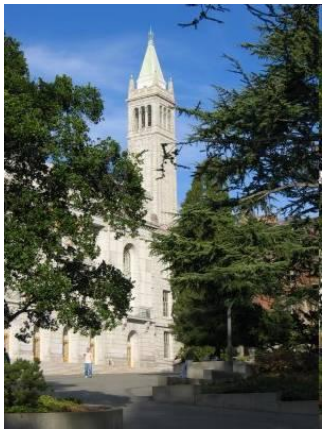
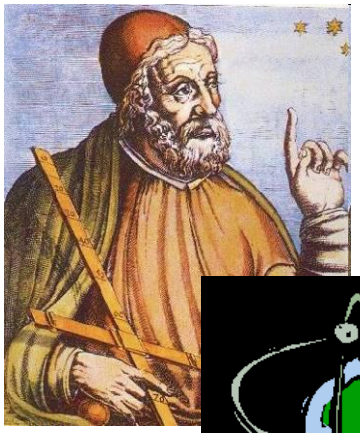
Modellierung von Echtzeitsystemen

Aktoren, Ausführungsmodelle

Werkzeuge: Ptolemy

Ptolemy

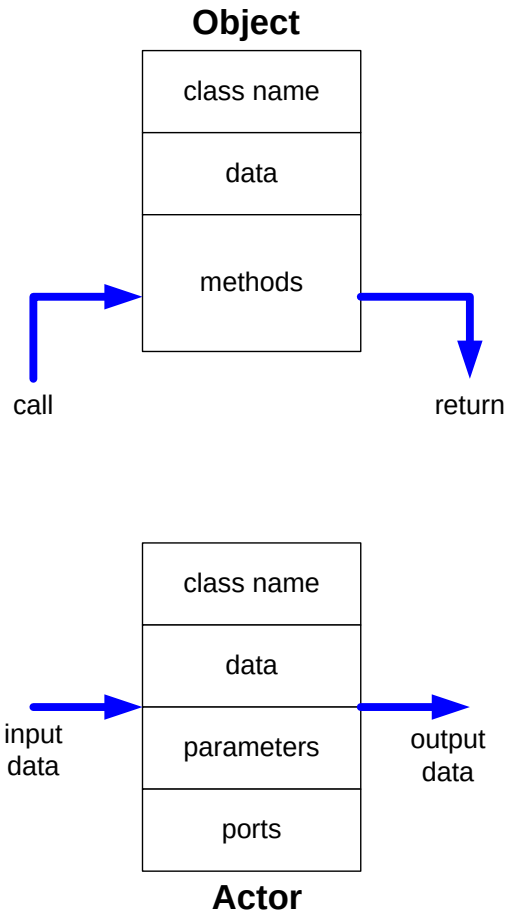




- Das Ptolemy*-Projekt an der UC Berkeley untersucht verschiedene Modellierungs-methodiken für eingebettete Systeme mit einem Fokus auf verschiedene Ausführungs-modelle (Models of Computation)
- Ptolemy unterstützt
  - Modellierung
  - Simulation
  - Codegenerierung
  - Formale Verifikation (teilweise)
- Weitere Informationen unter: http://ptolemy.eecs.berkeley.edu/

***Claudius Ptolemaeus**, (* um 100, vermutlich in Ptolemais Hermii, Ägypten; † um 175, vermutlich in Alexandria), war ein griechischer Mathematiker, Geograph, Astronom, Astrologe, Musiktheoretiker und Philosoph. Ptolemäus schrieb die *Mathematike Syntaxis* („mathematische Zusammen-stellung“), später *Megiste Syntaxis* („größte Zusammenstellung“), heute *Almagest* (abgeleitet vom Arabischen *al-Majisṭī*) genannte Abhandlung zur Mathematik und Astronomie in 13 Büchern. Sie war bis zum Ende des Mittelalters ein Standardwerk der Astronomie und enthielt neben einem ausführlichen Sternenkatalog eine Verfeinerung des von Hipparchos von Nicäa vorgeschlagenen geozentrischen Weltbildes, das später nach ihm *Ptolemäisches Weltbild* genannt wurde. (Wikipedia)

Ptolemy: Aktororientiertes Design



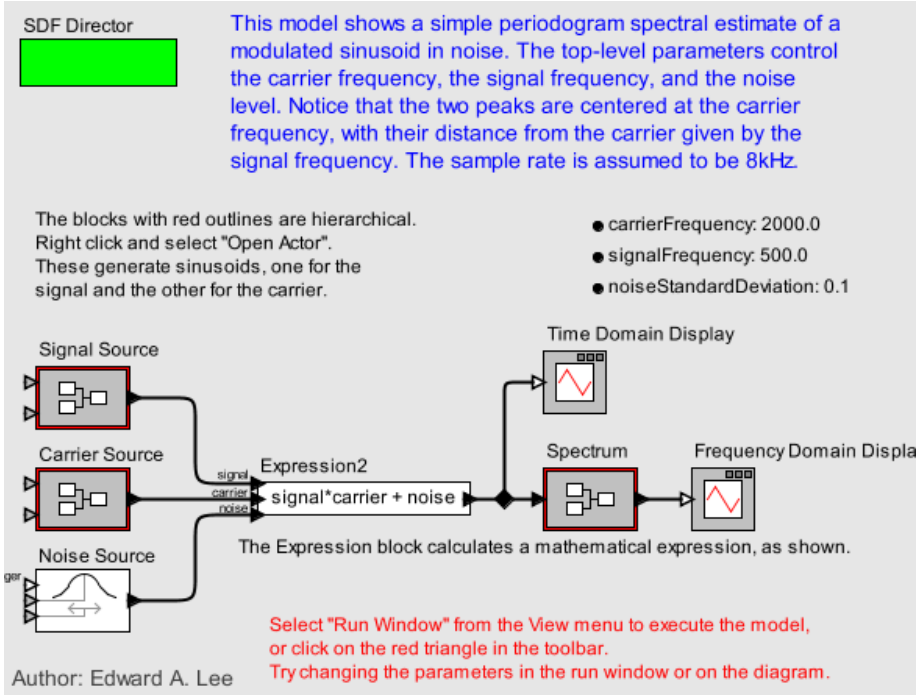
- Ptolemy-Modelle basieren auf Aktoren anstelle von Objekten
- Objekte:
  - Fokus liegt auf Kontrollfluss
  - Objekte werden manipuliert
- Aktoren
  - Fokus liegt auf Datenfluss
  - Aktoren manipulieren das System
- Vorteil beider Ansätze: erhöhte Wiederverwendbarkeit
- Vorteil von Aktoren: leichtere Darstellung von Parallelität

Ptolemy: Ausführungsmodelle

- Ausführungsmodelle (models of computation) bestimmen die Interaktion von Komponenten/Aktoren
- Die Eignung eines Ausführungsmodells hängt von der Anwendungs-domäne, aber auch der verwendeten Hardware, ab
- In Ptolemy wird durch die Einführung von „Dirigenten“ (director) die funktionale Ausführung (Verschaltung der Aktoren) von der zeitlichen Ausführung (Abbildung im Direktor) getrennt.
- Aktoren können unter verschiedenen Ausführungsmodellen verwendet werden (z.B. synchron, asynchron)
- Verschiedene Ausführungsmodelle können hierarchisch geschachtelt werden (modal models).
  - Typisches Beispiel: Synchroner Datenfluss und Zustandsautomaten

