Simulationstechnik 2

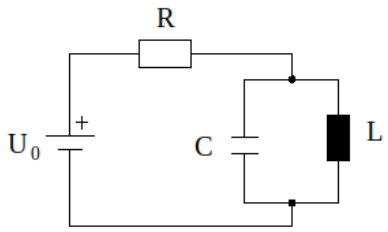
Peter Junglas 11.06.2023

Einführung in Modelica

Signalfluss-Modellierung eines RLC-Kreises

- RLC-Kreis:

  gegeben durch Schaltkreis

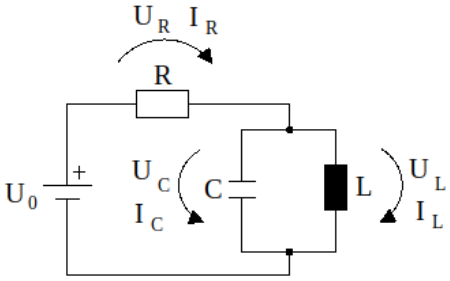
  Werte
  - U_0 = 9 V
  - R = 100 Ω
  - C = 1 mF
  - L = 10 mH

  Ziel
  - Simulation zur Bestimmung der Ströme und Spannungen

  Idee
  - DGL herleiten und modellieren

- Physikalische Beschreibung des Beispiels:

  Definition der Ströme und Spannungen

  1. Kirchhoffsches Gesetz (**Knotenregel**)

  $$I_R = I_C + I_L \qquad (1)$$

  2. Kirchhoffsches Gesetz (**Maschenregel**)

  $$\begin{aligned} U_C &= U_L && (2)\\ U_0 + U_R + U_C &= 0 && (3) \end{aligned}$$

  Bauteilgleichungen

  $$\begin{aligned} U_R &= R\,I_R && (4)\\ \dot{U}_C &= \frac{1}{C}\,I_C && (5)\\ U_L &= L\,\dot{I}_L && (6) \end{aligned}$$

  zusammen 6 Gleichungen für 6 Größen

- Mathematische Vereinfachung der Gleichungen:

  insgesamt ein DAE-System (**Differential Algebraic Equation**)

  in normale DGL umformen

  Vorgehen

- abgeleitete Größen bleiben übrig (**Zustandsgrößen**)
- alle anderen Größen werden damit ausgedrückt

hier: Zustandsgrößen U_C, I_L, daher

$$(2) \Rightarrow \quad U_L = U_C$$

$$(3) \Rightarrow \quad U_R = -U_0 - U_C$$

$$(4) \Rightarrow \quad I_R = \frac{1}{R} U_R = -\frac{U_0 + U_C}{R}$$

$$(1) \Rightarrow \quad I_C = I_R - I_L = -\frac{U_0 + U_C}{R} - I_L$$

damit schließlich

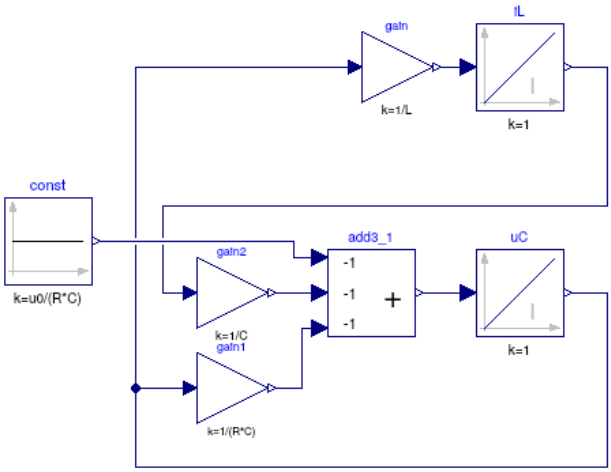
$$(5) \Rightarrow \quad \dot{U}_C = -\frac{1}{RC} U_C - \frac{1}{C} I_L - \frac{U_0}{RC}$$

$$(6) \Rightarrow \quad \dot{I}_L = \frac{1}{L} U_C$$

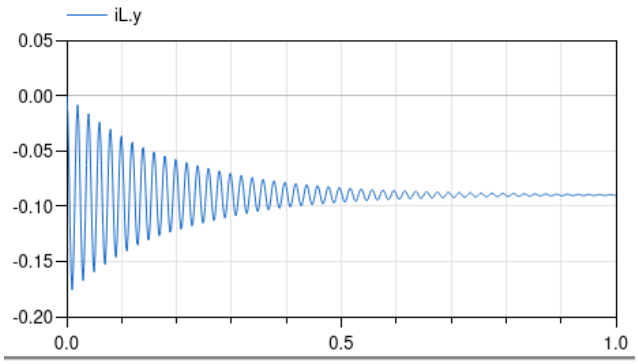
- Simulation der DGL:

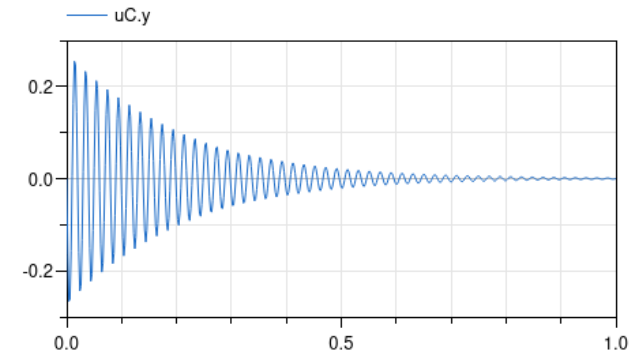
  Standardverfahren: Signalflussmethode
  - Basiskomponenten: Integrator, algebraische Operationen
  - hier in Modelica
  - weit verbreitet: Matlab/Simulink

  Gesamtmodell `RLCDgl`



  Simulationsergebnis


iL.y
0.05
0.00
-0.05
-0.10
-0.15
-0.20
0.0
0.5
1.0


uC.y
0.2
0.0
-0.2
0.0
0.5
1.0

Modellierung eines RLC-Kreises mit Modelica

- Physical Modeling:

  Grundidee
  - Bausteine entsprechen physikalischen Komponenten (Massen, Widerstände, Ventile)
  - Leitungen entsprechen physikalischen Verbindungen (Flansche, Kabel, Rohre)
  - Bewegungsgleichungen werden automatisch aufgestellt, vereinfacht und numerisch gelöst

  umfangreiche Standard-Bibliothek MSL für diverse physikalische Bereiche, u.a.
  - elektrisch (analog, digital, machines)
  - mechanisch (translatorisch, rotatorisch, 3d)
  - thermisch, fluid
  - magnetisch

  Vielzahl an freien oder kommerziellen Bibliotheken, z. B.
  - Hydraulik, Pneumatik
  - Fahrzeugdynamik
  - chemische Prozesse
  - elektrische Energiespeicher (Akkus, Batteriemananagement)
  - Windkraftanlagen

  Grundlage: Modelica-Sprache und Basis-Bibliothek MSL

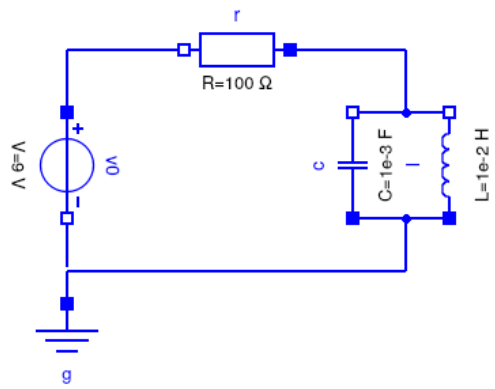
  Simulationsprogramme u.a.
  - Dymola
  - MapleSim
  - Wolfram SystemModeler
  - Simulation X
  - OpenModelica (frei)

  i. F. verwendet: Dymola (an der PHWT) bzw. OpenModelica (PHWT und zuhause)

- RLC-Kreis in Modelica:

  Komponenten der Bibliothek `Modelica.Electrical.Analog`
  - `Resistor`
  - `Capacitor`
  - `Inductor`
  - `ConstantVoltage`
  - `Ground`

  komplettes Modell `RLCDemo`



  - Ground definiert Null-Potential

  Simulationsergebnis identisch zu oben

Synchronmotor als Beispiel einer elektrischen Maschine

Grundlegendes Modell

- Einfacher Elektromotor in Modelica

  idealisierter Gleichstrom-Motor bzw. Generator
  `Modelica.Electrical.Analog.Basic.RotationalEMF`

  Parameter k beschreibt elektrisch/mechanische Kopplung ("Stärke des Magnetfelds")

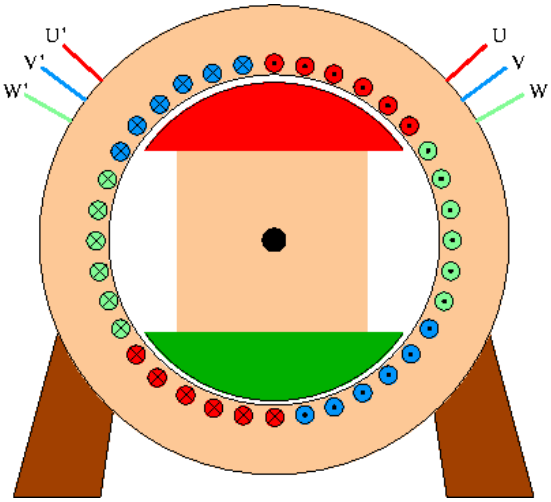
  entscheidende Gleichungen

  $$k\omega = U$$
  $$\tau = k\,I$$

  i. F. wesentlich detailliertere Beschreibung von Drehstrommotoren
  - verwendet Drehstrom explizit
  - berücksichtigt elektrische Eigenschaften der Motorspulen
  - beschreibt induktive Kopplungen
  - konkret am Beispiel der PMSM

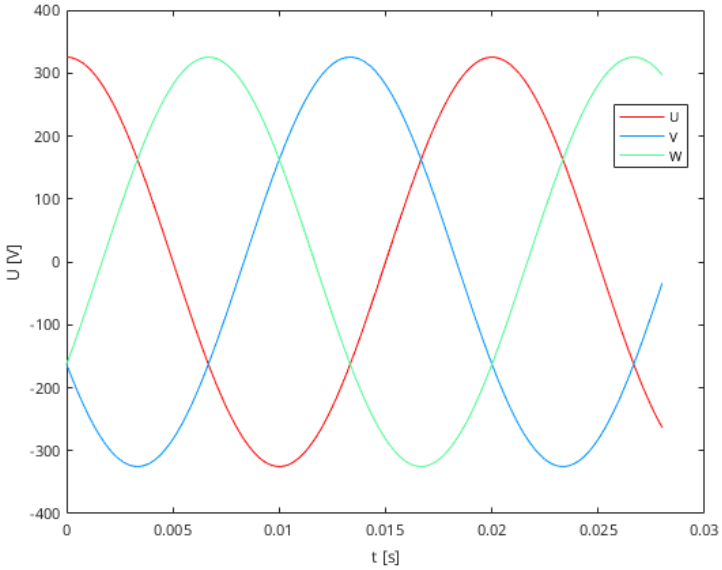
- Permanentmagnet-erregte Synchronmaschine (PMSM) [5]:

  rotierende elektrische Drehstrommaschine

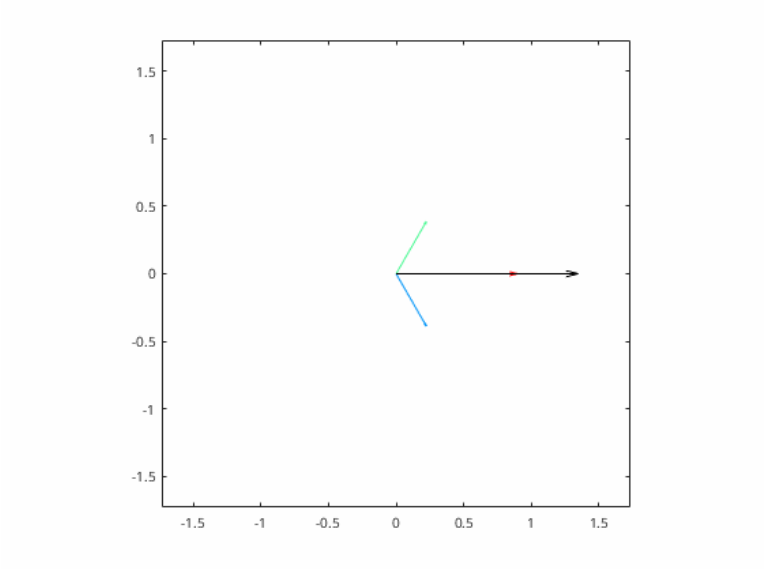


  i. F. Grundprinzip als Motor

  Stator enthält um 120° versetzte Spulen
  - werden mit Dreiphasenwechselstrom ("Drehstrom") mit Frequenz f versorgt



  - → rotierendes magnetisches Dipolfeld



- häufig Anschlüsse U', V', W' zu Sternpunkt N verbunden (**Sternschaltung**)

Rotor enthält Permanentmagnet mit festem Dipolfeld

- Wechselwirkung der Dipolfelder von Rotor und Stator
- → Rotor dreht mit Frequenz $f_R = f$

alternativ als Generator

- Rotor wird mechanisch gedreht
- → in den Statorspulen wird Wechselspannung induziert

Lastmoment am Rotor

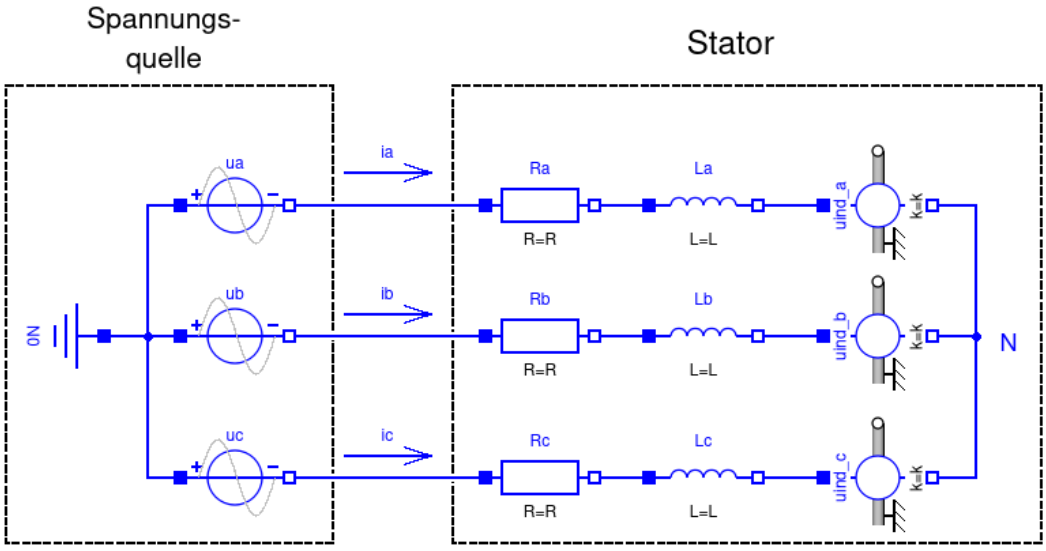
- → Rotorbewegung eilt um Polradwinkel ϑ nach
- → in den Statorspulen wird Polradwinkelspannung induziert

Erweiterung durch mehrere komplette Spulensätze (Polpaarzahl p)

- Frequenz des Rotors $f_R = f/p$

- Ersatzschaltbild:

Stator incl. Spannungsquelle



Komponenten

- ohmsche Widerstände der Spulendrähte
- Induktivitäten für Selbstinduktion der Statorspulen
- Induktionsspannungen durch Rotor

vernachlässigt u.a. magnetische Verluste, Hysterese- und Sättigungseffekte

Vereinfachung i. F.: alle R gleich groß, alle L gleich groß

- Gleichungen (nach [6]):

  Eingangsspannungen

$$\begin{aligned} u_a &= u_0 \cos(\omega_0 t) \\ u_b &= u_0 \cos\left(\omega_0 t - \frac{2\pi}{3}\right) \\ u_c &= u_0 \cos\left(\omega_0 t - \frac{4\pi}{3}\right) \end{aligned}$$

  - mit f_0 = 50 Hz, u_0 = 230 $\cdot\sqrt{2}$ V

  Spannungen pro Strang

$$\begin{aligned} u_a &= L\frac{di_a}{dt} + Ri_a + u_{ind,a} \\ u_b &= L\frac{di_b}{dt} + Ri_b + u_{ind,b} \\ u_c &= L\frac{di_c}{dt} + Ri_c + u_{ind,c} \end{aligned} \qquad (1)$$

  Induktionsspannungen vom Rotor mit einfachem Modell

$$\begin{aligned} u_{ind,a} &= k_m \omega \cos(\varphi) \\ u_{ind,b} &= k_m \omega \cos\left(\varphi - \frac{2\pi}{3}\right) \\ u_{ind,c} &= k_m \omega \cos\left(\varphi - \frac{4\pi}{3}\right) \end{aligned} \qquad (2)$$

  - φ = Winkel des Rotors, ω = dφ/dt
  - berücksichtigt gedrehte Geometrie

  zugehörige Momente im Rotor addiert zum Motormoment τ_M

$$\tau_M = k_m \left(i_a \cos(\varphi) + i_b \cos\left(\varphi - \frac{2\pi}{3}\right) + i_c \cos\left(\varphi - \frac{4\pi}{3}\right) \right) \qquad (3)$$

  Motormoment treibt Motorträgheit und (gegebene) äußere Last

$$\tau_M = J\dot{\omega} + \tau_L \qquad (4)$$

  Vereinfachung

  - Knoten N im Stator liefert

$$i_a + i_b + i_c = 0 \qquad (5)$$

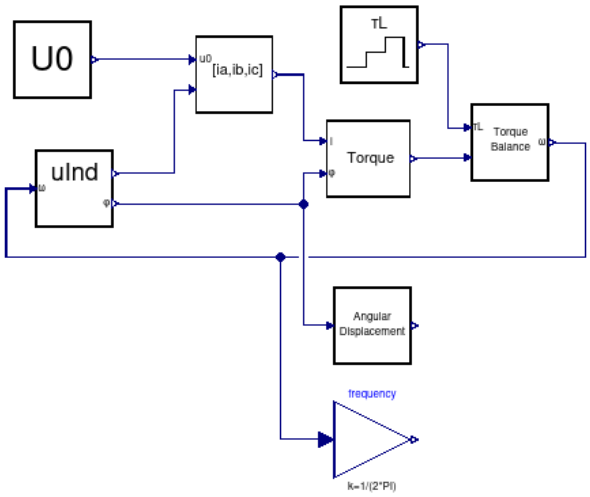
  - auch $u_{ind,i}$ sowie u_i addieren sich zu 0
  - → man braucht nur zwei der drei Gleichungen von (1) und (2)

