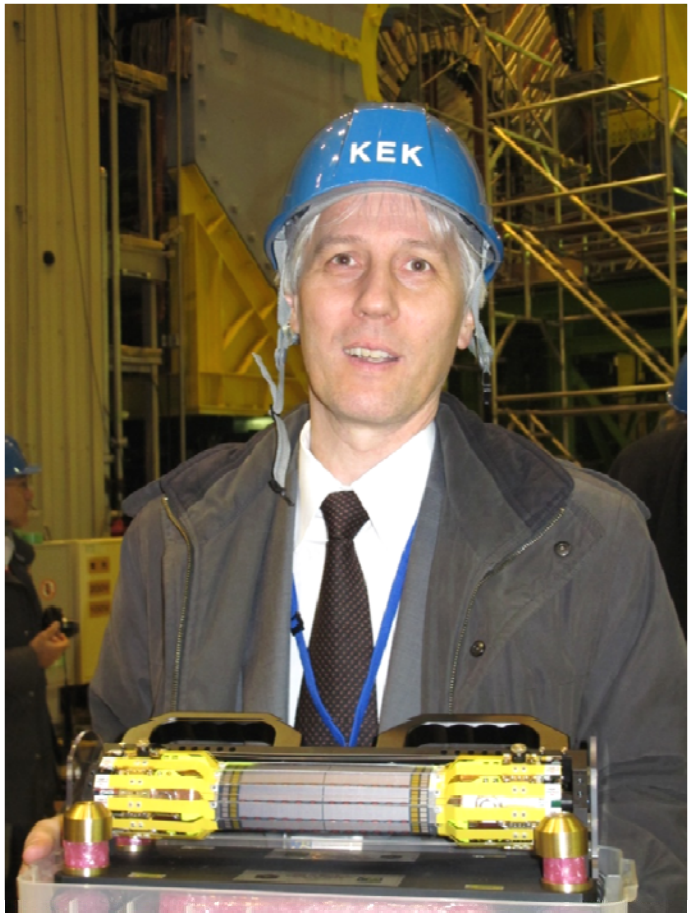

セキショウ
チャレンジ
スタジアム
つくば市消防
本部北消防署
クリーン
センター入口
ヒルトップ
クリニック
国立公文
書館分館
Kamisawa
上沢
408
Sa
佐
吉村百科
高エネルギー
加速器研究機構
Oho
大穂
高エネルギー
加速器研究
機構図書室
商工会
大穂支所