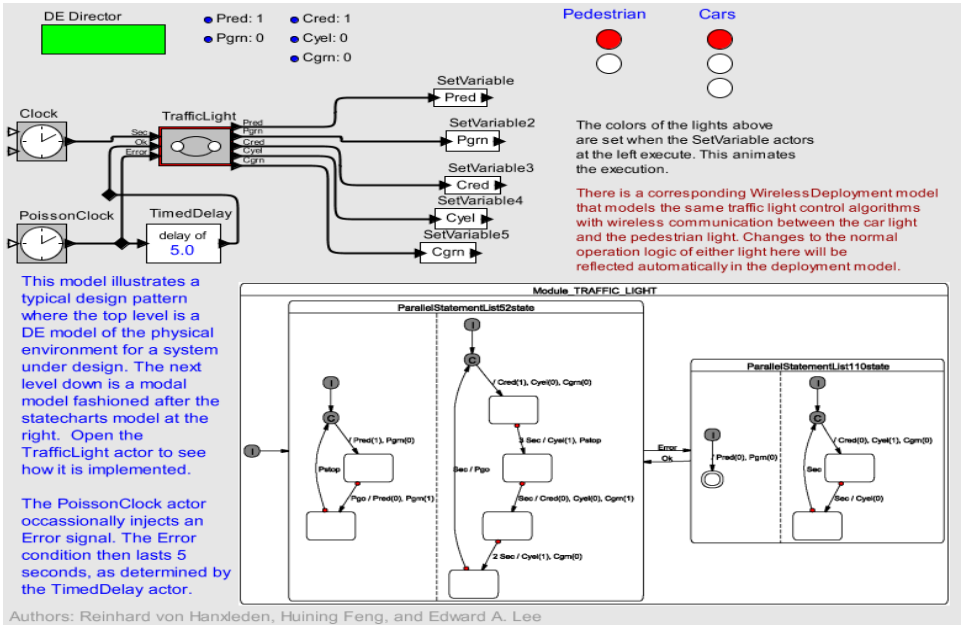
Example Ptolemy Model of Computation: Synchronuous Dataflow

- Prinzip:
  - Annahme: unendlich schnelle Maschine
  - Daten werden zyklisch verarbeitet (zeitgesteuert oder best effort)
  - Pro Runde wird genau einmal der Datenfluss ausgeführt
- Vorteile:
  - Statische Speicherallokation
  - Statischer Schedule berechenbar
  - Verklemmungen detektierbar
  - Laufzeit kann einfach bestimmt werden
- Werkzeuge:
  - Matlab/Simulink
  - Labview
  - EasyLab
- Anwendungsdomänen, u.a.:
  - Industrieautomatisierung (IEC 61131)
  - Regelungssysteme



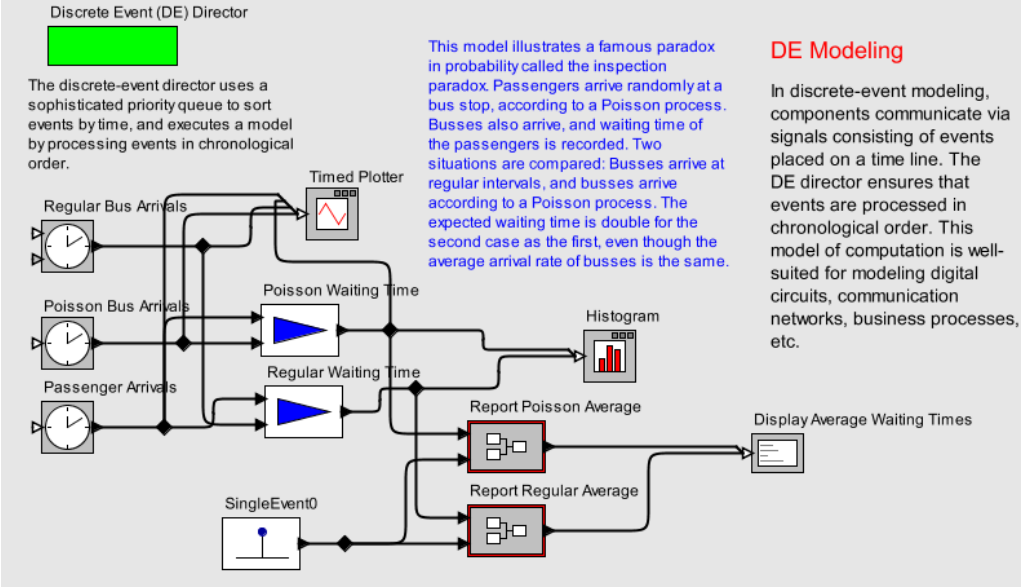
Example Ptolemy Model of Computation: Synchronous Reactive

- Prinzip:
  - Annahme: unendlich schnelle Maschine
  - Diskrete Ereignisse (DE) werden zyklisch verarbeitet (Ereignisse müssen nicht jede Runde eintreffen)
  - Pro Runde wird genau eine Reaktion berechnet
  - Häufig verwendet in Zusammenhang mit Finite State Machines
- Vorteile:
  - einfache formale Verifikation
- Werkzeuge:
  - Esterel Studio
  - Scade
- Anwendungsgebiete:
  - Ereignisbasierte Systeme, z.B. in der Avionik



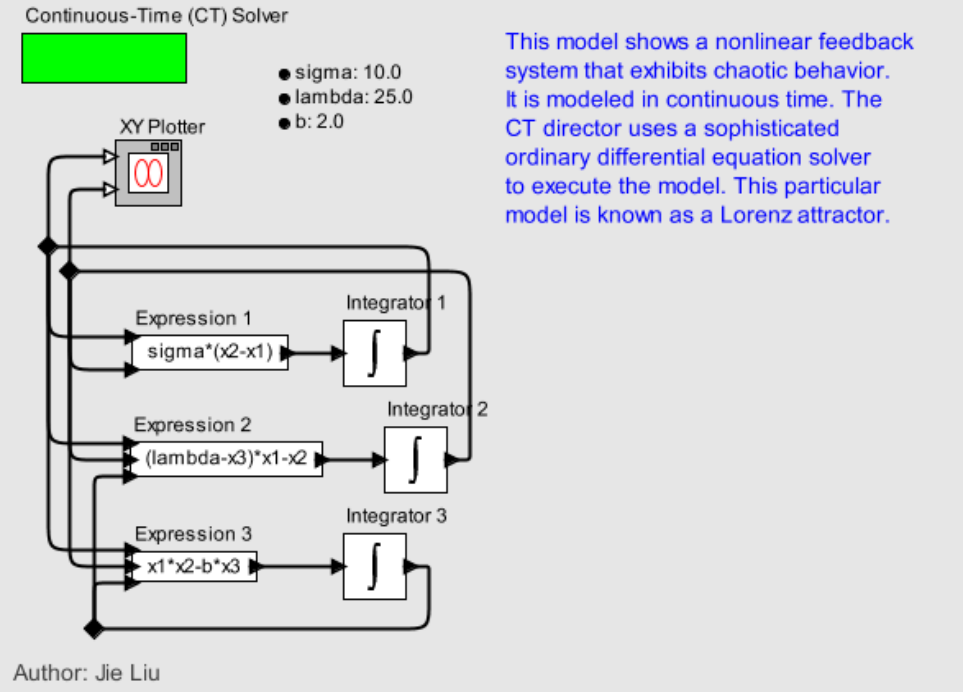
Example Ptolemy Model of Computation: Discrete Event

- Prinzip:
  - Kommunikation über Ereignisse
  - Jedes Ereignis trägt einen Wert und einen Zeitstempel
  - An jedem Aktor werden die Ereignisse in der Reihenfolge ihres Zeitstempels verarbeitet
- Variante:
  - Distributed Discrete Events
- Anwendungsgebiet:
  - Digitale Hardware
  - Telekommunikation
  - Verteilte, synchronisierte Systeme
- Werkzeuge:
  - VHDL
  - Verilog



Example Ptolemy Model of Computation: Continuous Time

- Prinzip:
  - Verwendung kontinuierlicher Signale (bestimmt gemäß Differential-gleichungen)
  - Ableitung des Codes durch Diskretisierung
- Anwendungsgebiet:
  - Simulation und Reglerauslegung
- Werkzeuge:
  - Simulink
  - Labview