  Bilanz

  - 7 Variablen: i_a, i_b, i_c, $u_{ind,a}$, $u_{ind,b}$, φ, τ_M
  - 7 Gleichungen: (1a,b), (2a,b), (3), (4), (5)

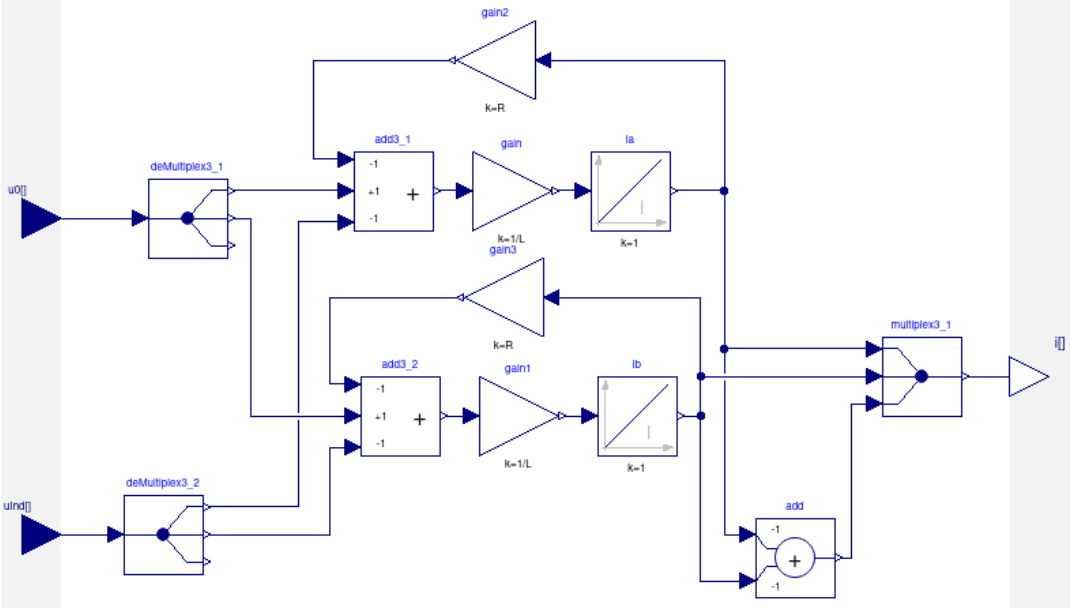
- Modell `testPMSM1`:

  Gesamtmodell in "Simulink-Manier"

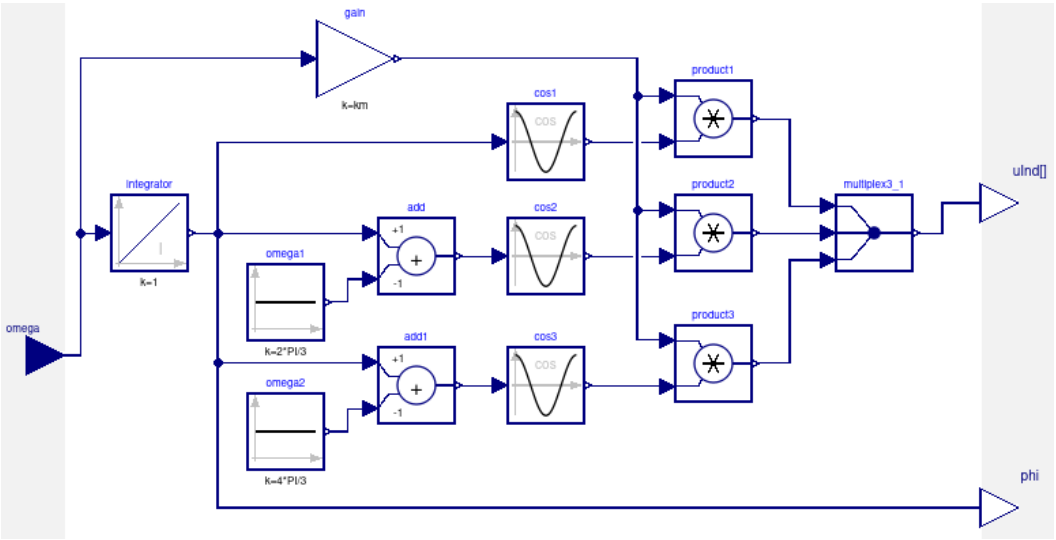


Submodelle der Gleichungen

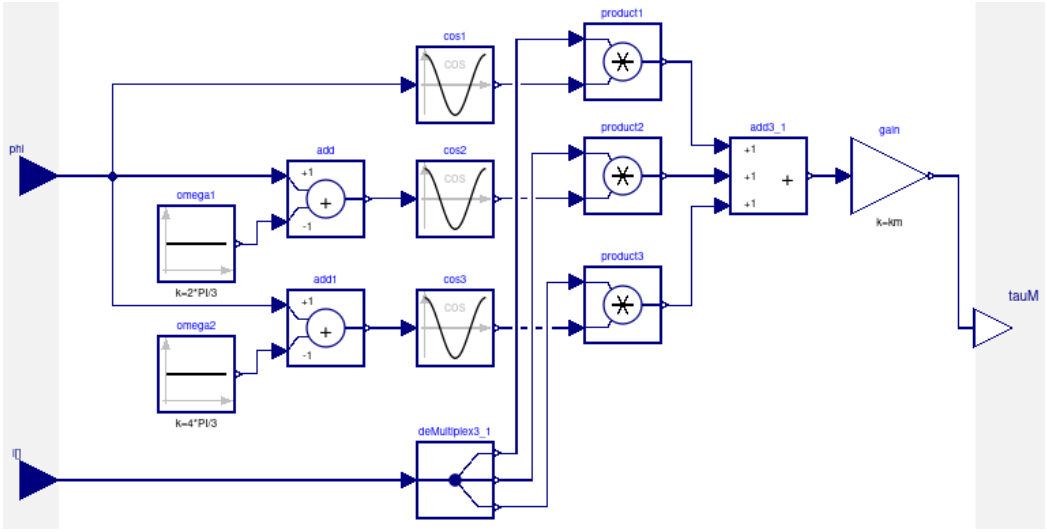
- `ComputeCurrents` (Gl. 1 und 5)



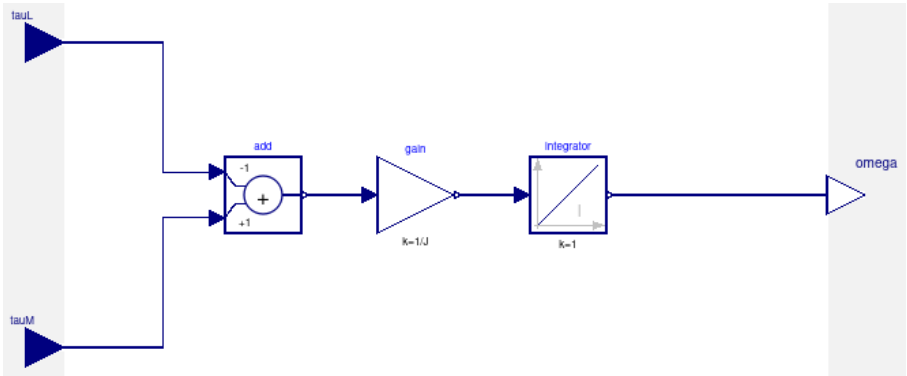
- `Induction` (Gl. 2)



- `EngineTorque` (Gl. 3)

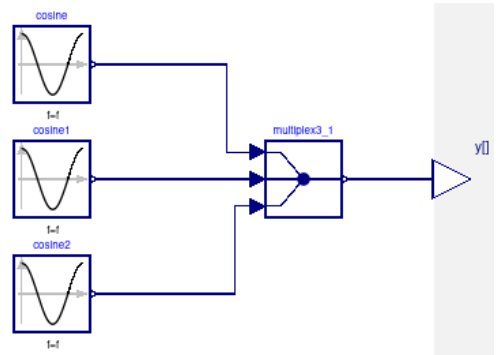


- `TorqueBalance` (Gl. 4)

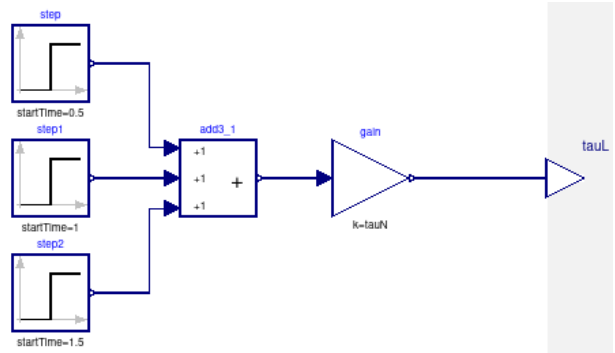


Eingabe-Blöcke

- `ThreePhaseVoltage`

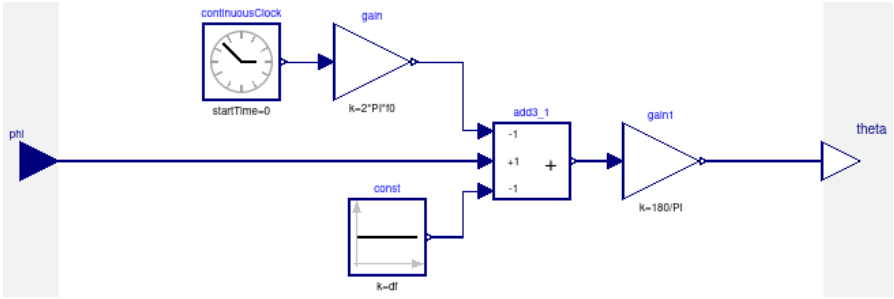


- `ExternalLoad`



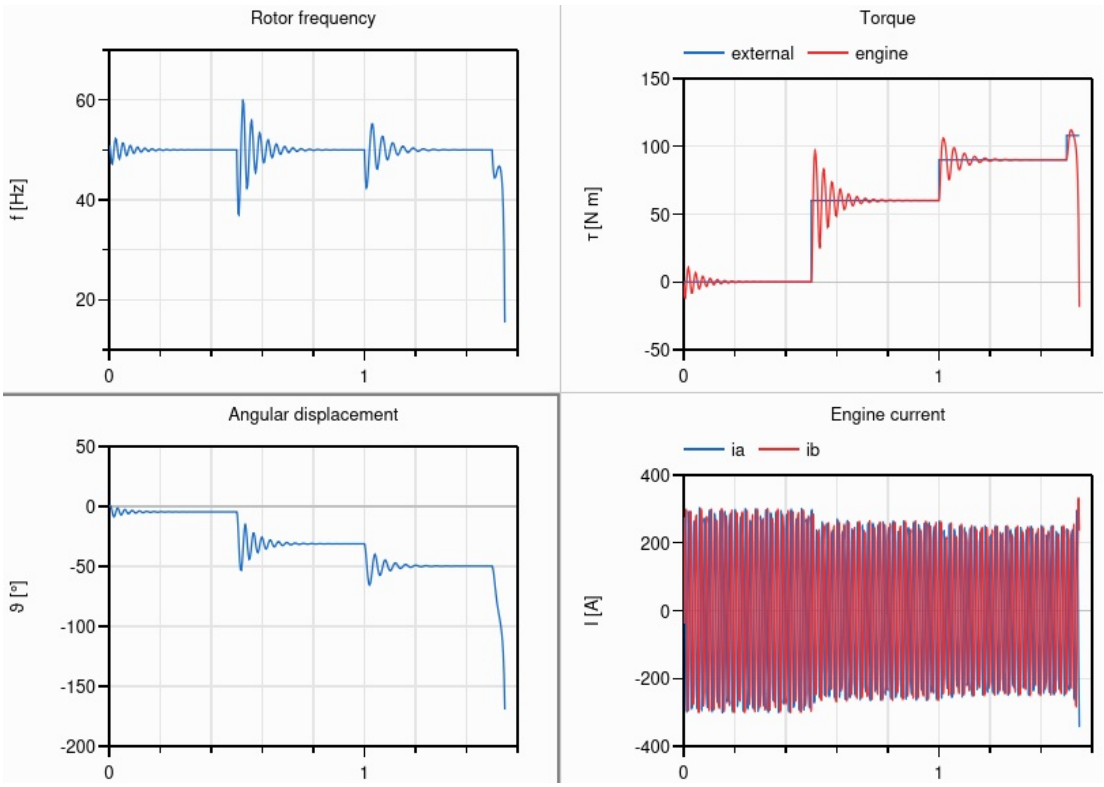
Ausgaben

- Frequenz f = ω/(2π)
- Polradwinkel ϑ = Winkel zwischen Rotor und Drehfeld
- berechnet in `AngularDisplacement`



- Simulationsergebnisse:

  zunächst Leerlauf, dann drei Lastsprünge



  Frequenz = f_0, bis auf Einschwingvorgänge

  Motormoment passt sich äußerem Moment an

  Wechselströme im Motor (natürlich), Bedeutung schwer zu erkennen

  Polradwinkel ϑ

  - immer negativ (Drehfeld zieht Rotor hinterher)
  - wird betragsmäßig größer bei höherer Last
  - größer als 90° → Motor gerät aus dem Takt

Vereinfachung durch Clarke- und Park-Transformation

- Clarke-Transformation:

  Ausgangspunkt
  - drei Unbekannte (z.B. i_a, i_b, i_c) mit Summe 0
  - kann auf zwei Größen reduziert werden
  - was ist geschickte Wahl?

  Antwort

$$\begin{aligned} i_\alpha &:= \frac{2}{3}\left(i_a \cos 0 + i_b \cos\frac{2\pi}{3} + i_c \cos\frac{4\pi}{3}\right) \\ &= \frac{2}{3}\left(i_a - \frac{1}{2}i_b - \frac{1}{2}i_c\right) \\ i_\beta &:= \frac{2}{3}\left(i_a \sin 0 + i_b \sin\frac{2\pi}{3} + i_c \sin\frac{4\pi}{3}\right) \\ &= \frac{2}{3}\left(\frac{\sqrt{3}}{2}i_b - \frac{\sqrt{3}}{2}i_c\right) \end{aligned}$$

  übersichtlicher in Vektor-Schreibweise

$$\vec{i} := \begin{pmatrix} i_a \\ i_b \\ i_c \end{pmatrix}, \quad \tilde{i} := \begin{pmatrix} i_\alpha \\ i_\beta \end{pmatrix}, \quad M := \frac{2}{3}\begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{pmatrix}$$

$$\Rightarrow \qquad \tilde{i} = M\vec{i}$$

  - (i_α, i_β)' heißt **Raumzeiger** (englisch: **space phasor**)

  Umkehrung mit

$$\widetilde{M} := \begin{pmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{pmatrix} \quad \Rightarrow \qquad \vec{i} = \widetilde{M}\tilde{i}$$

  Erweiterung auf beliebigen Dreiervektor durch Hinzunahme der Summe

  $i_\gamma = (i_a + i_b + i_c)/3$

  benannt nach Edith Clarke (1883 - 1959)
  - erste Frau in den USA Ingenieurin für Elektrotechnik
  - erstes weibliches Mitglied in der American Institute of Electrical Engineers
  - Professorin für Elektrotechnik an der University of Texas at Austin
  - Wikipedia

    "Im Gegensatz zu vielen ihrer Kollegen war sie versiert in höherer Mathematik,.."

- Umrechnen der PMSM-Gleichungen auf Clarke-Koordinaten:

  jeweils umrechnen

$$\tilde{u} := \begin{pmatrix} u_\alpha \\ u_\beta \end{pmatrix} = M \begin{pmatrix} u_a \\ u_b \\ u_c \end{pmatrix} = M\,\vec{u}$$

  - analog für i und u_{ind}

  Eingangsspannungen

$$\tilde{u} := u_0 \begin{pmatrix} \cos\omega_0 t \\ \sin\omega_0 t \end{pmatrix}$$

Strang-Gleichungen

$$\tilde{u} = L\frac{d\tilde{i}}{dt} + R\tilde{i} + \tilde{u}_{ind} \qquad (\tilde{1})$$

Induktionsspannungen

$$\tilde{u}_{ind} = k_m\omega \begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix} \qquad (\tilde{2})$$

Motormoment

$$\tau_M = \frac{3}{2}k_m(i_\alpha \cos\varphi + i_\beta \sin\varphi) \qquad (\tilde{3})$$

Momentenbilanz bleibt

$$\tau_M = J\dot{\omega} + \tau_L \qquad (\tilde{4})$$

Bilanz

- 6 Variablen: i_α, i_β, $u_{ind,\alpha}$, $u_{ind,\beta}$, φ, τ_M
- 6 Gleichungen: (1a,b), (2a,b), (3), (4)

- Park-Transformation:

  Clarke-Trafo mit anschließender Umrechnung von Raumzeigern ins Rotor-System

  $$\begin{aligned} i_d &:= i_\alpha \cos\varphi + i_\beta \sin\varphi \\ i_q &:= -i_\alpha \sin\varphi + i_\beta \cos\varphi \end{aligned}$$

  in Vektordarstellung

  $$\bar{\imath} := \begin{pmatrix} i_d \\ i_q \end{pmatrix} = Q\tilde{\imath} \quad \text{mit} \quad Q = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix}$$

  Vorteil

  - macht aus Größen mit Drehstrom-Symmetrie konstante Größen
  - zur Regelung geeignet (Vektorregelung)

- Umrechnen der Gleichungen auf d/q-Koordinaten:

  Induktionsspannungen

  $$\bar{u}_{ind} = k_m\omega \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \omega\bar{k} \qquad (\bar{2})$$

  - Wert direkt in (1) eingesetzt → Gl. (2) wird i. F. nicht gebraucht

  Eingangsspannungen

  $$\bar{u} := u_0 \begin{pmatrix} \cos(\omega_0 t - \varphi) \\ \sin(\omega_0 t - \varphi) \end{pmatrix}$$

  - konstant, wenn Rotor mit Eingangsspannung mitdreht

  Strang-Gleichungen

  $$\begin{aligned} u_d &= L\frac{di_d}{dt} + Ri_d - \omega L i_q + k_m\omega \\ u_q &= L\frac{di_q}{dt} + Ri_q + \omega L i_d \end{aligned}$$

  - oder in Vektorform

  $$\bar{u} = L\frac{d\bar{\imath}}{dt} + R\bar{\imath} - \omega L\,\varepsilon\,\bar{\imath} + \omega\bar{k} \qquad (\bar{1})$$

  - mit

$$\varepsilon := \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \quad \bar{k} := \begin{pmatrix} k_m \\ 0 \end{pmatrix}$$

Motormoment

$$\tau_M = \frac{3}{2} k_m i_d \qquad (\bar{3})$$

Momentenbilanz bleibt

$$\tau_M = J\dot{\omega} + \tau_L \qquad (\bar{4})$$

Bilanz

- 4 Variablen: i_d, i_q, φ, τ_M
- 4 Gleichungen: (1a,b), (3), (4)