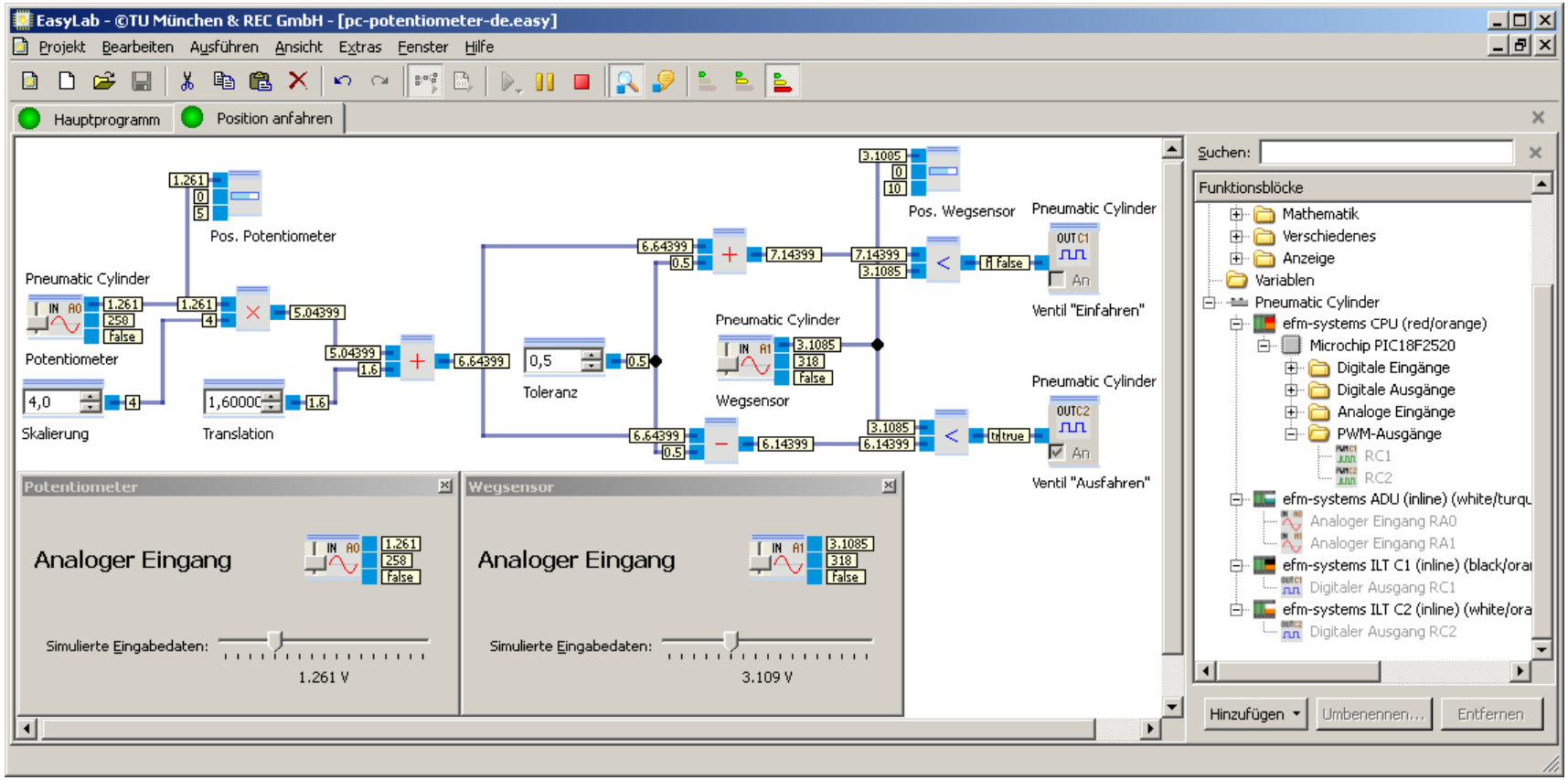

KEK

Modellierung von Echtzeitsystemen

Synchroner Datenfluss

Werkzeug: EasyLab

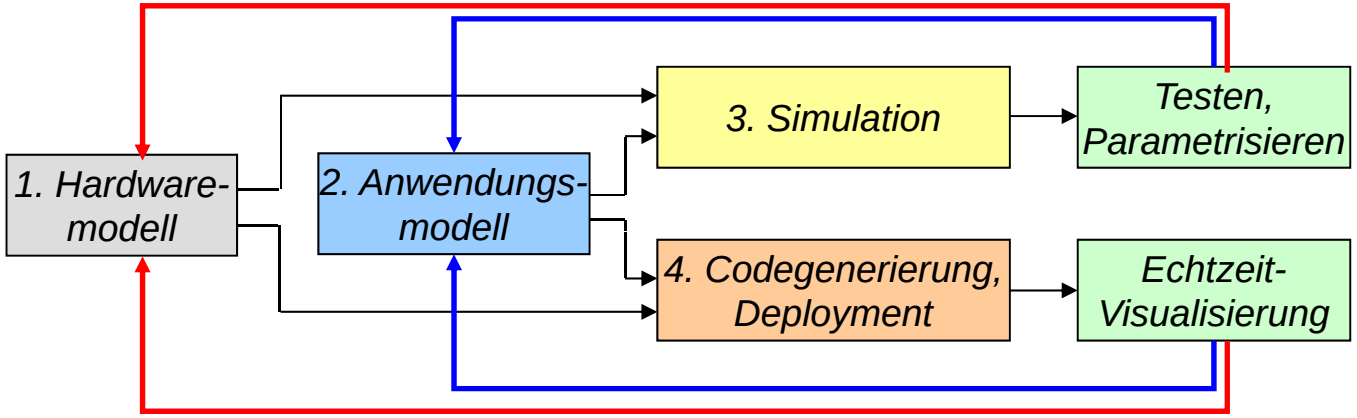
EasyLab



Beispielanwendung zur Regelung der Position eines Pneumatikzylinders

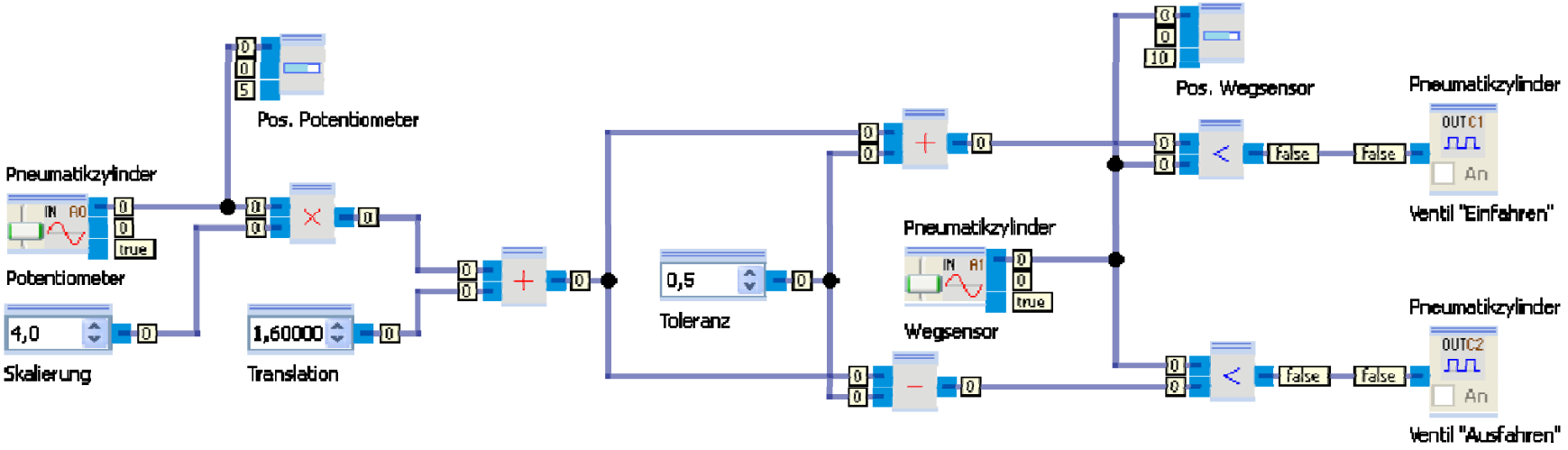
Entwicklungsprozess in EasyLab

1. Spezifikation der Zielhardware
2. Modellierung der Zustandslogik sowie der abzuarbeitenden Aufgabe je Zustand
3. Simulation des Programms zum Testen und zur Erkennung von Fehlern
4. Codegenerierung und Echtzeit-Visualisierung des Zustands der Zielhardware