Weitere Models of Computation

- Component Interaction:
  - Mischung von daten- und anfragegetriebener Ausführung
  - Beispiel: Web Server
- Discrete Time:
  - Erweiterung des synchronen Datenflussmodels um Zeit zwischen Ausführungen zur Unterstützung von Multiratensystemen
- Time-Triggered Execution
  - Die Ausführung wird zeitlich geplant
  - Anwendungsgebiet: kritische Regelungssysteme
- Process Networks
  - Prozess senden zur Kommunikation Nachrichten über Kanäle
  - Kanäle können Nachrichten speichern: asynchrone Nachrichten
  - Anwendungsgebiet: verteilte Systeme
- Rendezvous
  - synchrone Kommunikation verteilter Prozesse (Prozesse warten am Kommunikationspunkt, bis Sender und Empfänger bereit sind)
  - Beispiele: CSP, CCS, Ada

Modellierung von Echtzeitsystemen

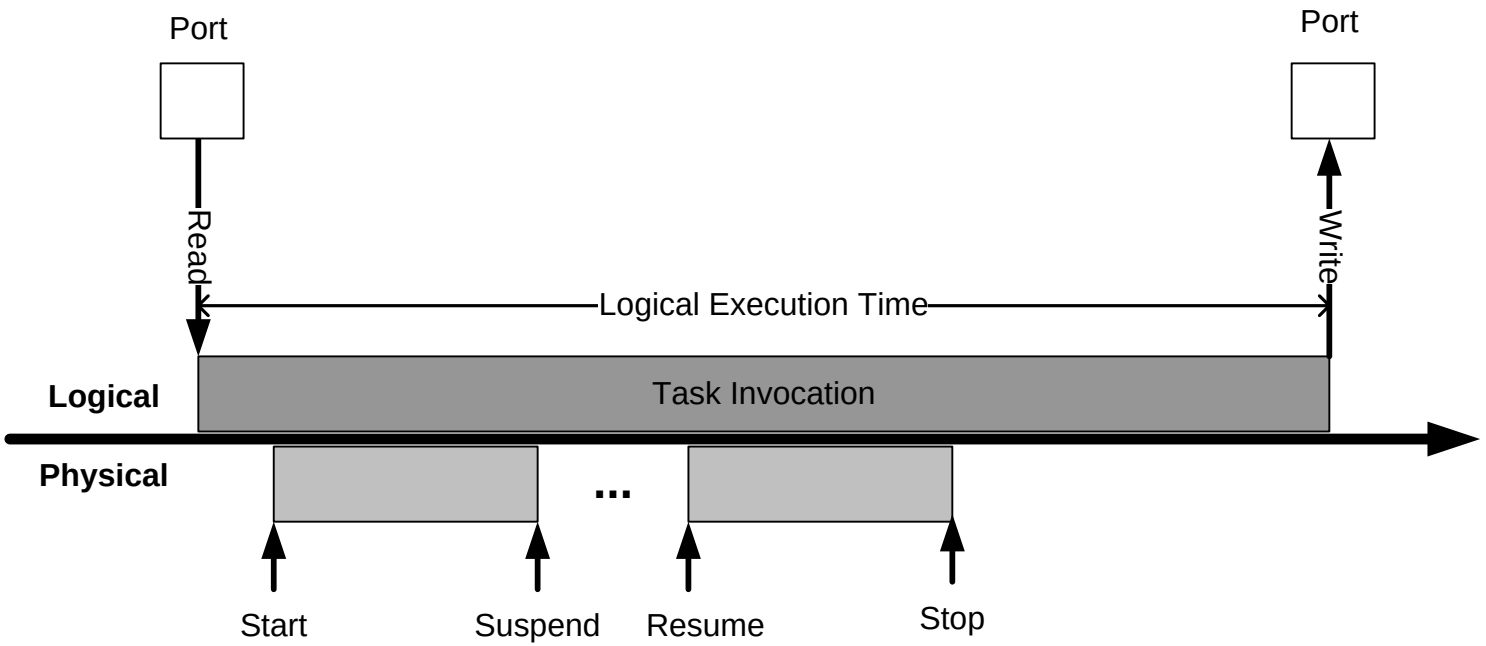
Logische Ausführungszeiten

Werkzeug: Giotto

Giotto: Hintergrund

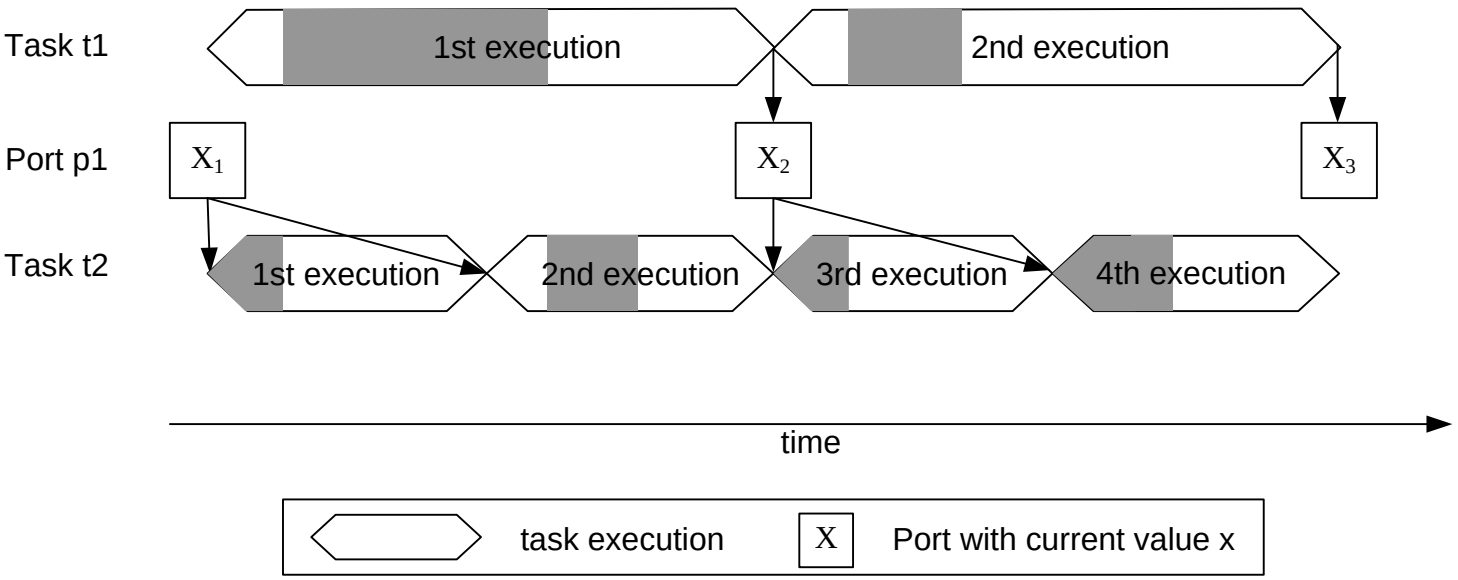
- Programmierumgebung für eingebettete Systeme (evtl. ausgeführt im verteilten System)
- Ziel:
  - strikte Trennung von plattformunabhängiger Funktionalität und plattformabhängigen Scheduling und Kommunikation
  - temporaler Determinismus ohne den Entwickler zu zwingen Implementierungsdetails zu definieren
- Hauptkonzept: Logische Ausführungszeiten
- Aktoren:
  - Tasks
    - Programmblock aus sequentiellen Code
    - keine Synchronisationspunkte, blockende Operationen erlaubt
    - Schnittstellen: Ports
- http://embedded.eecs.berkeley.edu/giotto/