- Zusammenhang von k_m und "Stärke" des Permanentmagneten:

  Flussverkettung ψ = magnet. Fluss incl. Windungszahl bei Spulen
  - ψ_{pm} zur Beschreibung des Permanentmagneten im Rotor
  - ψ_{pm} in d/q-Koordinaten konstant

  grundlegende Beziehung im d/q-System

  $$\overline{\psi} = L\,\bar{\imath} + \overline{\psi}_{pm}$$

  Bedeutung
  - Ableiten liefert

    $$\frac{d\overline{\psi}}{dt} = L\frac{d\bar{\imath}}{dt} = \bar{u}_{ind}$$

  - klar: Änderung des magnet. Flusses = induzierte Spannung

  Einführen von ψ und spezielle Wahl

  $$\overline{\psi}_{pm} = \begin{pmatrix} 0 \\ -k_m \end{pmatrix}$$

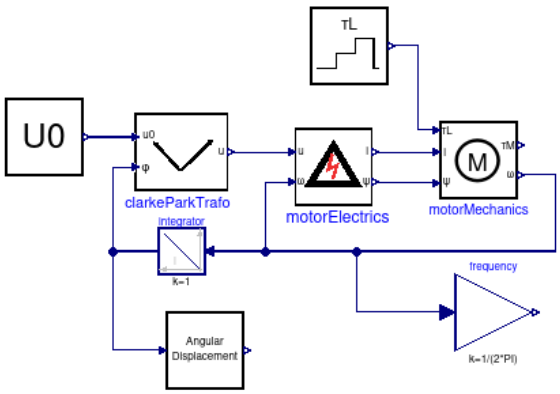
  - reproduziert d/q-Gleichungen

  Umschreiben der d/q-Gleichungen liefert

  $$\begin{aligned} \bar{u} &= \frac{d\overline{\psi}}{dt} - \omega\,\varepsilon\,\overline{\psi} + R\,\bar{\imath} \\ \tau_M &= \frac{3}{2}(\psi_d i_q - \psi_q i_d) = \frac{3}{2}\overline{\psi}'\,\varepsilon\,\bar{\imath} \end{aligned}$$

- Modell `testPMSM2`:

  Gesamtmodell



  Komponenten direkt in Modelica programmiert

- i. F. nur die wesentlichen Gleichungen im Modelica-Code

Komponente `ClarkeParkTrafo`

```
Q =[cos(phi),sin(phi); -sin(phi),cos(phi)];
u = Q*M*u0;
```

Komponente `MotorElectrics`

```
der(psi) = u + omega*epsi*psi - R*i;
L*i = psi - psi_pm;
```

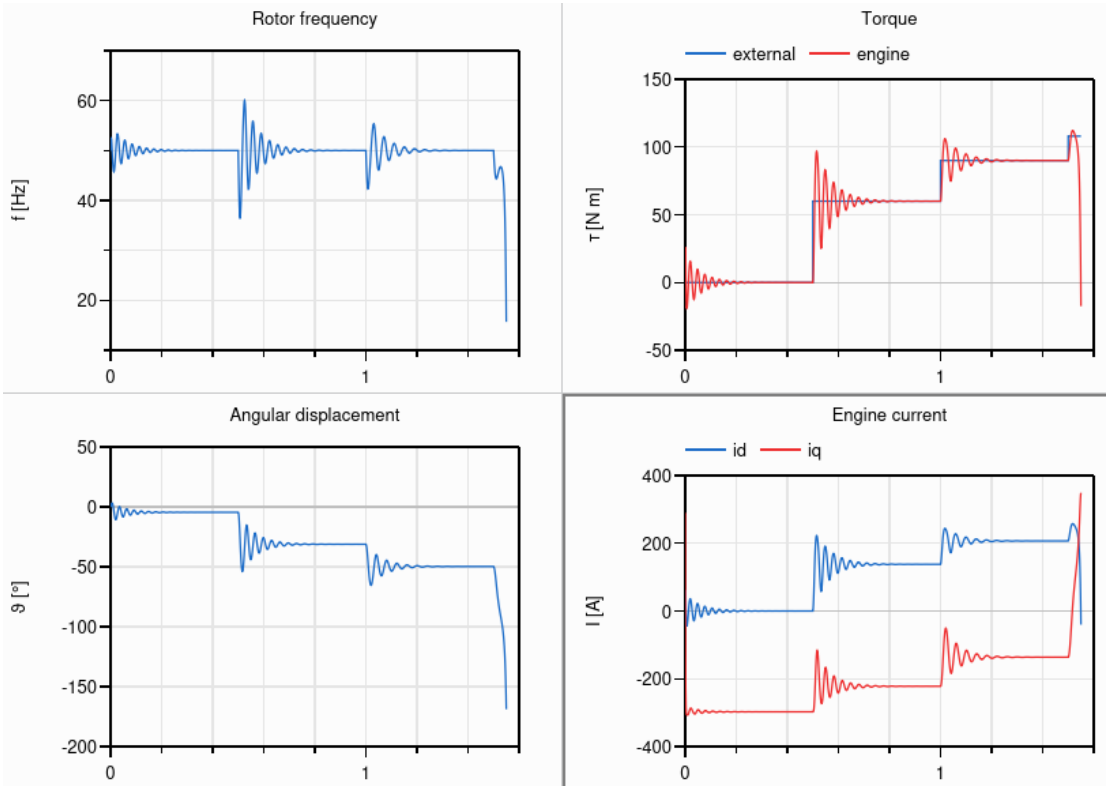
Komponente `MotorMechanics`

```
tauM = (3/2)*scalar(transpose(psi)*epsi*i);
der(omega) =(tauM - tauL)/J;
```

- `scalar` wandelt 1x1-Array in `Real`-Wert um

- Simulationsergebnisse:

  identisch zu `testPMSM1`



Motorströme im d/q-System im Gleichgewicht konstant

- Bibliothek `Modelica.Electrical.Polyphase`:

  Komponenten für mehrphasige elektrische Schaltungen
  - wichtiger Parameter m = Anzahl der Phasen
  - i. F. immer m = 3 (normaler Drehstrom)

  Connector `Plug`
  - besteht aus m Pins (Array)
  - graphisch unterschieden: `PositivePlug`, `NegativePlug`

  grundlegende Komponenten `Resistor`, `Capacitor`, `Inductor`
  - jeweils zwei Plugs mit m Widerständen (Kondensatoren, Induktivitäten) dazwischen

  übliche Quellen in package `Sources`
  - Bsp. `CosineVoltage` = 3 Wechselspannungsquellen, jeweils um 120° phasenverschoben

  Komponente `Star`
  - Anschlüsse: ein Plug, ein Pin
  - verbindet alle Plug-Pins mit dem Pin

- Bibliothek `Modelica.Electrical.Machines`:

  Komponenten für elektrische Maschinen [7]

  Connector `SpacePhasor`
  - ähnlich `Pin`, aber mit Zweiervektoren für Spannung v und Strom i
  - entspricht den Größen (u_α, u_β) bzw. (i_α, i_β) nach Clarke-Transformation

  Komponente `SpacePhasor`
  - macht Clarke-Transformation
  - Anschlüsse: zwei Plugs und ein SpacePhasor
  - Drehstrom zwischen den Plugs → Raumzeiger am SpacePhasor
  - zusätzlich zwei Pins für unsymmetrische Systeme (d.h. Ströme addieren sich nicht zu 0)
  - im einfachsten Fall Pins direkt verbinden

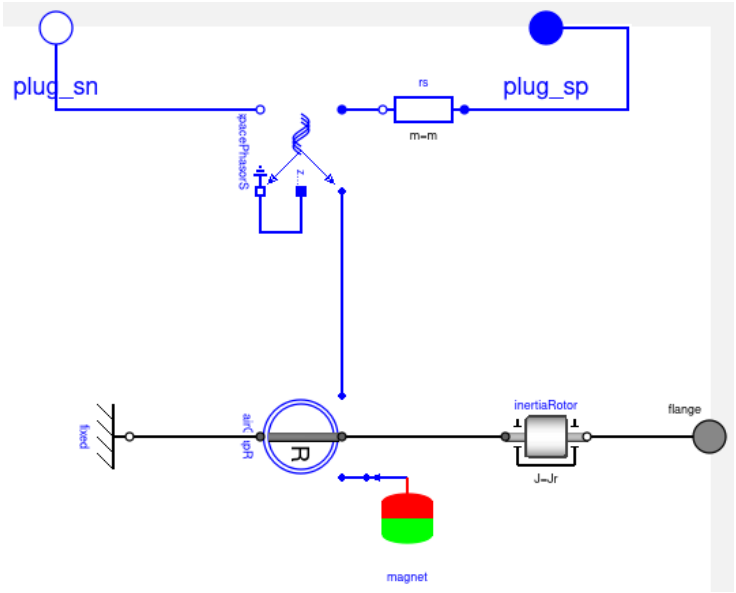
  Packages für Verluste
  - `Losses`: reibungs- und magnetische Verluste
  - `Thermal`: entsprechende Wärmeverluste

  package `BasicMachines`
  - komplette Modelle u.a. für Synchron-/Asynchron-/Gleichstrom-Maschinen
  - komplex, enthalten diverse Modelle für Verluste
  - grundlegende Bausteine in `BasicMachines.Components` (u.a. `PermanentMagnet`, `AirGapR`)

- Komponente `PMSM` aus `MultiPhysLib`:

  stark vereinfachte Version von `SM_PermanentMagnet` aus `BasicMachines.SynchronousMachines`

  Grundaufbau



Komponente `PermanentMagnet`

- im Rotorsystem: konstante Flussverkettung ψ
- hier: konstanter Strom am SpacePhasor-Anschluss
- Berechnung des Stroms als ψ_{pm}/L
- Version aus `Machines` hat immer q-Komponente 0
- Version in `SimT2Lib` hat d- und q-Komponente → reproduziert vorgegebene PMSM-Gleichungen exakt

Komponente `AirGapR` aus `Machines`

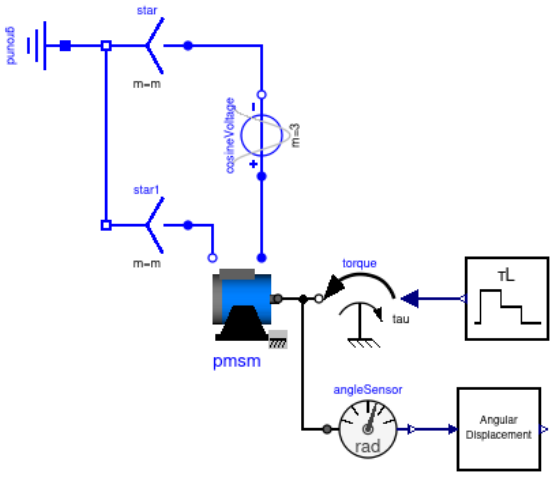
- modelliert die magnetischen und mechanischen Wechselwirkungen zwischen Rotor und Stator

Basisklasse `PartialAirGap`

- macht die Umrechnung zwischen Rotor- und Stator-System (d/q ↔ α/β)
- berechnet das Motormoment

- Modell `testPMSM3`:

  Gesamtmodell



  Simulationsergebnisse identisch zu `testPMSM2`

  für Unerschrockene

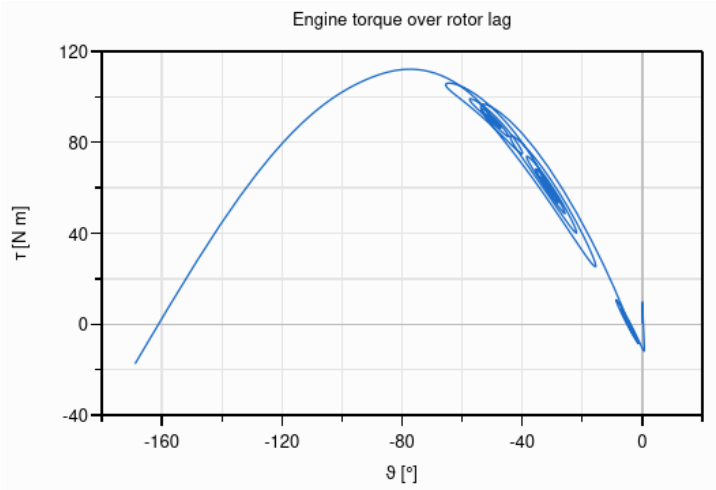
  - manuelles Aufstellen aller Gleichungen und systematische Vereinfachung
  - → Gleichungen von `testPMSM2` werden reproduziert

- Kennlinie Motormoment über Polradwinkel (vgl. [5, S.368, Abb. 8.56]):

  grundsätzliches Verhalten

- wachsendes Lastmoment → Nacheilen des Rotors nimmt zu
- → Polradwinkel ϑ wird kleiner (negativer)
- bei -90° instabil, maximales Moment erreicht

entsprechende Kurve mit `TestPMSM3` aufnehmen



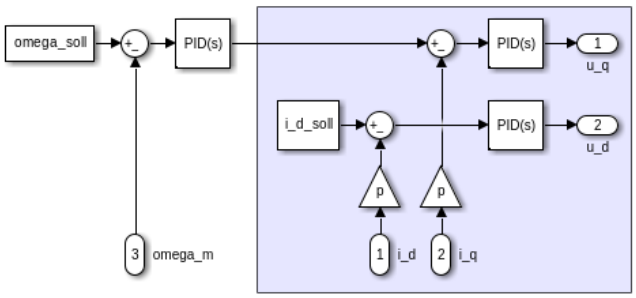
- Einschwingen verzerrt Bild
- verursacht auch instabilen Bereich schon kurz vor -90°

Abhilfe

- keine Lastsprünge
- Regelung für konstante Drehzahl

- Regelung eines Synchronmotors:

  Prinzip der Kaskadenregelung [8] [9]



  äußerer Regelkreis für Drehzahl

  - normaler PID-Regler
  - Stellgröße = Sollwert von i_q

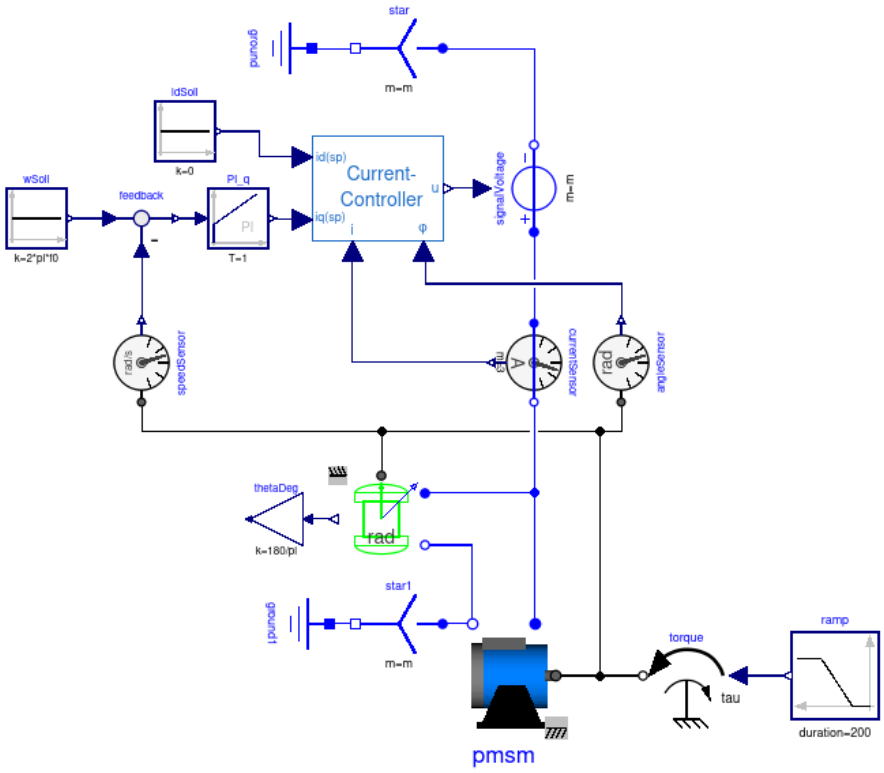
  innerer Regelkreis für Ströme

  - Vektorregelung für (i_d, i_q)
  - gemessenes i wird mit p (Zahl der Polpaare) multipliziert
  - Stellgröße = (u_d, u_q)
  - d/q-Richtung geschickt gewählt → Sollwert von i_d = 0 setzen

  wichtig: innerer Regelkreis schneller als äußerer!

