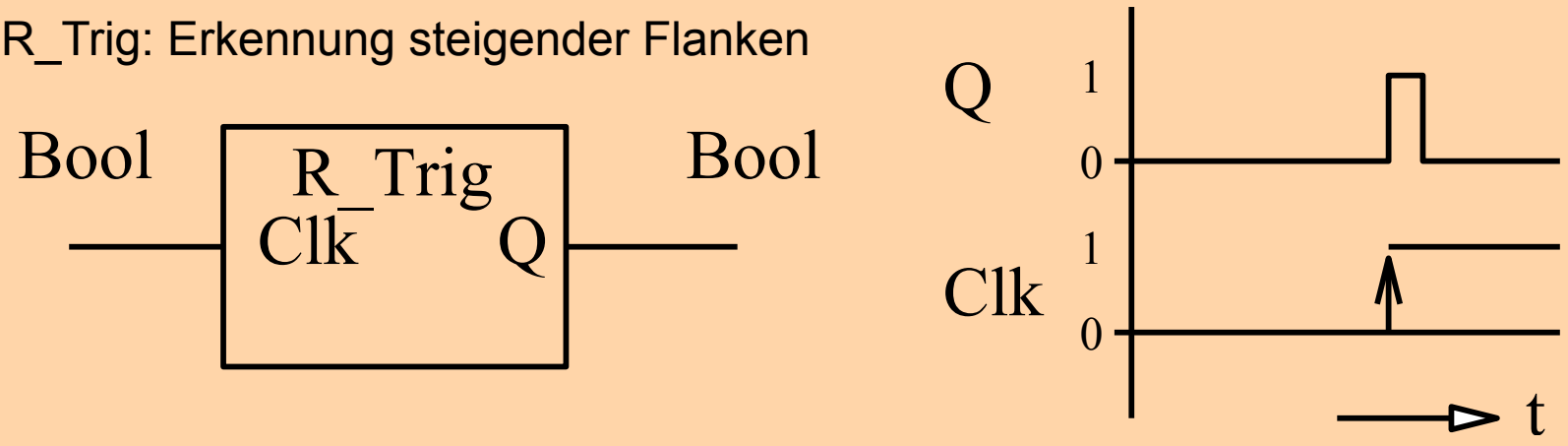
Pulsdauer genau ein SPS-Zyklus

F_Trig: Erkennung fallender Flanken



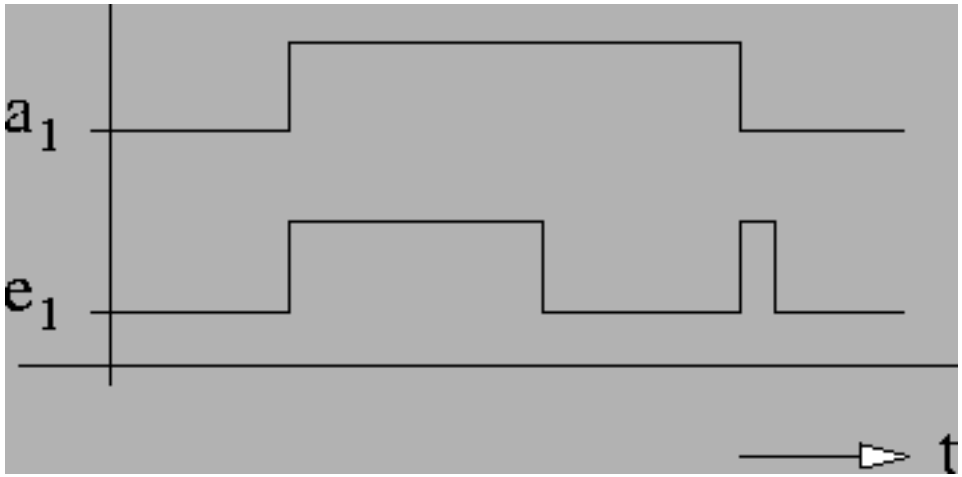
Nach einem Übergang 1->0 am Eingang bleibt am Ausgang bis zur nächsten Ausführung (T_z) eine 1; danach schaltet der Ausgang wieder auf 0.

R_Trig: Erkennung steigender Flanken

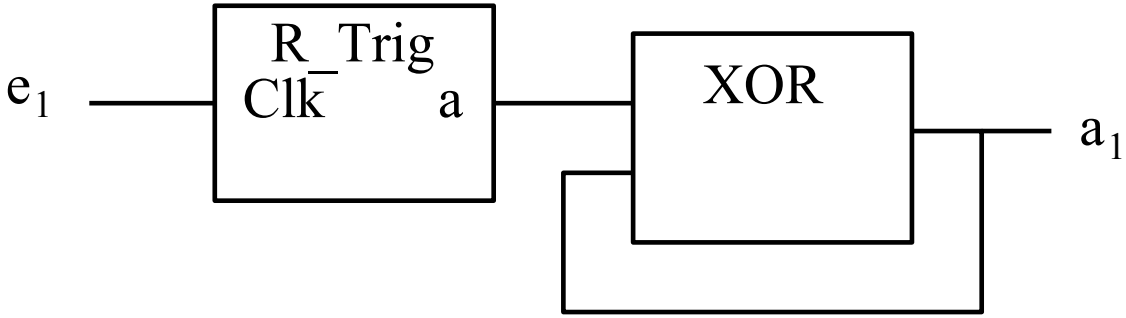


Beispiel

Ein- und Aussteuerung mit einem Taster



Lösung: Realisierung mit Standardfunktionen

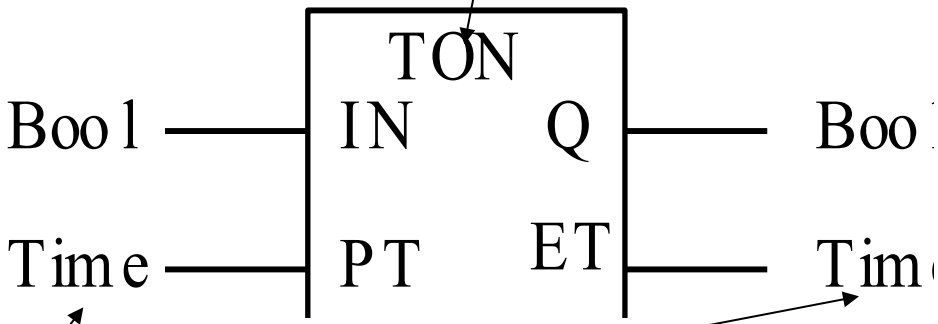


© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Zeitgeberbausteine

Zeitgeberbaustein Einschaltverzögerung (TON)
IN .... Start Zeitgeber (steigende Flanke)
PT ... Voreinstellung Zeit
Q ..... Ausgang
ET ... interne Zeit

TON: Einschaltverzögerung
TOF: Ausschaltverzögerung
TP: Pulsgeber

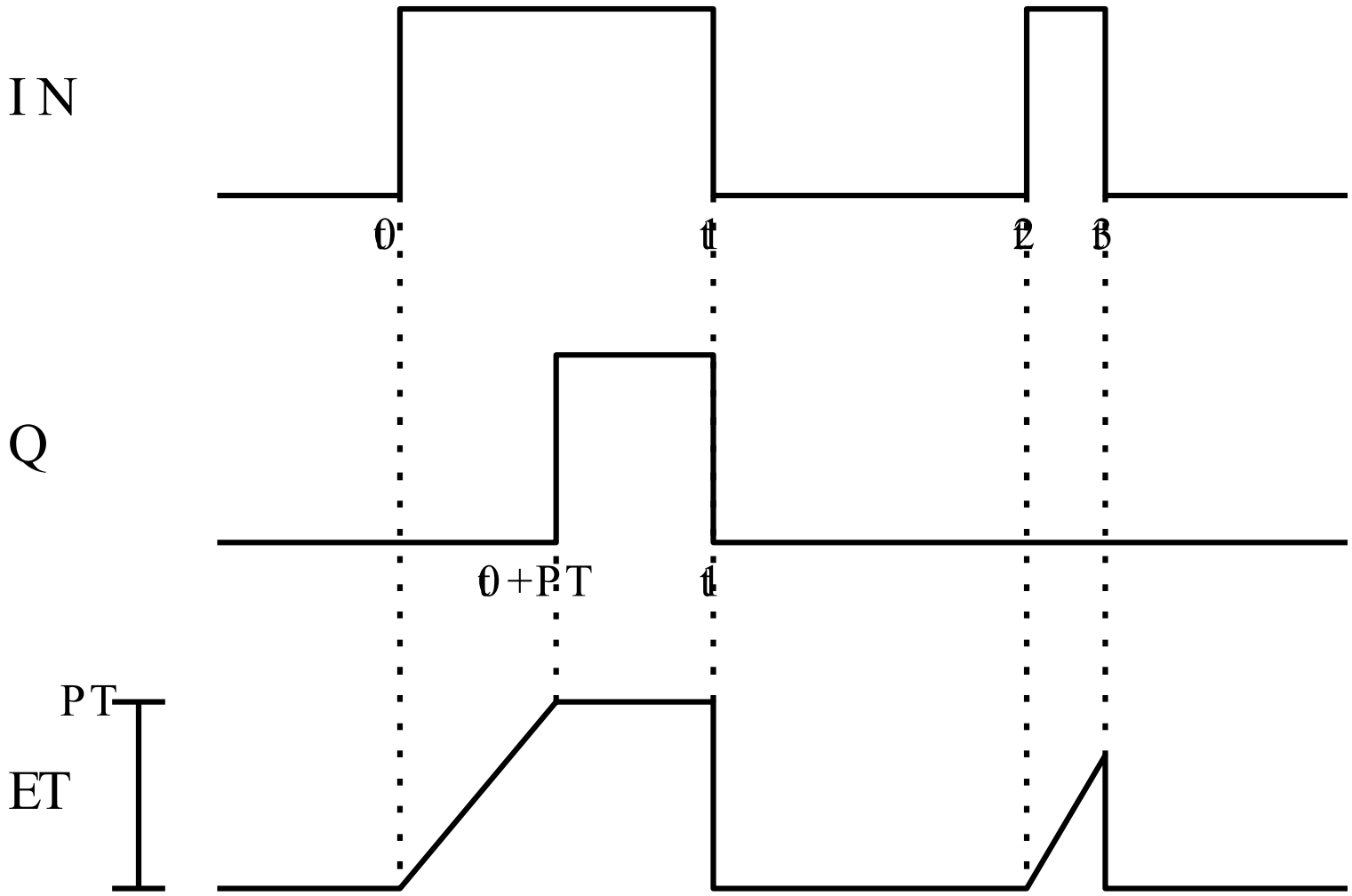


Einheiten:
d ..... Tag
h ..... Stunde
m .... Minute
s ..... Sekunde
ms .. Millisekunde

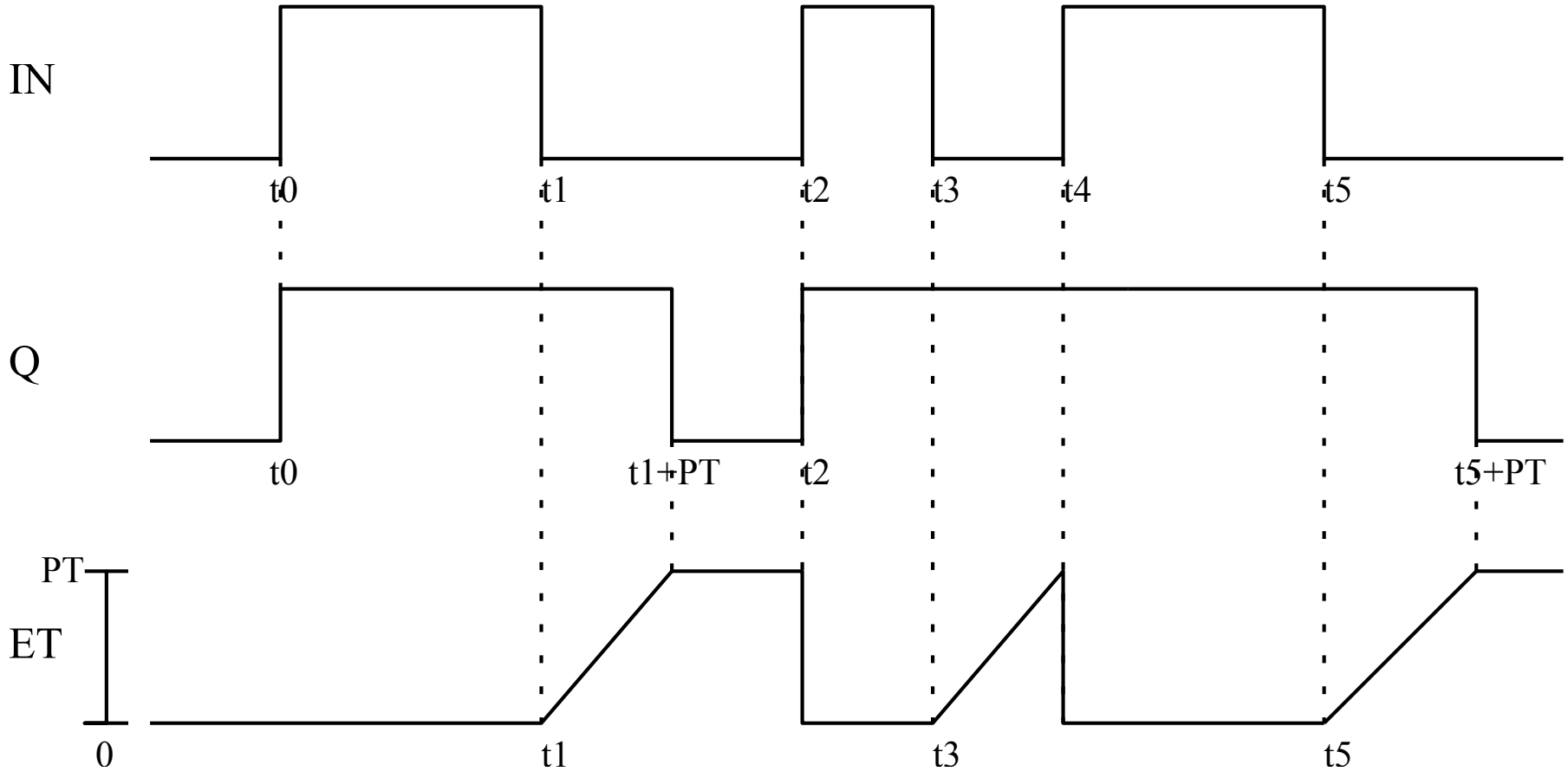
Angabe von Zeitdauern als Konstante:

```
T#14.7ms
TIME#3h_12m_4.3s
```

Zeitdiagramm TON

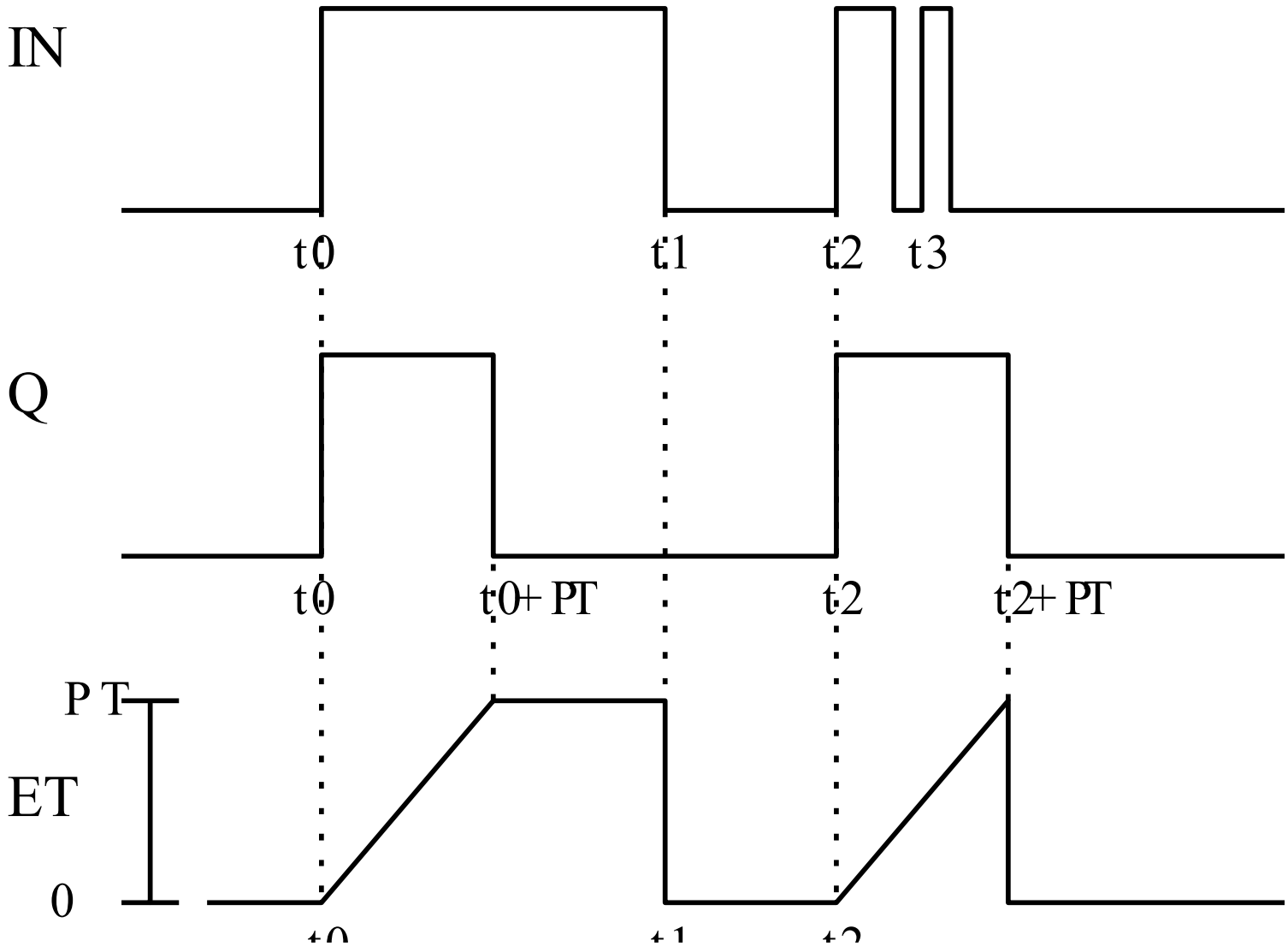


Zeitdiagramm TOF



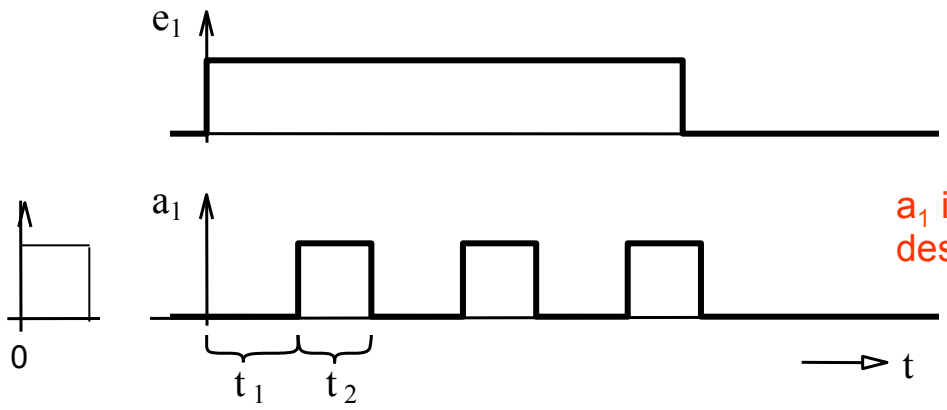
© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Zeitdiagramm TP



Beispiel: Blinklicht

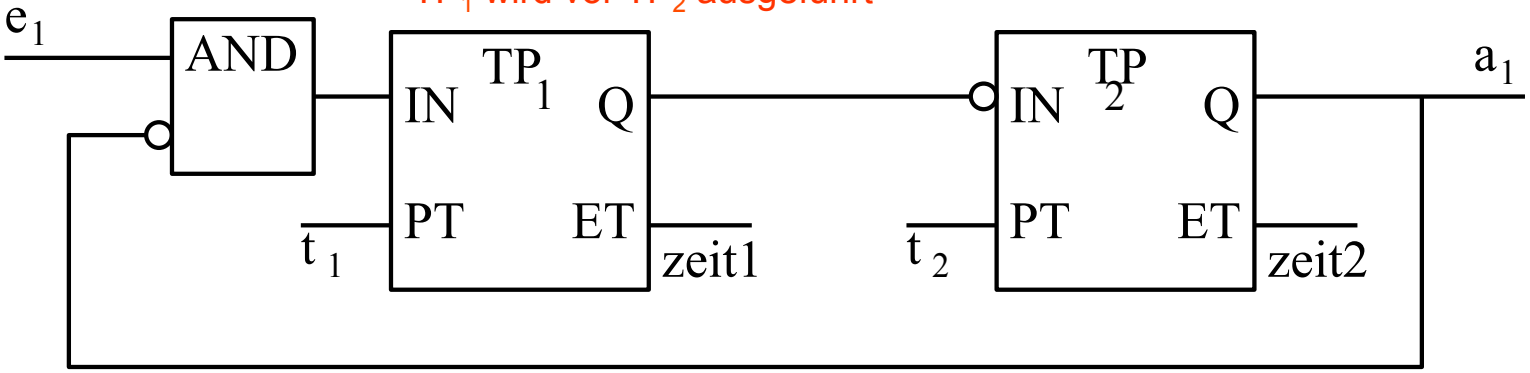
Blinklichtsteuerung aus elementaren Bausteinen: Wenn das Eingangssignal e_1 auf True wechselt (Flanke) soll das Blinklicht im Wechsel für die Zeitdauer t_1 ausgeschaltet und für die Dauer t_2 eingeschaltet werden. Wenn e_1 wieder auf False steht, soll das Blinklicht wieder ausgeschaltet werden.



a_1 ist beim Einschalten für die Dauer t_2 des Gesamtsystems einmalig 1!

```
VAR zeit1,zeit2:TIME;
END_VAR
```

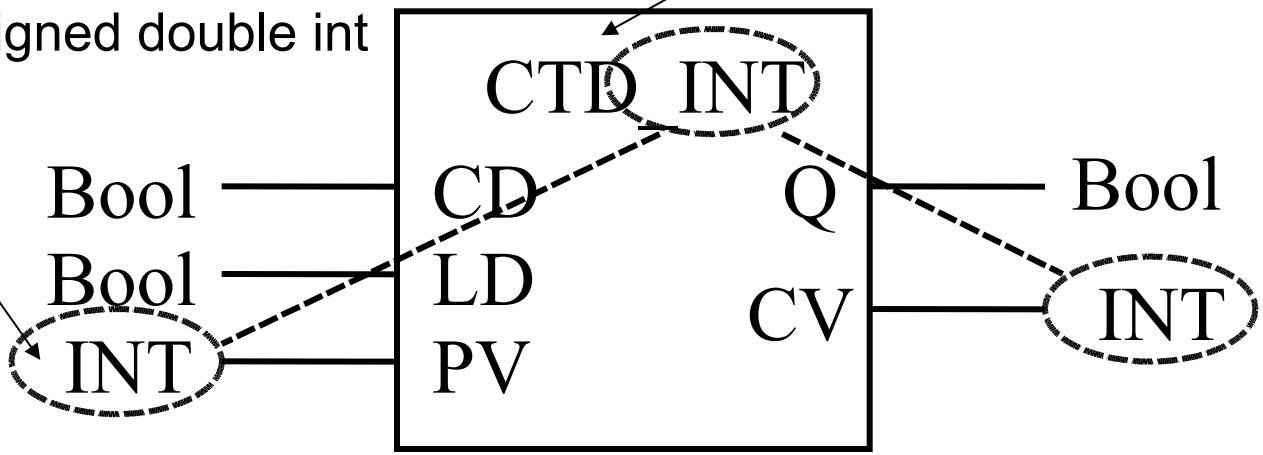
TP_1 wird vor TP_2 ausgeführt



Zählerbausteine

oder:
DINT: double int
UINT: unsigned int
UDINT: unsigned double int

CTD: Abwärtszähler
CTU: Aufwärtszähler
CTUD: Auf-/Abwärtszähler

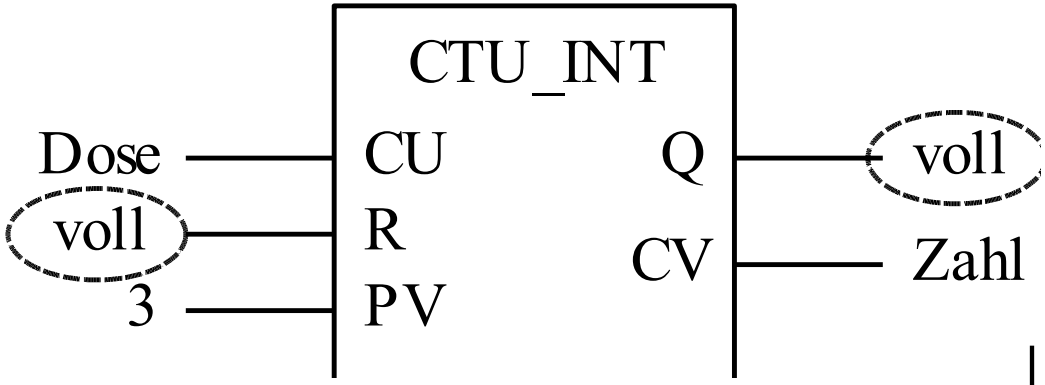


Abwärtszähler:
CD: Triggereingang
LD: Lade Daten
PV: Voreinstellungswert
Q: Ausgang
CV: Zählwert

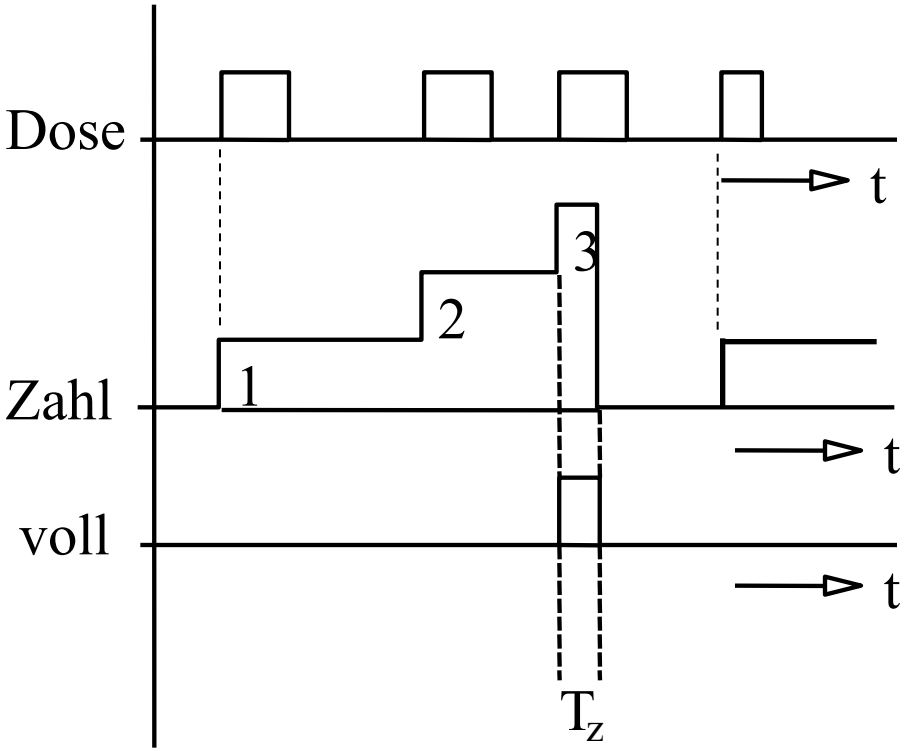
auch:
CU: Triggereingang
R: Rücksetzen
QU: CV >= PV
QD: CV <= 0

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Aufwärtszähler




```
VAR    Dose,voll:BOOL;
       Zahl:INT;
END_VAR
```



Kapitel 4: Speicherprogrammierte Steuerungen

4.1 Aufbau und Funktion einer SPS

4.2 Kontaktplan (KOP)

4.3 Funktionsbausteinsprache (FBS)

4.4 Strukturierter Text (ST)

4.5 Anweisungsliste (AWL)

4.6 Ablaufsprache (AS)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Grundaufbau einer POE

alternativ: PROGRAM, FUNCTION, FUNCTION_BLOCK

POE NAME
Schnittstellenvariable, lokale Variable ➔Deklarationsteil
Anweisungen ➔ Rumpf, Anweisungsteil
END_POE

PROGRAM:
Hauptprogramm, Deklaration von Variablen mit fester physikalischer Adresse.

FUNCTION_BLOCK:
Parametrierbare POE mit statischen Variablen und *mehreren* Ausgangsvariablen.

FUNCTION:
Parametrierbare POE ohne statische Variable und *einem* Funktionswert.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Variablen Deklaration

➔Aufrufschnittstelle

VAR_INPUT	Eingangsvariablen (call by value)
VAR_IN_OUT	Ein- und Ausgangsvariable (call by reference)
VAR_OUTPUT	Ausgangsvariable (Rückgabewert)

➔lokale Daten

VAR	lokale Variable (statisch!)

➔globale Daten

VAR_ACCESS	Zugriffspfade in der Konfiguration
VAR_GLOBAL	globale Variable (Deklaration)
VAR_EXTERNAL	externe Variable (spezielle Referenz)

END_VAR	Blockende für zuletzt geöffneten Deklarationsblock

Beispiele:

```
VAR
   alarm : BOOL;
END_VAR
```

```
VAR_INPUT
   ein, aus : BOOL;
   verzoegerung : TIME;
END_VAR
```

Es können mehrere Blöcke in beliebiger Reihenfolge angegeben werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Anweisungen

Zuweisung:

z.B. alarm := ein AND aus;

Unterprogrammaufruf:

z.B. lampe(S:=ein, R1:=aus);

Kontrollanweisung:

z.B.
```
IF NOT alarm
    THEN lampe(S:=ein, R1:=aus);
END_IF;
```

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

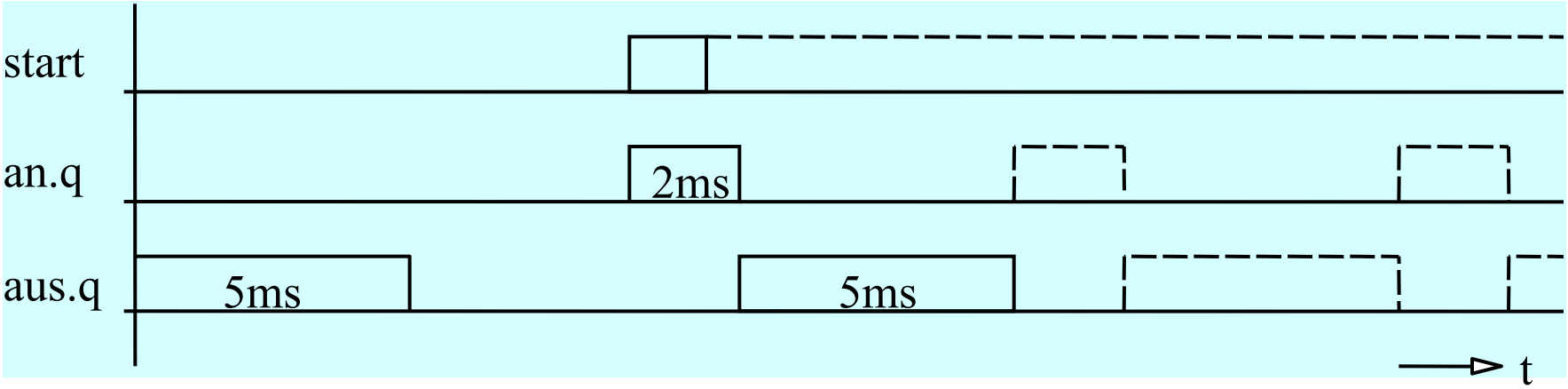
Beispiel Signallampe

```
FUNCTION_BLOCK takt
   VAR_INPUT start : BOOL; zeit1, zeit2 : TIME; END_VAR
   VAR_OUTPUT q : BOOL; END_VAR
   VAR an, aus :TP; END_VAR

   an(IN := start & NOT aus.q, PT := zeit1);
   aus(IN := NOT an.q, PT := zeit2);
   q := an.q;
END_FUNCTION_BLOCK

PROGRAM signallampe
   VAR schalter AT %IX2.1 : BOOL; lampe AT %QX3.5 : BOOL;
       blinken : takt ; END_VAR

   blinken(start := schalter, zeit1 := t#2ms,
           zeit2 := t#5ms);
   lampe := blinken.q;
END_PROGRAM
```

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Signallampe Zeitdiagramm

Zeitdiagramm des Beispiels:



Absolute Adressen sollten nur in Datendeklarationen verwendet werden. In den Anweisungsteilen sollten nur symbolische Adressen eingesetzt werden.

Absolute Adressen:

1. Zeichen: %
2. Zeichen: I : Eingang, Q : Ausgang, M : Merker
3. Zeichen: X: Bit, B : Byte, W : Wort, D : Doppelwort, L : Langwort
4. Eigentliche Adresse.

Beispiel: %QB7

Signallampe mit Zähler

mit Programmerweiterung

```
FUNCTION_BLOCK takt
   VAR_INPUT start : BOOL; zeit1, zeit2 : TIME; END_VAR
   VAR_OUTPUT q : BOOL; END_VAR
   VAR an, aus :TP; END_VAR

   an(IN := start & NOT aus.q, PT := zeit1);
   aus(IN := NOT an.q, PT := zeit2);
   q := an.q;
END_FUNCTION_BLOCK

PROGRAM signallampe
   VAR schalter AT %IX2.1 : BOOL; lampe AT %QX3.5 : BOOL;
       blinken : takt; zaehler : CTU_INT; END_VAR

   blinken(start := schalter AND NOT zaehler.Q,
           zeit1 := t#2ms, zeit2 := t#5ms);
   lampe := blinken.q;
   zaehler(CU := blinken.Q, R := NOT schalter, PV:=10);
END_PROGRAM
```

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Elementare Datentypen

BOOL	0,1 / FALSE, TRUE	Binär- und Bitfolge
BYTE	2#1011 oder 2#0000_1011	„
WORD	16#FF04	„
DWORD	50120	„
LWORD		„
SINT	-127	Ganzzahlen mit Vorzeichen
INT	0	„
DINT	+4012	„
LINT	-100000	„
USINT	0	Ganzzahlen ohne Vorzeichen
UINT	12	„
UDINT	3000	„
ULINT		„
REAL	74e3	Gleitpunktzahlen (IEC959)
LREAL	-0.123e-12	„
DATE/D	d#1998-06-25	Datum
TIME_OF_DAY/TOD	tod#13:50:12.30	Uhrzeit
DATE_AND_TIME/DT	dt#1999-12-24-18:00:00.00	Datum u. Uhrzeit
TIME/T	t#2ms	Zeitdauer
STRING	´Feder...´	Zeichenfolge

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Namen

1. Erlaubte Zeichen: a..zA..Z_0..9
2. Erstes Zeichen: a..zA..Z_
3. Groß- und Kleinschreibung wird nicht unterschieden

Beispiele: Ventil_3
_test_fbs_12
EndlageSchalter

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Typkonversion

Variablen, Konstanten und Funktionen haben einen bestimmten Typ. Bei gemischter Verwendung ist eine explizite Typkonversion notwendig.

```
VAR i : INT;
    r : REAL;
END_VAR

r:=INT_TO_REAL(i);
i:=TRUNC(r);

(* INT_TO_REAL() und TRUNC()
   sind Standardfunktionen *)
```

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Selbstdefinierte Datentypen

Aufzählung

```
TYPE
      werkstueck (feder,bolzen,gehaeuse);
      (* Konstanten: feder=1,bolzen=2,gehaeuse=3 *)
END_TYPE

VAR
      teil : werkstueck := feder;
      (* Zuweisung eines Anfangswertes ist optional *)
END_VAR

      (* teil:=4 ist eine Fehler! *)
```

Bereich

```
TYPE
      tankfuellung : INT(0...58);
      (* eingeschraenkter Wertebereich *)
END_TYPE
```

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Zusammengesetzte Datentypen (1)

Feld

```
TYPE
  messwert : ARRAY[1..50] of REAL;
END_TYPE

VAR
  druckverlauf : messwert;
  druck : REAL;
END_VAR

druck:=druckverlauf[15];
```

Zusammengesetzte Datentypen (2)

Struktur (Datenrecord)

```
TYPE
  ventil : STRUCT
             vorhanden : BOOL;
             ausschuss : BOOL;
             zylinder : BOOL;
             farbe : INT;
           END_STRUCT;
END_TYPE

VAR
  teile_liste : ARRAY[1..100] of ventil;
END_VAR

teile_liste[7].vorhanden:=TRUE;
```

Diverses

Gepufferte Variable

```
VAR_RETAIN zustand : INT := 0;
(* der Wert der Variablen bleibt
   auch beim Spannungsausfall erhalten *)
```

Konstante als Symbol

```
VAR_CONSTANT max_druck : REAL:=4.5; (*bar*)
```

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Ampelsteuerung in Strukturiertem Text auf Basis von Funktionsbausteinen

```
0001 PROGRAM OB1
0002 VAR
0003     rsGruen: RS;
0004     rsGelb: RS;
0005     rsRot: RS;
0006     TimerGelb: TON;
0007     TimerGruen: TON;
0008     FussgaengerGruen AT %QX1.4 : BOOL;
0009     FussgaengerRot AT %QX1.3 : BOOL;
0010     Start: BOOL := TRUE;
0011     TimerRotGelb: TON;
0012     TimerRot: TON;
0013     DruckMerker: RS;
0014     Gruen AT %QX1.2 : BOOL;
0015     Gelb AT %QX1.1 : BOOL;
0016     Rot AT %QX1.0 : BOOL;
0017     Druck AT %IX0.0 : BOOL;
0018 END_VAR
```

```
0001 (* Fussgängerampel Einschalten *)
0002 IF Start THEN
0003    rsGruen(SET := TRUE);
0004    Start := FALSE;
0005 END_IF;
0006
0007 (* Autos: grün, Fussgänger: rot *)
0008 rsGruen(SET := TimerRotGelb.Q);
0009 rsRot(RESET1 := TimerRotGelb.Q);
0010 rsGelb(RESET1 := TimerRotGelb.Q);
0011 DruckMerker(SET := Druck AND rsGruen.Q1, RESET1 := NOT rsGruen.Q1);
0012 TimerGruen(IN := Druckmerker.Q1, PT := T#4s);
0013
0014 (* Autos: gelb, Fussgänger: rot *)
0015 rsGelb(SET := TimerGruen.Q);
0016 rsGruen(RESET1 := TimerGruen.Q);
0017 TimerGelb(IN := rsGelb.Q1 AND NOT rsRot.Q1, PT := T#2s);
0018
0019 (* Autos: rot, Fussgänger: grün*)
0020 rsRot (Set := TimerGelb.Q);
0021 rsGelb(RESET1 := TimerGelb.Q);
0022 TimerRot(IN := rsRot.Q1, PT := T#4s);
0023
0024 (* Autos: rot+gelb, Fussgänger: rot *)
0025 rsRot (SET := TimerRot.Q);
0026 rsGelb(SET := TimerRot.Q);
0027 TimerRotGelb(IN := rsRot.Q1 AND rsGelb.Q1, PT := T#2s);
0028
0029 (* Ausgaben setzen *)
0030 Rot := rsRot.Q1;
0031 Gelb := rsGelb.Q1;
0032 Gruen := rsGruen.Q1;
0033 FussgaengerGruen := rsRot.Q1 AND NOT rsGelb.Q1;
0034 FussgaengerRot := NOT FussgaengerGruen;
```

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Operatoren

Priorität	Operation	Symbol	Bemerkung / Beispiel
1	Klammerung	(<Ausdruck>)	
2	Funktionsaufruf	<Funktionsname> (<Argumentliste>)	LN(a), MAX(x,y)
3	Potenzierung	**	
4	Negation	-	unäres Minus, Vorzeichen
4	Komplement	NOT	
5	Multiplikation	*	
5	Division	/	
5	Modulo	MOD	Rest bei Integer-Division
6	Addition	+	
6	Subtraktion	-	binäres Minus
7	Vergleich	<, >, <=, >=	
8	Gleichheit	=	
8	Ungleichheit	<>	
9	boolesches UND	&, AND	beide Operatoren möglich
10	boolesches EXKLUSIV-ODER	XOR	
11	boolesches ODER	OR	

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Deklaration und Aufruf von Funktionen

Funktionsdeklaration:

Bsp.:

```
FUNCTION Differenz : INT
VAR_INPUT
   w1, w2 : INT;
END_VAR
   Differenz := w1-w2;
END_FUNCTION
```

Typ des Rückgabewertes

Funktionsaufruf:

Bsp.:

```
VAR
   i1 : INT := 12; i2 : INT := 7;
   i3 : INT;
END_VAR
...
I3 := Differenz(w1:=i1, w2:=i2); (* MIT Formalparam. *)
I3 := Differenz(i1, i2); (* OHNE Formalparam. *)
```

Vor einem Funktionsaufruf müssen keine *Instanzen* in der Deklaration erzeugt werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Standardfunktionen

- numerische Funktionen:
  `ABS, SQRT, LN, LOG, EXP, SIN, COS, TAN, ASIN, ACOS, ATAN`
- Funktionen zur Bitmanipulation:
  `SHL, SHR, ROL, ROR`
- Funktionen zur Auswahl:
  `MAX, MIN`
- Funktionen für Zeichenfolgen:
  `REPLACE`
- Funktionen zur Typwandlung:
  `*_TO_**, TRUNC`

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Zuweisung und Funktionsaufruf

Anweisungstyp	Beispiel / Bemerkungen
Zuweisung	`a := b; CV := CV+1;` `c := SIN(x);`
Leeranweisung	`;;`

Aufruf eines FBS mit Zuweisung des Ergebnisses	`TB_L (Enable:=TRUE,`
	`Thigh:=T#300ms,Tlow:=T#300ms);`
	`Lampe := TB_L.Q;`

RETURN	`RETURN;` Verlassen einer Funktion oder eines Funktionsbausteins

Strukturierter Text ist eine an Pascal angelehnte höhere Programmiersprache mit zusätzlichen Sprachmitteln, wie Zeitverarbeitung oder Zugriff auf Prozesssignale.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kontrollstrukturen

Anweisungstyp	Beispiel / Bemerkung

IF	`IF d < 0.0`
	`  THEN n := 0;`
	`  ELSIF d = 0.0`
	`  THEN n := 1;`
	`  x1 := -b / (2.0*a);`
	`  ELSE n := 2;`
	`  x1 := (-b + SQRT (d)) / (2.0*a);`
	`END_IF;`

CASE	`CASE TW OF`
	`  1,5: anzeige := temperatur;`
	`  2: anzeige := umdrehung;`
	`  3: anzeige := brutto-tara;`
	`  4,6..10: anzeige := status (tw-4);`
	`  ELSE anzeige := 0;`
	`  fehler := 1;`
	`END_CASE;`

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Schleifen (1)

Anweisungstyp	Beispiel / Bemerkung

FOR Zählschleife	`FOR i := 1 TO 100 BY 2 DO`
	`  IF i = k`
	`  THEN EXIT;`
	`  END_IF;`
	`  zaehl[i] := i + 3;`
	`END_FOR;`
	`(* Die Zählschleife ist dann geeignet,`
	`wenn die Anzahl der Durchläufe von`
	`vornherein feststeht. *)`

WHILE abweisende Schleife	`i := 1;`
	`WHILE i <= 100 & i <> k DO`
	`  zaehl [i] := i + 3;`
	`  i := i + 2;`
	`END_WHILE;`
	`(* Im Extremfall wird die`
	`WHILE-Schleife nicht durchlaufen. *)`

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Schleifen (2)

Anweisungstyp	Beispiel / Bemerkung

REPEAT
nicht
abweisende
Schleife
	`i := 1;`
	`IF i <> k`
	`  THEN`
	`  REPEAT`
	`  zaehl [i] := i + 3;`
	`  i := i + 2;`
	`  UNTIL i > 100 OR i = k END_REPEAT;`
	`END_IF;`
	`(* Die REPEAT-Schleife wird immer mindestens einmal durchlaufen. *)`

EXIT	`(* EXIT verlässt bei geschachtelten Schleifen jeweils die innerste. *)`

Kapitel 4: Speicherprogrammierte Steuerungen

4.1 Aufbau und Funktion einer SPS

4.2 Kontaktplan (KOP)

4.3 Funktionsbausteinsprache (FBS)

4.4 Strukturierter Text (ST)

4.5 Anweisungsliste (AWL)

4.6 Ablaufsprache (AS)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

AWL-Programmierung

Assemblerähnliche, aber steuerungstechnisch erweiterte bzw. neu definierte Programmiersprache. Starke Verbreitung bei Verwendung älterer Steuerungssysteme (vor allem Siemens S5):

- größtmögliche Flexibilität gegenüber FUP und KOP, z.B. bei der Programmierung von Abläufen, Verwendung von Merkern usw.
- maximale Effizienz, da sehr maschinenorientiert
- fehleranfällig
- schwierig änderbar und erweiterbar
- ungeeignet als Dokumentation für Anlagenbetreiber
- keine Symbole (S5)
- Sprachmittel zur strukturierten Programmierung fehlen
- keine Datenstrukturen
- mangelhafte Modularisierung

Grundlage der Programmierung ist ein einfacher virtueller Prozessor mit einem abstrakten universellen Akkumulator (Rechen- und Verknüpfungsregister).

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Aufbau einer AWL-Anweisung

```
Sprungmarke*: Operator Operand(en)* Kommentar*

                                  * können entfallen
```

Beispiel:

ST-Anweisung:

```
      a1 := e1 & e2 OR NOT e3;
```

AWL-Anweisung:

```
      LD    e1     (*lade Akku mit e1*)
      AND   e2     (*Akku := e1 UND e2*)
      ORN   e3     (*Akku := Akku ODER-NICHT e3*)
      ST    a1     (*a1 := Akku*)
```

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Schleifen in AWL

ST-Anweisung

```
FOR zaehler := 1 TO 10 BY 1 DO
   ...
END_FOR
```

AWL-Anweisung

```
            LD 1
            ST zaehler
FOR1:       LD zaehler
            GT 10
            JMPC END_FOR1
            ...
            LD zaehler
            ADD 1
            ST zaehler
            JMP FOR1
END_FOR1:
```

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Operatoren der AWL

Abk. für aktuelles Ergebnis

Operatoren	Operatorgruppe	Erklärung
LD LDN	E	Lädt Operand (negiert) in AE
AND ANDN	W	UND-Verknüpfung des (negierten)
AND(ANDN(		Operanden mit AE
OR ORN	W	ODER-Verknüpfung des (negierten)
OR(ORN(		Operanden mit AE
XOR XORN	W	Exklusiv-ODER-Verknüpfung des
XOR(XORN(		(negierten) Operanden mit AE
ST STN	U	Speichert AE in den Operand
S	U	Setzt Operand auf TRUE, wenn AE=1
R	U	Setzt Operand auf FALSE zurück, wenn AE=1
ADD ADD(	W	Addiert Operand auf AE
SUB SUB(	W	Subtrahiert Operand von AE
MUL MUL(	W	Multipliziert Operand mit AE
DIV DIV(	W	Dividiert AE durch Operand
GT GT(	W	AE größer Operand
GE GE(	W	AE größer gleich Operand
EQ EQ(	W	AE gleich Operand
NE NE(	W	AE ungleich Operand
LE LE(	W	AE kleiner gleich Operand
LT LT(	W	AE kleiner Operand
)	U	Schließen der Klammerebene

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Sprungoperatoren der AWL

Operatoren	Operatorgruppe	Erklärung
JMP	- oder U[a]	(Un-) bedingter Sprung zu einer
JMPC JMPCN	U	Sprungmarke
CAL	-	(Un-)bedingter Aufruf eines
CALC CALCN	U[b]	Funktionsbausteins
RET	- oder U[c]	(Un-) bedingter Rücksprung aus
RETC RETCN	U	Funktion oder Funktionsbaustein
<Fun-Name>	W[b]	Aufruf einer Funktion

[a] implementierungsabhängig

[b] gilt nur für folgenden Operator, nach dem Sprung zum Baustein aber undefiniert.

[c] bei Funktionen: Funktionswert („U“)

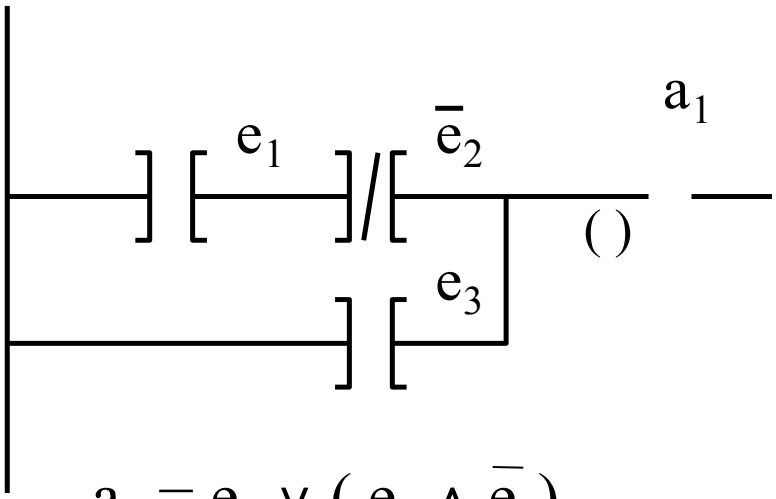
© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Operatorengruppen der AWL

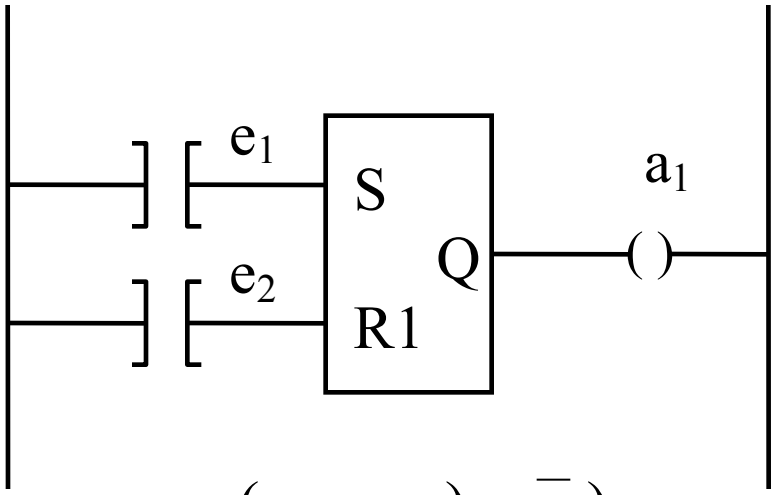
Einfluss der Operator-gruppe auf das AE	Kurzform	Beispieloperatoren
erzeugen	E	LD
weiterverarbeiten	W	GT
unverändert lassen	U	ST; JMPC
undefiniert setzen	-	Unbedingter Bausteinaufruf CAL (die nachfolgende Anweisung muss das AE neu laden, da es nach der FB-Rückkehr keinen eindeutig definierten Wert besitzt)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Übersetzung KOP in AWL



$a_1 = e_3 \vee (e_1 \wedge \bar{e}_2)$



$a_1 = (a_1 \vee e_1) \wedge \bar{e}_2)$

Eine Kontaktplandarstellung wird in ausführbaren Programmcode übersetzt. Nach IEC 61131-3 ergeben sich für die beiden Beispiele die folgenden AWL-Anweisungen:

```
Beispiel 1:  Beispiel 2:  mit RS-Glied ohne RS-Glied
  LD   e1                 LD e1       LD   e1
  ANDN e2                 S  a1       OR   a1
  OR   e3                 LD e2       ANDN e2
  ST   a1                 R  a1       ST   a1
```

Aufgabe

Gegeben ist die boolesche Funktion:

$$a = \overline{\overline{(e_1 \wedge e_2)}} \vee e_3$$

Wie lautet die Realisierung dieser Funktion in:

a) Funktionsbausteinsprache,
b) Kontaktplan,
c) Strukturiertem Text
d) Anweisungsliste?