Logische Ausführungszeit



Motivation siehe http://www.cs.uic.edu/~shatz/SEES/henzinger.slides.ppt

Kommunikation zwischen Tasks



Zusammenfassung

- Das Konzept der logischen Ausführungszeiten erlaubt eine Abstrahierung von der physikalischen Ausführungszeit und somit die Trennung von plattformunabhängigem Verhalten (Funktionalität und zeitl. Verhalten) und plattformabhängiger Realisierung (Scheduling, Kommunikation)
- Weitere Literaturhinweise:
  - Henzinger et al.: Giotto: A time-triggered language für embedded programming, Proceedings of the IEEE, vol.91, no.1, pp. 84-99, Jan 2003
  - Henzinger et al.: Schedule-Carrying Code, Proceedings of the Third International Conference on Embedded Software (EMSOFT), 2003

Modellierung von Echtzeitsystemen

Reaktive Systeme

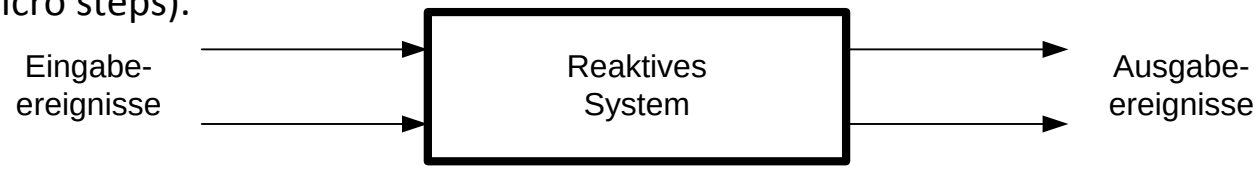
Werkzeuge: SCADE, Esterel Studio

Esterel

- Esterel ist im klassischen Sinne eher eine Programmiersprache, als eine Modellierungs-sprache
- Esterel wurde von Jean-Paul Marmorat und Jean-Paul Rigault entwickelt um die Anforderungen von Echtzeitsystemen gezielt zu unterstützen:
  - direkte Möglichkeit zum Umgang mit Zeit
  - Parallelismus direkt in der Programmiersprache
- G. Berry entwickelt die formale Semantik für Esterel
- Es existieren Codegeneratoren zur Generierung von u.a. **sequentiellen** C, C++ Code:
  - In Esterel werden (parallele) Programme in **einen** endlichen Automaten umgewandelt
  - Aus dem endlichen Automaten wird ein Programm mit **einem** Berechnungsstrang erzeugt → deterministische Ausführung trotz paralleler Modellierung.
- SCADE (ein kommerzielles Tool, dass u.a. die Esterel-Sprache verwendet) wurde bei der Entwicklung von Komponenten für den Airbus A380 eingesetzt.
- Ein frei verfügbarer Esterel-Compiler kann unter http://www-sop.inria.fr/esterel.org/files/ bezogen werden (siehe Links auf der Vorlesungs-Homepage).

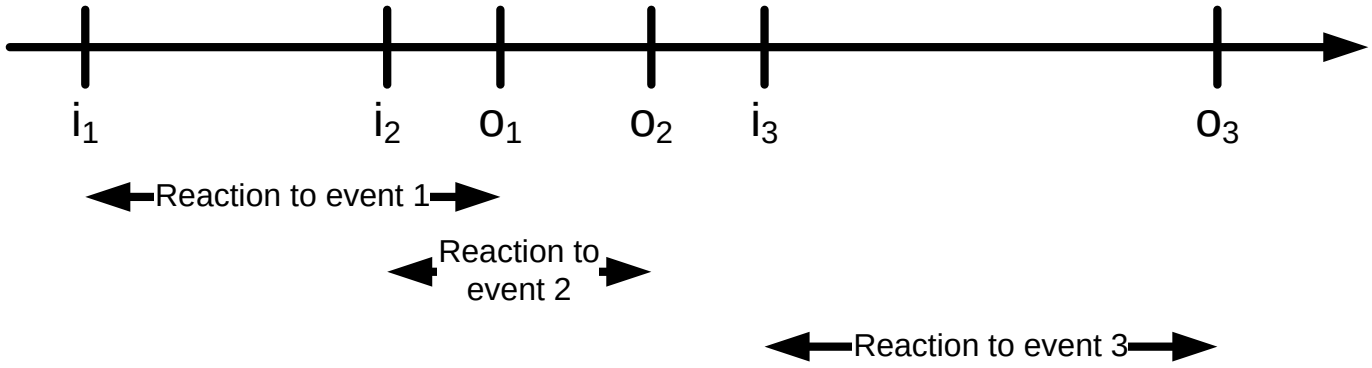
Einführung in Esterel

- Esterel gehört zu der Familie der **synchronen** Sprachen. Dies sind Programmiersprachen, die optimiert sind, um **reaktive Systeme** zu programmieren. Weitere Vertreter: Lustre, Signal, Statecharts
- Bei **reaktiven Systemen** erfolgen Reaktionen direkt auf **Eingabeereignisse**
- Synchrone Sprachen zeichnen sich vor allem dadurch aus, dass
  - Interaktionen (Reaktionen) des Systems mit der Umgebung die Basisschritte des Systems darstellen (**reaktives System**).
  - Anstelle von physikalischer Zeit die **logische Zeit** (die Anzahl der Interaktionen) verwendet wird.
  - Interaktionen, oft auch **macro steps** genannt, bestehen aus einzelnen Teilschritten (micro steps).



Reaktive Systeme - Allgemein

- In reaktiven Systemen (reactive / reflex systems) werden für Eingabeereignisse Ausgaben unter Einhaltung zeitlicher Rahmenbedingungen erzeugt.
- Reaktive Systeme finden u.a. Anwendung in der Industrie zur Prozesssteuerung und zur Steuerung / Regelung in Automobilen und Flugzeugen.
- Schwerpunkte bei der Umsetzung von reaktiven Systemen sind Sicherheit und Determinismus.
- Bearbeitung der Ereignisse kann sich überlappen (i input, o output)



Einschränkung in Esterel / synchronen Sprachen: Synchronitätshypothese

- Die Synchronitätshypothese (synchrony hypothesis) nimmt an, dass die zugrunde liegende physikalische Maschine des Systems unendlich schnell ist.

  → Die Reaktion des Systems auf ein Eingabeereignis erfolgt augenblicklich (ohne erkennbare Zeitverzögerung). Reaktionsintervalle reduzieren sich zu Reaktionsmomenten (reaction instants).

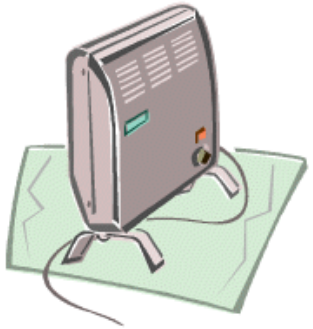
- **Rechtfertigung**: Diese Annahme ist korrekt, wenn die Wahrscheinlichkeit des Eintreffens eines zweiten Ereignisses, während der initialen Reaktion auf das vorangegangene Ereignis, sehr klein ist.

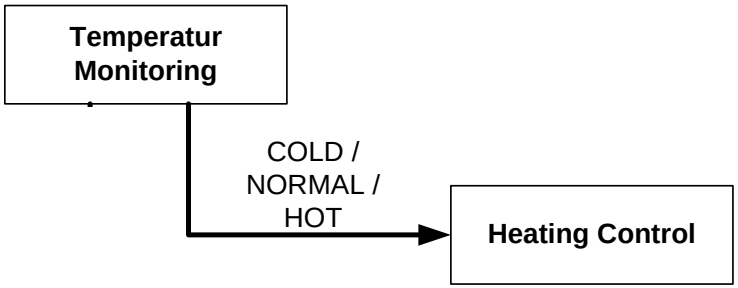
- Esterel erlaubt das gleichzeitige Auftreten von mehreren Eingabeereignissen. Die Reaktion ist in Esterel dann vollständig, wenn das System auf alle Ereignisse reagiert hat.