Synchroner Datenfluss

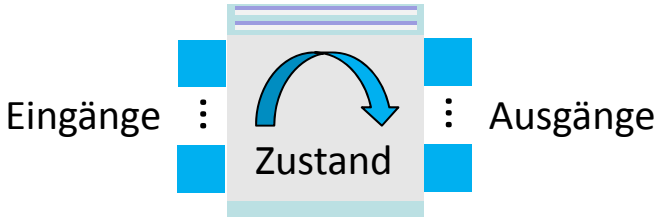
- Grundlagen
  - Synchronitätshypothese
  - „Black boxes“-Sicht
- Effizienz und Zuverlässigkeit
  - Berechnung statischer Schedules
  - Deterministisches Laufzeitverhalten
  - Statische Arbeitsspeicherallokation



Edward A. Lee and David G. Messerschmitt, “Static scheduling of synchronous data flow programs for digital signal processing,” IEEE Trans. Comput., vol. 36, no. 1, pp. 24–35, 1987

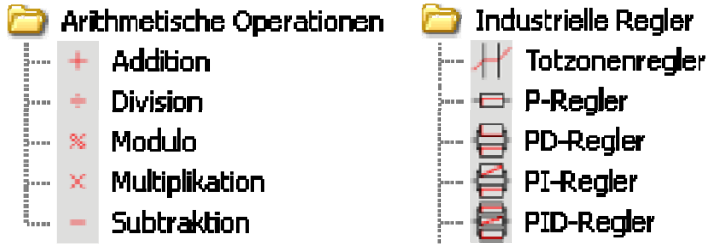
Aktoren (Funktionsblöcke)

- Beschrieben durch:
  - Typisierte Ein- und Ausgänge
  - Typisierte Zustandsvariablen
  - Zugeordnete Aktionen
    - Hardware-unabhängige Aktoren, z.B. mathematische Berechnungen, Regler
    - Hardware-abhängige Aktoren, z.B. Sensoren und Aktorik („Geräte“)



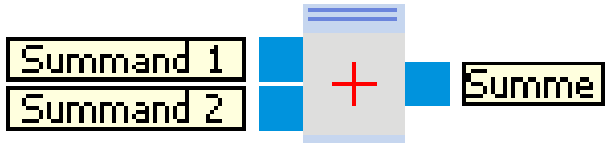
- Komposition von Aktoren
  - Typisierte Verbindungen

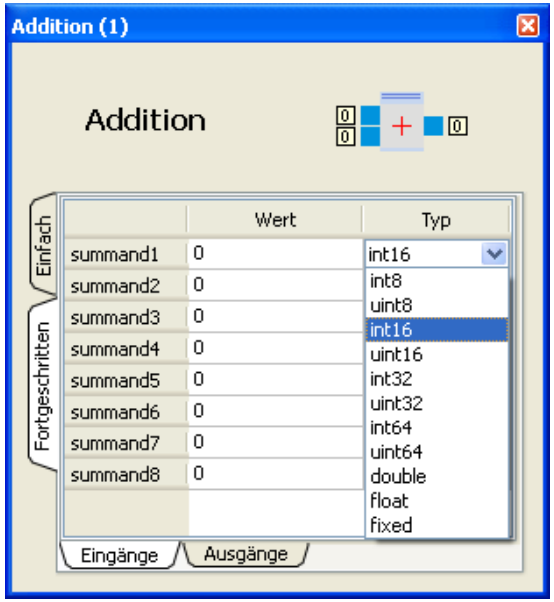
- Beispiele:



Überladen von Aktoren

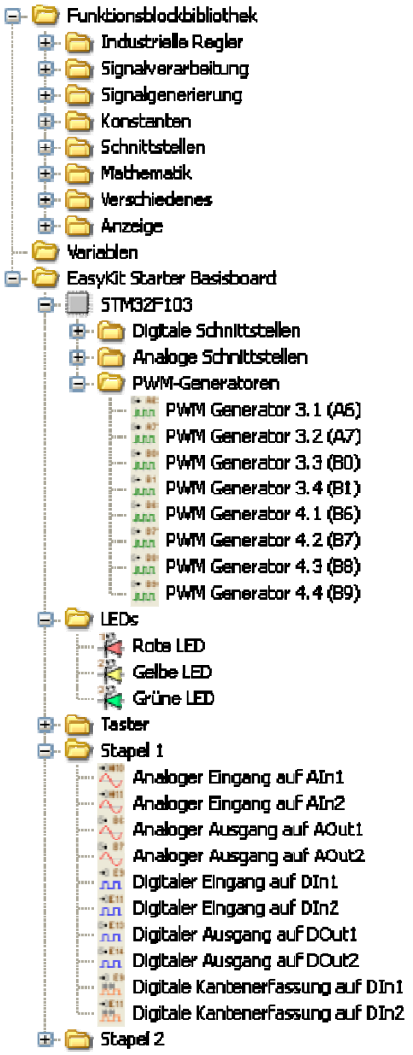
- Aktoren können typunabhängig angeboten werden
- Der Typ eines Aktors ist solange frei wählbar, bis sich durch Anschlussbelegung der Typ automatisch ergibt