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kapitel 4: Speicherprogrammierte Steuerungen

4.1 Aufbau und Funktion einer SPS

4.2 Kontaktplan (KOP)

4.3 Funktionsbausteinsprache (FBS)

4.4 Strukturierter Text (ST)

4.5 Anweisungsliste (AWL)

4.6 Ablaufsprache (AS)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Automaten

Man unterscheidet (vgl. Vorlesung Entwurf diskreter Steuerungen):

- sequentielle Automaten wie Zustandsgraphen, Mealy- und Moore-Automaten (nur ein aktiver Zustand)
- parallele Automaten wie Petri-Netze (SIPN) (mehrere gleichzeitig aktive Zustände)

Das zeitliche Verhalten kann in einfachen Zustandsgraphen oder PN nur schlecht beschrieben werden. Zur problemgerechten Beschreibung wurden die Schrittketten (Ablaufketten) entwickelt (DIN 40719). Sie wurden um die Möglichkeit zur Beschreibung paralleler Vorgänge erweitert.

- Ablaufsprache AS (IEC 61131-3)
- Schrittkettenprogrammierung S5-Grafcet oder S7-Graph
- S7 Zustandsprogrammierung in HiGraph

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Ablaufsprache AS

Programmierung von Ablaufsteuerungen mittels Schrittketten in IEC 61131-3. Die Ablaufsprache (engl: SFC Sequential Function Chart) ist den anderen IEC Sprachen übergeordnet. Der oberste Programmablauf wird immer in AS beschrieben.

Der Programmaufbau erfolgt durch *Schritte* und *Transitionen*, die in einer gerichteten Ablaufkette zusammengefügt werden.

In Analogie zu den SIPN (steuerungstechnisch interpretierte Petri-Netze) sind den Transitionen *Schaltbedingungen* und den Schritten (vgl. Stellen) *Aktionen* zugeordnet. Zwei gleiche AS-Elemente können nicht miteinander verbunden werden.

Voraussetzung für das Schalten in einen Schritt ist, dass der vorhergehende Schritt aktiv ist und die Transitionsbedingung erfüllt ist.

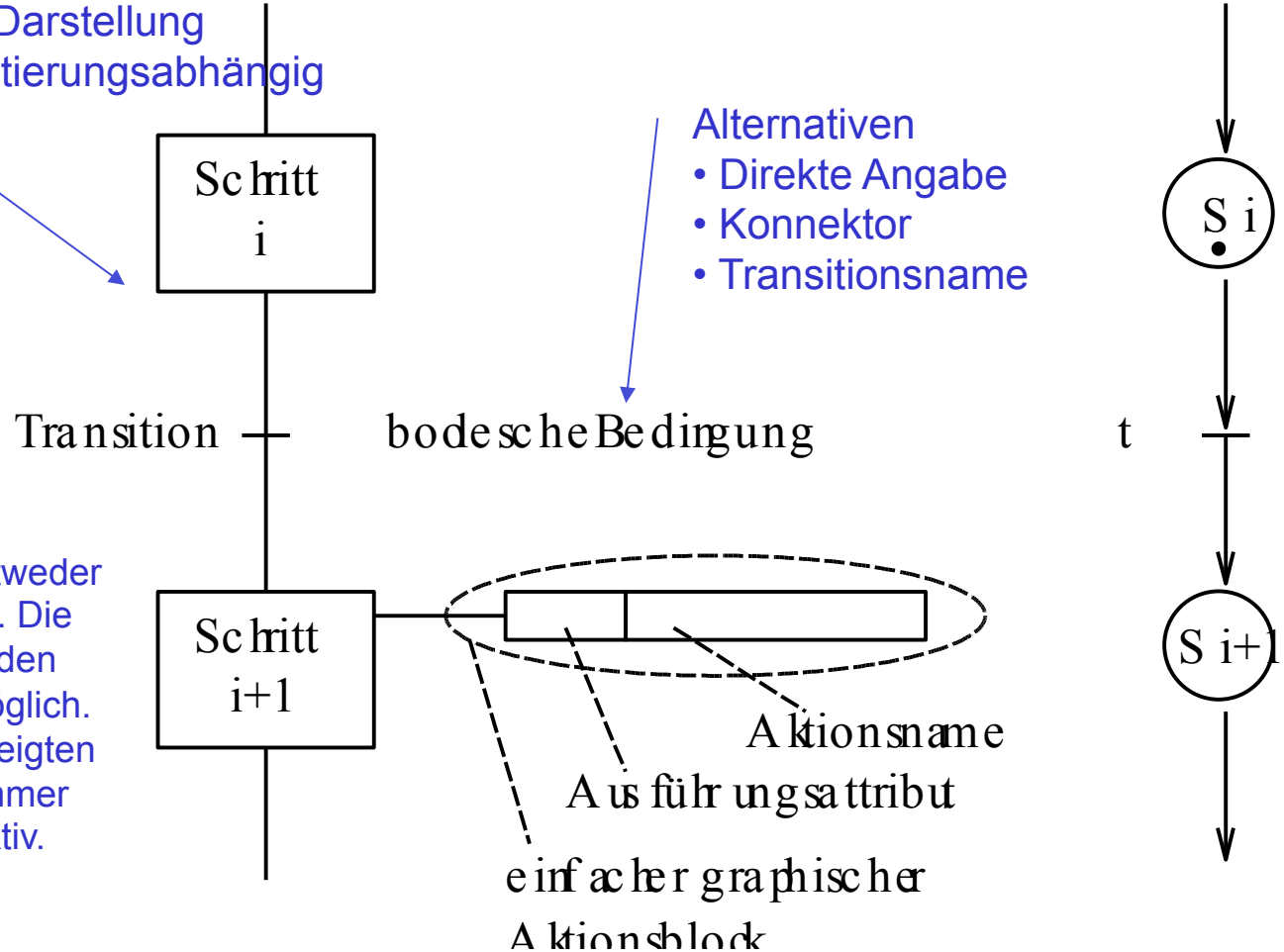
© 2015 B. Wagner - Leibniz Universität Hannover - RTS

AS und Petri-Netz Darstellung

Für AS existiert eine graphische und eine textuelle Darstellungsform.

Graphische Darstellung
ist implementierungsabhängig

Alternativen
- Direkte Angabe
- Konnektor
- Transitionsname



Ein Schritt ist entweder aktiv oder inaktiv. Die Abfrage ist über den Schrittmerker möglich. In einer unverzweigten Schrittkette ist immer nur ein Schritt aktiv.

AS und Petri-Netz Darstellung (bei AS implementierungsabhängig)

Ausführungsattribute

Oder „leer“

N	nicht gespeichert
S	gespeichert
R	rückgesetzt
L	zeitbegrenzt, oder bis Schritt inaktiv
D	zeitverzögert, solange bis Schritt inaktiv
P	Puls
SD	gesetzt (unbedingt), zeitverzögert
DS	zeitverzögert, gesetzt (bedingt)
SL	gesetzt, zeitbegrenzt (unbedingt)

Schritt muss nicht aktiv sein!

Nur gesetzt, wenn Schritt bei Ablauf der Zeit aktiv!

Programmeigene Überwachung über Variablen:

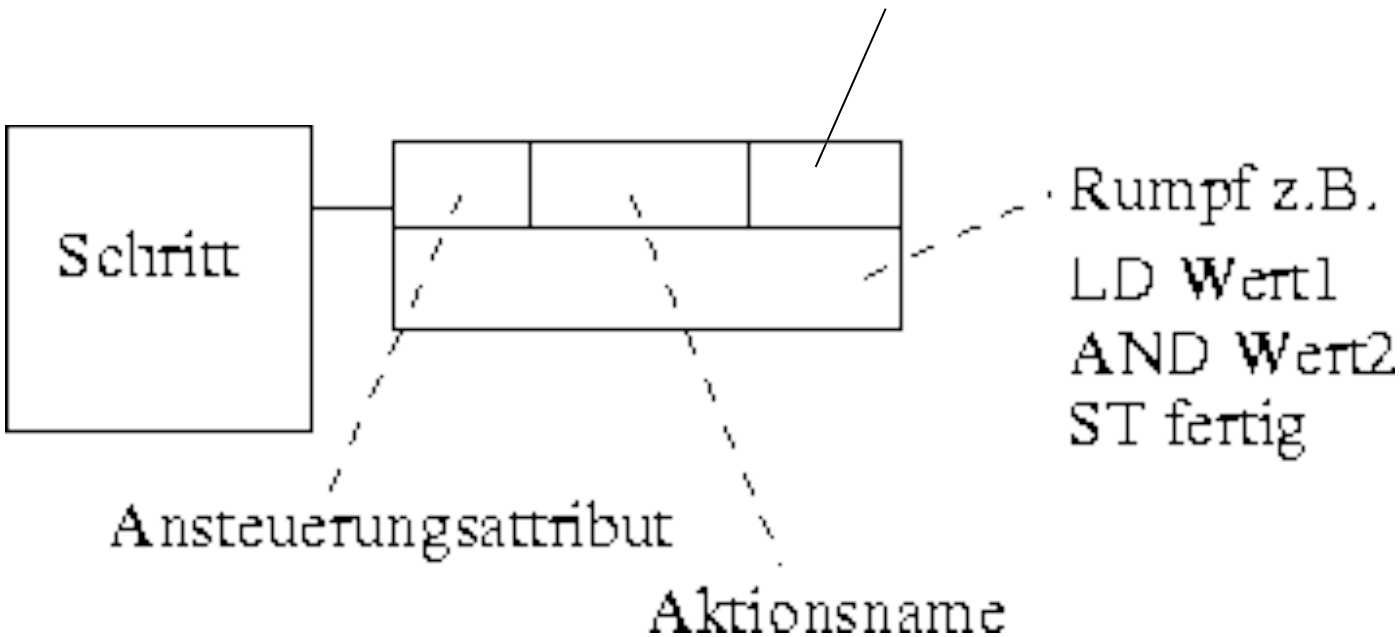
schritt.X ... Schrittmerker (BOOL)

schritt.T ... Zeitdauer seit Aktivierung (TIME)

Ersetzung durch konkreten Namen, des entsprechenden Schritts.

Vollständiger Aktionsblock

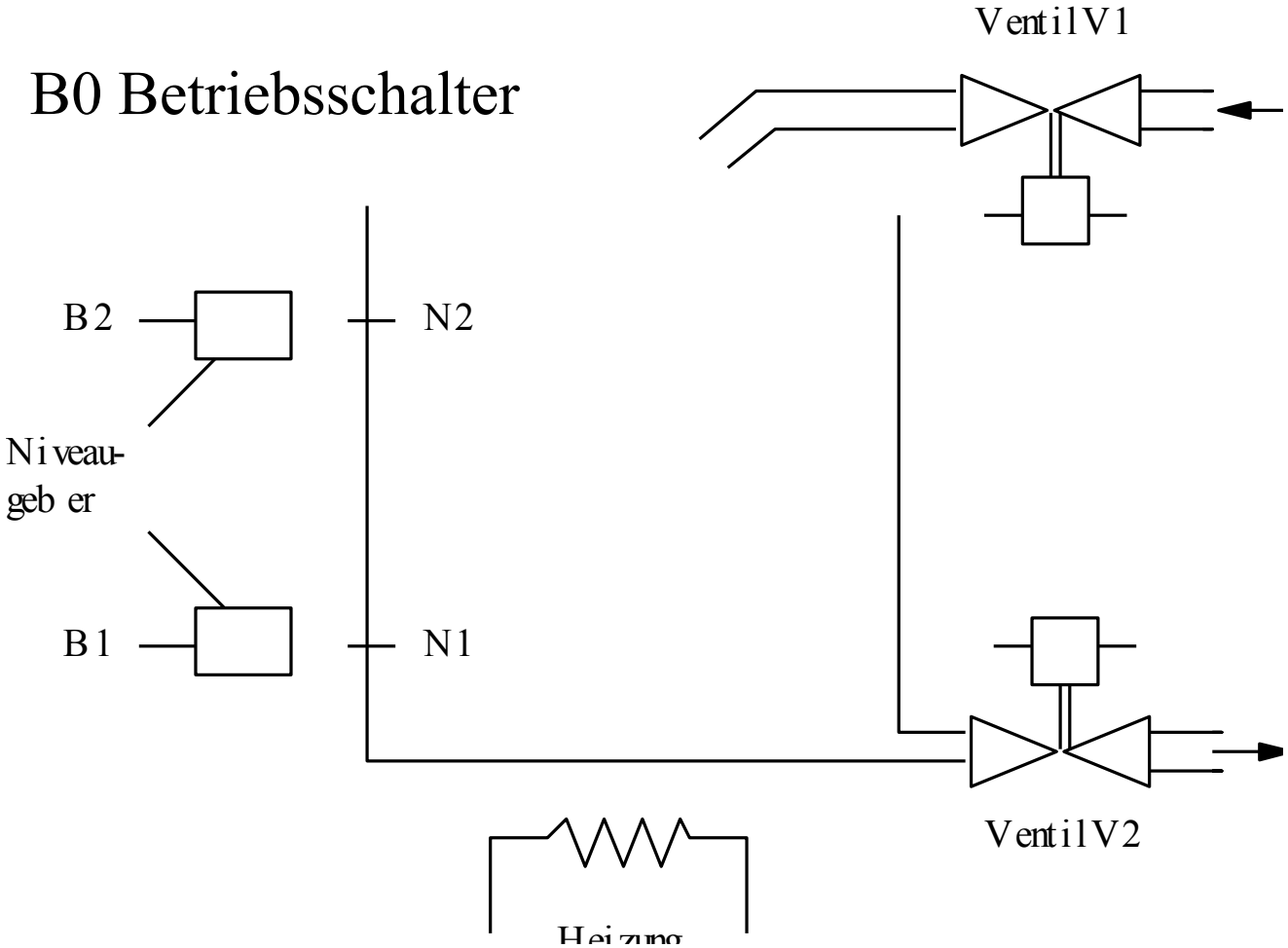
Boolesche Rückkopplungsvariable (optional), z.B.: fertig, die das Ende einer Aktion, eine Zeitüberschreitung oder einen Ausführungsfehler kennzeichnen.



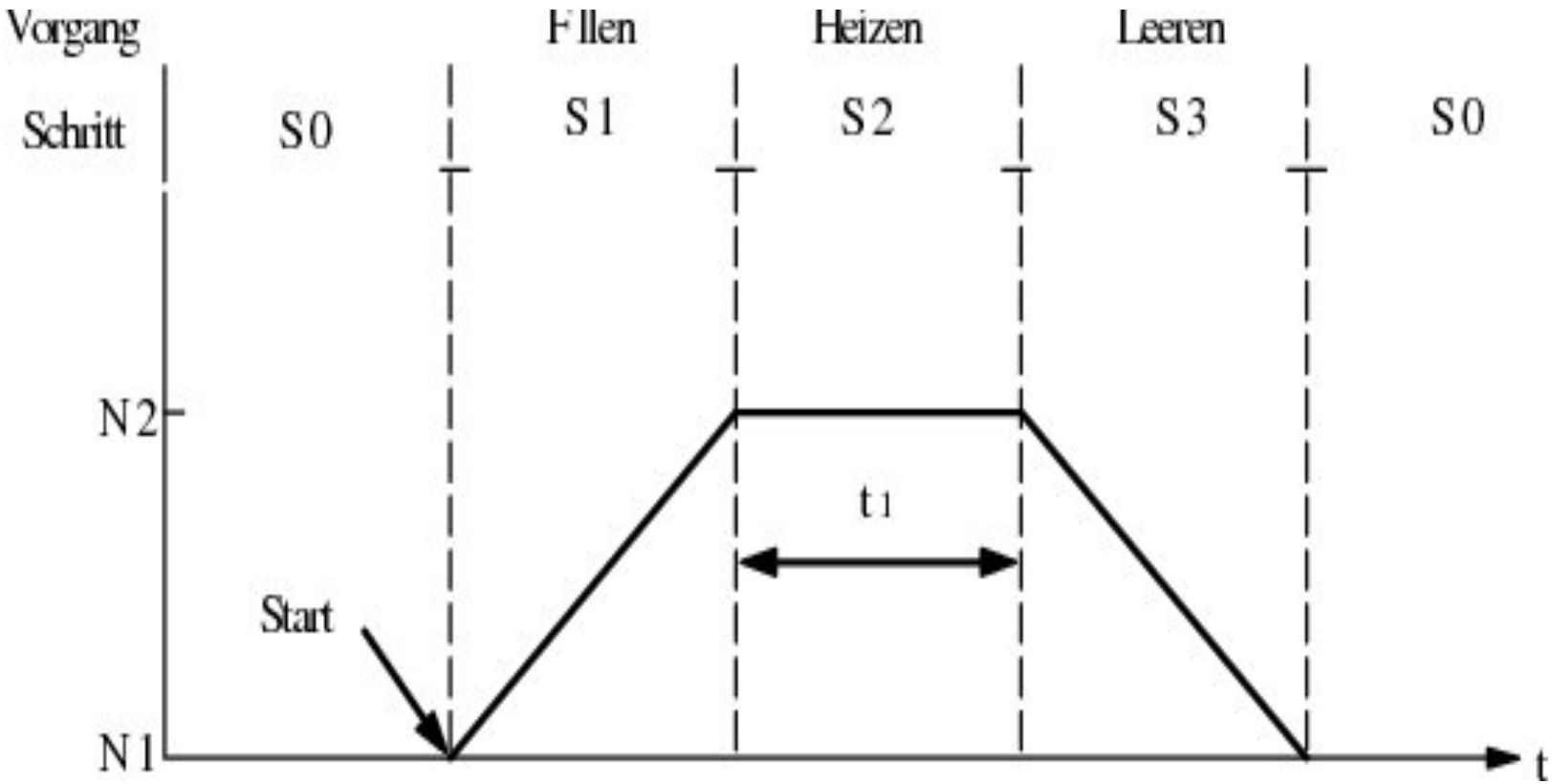
Der Rumpf kann in einer beliebigen Sprache der IEC 61131-3 (ST, FBS, AWL, KOP und AS) angegeben werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

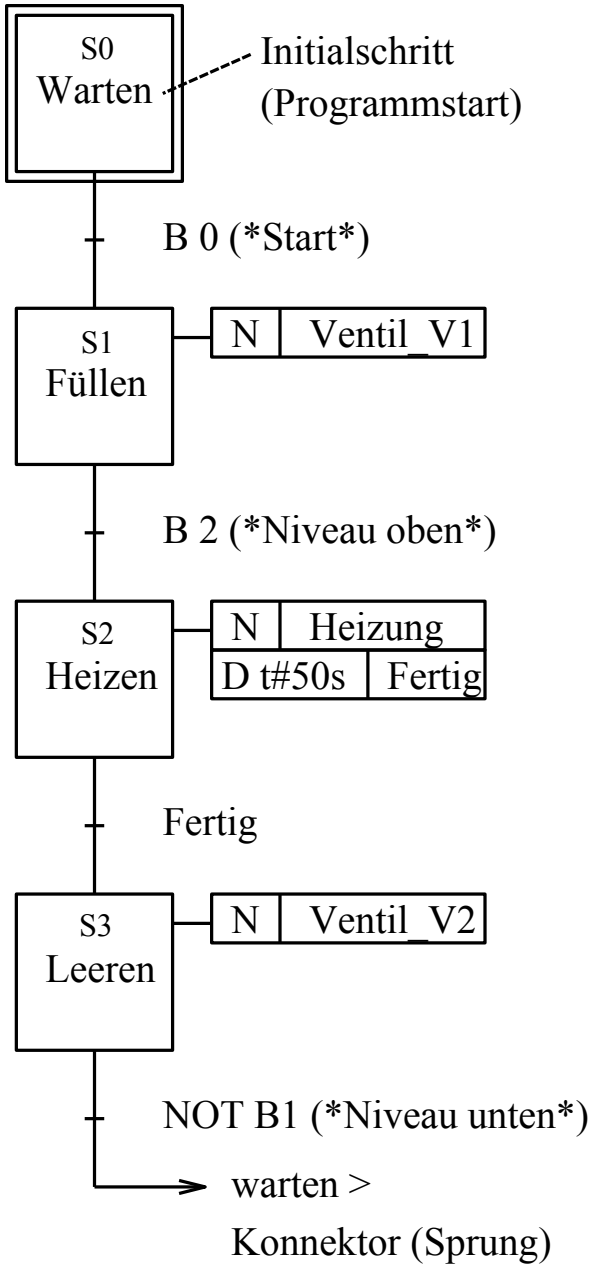
Technologieschema der Abfülleinrichtung



Ablaufdiagramm der Abfülleinrichtung

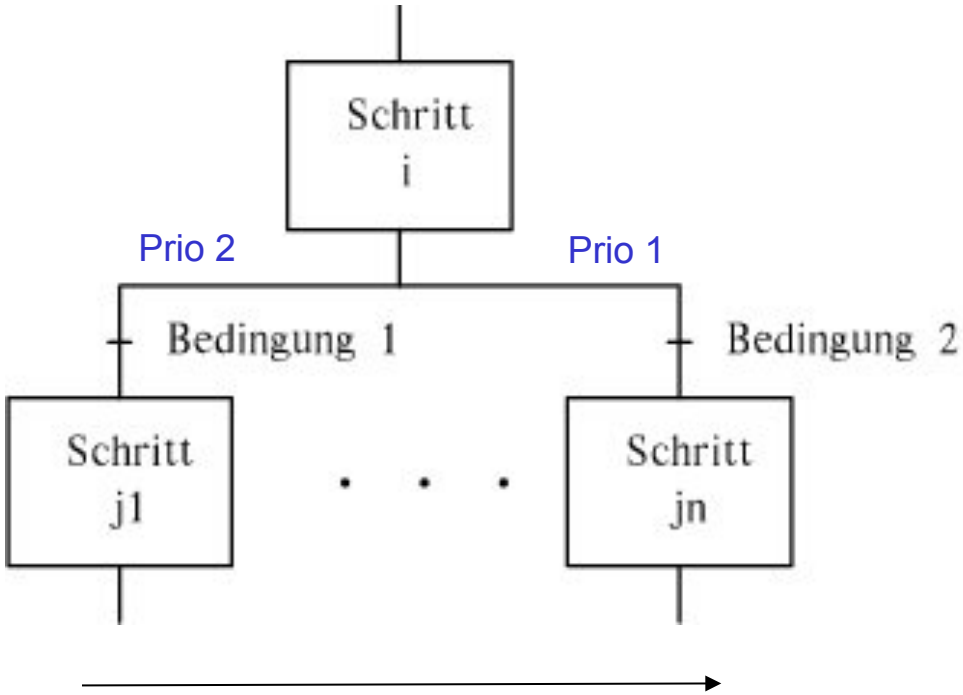


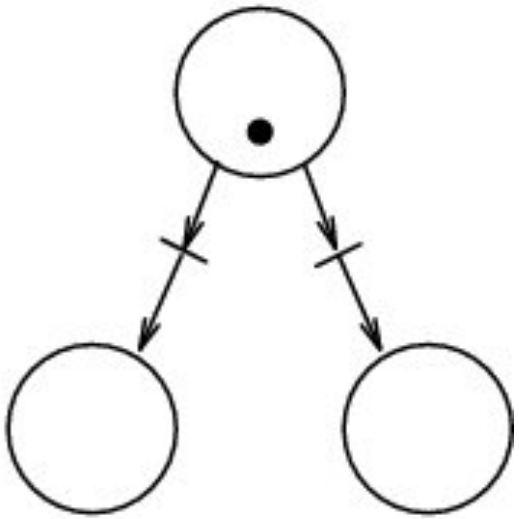
Steuerung der Abfülleinrichtung in AS



Alternativverzweigung

In Ablaufsteuerungen können in Abhängigkeit von Bedingungen alternative Schrittfolgen durchlaufen werden.

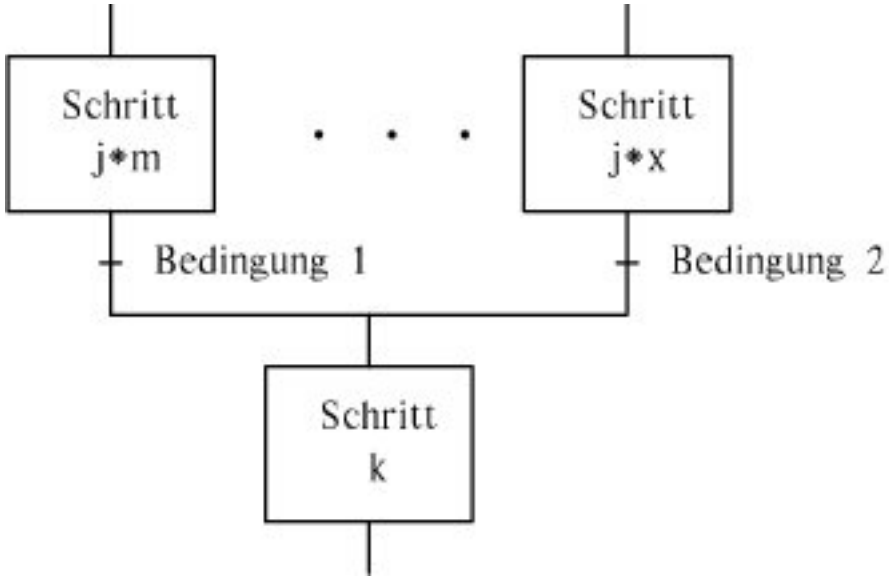


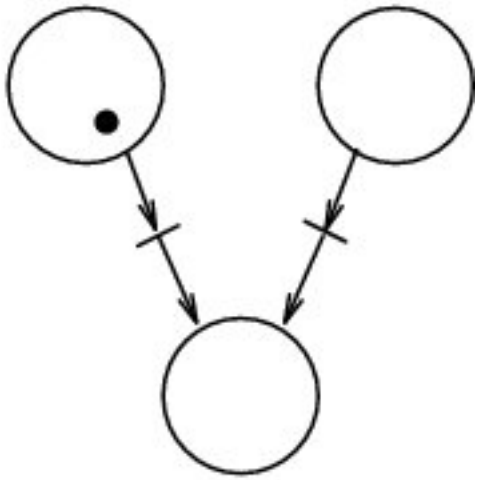


Wenn die Verzweigung mit einem Stern markiert ist, erfolgt die Auswertung von links nach rechts, der Zweig mit der ersten erfüllten Bedingung wird durchlaufen. Mit Prioritäten kann die Reihenfolge der Auswertung verändert werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

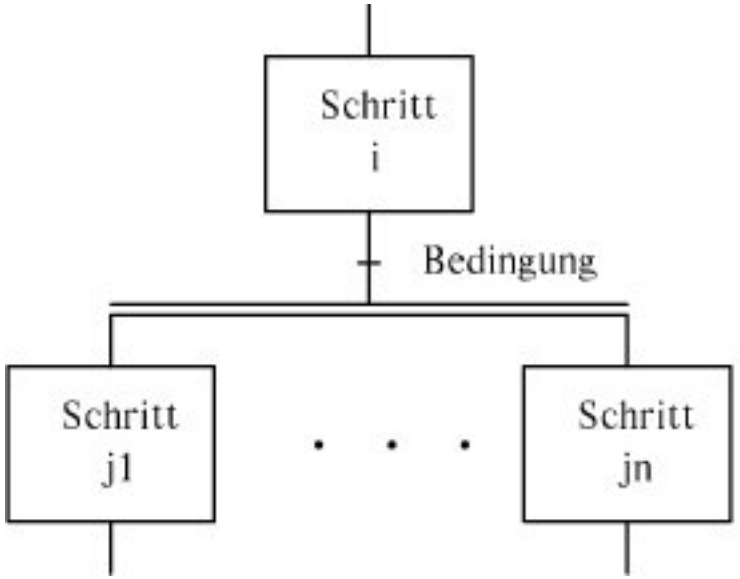
Alternativzusammenführung

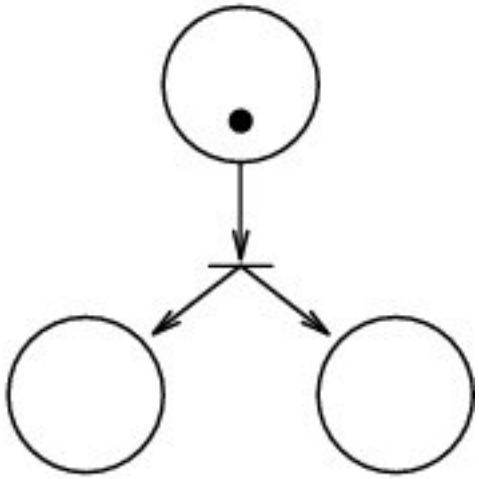




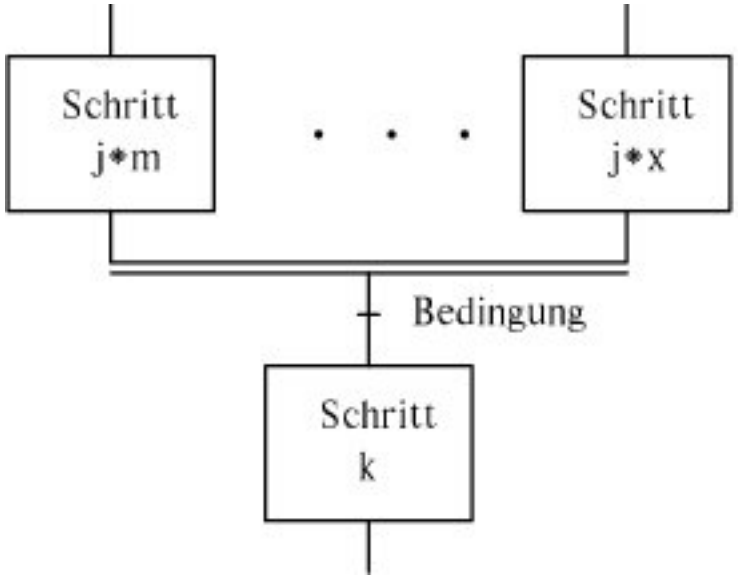
Simultanverzweigung

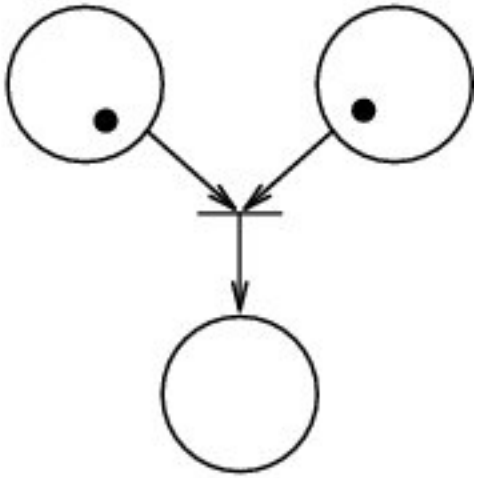
Vorgänge, die zeitgleich ablaufen, können durch parallel zu durchlaufende Schrittketten gesteuert werden.
Wichtige Eigenschaften sind: Verklemmungsfreiheit und Erreichbarkeit.





Simultanzusammenführung





Textuelle Ablaufsprache (1)

Neben der Graphik gibt es auch eine textuelle Form der Ablaufsprache:

Es stehen die folgenden Sprachelemente zur Verfügung:

```
INITIAL_STEP … END_STEP
STEP … END_STEP
TRANSITION … END_TRANSITION
ACTION … END_ACTION
```

Textuelle Ablaufsprache (2)

Das obige Beispiel der Abfülleinrichtung lautet dann:

```
INITIAL_STEP Warten:
   Ventil_V1 := 0;
   Ventil_V2 := 0;
   Heizung   := 0;
   fertig    := 0;
END_STEP

TRANSITION FROM Warten TO Fuellen
   := Start_B0;
END_TRANSITION

STEP Fuellen:
   Ventil_V1 := 1;
END_STEP

TRANSITION FROM Fuellen TO Heizen
   := Niveau_B2;
END_TRANSITION

STEP Heizen:
   Heizung := 1;
   fertig (D, T#25s);
END_STEP
```

```
TRANSITION FROM Heizen TO Leeren
   := fertig;
END_TRANSITION

STEP Leeren:
   Ventil_V2 := 1;
END_STEP

TRANSITION FROM Leeren TO Warten
   := NOT Niveau_B1;
END_TRANSITION

(* komplexere Aktionsbloecke werden in
   einem eigenen Abschnitt beschrieben:

   ACTION actionname
      ...
   END_ACTION

   Aufruf mit:
   actionname(Ausführungsattribut);
*)
```

Aufgabe

Entwickeln Sie eine alternative Lösung in Ablaufsprache zur Steuerung des Werktores aus der Einführung.

Versuchen Sie alternative und parallele Programmzweige zu implementieren, obwohl dies nicht zwingend erforderlich ist. Überprüfen Sie Ihre Lösung unter Verwendung eines SPS-Simulationsprogramms.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Zusammenfassung

Es wurden die textuellen und graphischen Sprachmittel und die Vorgehensweise zur Programmierung einer SPS entsprechend der Norm IEC 61131-3 vorgestellt:

- Kontaktplan
- Anweisungsliste
- Funktionsbausteinplan
- Strukturierter Text
- Ablaufsprache

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kapitel 5: Eingebettete Computersysteme

5.1 Einleitung

5.2 Begriffsbestimmung

5.3 Architektur von Mikrocomputersystemen (Unterbrechungsbehandlung und Speicherhierarchien)

5.4 Mikrocontroller

5.5 Ein-/Ausgabeschnittstellen

5.6 Embedded-PC

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Einleitung

Die Echtzeiteigenschaften eines Rechnersystems werden sowohl von der Hardware als auch von der Software bestimmt.

Alle Ebenen eines Gesamtsystems müssen Echtzeiteigenschaften aufweisen. Dazu gehören insbesondere:

- Rechnerhardware (Mikroprozessor, Speicherzugriff, Interruptbehandlung, System- und Erweiterungsbusse)
- Schnittstellen zu Sensoren und Aktoren
- Daten- bzw. Feldbusse und Netzwerke
- Betriebssystem inkl. Treiber und Bibliotheken sowie Anwendungsprogramme

Liegt eine Hardware vor, die einem RTOS nicht die vollständige Kontrolle über alle kritischen Komponenten erlaubt, ist das Gesamtsystem nicht echtzeitfähig. Beispielsweise können System Management Interrupts (SMI) aktueller PC-Hardware, wie Einschalten Lüfter, nicht durch das Betriebssystem kontrolliert werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kapitel 5: Eingebettete Computersysteme

5.1 Einleitung

5.2 Begriffsbestimmung

5.3 Architektur von Mikrocomputersystemen (Unterbrechungsbehandlung und Speicherhierarchien)

5.4 Mikrocontroller

5.5 Ein-/Ausgabeschnittstellen

5.6 Embedded-PC

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Begriffsbestimmung (1)

- Mikroprozessor: Zentraleinheit (Central Processing Unit, CPU) eines Datenverarbeitungssystems, bestehend aus Prozessorkern, Schnittstellen zur Außenwelt, Cache-Speicher und virtueller Speicherverwaltung meist auf einem Chip. Aufgabe einer CPU ist die Ausführung eines Programms, d.h. die Bearbeitung einer Abfolge von Befehlen.
- Mikroprozessorsystem: Technisches System, welches einen Mikroprozessor enthält. Neben einem Computer wäre dies auch eine Kaffeemaschine, wenn sie von einem Mikroprozessor gesteuert wird.
- Mikrocomputer: Ein Rechner, dessen Zentraleinheit aus einem oder mehreren Mikroprozessoren besteht und der außerdem über Speicher, Ein-/Ausgabeschnittstellen sowie einem Verbindungssystem verfügt.
- Mikrocomputersystem: Ein Mikrocomputer mit an die Ein-/Ausgabeschnittstellen angeschlossene Peripherie, z.B. Tastatur, Maus, Drucker.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Begriffsbestimmung (2)

- Mikrocontroller: Mikrocomputer auf einem Chip. Ziel ist die Lösung einer Steuerungs- oder Kommunikationsaufgabe mit möglichst wenigen Bausteinen, so dass Prozessorkern, Speicher und Ein-/Ausgabeschnitt-stellen speziell auf die Lösung solcher Aufgaben zugeschnitten sind.
- Einplatinencomputer: Kompakter Mikrocomputer (meist mit Mikrocontroller) auf einer Leiterplatte, wird oft für Aufgaben in der Mess-, Steuerungs- und Regelungstechnik eingesetzt. Preiswerter als eine SPS und daher wirtschaftlicher in Bereichen, in denen eine SPS überdimensioniert wäre.
- Industrie-PC: PC-kompatibler Mikrocomputer, der für Aufgaben im industriellen Bereich eingesetzt wird, z.B. Industrieautomation oder Test- und Prüfstände. Gegenüber normalen Büro-PCs existieren besondere Anforderungen, z.B. Robustheit oder Ausfallsicherheit.
- Embedded-PC: Modular aufgebauter, kompakter Industrie-PC, dessen Rechnerarchitektur sich am Standard-PC orientiert. Nicht benötigte Komponenten können entfallen, z.B. die Festplatte.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kapitel 5: Eingebettete Computersysteme

5.1 Einleitung

5.2 Begriffsbestimmung

5.3 Architektur von Mikrocomputersystemen (Unterbrechungsbehandlung und Speicherhierarchien)

5.4 Mikrocontroller

5.5 Ein-/Ausgabeschnittstellen

5.6 Embedded-PC

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Unterbrechungsbehandlung

Ein wichtiges Merkmal für die Echtzeitfähigkeit eines Mikroprozessors oder -controllers ist die Behandlung interner und externer Ereignisse.

Interne Ereignisse:

- Exception, plötzlich auftretende Fehlersituation im Prozessorkern, z.B. illegaler Befehl oder Division durch 0
- Software Interrupt, Unterbrechungswunsch des laufenden Programms, z.B. um eine Betriebssystemfunktion aufzurufen.

Externe Ereignisse werden durch Unterbrechungswünsche externer Systemkomponenten ausgelöst (Hardware Interrupt), um z.B. das Vorhandensein neuer Daten oder das Verstreichen eines Zeitintervalls anzuzeigen.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Reaktionen auf Ereignisse

Drei typische Reaktionsmöglichkeiten auf interne und externe Ereignisse:

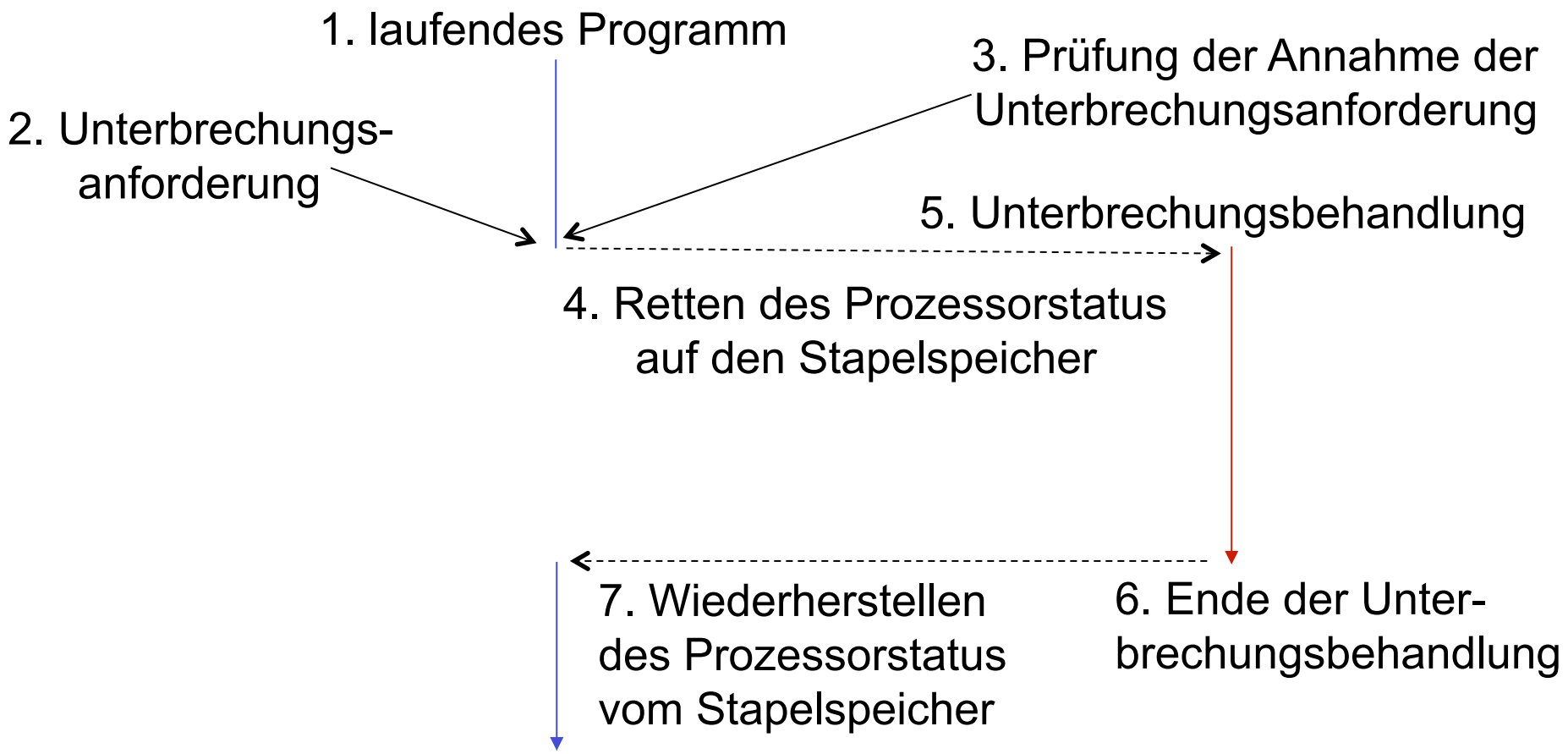
- Abbruch der aktuell durchgeführten Aufgabe. Beim Auftreten eines ernsten Fehlers wird das laufende Programm in Frage gestellt, die Weiterführung ist nicht sinnvoll. Es wird zu einem Fehlerbehandlungsprogramm verzweigt.
- Unterbrechung der aktuell durchgeführten Aufgabe und Wechsel zu einer neuen Aufgabe, wenn das Ereignis eine wichtigere Aufgabe anzeigt. Die unterbrochene Aufgabe wird später fortgesetzt.
- Fortsetzung der aktuell durchgeführten Aufgabe, wenn sie wichtiger ist als die durch das Ereignis angezeigte Aufgabe. Die verschobene Abarbeitung wird später nachgeholt.

Ein Unterbrechungswerk (Interrupt Unit) in der Steuereinheit des Mikroprozessors koordiniert diese Aktivitäten.

Die Behandlung einer Unterbrechung (Interrupt) erfolgt durch ein Unterbrechungsprogramm (Interrupt Service Routine, ISR).

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

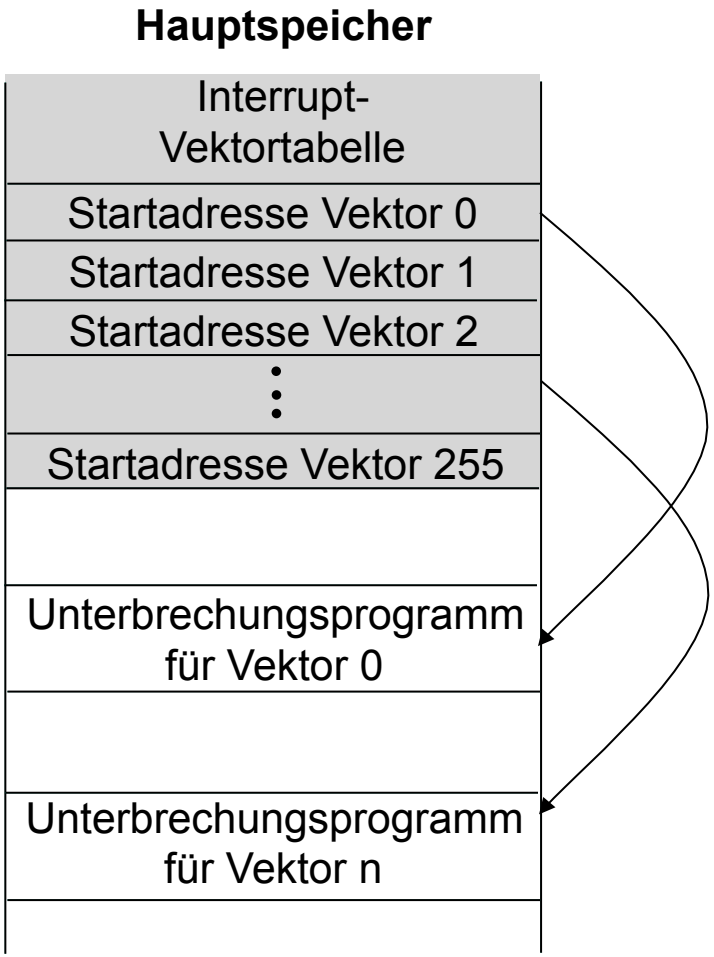
Ablauf einer Unterbrechung



Allgemein: Laufende Unterbrechungsbehandlungen können ihrerseits durch neue, wichtigere Unterbrechungsanforderungen unterbrochen werden!

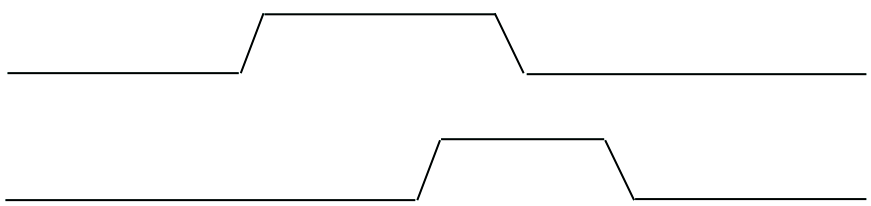
Eigenschaften der Unterbrechungsbehandlung

- Maskierbarkeit: Bestimmte Unterbrechungsquellen können kurzfristig gesperrt (maskiert) werden. Ein auftretendes Ereignis wird für die Dauer der Maskierung zurückgestellt.
- Vektorisierung: Jeder Unterbrechungsquelle wird eine Kennzahl zugeordnet. Dieser Interrupt-Vektor identifiziert eindeutig die Quelle, meist gibt es 256 davon. Über die Interrupt-Vektortabelle kann die Startadresse der zugehörige Behandlungsroutine ermittelt werden.

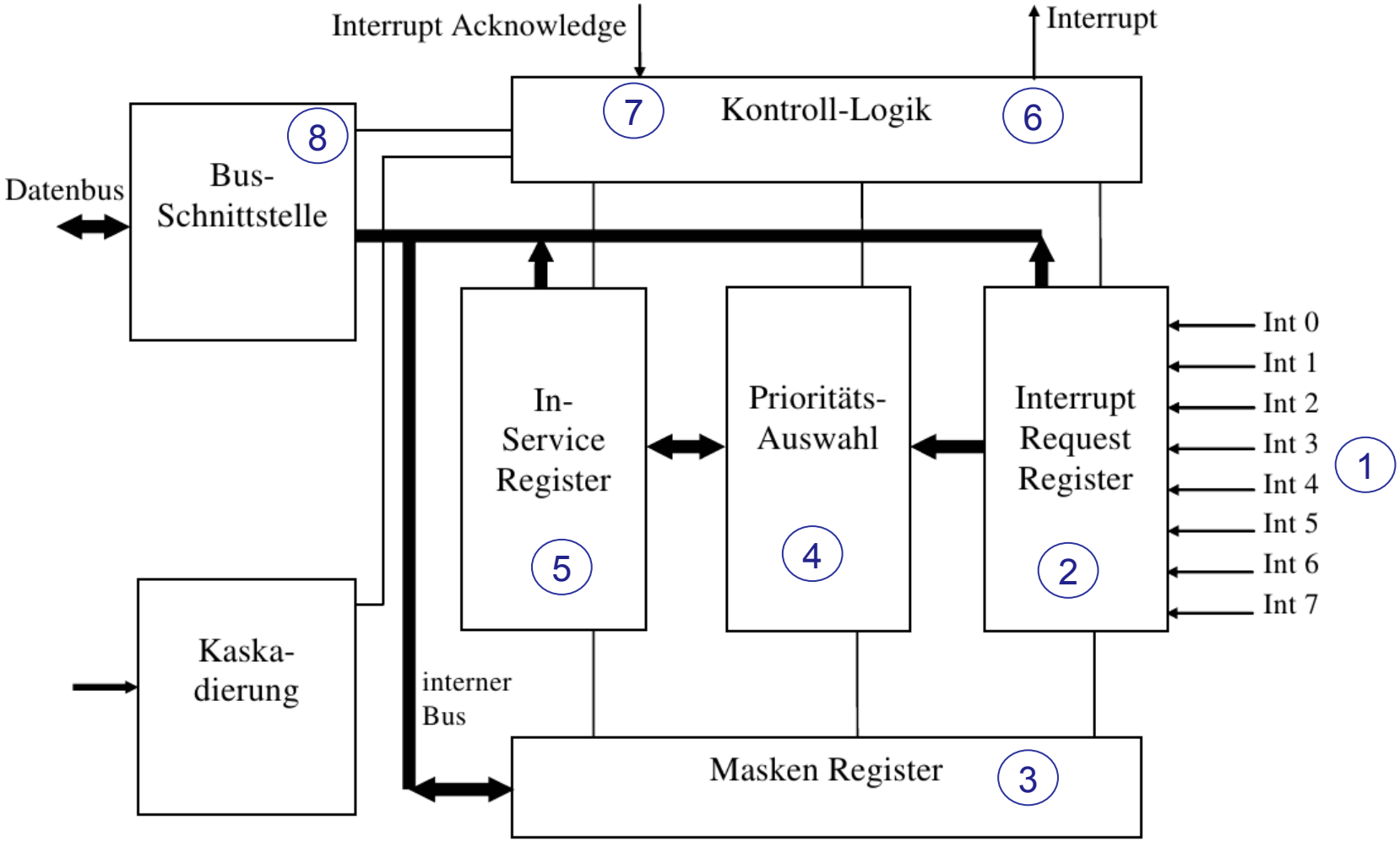


Interrupt-Prioritäten

- Verschiedenen Ereignissen können unterschiedliche Prioritäten zugeordnet werden, d.h. es gibt eine Ordnung nach Wichtigkeit.
- Im Allgemeinen wird das Fixed-Priority-Preemptive Schema benutzt. Die Prioritäten der Ereignisse sind fest und ändern sich nicht zur Laufzeit. Ereignisse höherer Priorität können Ereignisse unterbrechen, die weniger wichtig sind.
- Eine zentrale Prioritätensteuerung ermittelt die Anforderung höchster Priorität und gibt diese an den Mikroprozessor weiter. Eine freie Vergabe der Prioritäten an die Quellen ist möglich.
- Der Mikroprozessor muss eine Unterbrechungsanforderung quittieren. Erst dann wird der Eingangspegel zurückgesetzt.