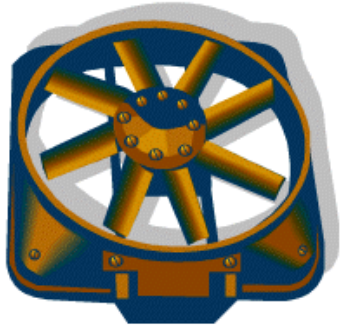
Determinismus

- Esterel ist deterministisch: auf eine Sequenz von Ereignissen (auch gleichzeitigen) muss immer dieselbe Sequenz von Ausgabe Ereignissen folgen.
- Alle Esterel-Anweisungen und -Konstrukte sind garantiert deterministisch. Die Forderung nach Determinismus wird durch den Esterel Compiler überprüft.
- Durch den Determinismus wird die Verifikation von Anwendungen wesentlich vereinfacht, allerdings birgt er auch die Gefahr, dass Ereignisse „vergessen“ werden, falls sie exakt zeitgleich mit höher priorisierten Ereignissen eintreffen.

Beispiel: Einfache Temperaturregelung







Beschreibung Beispiel

- Ziel: Regelung der Temperatur (Betriebstemperatur 5-40 Grad Celsius) mittels eines sehr einfachen Reglers.
- Ansatz:
  - Nähert sich die Temperatur einem der Grenzwerte, so wird der Lüfter bzw. die Heizung (Normalstufe) eingeschaltet.
  - Verbleibt der Wert dennoch im Grenzbereich, so wird auf die höchste Stufe geschaltet.
  - Ist der Wert wieder im Normalbereich, so wird (zur Vereinfachung) der Lüfter bzw. die Heizung wieder ausgeschaltet.
  - Wird die Betriebstemperatur über- bzw. unterschritten, so wird ein Abbruchssignal geschickt.

Esterel Code für Temperatur-Regelung (Auszug)

```
module TemperatureControler:
input TEMP: integer, SAMPLE_TIME, DELTA_T;
output HEATER_ON, HEATER_ON_STRONG,
       HEATER_OFF,VENTILATOR_ON, VENTILATOR_OFF,
       VENTILATOR_ON_STRONG, SIG_ABORT;

relation SAMPLE_TIME => TEMP;

  signal COLD, NORMAL, HOT in
    every SAMPLE_TIME do
      await immediate TEMP;
      if  ?TEMP<5 or ?TEMP>40 then emit SIG_ABORT
      elseif ?TEMP>=35 then emit HOT
      elseif ?TEMP<=10 then emit COLD
      else emit NORMAL
      end if
    end every
    ||
    loop
        await
          case COLD do
            emit HEATER_ON;
            abort
              await NORMAL;
              emit HEATER_OFF;
            when DELTA_T do
              emit HEATER_ON_STRONG;
              await NORMAL;
              emit HEATER_OFF;
            end abort
          case HOT do
              %...
        end await
      end loop
    end signal
end module
```

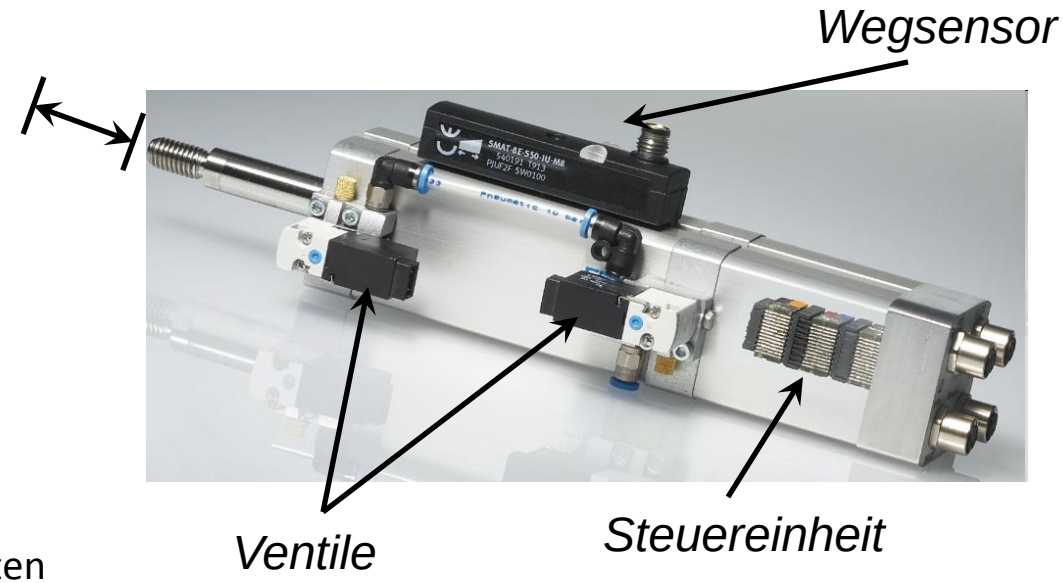
Modellierung von Echtzeitsystemen

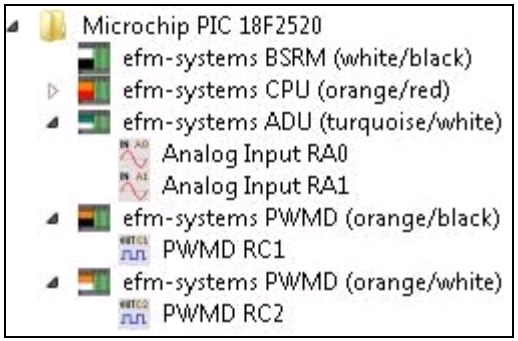
Synchroner Datenfluss

Werkzeug: EasyLab

Beispielanwendung – Pneumatischer Zylinder

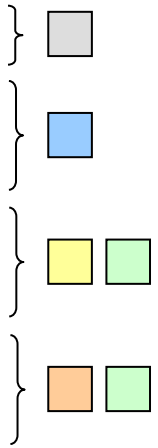
- Hardware
  - Zylinder
    - Positionssensor (Kolben)
    - Endlagenschalter
    - Zwei Magnetventile
  - Steuerungseinheit
    - Mikrocontroller
    - Analog-Digital-Wandler
    - Treiber für induktive Lasten
- Ziel: Positionssteuerung des Kolbens
- Umsetzung
  - Hardware-Modell aus Bibliothek für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