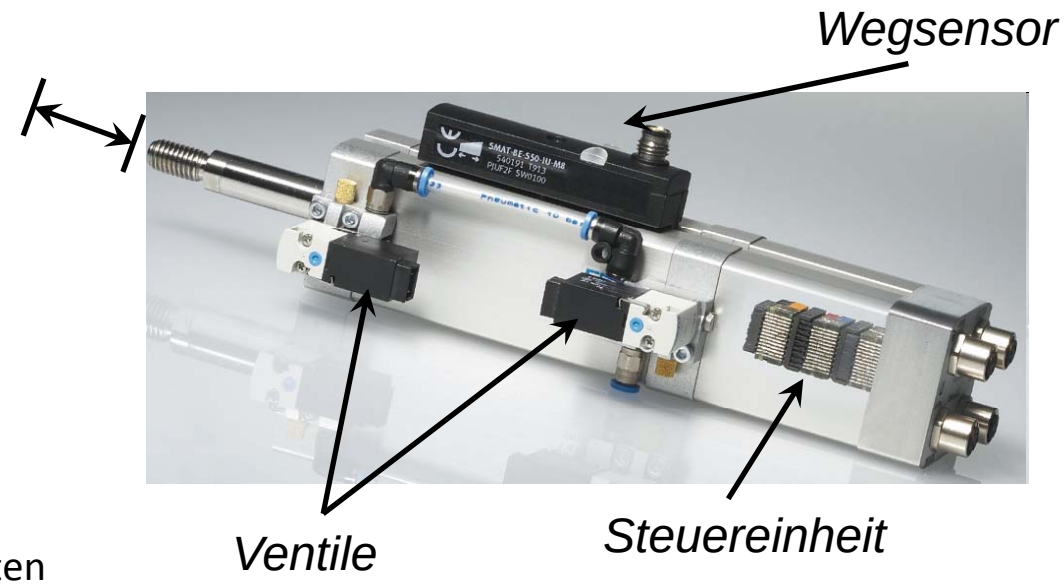
Geräte

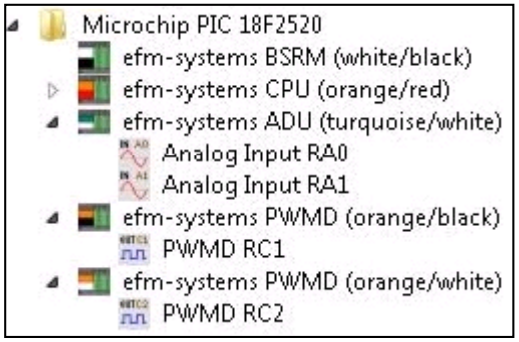
- Hardware
  - Mikrocontroller
  - Sensoren und Aktoren
- Geräte
  - Hierarchische Beschreibung der Hardware
  - Ressourcenmanagement
    - Modellierung der Hardwarefunktionalität
    - Beispiele: I/O-Pins, ADCs, Timer,
  - Schnittstelle zur Anwendungslogik
    - Hardwarezugriff
    - Geräte bieten eine Menge von Funktionsblöcken an



Anwendung – Pneumatischer Zylinder

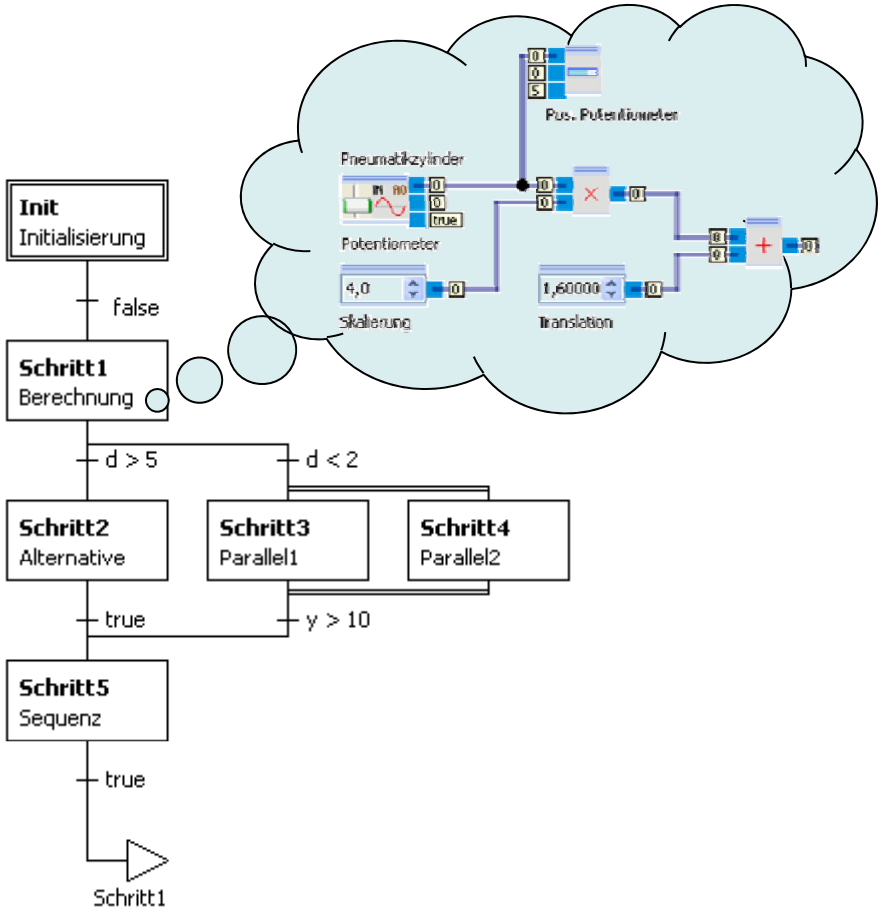
- Hardware
  - Zylinder
    - Positionssensor (Kolben)
    - Endlagenschalter
    - Zwei Magnetventile
  - Steuerungseinheit
    - Mikrocontroller
    - Analog-Digital-Wandler
    - Treiber für induktive Lasten
- Ziel: Positionssteuerung des Kolbens
- Umsetzung
  - Hardware-Modell aus Bibliothek für Match-X
  - Anwendungsmodell
    - Kleines Datenflussdiagramm
    - Integration der Hardwarefunktionalität