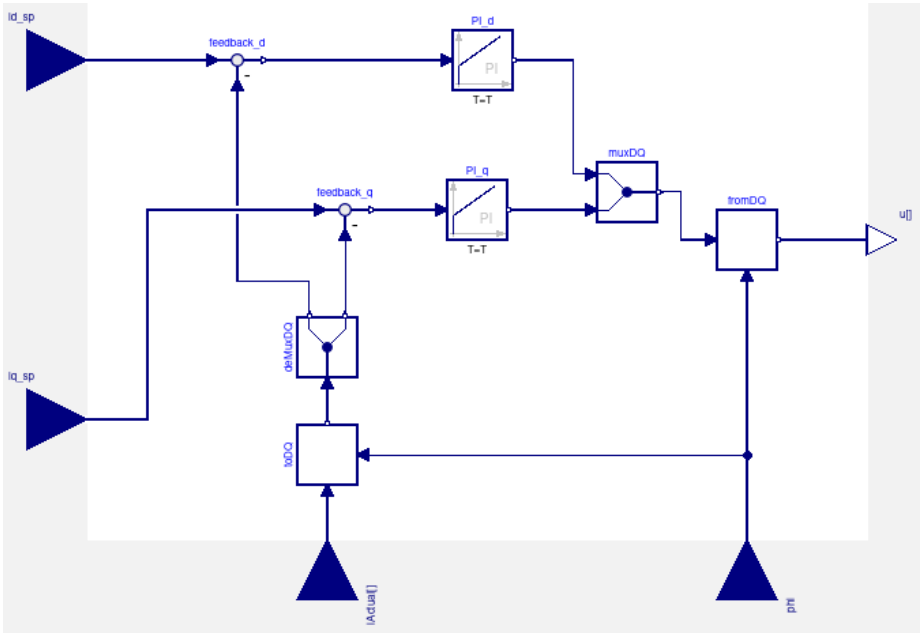
- Modell `RotorLagPMSM`:

  Gesamtmodell



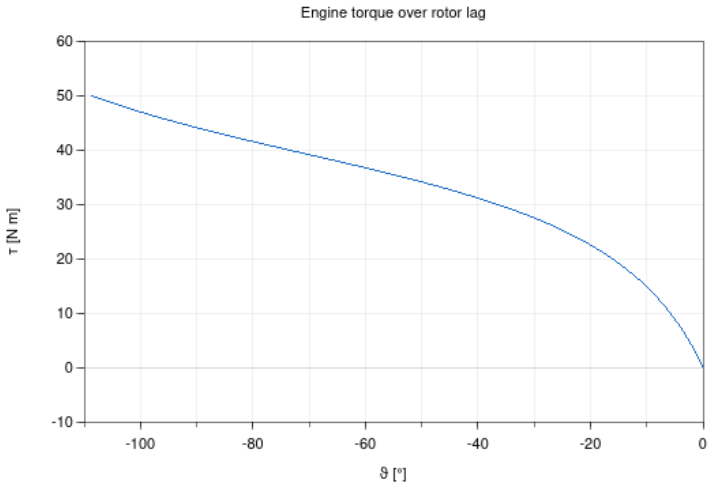
- Drehzahl wird auf vorgegebenen Wert geregelt
- `PMSM`-Modell hat $\psi_{pm,q} = 0$ (statt vorher $\psi_{pm,d} = 0$)
- Drehzahl ändert sich leicht → Block `AngularDisplacement` klappt nicht mehr
- besser: `Machines.Sensors.RotorDisplacementAngle`, misst Spannung und Rotation

Komponente `CurrentController`



- Istgröße i und Sollgröße u im a/b/c-System (Drehstrom)
- Umrechnung mit Hilfsblöcken `ToDQ` und `FromDQ` aus `Machines.Utilities`
- brauchen Rotorwinkel φ

Ergebnis



  - Stabilisierung durch Regelung ermöglicht Polradwinkel > 90°

- Aufgaben:

  Aufgabe 202

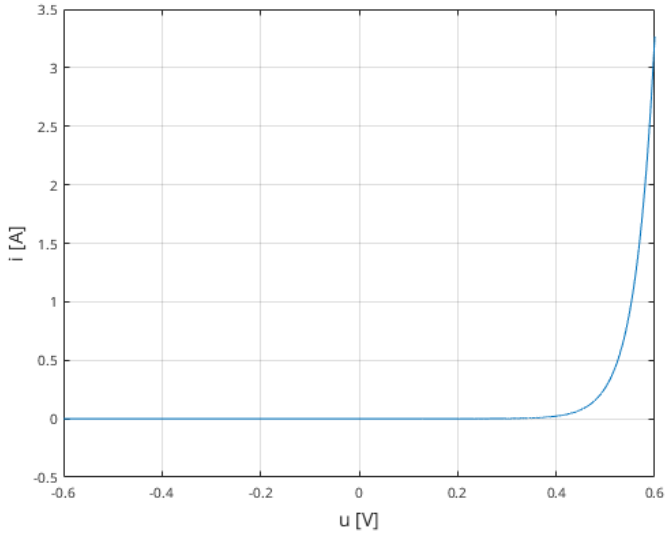
Hybride Systeme

- Behandlung von Unstetigkeiten
- Dynamische Ereignisse
- Zustandsdiagramme
- Cyber-Physikalische Modellierung

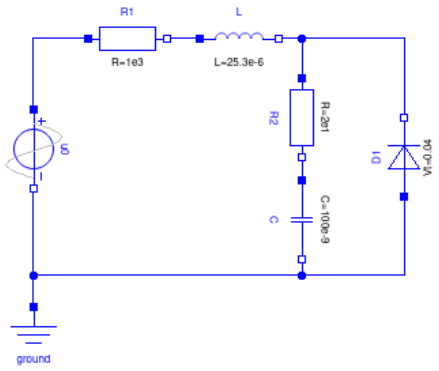
- Schaltkreis mit Diode (vgl. [10, 11]):

  Diode

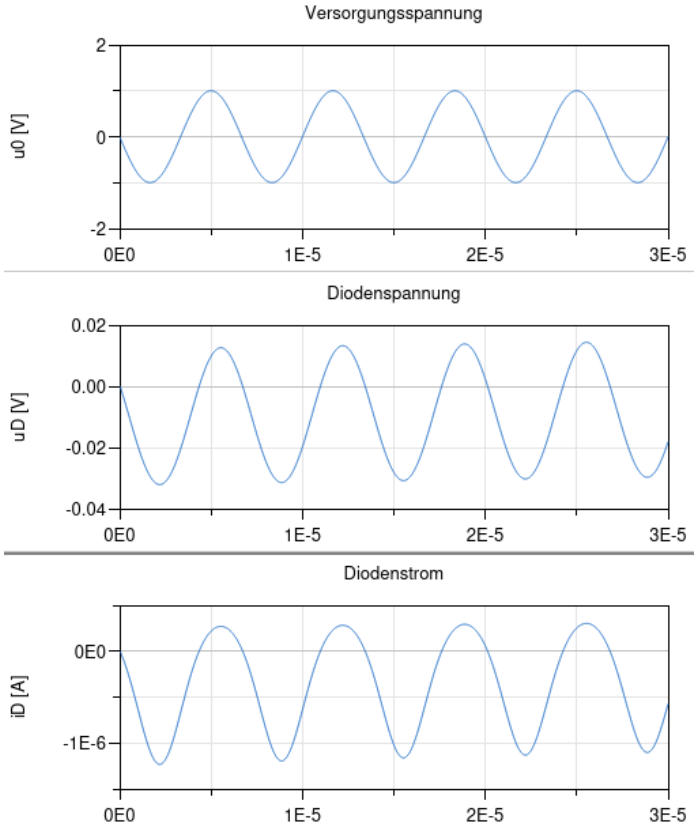
  - Halbleiter-Bauelement mit einem p-n-Übergang
  - leitet Strom in einer Richtung, sperrt in der anderen
  - typische Kennlinie



Beispiel-Schaltkreis `Diode1`



Ergebnisse



negative Spannung

- Diode leitet
- trotzdem Spannungsabfall wegen Innenwiderstand

positive Spannung

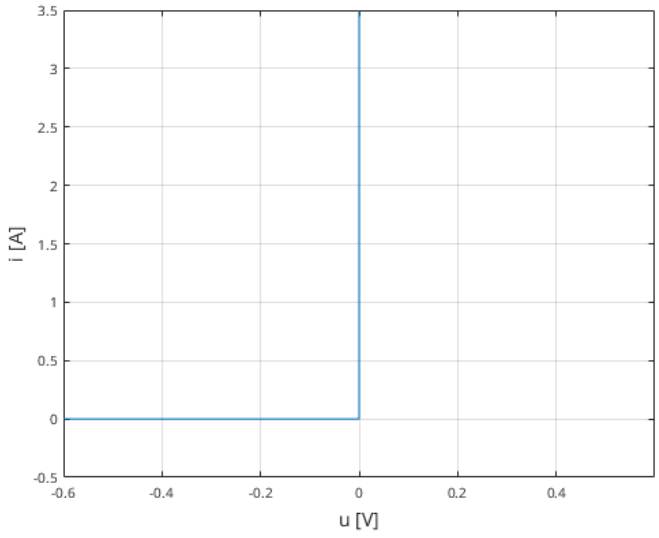
- Diode sperrt
- trotzdem Strom durch Diode (**Leckstrom**)

- Ideale Diode:

  Vereinfachung: Diode als Schalter

  - Diode leitet → kein Spannungsabfall
  - Diode sperrt → kein Leckstrom

  Kennlinie



  Wie kann man diese Kennlinie nachbilden?

- Ideale Diode in Modelica:

Problem

- Kennline keine Funktion

Idee

- Nachbilden mit Kurvenparameter s
- s = 0 → Knickpunkt (0,0)
- s < 0 → s ≙ Spannungswert, Kurve (s, 0)
- s > 0 → s ≙ Stromwert, Kurve (0, s)

Basismodell `OnePort`

- Elternmodell für einfache Bauelemente mit zwei Anschlüssen
- definiert positiven und negativen Pin (p, n)
- definiert Variable v = p.v - n.v
- definiert Variable i = p.i
- hat Gleichung p.i + n.i = 0
- Kindmodell muss nur noch Kennline i(v) definieren

Diodenmodell `SCDiode`

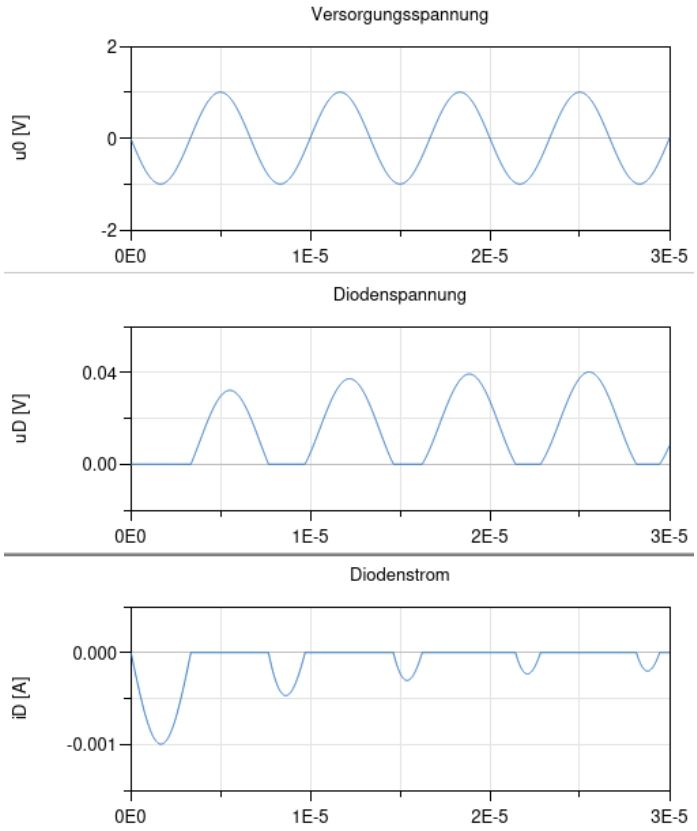
- "Short-cut"-Diode nach [L3]
- Modelica-Code

```
model SCDiode "short-cut diode after Fritzson 2015, p.56"
  extends Modelica.Electrical.Analog.Interfaces.OnePort;
  Real s;
  Boolean off;
equation
  off = s < 0;
  v = if off then s else 0;
  i = if off then 0 else s;
  annotation (...);
end SCDiode;
```

- Modell `Diode2` mit idealer Diode:

  Diode aus MSL ersetzt durch `SCDiode`

  läuft problemlos, Ergebnis



alternativ `Diode3` mit MSL-Komponente `IdealDiode`

- allgemeinere Version mit Knickpunkt und anderen Steigungen
- reproduziert Ergebnisse von `Diode2` bei $R_{on} = 0$, $G_{off} = 0$

- Simulation mit Unstetigkeiten:

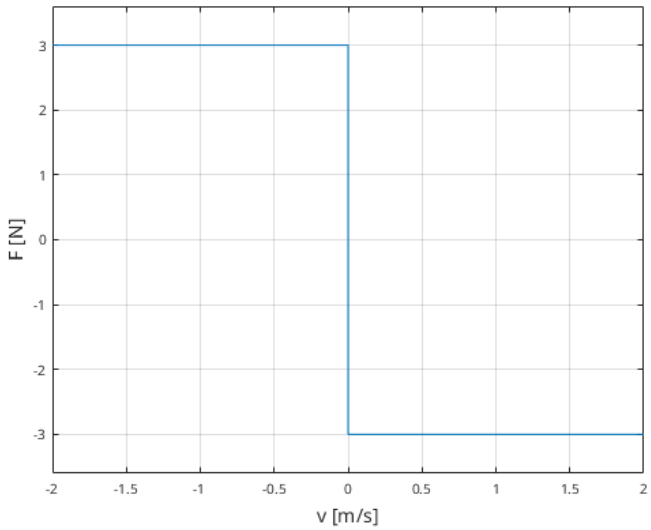
  Funktion ist unstetig → "normaler" ODE-Solver kann das nicht

  Beschreibung von Unstetigkeiten
  - zusätzliche Funktion h(t) aus Systemgrößen
  - Unstetigkeit tritt ein ↔ h(t) wechselt Vorzeichen
  - Sprachweise: Ereignis ("Event") ist eingetreten
  - in Modelica h u.a. aus Bedingungen in if

  Grundvorgehen eines ODE-Solvers mit Ereignisbehandlung
  - normaler ODE-Solver berechnet Systemgrößen und h
  - h wechselt Vorzeichen → Zeitpunkt t_0 berechnen, an dem h = 0 (z. B. mit Newton)
  - Solver integriert "normal" bis t_0
  - Solver ändert ggf. Werte der Systemvariablen
  - Solver integriert (mit "anderem Zweig") ab t_0 weiter

- Haft- und Gleitreibung:

  Gleitreibung ist unstetig

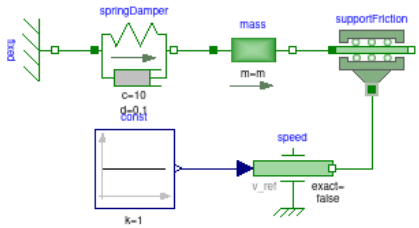


grundsätzlich wieder durch Parametrisierung abbildbar

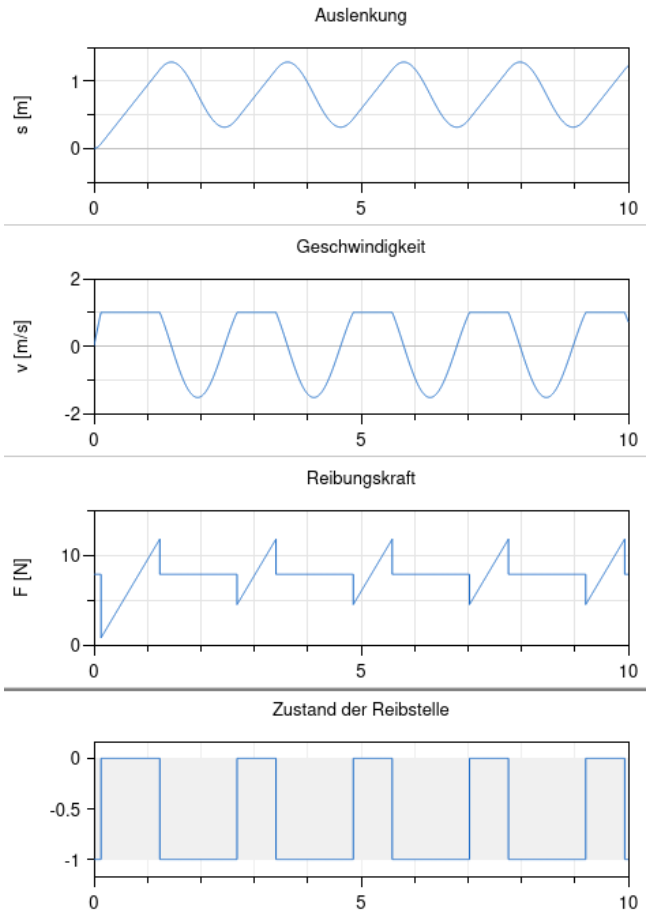
deutlich komplexer durch Haftreibung

MSL-Komponente `SupportFriction` beschreibt zusätzliche Modi

Beispiel `Stickslip`



- Ergebnisse

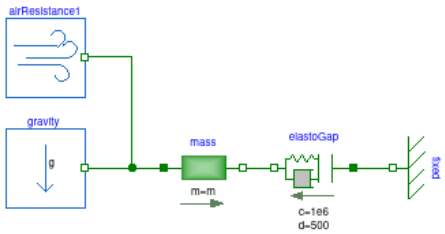


Dynamische Ereignisse

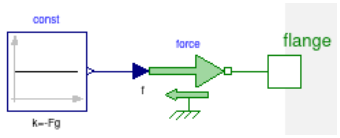
- Beispiel `BouncingBall1`:

  betrachtetes System
  - Ball fällt unter Schwerkraft-Einfluss incl. Luftwiderstand
  - wird beim Aufprall auf dem Boden reflektiert
  - durch Verformung geht dabei ein Teil der kinetischen Energie verloren

  Modellierung



  Komponente `Gravity1d`
  - konstante Kraft

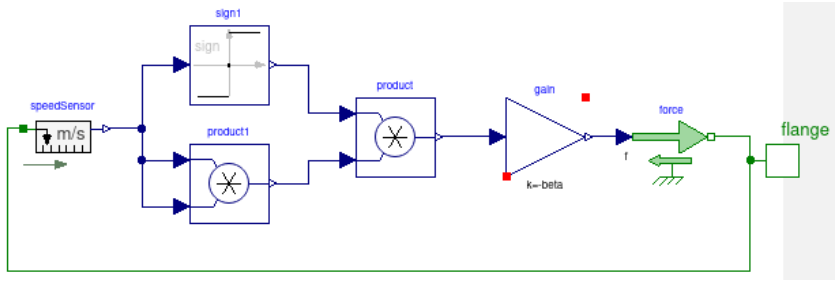


  - unschön: braucht Masse → Masse als globaler Parameter im Modell

  Komponente `AirResistance1d`
  - Luftwiderstand

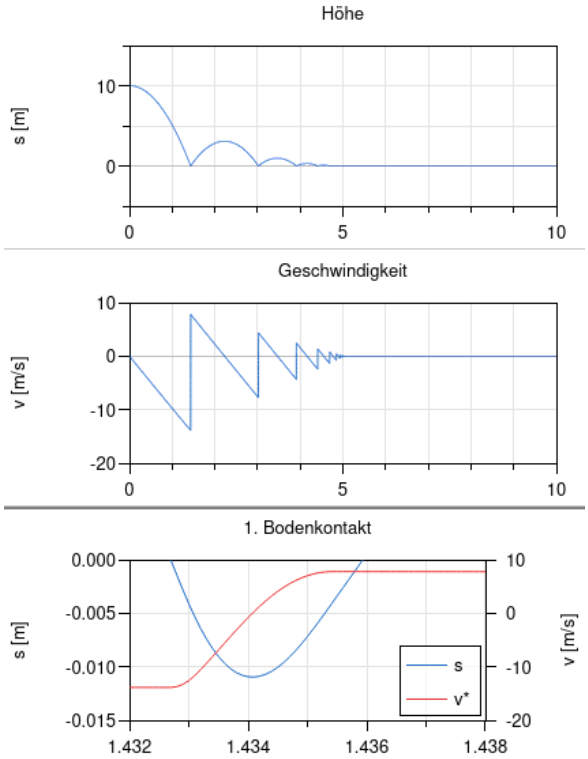
$$F_r = -\beta v^2 \text{sign}(v)$$

  - einfache Implementierung



  Komponente `ElastoGap` aus der MSL
  - grundsätzlich paralleles Feder-Dämpfer-System
  - Kraftübertragung nur in einer Richtung (nur push, kein pull)
  - enthält komplexe Korrekturen für sinnvolles Verhalten

  Ergebnisse



- Vereinfachung des Verhaltens:

  Details beim Kontakt interessieren meistens nicht

  Idealisierung
  - Kontakt ist instantan (Event)
  - passiert bei Höhe 0
  - Geschwindigkeit springt von v auf -μ v (0 < μ < 1)

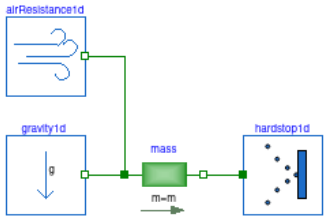
  Implementierung in Modelica (1. Versuch!)

```
model Hardstop1dA "1D translational hardstop"
  parameter Real mu = 0.9 "Coefficient of restitution";
  Position s;
  Velocity v;
  Modelica.Mechanics.Translational.Interfaces.Flange_a flange_a
                                                  annotation (...);
equation
  s = flange_a.s;
  v = der(s);
  when s <= 0 then
    reinit(v, -mu*pre(v));
  end when;
  flange_a.f = 0;
  annotation (...);
end Hardstop1dA;
```

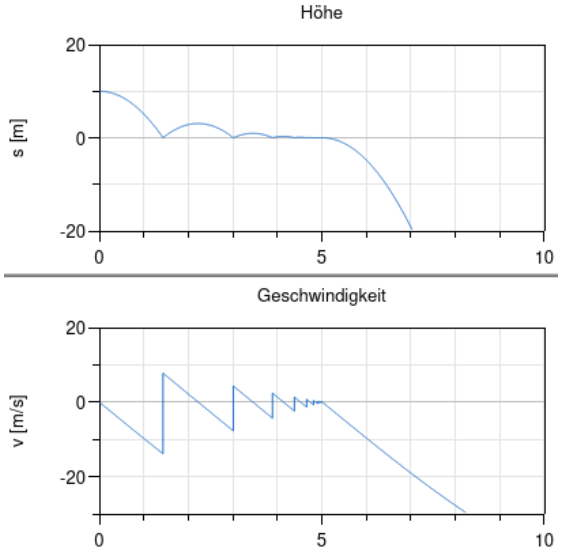
  `when A` löst ein Event aus, sobald A `true` wird

  beim Event ändert `reinit` v auf -μ v
  - verwendet den unmittelbar vorhergehenden Wert von v (`pre(v)`)
  - nötig, da Gleichungen verwendet werden, keine Zuweisungen

  Gesamtmodell `BouncingBall2A`



Ergebnis



- Ball fällt durch den Boden

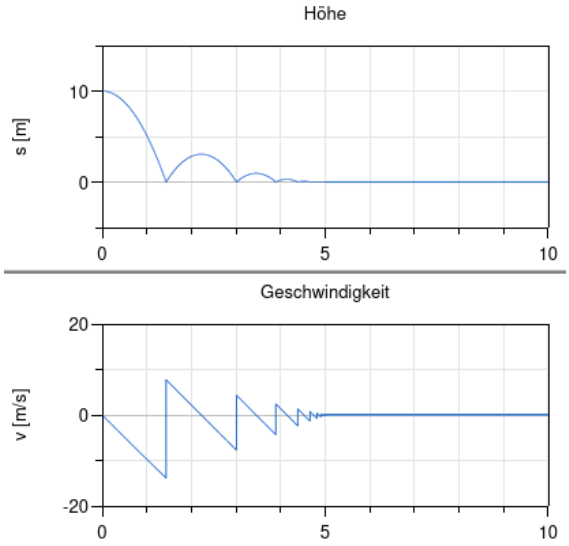
Ursache: in der Ruhelage fehlt die Gegenkraft zur Schwerkraft

- Funktionierende Version `BouncingBall2B`:

verbesserter `Hardstop1D`

```
model Hardstop1dB "1D translational hardstop"
  parameter Real mu = 0.9 "coefficient of restitution";
  parameter Acceleration g = 9.81  "gravity constant";
  parameter Mass m = 1.0 "mass";
  Position s;
  Velocity v;
  Modelica.Mechanics.Translational.Interfaces.Flange_a flange_a
annotation ();
  Boolean flying;
equation
  flying = not (s <= 0);
  s = flange_a.s;
  v = der(s);
  flange_a.f = if flying then 0 else -m*g;
  when s <= 0 then
    reinit(v, -mu*pre(v));
  end when;
end Hardstop1B;
```

Ergebnis



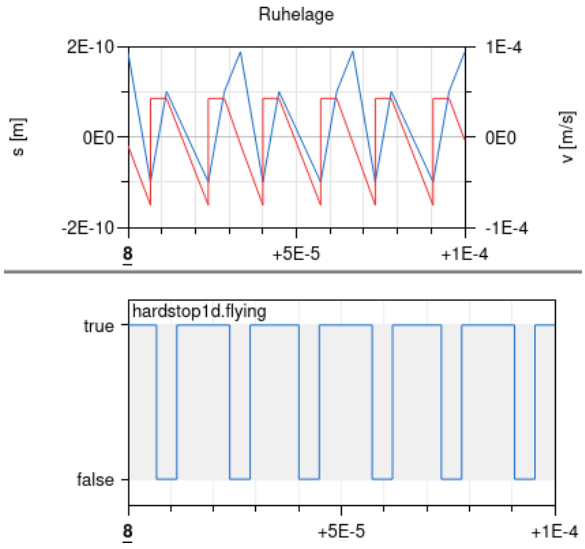
- grundsätzlich ok

Problem

- numerische Ungenauigkeit führt zu riesiger Zahl an Events

CPU-time for integration	22.1 s
Number of f-evaluations (dynamics)	5816168
Number of state events	595918
Minimum integration stepsize	1.51e-14

Verhalten während der Ruhelage



- Verbesserte Lösung `BouncingBall2C`:

  Trick zur Reduktion der Events

  - ändere

    ```
    flying = not (s <= 0);
    ```

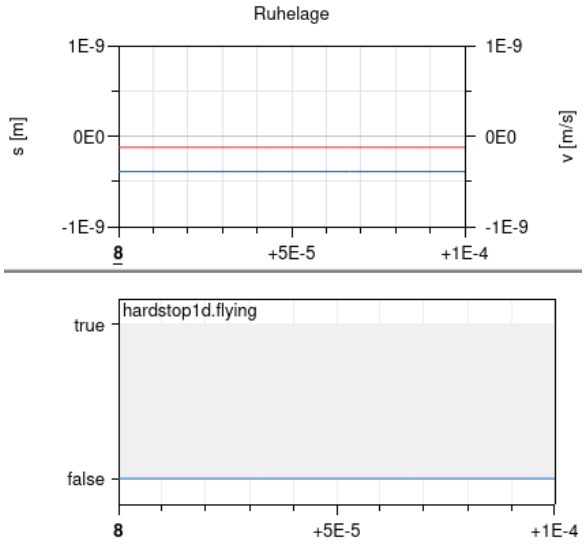
  - zu

    ```
    flying = not (s <= 0 and v <= 0);
    ```

  Idee

  - `(not flying)` tritt seltener auf (nämlich nicht beim Hochkommen)
  - → beim Hochkommen keine zusätzliche Gegenkraft
  - → Ball bleibt unter 0

  Verhalten während der Ruhephase



- s und v sind beide minimal < 0
- → flying = false
- → Gegenkraft hält alles stabil

Rechnung wird sehr viel schneller

CPU-time for integration	1.09 s
Number of f-evaluations (dynamics)	617
Number of state events	44
Minimum integration stepsize	2.12e-09

Zustandsdiagramme

- Zustandsdiagramm (**Statechart**):

  graphische Beschreibung des Verhaltens eines diskreten Systems

  viele leicht unterschiedliche Fassungen, z.B.
  - Statechart [12] (theoretische Basis)
  - Stateflow (Simulink-Erweiterung von Mathworks)
  - State Machines (Modelica-Version)

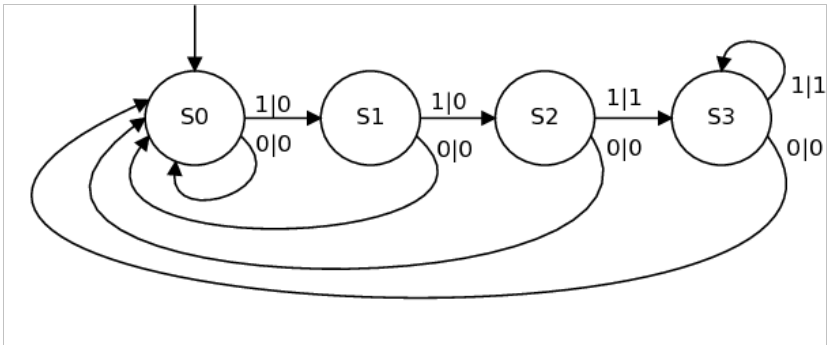
  Grundidee
  - System befindet sich in einem von endlich vielen Zuständen
  - ein Zustand ist als Anfangszustand ausgezeichnet
  - Zustand wechselt bei Eintreten vorgegebener Ereignisse
  - in einem Zustand können Aktionen ausgeführt werden

  Modelica-spezifisch
  - State Machine ist getaktet
  - führt bei jedem Takt ihre internen Aktionen aus
  - Takt gegeben durch Clock
  - entsprechende **Clocked Variables** haben nur während eines Clock-Ticks einen Wert

- Beispiel: Erkennung eines Eingangssignals:

  Aufgabenstellung
  - automatische Fertigungskontrolle
  - kein Problem bei bis zu 2 Fehlern
  - Fehlersignal (manuelles Eingreifen nötig) bei 3 aufeinanderfolgenden Fehlern

  Darstellung als endlicher Automat mit vier Zuständen
  - Eingangsfolge aus `false` ("ok") und `true` ("Fehler")
  - Ausgabe `true` bei drei aufeinanderfolgenden Fehlern, sonst `false`



  - Eingang `true` in Zustand S3 → bleibt in Zustand S3
  - alternative Möglichkeit: geht in Zustand S1

- Aufbau von `ErrorDetector1A` in Modelica:

  Zustände `s0`, `s1`, `s2`, `s3` erzeugen mit `Create Local State` und anordnen
  - erzeugt in Modelica jeweils einen Block + eine Variable vom entsprechenden Typ, z.B.

```
block S0
   ...
end S0;
S0 s0;
```

  Anfangszustand festlegen
  - von oberer Kante von `s0` nach oben ziehen
  - beenden mit Doppelklick

- `Create Initial State` wählen
- erzeugt Gleichung `initialState(s0);`

Transitionen hinzufügen

- von `s0` nach `s1` ziehen, Condition `u` (Eingangsvariable)
- erzeugt Gleichung `transition(s0,s1,u);`
- analog `s1` → `s2` und `s2` → `s3`
- von `s1` → `s0` mit Condition `not u`
- analog `s2` → `s0` und `s3` → `s0`

States mit booleschen Eingängen versehen

- Zustand öffnen
- `Modelica.Blocks.Interfaces.BooleanInput u` hinzufügen
- entweder graphisch aus Bibliothek oder mit Modelica-Code direkt in Block-Definition

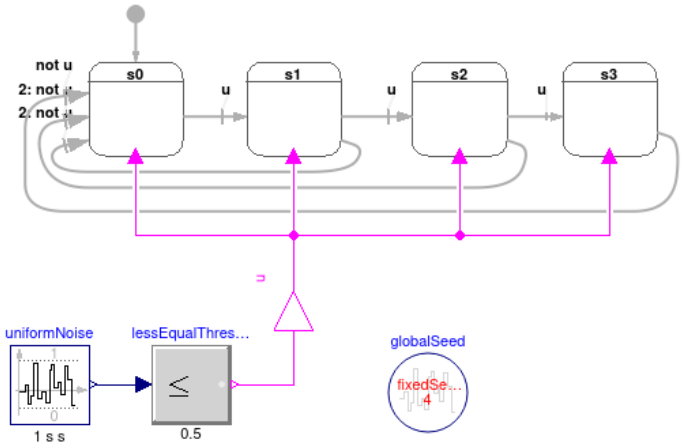
Eingabesignal erzeugen

- `Modelica.Blocks.Noise.UniformNoise` und `GlobalSeed` hinzufügen
- Parameter setzen: `samplePeriod=1, y_min=0, y_max=1`
- → erzeugt jede Sekunde gleichverteilte Zufallszahl aus [0,1]
- Block `LessEqualThreshold` mit `threshold=0.5` dahinter
- → erzeugt jede Sekunde Zufallswert true/false

Eingabesignal mit Block-Eingängen verbinden

- Problem: Variable `u` ist jeweils block-intern
- `u` wird aber in `transition` benutzt (außerhalb der State-Blöcke)
- daher Variable `u` auf oberem Level einführen
- dazu `BooleanOutput u` hinzufügen
- mit `LessEqualThreshold`-Block und allen `u`-Eingängen verbinden

Gesamtmodell

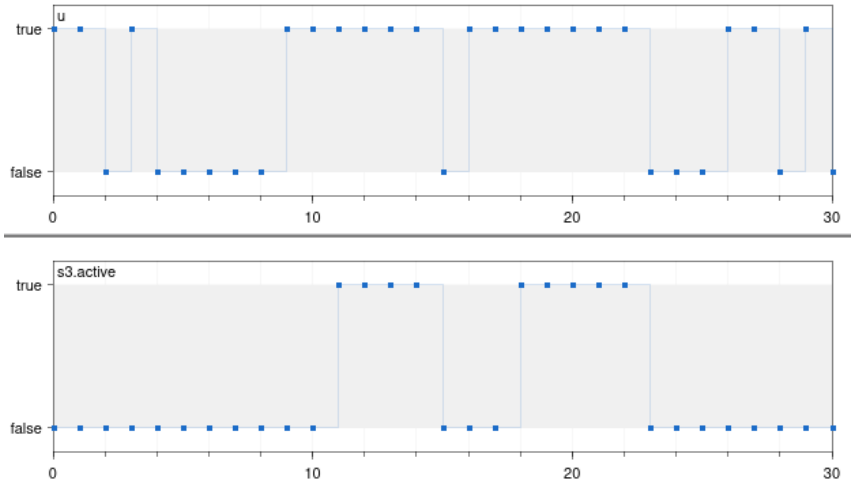


Eingänge in States hier gar nicht nötig

- Simulation von `ErrorDetector1A`:

  `Simulate` liefert Warnings, zunächst ignorieren

  Input `u` und Zustand `s3.active` plotten



Variable sind **clocked** → Werte nur in festen Zeitabständen definiert

- Zeittakt hier erschlossen (aus `samplePeriod` von `Noise`)
- Ursache der Warnungen

das erste Mal drei `true`'s hintereinander bei t = 11

- `s3.active` wird sofort `true` bei t = 11
- Grund: alle Transitionen sind `immediate` (nachprüfen!)
- Darstellung: Querschnitt am Ende des Pfeils
- könnte in Modellen zu "algebraischen Schleifen" führen
- Ausweg: `immediate = false` (**delayed transition**),
- Darstellung: Querschnitt am Anfang des Pfeils
- → Transition erfolgt einen Schritt später (vgl. `ErrorDetector1B`)

- State Machine als Komponente:

  komplette State Machine in eigene Komponente `CounterSM`

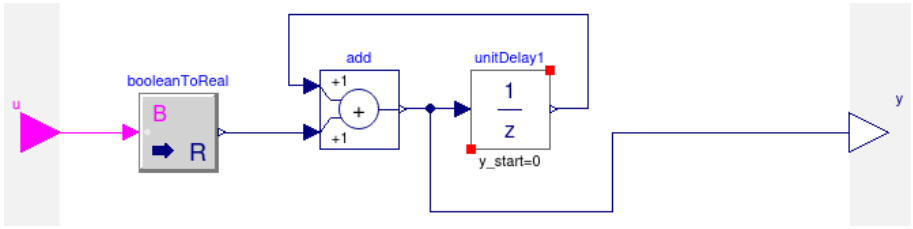
  - booleschen Eingang `u` mit allen Zustandseingängen verbinden
  - boolescher Ausgang `s3Active`
  - explizite Gleichung hinzufügen

    ```
    s3Active = activeState(s3);
    ```

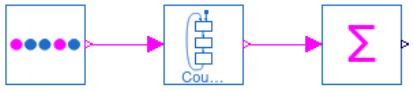
  Eingangssignal in Komponente `ErrorSignal`

  - zusätzlicher Parameter p = Wahrscheinlichkeit eines Einzelfehlers
  - dazu `threshold` von `LessEqualThreshold` auf p setzen

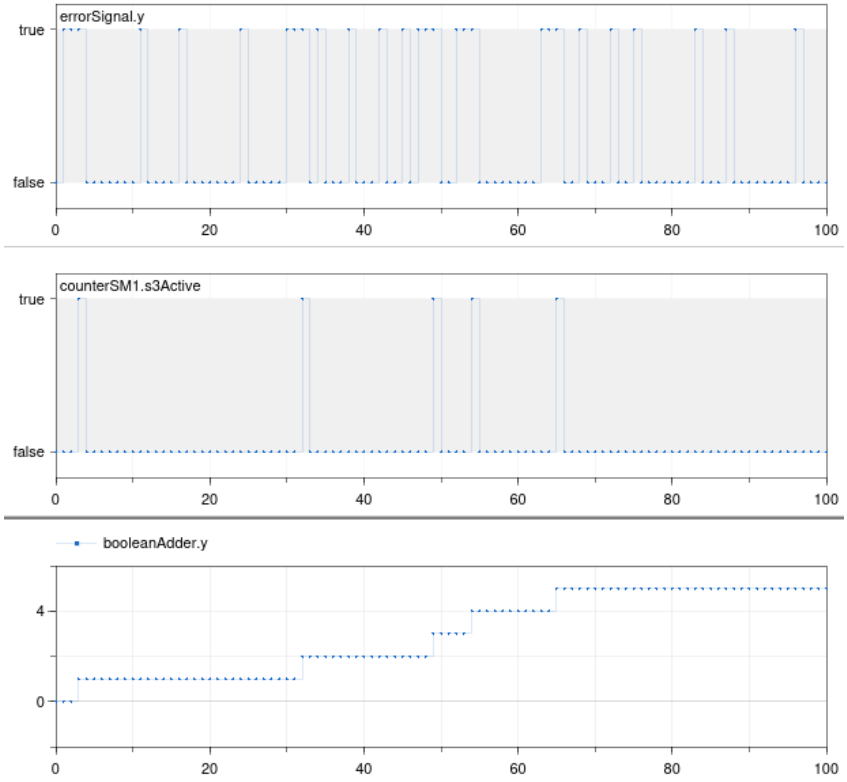
  zusätzlicher Zähler `BooleanAdder` für Anzahl der Dreifachfehler



  Gesamtmodell `ErrorDetector2`



  typisches Ergebnis bei p = 0.3



- Transition nach Wartezeit:

  Wie kann man einen Zustand nach vorgegebener Wartezeit verlassen?