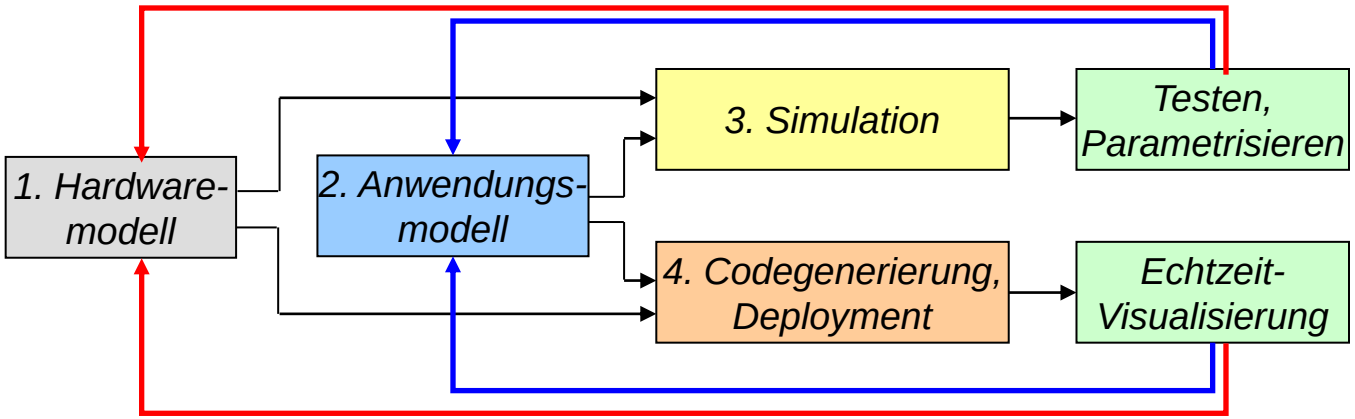




Entwicklungsprozess in EasyLab

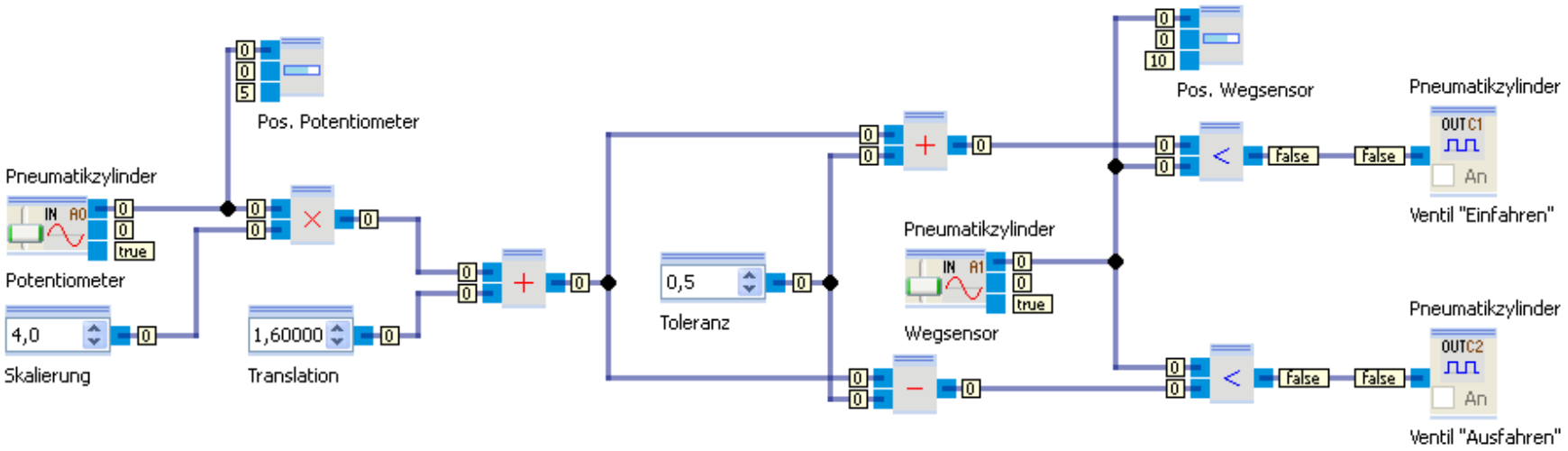
1. Spezifikation der Zielhardware
2. Modellierung der Zustandslogik sowie der abzuarbeitenden Aufgabe je Zustand
3. Simulation des Programms zum Testen und zur Erkennung von Fehlern
4. Codegenerierung und Echtzeit-Visualisierung des Zustands der Zielhardware





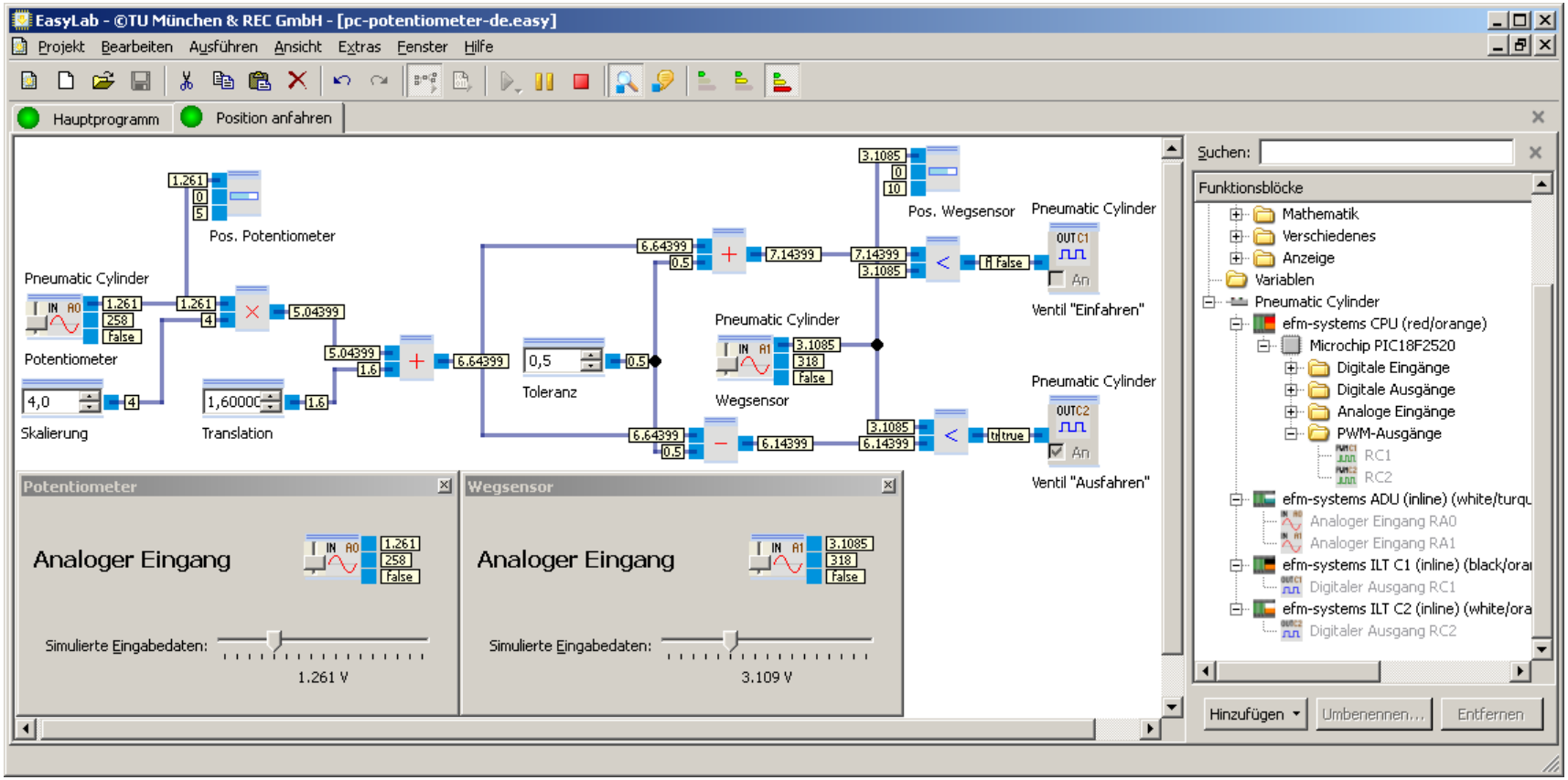
Synchroner Datenfluss

- 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

EasyLab - Demo



Beispielanwendung zur Regelung der Position eines Pneumatikzylinders

Erfolgskontrolle: Was sollten Sie aus dem Kapitel mitgenommen haben?

- Sensibilisierung für Semantik der verschiedenen modellbasierten Entwicklungswerkzeuge
- Kenntnisse der verschiedenen Models of Computation und ihrer Anwendungsfelder
- Beispiel für Klausurfragen (Klausur 2010/2011):

1. Reaktive, synchrone Sprachen wie Esterel basieren auf der Synchronitätshypothese. Erläutern Sie diese Hypothese, erklären Sie den wesentlichen Vorteil dieser Hypothese und beschreiben Sie, wie eine Implementierung aussieht. (6 Punkte = 6 Minuten)

   Antwort: siehe Folien, Implementierung durch Blockierung (z.B. Abschalten von Interrupts oder Zwischenspeichern)

2. Gegeben seien folgende drei Ausführungsmodelle
   - (i) Synchronous Dataflow (Synchroner Datenfluss)
   - (ii) Synchronous Reactive
   - (iii) Time-Triggered Execution (zeitgesteuerte Ausführung)

   und folgende drei Anwendungen:
   - 1. Fahrzeugsteuerung: Mehrere periodische Prozesse werden im verteilten System ausgeführt.
   - 2. Transaktionssystem: Eine Ampelsteuerung mit einem zentralen Steuerrechner soll entwickelt werden. Die Steuerung soll auf verschiedenste Ereignisse reagieren können und abhängig von diesen Ereignissen verschiedene Aktionen ausführen.
   - 3. Regelungsanwendung: Eine Spannungsquelle soll periodisch geregelt werden.