EasyLab: Zustandsfluss-Diagramme

- Zustandsfluss-Diagramme
  - An IEC-61131-3 angelehnt
  - Zustand: Referenz auf SDF-Modell
  - Transitionsbedingungen: Boolesche Ausdrücke mit Variablen
  - Zustände referenzieren Datenfluss-Modelle
- Komposition von Zuständen
  - Zustandsfolgen
  - Alternativ- und Parallelzweige
  - Sprünge
- Vorteile
  - Modellierung von Zuständen (vgl. Automaten)
  - Diagrammart weit verbreitet in Anwendungsdomäne



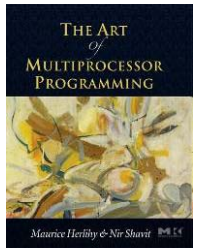
Kapitel 4

Nebenläufigkeit

Inhalt

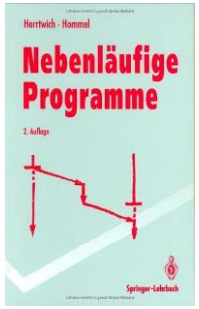
- Motivation
- Unterbrechungen (Interrupts)
- (Software-) Prozesse
- Threads
- Interprozesskommunikation (IPC)

Literatur



Maurice Herlihy, Nir Shavit,
The Art of Multiprocessor
Programming, 2008

A.S.Tanenbaum, Moderne
Betriebssysteme, 2002



R.G.Herrtwich, G.Hommel,
Nebenläufige Programme
1998

- Edward Lee: The Problem with Threads:
  http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-1.pdf
- http://www.beyondlogic.org/interrupts/interupt.htm
- http://www.llnl.gov/computing/tutorials/pthreads/

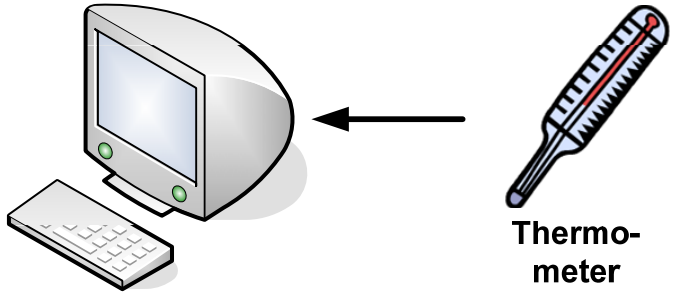
Definition von Nebenläufigkeit

- **Allgemeine Bedeutung**: Nebenläufige Ereignisse sind nicht kausal abhängig. Ereignisse (bzw. Ereignisfolgen) sind dann nebenläufig, wenn keines eine Ursache im anderen hat.
- **Bedeutung in der Informatik**: Nebenläufig bezeichnet hier die Eigenschaft von Programmcodes, nicht linear hintereinander ausgeführt werden zu müssen, sondern zeitlich parallel zueinander ausführbar zu sein.
- Aktionen (Programmschritte) können parallel (gleichzeitig oder quasi gleichzeitig) ausgeführt werden, wenn keine das Resultat der anderen benötigt. Die parallele Ausführung von mehreren unabhängigen *Prozessen* (siehe später) auf einem oder mehreren Prozessoren bezeichnet man als *Multitasking*. Die parallele Ausführung von Teilsequenzen innerhalb eines Prozesses heißt *Multithreading*.

Motivation

- Gründe für nebenläufige Ausführung von Programmen in Echtzeitsystemen:
  - Echtzeitsysteme sind häufig verteilte Systeme (Systeme mit mehrere Prozessoren).
  - Zumeist werden zeitkritische und zeitunkritische Aufgaben parallel berechnet.
  - Bei reaktiven Systemen ist die maximale Antwortzeit häufig limitiert.
  - Abbildung der parallelen Abläufe im technischen Prozeß
- Aber: kleinere (Monoprozessor-)Echtzeit-Systeme verzichten häufig auf die parallele Ausführung von Code, weil der Aufwand für die Prozeßverwaltung zu hoch ist.
  Dennoch auch hier: typischerweise Parallelverarbeitung in „Hauptprogramm“ und „Unterbrechungsbehandler“ (interrupt service routine, interrupt handler)