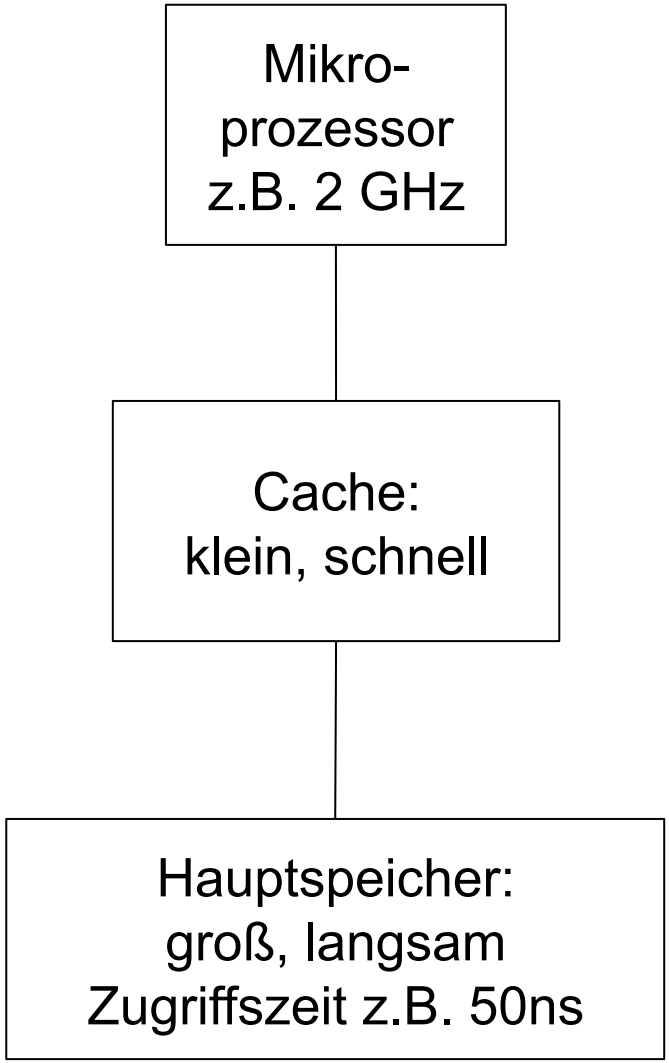


Interrupt-Controller 8259 von Intel



Ablauf: Int x → IRR → maskiert? → Prio.-Auswahl → Vergleich mit In-Service Reg. → Kontrolllogik ggf. neuer Int. → IAck → Vektor auf Datenbus

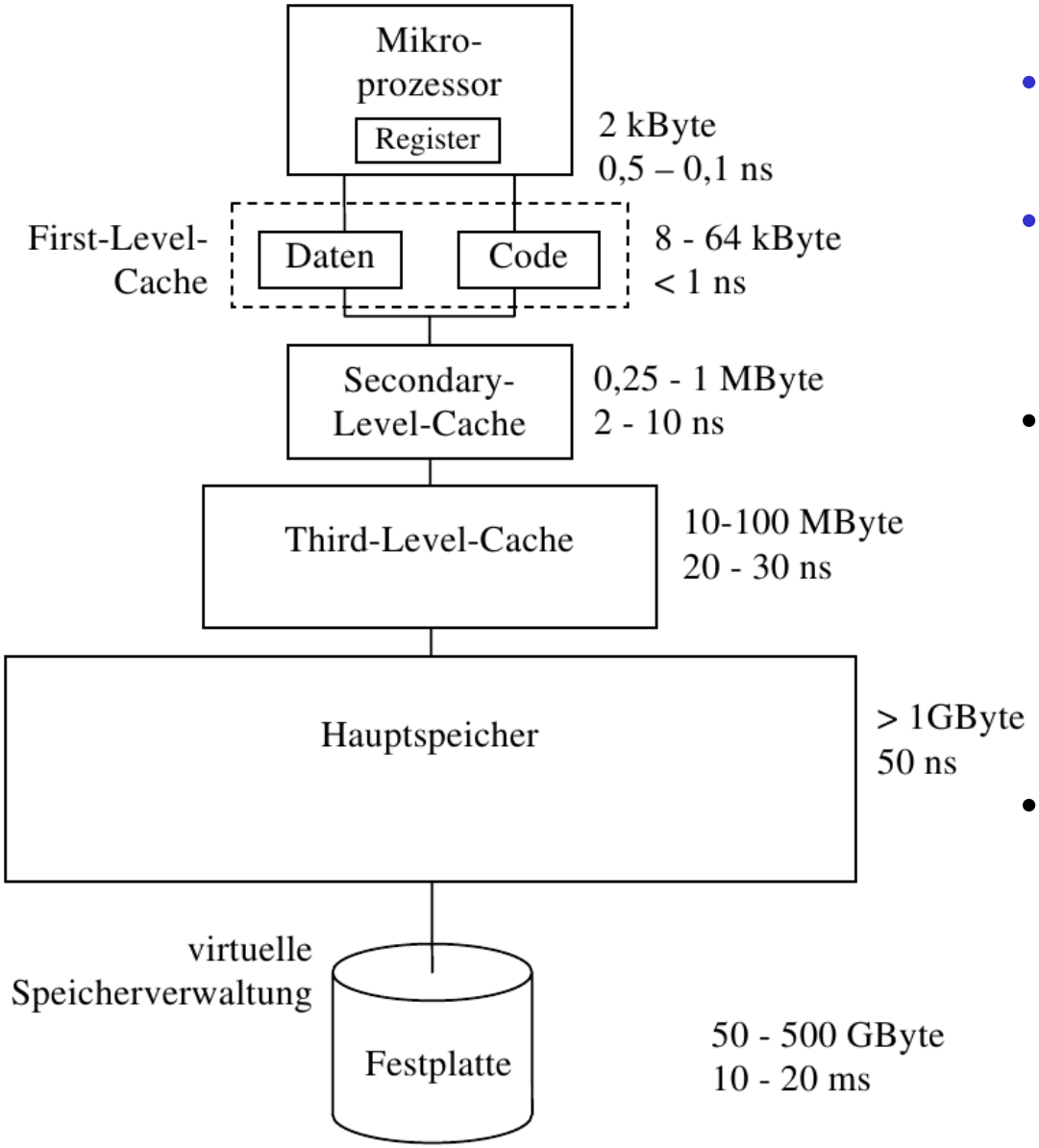
Speicherhierarchien



- Prozessor wird durch Hauptspeicherzugriff gebremst (im Bsp. 100 Taktzyklen)
- Allgemein: Geschwindigkeit und Größe von Speicher umgekehrt proportional
- Abhilfe: Cache (Versteck)
- Programme besitzen meist Lokalitätseigenschaften
- Räumliche Lokalität: als nächstes ausgeführte Befehle befinden sich in benachbarten Speicherbereichen
- Zeitliche Lokalität: Auf ein zugegriffenes Datum wird wahrscheinlich später wieder zugegriffen
- Cache-Hit: Speicherwort im Cache, anderenfalls spricht man von Cache-Miss

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Mehrstufige Speicherhierarchie



- Harvard-Architektur: getrennte Instruktions- und Daten-Caches
- Mehrere Cache-Ebenen mildern die Folgen von Cache-Misses ab.
- Erweiterte Betrachtung mit Prozessorregistern (schnellstmöglicher Zugriff auf wenige Daten) und Festplatte (virtuelle Speicherverwaltung über eine Memory Management Unit, MMU)
- Virtueller Speicher inkl. Festplatte größer als physikalischer Hauptspeicher, Auslagerung von nicht benötigten Speicherbereichen möglich.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Speicherhierarchien und Echtzeit

Ein-/Auslagerungsvorgänge mit Festplattenzugriff sind unter Echtzeitaspekten problematisch, da unkalkulierbare Wartezeiten auftreten können. Abhilfe: Entweder

- Kompletter Verzicht auf Auslagerungsspeicher

oder

- Verhinderung der Auslagerung einzelner kritischer Aufgaben (z.B. unter Linux/Xenomai mittels Systemaufruf `mlockall`).

Caches können die Abschätzung der WCET erschweren und Einfluss auf den Jitter einzelner Aufgaben haben, z.B. wenn eine periodische Berechnung 99 Mal vollständig aus dem Cache geladen wird, aber beim 100. Mal einen Cache-Miss erzeugt.

Kapitel 5: Eingebettete Computersysteme

5.1 Einleitung

5.2 Begriffsbestimmung

5.3 Architektur von Mikrocomputersystemen (Unterbrechungsbehandlung und Speicherhierarchien)

5.4 Mikrocontroller

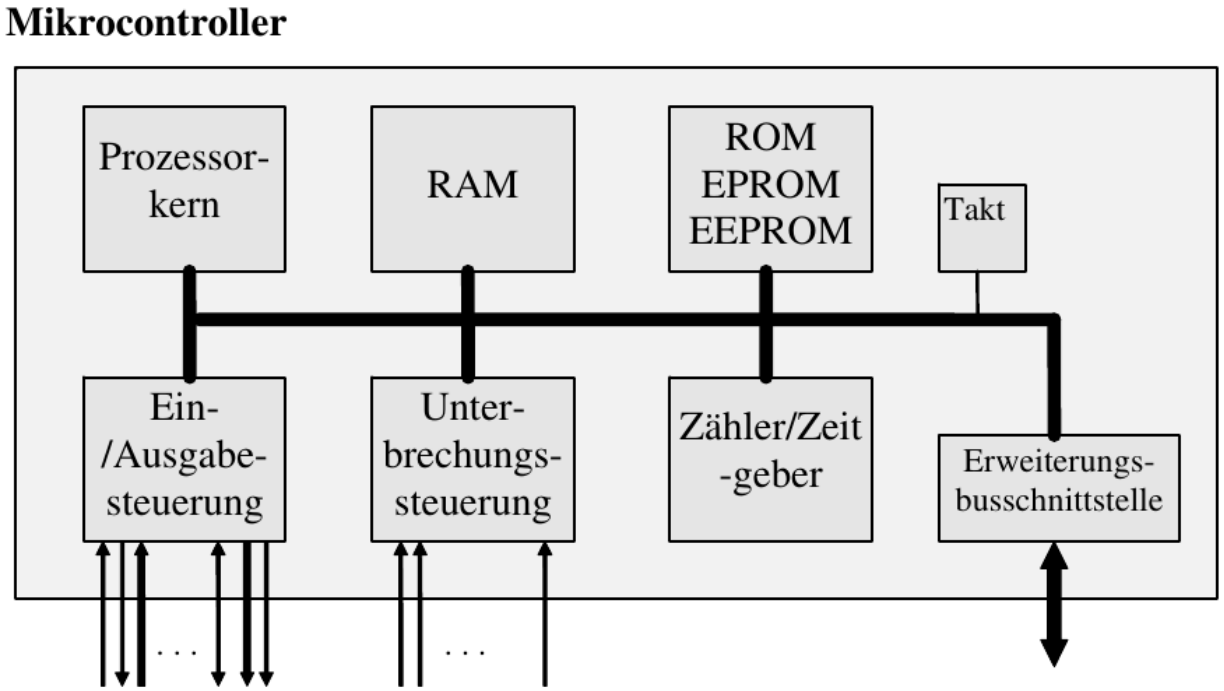
5.5 Ein-/Ausgabeschnittstellen

5.6 Embedded-PC

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Mikrocontroller

Ein-Chip-Mikrocomputer mit speziell für Steuerungs- oder Kommunikationsaufgaben zugeschnittener Peripherie:



Zusätzlich sind nur wenige weitere Komponenten nötig, wie eine Stromversorgung und ein Quarzbaustein. Im Vordergrund steht der kostengünstige Einsatz von Rechen- und Steuerleistung.

Einsatz von Mikrocontrollern

Meist wird eine Anwendungen einmal programmiert, die dann für gesamte Lebensdauer auf dem Mikrocontroller ausgeführt wird. Anwendungsfelder sind z.B.:

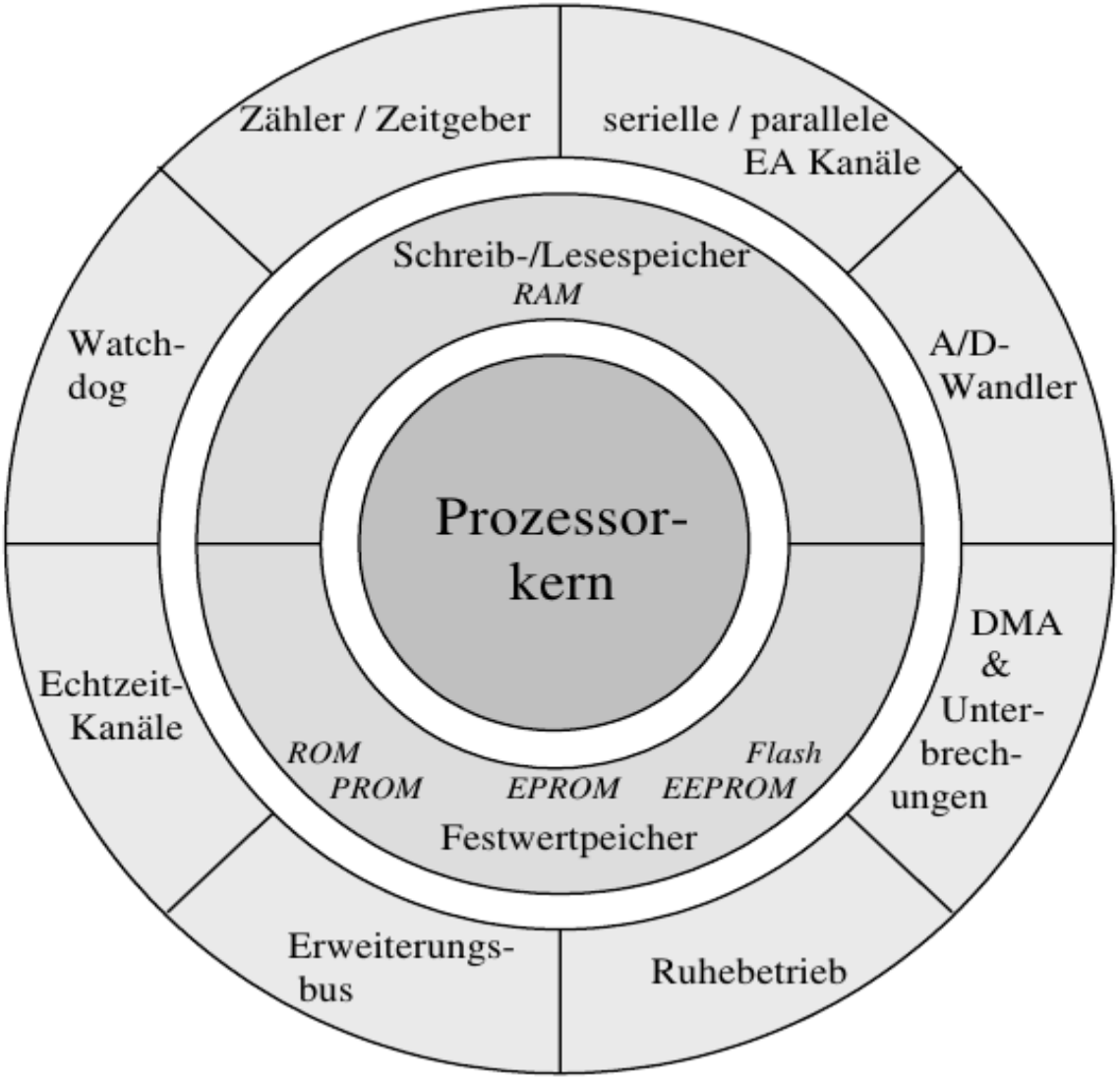
- Kfz-Technik (ABS und ESP)
- Medizintechnik
- Fabrikautomation, Industrieroboter

Mikrocontroller sind häufige und wichtige Bestandteile von Echtzeitsystemen!

Mikrocontroller sind aus herkömmlichen Mikroprozessorarchitekturen abgeleitet, die je nach Aufgabengebiet Spezialisierungen erfahren haben, und werden auch dann noch weitergeführt, wenn der entsprechende Mikroprozessor nicht mehr produziert wird.

Beispiel: Rockwell R6511 Mikrocontroller, basierend auf dem 6502 (8-Bit-Mikroprozessor des Commodore VC-20, Atari 800 XL und Apple I und II).

Schalenmodell eines Mikrocontrollers



- Der Prozessorkern ist das Herzstück eines Mikrocontrollers.
- Um den Kern gruppieren sich die Speichereinheiten.
- An der Außenschale befinden sich die Peripherieeinheiten.

Prozessorkern eines Mikrocontrollers

Der Prozessorkern eines Mikrocontrollers unterscheidet sich prinzipiell nicht von dem eines Mikroprozessors, fällt jedoch häufig einfacher aus, denn:

- die Kosten sind entscheidend
- reine Rechenleistung spielt nicht die dominierende Rolle
- Stromverbrauch und Echtzeitverhalten stehen im Vordergrund

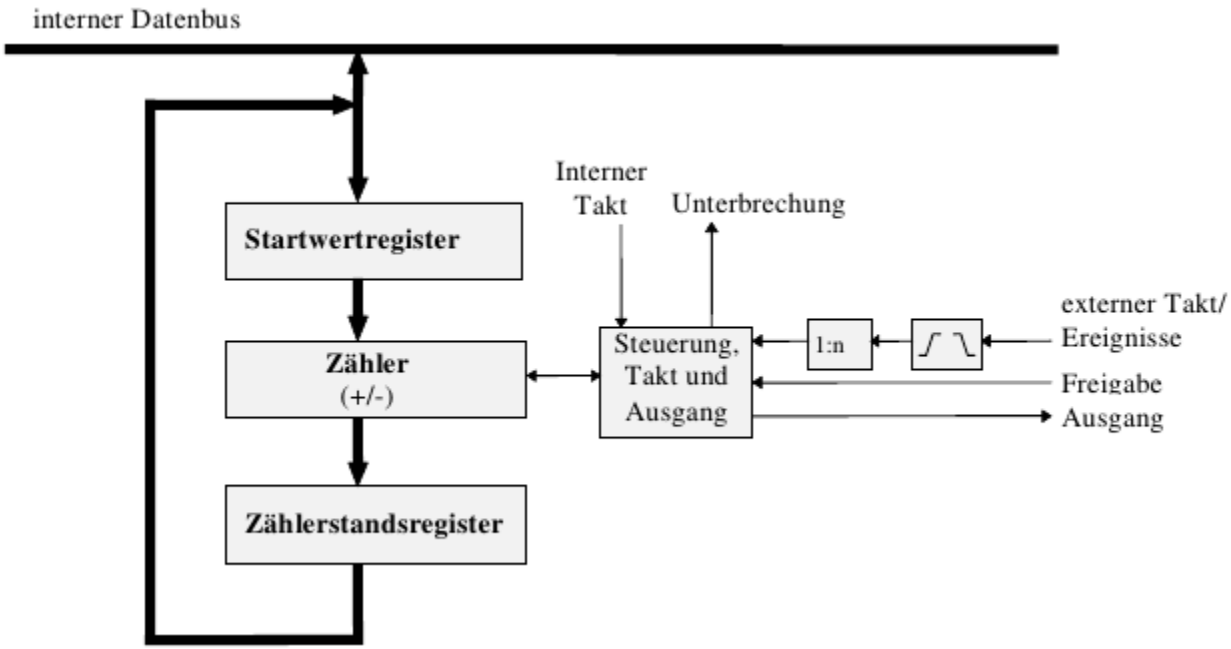
Durch Vereinfachung bzw. völligen Verzicht von Prozessorpipelines, Caches und virtueller Speicherverwaltung wird ein verbessertes Echtzeitverhalten erzielt.

Zum Teil findet man ein Scratch-Memory, eine Art Cache, dessen Belegung nicht dynamisch vom Prozessor, sondern statisch vom Compiler bzw. Anwender bestimmt wird.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Zähler und Zeitgeber (Timer)

Für den Echtzeiteinsatz sind Zähler und Timer wichtige Komponenten. Mikrocontroller haben häufig mehrere unabhängige Zähler und Timer.



- Zähler haben meist eine Breite von 16, 24 oder 32 Bit und können auf- oder abwärts zählen.
- Ein Vorteiler kann die Frequenz der Zählschritte reduzieren.
- Beim Erreichen eines Wertes, z.B. 0, kann ein Interrupt ausgelöst werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Watchdogs

Watchdogs („Wachhunde“) sind im Echtzeitbereich wichtige Komponenten zur Überwachung von Programmaktivitäten eines Mikrocontrollers, die darauf abgerichtet sind, Verklemmungen oder Abstürze zu erkennen und darauf zu reagieren.

Ein Programm muss in regelmäßigen Abständen ein „Lebenszeichen“ an den Watchdog senden. Anderenfalls geht der Watchdog von einem abnormalen Zustand des Programms aus und leitet Gegenmaßnahmen ein, z.B. Rücksetzen des Mikrocontrollers oder Neustart des Programms.

Ein Watchdog ist ein spezieller Timer, der einen Alarm auslöst, wenn er nicht regelmäßig zurückgesetzt wird. Die Zeitspanne ist programmierbar (meist zwischen wenigen Millisekunden und einigen Sekunden).

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Serielle und parallele Ein-/Ausgabekanäle

- IO-Ports sind die grundlegenden digitalen Schnittstellen eines Mikrocontrollers.
- Parallele IO-Ports ermöglichen das gleichzeitige Lesen bzw. Schreiben mehrerer digitaler Signale und können einzelbitweise oder in Bitgruppen als Ein- oder Ausgang konfiguriert werden.
- Serielle IO-Ports dienen der Kommunikation zwischen Mikrocontroller und Peripherie unter Verwendung möglichst weniger Leitungen.
- Heutzutage häufig anzutreffen sind GPIOs (General Purpose Input Output). GPIOs sind vielfältig verwendbar, teilweise existieren auf modernen Einplatinencomputern mehrere Dutzend davon. Normalerweise kann jeder GPIO einen Interrupt auslösen, z.B. bei steigender oder fallender Signalflanke.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kapitel 5: Eingebettete Computersysteme

5.1 Einleitung

5.2 Begriffsbestimmung

5.3 Architektur von Mikrocomputersystemen (Unterbrechungsbehandlung und Speicherhierarchien)

5.4 Mikrocontroller

5.5 Ein-/Ausgabeschnittstellen

5.6 Embedded-PC

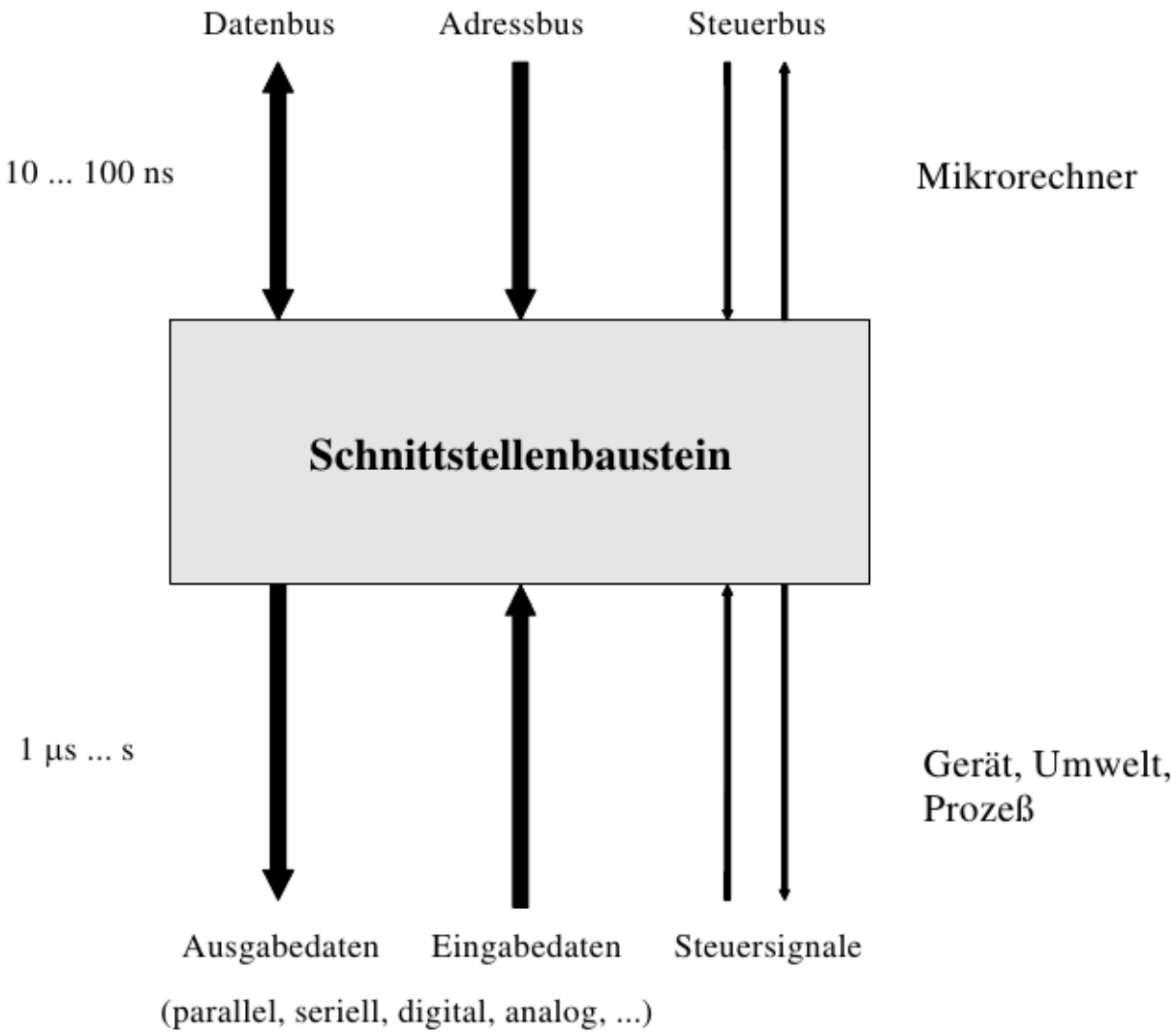
Schnittstellen

Schnittstellen zur Peripherie haben die Aufgabe, ein Mikrocomputersystem mit der Umwelt zu verbinden. Dies geschieht über Schnittstellenbausteine (Interface Controller), die folgende Aufgaben haben:

- Pufferung von Ein- und Ausgabedaten wegen unterschiedlicher Geschwindigkeiten von Mikrorechner und Peripheriekomponenten
- Umsetzung von Daten (z.B. digital/analog oder seriell/parallel)
- Erzeugung von Steuer- und Handshake-Signalen: Peripheriekomponenten und Mikrocomputer müssen synchronisiert werden, um langsame und schnelle Komponenten aneinander anzupassen und Datenüberläufe zu vermeiden
- Annahme und Erzeugung von Unterbrechungsanforderungen (Interrupt Requests)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Schnittstellenbausteine



- Ein Schnittstellenbaustein wird von einem Mikroprozessor genau wie Speicher gehandhabt.
- Durch Schreiben und Lesen auf definierten Adressen kann der Mikroprozessor Kommandos, Statusinformationen oder Daten mit einem Schnittstellenbaustein austauschen.

Adressraumintegration von Speicher und Ein-/Ausgabe

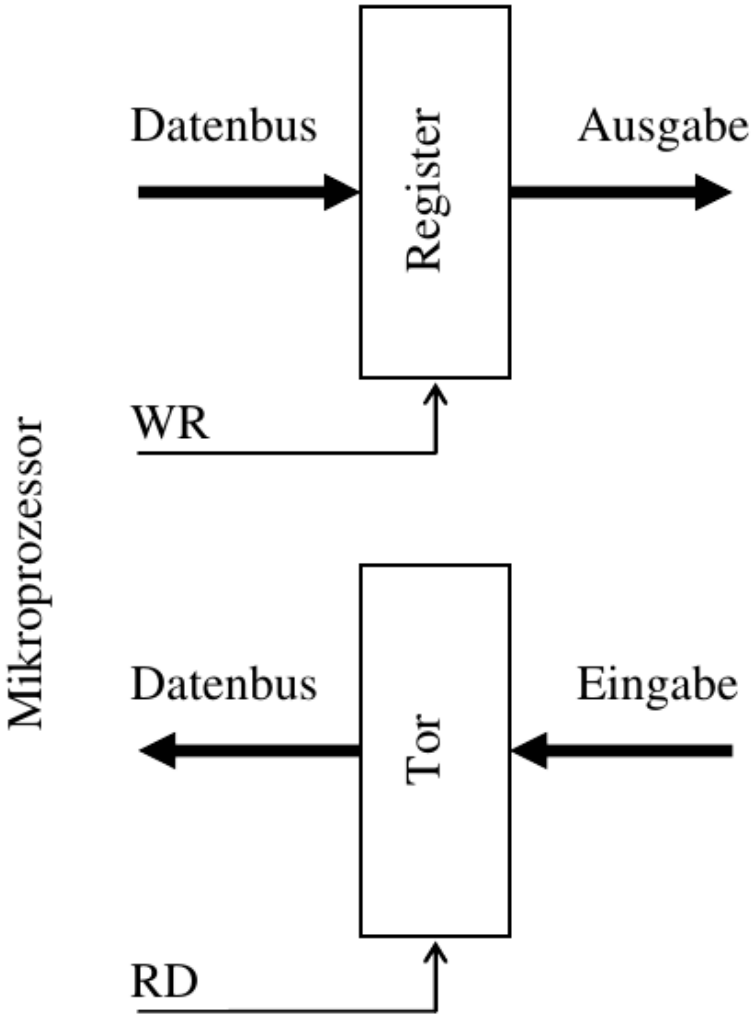
Schnittstellenbausteine verfügen über Register (Kommando-, Steuer-, Status- und Datenregister), die in den Adressraum des Mikroprozessors integriert sind. Es gibt zwei Arten der Integration:

- Memory Mapped IO: Speicher und Schnittstellenbausteine teilen sich einen Adressraum, der Prozessor kann dadurch die Befehle zum Zugriff auf „normalen“ Speicher nutzen.
- Isolated IO: Der Prozessor verfügt über zwei getrennte Adressbereiche für Speicher und IO. Die Nutzung von Schnittstellen reduziert somit nicht den für Speicher verfügbaren Adressraum. Allerdings muss der Prozessor über separate Befehle zum Zugriff auf den IO-Adressbereich verfügen.

In der x86-Welt (Intel und AMD) sind beide Arten üblich. Z.B. wird die serielle RS232-Schnittstelle im Isolated-IO-Adressraum angesprochen, andere Peripherie wie CAN- oder Netzwerkkarten sind in den Speicherbereich (Memory Mapped IO) eingeblendet oder haben sogar Register in beiden Adressbereichen.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Einfache Schnittstellen



- Einfache Schnittstellen bestehen nur aus Registern, Puffern oder Toren
- Daten werden direkt weitergegeben
- Eine einzige Ein- sowie eine Ausgabe-adresse im Adressraum
- WR: Schreibsignal, Übernahme in das Register
- RD: Lesesignal öffnet ein Tor und stellt am Eingang anliegende Daten zur Verfügung
- Das Zeitverhalten einfacher Schnittstellen bestimmt direkt der Mikroprozessor.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

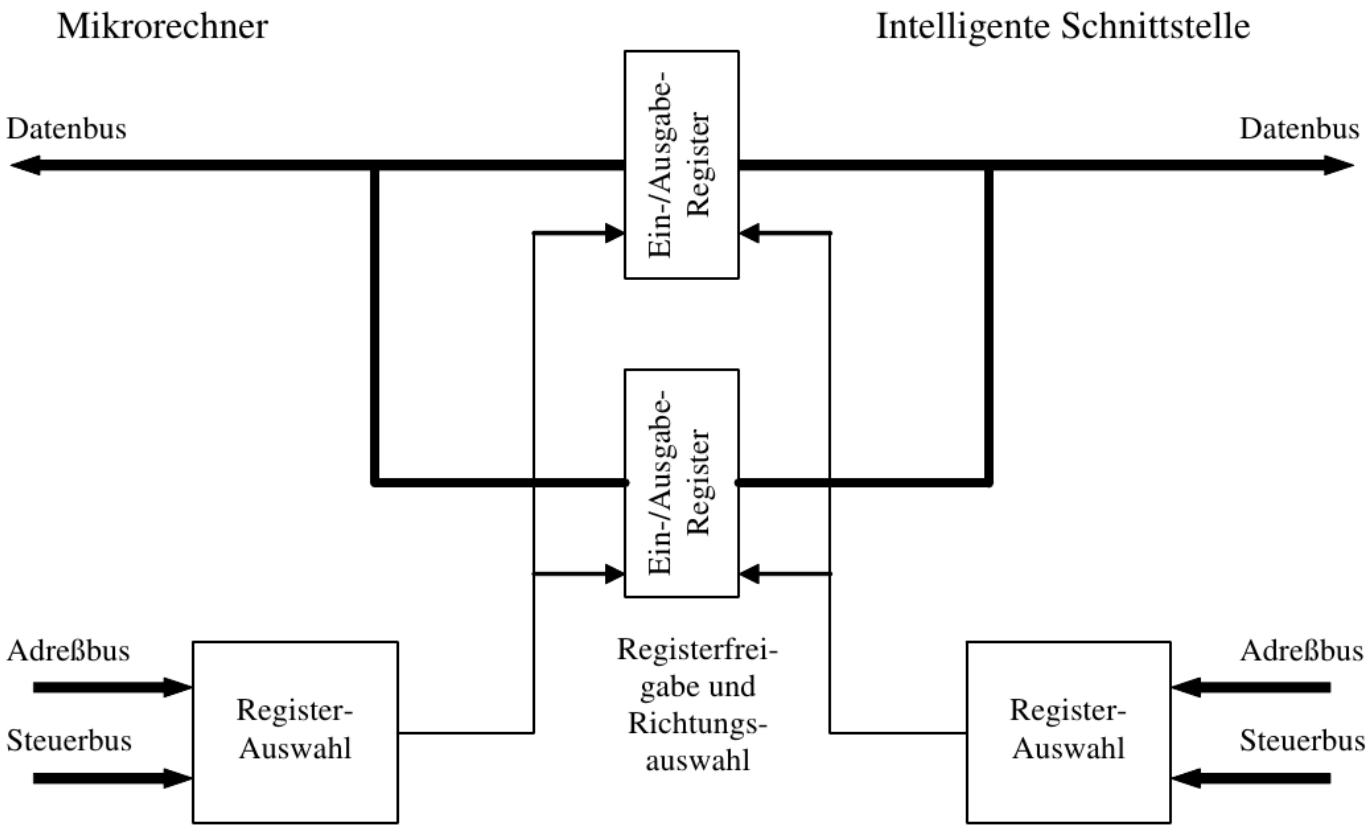
Komplexe und intelligente Schnittstellen

Komplexe Schnittstellen beinhalten programmierbare Bausteine und belegen mehrere Ein-/Ausgabeadressen, über die Daten ausgetauscht und die Funktionsweise konfiguriert werden kann. Dazu besitzen sie mehrere interne Register. Beispiel: serielle Schnittstelle, bei der die Betriebsart (Übertragungsrate, Datenformat, Kodierung, …) eingestellt werden muss und neben dem Datenaustausch viele Steuer- und Statussignale (z.B. Sender bzw. Empfänger bereit, Zeichen empfangen, …) anfallen.

Intelligente Schnittstellen haben einen eigenen Prozessor und lokalen Speicher. Sie stellen quasi ein eigenständiges Mikrorechnersystem dar, das an das eigentliche Mikrocomputersystem angebunden ist und dieses von umfangreichen Ein-/Ausgabeaufgaben entlastet. Es findet somit eine Parallelverarbeitung statt, was intelligente Schnittstellen für Echtzeitsysteme attraktiv macht.

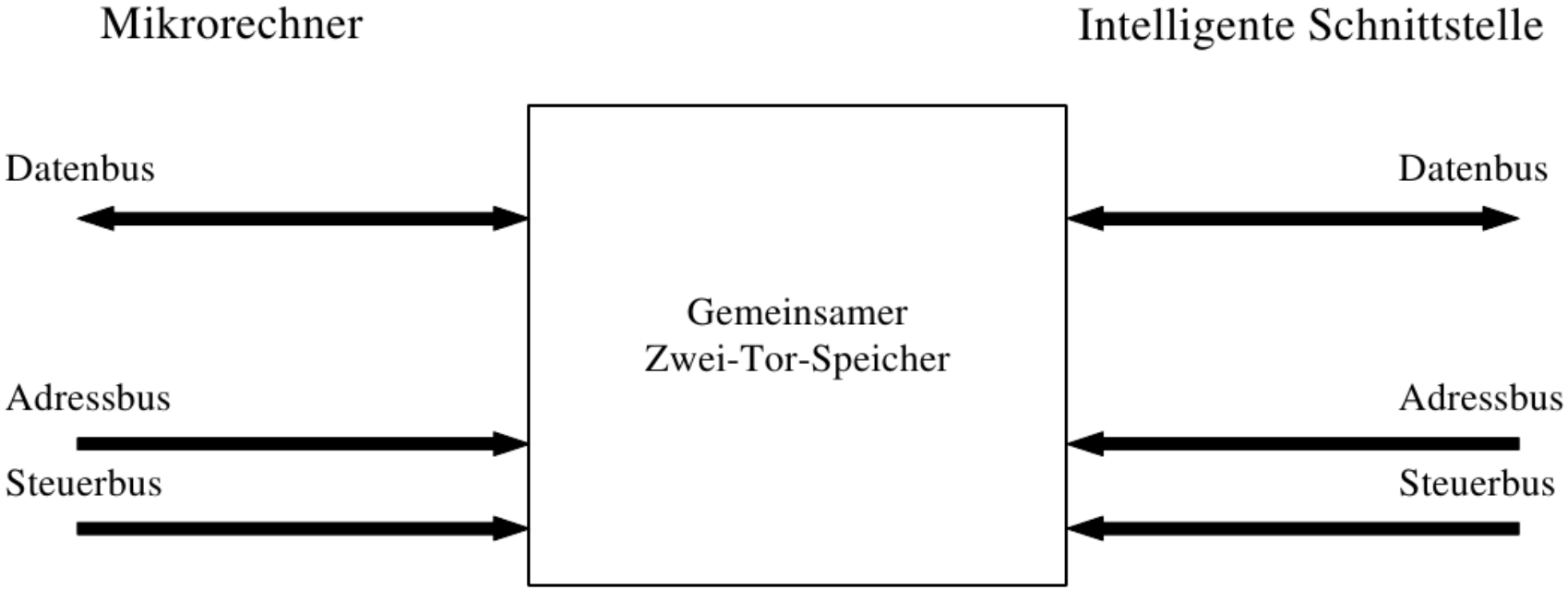
© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Ankopplung intelligenter Schnittstellen über I/O-Register



- Geeignet für die Übergabe kleiner Datenmengen, die komplexe Verarbeitungsschritte nach sich ziehen
- Beschränkte zeitliche Entkopplung
- Einsatz von FIFOs mildert das Warten der einen auf die andere Seite

Ankopplung über gemeinsamen Zwei-Tor-Speicher



- Austausch großer Datenmengen ohne Warten
- Zur Synchronisation werden oft zusätzliche I/O-Register verwendet, d.h. zuerst werden Daten in den Speicher geschrieben, danach wird durch Beschreiben eines Registers der Gegenseite das Signal zum Lesen gegeben.

Ankopplung intelligenter Schnittstellen über einen Buskoppler

- Die direkte Busanbindung ist die schnellste und direkteste Form.
- Die Busse von Mikrocomputer und intelligenter Schnittstelle werden zeitweilig verbunden.
- Multi-Master-System
- Der eine Teilnehmer kann auf die Komponenten des jeweils anderen zugreifen.
- Keine zeitliche Entkopplung, da ein Teilnehmer warten muss, wenn der andere Daten überträgt.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kapitel 5: Eingebettete Computersysteme

5.1 Einleitung

5.2 Begriffsbestimmung

5.3 Architektur von Mikrocomputersystemen (Unterbrechungsbehandlung und Speicherhierarchien)

5.4 Mikrocontroller

5.5 Ein-/Ausgabeschnittstellen

5.6 Embedded-PC

Embedded-PC

- Modular aufgebauter, kompakter Industrie-PC, dessen Rechnerarchitektur sich am Standard-PC orientiert.
- Es werden nur Komponenten eingefügt, die das jeweilige Einsatzszenario fordert. Die normalerweise vorhandenen zugehörigen Schnittstellen entfallen.
- Z.B. „headless“: Keine Anzeige wie Monitor oder Display, keine Bediengeräte wie Maus oder Tastatur.
- Wegen geringer Leistungsaufnahme meist lüfterlos.
- Statt einer Festplatte werden oft CF-Karten eingesetzt.
- Vorteile der Standard-PC-Architektur: gewohnte Hard- und Software-Umgebung und die Verfügbarkeit entsprechender PC-Software und Entwicklungsumgebungen.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

PC/104

PC/104 ist ein Standard für PC-kompatible Module, in dem die Modulgröße, die Lage möglicher Erweiterungsanschlüsse sowie die Bedeutung der Pins des ISA-kompatiblen PC/104-Steckers definiert sind. Die Leiterplattengröße eines Moduls beträgt 90,17mm x 95,89mm.

Einzelne PC/104-Module können über ihre Steckverbindung übereinander „gestapelt“ werden. So wird aus einem einzelnen CPU-Modul mit wenigen Schnittstellen ein Rechnersystem mit der gewünschten Einsatzmöglichkeit (z.B. Erweiterungsmodule mit mehreren seriellen oder CAN-Anschlüssen).

PC/104-Plus: Weiterentwicklung von PC/104, verfügt über einen PCI-Bus.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Beispiel: SPB (Scalable Processing Box)

Zusammenfassung

Eingebettete Steuergeräte, wie sie in Kapitel 3 am Beispiel eines Kfz-Motorsteuergerätes eingeführt wurden, sind eine Sonderform von eingebetteten Computersystemen. In diesem Kapitel wurden die grundsätzliche Architektur, die wichtigsten Komponenten und zentralen Begriffe solcher in der Automatisierungstechnik und vielen anderen Bereichen eingesetzten Echtzeitsysteme vorgestellt. Unter dem Aspekt der Echtzeit sind die folgenden Punkte besonders wichtig:

- Programmunterbrechung als Reaktion auf externe Ereignisse
- Prioritäten von Tasks
- Speicherhierarchien und Echtzeit
- Mikrocontroller mit Prozessorkern, Zähler, Zeitgeber, Watchdog, Speicherformen, IO-Register, Ein-/Ausgabekanäle, A/D-Wandler ...

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Kapitel 6: Echtzeitbetriebssysteme

6.1 Einleitung

6.2 Schichtenmodelle

6.3 Taskverwaltung

6.4 QNX als Beispiel für ein industrielles RTOS

6.5 Echtzeitbetriebssystem Linux/Xenomai

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Zum Nachlesen

[WB2005] Wörn, H. und Brinkschulte, U.: Echtzeitsysteme.
Springer-Verlag Berlin Heidelberg 2005
Kapitel 6 „Echtzeitbetriebssysteme“

[Ya+2008] Yaghmour, K; Masters, J.; Ben-Yossef, G. und Gerum, P.:
Building Embedded Linux Systems, 2nd Edition.
O‘Reilly Media Inc. 2008
Kapitel 12 und 13

[Sm2007] Smolorz, S.: Echtzeit-Linux mit Xenomai.
Zeitschrift Elektronik, Ausgabe 03/2007, S. 86 ff.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Einleitung

Ein Echtzeitbetriebssystem (Real-time Operating System, RTOS) stellt in modernen Echtzeitsystemen als zentrale Basiskomponente das Umfeld, die Werkzeuge und Dienste bereit, ohne die Echtzeitapplikationen ihre Funktion (vor allem unter dem Aspekt der Rechtzeitigkeit) nicht erfüllen könnten.

Die klassischen Aufgaben eines Standardbetriebssystems sind zunächst auch die Aufgaben eines Echtzeitbetriebssystems:

- Taskverwaltung (Scheduling): Zuteilung des Prozessors an die Tasks
- Betriebsmittelverwaltung (im Wesentlichen Speicherverwaltung und Ein-/Ausgabeverwaltung): Zuteilung von Speicher sowie Zuteilung von Ein-/Ausgabegeräten an die Tasks
- Interprozesskommunikation (IPC): Kommunikation zwischen den Tasks
- Synchronisation: Zeitliche Koordination der Tasks
- Schutzmaßnahmen: Schutz der Betriebsmittel vor unberechtigten Taskzugriffen

Aufgaben eines RTOS

Die Kernaufgaben eines Echtzeitbetriebssystems sind die Wahrung der

- Rechtzeitigkeit
- Gleichzeitigkeit
- Verfügbarkeit

Der Durchsatz und eine für alle Anwendungen gute Reaktionszeit (z.B. auf einer grafischen Benutzungsoberfläche) sind nur sekundär. Primär ist die zeitliche Vorhersagbarkeit.

In Echtzeitbetriebssystemen ersetzen spezielle Algorithmen die in Standardbetriebssystemen normalerweise eingesetzten Pendants, z.B. im Rahmen der Speicherreservierung und -freigabe. Beispielsweise wäre während einer Reorganisationspause im Zuge eines Speicherbereinigungsvorgangs die Anforderung der Verfügbarkeit verletzt.

Kapitel 6: Echtzeitbetriebssysteme

6.1 Einleitung

6.2 Schichtenmodelle

6.3 Taskverwaltung

6.4 QNX als Beispiel für ein industrielles RTOS

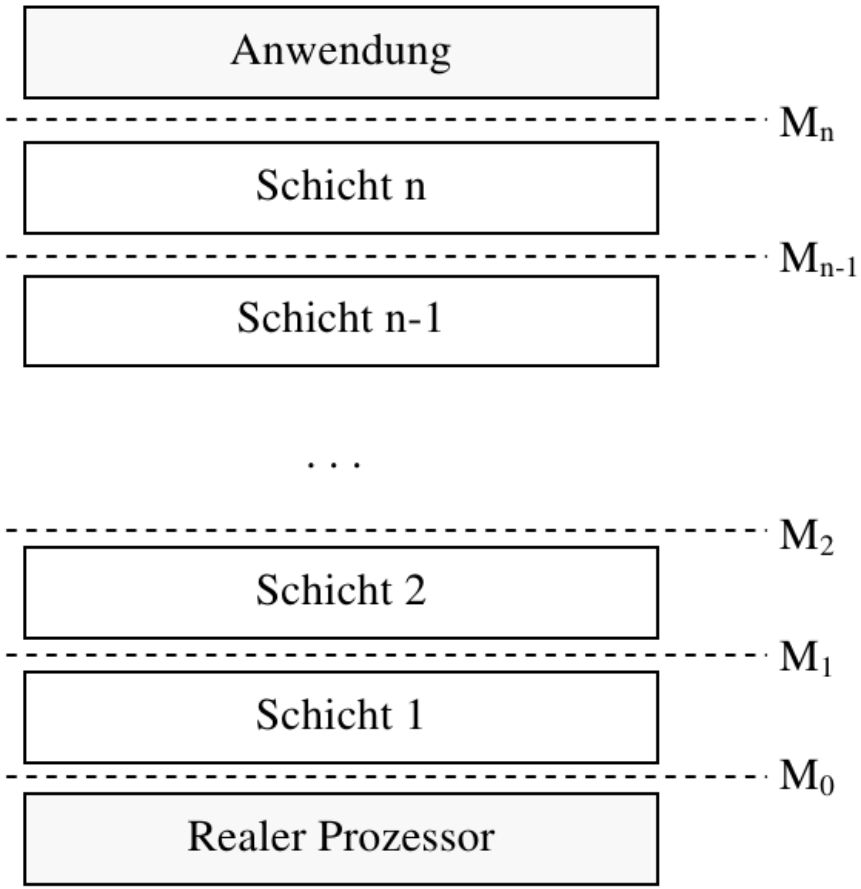
6.5 Echtzeitbetriebssystem Linux/Xenomai

Schichtenmodelle

Die ersten Betriebssysteme hatten einen monolithischen Aufbau.
Folge:

- schlechte Wartbarkeit
- schlechte Anpassbarkeit
- hohe Fehleranfälligkeit

Es entstand das hierarchische Schichtenmodell: Basierend auf dem realen Prozessor bauen sich Softwareschichten mit zunehmendem Abstraktionsgrad auf. Eine Schicht nutzt die Funktionalität der unter ihr liegenden Schicht, erweitert sie und stellt ihrerseits der über ihr liegenden Schicht ihre Funktionalität zur Verfügung.