  typischer Baustein **Monoflop**
  - Eingangssignal wird true → Ausgang bleibt feste Zeit auf true

  Idee:
  - Variable `step` wird bei jedem Zeitschritt um 1 erhöht
  - Transition, wenn `step >= stepMax`

  Problem: `step` muss innerhalb und außerhalb des Zustands definiert sein

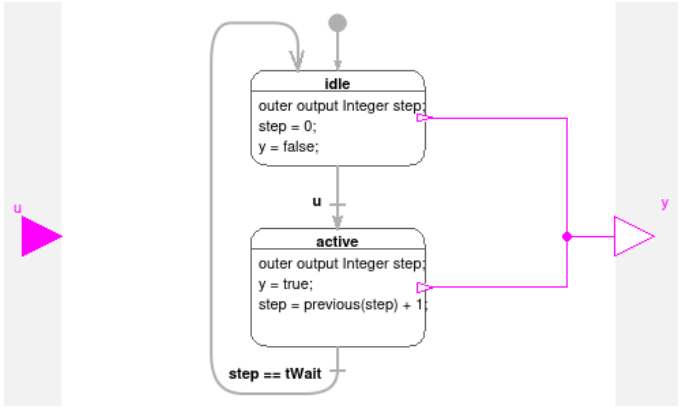
  Lösung
  - `step` wird außen als `inner` definiert

    ```
    inner Integer step(start=0);
    ```
  - im Zustand ist `step` als `outer output` definiert

    ```
    outer output Integer step;
    ```

  Ausgangswert `y` definieren
  - `MonoFlop1` und Zustände haben `BooleanOutput`
  - Zustände definieren Wert von `y`
  - Zustandsausgänge verbunden mit Komponentenausgang
  - seltsam: mehrere Ausgänge an einem Eingang
  - aber: klappt, da immer genau ein Zustand aktiv

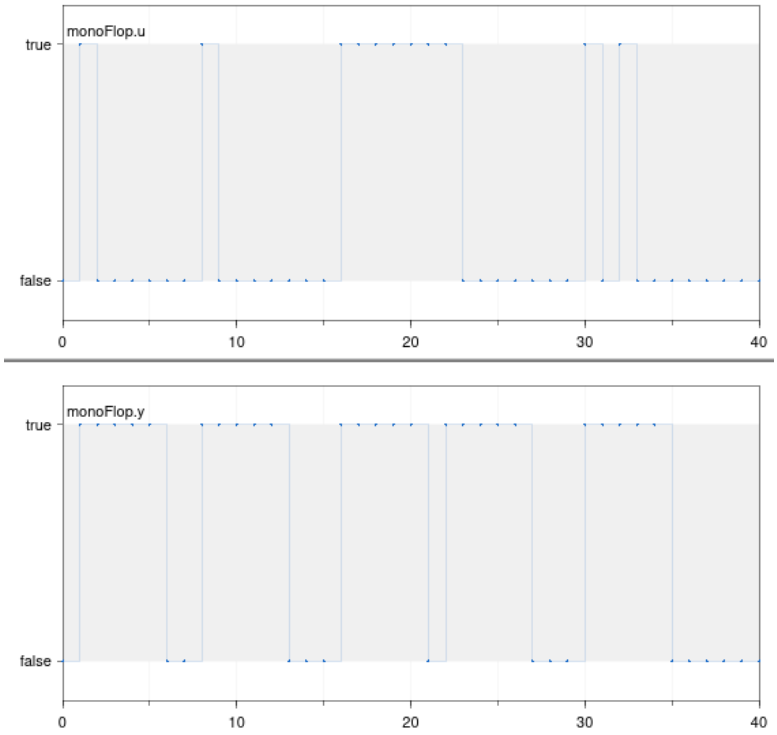
  Gesamtaufbau des `MonoFlop1`



Anzeige des Verhaltens direkt in der Graphik

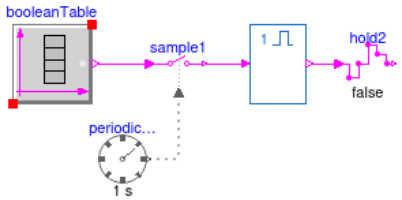
- dazu `Show text` beim Erzeugen des Zustands
- nachträglich: Diagram-annotation `textString="%stateText"`

typisches Verhalten von `TestMonoFlopA`



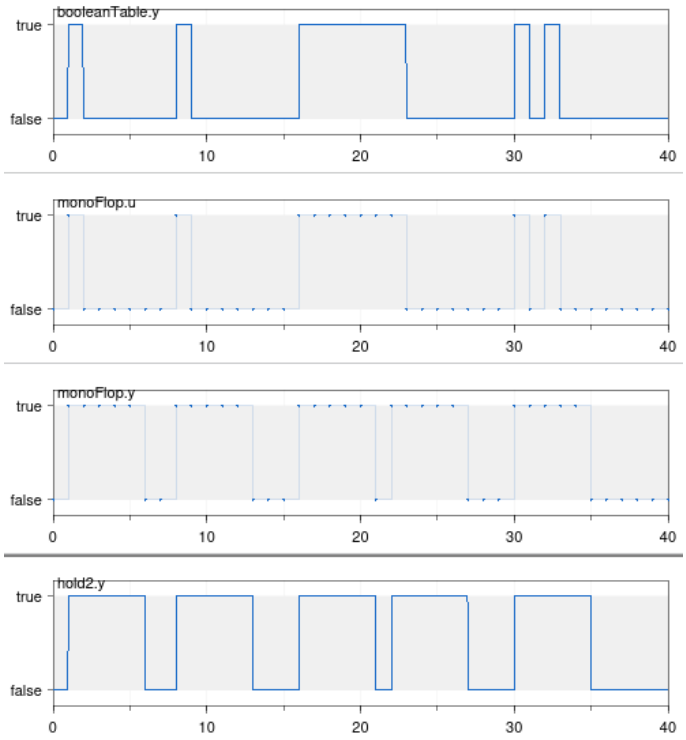
- Verbesserung `TestMonoFlopB`:

  Beseitigung der Warnungen durch explizite Definition einer Clock



  - `SampleClocked` aus `Modelica.Clocked.BooleanSignals.Sampler` tastet Eingangssignal ab und erzeugt ein "clocked" Ausgangssignal
  - Zeitwerte kommen von `Modelica.Clocked.ClockSignals.Clocks.PeriodicExactClock`
  - Umwandlung von clocked MonoFlop-Ausgang in "normale" Ausgangswerte mit `Hold`-Komponente

  Anzeige der "normalen" und "clocked" Signale



- Echte Angabe der Wartezeit tp:

  Problem: Änderung der Clock-Rate ändert auch die Wartezeit des MonoFlops

  Lösung 1

  - Sample-Intervall bestimmen mit

    ```
    Real Ts = interval(signal1);
    ```

  - dabei `signal1` irgendein clocked Signal
  - Transition dann bei

    ```
    step == integer(tp/Ts)
    ```

  Lösung 2 (vgl. `TestMonoFlopC`)

  - komplett auf `step` verzichten
  - Transition bei

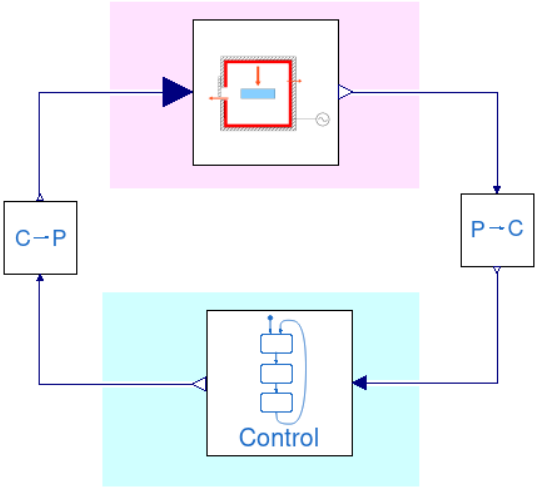
    ```
    timeInState() >= tp
    ```

  - klappt nicht bei geschachtelten Zuständen (s.u.)

Cyber-Physikalische Modellierung

- Cyper-Physikalisches System:

  Kombination aus physikalischem System und Berechnungssystem



  Physikalisches System
  - Fertigungsmaschine
  - (mehrere) Roboter
  - Fahrzeug
  - jeweils incl. Aktuatoren und Sensoren

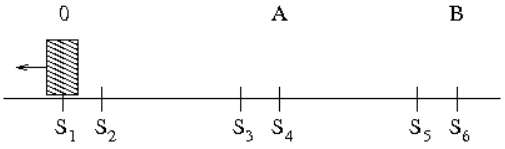
  Berechnungssystem
  - einfacher Regler
  - komplexe Steuerung
  - mehrere gekoppelte Rechner

  Kopplungen
  - Wandlung zwischen Signaltypen (z.B. getaktet ↔ kontinuierlich)
  - Transport der Signale durch Bussysteme oder Netzwerke
  - komplexe Kommunikation zwischen Komponenten

  Beispiele
  - komplexe Fertigungsmaschine incl. Steuerung
  - mehrere kooperierende Roboter
  - medizinische Geräte
  - autonomes Fahrzeug

- Beispiel Robotersteuerung:

  Roboter soll auf Startsignal hin Objekt von A nach B bringen



  Motorverhalten beschrieben durch
  - drei Geschwindigkeiten: 0, langsam, schnell
  - zwei Richtungen: nach links, nach rechts
  - zusammmengefasst in Variable v

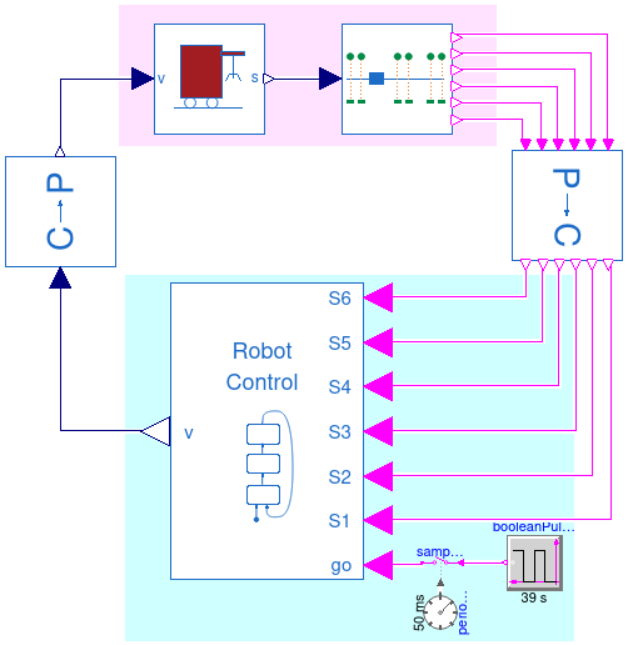
  Sensoren S1 .. S6 zeigen Erreichen einer Position an

  Ablauf
  - Startposition ist 0, v = 0
  - Startsignal → Roboter fährt nach A, zunächst schnell, dann ab S3 langsam
  - bleibt bei S4 stehen, greift das Objekt (als Zeitverzögerung modelliert)

- fährt nach B, zunächst schnell, ab S5 langsam
- bleibt bei S6 stehen, setzt Objekt ab (Zeitverzögerung)
- fährt zurück nach 0, erst schnell, langsam ab S2
- bleibt bei S1 stehen und wartet auf neues Startsignal

- Gesamtmodell `RobControl`:

  entspricht allgemeinem Schema



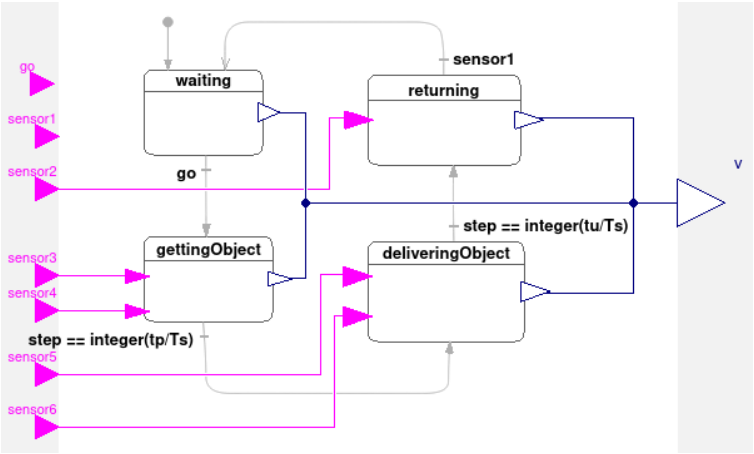
  Physikalisches Modell
  - einfaches Robotermodell mit Input für v und Ausgabe des Orts s
  - Modell der Sensoren S1 .. S6 längs der Strecke

  Berechnungs-Modell
  - Steuerung durch eine State Machine
  - zusätzlich: Erzeugung von Start-Impuls (go)

  Wandler clocked → continuous bzw. zurück

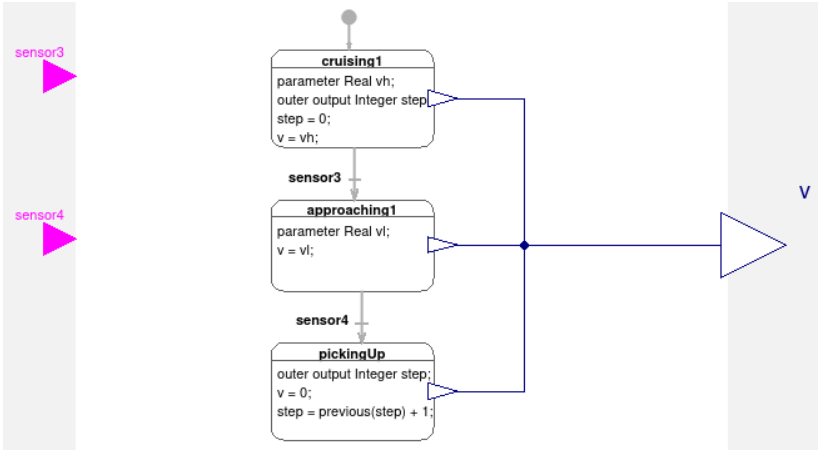
- Modell der Steuerung `RobControlSM`:

  Gesamtaufbau

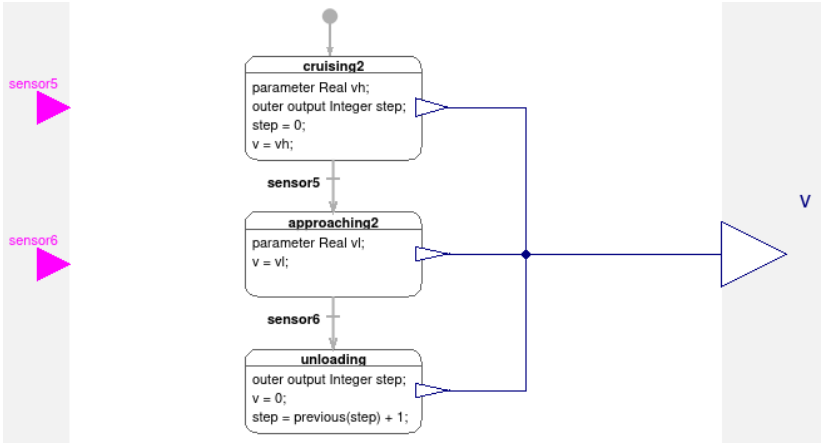


  - Zustand `waiting` setzt v = 0
  - alle anderen enthalten Unterzustände (hierarchische State Machine)

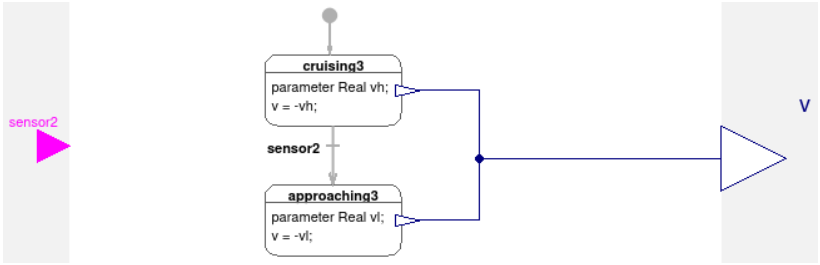
  Aufbau von `GettingObject`



Aufbau von `DeliveringObject` i. W. identisch



Aufbau von `Returning`



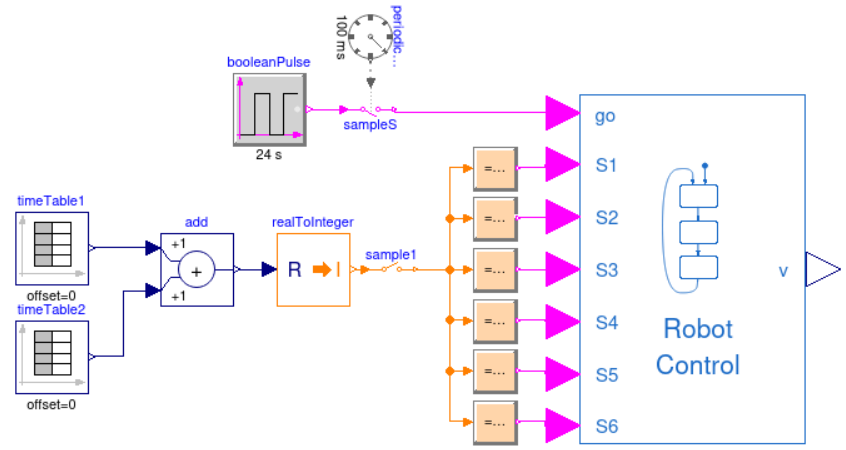
Implementierungsdetails

- `timeInState()` geht nicht über Hierarchiestufen
- Variable `step` wandert durch die Maschinen