   Finden Sie eine passende Zuordnung von den Ausführungsmodellen (jeweils nur einmal verwenden) zu den Anwendungen und begründen Sie kurz Ihre Antwort. (6 Punkte = 6 Minuten)

   Antwort:
   - (i) Synchronous Data Flow <-> Regelungsanwendung: Synchronous Data Flow eignet sich vor allem für periodischen Berechnungen, die auf einem Rechner ausgeführt werden (da die Synchronitätshypothese so leichter umgesetzt werden kann).
   - (ii) Synchronous Reactive <-> Transaktionssystem: Synchronous Reactive eignet sich vor allem für die Modellierung und Entwicklung von ereignisbasierten Systemen, wie der Ampelsteuerung.
   - (iii) Time-Triggered Execution <-> Fahrzeugsteuerung: Zeitgesteuerte Ausführung ist ideal um periodische Prozesse im verteilten System auszuführen.

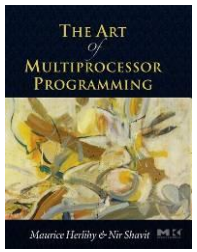
Kapitel 4

Nebenläufigkeit

Inhalt

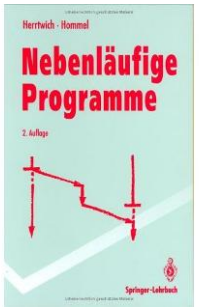
- Motivation
- Unterbrechungen (Interrupts)
- (Software-) Prozesse
- Threads
- Interprozesskommunikation (IPC)

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/

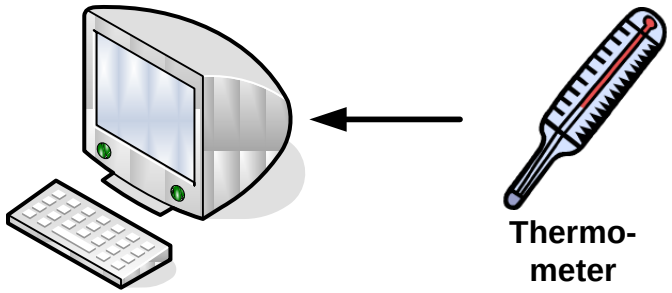
Definition von Nebenläufigkeit

- **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)

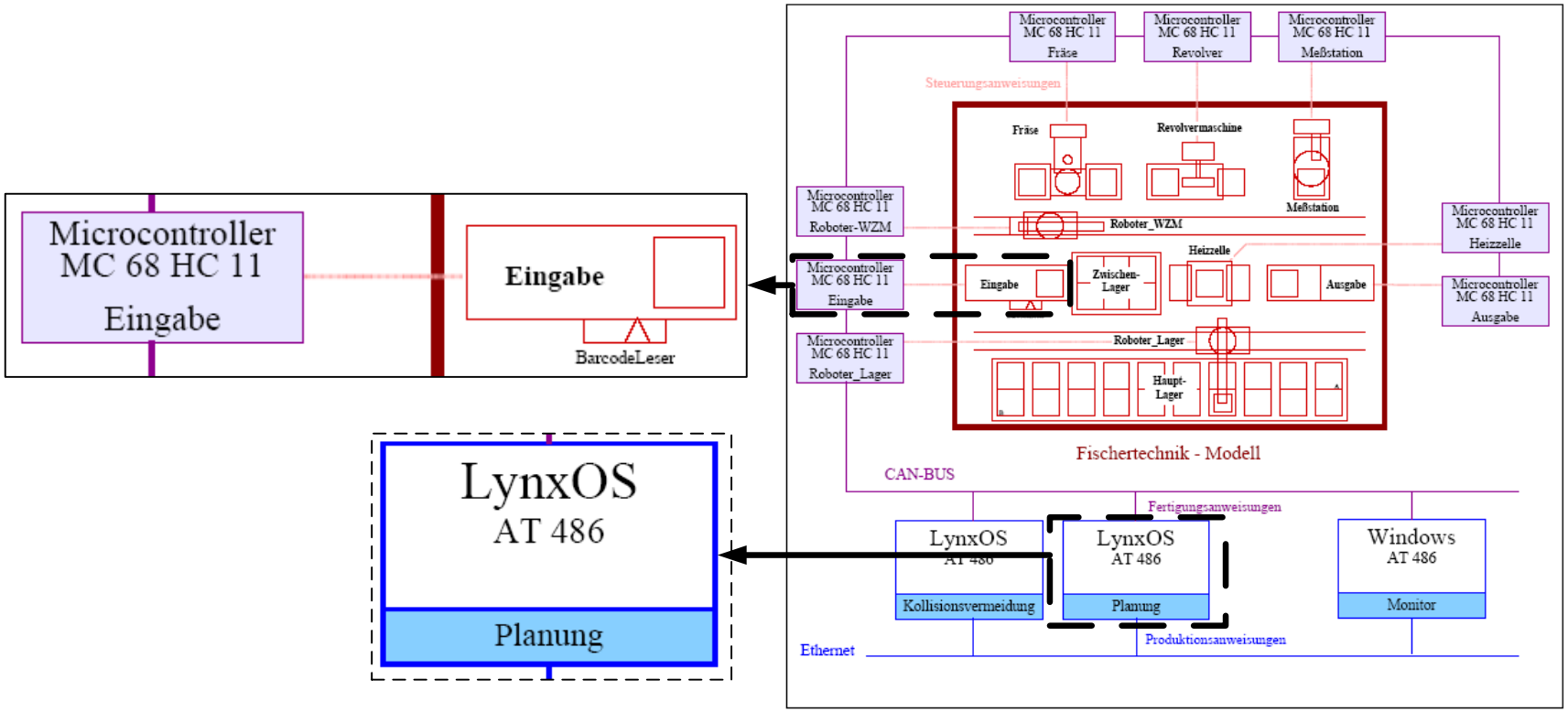
Anwendungsfälle für Nebenläufigkeit (Unterbrechungen)



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

Allgemeines Anwendungsgebiet: hauptsächlich zur Anbindung von externer Hardware

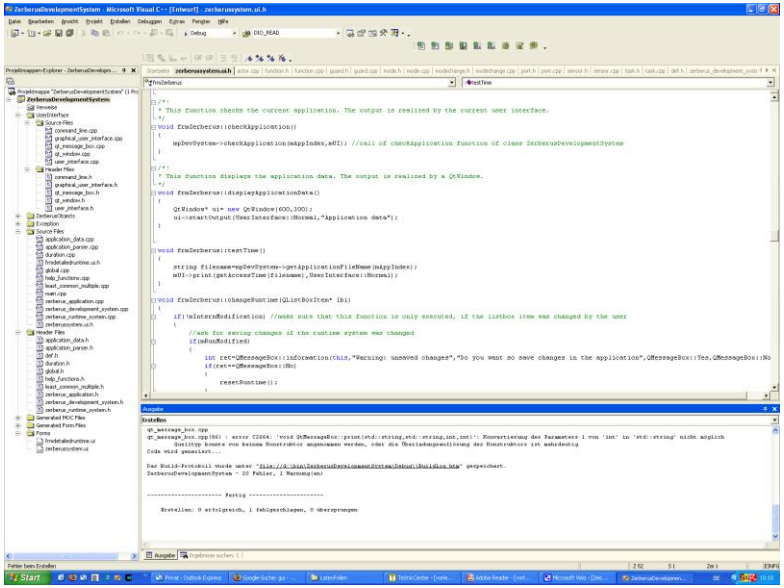
Anwendungsfälle für Nebenläufigkeit (Prozesse)