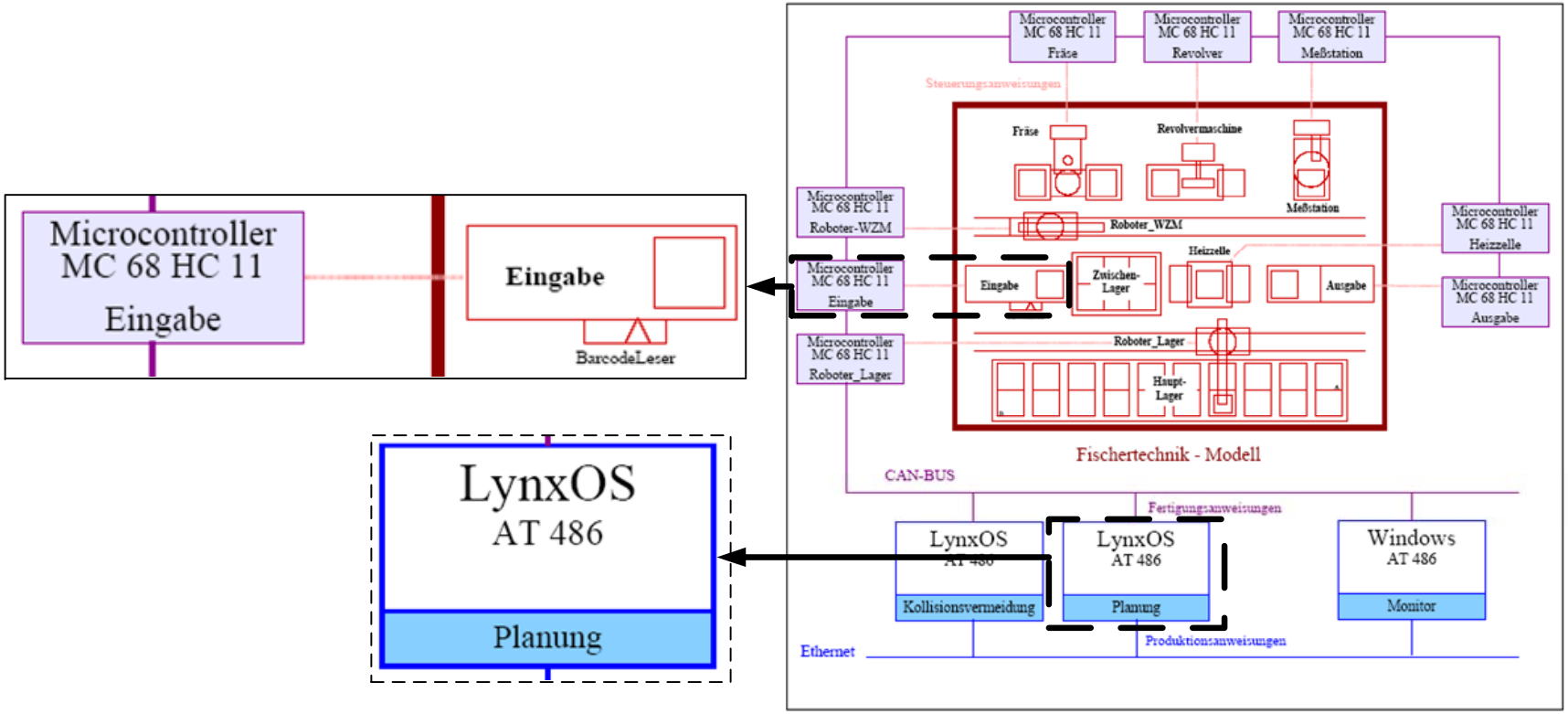
Anwendungsfälle für Nebenläufigkeit (Unterbrechungen)



Signal falls Temperaturwert überschritten wird
⇨ **Unterbrechungen (interrupts)**

Allgemeines Anwendungsgebiet: hauptsächlich zur Anbindung von externer Hardware

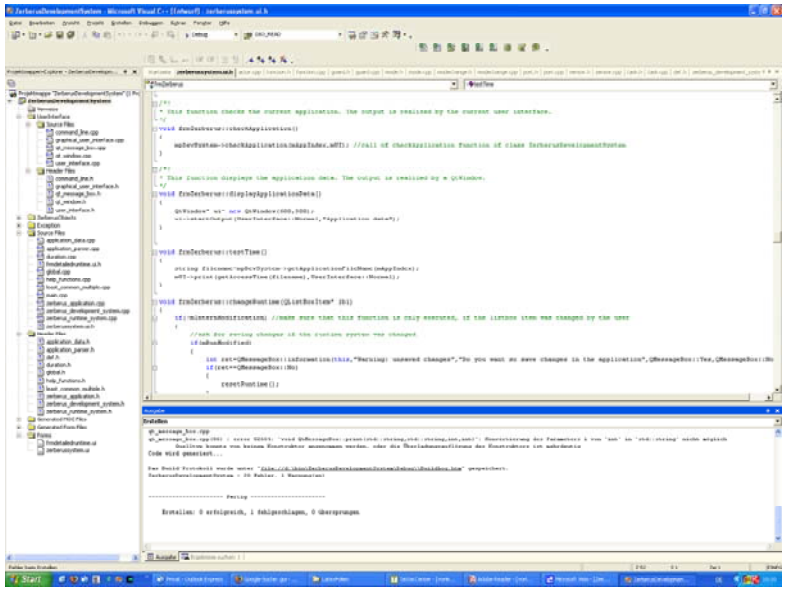
Anwendungsfälle für Nebenläufigkeit (Prozesse)



Verteiltes System zur Steuerung der Industrieanlage ⇨ **Prozesse (tasks)**

Allgemeine Anwendungsgebiete: verteilte Systeme, unterschiedlichen Anwendungen auf einem Prozessor

Anwendungsfälle für Nebenläufigkeit (Threads)



Reaktion auf Nutzereingaben trotz Berechnungen (z.B. Übersetzen eines Programms)

⇨ **leichtgewichtige Prozesse (Threads)**

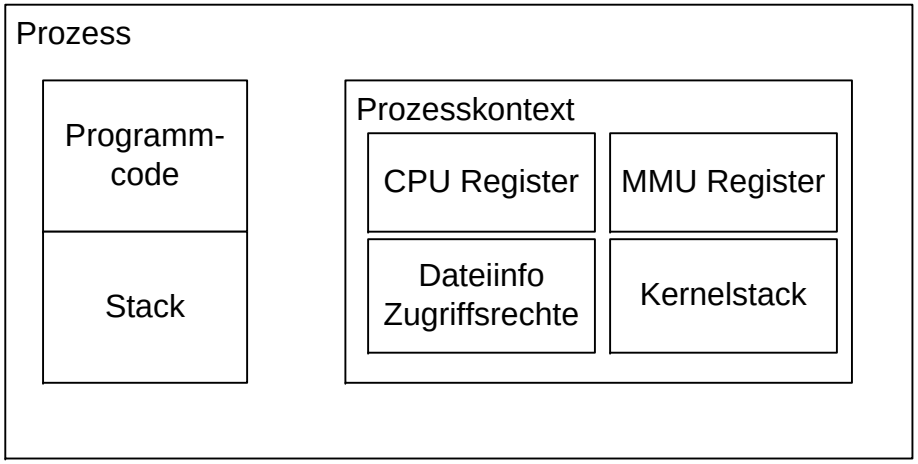
Allgemeines Anwendungsgebiet: unterschiedliche Berechnungen im gleichen Anwendungskontext