(aus: [WB2005])

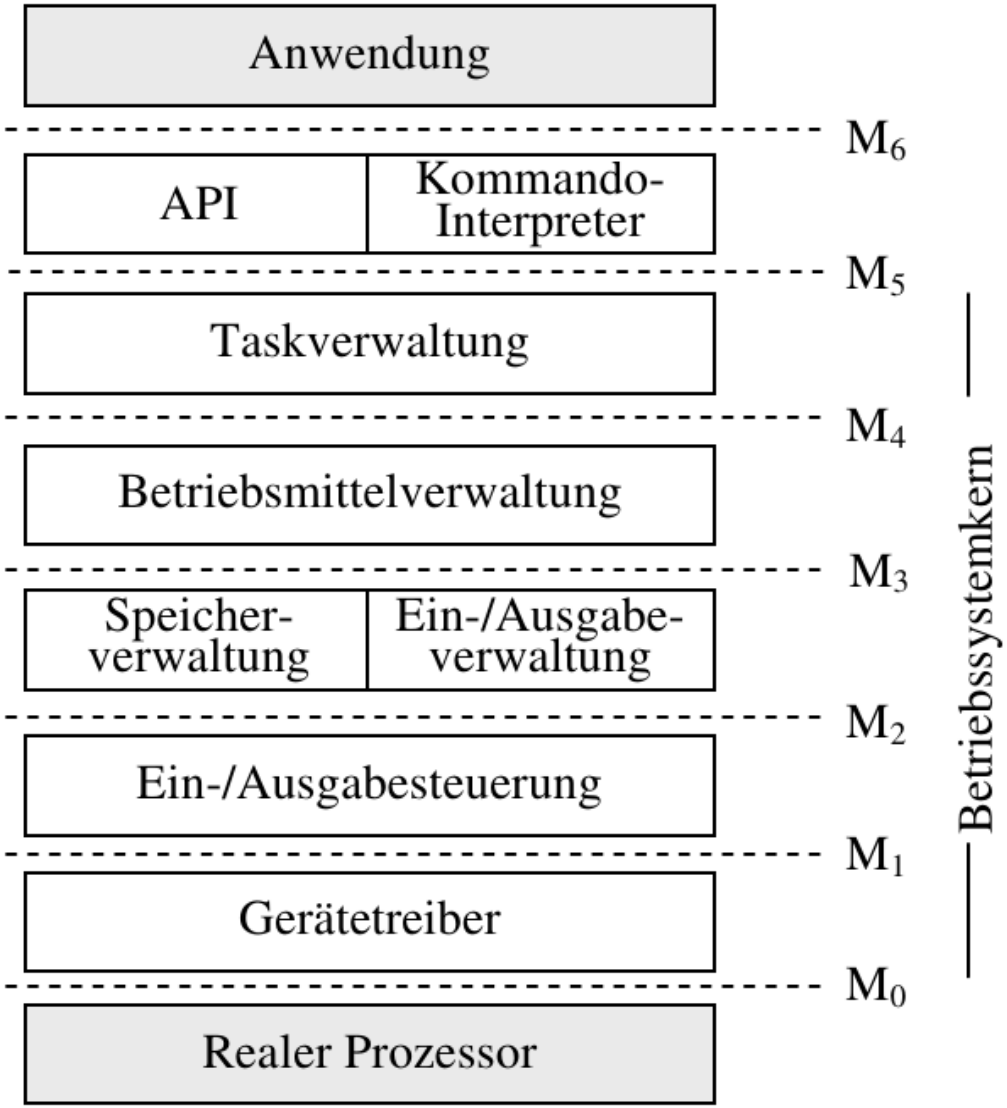
Schichtenmodell eines Betriebssystems

Gerätetreiber:

- Abstraktion von der Hardware
- hardwareabhängige Steuerung der einzelnen Geräte
- liefert eine hardware-unabhängige Schnittstelle an die darüber liegende Schicht
- im Idealfall einzige hardware-abhängige Schicht
- gerätespezifische Eigenheiten und Anpassungen nur hier

Ein-/Ausgabesteuerung:

- logische, hardware-unabhängige Steuerung der Geräte



(aus: [WB2005])

Schichtenmodell eines Betriebssystems

Ein-/Ausgabeverwaltung:

- wacht über die Zuteilung der Geräte an Tasks
- leitet Geräteanfragen und -antworten weiter

Speicherverwaltung: Reservierung (Allokation) und Freigabe (Deallokation) von Speicher

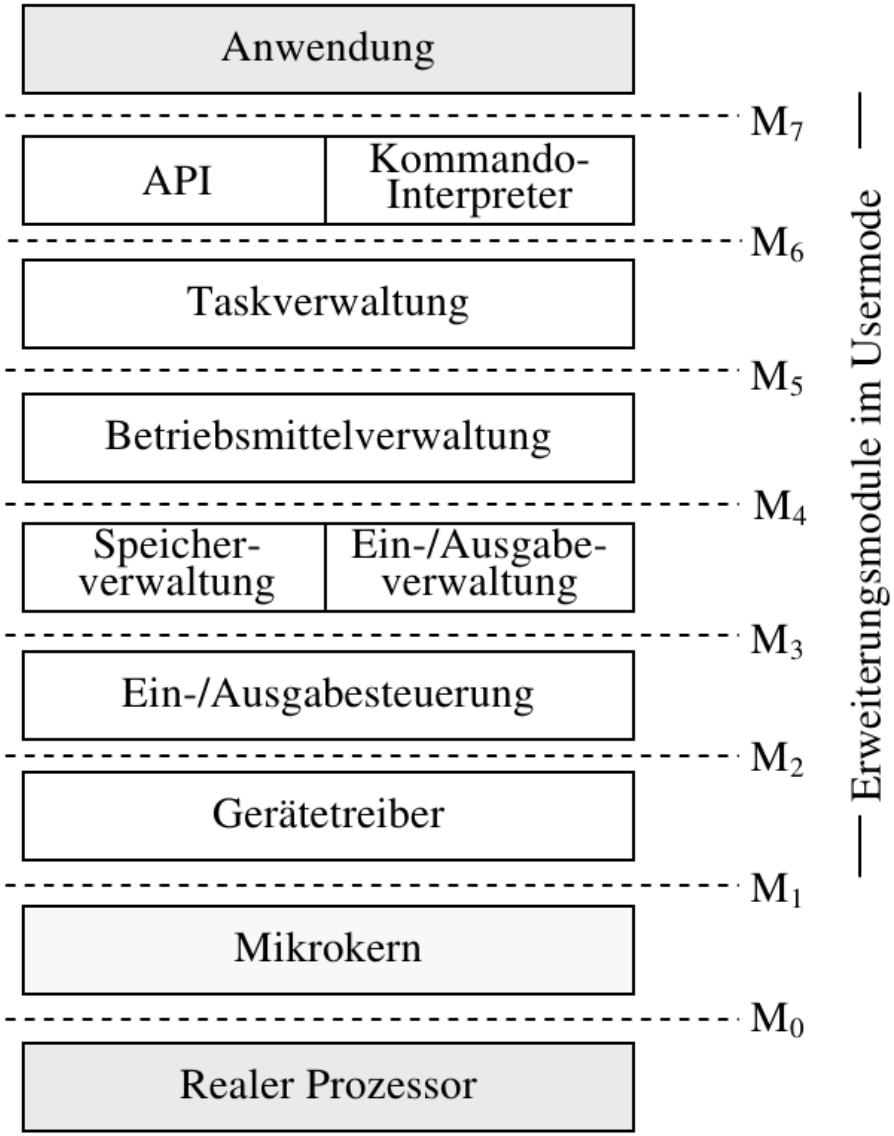
API (Application Programming Interface): Schnittstelle zu Anwenderprogrammen

Betriebssystemkern (Kernel):

- übernimmt kritische Aufgaben
- läuft im Kernelmode des Prozessors, in dem privilegierte Befehle und Speicherbereiche freigegeben sind, im Gegensatz zum Usermode
- erstreckt sich über mehrere Schichten (im Bild sind es fünf) und heißt deshalb Makrokernbetriebssystem

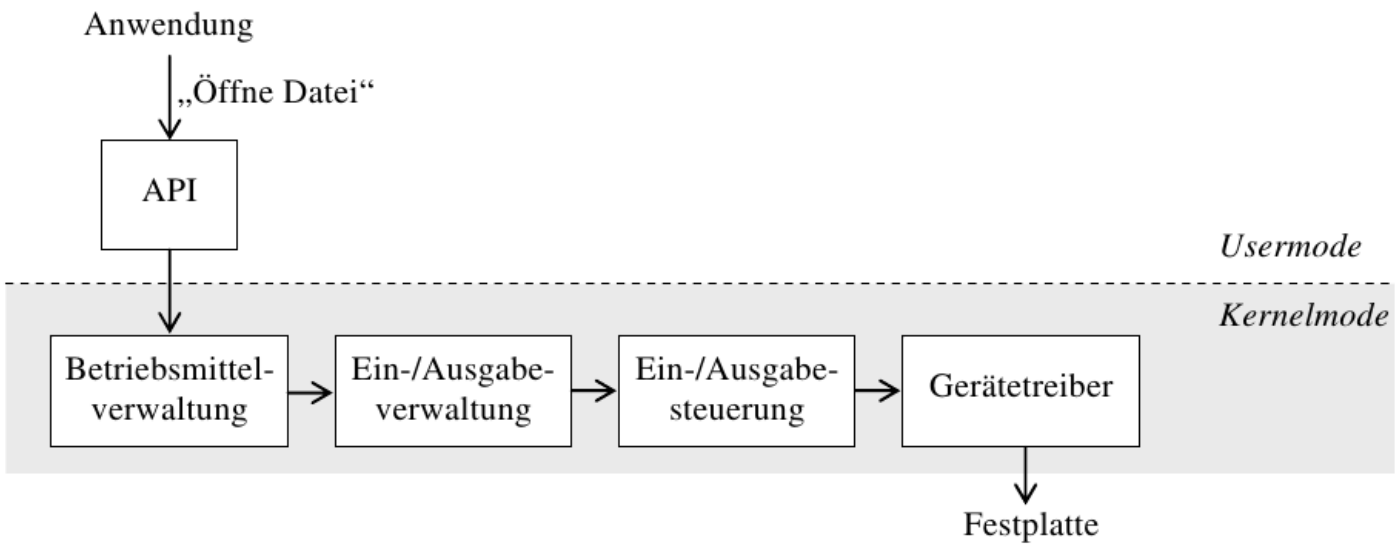
Mikrokernbetriebssystem

- besser an eine Aufgabenstellung und an die geringen Ressourcen eingebetteter Systeme anpassbar, besser skalierbar
- leichtere Portierbarkeit auf andere Plattformen aufgrund des Baukastenprinzips
- Kritische (d.h. nicht unterbrechbare) Bereiche beschränken sich auf den Mikrokern und sind im Idealfall sehr kurz. Solch ein präemptiver Kern ist fast immer unterbrechbar und bietet eine schnelle Reaktion auf auftretende Ereignisse. Bei Makrokernen können sich kritische Bereiche über mehrere Schichten erstrecken und entsprechend lang sein.

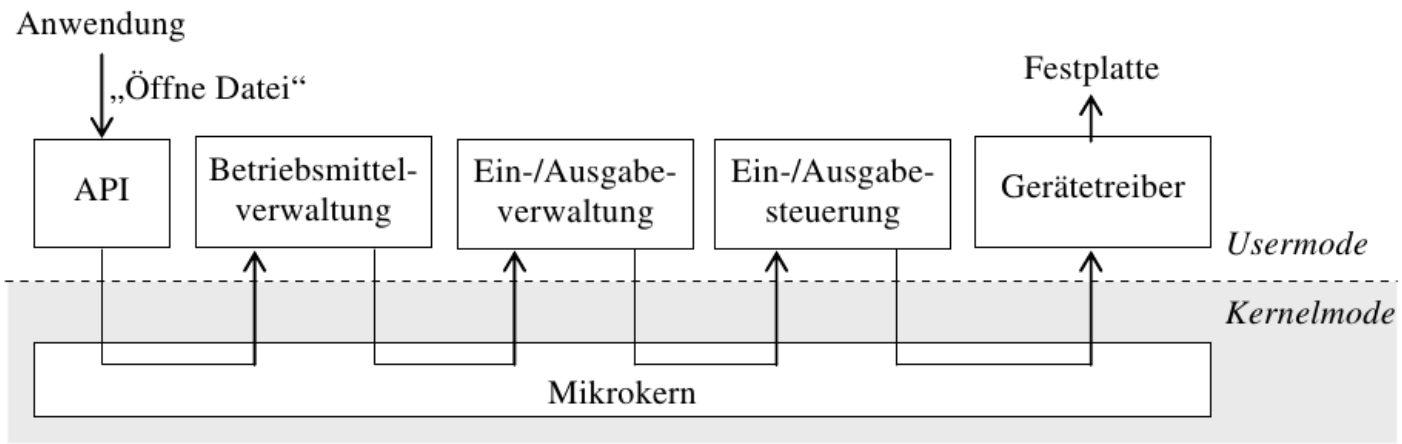


(aus: [WB2005])

Nachteile eines Mikrokernels



a: Makrokernbetriebssystem



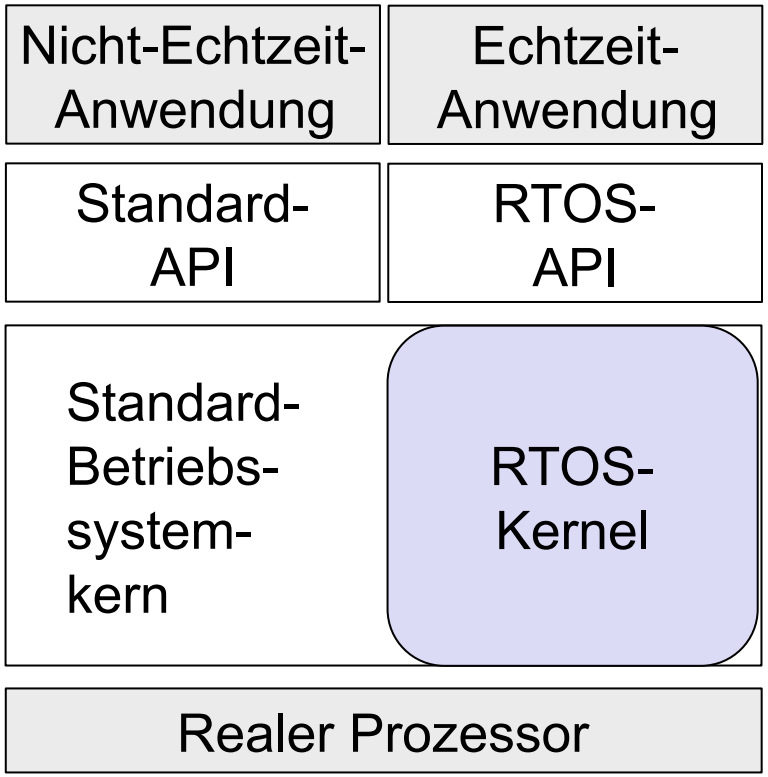
b: Mikrokernbetriebssystem

(aus: [WB2005])

- häufigerer Wechsel zwischen User- und Kernelmode
- Häufigeres Kopieren von Speicher-bereichen statt einfacher Zeiger-übergaben
- Performance-Overhead

Dual-Kernel

- Ein Standardbetriebssystem wird um einen kleineren RTOS-Kernel erweitert, beide Kernel arbeiten zusammen (kooperativer Ansatz).
- Jeder Kernel hat seine eigenen Schichten für Gerätetreiber, E/A-Steuerung, Betriebsmittelverwaltung und Taskverwaltung (s. Folie 7).
- Der RTOS-Kernel muss lediglich für ihn wichtige Funktionalitäten implementieren, ansonsten profitiert er von vorhandenen Funktionen.
- Der RTOS-Kernel muss stets die Kontrolle über eintreffende Ereignisse haben und den Standard-Kernel immer unterbrechen können.
- Real-Time (RT) und Non-Real-Time (NRT)-Programme können nebeneinander existieren.
- Beispiel: Linux/Xenomai



Kapitel 6: Echtzeitbetriebssysteme

6.1 Einleitung

6.2 Schichtenmodelle

6.3 Taskverwaltung

6.4 QNX als Beispiel für ein industrielles RTOS

6.5 Echtzeitbetriebssystem Linux/Xenomai

Taskverwaltung

- Kernaufgabe von Betriebssystemen
- wesentliche Unterschiede zwischen Standard- und Echtzeitbetriebssystemen
- Rechtzeitigkeit und Gleichzeitigkeit gelten gerade für die Tasks einer Echtzeitanwendung!
- In Echtzeitanwendungen werden daher andere Scheduling-Strategien angewendet als in Standard-Betriebssystemen.
- „Task“ bezeichnet sowohl einen Prozess (im Sinne von Unix) als auch einen Thread (Ablauffaden).
- andere Definition als in [WB2005]

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Taskmodell

Prozess:

- auf einem Rechner ablaufendes Programm mit allen zugehörigen Variablen und Betriebsmitteln.
- ein vom Betriebssystem gesteuerter Vorgang der Abarbeitung eines sequenziellen Programms

Mehrere Tasks können quasi-parallel ablaufen, der Wechsel zwischen den Tasks wird vom Betriebssystem gesteuert. Ein RTOS muss hierbei die Einhaltung der geforderten Zeitbedingungen sicherstellen (sofern möglich).

Threads stellen eine weitere Hierarchiestufe von parallel ausführbaren Objekten dar. Ein Prozess kann mehrere Threads enthalten. Es entsteht eine zweistufige Hierarchie der Parallelität:

(nach: [WB2005])

Prozesse und Threads

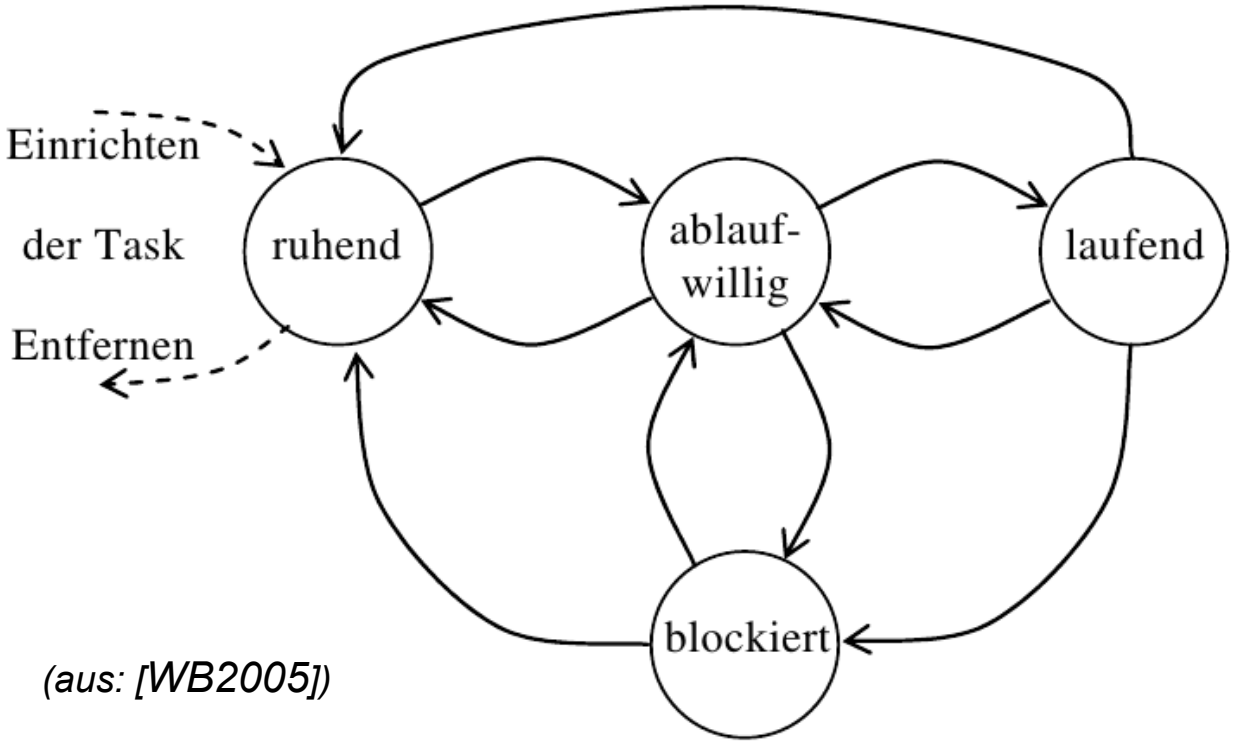
Ein Prozess besitzt eigene Variablen und Betriebsmittel und wird von anderen Prozessen durch das Betriebssystem abgeschirmt. Da jedem Prozess ein eigener Adressraum zur Verfügung steht, können Prozesse nur über die Mittel der Interprozesskommunikation Informationen austauschen.

Ein Thread wird auch leichtgewichtiger Prozess genannt. Er benutzt die Variablen und Betriebsmittel des Prozesses, dessen Teil er ist. Die Threads eines Prozesses können Informationen mit Hilfe globaler Variablen austauschen, denn alle Threads teilen denselben Adressraum innerhalb eines Prozesses.

Die Kommunikation zwischen Threads ist direkter und schneller als zwischen Prozessen, allerdings ist auch der Schutz wesentlich geringer. Aber vor allem der Kontextwechsel zwischen Threads ist schneller und kann in wenigen Taktzyklen erfolgen. Beim Wechsel von einem Prozess zum anderen müssen umfangreiche Zustandsinformationen des alten Prozesses gesichert und des neuen geladen werden, was mehrere tausend Taktzyklen dauern kann.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

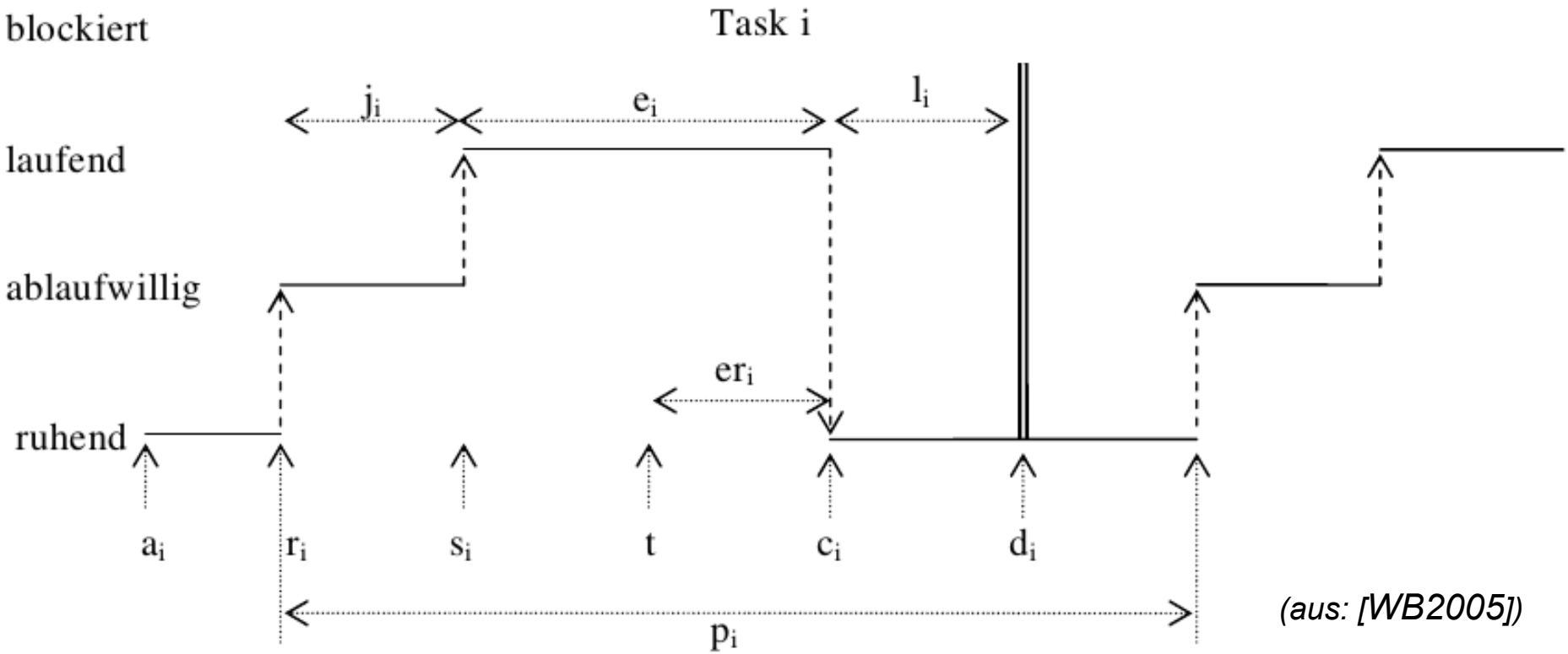
Taskzustände



(aus: [WB2005])

- Ruhend (dormant): Task nicht ablaufbereit, da Zeitbedingungen oder andere Voraussetzungen (noch) nicht erfüllt sind
- Ablaufwillig (runnable): Alle Zeitbedingungen oder andere Voraussetzungen zum Ablauf sind erfüllt und die Betriebsmittel sind zugeteilt, nur der Prozessor noch nicht
- Laufend (running): Der Task wird auf dem/einem Prozessor ausgeführt. Je nach Anzahl der Prozessoren bzw. Prozessorkerne können genauso viele Tasks in diesem Zustand sein.
- Blockiert (suspended, blocked): Der Task wartet auf ein Ereignis (z.B. einen Eingabewert oder eine Interprozesskommunikation) oder das Freiwerden eines Betriebsmittels (Synchronisation).

Zeitparameter eines Echtzeit-Tasks



(aus: [WB2005])

a_i: Ankunftszeit (Arrival Time): Zeitpunkt, an dem der Task dem Betriebssystem bekannt gemacht wird; wird in den Zustand „ruhend“ gesetzt

r_i: Anforderungszeit (Request Time): Task wird ablaufwillig und bewirbt sich bei der Taskverwaltung um den Prozessor

Zeitparameter eines Echtzeit-Tasks

s_i: Startzeit (Start Time): Erstmalige Zuteilung des Prozessors nach der Anforderung; die Ausführung beginnt.

c_i: Beendigungszeit (Completion Time): Ende der Taskausführung, der Task hat seine Aufgabe erledigt und kehrt in den Zustand „ruhend“ zurück.

d_i: Zeitschranke (Deadline): Zeitpunkt, zu dem die Taskausführung spätestens beendet sein muss (ggf. auch weich).

p_i: Periode (Period): Die Wiederholungsrate bei periodischen Tasks ergibt sich aus der Zeitdifferenz zwischen zwei aufeinander folgenden Anforderungszeiten.

e_i: Ausführungszeit (Execution Time): Zeitdauer, die ein Task zur Ausführung benötigt, wenn er im Zustand „laufend“ ist, d.h. die reine Ausführungszeit ohne Wartephasen. Die WCET als obere Schranke zählt ebenfalls hierzu.

er_i: Restausführungszeit (Remaining Execution Time): noch verbleibende Ausführungszeit von der aktuellen Zeit t bis zur Beendigungszeit

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Zeitparameter eines Echtzeit-Tasks

l_i: Spielraum (Laxity): Verbleibende Restzeit zwischen Beendigungszeit und Zeitschranke; je näher die Beendigungszeit an der Zeitschranke, desto geringer der Spielraum, der für die Einplanung des Tasks zur Verfügung steht.

j_i: Reaktionszeit (Reaction Time, Release Jitter): Verzögerung zwischen Anforderung und Start der Ausführung; die Zeit, innerhalb derer der Task nach einem Ereignis mit der Ereignisbehandlung reagiert; Schwankungen können vorkommen.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Echtzeitscheduling

Die Hauptaufgabe der Taskverwaltung, die der Scheduler übernimmt, liegt in der Zuteilung des Prozessors an die ablaufwilligen Tasks. Es gibt verschiedene Zuteilungsstrategien, die Scheduling-Verfahren genannt werden.

Ein Echtzeitscheduler hat die Aufgabe, die Zuteilung so vorzunehmen, dass alle Zeitbedingungen eingehalten werden – sofern möglich!

Die Menge der verwalteten Tasks heißt Taskset.

Drei Fragen müssen beantwortet werden, um ein Echtzeitscheduling-Verfahren bewerten zu können:

- Können für ein Taskset alle Zeitbedingungen überhaupt eingehalten werden? Wenn ja, existiert wenigstens ein Schedule, d.h. eine zeitliche Aufteilung des Prozessors an die Tasks, der die Aufgabe löst.
- Kann dieser Schedule in endlicher Zeit berechnet werden?
- Findet das verwendete Scheduling-Verfahren diesen Schedule (wenn er existiert und in endlicher Zeit berechnet werden kann)? Wenn ja, handelt es sich um ein optimales Scheduling-Verfahren.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Prozessorauslastung

Zentrale Größe für die Bewertung von Schedulingverfahren (Scheduling Analysis) ist die Prozessorauslastung H:

H = Benötigte Prozessorzeit / Verfügbare Prozessorzeit

Beispiel: Ein periodischer Task hat eine Ausführungszeit von 100 ms und eine Periodendauer von 200 ms. Dann ist

H = 100 ms / 200 ms = 0,5 = 50%

Kommt ein weiterer Task mit einer Ausführungszeit von 50 ms und der Periodendauer 100 ms hinzu, erhöht sich die Auslastung auf

H = (100 ms / 200 ms) + (50 ms / 100 ms) = 1,0 = 100%

Allgemein berechnet sich die Prozessorauslastung für ein Taskset aus n periodischen Tasks wie folgt:

$$H = \sum_{i=1}^{n} \frac{e_i}{p_i}$$

mit e_i: Ausführungszeit, p_i : Periodendauer von Task i

Prozessorauslastung

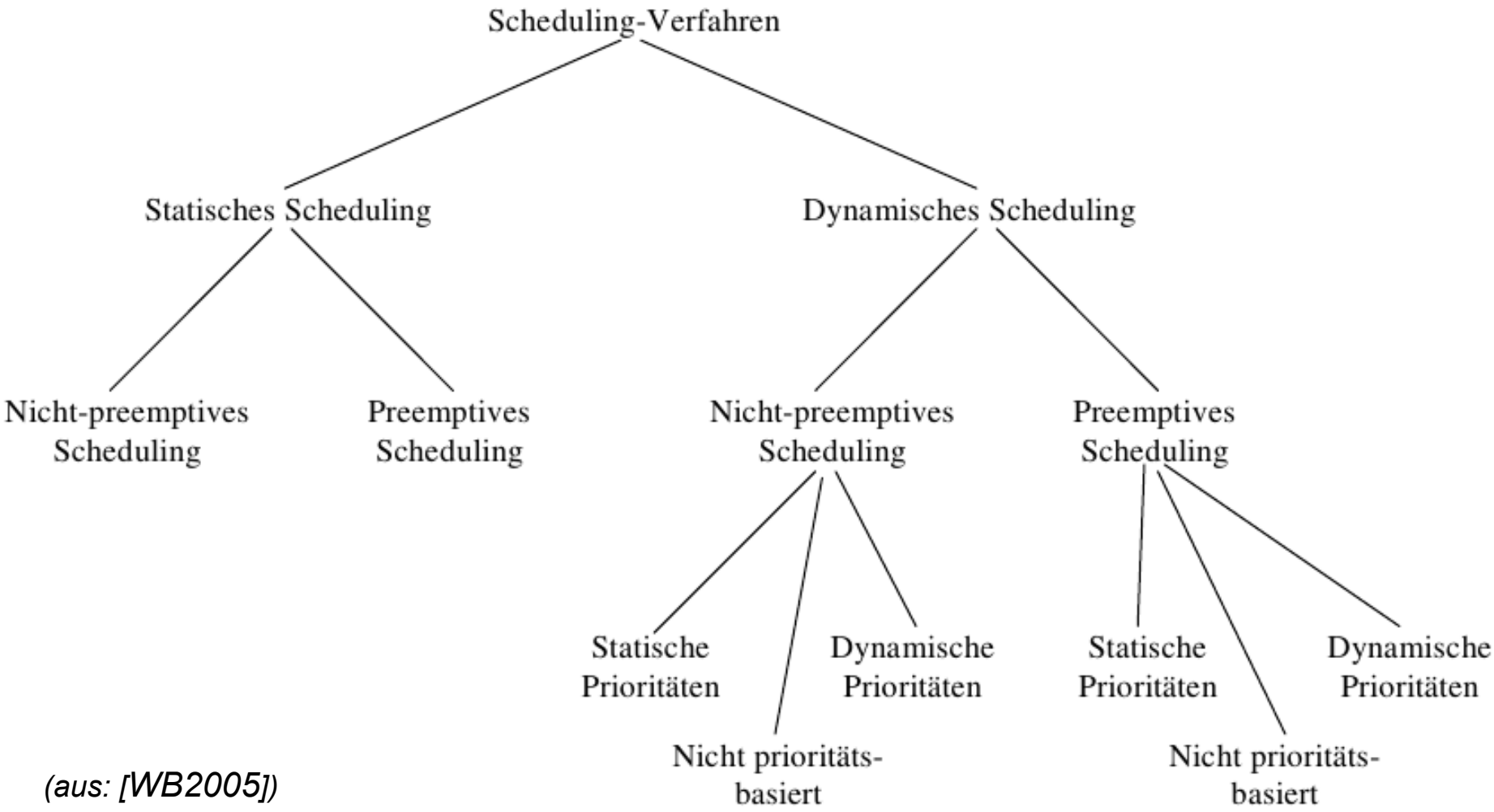
Bei einer Prozessorauslastung von mehr als 100% (und nur einem verfügbaren Prozessor) ist das Taskset nicht mehr ausführbar, d.h. es existiert kein Schedule, der alle Zeitbedingungen einhalten kann.

Ist H ≤ 100%, sollte allerdings ein Schedule existieren. Es hängt von der Qualität des eingesetzten Scheduling-Verfahrens ab, ob der Schedule tatsächlich gefunden wird.

Die Formel für H (auf der vorigen Folie) gilt streng genommen nur für periodische Tasks, deren Zeitschranken identisch mit ihren Perioden ist, d.h. bis zur nächsten Aktivierung eines Tasks muss die vorherige Aufgabe erledigt sein. Diese Einschränkung ist für viele Anwendungen akzeptabel.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Klassifizierung von Scheduling-Verfahren



(aus: [WB2005])

Statisches Scheduling

- Vor der Ausführung wird eine Zuordnungstabelle mit den Startzeiten der Tasks berechnet.
- Umfangreiche Analysen über Zeitschranken, Ausführungszeiten, Abhängigkeiten, Reihenfolgen usw. können durchgeführt werden.
- Die Zuteilung erfolgt zur Laufzeit durch eine Dispatcher.
- entspricht den Prinzipien der synchronen Programmierung
- minimaler Overhead zur Laufzeit
- starr und beschränkt auf periodische Ereignisse

Dynamisches Scheduling

- berechnet die Zuordnung der Tasks an den Prozessor zur Laufzeit
- Analysen über Zeitschranken und Ausführungszeiten führen zu erhöhtem Laufzeit-Overhead
- entspricht der asynchronen Programmierung
- Vorhersagbarkeit geringer als bei statischem Scheduling
- flexiblerer Ablauf möglich
- Reaktion auf aperiodische Ereignisse möglich
- Echtzeitbetriebssysteme benutzen im Allgemeinen dynamisches Scheduling.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Prioritäten

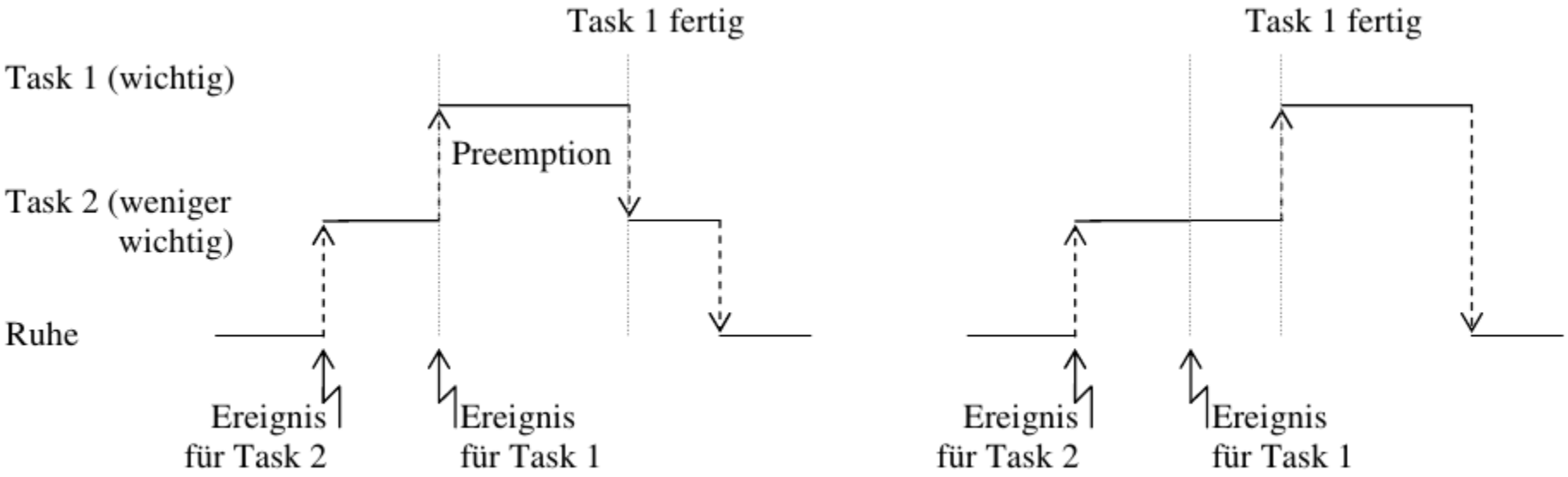
- Statische und dynamische Prioritäten gehören zum dynamischen Scheduling.
- Jeder Task besitzt eine Priorität, auf Basis derer sich die zeitliche Zuordnung zum Prozessor bestimmt.
- Bei statischen Prioritäten wird vor Beginn der Ausführung jedem Task eine Priorität zugeordnet, die sich zur Laufzeit nicht ändert.
- Bei dynamischen Prioritäten können die Taskprioritäten zur Laufzeit an die Gegebenheiten angepasst werden.
- Manche Scheduling-Verfahren verzichten völlig auf Prioritäten und treffen ihre Scheduling-Entscheidungen an Hand anderer Kriterien, z.B. der Prozessorauslastung.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Preemption („Vorkaufsrecht“)

Präemptives Scheduling: Ein wichtiger Task kann zur Laufzeit einen unwichtigeren verdrängen. Dadurch kommt immer der wichtigste ablaufwillige Task sofort zur Ausführung.

Nicht-präemptives Scheduling (kooperatives Scheduling): Ein laufender Task wird nicht verdrängt. Er räumt den Prozessor erst, wenn er nicht mehr ablaufwillig ist oder blockiert. Danach erst kommt der wichtigste ablaufwillige Task zur Ausführung.



a) Preemptives Scheduling *(aus: [WB2005])* **b) Nicht-preemptives Scheduling**

FIFO-Scheduling

- First In First Out: Wer zuerst kommt, erhält den Prozessor
- Derjenige Task, der am längsten auf die Ausführung wartet, erhält den Prozessor zugeteilt.
- Ein laufender Task wird nicht unterbrochen.
- nicht-präemptives, dynamisches Scheduling ohne Prioritäten
- einfacher Algorithmus, geringe Rechenzeit des Schedulers
- für Echtzeitanwendungen eher nicht geeignet, da Zeitbedingungen der einzelnen Tasks keine Rolle spielen, d.h. Gewährleistung der Rechtzeitigkeit nicht möglich

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

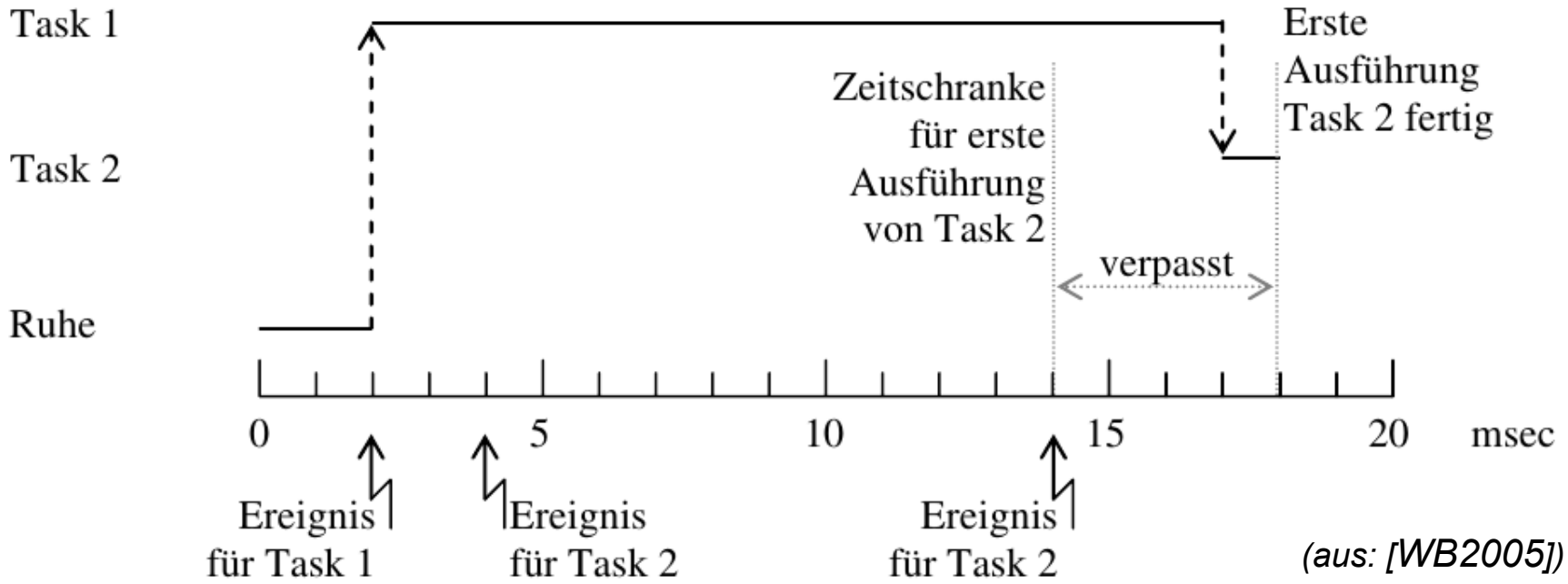
FIFO-Scheduling: Beispiel

Gegeben seien zwei Tasks mit folgenden Parametern:

Task 1: p_1 = 150 ms, e_1 = 15 ms Task 2: p_2 = 10 ms, e_2 = 1 ms

Alle Zeitschranken seien identisch zu den jeweiligen Perioden der Tasks.

H = (15 ms / 150 ms) + (1 ms / 10 ms) = 0,2 = 20%



(aus: [WB2005])

Der FIFO-Scheduler ist kein optimales Schedulingverfahren für Echtzeitanwendungen!

Fixed-Priority-Scheduling

Jedem Task wird eine feste Priorität zugeteilt. Der ablaufwillige Task höchster Priorität erhält den Prozessor. Es handelt sich also um dynamisches Scheduling mit statischen Prioritäten und kommt auch in der Unterbrechungsverarbeitung von Mikroprozessoren vor (vgl. Kapitel 5). Man unterscheidet zwischen

- Fixed-Priority-Preemptive-Scheduling (FPP) und
- Fixed-Priority-Non-Preemptive-Scheduling (FPN)

Welcher Task welche Priorität erhält, muss der Entwickler einer Echtzeitanwendung entscheiden. Die richtige Prioritätenvergabe entscheidet darüber, ob die Zeitbedingungen eingehalten werden oder nicht.

Um die Prioritäten der einzelnen Tasks zu bestimmen, kann bei rein periodischen Anwendungen das Rate-Monotonic-Scheduling (RMS) verwendet werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Rate-Monotonic-Scheduling

$PR_i \sim \frac{1}{p_i}$ mit p_i : Periodendauer der Task i, PR_i : Priorität der Task i

Je kürzer die Periode eines Task, desto höher seine Priorität. Es soll für alle Tasks folgendes gelten:

- präemptives Scheduling wird verwendet
- p_i ist konstant
- $d_i = p_i$
- e_i ist konstant und bekannt
- Die Tasks sind voneinander unabhängig, d.h. sie blockieren sich nicht gegenseitig z.B. durch Synchronisation.

Sind alle Voraussetzungen erfüllt, kann vorab überprüft werden, ob alle Zeitschranken eingehalten werden.

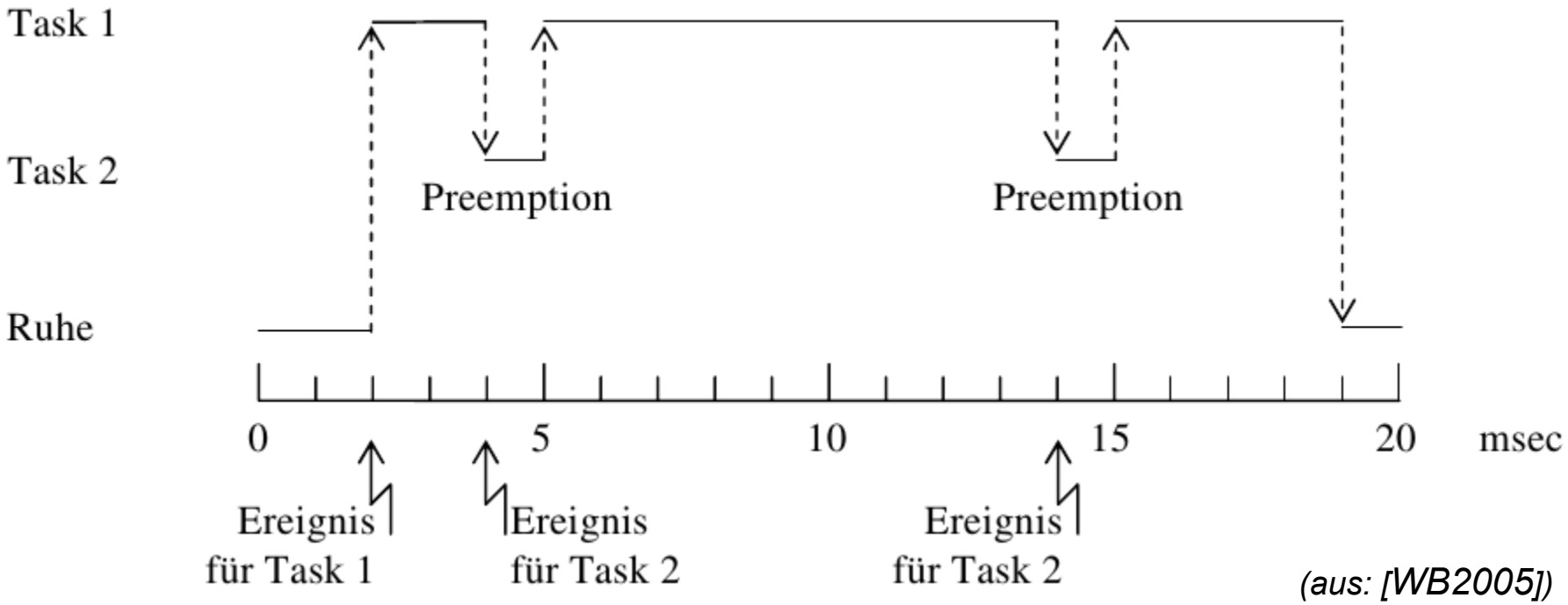
© 2015 B. Wagner - Leibniz Universität Hannover - RTS

RMS: Beispiel

Gegeben seien zwei Tasks mit folgenden Parametern (aus FIFO-Scheduling-Bsp.):

Task 1: p_1 = 150 ms, e_1 = 15 ms Task 2: p_2 = 10 ms, e_2 = 1 ms

Task 2 erhält deine höhere Priorität als Task1, da Task 2 eine kürzere Periode hat.

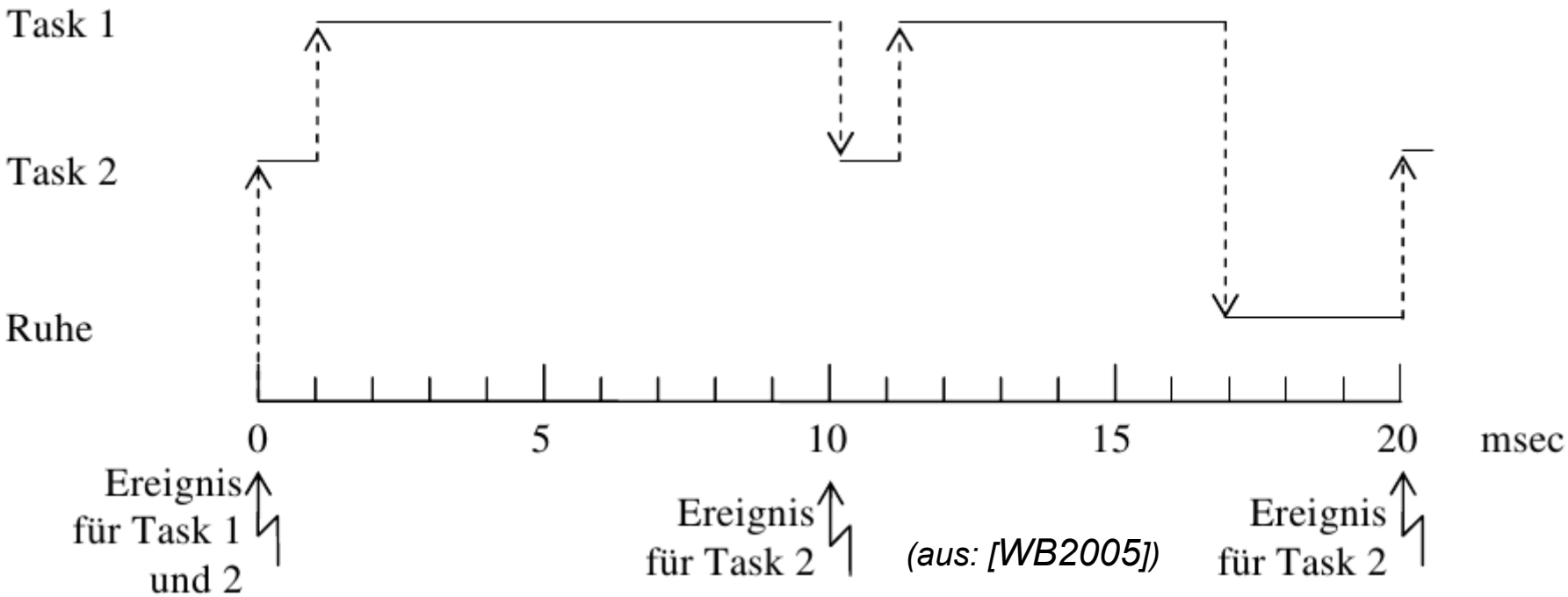


(aus: [WB2005])

Beide Tasks halten ihre Deadlines ein. Task 2 ist immer sofort an der Reihe, die Ausführungszeit von Task 1 erhöht sich nur leicht von 15 auf 17 ms.

RMS: Beispiel 2

Der ungünstigste Fall bei Echtzeitsystemen ist das gleichzeitige Eintreffen von Ereignissen. Auch hier ist das Taskset unter Einhaltung der Deadlines ausführbar:



(aus: [WB2005])

Das Fixed-Priority-Scheduling mit RMS und Präemptivität liefert also wesentlich bessere Ergebnisse als das FIFO-Scheduling.

Busy Period Analysis

Wenn bei einem periodischen, vollständigen Taskset und RMS kein Verstoß gegen die Zeitschranken auftritt, bis der Prozessor zum ersten Mal in den Ruhezustand übergeht, dann wird auch danach keine Verletzung der Zeitschranken vorkommen.

Dazu existiert ein mathematischer Beweis, so dass man das Verhalten von Echtzeittasks nur über einen bestimmten Zeitraum beobachten muss, um Aussagen über die Einhaltung von Zeitschranken machen zu können.