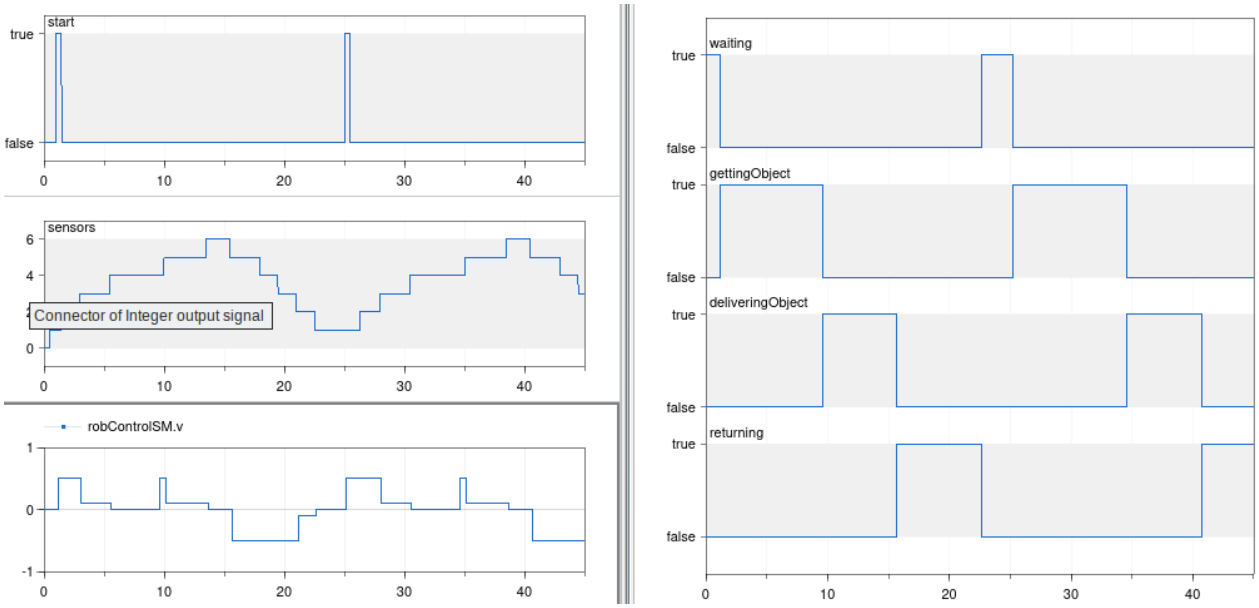
`RobControlSM`	`inner Integer step(start=0);`
`GettingObject`	`inner outer output Integer step;`
`Cruising1`	`outer output Integer step;`

Test der Steuerung `TestRobControlSM`

- Eingabewerte zu sinnvoll gewählten Zeiten vorgeben

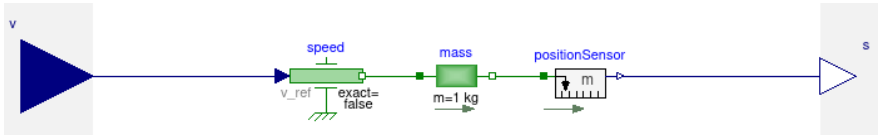


- Ergebnisse

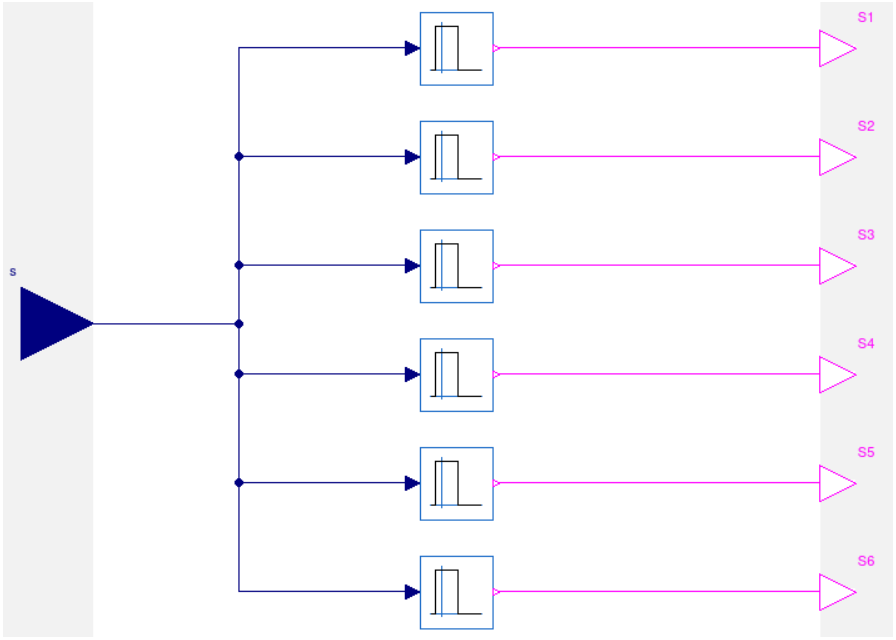


- Weitere Bausteine von `RobControl`:

`SimpleRobot`



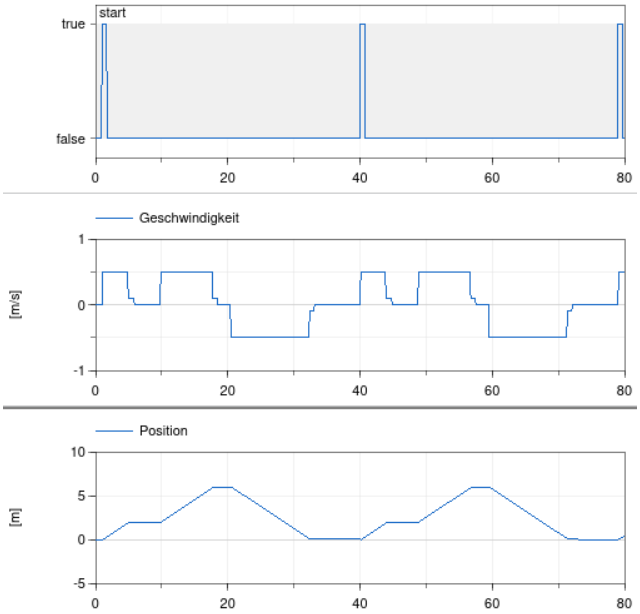
`SimpleRobotSensors`



- verwendet einfache Komponente `IntervalTester`

Wandler mit `Sample` bzw. `Hold`

Ergebnisse



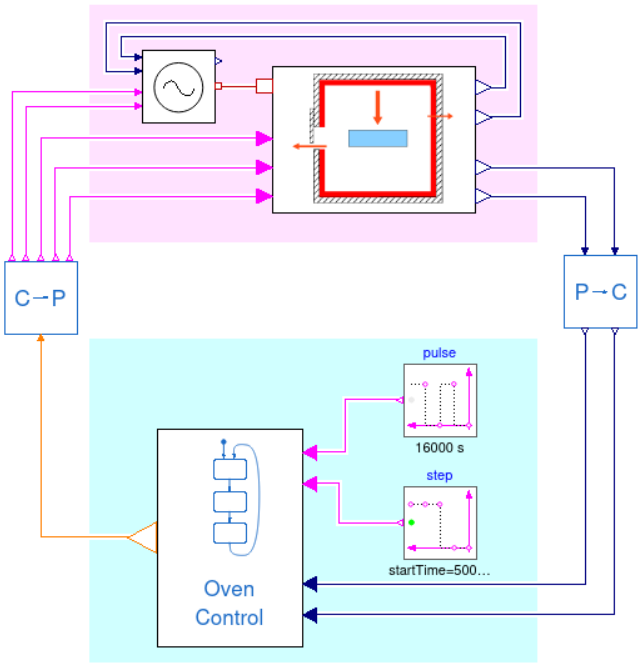
- Beispiel Industrieofen:

  Ofen zum Glühen
  - Werkstoff wird zunächst auf vorgegebene Temperatur erhitzt
  - Temperatur wird gegebene Zeit gehalten
  - Prozess wird mit anderer Temperatur wiederholt
  - ggf. wird am Ende der Ofen abgekühlt

  besonders interessant: Energieverbrauch
  - Modellierung der Wärmeflüsse
  - Ofen zu Werkstück (Strömung + Strahlung)
  - Ofen zu Umgebung (Leitung durch die Wand)
  - Ofen zu Umgebung (Strömung + Strahlung durch offene Tür)

- Gesamtmodell `OvenControl`:

  wieder nach allgemeinem CPM-Schema



physikalisches Modell aus Energieversorgung und Ofen

Energieversorgung `PowerSupply`

- gibt zum Aufheizen Leistung P_h ab
- gibt benötigte Leistung zum Halten der Temperatur ab
- hat immer eine Grundlast P_0
- Eingänge `Ploss` und `Pload` vom Ofen
- Eingänge `fullHeating` und `holdingTo` von der Steuerung
- Heatport für Wärme zum Ofen
- Ausgang P = aktuelle Gesamtleistung

Ofenmodell `Oven`

- HeatPort für Wärme
- Eingänge `doorOpen`, `heatingParts`, `restartTl`
- Ausgänge `Ploss`, `Pload` und `To` (Oven), `Tl` (Load)

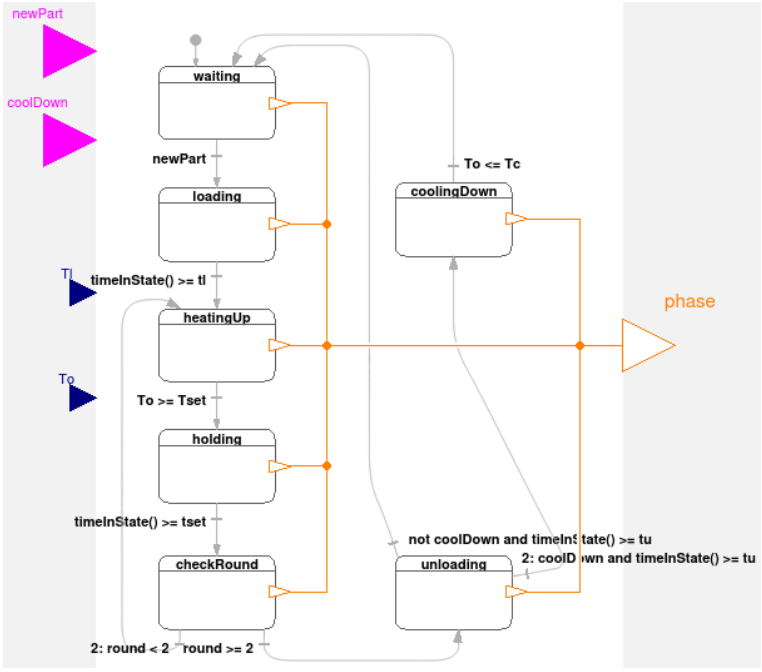
Ofen-Steuerung `OvenControlSM`

- Eingänge `newPart` und `coolDown`
- Eingänge `To` und `Tl`
- Ausgang `phase`

Wert	Zustand	Signale
0	Waiting	-
1	Loading	`doorOpen`
2	HeatingUp	`fullHeating`, `heatingParts`
3	Holding	`holdingTo`, `heatingParts`
4	Unloading	`doorOpen`, `restartTl`
5	CoolingDown	-

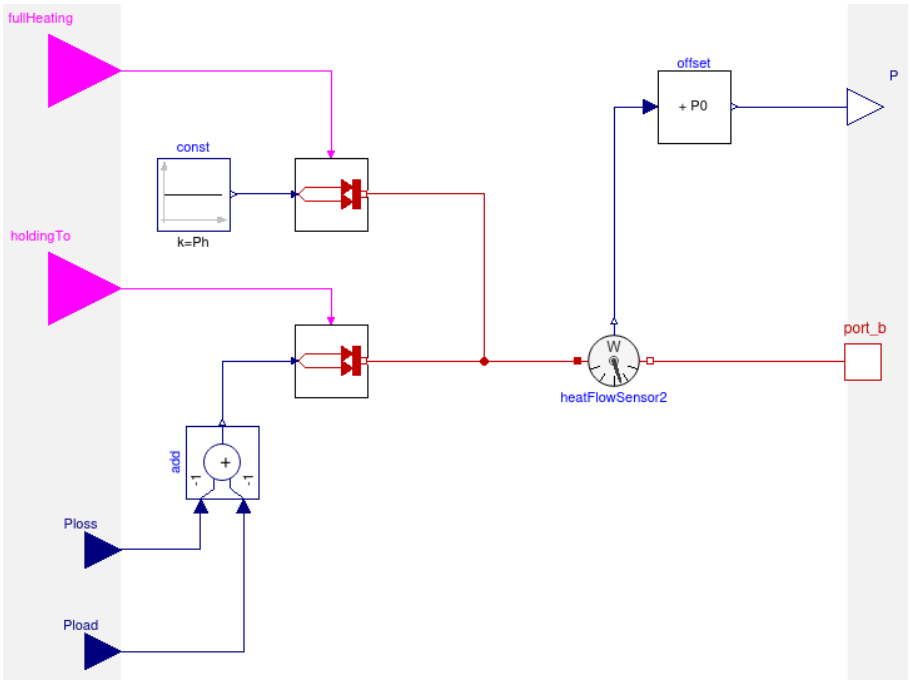
- Implementierung:

  `OvenControl`



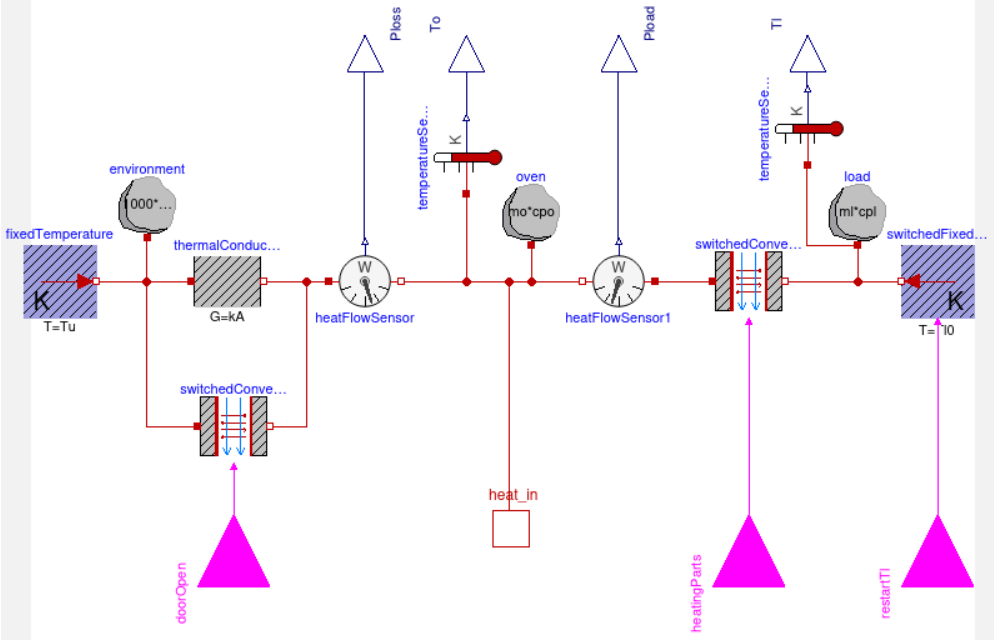
- verwendet T_o statt T_l (leichter zu messen)

`PowerSupply`



- verwendet neue Komponente `SwitchedHeatFlow`
- mögliche Erweiterung: T_o-Wert und Regler statt direkte Bestimmung der Wärmen in Holding-Phase

`Oven`



- verwendet neue Komponenten `SwitchedConvectionRadiation` und `SwitchedFixedTemperature`

Vorzeichen bei `Q_flow` beachten!

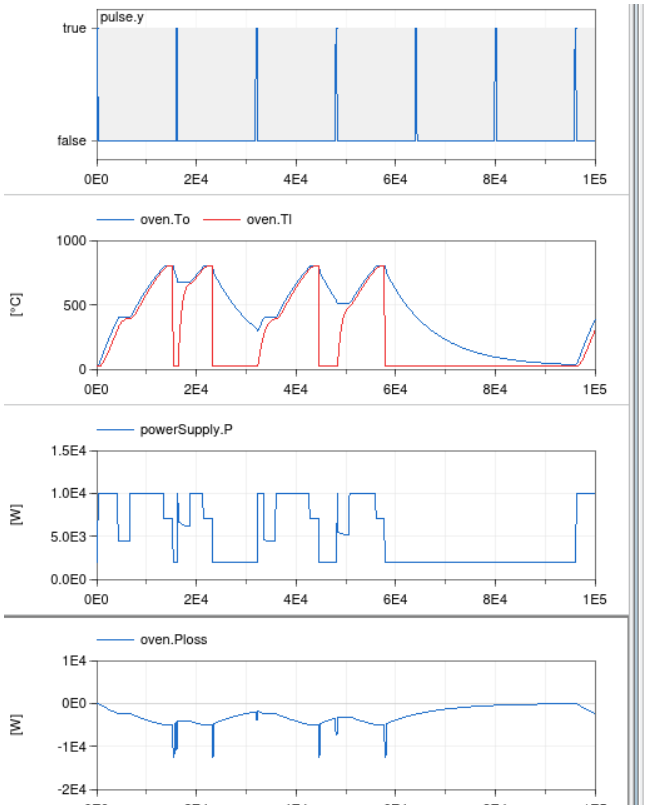
- `HeatFlowSensor` liefert positiven Wert für Fluss von `port_a` nach `port_b`
- Wärme vom Ofen weg soll negativ sein
- zugeführte Wärme bei Holding ist positiv, daher Vorzeichen in `PowerSupply` umdrehen

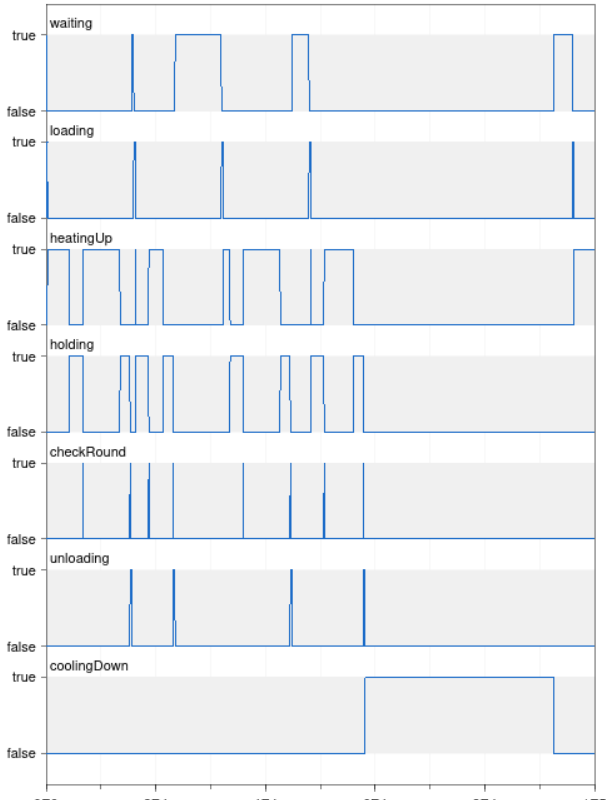
`P2C` enthält `Sample` und `Clock`

`C2P` enthält `Hold` und Umrechnung von `phase` zu Signalausgängen (s. Tabelle)

- Ergebnisse:

  Bearbeitung von 4 Werkstücken, dann Kühlen





- T_I folgt T_o gut
- Aufheizphase des Ofens am Anfang und bei Wartezeit zwischendurch

mögliche Verbesserung: keine Wartezeiten zwischen den Werkstücken

- Aufgaben:

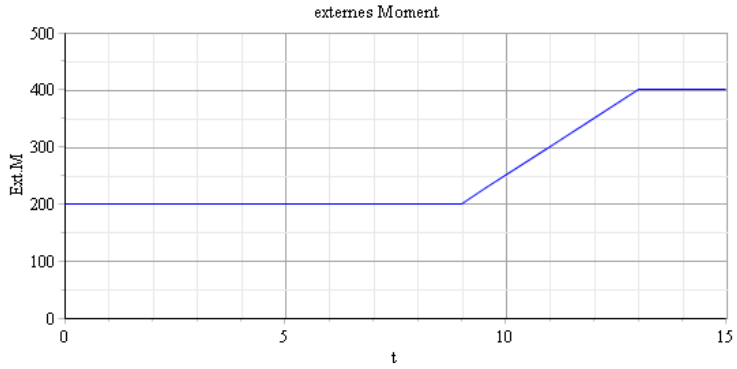
  Aufgabe 203

Aufgaben

Aufgabe 47

In [1] wird ein Modell für die Funktion einer Kupplung beschrieben, das aus einer rotatorischen Coulomb-Reibstelle und zwei Trägheiten mit unterschiedlichen Anfangsdrehzahlen besteht. Erstellen Sie ein entsprechendes MapleSim-Modell mit folgenden Bausteinen:

- Coulomb-Reibstelle
  - Gleitreibungsmoment Mg = 190 Nm
  - Haftreibungsmoment Mh = 250 Nm
- motorseitiges Trägheitsmoment
  - $J_1 = 1\ kg\ m^2$
  - Anfangsdrehgeschwindigkeit $\omega_1(0) = 200\ 1/s$
- abtriebseitiges Trägheitsmoment
  - $J_2 = 5\ kg\ m^2$
  - Anfangsdrehgeschwindigkeit $\omega_2(0) = 0$
- zeitlich veränderliches Antriebsmoment



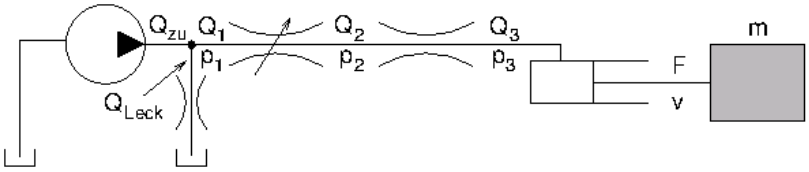
Aufgabe 48

a. Vereinfachen Sie das Modell `heattransfer2`, indem Sie ein geeignetes Submodell einführen. Verlängern Sie die Kette und vergleichen Sie die Ergebnisse von Mitstrom- und Gegenstromkühlung.

b. Erstellen Sie ein analoges Modell mithilfe von Elementen aus der `FluidHeatFlow`-Bibliothek.