Nebenläufigkeit

Prozesse

Definition

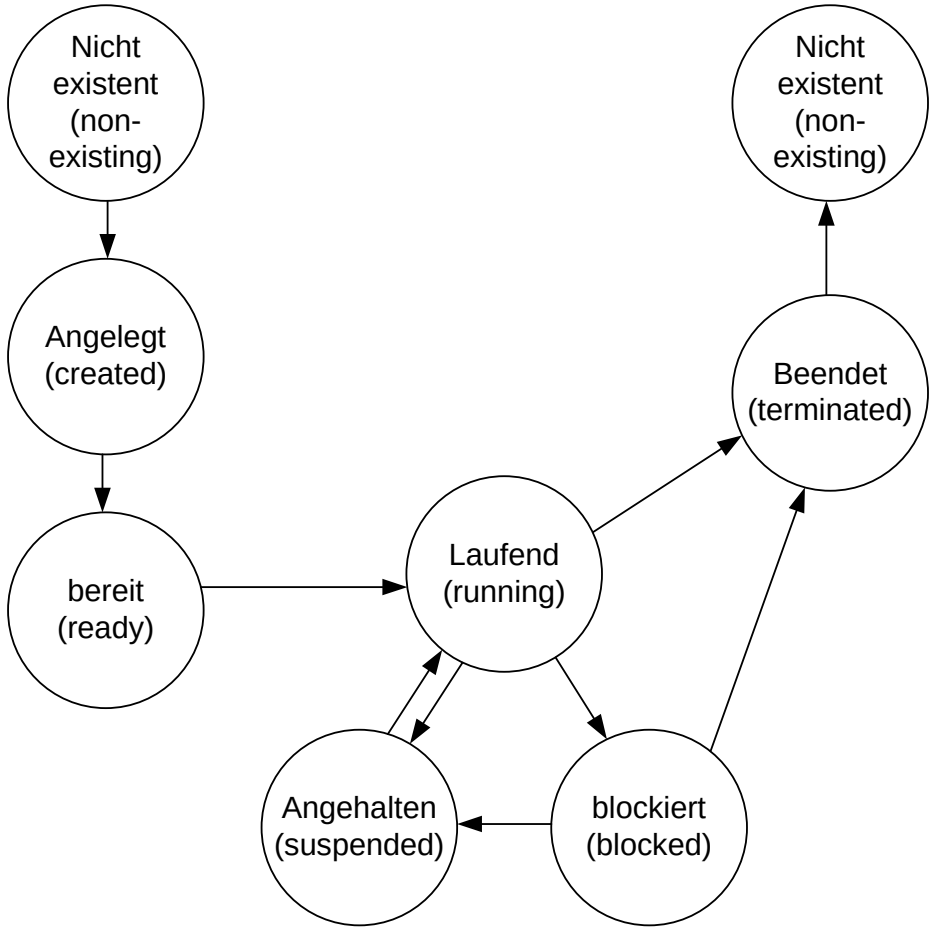
- **Prozess**: Abstraktion eines sich in Ausführung befindlichen Programms
- Die gesamte Zustandsinformation der Betriebsmittel für ein Programm wird als eine Einheit angesehen und als Prozess bezeichnet.
- Prozesse können weitere Prozesse erzeugen ⇨ Vater-,Kinderprozesse.



Prozessausführung

- Zur Prozessausführung werden diverse Resourcen benötigt, u.a.:
  - Prozessorzeit
  - Speicher
  - sonstige Betriebsmittel (z.B. spezielle Hardware)
- Die Ausführungszeit ist neben dem Programm abhängig von:
  - Leistungsfähigkeit des Prozessors
  - Verfügbarkeit der Betriebsmittel
  - Eingabeparametern
  - Verzögerungen durch andere (wichtigere) Aufgaben

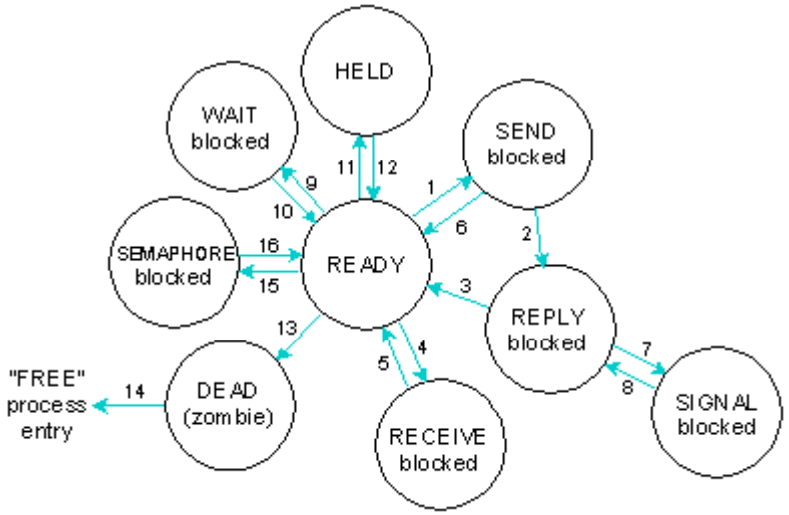
Prozesszustände (allgemein)



Prozesse in QNX[1]

The transactions depicted in the previous diagram are as follows:

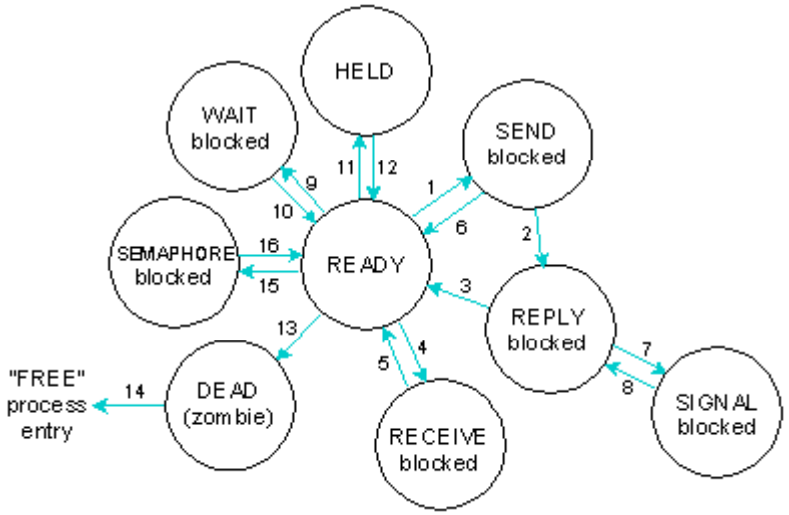
1. Process sends message.
2. Target process receives message.
3. Target process replies.
4. Process waits for message.
5. Process receives message.
6. Signal unblocks process.
7. Signal attempts to unblock process; target has requested message signal catching.
8. Target process receives signal message.



[1] http://www.qnx.com/developers/docs/qnx_4.25_docs/qnx4/sysarch/proc.html#LIFECYCLE

Prozesse in QNX

9. Process waits on death of child.
10. Child dies or signal unblocks process.
11. SIGSTOP set on process.
12. SIGCONT set on process.
13. Process dies.
14. Parent waits on death, terminates itself or has already terminated.
15. Process calls *semwait()* on a non-positive semaphore.
16. Another process calls *sempost()* or an unmasked signal is delivered.



Fragen bei der Implementierung

- Welche Betriebsmittel sind notwendig?
- Welche Ausführungszeiten besitzen einzelne Prozesse?
- Wie können Prozesse kommunizieren?
- Wann soll welcher Prozess ausgeführt werden?
- Wie können Prozesse synchronisiert werden?

Klassifikation von Prozessen

- periodisch vs. aperiodisch
- statisch vs. dynamisch
- Wichtigkeit der Prozesse (kritisch, notwendig, nicht notwendig)
- speicherresident vs. verdrängbar
- Prozesse können auf
  - einem Rechner (Pseudoparallelismus)
  - einem Multiprozessorsystem mit Zugriff auf gemeinsamen Speicher
  - oder auf einem Multiprozessorsystem ohne gemeinsamen Speicher

  ausgeführt werden.

Nebenläufigkeit

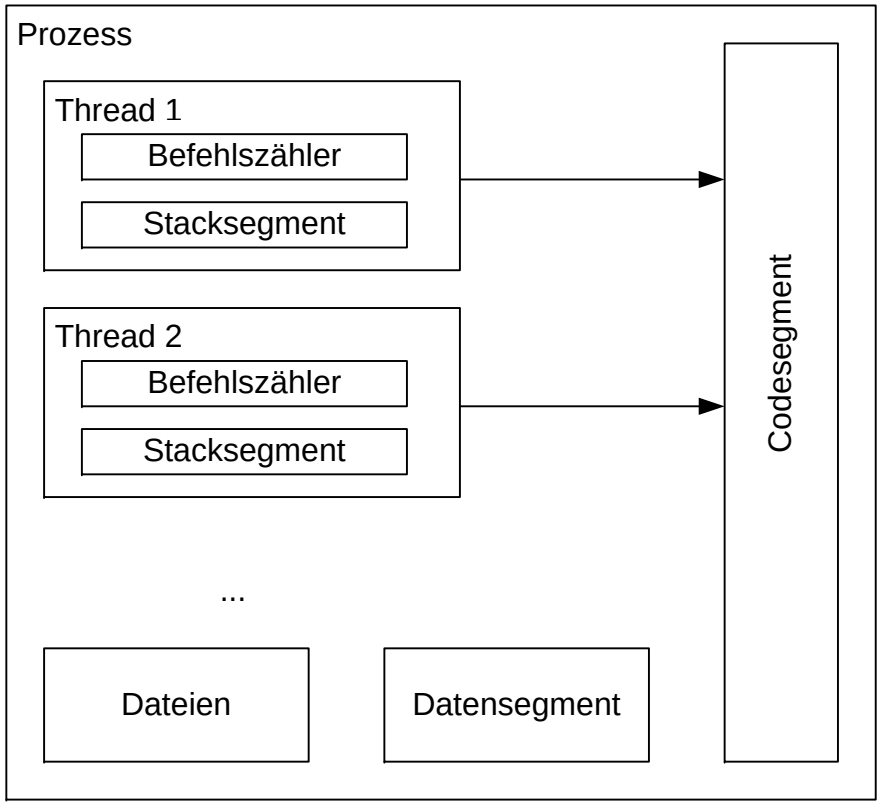
Threads

Leichtgewichtige Prozesse (Threads)

- Der Speicherbedarf von Prozessen ist in der Regel groß (CPU-Daten, Statusinformationen, Angaben zu Dateien und EA-Geräten...).
- Bei Prozesswechsel müssen die Prozessdaten ausgetauscht werden $\Rightarrow$ hohe Systemlast, zeitaufwendig.
- Viele Systeme erfordern keine komplett neuen Prozesse.
- Vielmehr sind Programmabläufe nötig, die auf den gleichen Prozessdaten arbeiten.

⇨ Einführung von Threads

Threads


Prozess
Thread 1
Befehlszähler
Stacksegment
Thread 2
Befehlszähler
Stacksegment
...
Dateien
Datensegment
Codesegment