Optimale Prioritätenverteilung

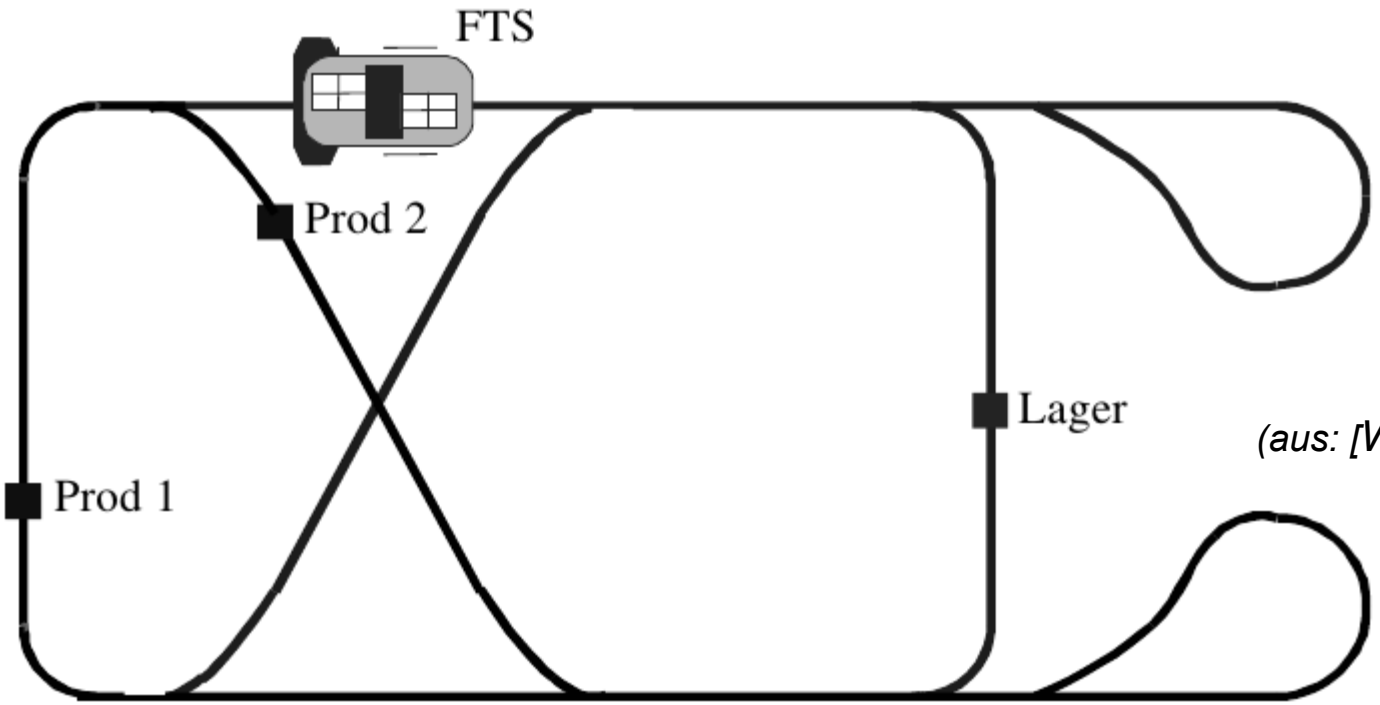
Für periodische Tasks, deren Zeitschranken ihren Perioden entsprechen, liefert RMS eine optimale Prioritätenverteilung unter den beiden Voraussetzungen der festen Prioritäten und der Präemptivität.

Auch diesen Satz kann man mathematisch beweisen.

Ist Fixed-Priority-Scheduling optimal?

Liefert das Fixed-Priority-Verfahren immer dann einen ausführbaren Schedule, wenn ein solcher existiert?

Beispiel: Fahrerloses Transportsystem (FTS) mit aufgeklebter Fahrspur und Landmarken



(aus: [WB2005])

FTS mit drei Tasks

T_1: Kameradatenverarbeitung, periodisch, erkennt eine auf dem Boden aufgeklebte Fahrspur und meldet Abweichungen an die Motorsteuerung.

$$p_1 = d_1 = 10 \text{ ms}, e_1 = 1 \text{ ms}$$

T_2: Motorsteuerung, periodisch, steuert und regelt die Fahrzeugmotoren so, dass sich das Fahrzeug immer mittig über der aufgeklebten Fahrspur bewegt.

$$p_2 = d_2 = 10 \text{ ms}, e_2 = 5 \text{ ms}$$

T_3: Transpondererkennung, **aperiodischer** Task, stoppt das Fahrzeug, sobald der Transponder eine Landmarke, z.B. eine Produktionsstation, erkannt hat. Die Positionsgenauigkeit eines Stopps beträgt 1cm, die Fahrgeschwindigkeit 0,65 m/s:

$$d_3 = 1 \text{ cm} : 0{,}65 \text{ m/s} = 15{,}4 \text{ ms}, e_3 = 5{,}5 \text{ ms}$$

Periodisierung von T_3: Annahme, dass im schlimmsten Fall Transponder-ereignisse alle 15,4 ms auftreten:

$$p_3 = d_3 = 15{,}4 \text{ ms}$$

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

FTS mit drei Tasks

Prozessorauslastung:

$$H = e_1 / p_1 + e_2 / p_2 + e_3 / p_3 =$$

$$(1\ \text{ms} / 10\ \text{ms}) + (5\ \text{ms} / 10\ \text{ms}) + (5{,}5\ \text{ms} / 15{,}4\ \text{ms}) = 95{,}71\ \%$$

Da H < 100% ist, sollte ein ausführbarer Schedule existieren.

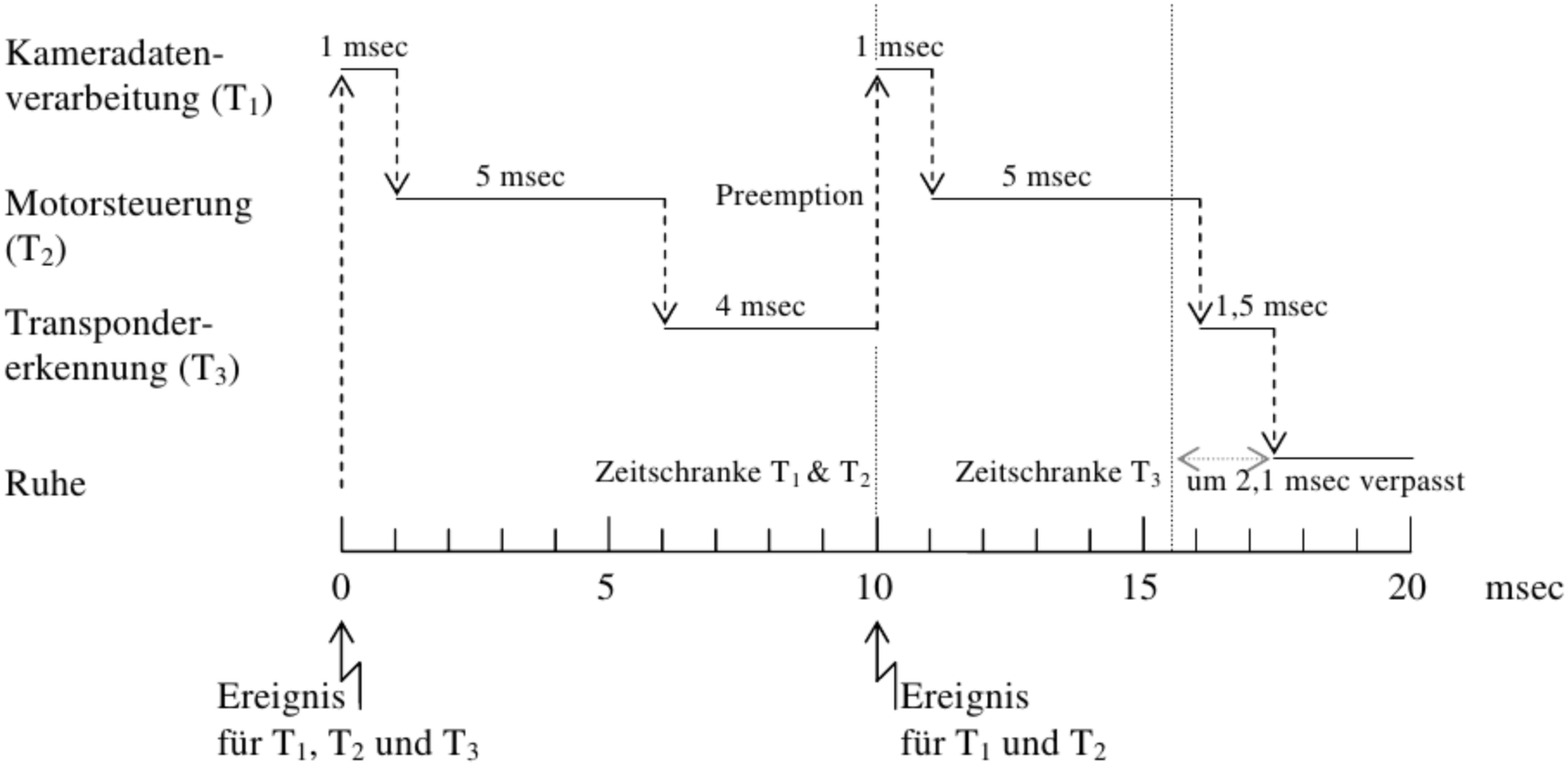
Nach RMS ergeben sich zwei mögliche Prioritätenverteilungen, da p_1 und p_2 gleich sind:

T_1 (Kameradatenverarbeitung):	hohe Priorität
T_2 (Motorsteuerung):	mittlere Priorität
T_3 (Transpondererkennung):	niedrige Priorität

Oder

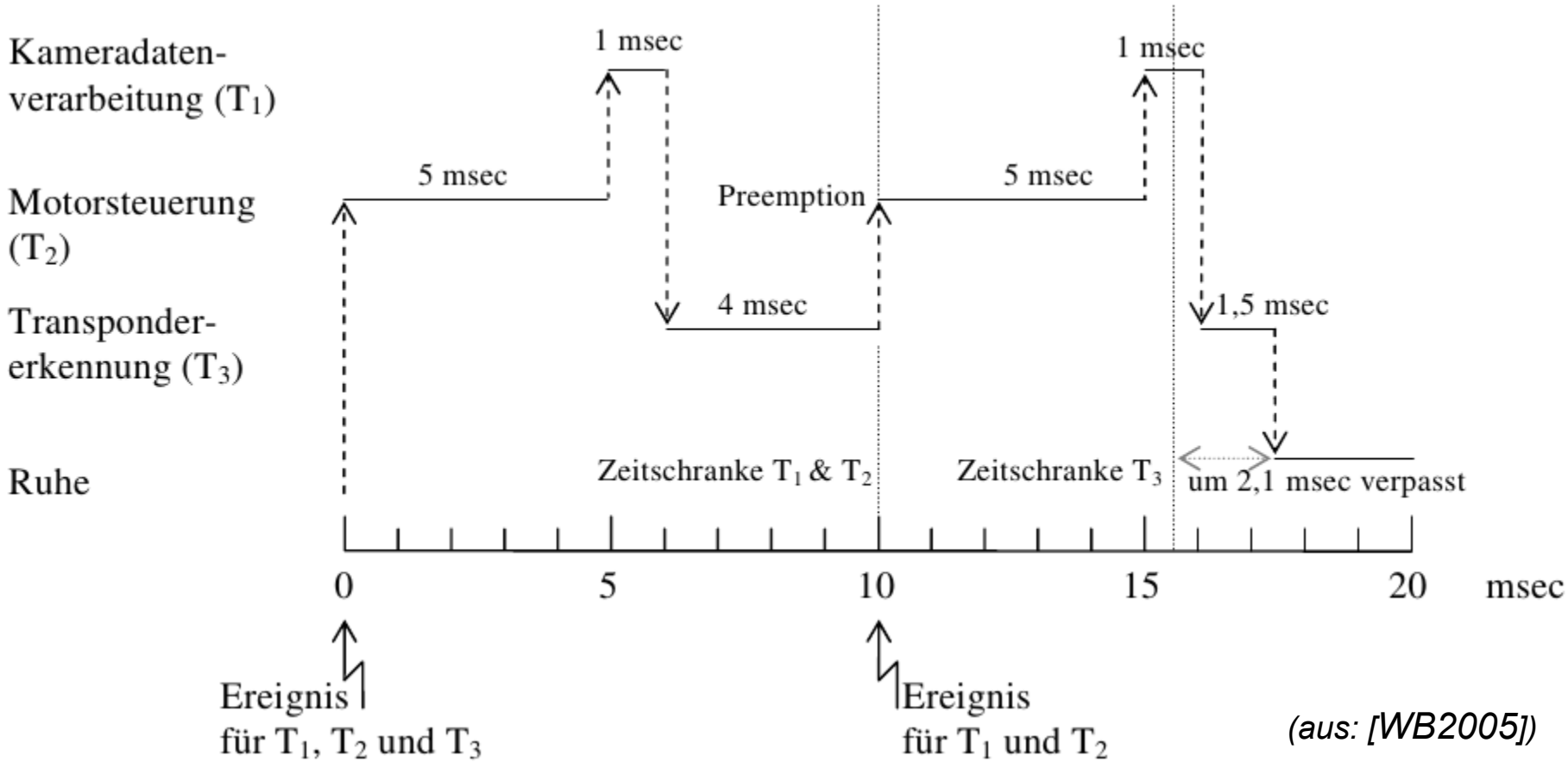
T_1 (Kameradatenverarbeitung):	mittlere Priorität
T_2 (Motorsteuerung):	hohe Priorität
T_3 (Transpondererkennung):	niedrige Priorität

FTS-Beispiel: Scheduling 1



(aus: [WB2005])

FTS-Beispiel: Scheduling 2



(aus: [WB2005])

Das Scheduling mit festen Prioritäten und Präemption ist kein optimales Scheduling-Verfahren. Dies gilt auch für nicht-präemptives Scheduling.

Obergrenze der Prozessorauslastung

Es kann eine Obergrenze der Prozessorauslastung berechnet werden, bis zu der RMS mit Präemption immer einen ausführbaren Schedule findet:

$$H_{\max} = n(2^{\frac{1}{n}} - 1) \qquad \text{n = Anzahl der Tasks}$$

Liegt H für eine Echtzeitanwendung unter H_{max}, so findet RMS einen Schedule, in dem alle Zeitbedingungen eingehalten werden.

Liegt H über H_{max}, ist die Ermittlung eines ausführbaren Schedules zwar immer noch möglich, kann aber nicht garantiert werden.

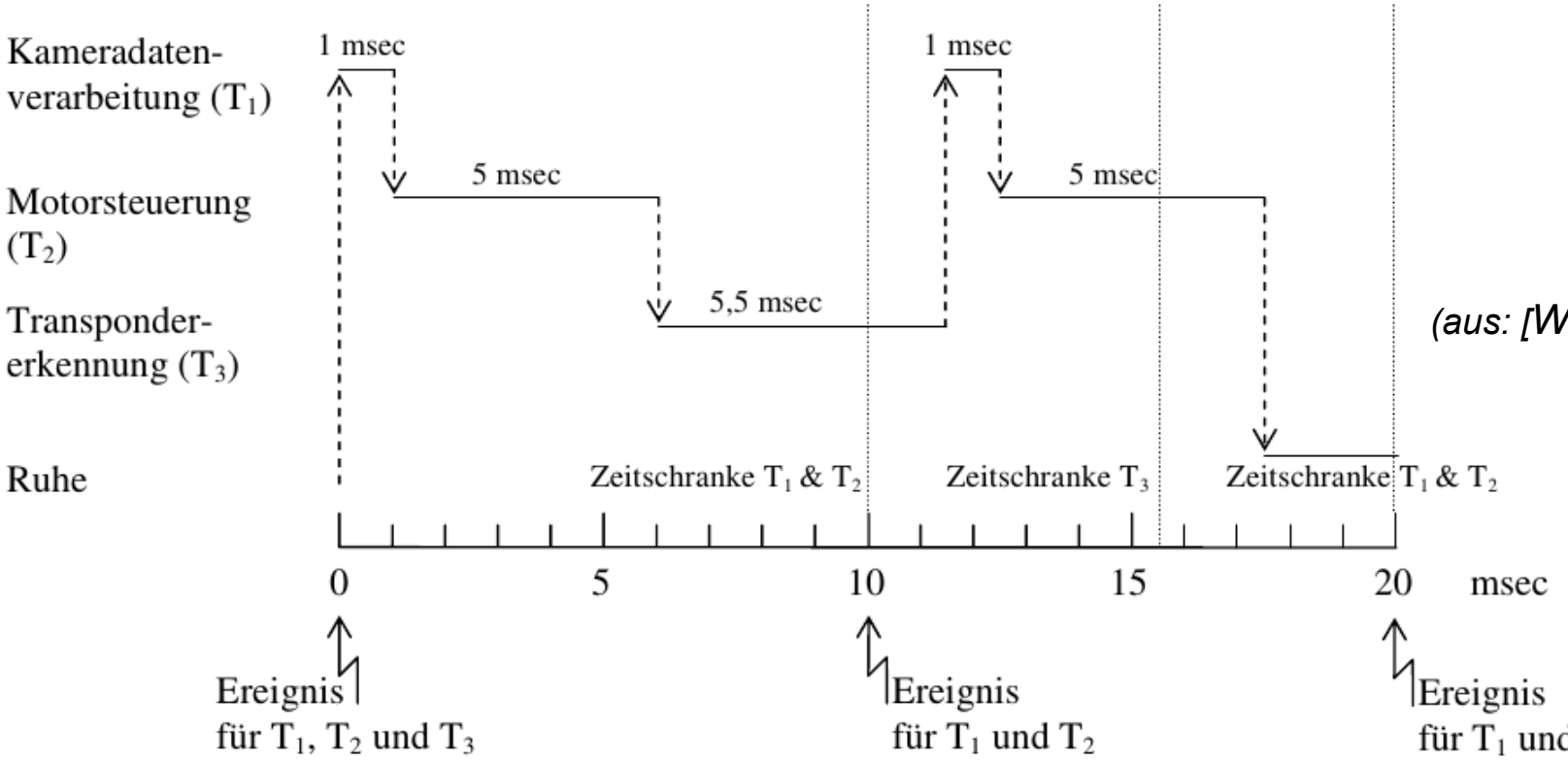
Für das FTS-Beispiel:

$$H_{max} = 3(2^{1/3} - 1) = 0{,}78 = 78\%$$

(vgl. H = 95,71%)

Earliest Deadline First (EDF)

Die Priorität eines Tasks berechnet sich aus der Nähe zu ihrer Zeitschranke, ändert sich also während des Ablaufs im Verhältnis zu den Prioritäten der anderen Tasks. Es handelt sich also um dynamisches Scheduling mit dynamischen Prioritäten. Fast immer wird die präemptive Variante verwendet, d.h. es kommt immer derjenige ablaufwillige Task zur Ausführung, dessen zeitlicher Abstand zur Zeitschranke am geringsten ist.



(aus: [WB2005])

Ist EDF ein optimales Scheduling?

Die Vermutung liegt nahe, da EDF das FTS-Problem mit H = 95,7% löst.

In der Tat ist EDF-Scheduling auf Einprozessorsystemen ein optimales Scheduling-Verfahren, d.h.

$$H_{max} = 100\%$$

Für periodische Tasks, bei denen die Zeitschranke der Periode entspricht, gilt:

$$\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$$

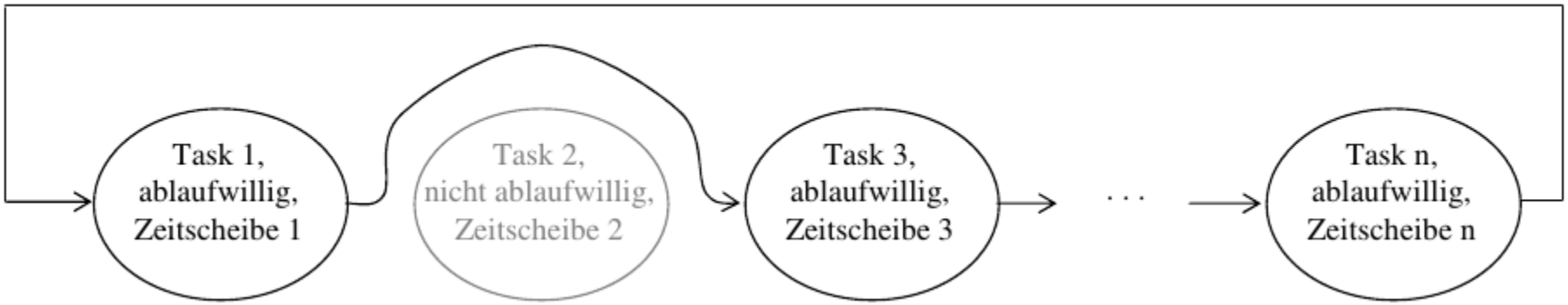
Nachteile von EDF:

- Höherer Rechenaufwand, da Prioritäten dynamisch berechnet werden.
- Nicht optimal bzgl. der Einhaltung von Datenraten. Daten können verloren gehen, wenn zwischen Ereignis und Ausführung des zugehörigen Tasks zu viel Zeit liegt (z.B. bei T_3). Geeignete Gegenmaßnahmen sind beispielsweise Hardware-Puffer, die eine ausreichende Menge von Daten zwischenspeichern können.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Time-Slice-Scheduling (Zeitscheibenverfahren)

Jeder Task erhält eine feste Zeitscheibe zugeteilt, die für jeden Task einzeln festgelegt werden kann. Die Reihenfolge der Taskausführung entspricht der Reihenfolge der ablaufwilligen Tasks in der Taskverwaltungsliste des Betriebssystems.



(aus: [WB2005])

Es handelt sich um präemptives dynamisches Scheduling ohne Prioritäten. Wenn die Zeitscheiben feinkörnig genug eingestellt werden können und die Zeitscheiben proportional zur Prozessorauslastung der einzelnen Tasks gewählt werden, nähert sich das Verfahren der Optimalität.

FTS-Beispiel mit Time-Slice-Scheduling

Prozessorauslastung:

T_1: $H_1 = e_1 / p_1$ = 1 ms / 10 ms = 10%

T_2: $H_2 = e_2 / p_2$ = 5 ms / 10 ms = 50%

T_3: $H_3 = e_3 / p_3$ = 5,5 ms / 15,4 ms = 35,714% ~ 36%

Zeitscheiben auf Proportionalitätsbasis 1% entspricht 0,05 ms:

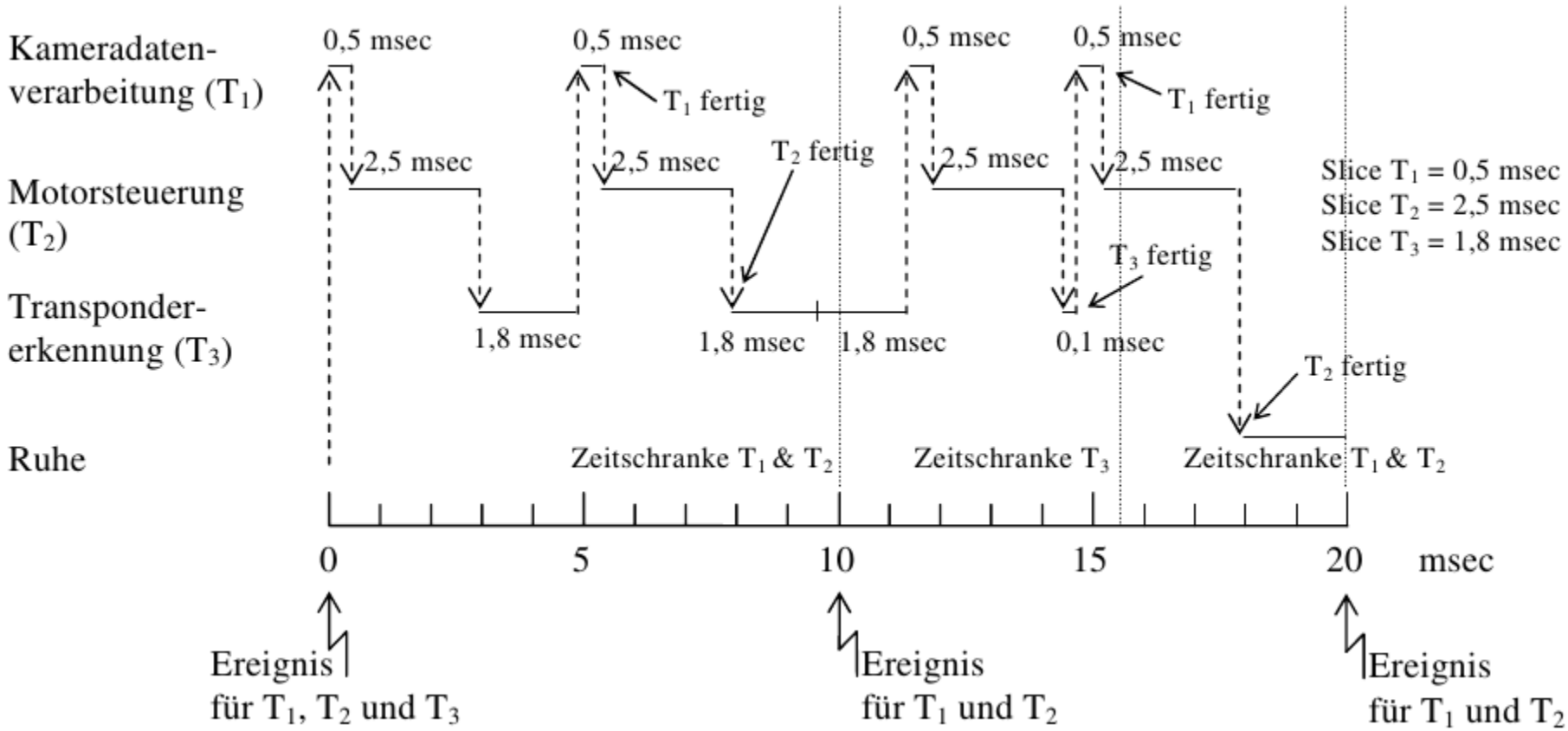
T_1: Zeitscheibe 0,5 ms (~10%)

T_2: Zeitscheibe 2,5 ms (~50%)

T_3: Zeitscheibe 1,8 ms (~36%)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

FTS: Time-Slice-Scheduling



(aus: [WB2005])

Alle Zeitschranken werden eingehalten.

Interessant ist der Zeitpunkt 9,6 ms: T3 erhält direkt die nächste Zeitscheibe zugeteilt, da kein anderer Task ablaufwillig ist.

Time-Slice-Scheduling: Bewertung

Vorteile:

- einfache Realisierung
- kein Aushungern von Tasks

Nachteile:

- bei zu grobkörnigen Zeitscheiben entsteht Verschnitt
- Je mehr Tasks rechenwillig sind, desto länger dauert es, bis ein bestimmter Task erneut aktiv wird.
- Wenn die Zeitscheibe eines Tasks abgelaufen ist und ein Ereignis für diesen Task trifft dann ein, kann die Reaktionszeit auf dieses Ereignis sehr lang sein. Das Einhalten von Datenraten kann hier wie bei EDF zu einem Problem werden.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

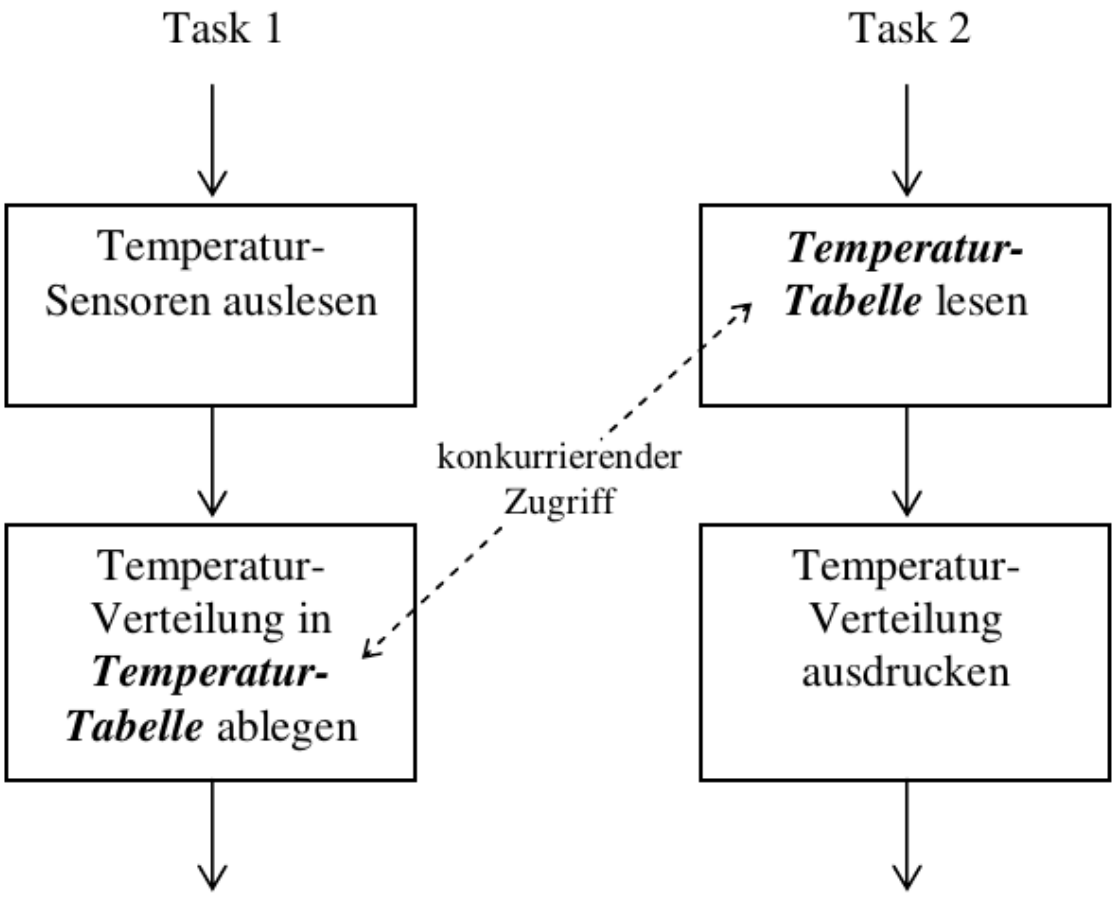
Synchronisation von Tasks

Sind Tasks nicht unabhängig voneinander, sondern benutzen gemeinsame Betriebsmittel, muss der auf sie erfolgende Zugriff der Tasks koordiniert werden. Gemeinsame Betriebsmittel sind:

- Daten. Mehrere Tasks greifen lesend und schreibend auf Variablen oder Datenfelder zu. Bei unkoordiniertem Zugriff können inkonsistente Zwischenzustände der Daten entstehen.
- Geräte. Benutzen mehrere Tasks gemeinsam Sensoren oder Aktoren, muss der koordinierte Zugriff sichergestellt werden, um z.B. widersprüchliche Kommandos an einen Schrittmotor zu vermeiden.
- Programme. Wenn mehrere Tasks gemeinsam die Dienste eines Programms (z.B. Gerätetreiber) in Anspruch nehmen, dürfen durch gleichzeitigen Zugriff keine inkonsistenten Zustände dieses Programms entstehen.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

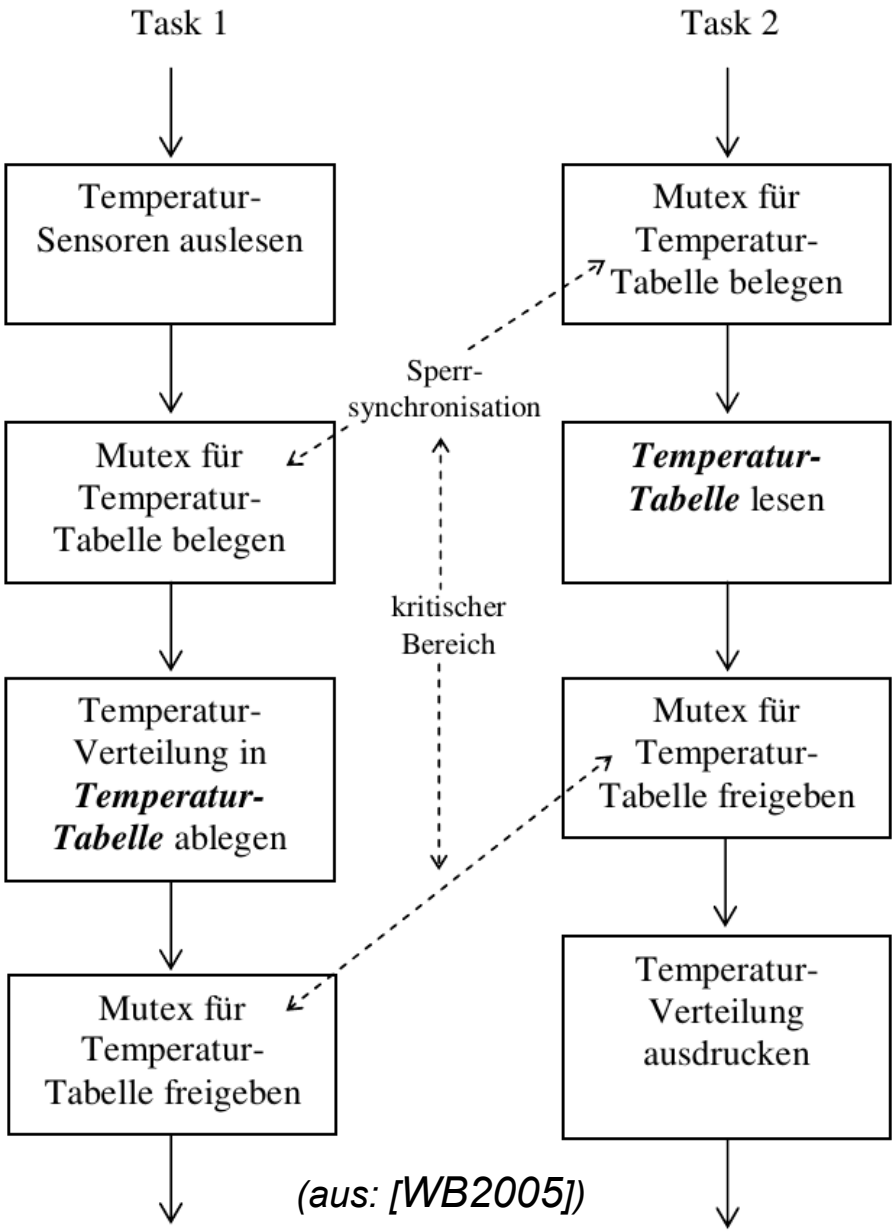
Beispiel: konkurrierender Taskzugriff



(aus: [WB2005])

Ohne Synchronisation kann es zu fehlerhaften Ergebnissen kommen, wenn Task 1 erst einen Teil der Tabelle aktualisiert hat und Task 2 währenddessen darauf zugreift. In diesem Fall erhält Task 2 neue und alte Temperaturwerte.

Sperrsynchronisation



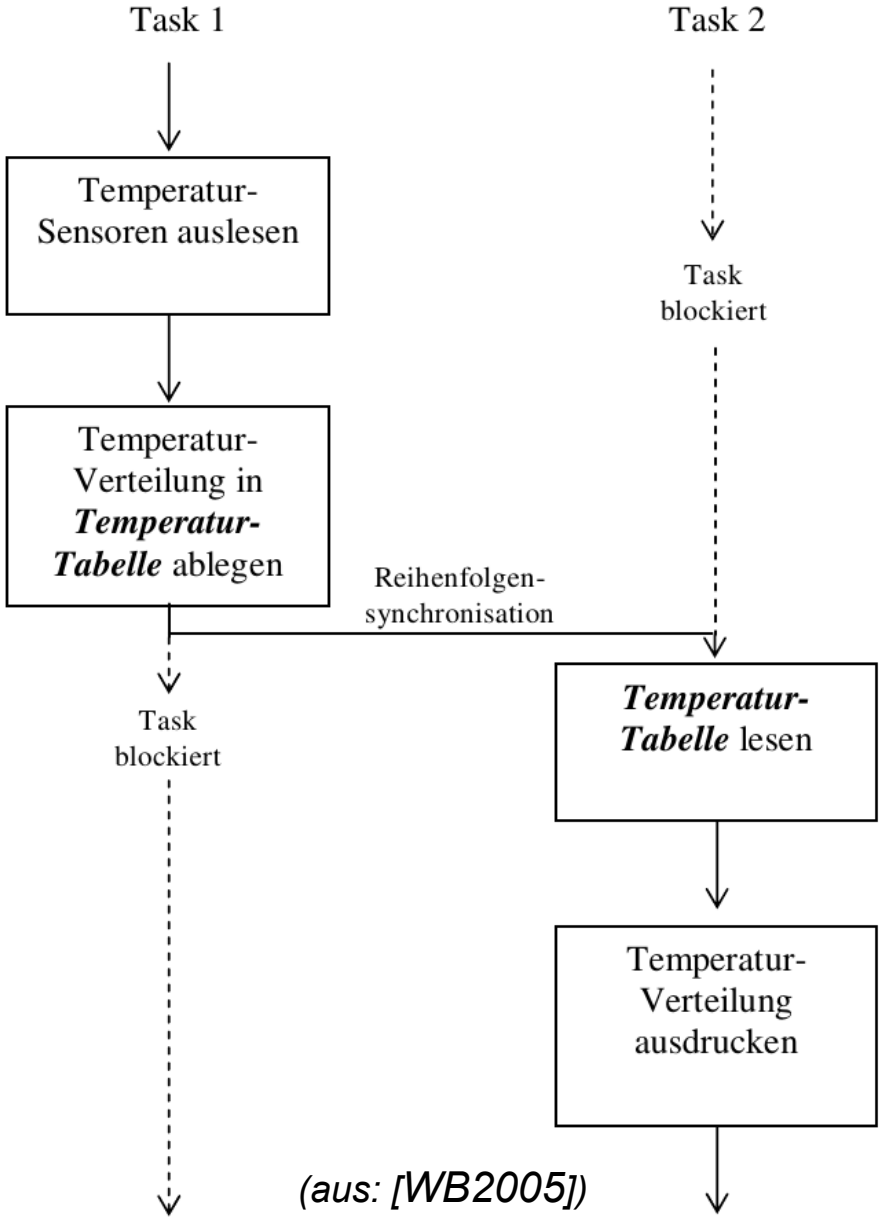
(aus: [WB2005])

Der wechselseitiger Ausschluss, (auch Mutual Exclusion oder Mutex) stellt sicher, dass zu einem Zeitpunkt nur ein Task auf ein Betriebsmittel zugreift.

Vor dem Zugriff versucht jeder Task, den Mutex zu belegen. Ist der Mutex bereits belegt, wird der Task so lange verzögert, bis der andere Task den Mutex wieder freigibt.

Bereiche, in denen Tasks synchronisiert werden müssen, heißen kritische Bereiche.

Reihenfolgensynchronisation



(aus: [WB2005])

Diese Art der Synchronisation wird auch Kooperation genannt.

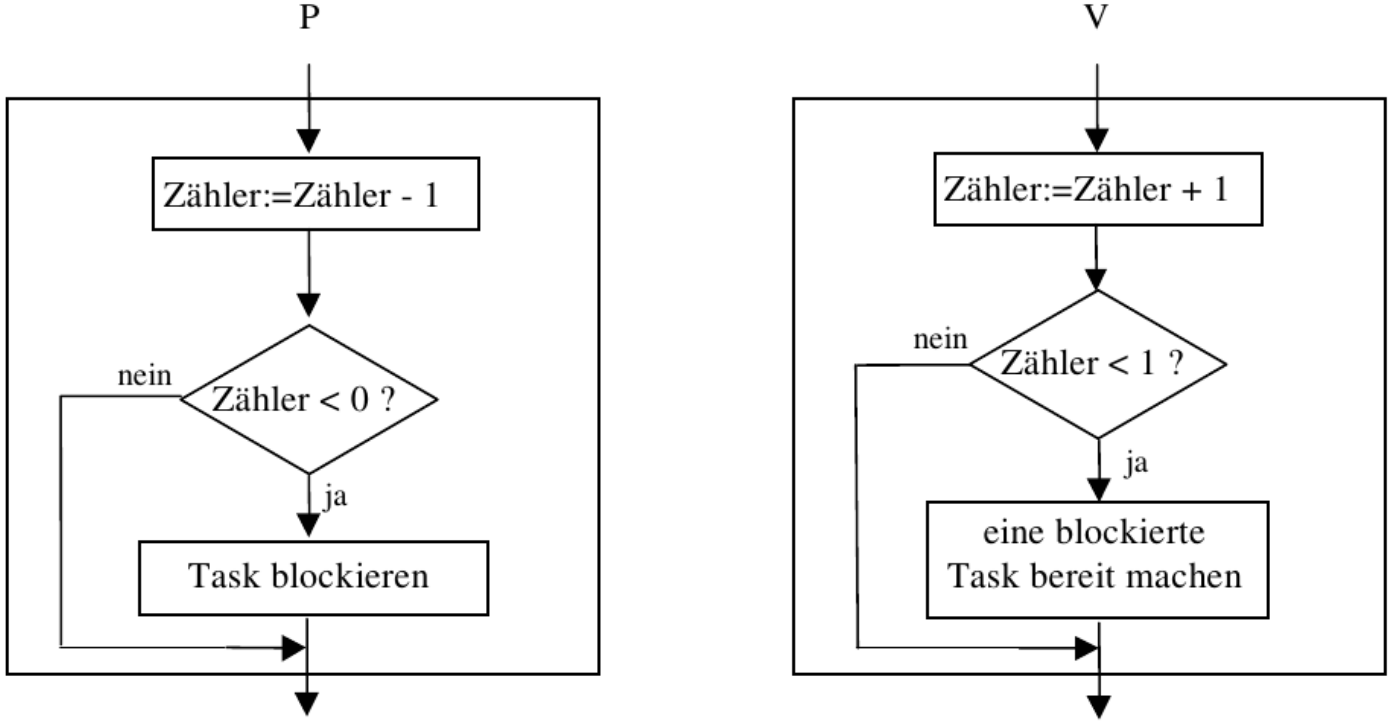
Die Sperrsynchronisation sagt nichts über die Reihenfolge der Zugriffe aus. Hier dagegen wird diese Reihenfolge exakt definiert.

Zuerst liest Task 1 die Temperaturen aus und trägt sie in die Tabelle ein. Erst dann darf Task 2 auf die Tabelle zugreifen und die Temperaturverteilung ausdrucken.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Semaphore

Das Synchronisationsmittel Semaphore besteht aus einer Zählvariable und zwei nicht unterbrechbaren Operationen Passieren (P) und Verlassen (V).



(aus: [WB2005])

Der initiale positive Wert eines Semaphors entspricht der Anzahl an Tasks, die sich zur gleichen Zeit in dem durch den Semaphore geschützten kritischen Bereich aufhalten dürfen.

Ein negativer Zählerwert entspricht der Anzahl an Tasks, die auf den Eintritt in den kritischen Bereich warten.

Semaphore

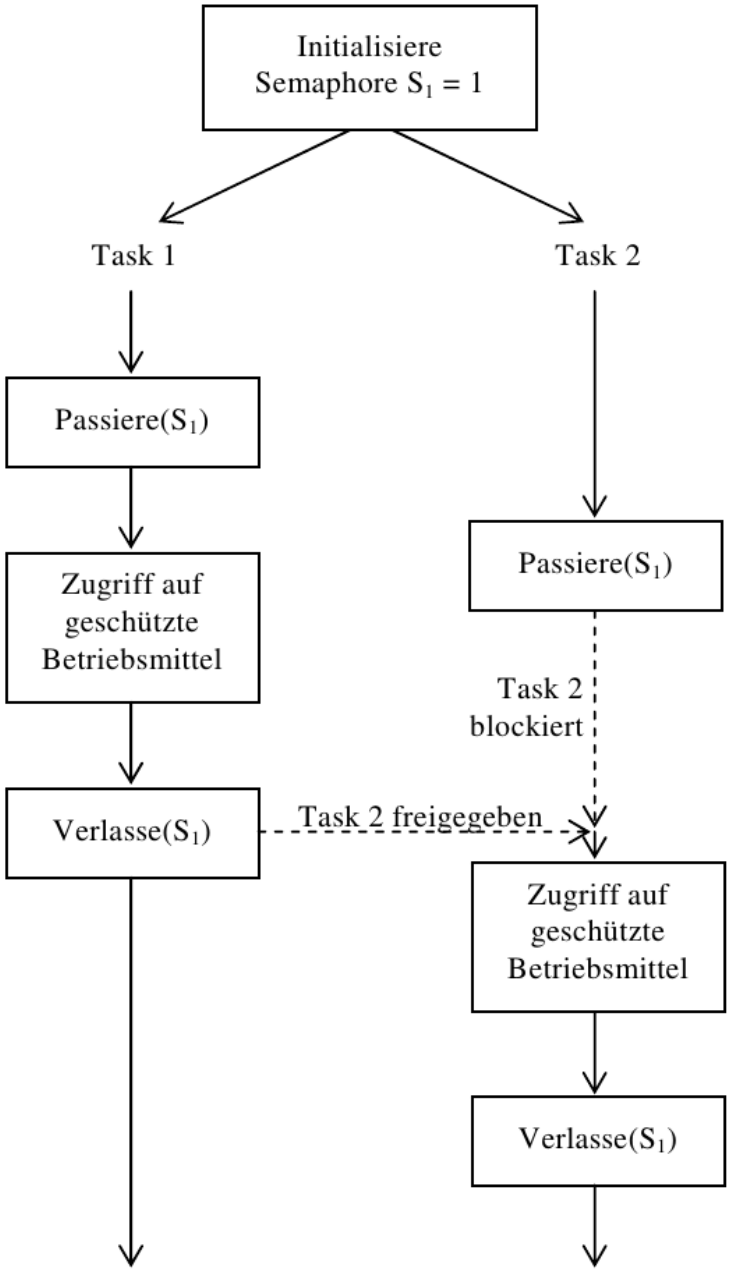
Die Operationen Passieren und Verlassen müssen atomar sein, d.h. P darf V nicht unterbrechen und umgekehrt.

In Echtzeitbetriebssystemen wird bei Verlassen eines Semaphors derjenige blockierte Task zur Ausführung gebracht, der die höchste Priorität besitzt. Bei Standardbetriebssystemen kann dies ein beliebiger Task sein, abhängig von der Implementierung des Betriebssystems.

Ein Semaphore kann zur Realisierung beider Synchronisationsarten verwendet werden. Bei der Sperrsynchronisation (Mutex) wir dabei die Zählvariable mit 1 initialisiert, so dass immer nur ein Task Zugriff auf das Betriebsmittel erhält, sich somit auch nur ein Task zur gleichen Zeit im kritischen Bereich aufhalten kann.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Mutual Exclusion mit einem Semaphore



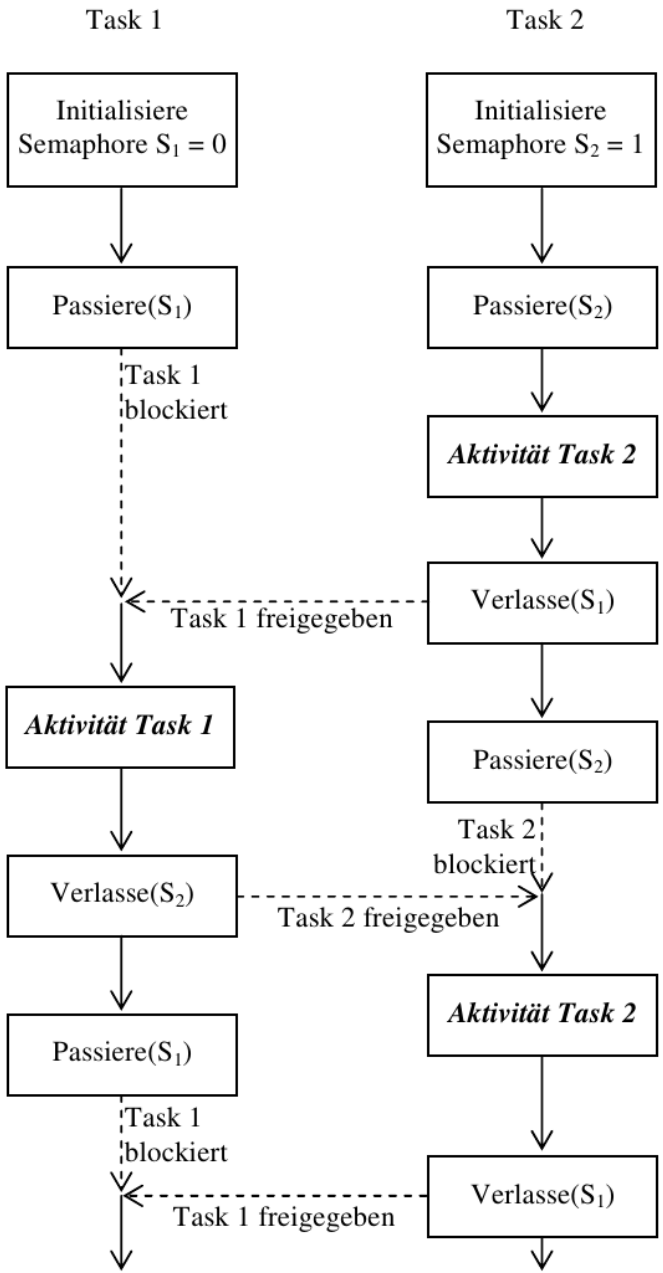
Bei der Sperrsynchronisation gibt immer derjenige Task den Semaphore wieder frei, der ihn zuvor auch belegt hat.

(aus: [WB2005])

Reihenfolgensynchronisation

Realisierung durch die wechselseitige Belegung mehrerer Semaphore.

Die Aktivitäten von Task 1 und Task 2 werden immer wechselseitig ausgeführt.