Aufgabe 49

- Erweitern Sie das Hydraulik-Modell `hydraulik4` um ein Drosselventil:



- Der Durchfluss wird beschrieben durch

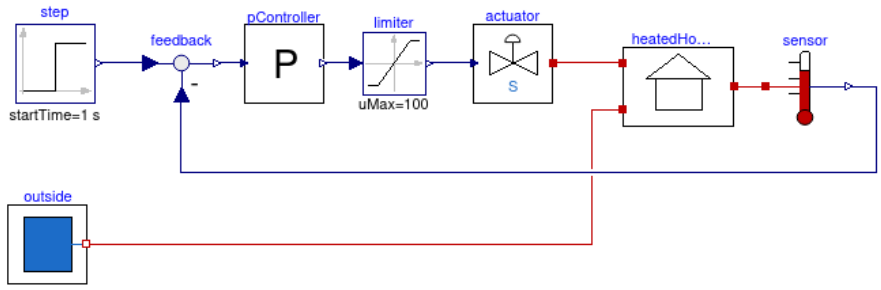
$$
\begin{aligned}
Q_1 &= Q_2 \\
Q_1 &= \alpha A \operatorname{sign}(p_1 - p_2) \sqrt{\frac{2}{\rho} |p_1 - p_2|}
\end{aligned}
$$

  Dabei ist A die (veränderliche) Fläche des Ventils und ρ die Dichte des Fluids. Die Durchflussziffer α beschreibt pauschal kompliziertere Strömungsphänomene, die Signums- und Betragsfunktion sorgen dafür, dass Flüsse in beide Richtungen korrekt behandelt werden.

- Simulieren Sie das Verhalten des Systems, wenn die Fläche von nahezu 0 auf den Maximalwert anwächst.

Aufgabe 200

- Erstellen Sie ein Modell zur Regelung der Temperatur eines Hauses. Das Hausmodell (die Regelstrecke) enthält die Heizleistung als Stellgröße und die Außentemperatur als Störgröße und basiert auf der Thermal-Bibliothek. Fügen Sie außerdem einen einfachen Zweipunktregler hinzu, der bei positiver Regeldifferenz einen festen Ausgangswert ausgibt, sonst den Wert 0. Untersuchen Sie in einem Regelkreis das Aufheizen des Hauses sowie das Verhalten bei sinkender Außentemperatur T_a. (Hinweis: Die Lösung von Aufgabe 13 enthält komplette Modelle incl. der benötigten Parameter - allerdings in Simulink.)
- Ersetzen Sie den Zweipunktregler durch einen P-Regler, dessen Regelungsspannung auf ±100 V begrenzt wird. Das Modell soll nun um Modelle für die Temperaturmessung (Sensor) und das Stellglied (Aktuator) erweitert werden:



  - Der Sensor zur Temperaturmessung erzeugt aus der Temperatur T_{in} eine Ausgangsspannung x. Aufgrund seiner eigenen Wärmekapazität C_S und des Wärmewiderstands R_S verhält er sich wie eine Verzögerung (PT1-Glied).
  - Mit der Reglerspannung u wird ein Elektromotor betrieben, dessen Winkelgeschwindigkeit proportional zur Spannung ist:

    $$\omega = K_M u = \dot{\varphi}$$

    Entsprechend der Winkelstellung, die immer zwischen 0° und 90° liegt, klappt ein Ventil auf und zu. Vereinfacht wird angenommen, dass die zugeführte Heizleistung proportional zum Sinus des Winkels ist:

    $$\dot{Q}_H = K_V \sin\varphi$$

  - Erweitern Sie das Modell um entsprechende Sensor- und Aktuator-Komponenten und untersuchen Sie, wie sich das Verhalten des Systems dadurch ändert.

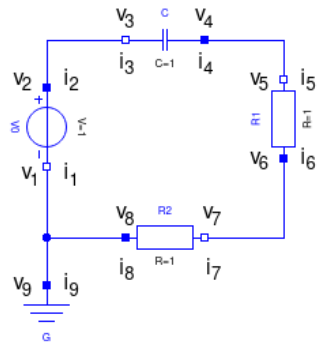
    Werte:
    - P = 20
    - C_S = 40 J/K, R_S = 0.05 K/W
    - K_M = 0.3/(V s), K_V = 18500 W
- Lösung

Aufgabe 50

- Ersetzen Sie im Feder-Masse-Beispiel die Feder durch zwei parallele Federn. Stellen Sie das zugehörige Gleichungssystem auf und machen Sie mit dem Tarjan-Algorithmus daraus eine Folge von Zuweisungen. Führen Sie nun alle Unbekannten direkt auf die Zustandsgrößen zurück und bestimmen somit schließlich die Bewegungsgleichung explizit.

Aufgabe 51

- Stellen Sie für eine Reihenschaltung aus Spannungsquelle, Kondensator und zwei Widerständen



  das Gleichungssystem auf und erzeugen Sie mit dem Tarjan- Algorithmus daraus eine Zuweisungskette. Vereinfachen Sie die dabei auftretende algebraische Schleife mit dem Tearing-Verfahren und bestimmen Sie die Bewegungsgleichung.

- Hinweis: Verwenden Sie als einzige interne Bauteilvariable die Spannungsdifferenz v_c am Kondensator, alle anderen Beziehungen drücken Sie direkt mit den Anschlussgrößen aus.

Aufgabe 52

- Ersetzen Sie im Feder-Masse-Beispiel die Masse durch zwei hintereinander gekoppelte Massen. Verwenden Sie den Pantelides-Algorithmus, um den Index so lange zu reduzieren, bis das System nicht mehr strukturell singulär ist. Führen Sie dann geeignete Dummy-Ableitungen ein und verwandeln Sie das entsprechende System in eine Folge von Zuweisungen. Bestimmen Sie schließlich explizit die Bewegungsgleichungen.

Aufgabe 201

- Vereinfachen Sie die Gleichungen der drei Beispielmodelle (Feder-Masse, Feder-Feder-Masse, Feder-Dämpfer) durch Ersetzen von Aliasvariablen. Führen Sie anschließend die graphischen Reduktionsmethoden durch und bestimmen Sie die sich ergebenden Bewegungsgleichungen.
- Lösung

Aufgabe 54

- Erstellen Sie Eingangssignale, die folgendes Fahrmanöver nachbilden: Anfahren im 1. Gang, anschließend Vollgas, zur Zeit t = 6 s Hochschalten in den 2. Gang. Untersuchen Sie die dabei auftretenden Probleme im `Triebstrang3`-Modell und beseitigen Sie diese.
- Lösung

Aufgabe 57

- Erweitern Sie das Modell `TriebstrangR` um eine Messung des Benzinverbrauchs für einen Beschleunigungsvorgang auf v_{max} = 60 m/s. Machen Sie dazu den einfachen Ansatz, dass der Momentanverbrauch proportional zur Fahrpedalstellung ist. Untersuchen Sie, wie sich der Verbrauch durch Ändern ausgewählter Parameter verringern lässt.

Aufgabe 202

- Erstellen Sie ein Triebstrang-Modell, das zwei permanentmagnet-erregte Synchronmotoren zum Antrieb der beiden Hinterräder verwendet. Modellieren Sie damit die Beschleunigung auf Höchstgeschwindigkeit.
- Verwenden Sie als Antrieb den Motor EMRAX 208 Low Voltage (LV) der Firma EMRAX (Datenblatt). Die entscheidende Seite enthält alle benötigten Kennzahlen.
- Lösung

Aufgabe 58

- Erstellen Sie mithilfe der MultiBody-Bibliothek ein einfaches Modell für die Laufkatze aus [L1, K6.1] und vergleichen Sie das Ergebnis der Simulation mit dem dortigen Ergebnis.

Aufgabe 60

- Ein SCARA-Roboter besteht aus einem horizontal angeordneten Zweiachsen-System, an dessen Ende eine weitere Achse vertikale Bewegungen ermöglicht. Erstellen Sie ein entsprechendes Modell und erweitern Sie es um eine detailliertere Beschreibung der Servomotoren sowie eine Positionierungsregelung. Orientieren Sie sich für Details und Parameterwerte am ARGESIM-Benchmark C11.
- Achtung: Lassen Sie die Strombegrenzung für die Servomotoren, die in der Beschreibung des C11-Systems vorgesehen ist, weg! Wenn Sie sie doch implementieren wollen, stellen Sie zunächst die Strom-Spannungs-Kennlinie einer idealen Strombegrenzung auf. Lesen Sie dann [2] oder Kap. 9 von [L5] und programmieren Sie mit der dort vorgestellten Parameter-Methode die Strombegrenzung direkt in Modelica.

Aufgabe 61

- Erstellen Sie ein realistischeres Modell, indem Sie berücksichtigen, dass ein Rad nur bei Stauchung als Feder betrachtet werden kann, aber nicht bei Streckung - es klebt ja nicht an der Straße! Wie wirkt sich diese Änderung auf das Schwingverhalten aus?

Aufgabe 62

- Erweitern Sie das Trébuchet-Modell so, dass sich die Schlinge nicht zu einem vorgegebenen Zeitpunkt löst, sondern sobald sie einen bestimmten Winkel gegen die Horizontale erreicht. Sorgen Sie außerdem dafür, dass die Simulation stoppt, wenn das Geschoss auf dem Boden aufschlägt.

Aufgabe 63

- Erstellen Sie ein Modell für den Dieselprozess, der sich vom Ottoprozess darin unterscheidet, dass die Wärmezufuhr nicht isochor, sondern isobar geschieht.

Aufgabe 64

- Ein Joule-Prozess startet bei p1 = 1.2 bar und T1 = 320 K, die Pumpe erhöht den Druck auf 7 bar, der Erhitzer die Temperatur danach auf 800 K. Hinter der Turbine herrscht wieder der Anfangsdruck. Wie groß müssen die Parameter τ der Pumpe, $\dot{Q}$ des Erhitzers und R des Verbrauchers in der Turbine sein, um diese Werte zu erhalten?
- Lösen Sie die Aufgabe mit Thermodynamik-Grundkenntnissen - oder notfalls durch Experimentieren mit dem Modell.

Aufgabe 65

- Beim Ericsson-Prozess werden die Vorgänge in Pumpe und Turbine nicht durch Isentropen, sondern durch Isothermen beschrieben. Erstellen Sie ein entsprechendes Modell, indem Sie

  a. eine Komponente für eine isotherme Turbomaschine erstellen,

  b. stattdessen mit Zwischenkühlern und -erhitzern die sich bei Kompression bzw. Expansion ändernde Temperatur ausgleichen.

Aufgabe 203

- Erstellen Sie ein Modell `Triebstrang7`, das das Beschleunigen von `Triebstrang6C` mit dem Reifenmodell von `TriebstrangR` kombiniert. Ersetzen Sie dabei das Fahrermodell durch ein Modell, das auf einer State Machine basiert und das Verhalten des alten Modells reproduziert.
- Lösung

Aufgabe 33

- Erweitern Sie das Modell `Bevoelkerung3` durch einen Term, der die Jagd auf die entsprechenden Tiere beschreibt, wobei die Zahl der pro Zeit erlegten Tiere konstant sein soll. Wie groß kann die Beute maximal werden, ohne dass die Population zusammenbricht? Hängt der Wert vom Anfangsbestand ab? Können Sie Ihr Ergebnis auch analytisch berechnen?

Aufgabe 35

- Ersetzen Sie die Stocks und Flows des Modells `Raeuberbeute2B` durch diskrete Versionen, so dass die Größen der Räuber- und Beute-Populationen ganzzahlig sind. Geeignete Blöcke finden Sie in der `SystemDynamics`-Bibliothek, spätestens nach einem kurzen Blick auf den Quellcode. Vergleichen Sie die Ergebnisse mit denen der kontinuierlichen Version. Wo liegen Gleichgewichtszustände?

Aufgabe 36

- Forrester untersuchte in [13], wie sich Änderungen einzelner Parameter, jeweils erst ab dem Jahr 1970, auf die Simulationsergebnisse auswirken, u. a.
  - Die Verbrauchsrate natürlicher Ressourcen NRUN sinkt von 1 auf 0.25.
  - Die Geburtenrate BRN sinkt von 0.040 auf 0.028.
  - NRUN sinkt auf 0.25, gleichzeitig sinkt die Verschmutzungsrate POLN von 1 auf 0.7.
- Erstellen Sie entsprechend geänderte Modelle - mithilfe des `TimeSwitchedConverter`-Blocks - und untersuchen Sie deren Ergebnisse.

Aufgabe 37

- Beide Maschinen der Fertigungsstraße sollen von einer einzigen Person betreut werden. Dazu wird Maschine 2 so umgebaut, dass sie ebenfalls vier Teile in zwei Zeiteinheiten bearbeitet. Der Betreuer halte sich bei Maschine 1 auf, wenn sie gerade leer ist, sonst bei Maschine 2.
- Erweitern Sie das Modelica-Modell um den Betreuer und vergleichen Sie das Ergebnis. Kann man mit dem Modell den theoretisch möglichen Durchsatz erreichen?

Anhang

Literatur

1. P. Junglas: Praxis der Simulationstechnik
   Europa-Lehrmittel-Verlag 2014, 623 S.

2. P. Fritzson: Introduction to modeling and simulation of technical and physical systems with Modelica
   Wiley 2011, 211 S.

3. P. Fritzson: Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach
   Wiley 2015, 1223 S.

4. F. E. Cellier: Continuous System Modeling
   Springer 1991, 755 S.

5. F. E. Cellier and E. Kofman: Continuous System Simulation
   Springer 2006, 643 S.

6. Modelica Association: Modelica - A Unified Object-Oriented Language for Systems Modeling - Language Specification
   Online

7. C. Kral: Modelica - Objektorientierte Modellbildung von Drehfeldmaschinen
   Fachbuchverlag Leipzig 2019, 347 S.

8. H. Bossel: Modellbildung und Simulation
   Vieweg 1992, 400 S.

9. B. Hannon, M. Ruth: Dynamic Modeling
   Springer, 2. Aufl 2001, 410 S.

Nachweise

Text

1. H. Scherf: Modellbildung und Simulation dynamischer Systeme, Oldenbourg, 4. Aufl. 2010.
2. M. Otter, H. Elmqvist und S. E. Mattsson: The New Modelica MultiBody Library. In: Proc. 3rd Int. Modelica Conf., Linköping, Sweden, 2003, S. 311–330.
3. M. Gipser: Systemdynamik und Simulation. Teubner, Stuttgart, Leipzig (1999) (online).
4. D. Ammon: Modellbildung und Systementwicklung in der Fahrzeugdynamik. Vieweg+Teubner, Stuttgart (2013).
5. R. Busch: Elektrotechnik und Elektronik. Springer, 7. Aufl. (2015).
6. S. Rupp: Modellierung von Anlagen und Systemen, Teil 2. Vorlesungsskript DHBW CAS (2019), (online).
7. C. Kral: Modelica - Objektorientierte Modellbildung von Drehfeldmaschinen. Carl Hanser (2019).
8. W. Leonhard: Regelung elektrischer Antriebe. Springer, 2. Aufl. (2000).
9. D. Schröder: Elektrische Antriebe - Regelung von Antriebssystemen. Springer, 4. Auflage (2015).
10. A. Körner, F. Breitenecker: State Events and Structural-dynamic Systems: Definition of ARGESIM Benchmark C21. SNE Simulation News Europe. 2016; 26(2):117–122.
11. J. P. Disselkamp, P. Junglas, A. Niehüser, P. A. Schönfelder: Solution to ARGESIM Benchmark C21 'State Events and Structural-dynamic Systems' based on Modelica Components. SNE Simulation News Europe. 2018; 28(2):39–48.
12. D. Harel: Statecharts: A visual formalism for complex systems. Science of Computer Programming. 1987; 8, 231-274.
13. J. W. Forrester: World Dynamics. Wright-Allen Press, 2. Aufl. 1973.

Modelle

- Bibliotheken:
  - SimT2Lib.mo
  - SystemDynamics.mo
  - SystemDynamicsExamples_DE.mo
- Hilfsdateien:
  - motorkennfeld.txt
  - gaspedal.txt
  - kupplung.txt
  - schaltgetriebe.txt
  - icon-karosserie.png
  - oven.png