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)**

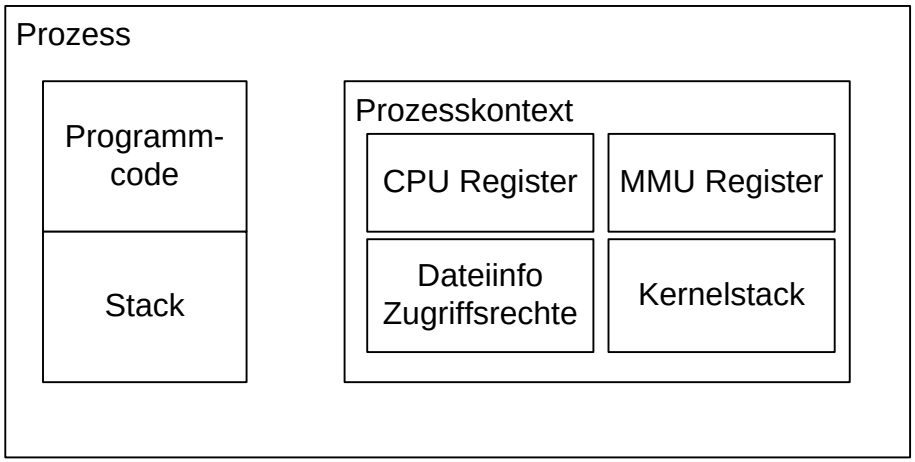
Allgemeines Anwendungsgebiet: unterschiedliche Berechnungen im gleichen Anwendungskontext

Nebenläufigkeit

Prozesse

Definition

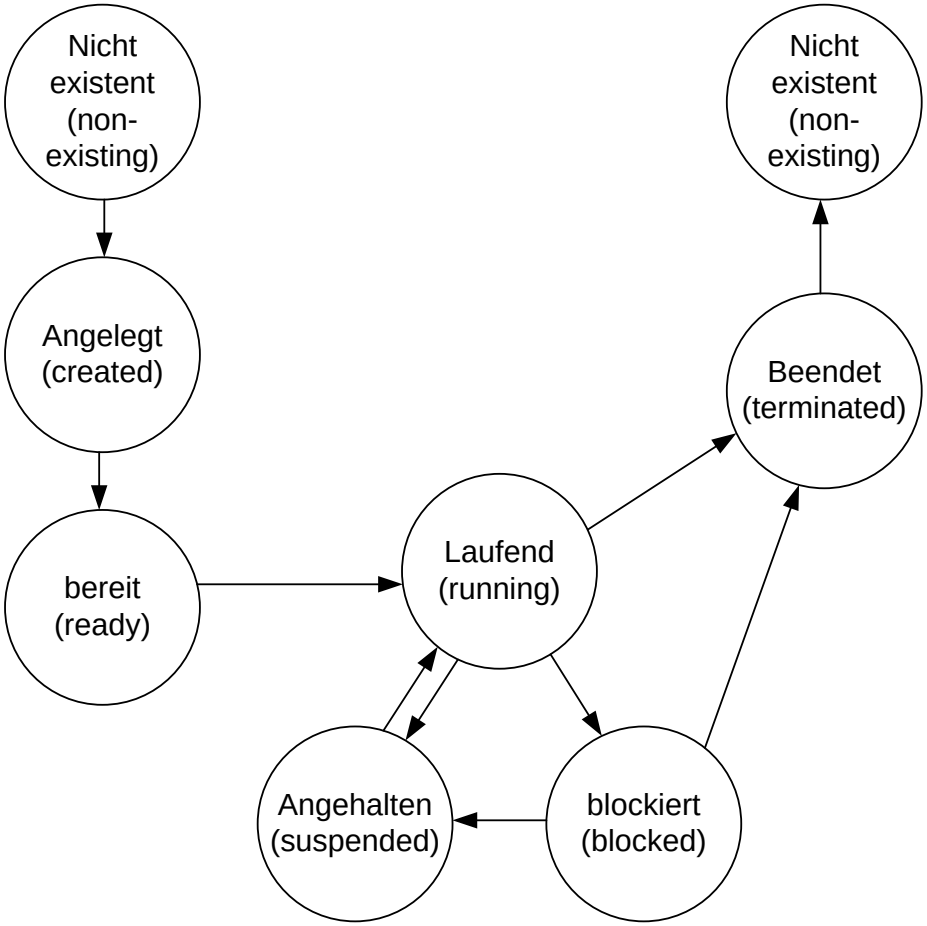
- **Prozess**: Abstraktion eines sich in Ausführung befindlichen Programms
- Die gesamte Zustandsinformation der Betriebsmittel für ein Programm wird als eine Einheit angesehen und als Prozess bezeichnet.
- Prozesse können weitere Prozesse erzeugen ⇨ Vater-,Kinderprozesse.



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

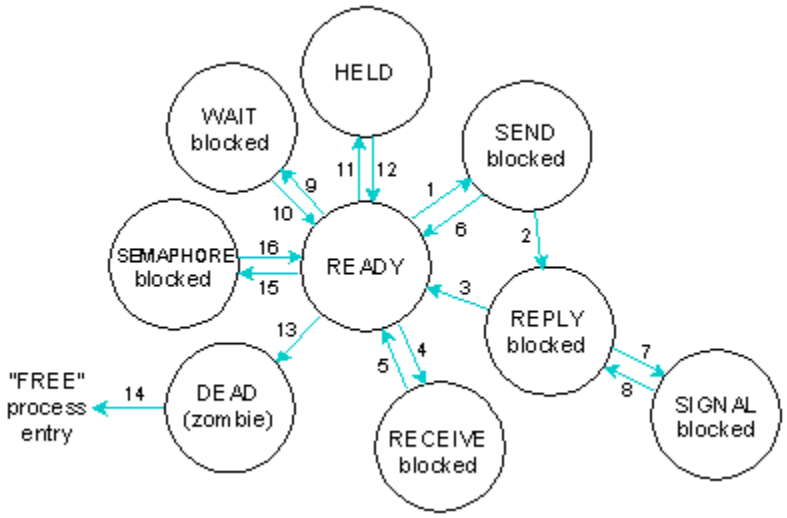
Prozesszustände (allgemein)



Prozesse in QNX[1]

The transactions depicted in the previous diagram are as follows:

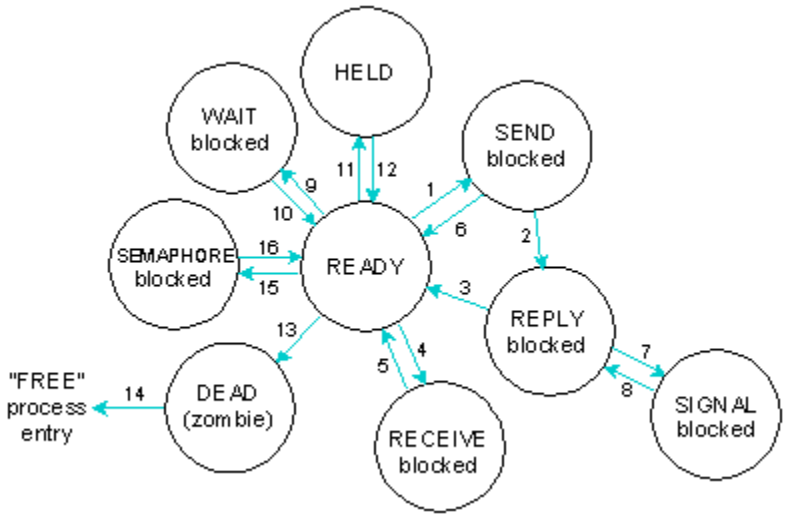
1. Process sends message.
2. Target process receives message.
3. Target process replies.
4. Process waits for message.
5. Process receives message.
6. Signal unblocks process.
7. Signal attempts to unblock process; target has requested message signal catching.
8. Target process receives signal message.



[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.