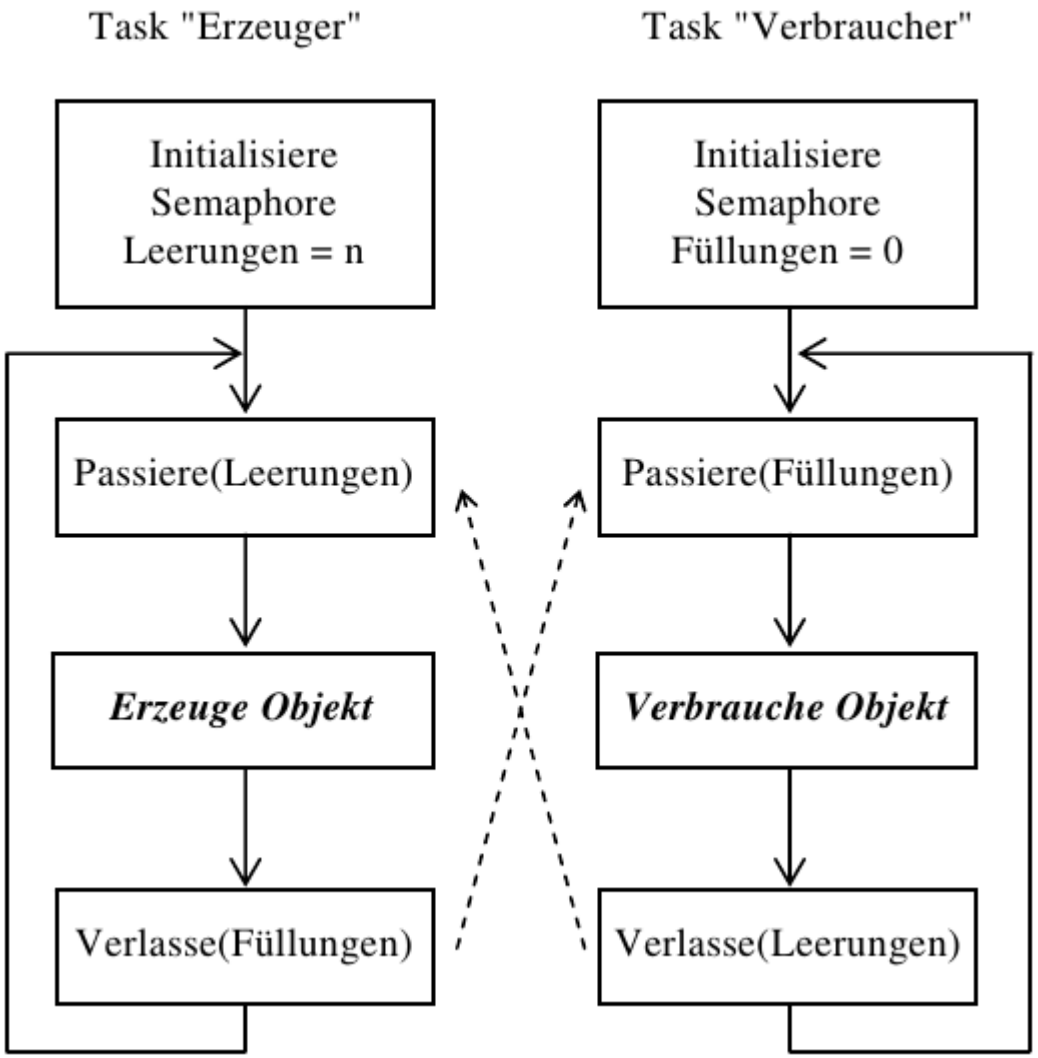
(aus: [WB2005])

Bei der Reihenfolgen-synchronisation kann die Freigabe eines Semaphors auch durch einen anderen Task als denjenigen erfolgen, der den Semaphore belegt hat.

Producer-Consumer-Synchronisation

Der Semaphore „Leerungen“ gibt die Anzahl der gerade freien Plätze an.

Der Semaphore „Füllungen“ gibt die Anzahl der erzeugten Objekte an, die dem Verbraucher zur Verfügung stehen.



(aus: [WB2005])

Verklemmungen

Bei der Synchronisation von Tasks kann es zu Verklemmungen kommen. Die Fortführung eines oder mehrerer Tasks wird blockiert.

Beispiel Verkehrskreuzung mit Rechts-vor-links-Vorfahrt: Kommen aus allen vier Richtungen gleichzeitig Fahrzeuge an, muss jedes auf das rechts von ihm stehende Fahrzeug warten. Nur ein Bruch der Vorfahrtsregel löst die Situation.

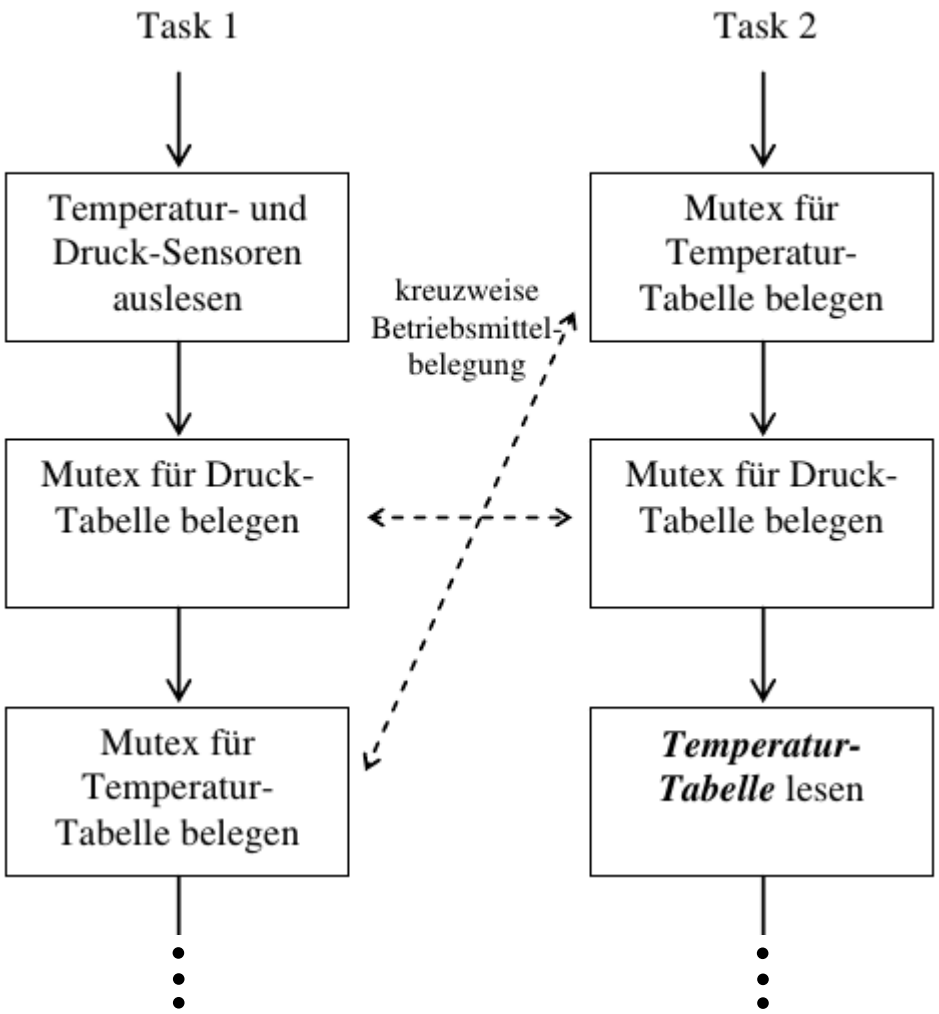
In der Taskverwaltung unterscheidet man zwei Typen von Verklemmungen:

- Deadlocks (Blockierung, Deadly Embrace)
- Livelocks (Starvation, Aushungern)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Deadlock

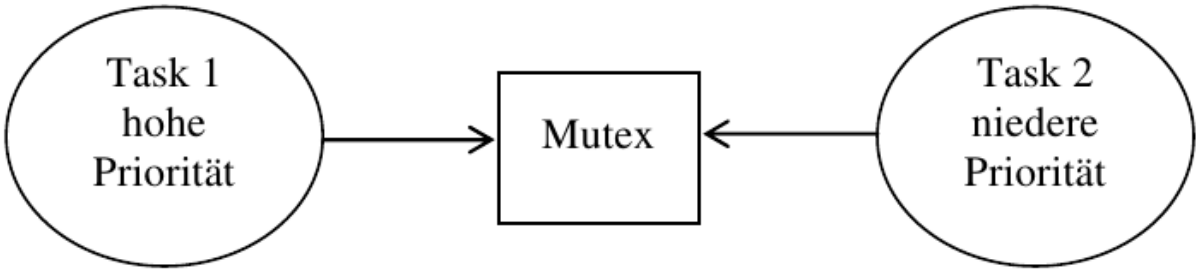
Mehrere Tasks warten auf die Freigabe von Betriebsmitteln, aber blockieren sich gegenseitig. Sie warten auf Ereignisse, die nicht mehr eintreten können. Die Folge ist ein Stillstand der Tasks.

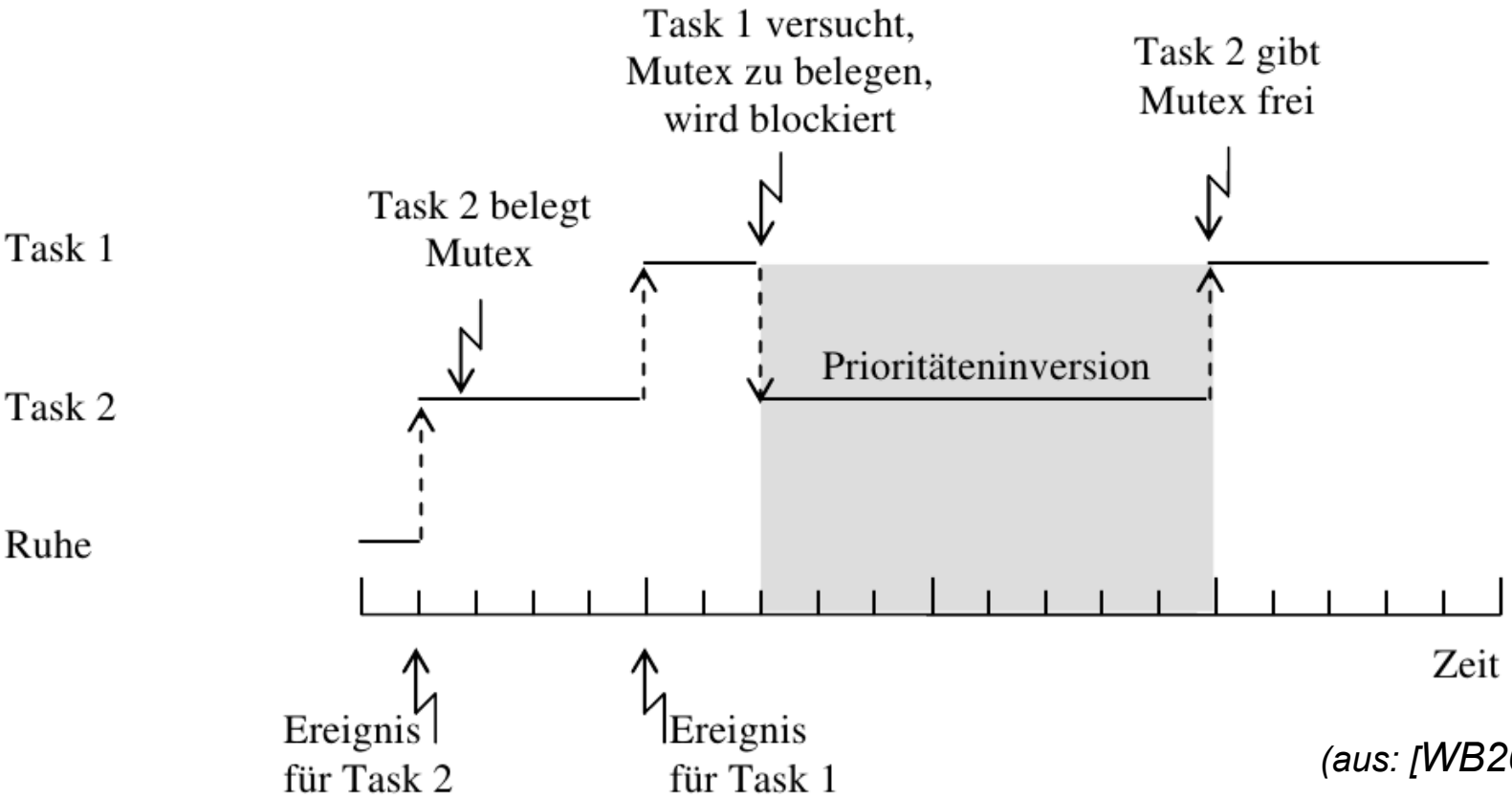


Sollen mehrere Betriebsmittel gleichzeitig geschützt werden, muss dies von allen betroffenen Tasks in genau derselben Reihenfolge getan werden.

(aus: [WB2005])

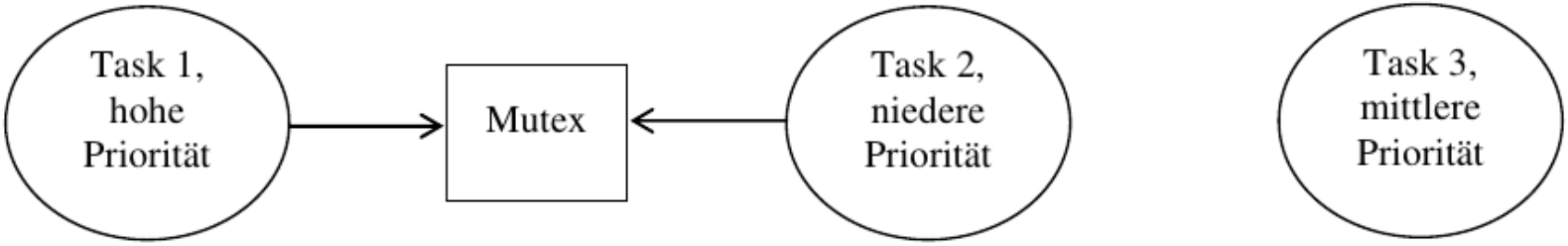
Prioritäteninversion

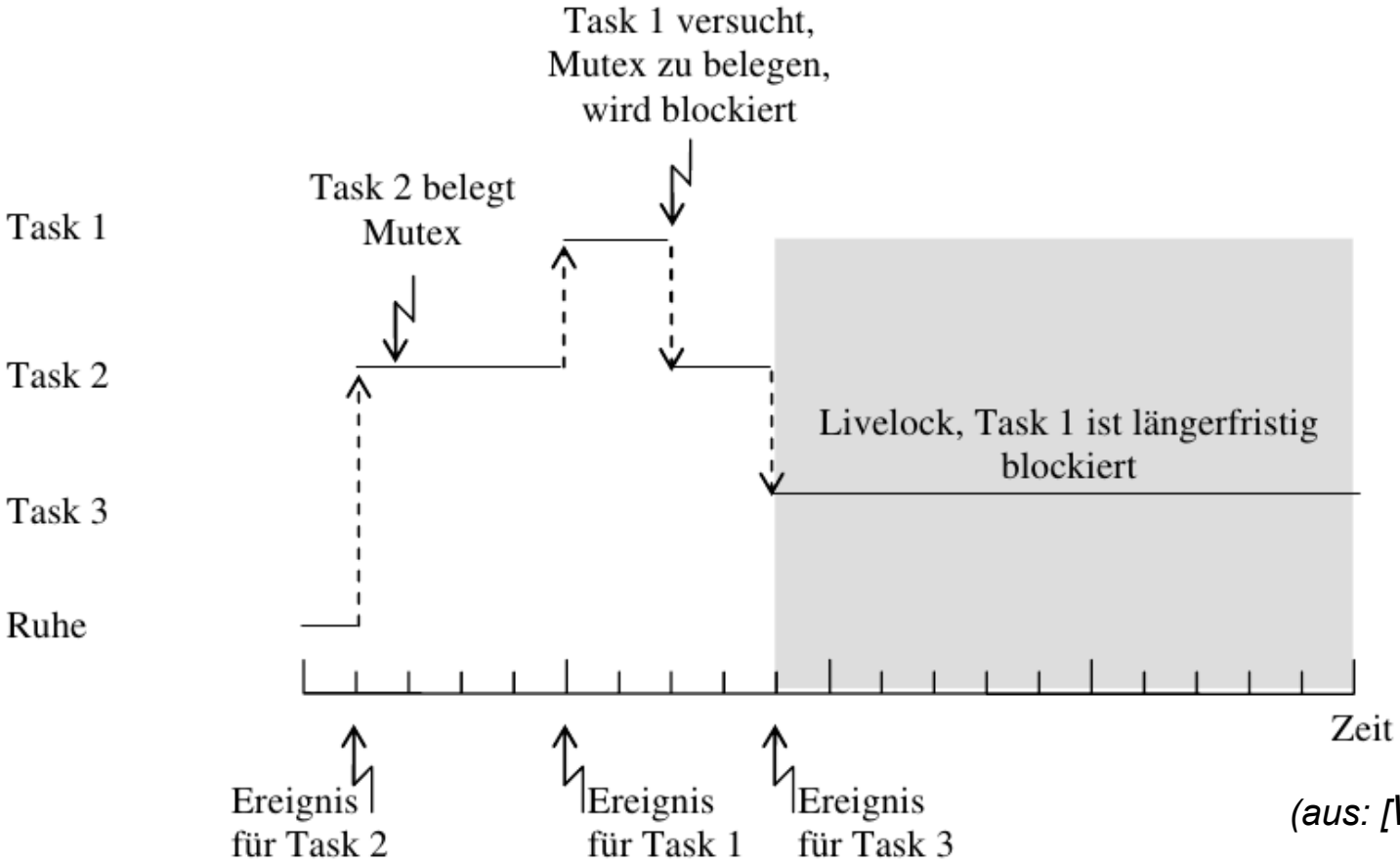




(aus: [WB2005])

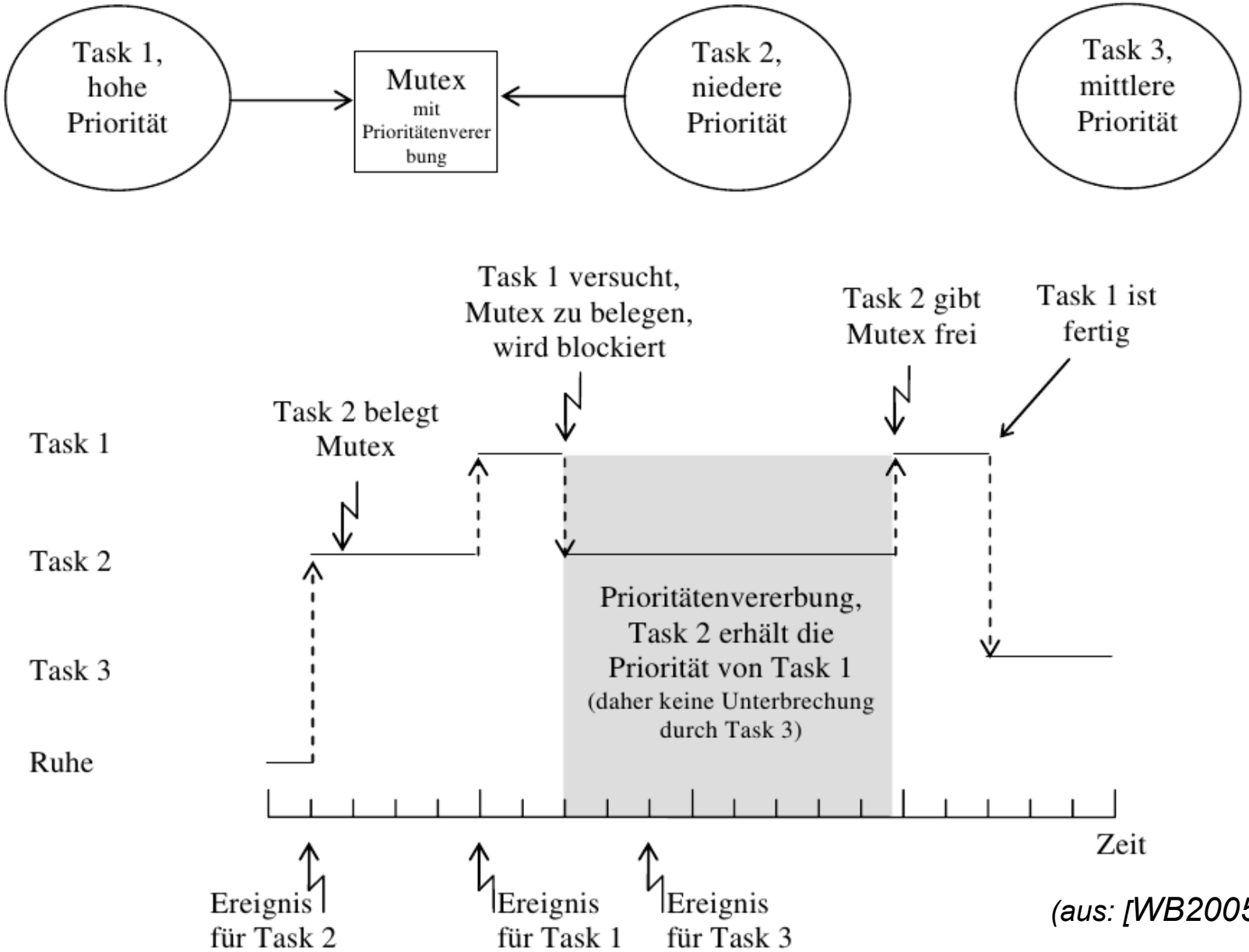
Livelock durch Prioritäteninversion





(aus: [WB2005])

Prioritätenvererbung



(aus: [WB2005])

Task-Kommunikation

Synchronisation und Kommunikation von Tasks gehören eng zusammen:

- Synchronisation entspricht einer Art informationsloser Kommunikation.
- Kommunikation kann zur Synchronisation benutzt werden. Wenn z.B. ein Task auf eine Botschaft wartet und blockiert, kann er beim Eintreffen ablaufwillig werden und dann den Inhalt der Botschaft bearbeiten.

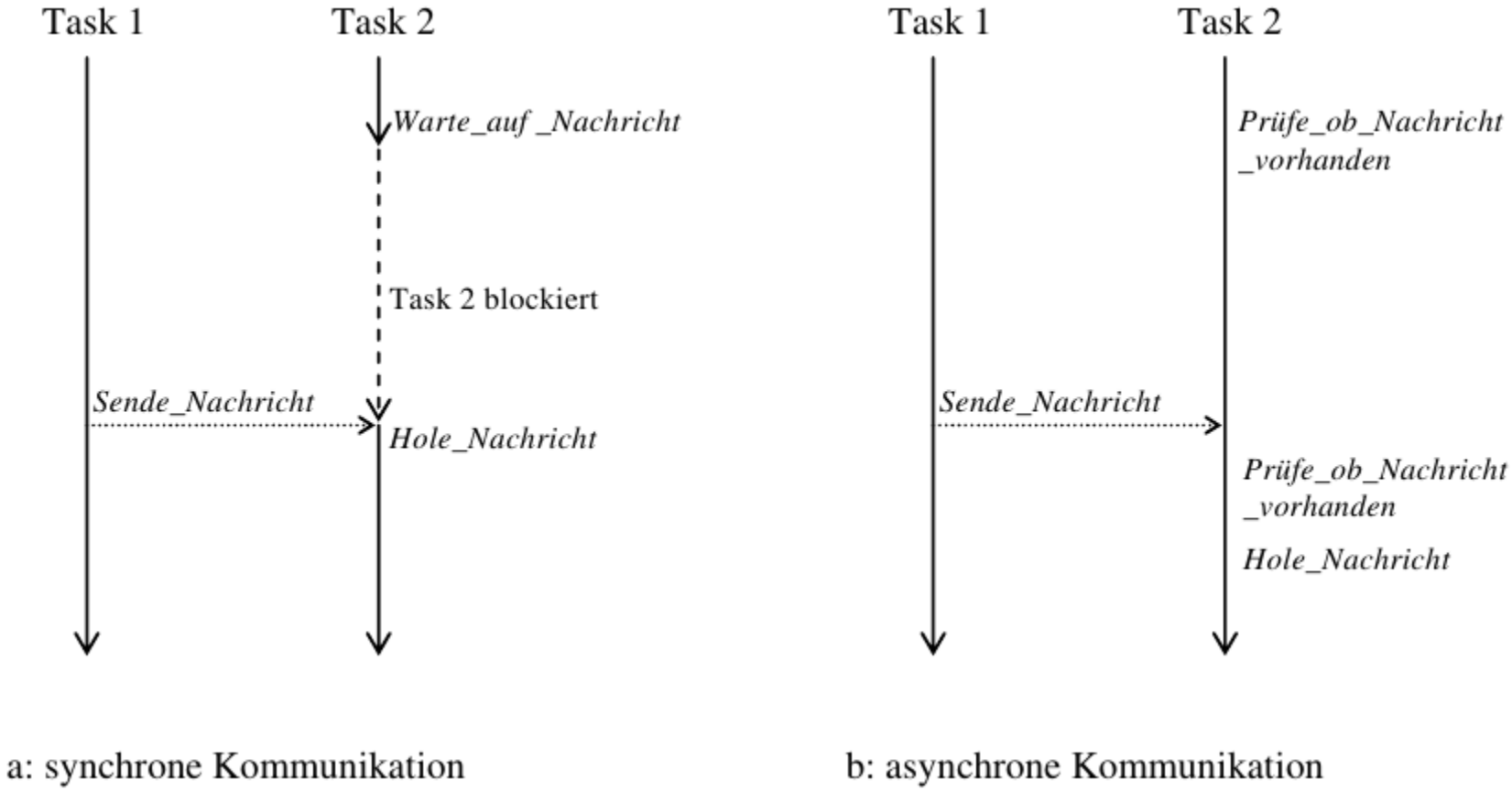
Es gibt zwei Varianten der Task-Kommunikation:

- Gemeinsamer Speicher: bevorzugte Variante in lokalen Systemen
- Nachrichten: Variante in verteilten Systemen

Die prioritätsbasierte Kommunikation ermöglicht die Vergabe von Prioritäten an die einzeln übergebenen Botschaften, d.h. Informationen hoher Priorität können solche mit niedrigerer Priorität überholen. So wird eine Prioritäteninversion vermieden, die auftreten kann, wenn Botschaften für unterschiedlich priorisierte Tasks über ein und denselben Kommunikationskanal übertragen werden.

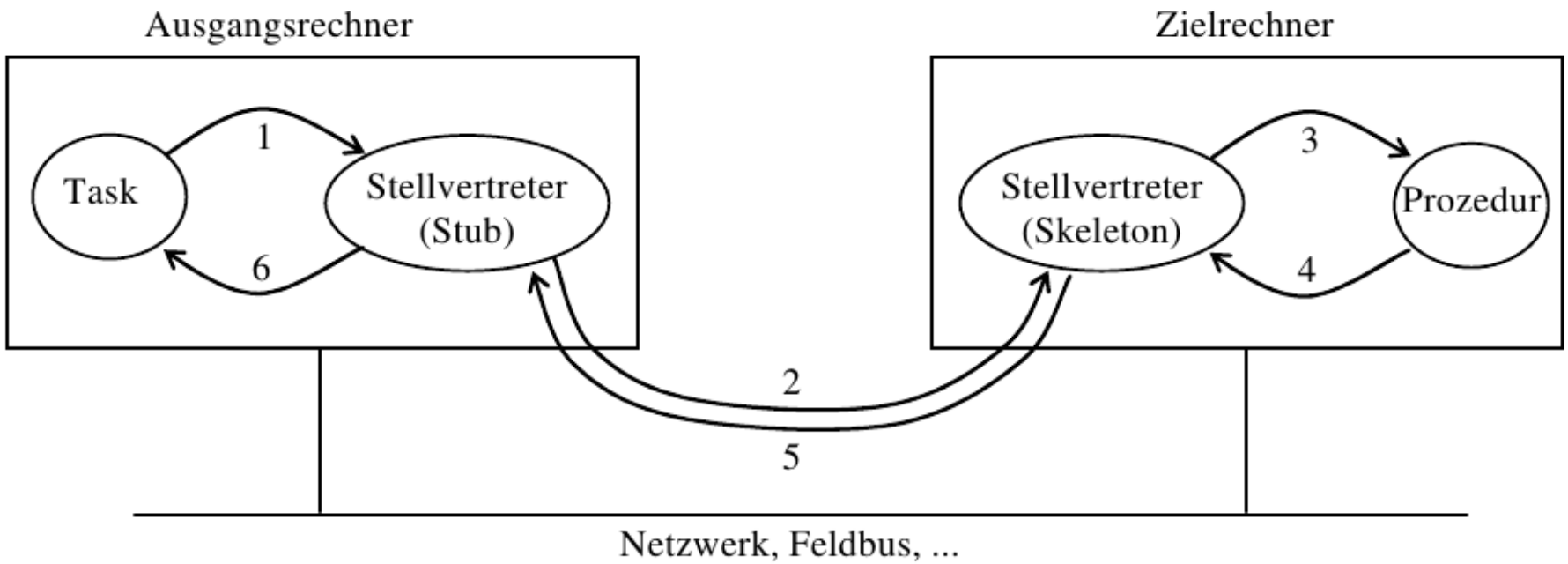
© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Zeitliche Koordination der Task-Kommunikation



(aus: [WB2005])

Remote Procedure Call (RPC)



(aus: [WB2005])

Kapitel 6: Echtzeitbetriebssysteme

6.1 Einleitung

6.2 Schichtenmodelle

6.3 Taskverwaltung

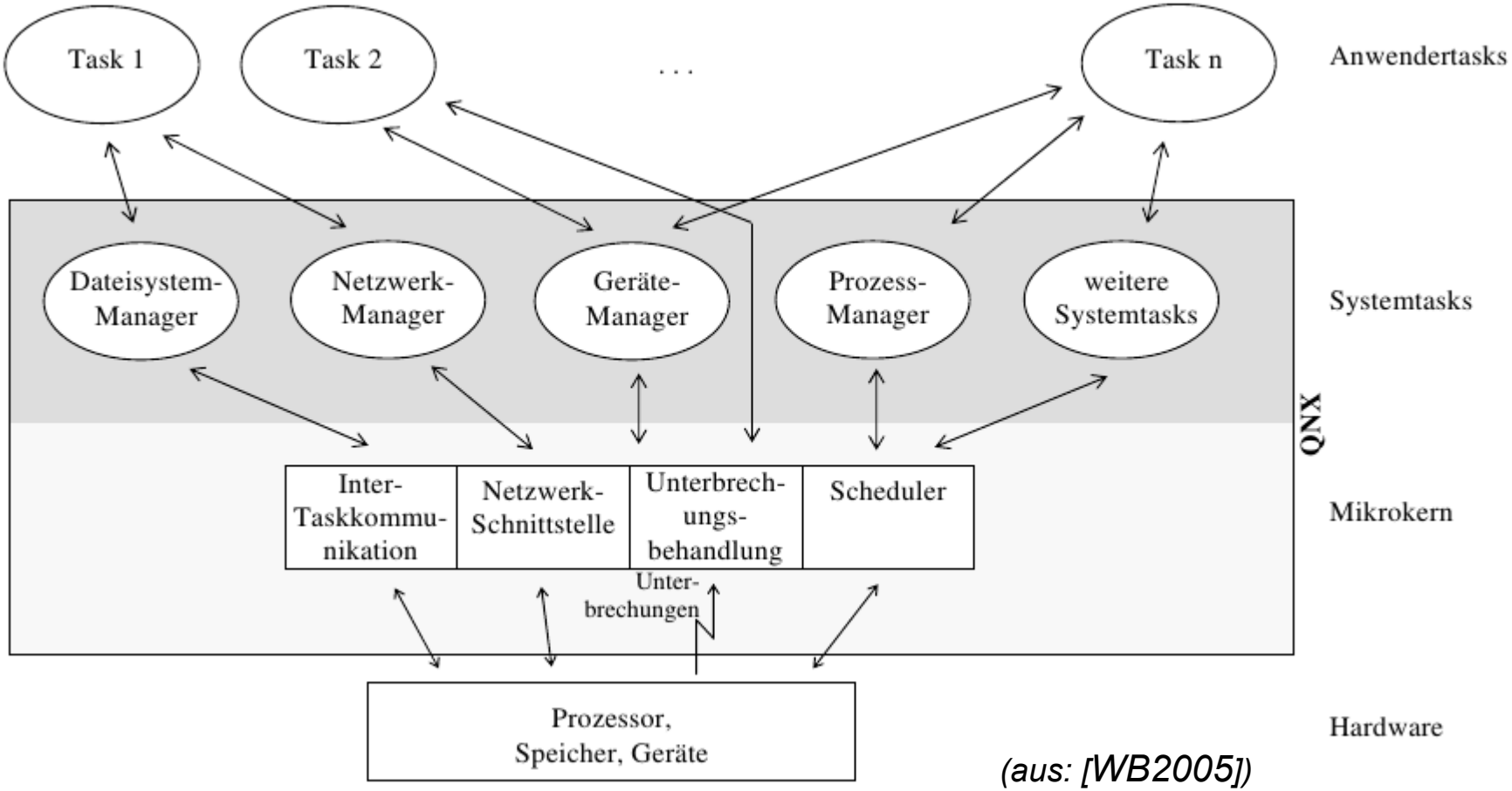
6.4 QNX als Beispiel für ein industrielles RTOS

6.5 Echtzeitbetriebssystem Linux/Xenomai

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

QNX

- besitzt einen schlanken Mikrokern von ca. 8 kB
- Systemtasks werden wie Anwendertasks behandelt.
- Der Scheduler kennt das FIFO-, FPP-, FPN- und Timeslice-Scheduling.



(aus: [WB2005])

Kapitel 6: Echtzeitbetriebssysteme

6.1 Einleitung

6.2 Schichtenmodelle

6.3 Taskverwaltung

6.4 QNX als Beispiel für ein industrielles RTOS

6.5 Echtzeitbetriebssystem Linux/Xenomai

Linux und Xenomai

Linux (genauer GNU/Linux) ist ein Standardbetriebssystem und in seiner üblichen Einsatzform auf Desktop-PCs, Servern und Mobilgeräten nicht echtzeitfähig. Um Linux echtzeitfähig zu machen, sind tiefgreifende Veränderungen am Betriebssystem notwendig, z.B. durch die Echtzeiterweiterung Xenomai.

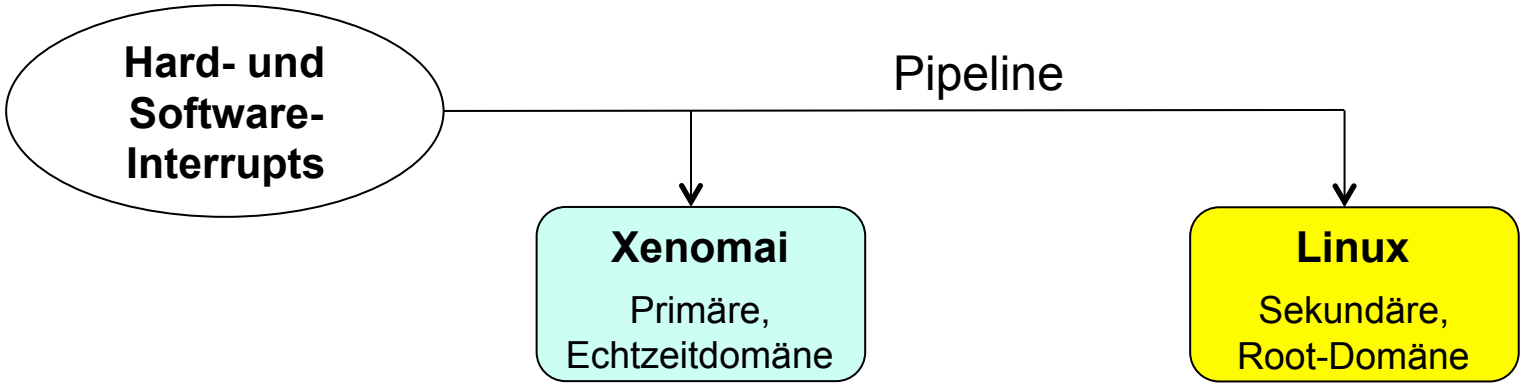
Xenomai verfolgt den Dual-Kernel- oder Co-Kernel-Ansatz:

- Der Linux-Kernel wird so weit modifiziert, dass er nahtlos mit dem Xenomai-Kernel zusammenarbeitet.
- RT-Tasks verwaltet der Xenomai-Scheduler, die sonstigen Tasks bleiben unter der Kontrolle des Linux-Schedulers.
- Xenomai hat jederzeit eine höhere Priorität als der Linux-Kernel. Weder der Linux-Scheduler noch die Linux-Interrupt-Behandlung dürfen Xenomai beeinträchtigen. Dagegen ist der Linux-Kernel für Xenomai vollständig unterbrechbar. Linux wird also zu einem untergeordneten Teil des Gesamtsystems, dem in wichtigen Bereichen die Kontrolle über die Hardware entzogen wird.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

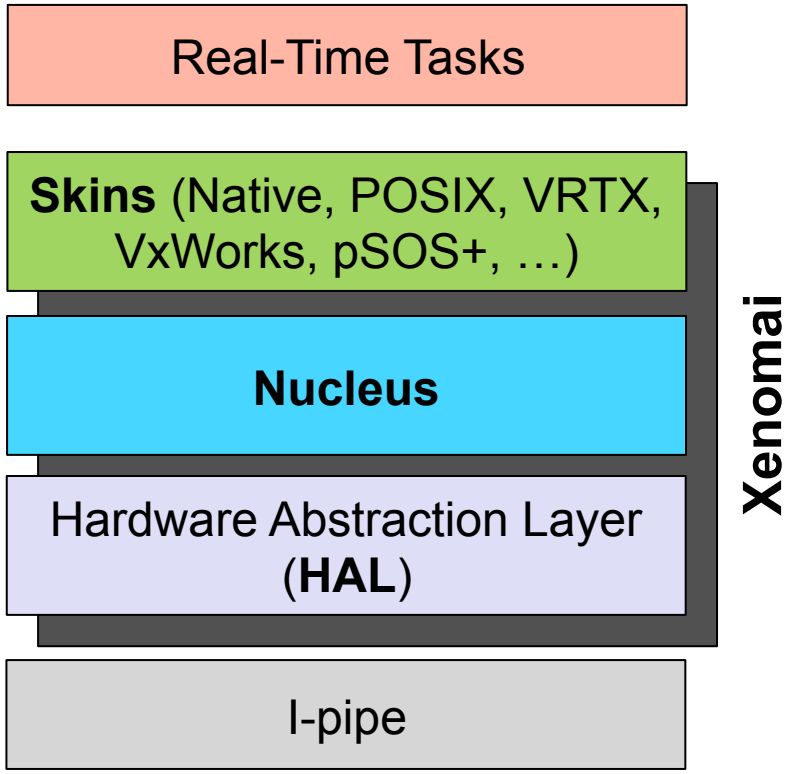
Die Interrupt-Pipeline (I-pipe)

- Grundlage für die Echtzeitfähigkeit von Xenomai
- modifiziert den Kernel, übernimmt den direkten Zugriff auf einige elementare Hardware-Funktionen (hauptsächlich solche, die mit Interrupts zu tun haben) und virtualisiert sie für den Rest des Systems.
- Der Dual-Kernel wird in Domänen eingeteilt:



- Xenomai hat immer das Vorrecht, Interrupts primär zu empfangen. Interrupts, für die sich Xenomai nicht registriert hat, werden für die Behandlung durch Linux vorgemerkt, die erst dann stattfindet, wenn in der Xenomai-Domäne keine Aufgaben mehr zu erfüllen sind.
- Das hardwaremäßige Abschalten von Interrupts, das der Linux-Kernel normalerweise nutzt, wirkt sich nur auf die sekundäre Domäne aus.

Architektur von Xenomai

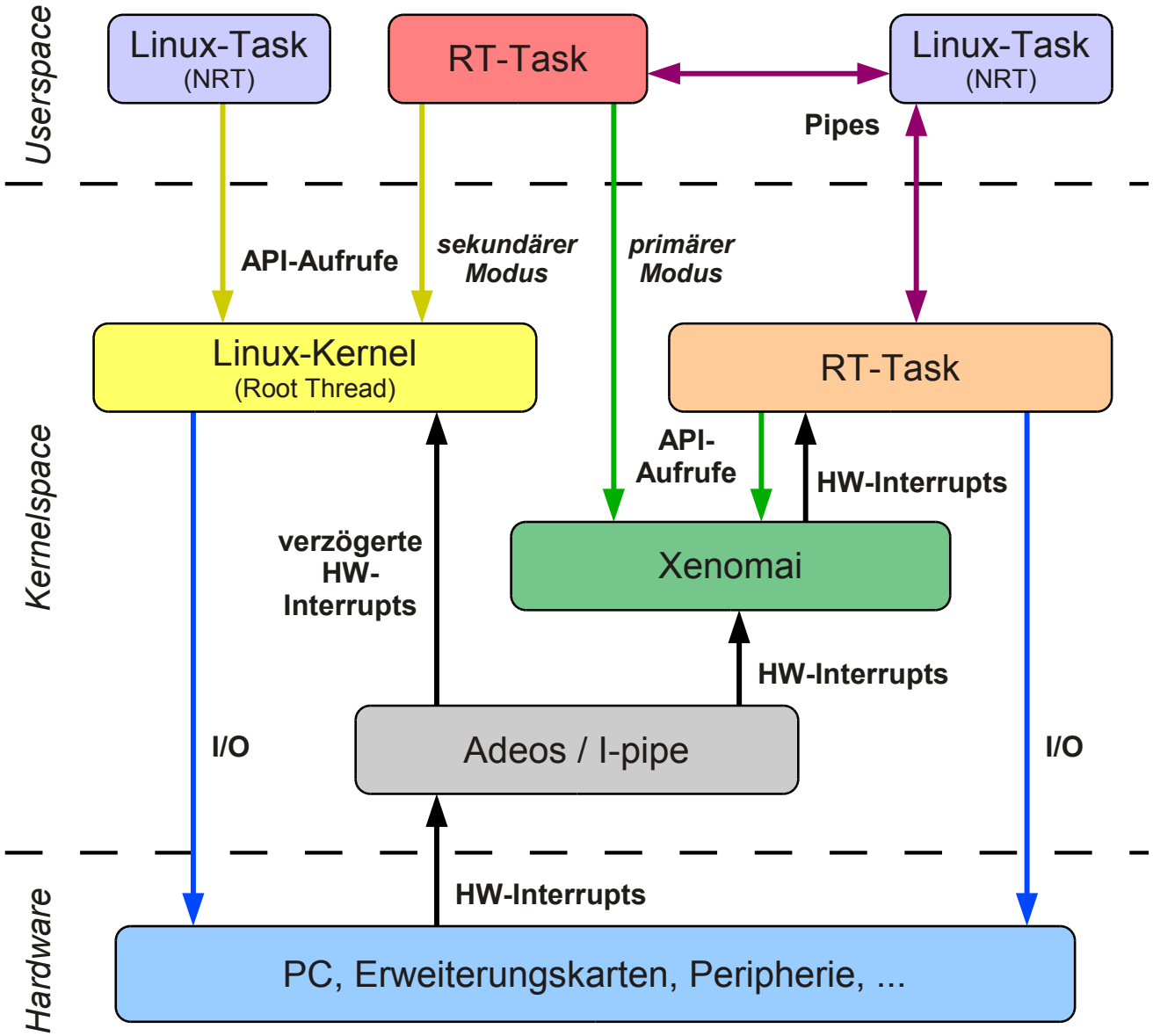


Eine Besonderheit von Xenomai sind die Skins: verschiedene RT-APIs, die die Portierung von Applikationen, die für ein anderes RTOS geschrieben wurden, erleichtern. Daneben gibt es auch einen eigenen Native-Skin und einen POSIX-Skin.

Der Nucleus genannte RTOS-Kern stellt generische Echtzeitfunktionen (Scheduler, Synchronisation, Timer, RT-Speicherallokation, Task-Kommunikation, …) zur Verfügung, die die Skins nutzen und weiter ausprägen können.

Der Nucleus zeichnet sich durch Hardware-Neutralität aus, alle hardware-abhängigen Details sind im HAL gekapselt.

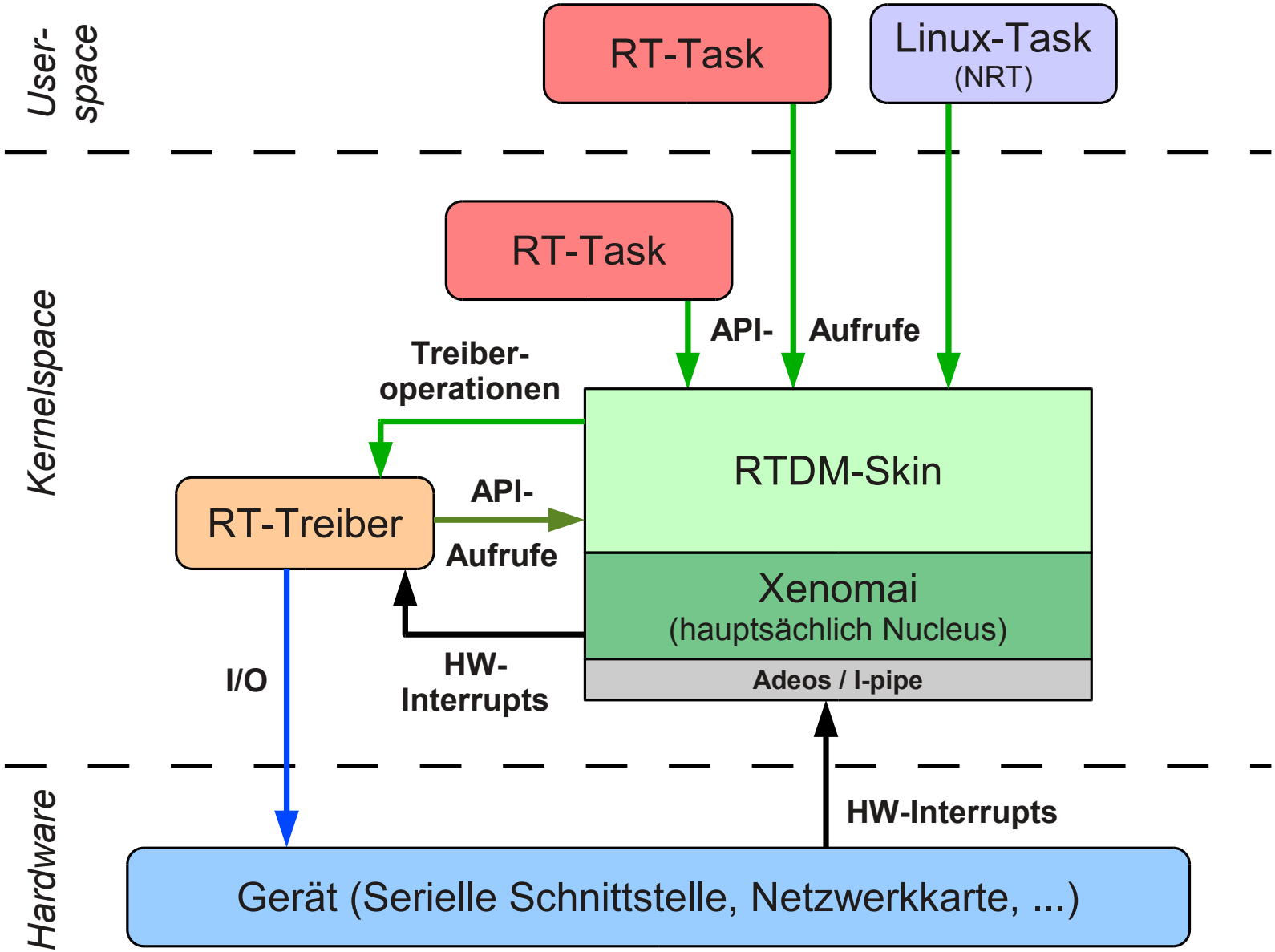
Integriertes Linux/Xenomai-System



Die Kommunikation zwischen RT- und NRT-Tasks ist auf zweierlei Weise möglich:

1. RT-Pipes
2. Jeder RT-Task kann zeitweilig vom Linux-Scheduler verwaltet werden, läuft dann aber auch nicht unter Echtzeit-bedingungen

Real-Time Driver Model (RTDM)



Xenomai im realen Einsatz

Am RTS werden alle autonomen Service-Roboter mit Xenomai betrieben. Für einen ersten Eindruck:

http://www.youtube.com/user/unihannoverrts

Vierbeiniger Roboter aus Genua:

http://www.golem.de/news/roboter-hyq-wird-zum-robotischen-zentaur-1205-91829.html

In der Industrie wird Xenomai ebenfalls von vielen Firmen benutzt, z.B. von Siemens, Airbus, Heidelberger Druckmaschinen, aber auch von der NASA.

Zusammenfassung

Es wurden die wesentlichen Aspekte eines Echtzeitbetriebsystems vorgestellt:

- Schichtenmodelle mit API, Taskverwaltung, Betriebsmittelverwaltung, Gerätetreibern, Mikrokernel, usw.
- Prozesse, Threads und Tasks
- Prozessorauslastung, Zeitparameter und Prioritäten
- statisches und dynamisches Echtzeitscheduling (FIFO, FPP, FPN, RMS, EDF, TSS ...)
- Synchronisation von Tasks (Semaphore, Mutual Exclusion, Reifenfolge, Producer-Consumer)
- Deadlock, Prioritäteninversion, Prioritätenvererbung
- Task-Kommunikation
- RTOS-Beispiele: QNX, Linux/Xenomai

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

7. Echtzeitkommunikation

7.1 Steuerungsnetzwerke

7.2 Netzwerkgrundlagen

7.3 Feldbusse

7.4 Interbus nach DIN 19258

7.5 CAN – Controller Area Network

7.6 Profibus – Process Field Bus nach DIN 19245

7.7 RTnet – Real-Time Ethernet

7.8 RACK – Robotics Application Construction Kit

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Echtzeitkommunikation

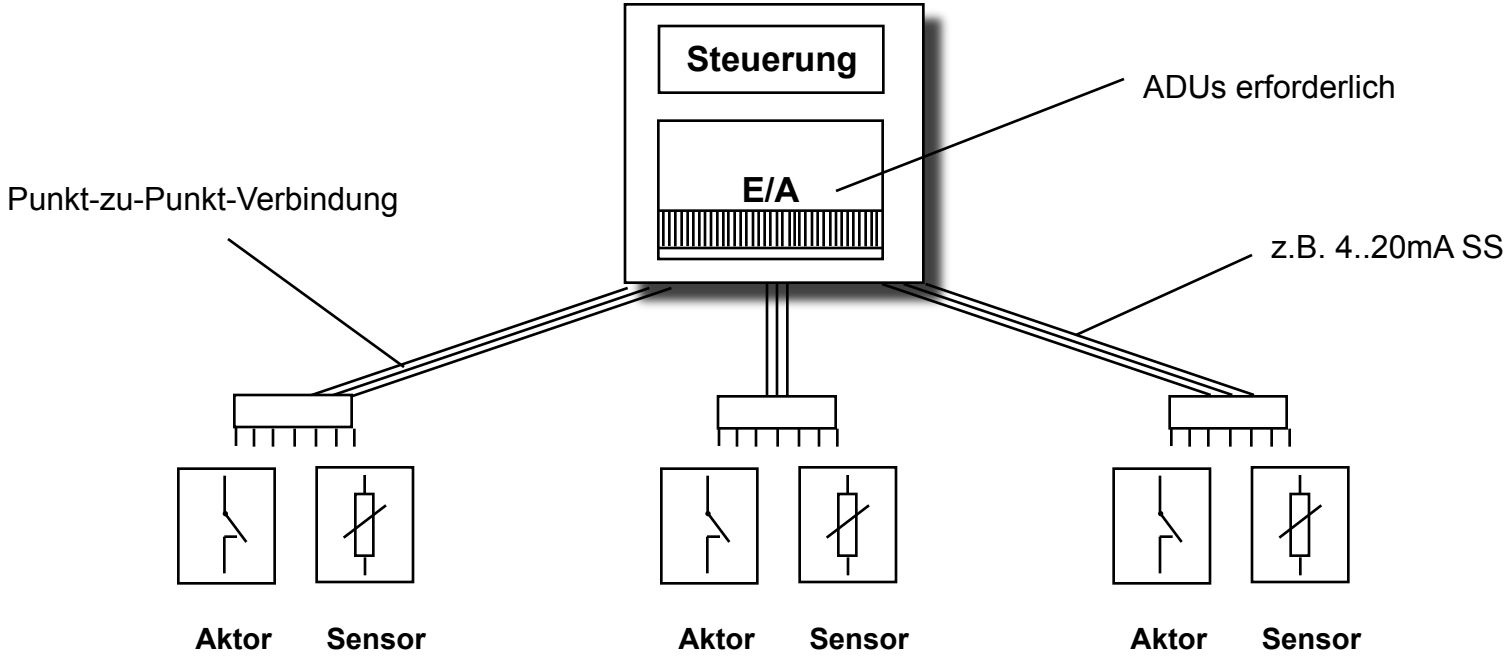
Die Realisierung von verteilten Echtzeitsystemen bringt es mit sich, dass auch die Kommunikation im Netzwerk in vorab bestimmbaren, festen Zeitspannen erfolgen kann.

Eine Vielzahl von Feldbussen wurden entwickelt, um die hohen Anforderungen an ein deterministisches und fehlerfreies Übertragen von Informationen erfüllen zu können.

Feldbusse verbinden verteilte Computersysteme aber auch den technischen Prozess mit einzelnen Steuerrechnern (dezentrale E/A).

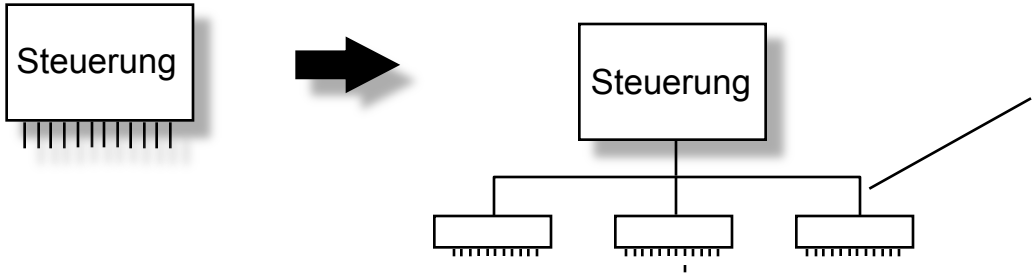
Die zentrale Automatisierungslösung

Die Steuerungsebene enthält den größten Teil der geforderten Intelligenz. Die Zahl der Schnittstellen zum Prozeß und die Aufgaben nehmen ständig zu. Folge: Die zentralen Automatisierungssysteme werden komplex, sind schwer zu beherrschen und sind unflexibel.



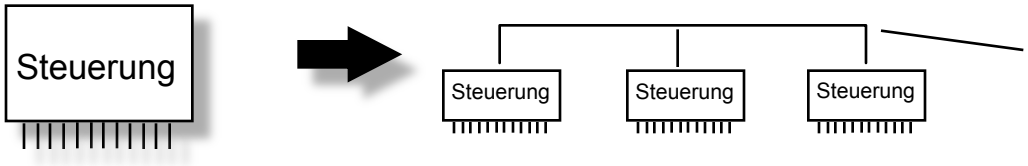
Dezentralisierung heißt

- **dezentrale E/A**
  Anschluß der Peripherie vor Ort (vielfältige Austauschbarkeit von Feldgeräten, Vorverarbeitung, Offenheit)



- Buslaufzeit: < 10 ms
- Datenmengen: gering
- Ziel: einfache Installation

- **dezentrale CPUs**
  Verteilung der Automatisierungsaufgaben (übersichtliche, schnelle Fehlersuche, leichte Wartbarkeit)



- Buslaufzeit: > 100 ms
- Datenmengen: groß
- Ziel: einfache Inbetriebnahme und Erweiterbarkeit

Dezentrale Strukturen - und ihre Vorteile

- geringerer Verkabelungsaufwand (Kostenreduktion)
- mehr Funktionalität durch intelligente Feldgeräte (z.B. Diagnose, digitale Signalverarbeitung, Parametrierbarkeit)
- kurze Wege bei kritischen, analogen Signalen (robuste Datenübertragung durch Digitalisierung und Codesicherung)
- bessere Übersicht (Fehlersuche und -behebung)
- modulare Struktur bringt Flexibilität (Erweiterung / Änderung)
- kürzere Reaktionszeiten (dezentrale Steuergeräte)
- mehr Informationsgehalt (höhere Genauigkeit)
- Offenheit durch standardisierte Übertragungsverfahren und Geräteschnittstellen
- größere Teilnehmerzahl möglich

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

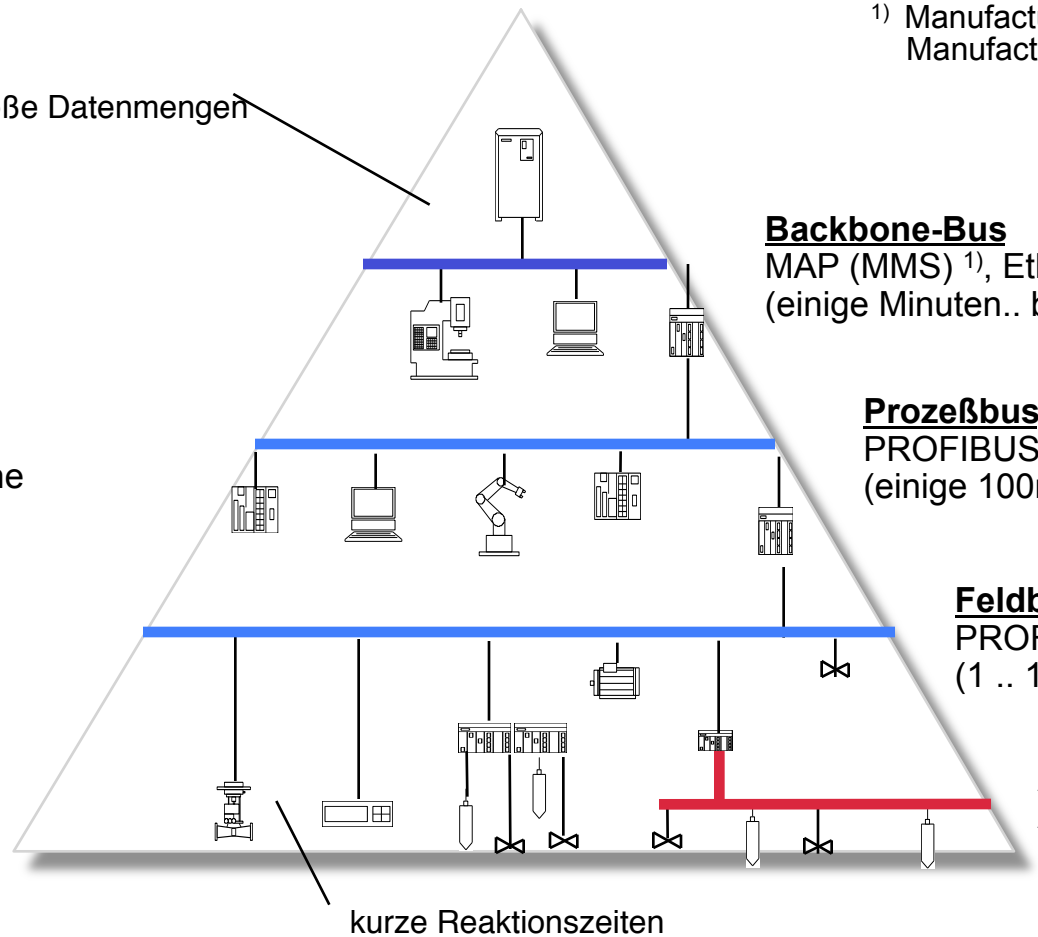
Ausprägungen von Bussystemen in der Automatisierungstechnik

- **Feldbus** und **Aktor/Sensor-Bus** für kleine Datenraten mit kurzen Reaktionszeiten (teilweise in Echtzeit), zur Erfüllung einer einzelnen Automatisierungsaufgabe.
- **Prozeßbus (LAN)** zum Datenaustausch mit mehreren Feldbussen (Gesamtanlage)
- **Backbone-Bus (WAN)** verteilte Datenkommunikation mit großer Komplexität und großen Entfernungen (z.B. Intra- bzw. Internet).

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Einordnung eines Feldbusses in die Automatisierungshierarchie

[1] Manufacturing Automation Protocol
Manufacturing Message Specification



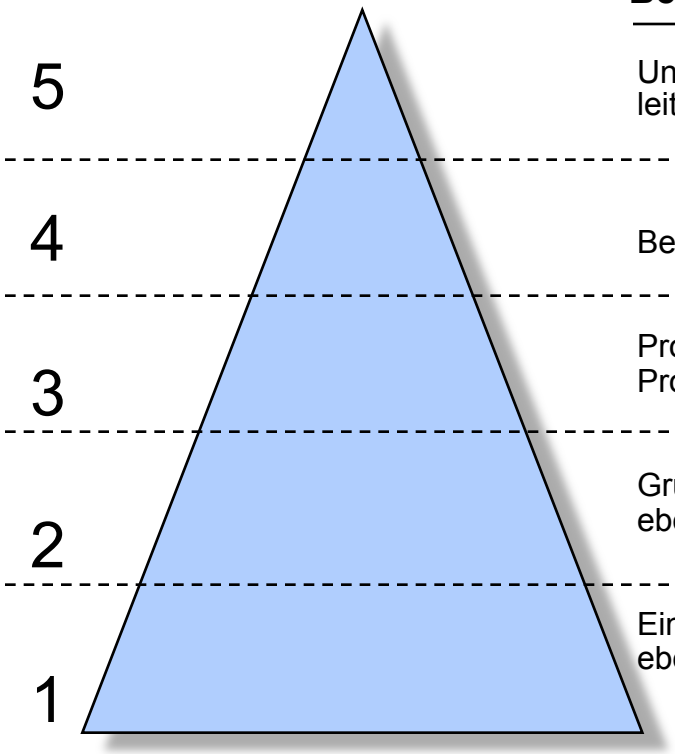
Backbone-Bus
MAP (MMS) [1], Ethernet
(einige Minuten.. bis Stunden .. Tage)

Prozeßbus
PROFIBUS-TF
(einige 100ms.. einige sec)

Feldbus
PROFIBUS-FMS, Interbus
(1 .. 10 ms)

Aktor/Sensor-Bus
ASI, Interbus, CAN,
PROFIBUS-DP
(einige 100 µs .. einige ms)

Leitebenen



	Bez. d. Hierarchie	Typ. Gerätebez.	Benutzer
5	Unternehmens-leitebene	Planungrechner	Unternehmensleitung
4	Betriebsleitebene	Leitrechner	Betriebsleiter, Manager
3	Prozeßleit- bzw. Prozeßrechnerebene		Operator, Schichtleiter
2	Gruppenleit-ebene	Zell Controller SPS, PC	Produktionspersonal
1	Einzelleit-ebene	Aktuator/Sensor	Wartungspers. Inbetrieb-setzer

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

7. Echtzeitkommunikation

7.1 Steuerungsnetzwerke

7.2 Netzwerkgrundlagen

7.3 Feldbusse

7.4 Interbus nach DIN 19258

7.5 CAN – Controller Area Network

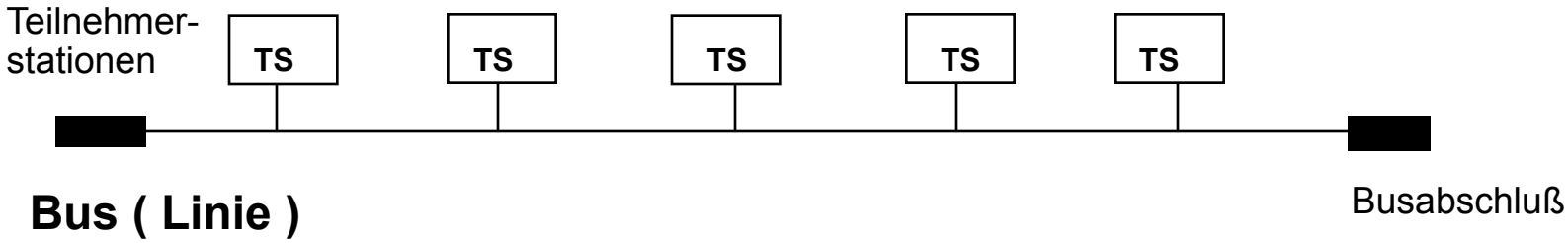
7.6 Profibus – Process Field Bus nach DIN 19245

7.7 RTnet – Real-Time Ethernet

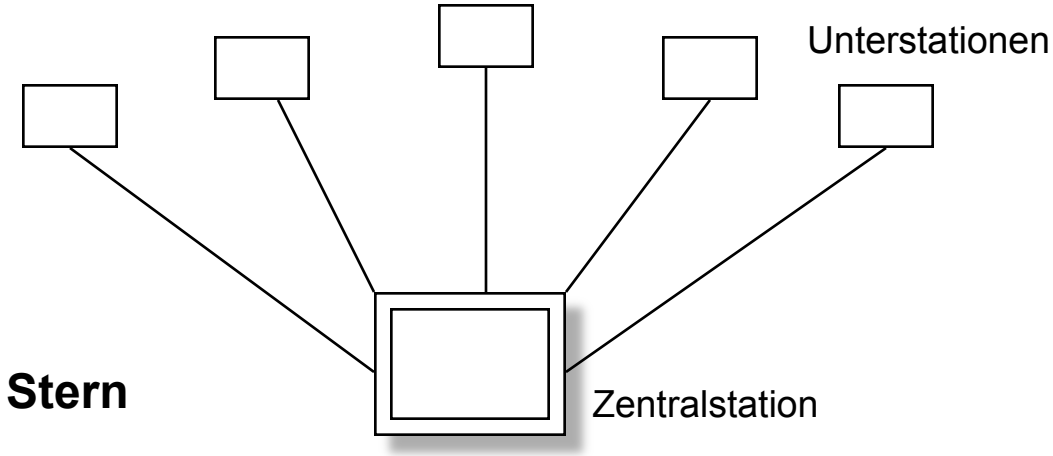
7.8 RACK – Robotics Application Construction Kit

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Netztopologien

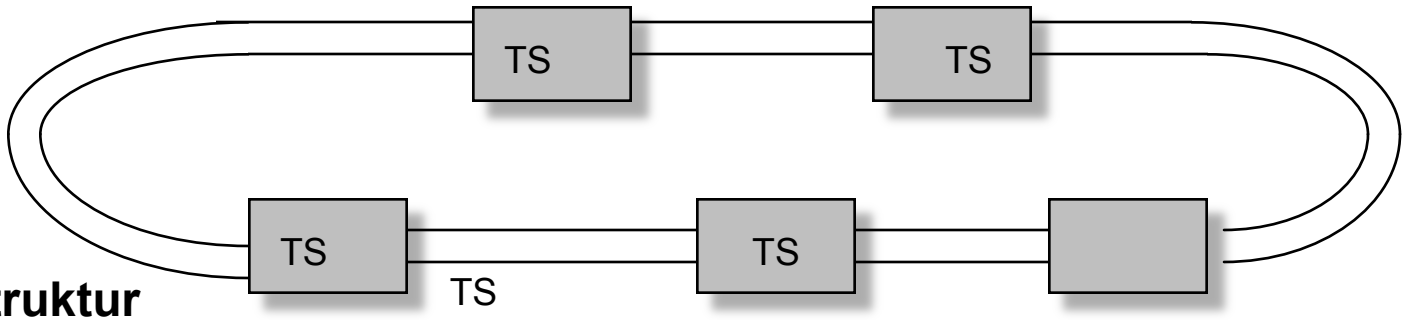


Bus (Linie)

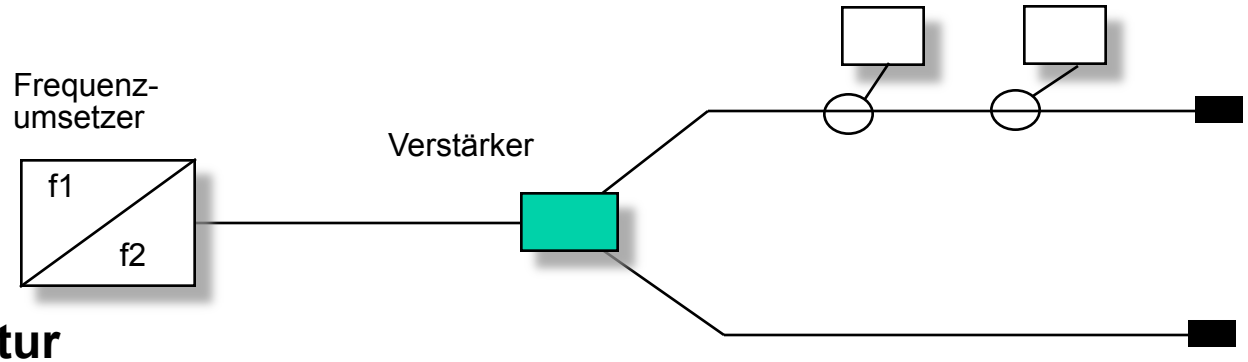


Stern

Netztopologien



Ringstruktur



Baumstruktur

Zugriffsberechtigung / Zugriffsverfahren / Übertragungstechnik

- **Zugriffsberechtigung**
  - stochastisch
  - prioritätsgesteuert
  - Master / Slave
  - Multimaster mit Token Passing
- **Zugriffsverfahren**
  - CSMA / CD (Carrier Sense Multiple Access / Collision Detect)
  - CSMA / CA (- “ - / Collision Avoidance)
  - Summenrahmenprotokoll (Schieberegister)
- **Übertragungstechnik**
  - RS 485 Technik nach EIA (Electronic Industries Association)
  - Lichtwellenleiter

Zugriffsberechtigung / Zugriffsverfahren / Übertragungstechnik

- **Schieberegister mit Summenrahmenprotokoll**: Der Busmaster sendet in jedem Buszyklus die Ausgangsdaten an alle Slaveteilnehmer wie durch ein Schieberegister (Interbus) und bekommt als Antwort die Eingangsdaten der Slaveteilnehmer zurück. Die Buszykluszeit ist berechenbar und wird durch azyklische Parametertelegramme, sowie im Fehlerfall verlängert. Der Datenumfang ist bei diesem Verfahren gering, jedoch bei hoher Protokolleffizienz.
- **Prioritätsgesteuerter Datenverkehr**: Jeder Teilnehmer am Bus, der Daten zu senden hat, kann diese zum Zeitpunkt des Entstehens senden. (CAN-Bus). Die Zeit, wann welche Daten gesendet werden, ist nicht vorhersagbar. Allerdings können Nachrichten mit hoher Priorität dann abgesetzt werden, wenn sie entstehen bzw. vorliegen (z.B. Zündzeitpunkt im Benzinmotor).
- **Master/Slave**: Die Initiative des Datenverkehrs geht vom Master aus. Ein Slaveteilnehmer antwortet nur auf Anfrage. Die Buszykluszeit eines solchen Systems ist berechenbar (deterministisches Verhalten). Nur im Störungsfall wird der Buszyklus für die Übertragung von Diagnosetelegrammen verlängert.
- **Multimaster mit Token Passing**:
  Sind mehrere Master am Bus beteiligt, teilen sie sich das Zugriffsrecht zum Bus durch Austausch eines Token zu. (PROFIBUS, Aktor-Sensor-Interface). Wer im Besitz des Tokens ist, hat alleiniges Buszugriffsrecht und darf innerhalb einer spezifizierten Zeit - oder bis er fertig ist - den Datenaustausch durchführen.

Mögliche Verbindungsformen

- **Parallele Verbindung** (Centronics, ISA-/PCI-Bus, Multibus...)
  - Kurze Datenstrecken im Bereich einige cm bis einige Meter
  - Extrem kurze Übertragungszeiten
  - Daten-, Adress- und Steuerbusse in Zentraleinheiten von Rechnern

- **Serielle Verbindung** (RS 232, RS 485, Ethernet ...)
  - Datenstrecken einige Meter bis mehrere Kilometer
  - schnelle Reaktionszeiten
  - Rechnernetze und Feldbusse

Zwei symmetrische Zweidrahtleitungen für Duplexbetrieb mit Differenzspannung -10V bis +10V

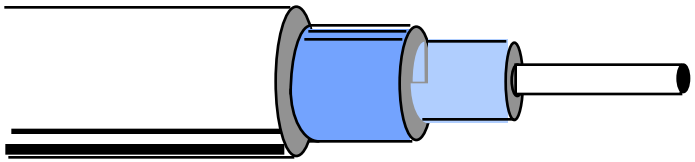
Übertragungsmedien für serielle Busse

- serielle Standardschnittstelle (asymmetrisch)
  (50 m bei max. 19,2 kbps, 2 Stationen)

V.24

- Geschirmte paarig verdrillte Zwei- oder Vierdrahtleitung (symmetrisch)
  (400 m bei 500 kbps, 32 Stationen)

RS 485



- Koaxial- bzw. Triaxialkabel
  (2,5 km bei 10 Mbit/sec, 1000 Stationen)

Koax

- Lichtwellenleiter
  (100 km bei größer 100 Mbit/sec, 2 Stationen)

LWL

7. Echtzeitkommunikation

7.1 Steuerungsnetzwerke

7.2 Netzwerkgrundlagen

7.3 Feldbusse

7.4 Interbus nach DIN 19258

7.5 CAN – Controller Area Network

7.6 Profibus – Process Field Bus nach DIN 19245

7.7 RTnet – Real-Time Ethernet

7.8 RACK – Robotics Application Construction Kit

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Anforderungen an ein Feldbussystem

- deterministisches Zeitverhalten (Echtzeit)
- zuverlässige und montagefreundliche Busanschlüsse
- bidirektionaler Informationsfluss
- An- bzw. Abkoppeln von Geräten ohne Auswirkung
- Ausfall von Geräten ohne Funktionsbeeinträchtigung für den restlichen Bus, bzw. Möglichkeit zur Fehlerlokalisierung
- erdfrei und galvanisch getrennt
- Zündschutzart „Eigensicher"
- Versorgungsspannung über Bus
- Offenheit

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Branchenspezifische Anforderungen an Feldbusse

Anforderungen	Anlagentechnik	Automobil	Fertigung	Gebäude	Verfahrenstechnik
Anlage	binär I/O´s	binär/analog I/O´s	SPS,NC,Roboter	binär/analog I/O´s	
Reaktionszeit	< 1ms	< 1ms	einige ms ... 1min	< 50ms	einige hundert ms
Meldungen		ereignisgest.	zykl. im ms Bereich		
Ausdehnung	< 1km	10..50m	50m	> 500m	einige m...2..3 km
Temperaturber. in Grad C	- 40...	- 30...120	normal	normal	normal
Schutzart	IP 54/65	IP65			IP65
Besonderheiten		klein, gesichert, fest, geringe Serienkosten	EMV-fest		Ex-Bereiche

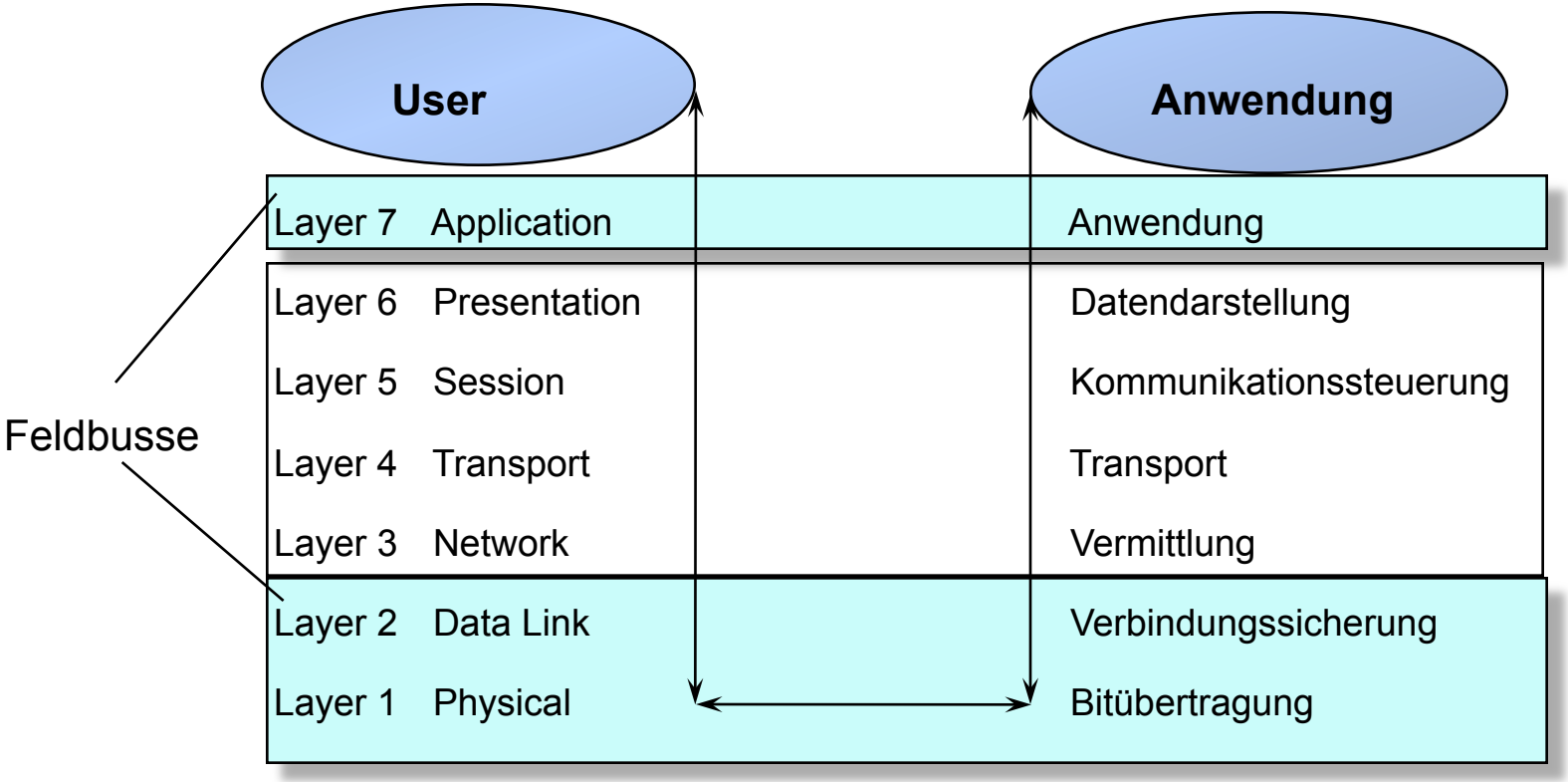
© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Feldbusse

• Interbus		DIN 19258
• CAN	Controller Area Network	ISO 11898
• Profibus	Process Field Bus	DIN 19245
• ASI	Aktuator/Sensor-Interface	IEC Entwurf
• EtherCAT	Echtzeit-Ethernet	IEC61151
• FlexRay	Fahrzeugkommunikation	
• Bitbus		IEEE 1118
• EIB	Europäischer Installationsbus	UTE 46 (Frankreich)
• LON	Local Operating Network	-
• P-Net		EN 50170
• FIP	Factory Information Protocol	EN 50170
• IEC 685, DIN Messbus, usw.		

ISO / OSI Modell

Open System Interconnection



7 Schichten, die nur Schnittstellen zu den jeweils benachbarten Schichten haben

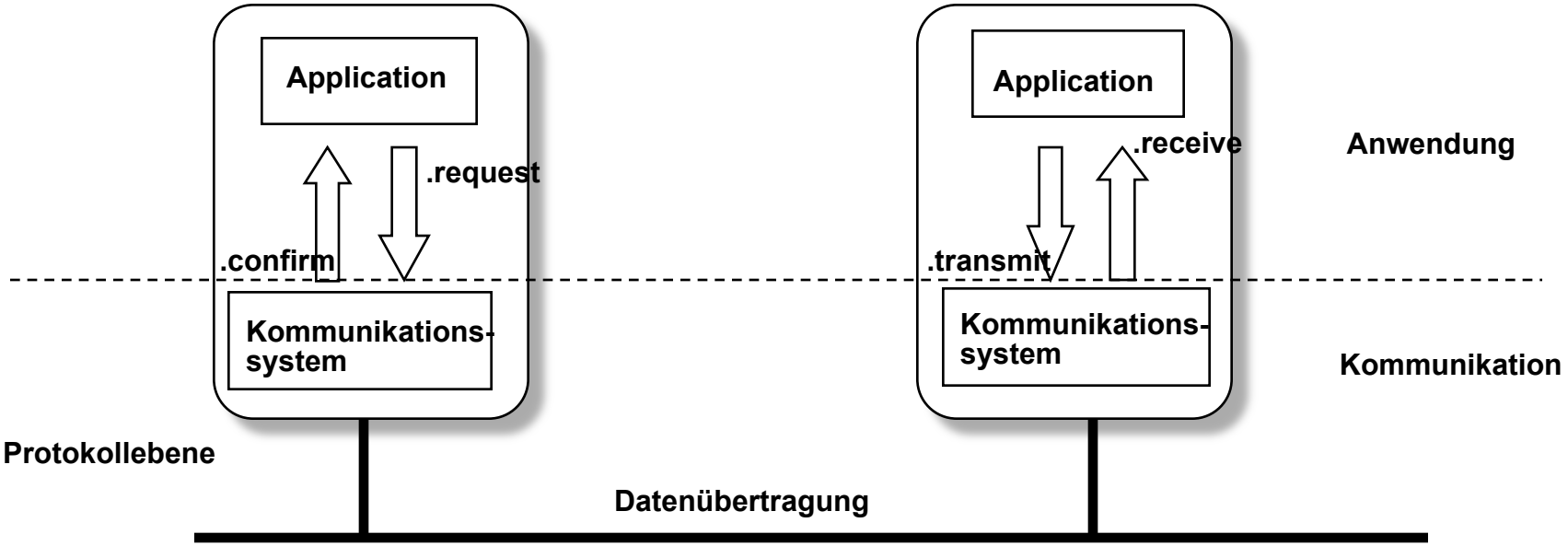
Feldbusse im ISO 7498 / OSI Modell

Feldbusse mit reduziertem Ausbau wegen Echtzeit.

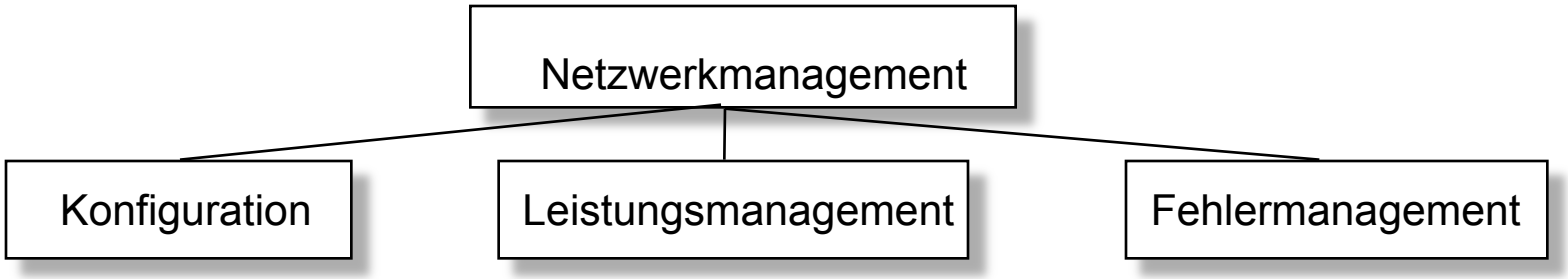
- L1: Physical Layer
  Art des Signals, Art des Kabels (z.B. twisted pair), Stecker mit Anschlußbelegung, Kodierung
- L2: Data Link Layer
  fehlerfreie Übertragung, Zugriffsmechanismen, Strategie bei Buskollisionen, Datenformat, Adressierung, Kontrollbits
- L7: Application Layer
  Kommunikationsfunktionen für das Automatisierungsprogramm und anwendungsspezifische Protokolle

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Ablauf einer Datenübertragung



Dienste eines Feldbussystems



- Das **Netzwerkmanagement** enthält alle Programmteile, die zum Projektieren und zur Inbetriebnahme eines Automatisierungssystems notwendig sind.
- Das **Leistungsmanagement** enthält Funktionen zum Optimieren einer Automatisierungsaufgabe hinsichtlich effizientem Datenaustausch.
- Das **Fehlermanagement** sorgt dafür, daß alle auftretenden Fehler an den Host schnellstmöglich gesendet werden. Es dient dem Erkennen und Melden von Fehlerzuständen innerhalb einer Automatisierungsaufgabe.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

7. Echtzeitkommunikation

7.1 Steuerungsnetzwerke

7.2 Netzwerkgrundlagen

7.3 Feldbusse

7.4 Interbus nach DIN 19258

7.5 CAN – Controller Area Network

7.6 Profibus – Process Field Bus nach DIN 19245

7.7 RTnet – Real-Time Ethernet

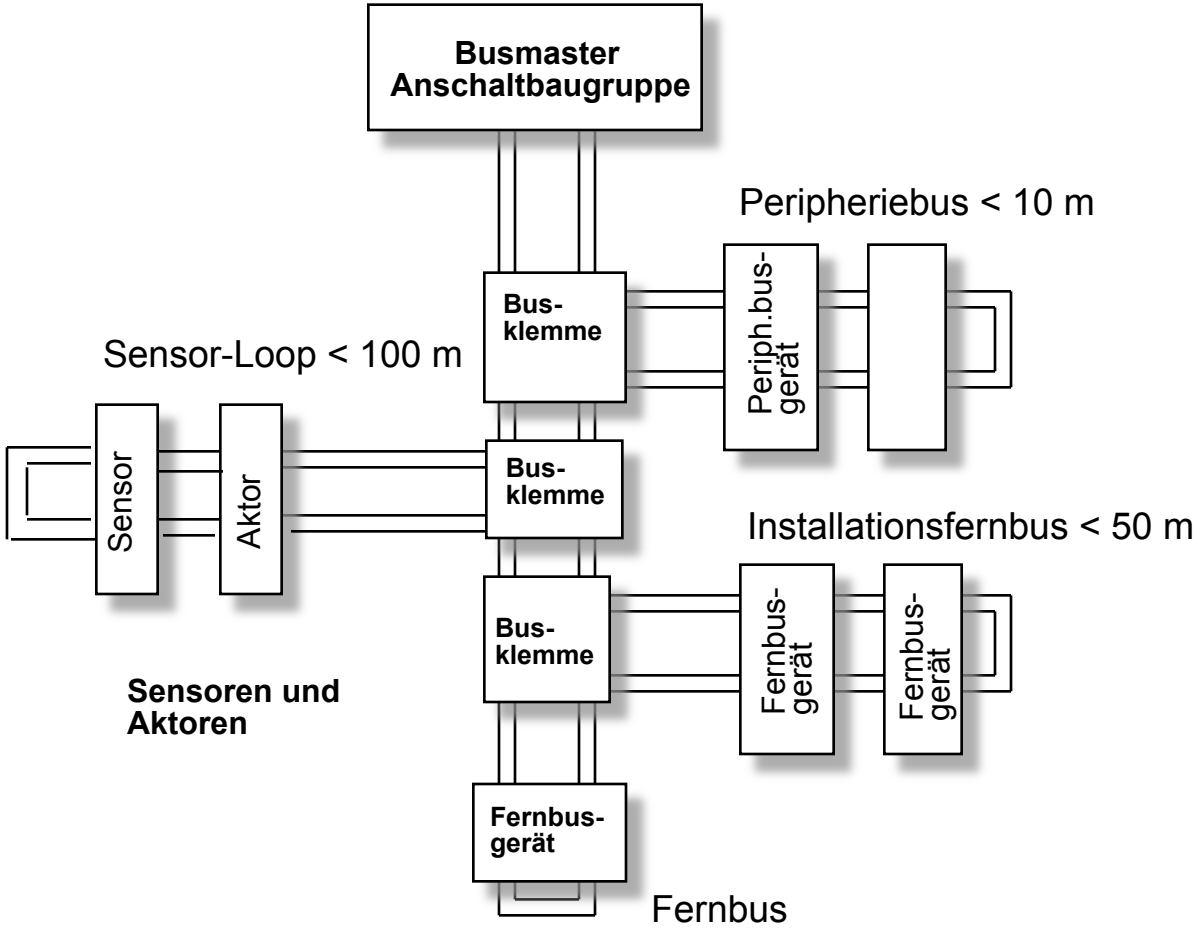
7.8 RACK – Robotics Application Construction Kit

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Interbus nach DIN 19258

- Für den Sensor-Aktuator-Bereich geeignet
- Die Zielsetzung ist, daß zyklisch anfallende Daten ohne großen Overhead übertragen werden (SPS).
- Der Interbus benutzt eine Ringtopologie mit Summenrahmentelegramm als Übertragungsformat.
  - Das Übertragen von Ein- und Ausgangsdaten erfolgt in einem Zyklus und damit in bekannter Zeitspanne (Echtzeit!).
  - Jeder Teilnehmer fügt seine Eingangsdaten hinzu, nachdem er zuvor seine empfangenen Daten entnommen hat.
- Von Fa. Phönix Contact entwickelt und seit 1990 offengelegt, seit 1994 genormt.
- Die Anwenderinteressen werden durch den Interbus Club unterstützt, der auch die Zertifizierungen von Geräten durchführt.

Topologie des Interbus



Interbus: Übersicht

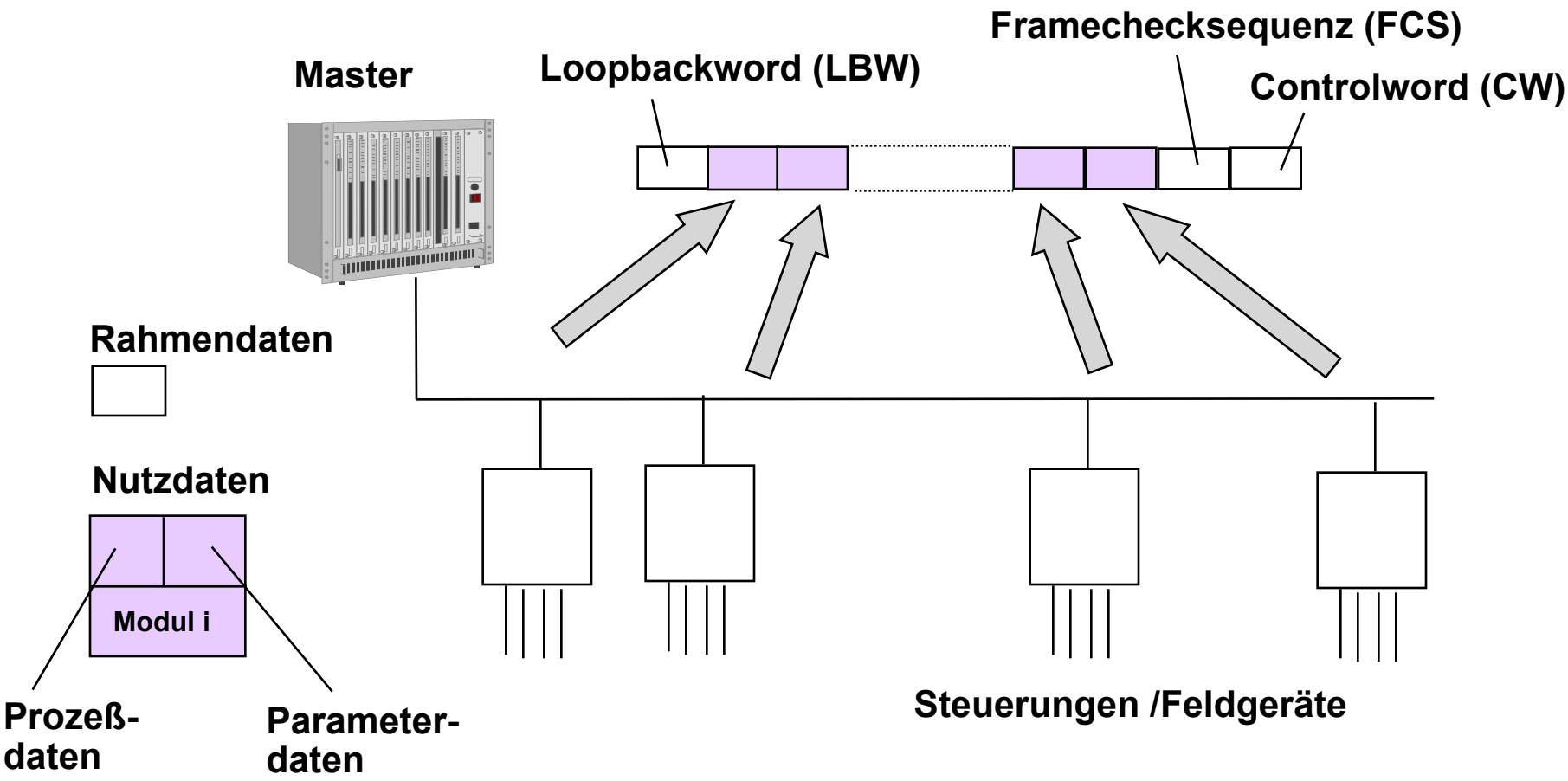
- Datenring mit zentralem Master-Slave-Zugriffsverfahren
- Zyklisches Polling aller Teilnehmer mit festem Zeitraster
- Deterministisches Verhalten (Echtzeitverhalten)
- Auslegung als Fernbus mit Installations-, Peripheriebus und Sensor-Loop (Baumstruktur)
- Ankoppeln von 256 Teilnehmer am Fernbus möglich
- Anzahl der Teilnehmer am Peripheriebus durch Spannungsabfall auf 8 Teilnehmer (Abstand max. 1,5 m) begrenzt.
- Beim Sensor-Loop können nur 64 Teilnehmer/ Segment angeschlossen sein.
- Gesamtausdehnung max. 12,8 km (Festlegung in der Norm wegen Test)
- Zeitgleiches Senden und Empfangen durch Ringstruktur
- Verwendung des Summenrahmenprotokolls (Schieberegister), d.h. das Telegramm beginnt und endet am Master.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Physikalische Eigenschaften

- Im **Fernbus** wird ein auf RS 485 basierendes Protokoll verwendet. Die Übertragungsrate beträgt 500 kbit/s. Die Fernbusleitung besteht aus 5 Adern, die paarweise verdrillt sind. Der Ring wird, um die Installation zu vereinfachen, mit einem einzigen Kabelstrang realisiert.
- Der **Peripheriebus** arbeitet mit CMOS-Pegeln und benötigt 15 Adernpaare. Die Busklemme versorgt die Elektronik im Peripheriebus (max. 300 kbit/s). Das entspricht beim Fernbus einer Baudrate von 500 kbit/s bei 2-Leiter-Technik.
- Die Interbus Spezifikation schreibt eine **galvanische Trennung** der Teilnehmer vor (außer Peripheriebus). Jeder Fernbusteilnehmer hat eine eigene Spannungsverstärkung und wirkt deshalb als **Repeater**.
- Der Abstand der Teilnehmer am Fernbus < 400 m, am Peripheriebus < 1,5 m.
- **LWL** als optionales Medium verfügbar.
- **Sensor-Loop:** Mit spezieller Busklemme erfolgt die Umsetzung zur gleichzeitigen Übertragung von Daten und Versorgungsspannung auf einer 2-Drahtleitung. Zwischen 2 Sensoren dürfen max. 10m liegen --> max. 100m Ausdehnung. Das Nutzsignal wird auf die Versorgungsspannung aufmoduliert. Stromaufnahme von 40 mA je Teilnehmer möglich.

E/A orientiertes Übertragungsverfahren (Interbus)

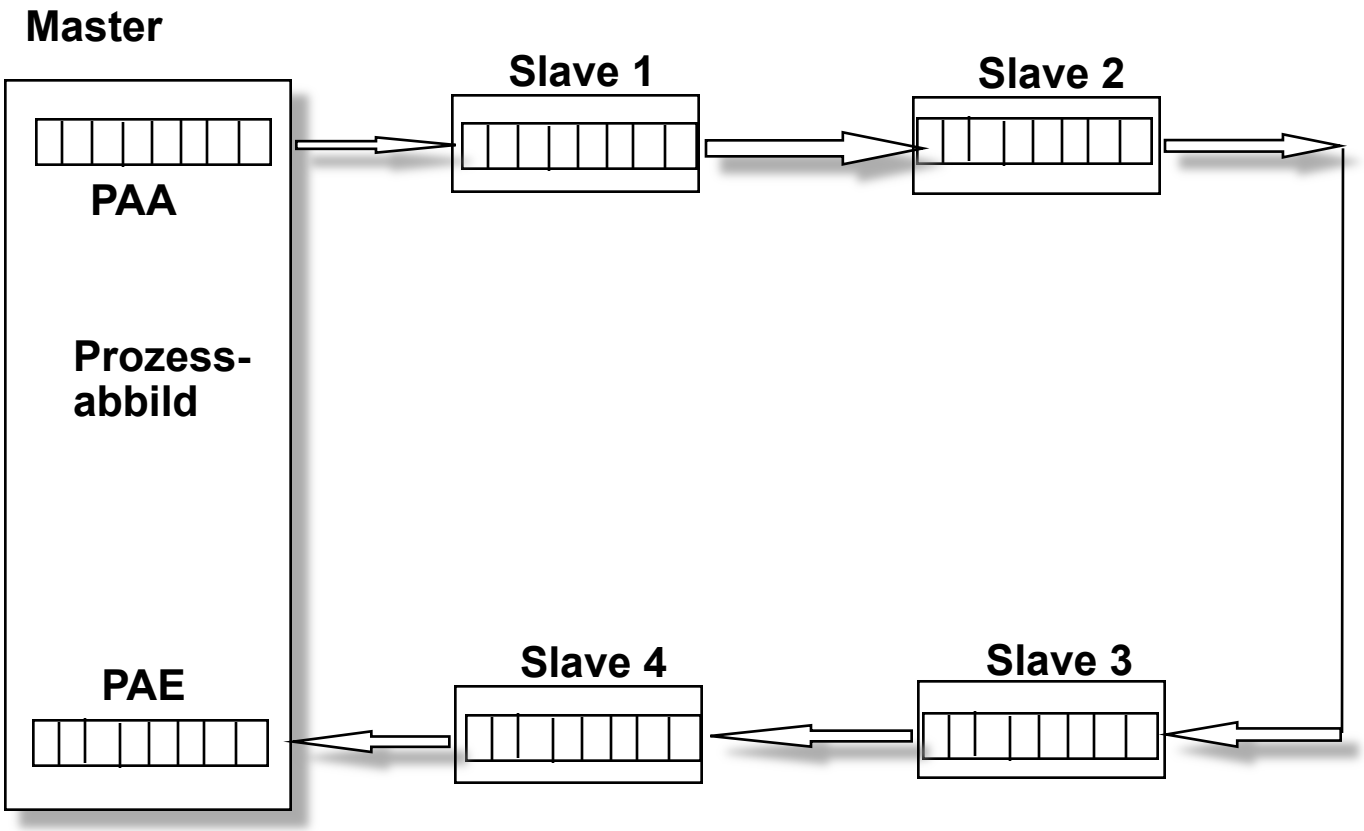


Ein-/Ausgangsorientiertes Datenübertragungsverfahren

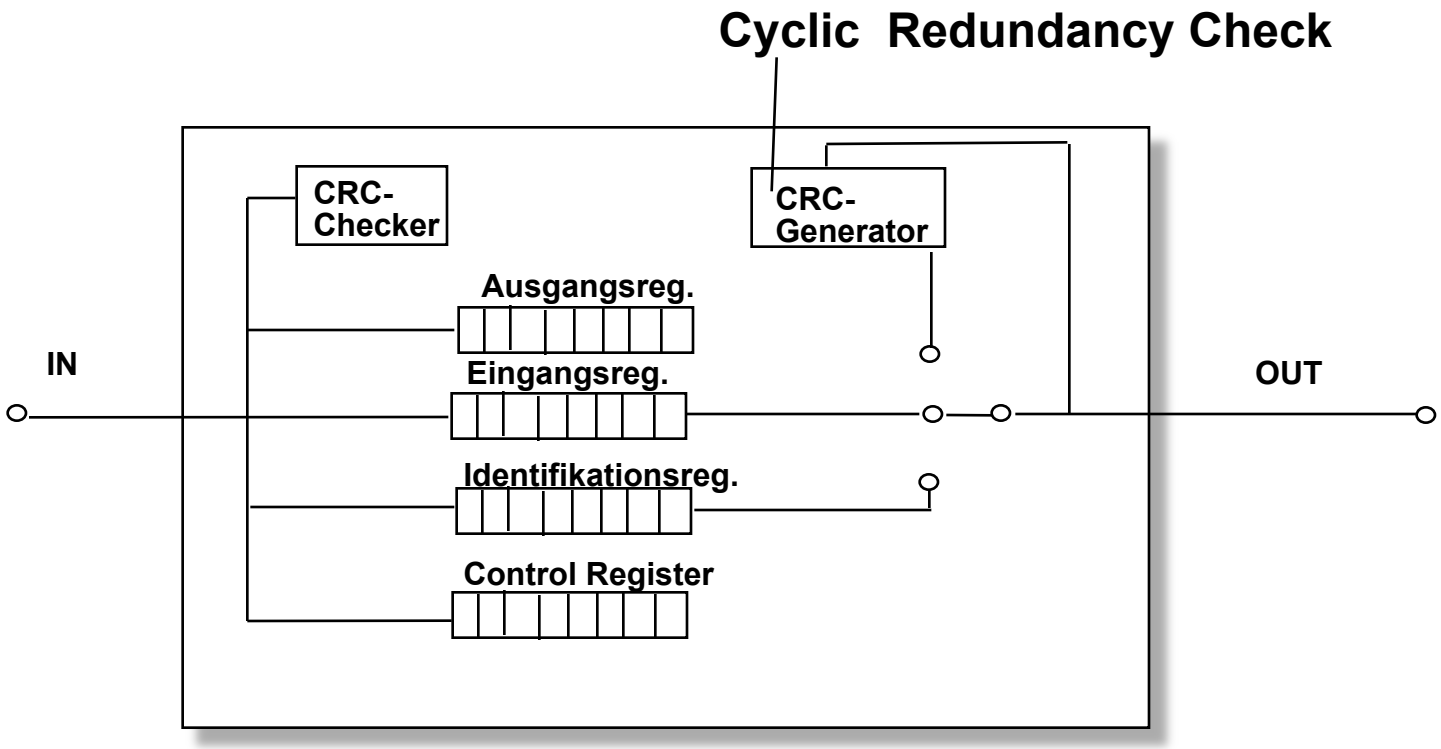
- Geeignet für mittlere Geschwindigkeiten mit **festem Zeitraster**.
- Alle E/A´s werden in **einem Zyklus** (Summenrahmenprotokoll) angesprochen. Die Rahmendaten werden nur einmal benötigt.
- Die Daten sind durch die **physikalische Lage** eindeutig einem Teilnehmer zugeordnet. Es gibt keine Busadressen. Folglich ist keine Adresseinstellungen am Feldgerät notwendig. Der Austausch eines defekten Feldgerätes wird einfach (Wiederholung der Identifikationszyklen).
- Jedes Gerät hat eine **feste Datenlänge** für Prozeßdaten (E/A-Signalwerte) und Parameterdaten (Nachrichten), die sich während des Betriebs nicht ändert (z.B. Ventile, Encoder). Die maximale Länge des Summenrahmentelegramms beträgt 512 Byte (4096 Bit).
- Die E/A-Daten werden im Prozeßabbild (PAE/PAA) des Masters gespeichert.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Struktur eines Interbus Systems



Aufbau eines Interbus Teilnehmers



Der Datenstrom zwischen zwei Teilnehmern wird vom zugehörigen CRC-Generator und -Checker durch das gleiche Polynom geteilt. Zu Beginn der Übernahmesequenz eines Datenzyklus werden die vom CRC-Generator ermittelten Polynomreste an den nachfolgenden CRC-Checker gesendet. Sind die Werte ungleich, liegt ein Übertragungsfehler vor.

Aufbau eines Interbus Teilnehmers

- Die E/A Daten werden durch die Schieberegister transportiert.
- Eingangsregister dient zur Speicherung der Eingangsdaten bei Datenübernahme.
- Parallel zum Eingangsregister sind die Ausgangsregister und die CRC-Register angeordnet, durch die ebenfalls die Daten geschoben werden.
- Die Identifikationsregister beinhalten Informationen zum Unterscheiden der Teilnehmer. Aus den Identifikationsregistern holt sich der Master im Identifikationszyklus die Teilnehmerinformationen.
- Die einzelnen Register werden über den Selektor in den verschiedenen Phasen an den Bus geschaltet
- Die Anzahl der Schieberegister bestimmt die Datenbreite
- Für die Busschnittstelle ist kein Mikroprozessor notwendig.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Der Identifikationscode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Gerätegruppe

Datenbreite

Management Bits

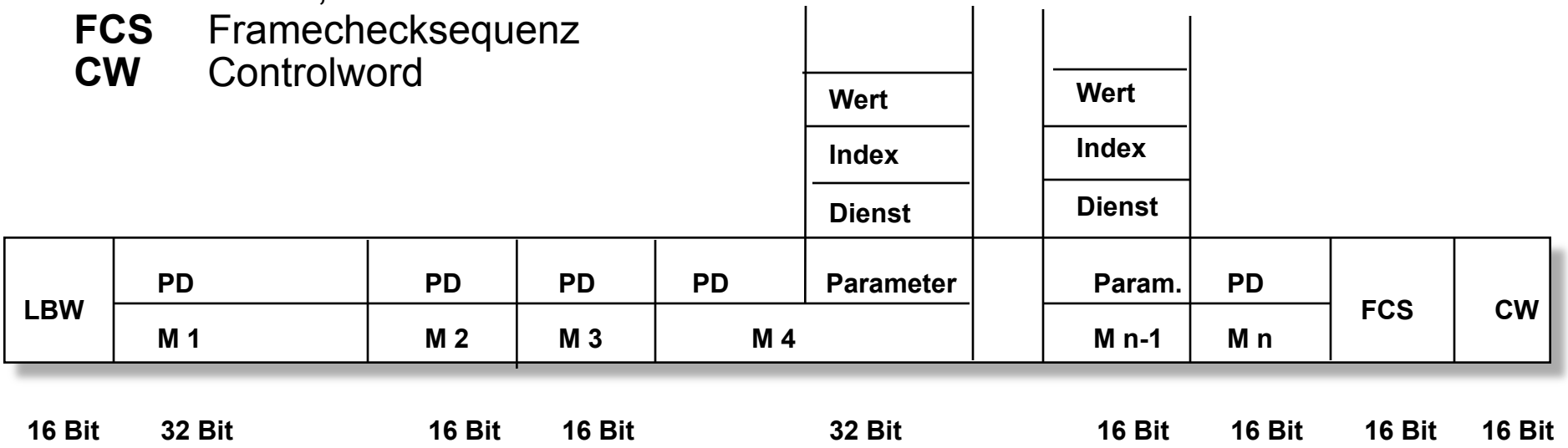
ID 15	ID 14	ID 13	Bedeutung
x	x	1	Rekonfigurations anforderung
x	1	x	CRC-Fehler
1	x	x	Modulfehler

Je Teilnehmer existiert ein ID-Code.

Beim Start der Anlage werden vom Master Identifikationszyklen durchgeführt, um den Busaufbau zu ermitteln. Im Fehlerfall wird durch erneute Identifikationszyklen die Fehlerstelle lokalisiert.

Summenrahmentelegramm

LBW Loopbackword
PD Prozessdaten
M Modul, bzw. Teilnehmer
FCS Framechecksequenz
CW Controlword

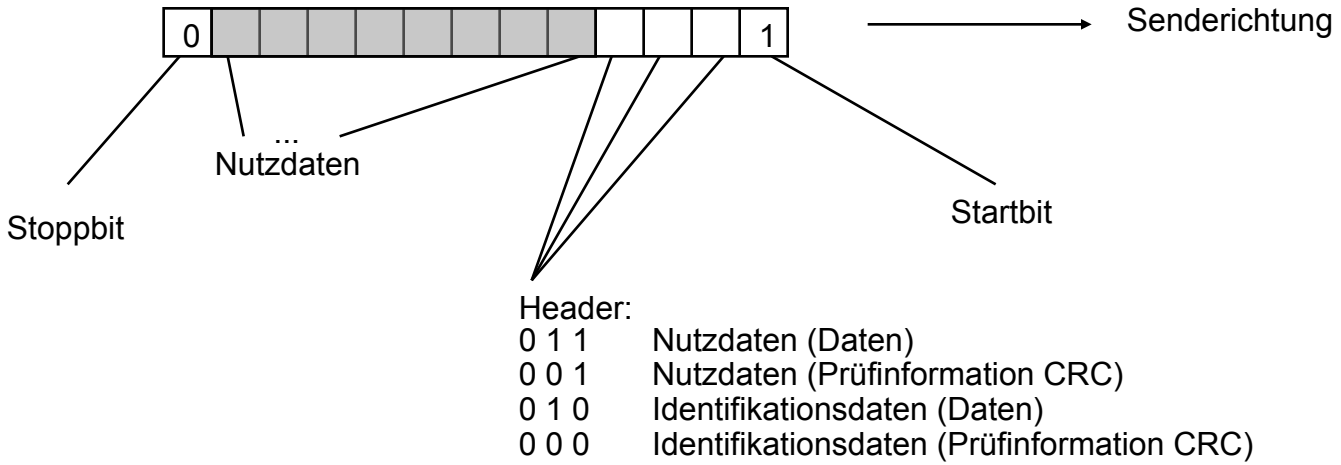


Interbus Telegramm

- Der Master sendet ein komplettes Übertragungstelegramm an die sequentiell im Sinne eines **verteilten Schieberegisters** verbundenen Teilnehmer. Zusammen bilden sie aus der Sicht des Masters einen Einzelteilnehmer.
- Ein Teilnehmer entnimmt die ihm zugeordneten Daten und antwortet mit einem Einzeltelegramm, dass über das Schieberegister zum Master gesendet wird.
- **Parameterdaten** werden in Datenblöcke zerlegt und sequentiell in das Protokoll eingefügt. Zuständig ist das **Peripheral Communication Protocol (PCP)**.
- Prozeßdaten und Parameter können gleichzeitig übertragen werden.
- Im Telegrammformat wird zwischen **Identifikations- und Datenzyklus** unterschieden. Jeder Zyklus besteht aus einem Kontrollwort (LBW), Daten, CRC-Wort (FCS), und Endesequenz (CW).
- Jeweils acht Nutzbit werden zusammen mit 5 Bit Statusinformation asynchron übertragen.
- In Datenübertragungspausen werden Statustelegramme übertragen (5 Bit). Sie verhindern einen automatischen Verbindungsabbau nach 20 ms.
- Daten und Status werden durch das Markerbit unterschieden.

Asynchrone Datenpakete, Statusbits

Die Übertragung des Summenrahmens erfolgt mit einem asynchronen Start-Stop-Verfahren in Datentelegrammen:
(**8 Bit Daten + 5 Bit Statusinformation**)



Der Busmaster teilt den Teilnehmern über die Headerbits mit, ob Daten im Schieberegister gesendet werden, Prüfinformation ausgetauscht wird und wann sie die Ausgabedaten übernehmen bzw. die Eingabedaten senden sollen.

Übertragungszeit

$t_ü = t_{Datenübertragungsphase} + t_{Datensicherungssphase} + 2*t_{Busverzögerungszeit}$

mit den entsprechenden Werten ergibt sich:

$t_ü = [13 * (6 + n) + 4*m] * t_{Bit} + t_{SW} + 2 * t_{Phys}$

mittlerer Wert, abh. von Fernbusgeräten

mit:

$t_ü$	Übertragungzeit
n	Anzahl Nutzdatenbytes
m	Anzahl Fernbusteilnehmer
t_{Bit}	Bitdauer (2 µs bei 500 kBit/s)
t_{SW}	Softwarelaufzeit (typischer Wert: 200 µs)
t_{Phys}	Laufzeit auf dem Übertragungsmedium (ca. 2,5 µs für 400m Kupferdraht)

Interbus Sicherungsmechanismen

- Bei 2-Leitertechnik wird nur der Datenstrom überwacht. Bei der historischen 8-Leitertechnik werden die Leiter auf Pegelgleichheit überprüft.
- Im LBW schickt der Master ein Kontrollwort durch sämtliche Teilnehmer. Ergibt der abschließende Vergleich einen Unterschied, z.B. weil ein Teilnehmer nachträglich eingefügt wurde, werden die Daten der Slaves nicht übernommen
- Alle Übertragungsstrecken werden durch die jeweils angeschlossenen Teilnehmer mit Hilfe eines 16 Bit CRC-Polynoms überwacht. Ein erkannter Fehler kann durch Veränderung eines Datenbits im Controlword (CW) bis zum Master weitergegeben werden.
- Durch 16-Bit CRC erreicht man eine HD=4 (Hamming Distanz) [1)]
- Bei Fehlern bestimmt der Master mit Hilfe von ID-Zyklen den Fehlerort.
- Fehlerhafte Segmente können automatisch vom Ring entfernt werden. Bei Ausfall eines Peripheriebusteilnehmers, kann der Master das Segment abkoppeln.

[1)] Es werden bis zu 3 fehlerhafte Bits erkannt.

7. Echtzeitkommunikation

7.1 Steuerungsnetzwerke

7.2 Netzwerkgrundlagen

7.3 Feldbusse

7.4 Interbus nach DIN 19258

7.5 CAN – Controller Area Network

7.6 Profibus – Process Field Bus nach DIN 19245

7.7 RTnet – Real-Time Ethernet

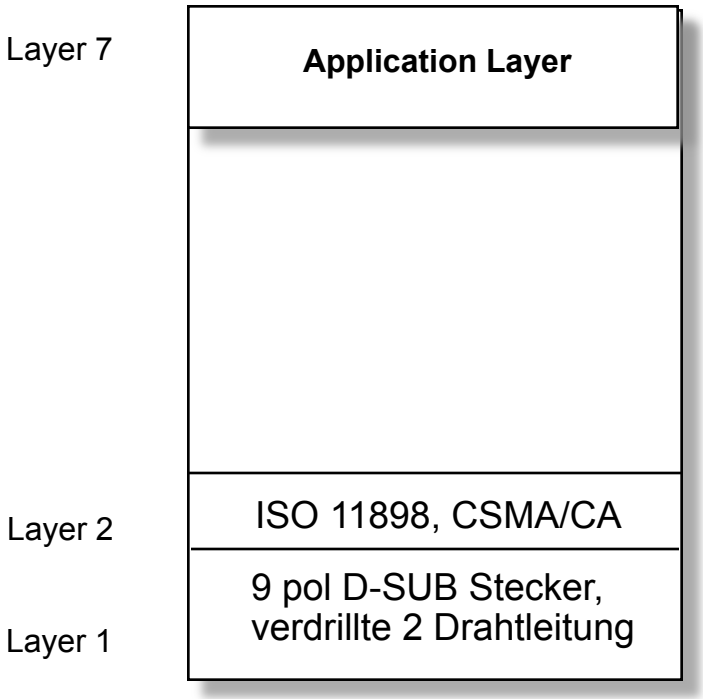
7.8 RACK – Robotics Application Construction Kit

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

CAN - Controller Area Network

- CAN wurde von BOSCH und Intel entwickelt zur Vernetzung von Sensoren, Aktuatoren und Steuergeräten im Automobil.
- Die europäisch orientierte Nutzerorganisation ist `CAN in Automation` (CiA).
- CiA hat die Anwenderschicht CAL = CAN Application Layer definiert ==> offener Industriestandard.
- Die Einsatzfelder liegen derzeit vorwiegend in der Automobilindustrie mit den Anforderungen: Störfestigkeit, kurze Verzögerungszeiten und prioritätsgesteuerter Datenverkehr.
- Verfügbarkeit von preiswerten Chips in hoher Stückzahl ist garantiert. Dies macht den CAN-Bus zunehmend auch für andere Anwendungsbereiche interessant. Es gibt Mikrocontroller mit integrierter CAN-Schnittstelle.
- Es sind mehrere Übertragungsprotokolle verfügbar.

Einordnung von CAN im ISO / OSI Modell



Bitrate	Buslänge
1MBit	< 40 m
500 kBit	< 100 m
100 kBit	< 500 m
50 kBit	< 1000 m

Topologie von CAN

CAN

- Zwei Versionen sind erhältlich:

Version	2.0A	2.0B
Identifierlänge	11Bit	29 Bit

  ==> unterschiedlicher Adressraum und Priorisierungsstufen
- Multimaster CSMA/CA Buszugriffstechnik, d.h. im Konfliktfall, wenn mehrere Teilnehmer gleichzeitig zugreifen, hat die höchstpriore Nachricht Vorrang (bitweise Buszugriffsarbitrierung im Identifierfeld). Jeder Master kann zu jedem Zeitpunkt auf den Bus zugreifen.
- Übertragungsgeschwindigkeiten: 10 kbit/s ... 1 MBit/s (40 m). Ausdehnung ist abhängig von Single-/Multi-Masterbetrieb und wird begrenzt aufgrund der bitweisen Arbitrierung. Im Single-Masterbetrieb ist die Ausdehnung nur von den physikalischen Bustreibern abhängig.
- Hohe Datensicherheit durch Hamming Distanz = 6.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Topologie von CAN

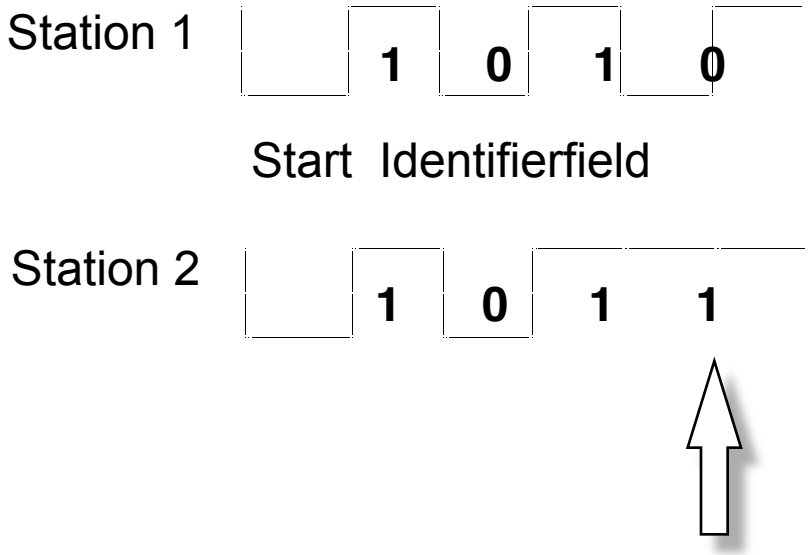
- Das Bussystem ist als Topologie ausgeführt mit Master und Slavebeteiligung. Verdrillte 2-Drahtleitung mit Abschlußwiderstand an den Segmentenden gemäß **modifizierter RS 485** Physik nach ISO DIS 11898. Das Netz kann mittels Repeater verlängert werden.
- **Maximal 64 Knoten** sind pro Segment möglich.
- Der Datenverkehr zwischen Master und Slave erfolgt mittels Polling oder ereignisgesteuert.
- Die Nachrichten werden mit unterschiedlicher Priorität als **Broadcasting** übertragen.
- **Bitstuffing** für genügend Pegelwechsel zur Resynchronisierung. Nach 5 gleichen Bits wird automatisch ein Verletzungsbit eingefügt. Dies wird von allen Teilnehmern überwacht.
- **Latenzzeit** kann nur für die höchstpriore Nachricht garantiert werden. Höchstpriore Nachricht wird nur solange verzögert, bis momentan gesendete Nachricht beendet ist. (bei 1 MBit/s = 134 µs)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Buszugriff CSMA / CA

Non Return to Zero

NRZ Signale



1 = Rezessiv (nachgebend)

0 = Dominant (überstimmend)

Station 2 stoppt den Sendevorgang (0 ist dominant)

CAN

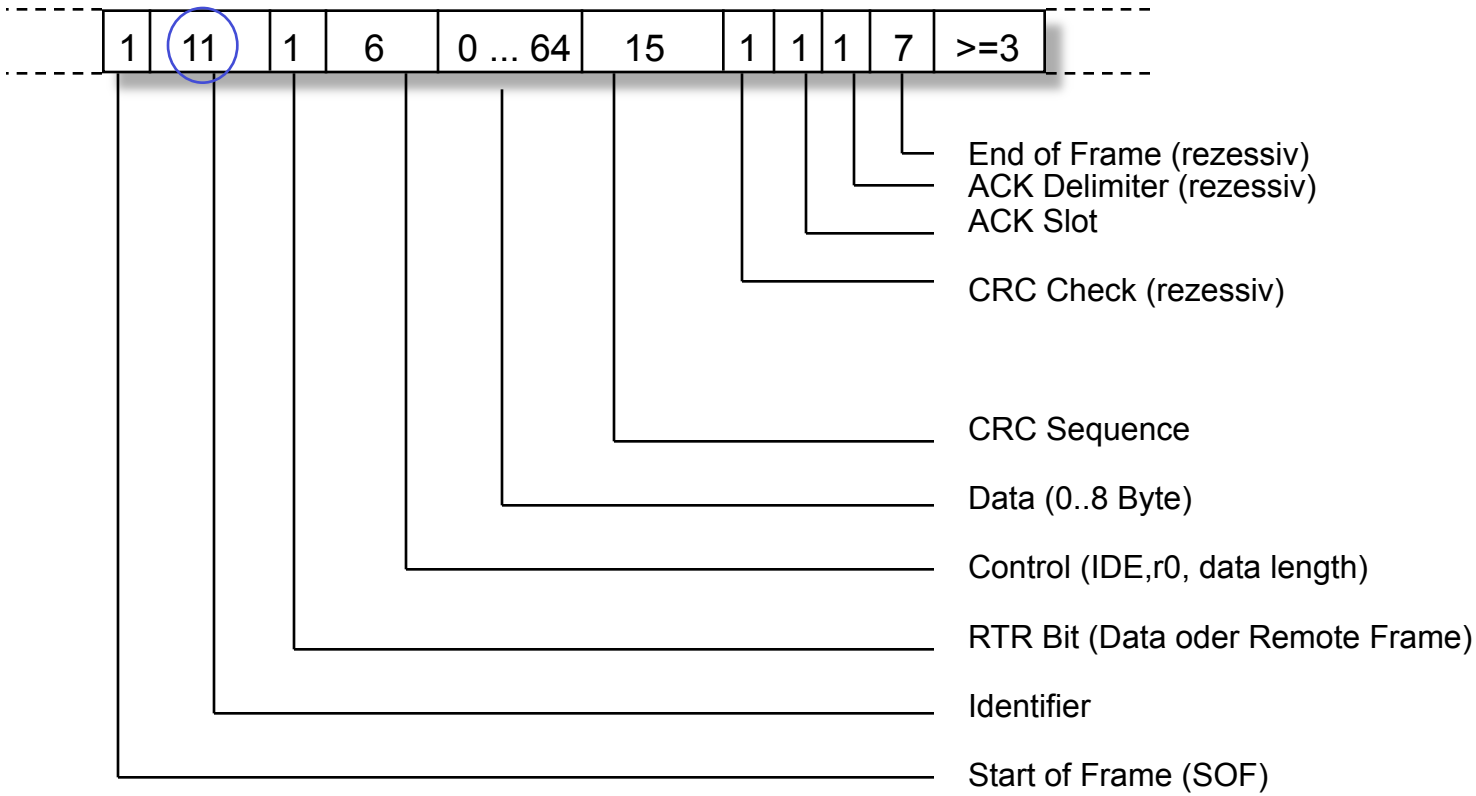
Man unterscheidet:

- "Full CAN Version" mit internem Auftragsspeicher
- "Basic CAN" mit registerbasierter Nachrichtenfilterung.
  Einstellbare Gruppenzugehörigkeit und Filtermaske
  ACR ... acceptance code register
  AMR ... acceptance mask register
- Bei "SLIO =Serial link I/O" sind Empfangs-und Sendemailboxes direkt mit I/O´s verbunden.

  CAN-Controller-Chips:
  Intel 82526, Philips 82C200 u.a.
  Philips 87C592 (8051-kompatibler Mikrocontroller mit integriertem 82C200)

Telegrammformat

Es gibt auch ein „Extended Format“ mit 29 Bit.
Die Struktur ist geringfügig anders.



CAN - Telegrammformat

- **Startbit** kennzeichnet den Telegrammanfang und dient der Phasensynchronisation.
- **Identifier** enthält Name, Priorität und die Botschaft der Objekte
- Das **RTR R**emote **T**ransmission **R**equest Bit kennzeichnet das Telegramm als Datentelegramm (low) oder als Anforderungstelegramm (high).
- Im **Controlfeld** dienen die 4 niederwertigsten Bits der Längenangabe.
- **Data** besteht aus 0 ... 8 Bytes
- **CRC** Code + CRC Delimiter zeigt die Richtigkeit oder den Fehlercode der vorangegangenen Bits an.
- Im **ACK Slot und ACK** Delimiter wird die Quittung für eine korrekt empfangene Sendung mitgeteilt.
- **End of Frame** zeigt das Ende des Datentelegramms an.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

CAN Data Link Layer

- **Buszugriff bei Multi-Master-Betrieb:**
  - Datenkollision muß vermieden oder gelöst werden
  - Problemlösung: Kollisionsvermeidung durch CSMA/CA
  - Nachricht mit kleinstem Identifier erhält die Sendeberechtigung
  - Dominant ist log=0, rezessiv ist log=1
- **4 Nachrichtentypen verfügbar:**
  - Data Frame für Datenübertragung von 0 .. 8 Bytes
  - Remote Frame zur Sendeaufforderung eines anderen Busteilnehmers
  - Error Frame zum Melden von Übertragungsfehler (6 Bits mit <u>gleichem</u> Wert)
  - Overload Frame zum Anzeigen eines Teilnehmers, dass er keine Daten aufnehmen kann. Eventuell selbstständiges Anhalten eines Teilnehmers.

CAN Data Link Layer

- **Mechanismen zur Fehlererkennung**
  - Vergleich der gesendeten Bitpegel mit dem Buspegel
  - CRC Prüfsumme über das Telegramm
  - Acknowledgeprüfung (mindestens 1 Knoten muß antworten)
  - Bit Stuffing wird von jedem Teilnehmer überprüft
  - Fehlerzähler zur Erkennung von dauernden und sporadischen Fehlern
  - Formatfehler: dominantes Bit in Delimiter oder End-of-frame-Feld
- **Übertragungssicherung**
  - Typische Restfehlerwahrscheinlichkeit $4{,}7 * 10^{-11}$
  - HD=6
  - Netzwerkweite Datenkonsistenz
  - Unterscheidung hinsichtlich Störung und Ausfall eines Teilnehmers

Mechanismen zur Fehlererkennung

- Ein Empfänger kennzeichnet eine fehlerhafte Nachricht dadurch, dass er kein ACK-Slot sendet. Weiterhin kann ein fehlerhaftes Telegramm durch einen Error Frame überschrieben werden, damit es keiner empfangen kann. Der Sender unternimmt dann einen neuen Versuch.
- Eine dauernde Blockade des Busses durch fehlerhafte Telegramme wird durch eine vorgebbare Anzahl ausgesendeter Fehlertelegramme verhindert. Der Teilnehmer begibt sich in den Passivmodus. Nach einer bestimmten Zeit schaltet er sich ab.

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

7. Echtzeitkommunikation

7.1 Steuerungsnetzwerke

7.2 Netzwerkgrundlagen

7.3 Feldbusse

7.4 Interbus nach DIN 19258

7.5 CAN – Controller Area Network

7.6 Profibus – Process Field Bus nach DIN 19245

7.7 RTnet – Real-Time Ethernet

7.8 RACK – Robotics Application Construction Kit

© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Profibus - Process Field Bus nach DIN 19245

- RS 485
- 9,6 bis 1500 kbps (1200 - 200 m)
- Linienbus (max. 32/127 Teilnehmer - ohne/mit Repeater)
- hybrides Buszugriffsverfahren: Master / Slave + Token-Passing
- Multimaster (Sendeberechtigung wird durch einen Token geregelt, der zyklisch im Ring (kooperativ) weitergegeben wird)
- Ausprägungen:
  - Profibus-TF (Technologische Funktionen)
  - Profibus-FMS (Fieldbus Messaging Specification)
  - Profibus-DP (Dezentrale Peripherie)

7. Echtzeitkommunikation

7.1 Steuerungsnetzwerke

7.2 Netzwerkgrundlagen

7.3 Feldbusse

7.4 Interbus nach DIN 19258

7.5 CAN – Controller Area Network

7.6 Profibus – Process Field Bus nach DIN 19245

7.7 RTnet – Real-Time Ethernet

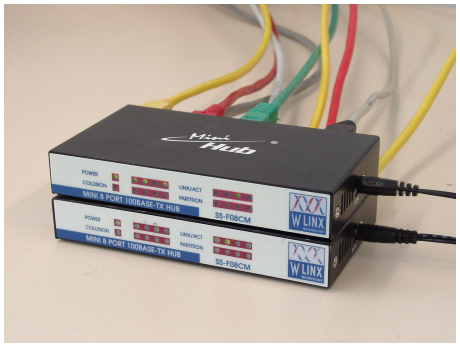
7.8 RACK – Robotics Application Construction Kit

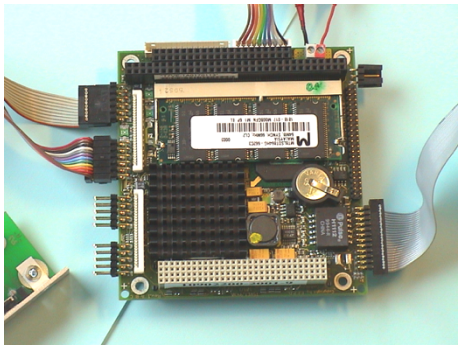
RTnet – Real-Time Ethernet

Ethernet-Technologie:

- Kostengünstige Komponenten
  - ➔ Stecker, Kabel, Verteiler, Interfaces ...
- Hohe Übertragungsrate
  - ➔ 10/100 MBit/s bis 10 GBit/s
- Ein-Kabel-Lösung
  - ➔ Echtzeit-Daten und Standard-TCP/IP über einen Anschluss

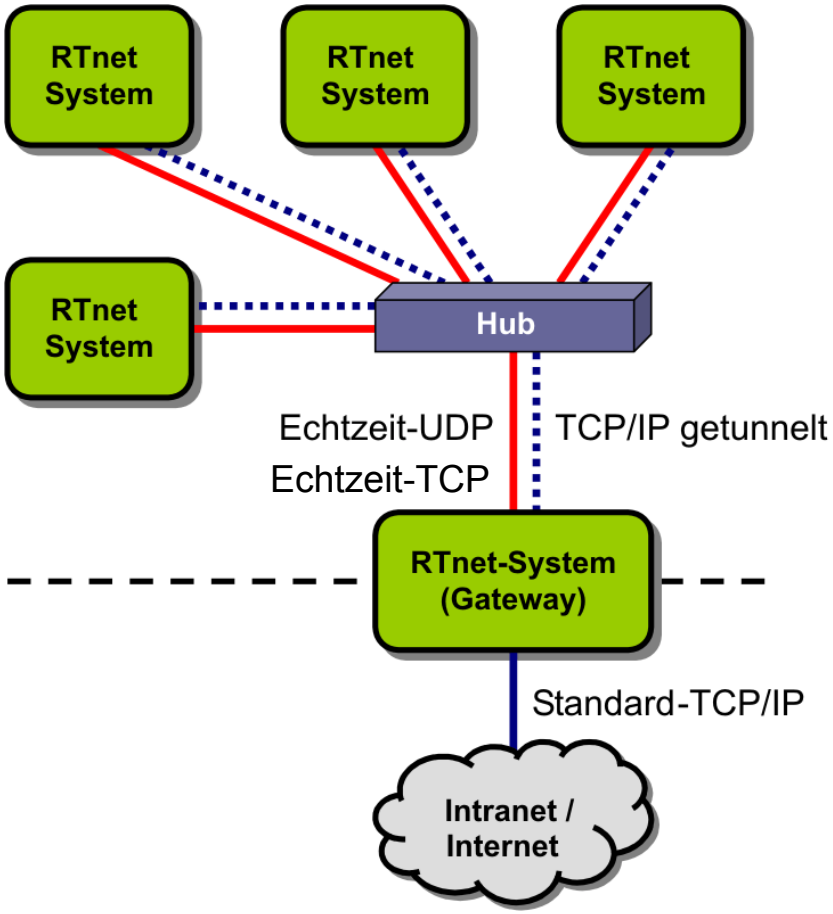
► Software-Lösung als Open Source
  - ➔ Hoch flexibel
  - ➔ Herstellerunabhängig, langfristig verfügbar
  - ➔ Lizenzkostenfrei





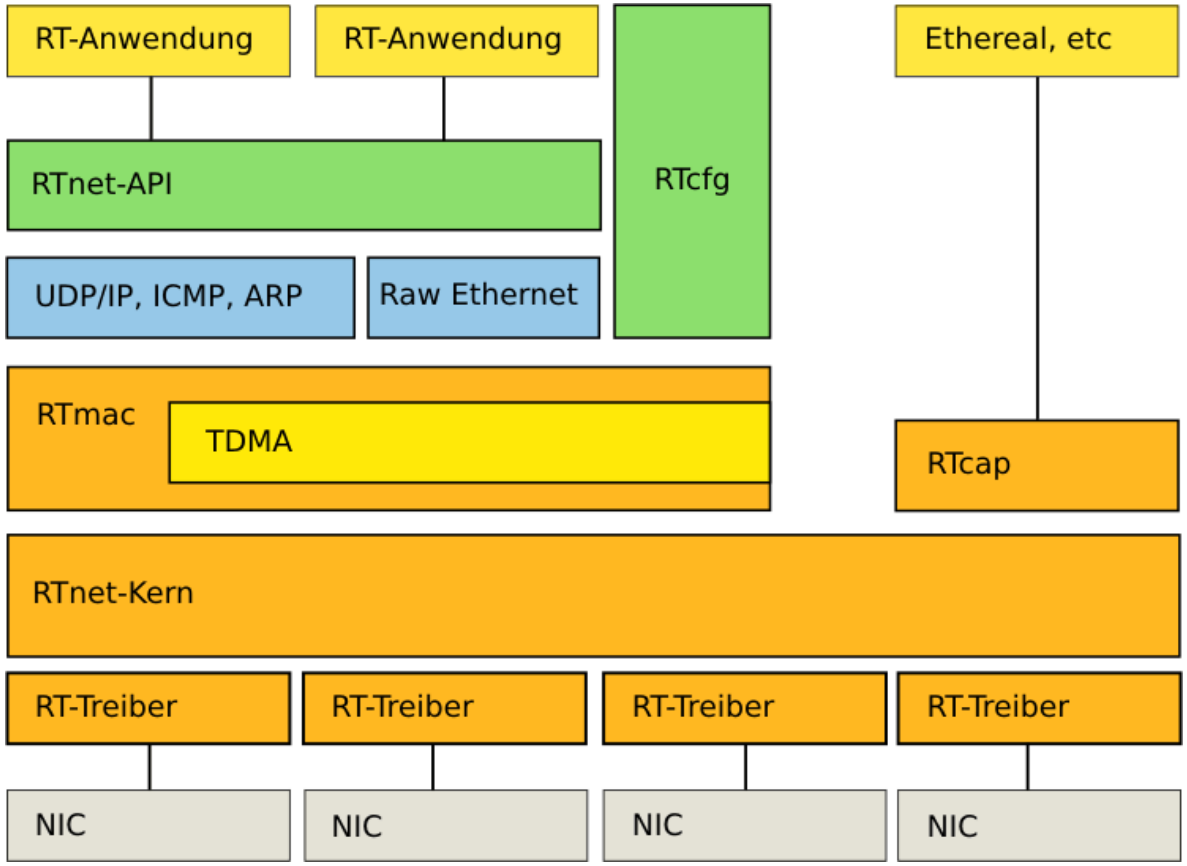
Struktur eines RTnet-Netzwerks

- Dediziertes Netzwerk
- Keine Einschränkung der Netzwerkstruktur
- Passive Verteiler (Hubs)
- Switches verwendbar (höhere Latenz, keine Vorteile)
- Protokolltunnel zur Weiterleitung von TCP/IP
- Gateway stellt Verbindung zum Intra-/Internet her



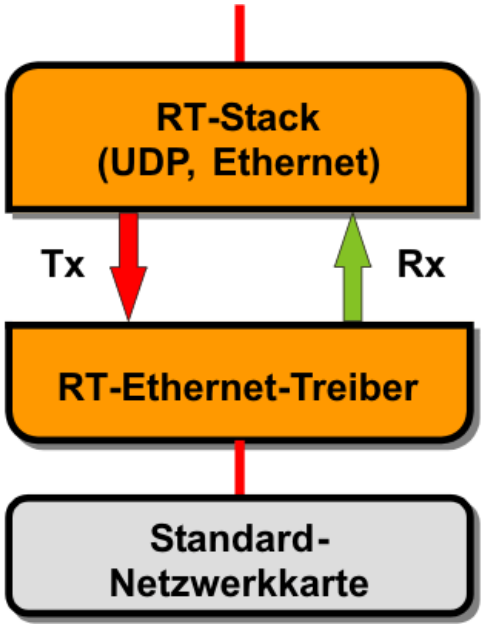
Modularer Stack-Aufbau

- RTDM-Socket API
- RT-TCP/IP eingeschränkt möglich
- Erweiterbarer Stack (Layer 3 und 4)
- Optionale Medienzugriffskontrolle (RTmac)
- Konfigurationsdienst RTcfg
- Echtzeit-Capturing
- Linux-ähnliche NIC-Treiberschicht (NIC = Network Interface Card, *Netzwerkkarte*)
- Virtueller Linux Netzwerk-Adapter

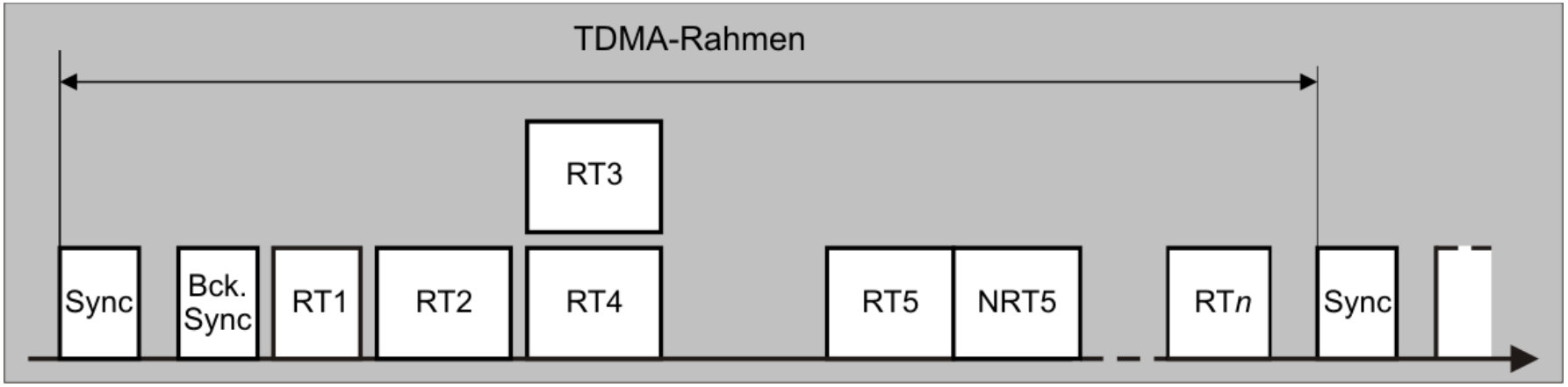


Echtzeitfähiges Ethernet

- Anforderungen an Protokoll-Stack
  - Deterministische Ausführungsdauer
  - Entkoppelung der lokalen Nutzer
- Anforderungen an Übertragung
  - Keine Kollisionen
  - Garantierte Bandbreite (QoS)
- RTmac Plug-In steuert Sendezugriff
- Unterschiedliche Zugriffsverfahren möglich

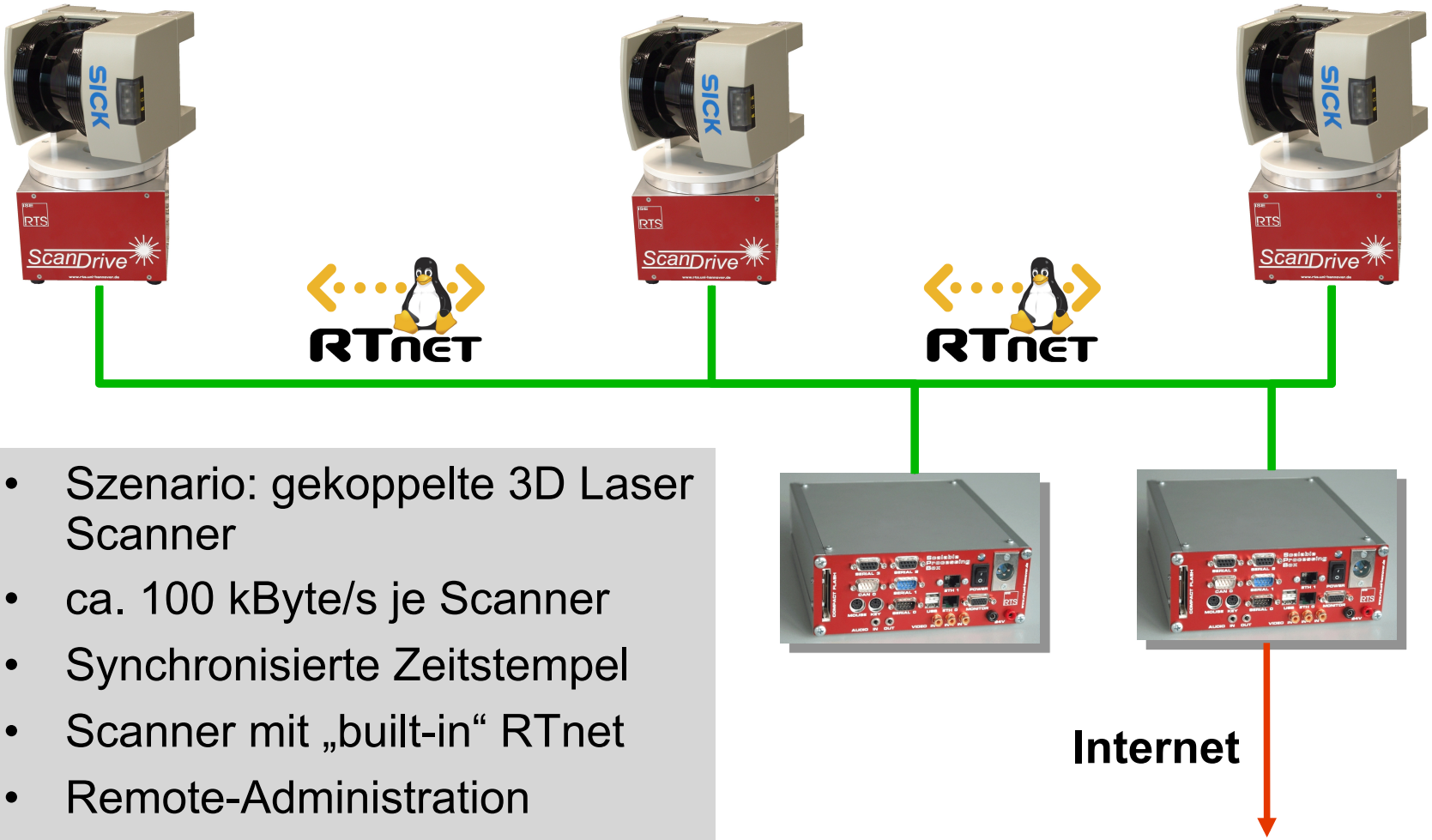


TDMA (Time Division Multiple Access)



- Master oder Backup-Master senden periodische Sync-Frames
- Freie Sende-Slot Konfiguration
  - Stationszuordnung
  - Offset zum Sync-Frame
  - Periodizität und Phase => alternierende Slot-Nutzung
  - Inhalt (Echtzeit/Nicht-Echtzeit)
- Uhrensynchronisation

Verteilte Echtzeitanwendungen



- Szenario: gekoppelte 3D Laser Scanner
- ca. 100 kByte/s je Scanner
- Synchronisierte Zeitstempel
- Scanner mit „built-in“ RTnet
- Remote-Administration

7. Echtzeitkommunikation

7.1 Steuerungsnetzwerke

7.2 Netzwerkgrundlagen

7.3 Feldbusse

7.4 Interbus nach DIN 19258

7.5 CAN – Controller Area Network

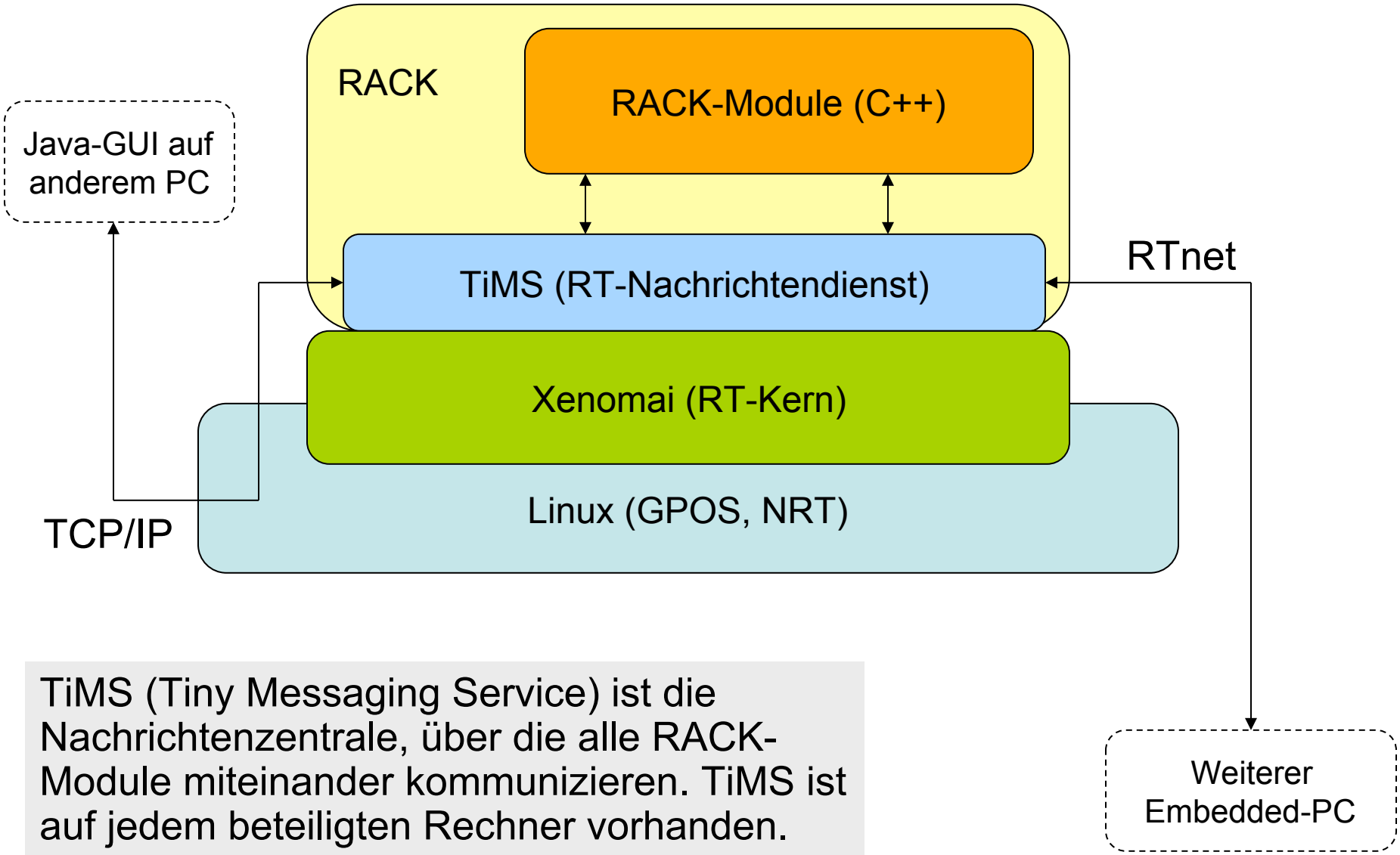
7.6 Profibus – Process Field Bus nach DIN 19245

7.7 RTnet – Real-Time Ethernet

7.8 RACK – Robotics Application Construction Kit

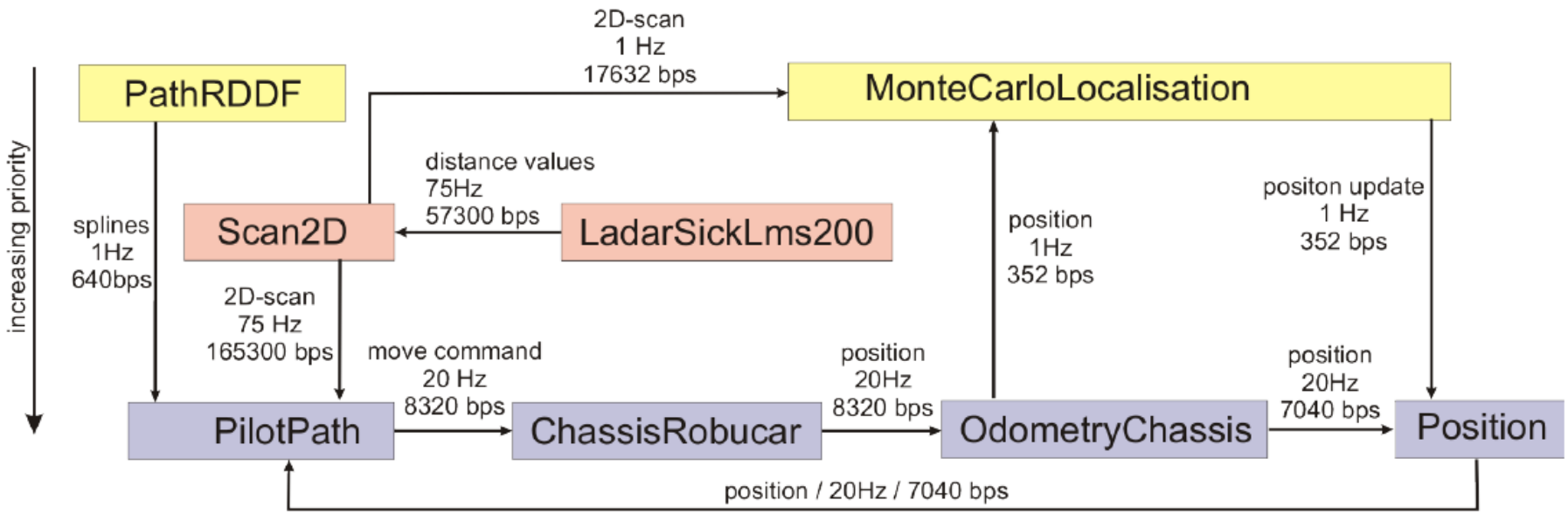
© 2015 B. Wagner - Leibniz Universität Hannover - RTS

Echtzeitfähige Robotik-Middleware RACK



TiMS (Tiny Messaging Service) ist die Nachrichtenzentrale, über die alle RACK-Module miteinander kommunizieren. TiMS ist auf jedem beteiligten Rechner vorhanden.

Beispiel: RACK-Module auf einem Roboter



Jedes RACK-Modul hat eine spezifische Aufgabe und tauscht Daten mit anderen Modulen mit Hilfe von klar definierte Nachrichten aus. So werden Funktionalitäten gekapselt und können gleichzeitig abstrahiert werden (low-level: Einlesen von Scan-Daten, high-level: Navigationsalgorithmus). Außerdem kann RACK nahezu unverändert auf einer Vielzahl von Robotern betrieben werden, indem nur die Module ausgetauscht werden, die mit unterschiedlicher Hardware interagieren müssen (z.B. ChassisRobucar oder LadarSickLms200).

Mailboxen und Proxies

- **Mailbox**: Quelle und/oder Senke von Nachrichten
- RACK-Module können über Mailboxen dreier Arten verfügen:
  - Command-Mailbox (immer)
  - Work-Mailbox (Ursprung einer Einzelanfrage)
  - Daten-Mailbox
    - ggf. mehrere
    - „Abonnement“ der Datenströme anderer Module (Publish/ Subscribe-Prinzip)
- **Proxies**:
  - ermöglichen RPC (Remote Procedure Calls), siehe auch Kapitel 6, Folie 63
  - Kommandos an andere Module über Proxies

Zusammenfassung

In einem Echtzeitsystem muss auch die Kommunikation zwischen den Komponenten und der Peripherie in Echtzeit erfolgen. Das bedeutet, dass auch der Medienzugriff und die Übertragung innerhalb fester, bekannter zeitlicher Grenzen erfolgt.

Es wurden mehrere Feldbusse und eine echtzeitfähige Middleware vorgestellt:

- **Interbus** (synchrone, zyklische Datenübertragung für SPSen)
- **CAN** (asynchrone Nachrichten mit Prioritäten)
- Profibus (Busfamilie für die Automatisierungstechnik)
- **RTnet** (zeitscheibenbasierter Medienzugriff)
- **RACK** (Robotik-Middleware mit Echtzeit-Nachrichten)

© 2015 B. Wagner - Leibniz Universität Hannover - RTS