Basics of Simulink

TUM Graduate School Training

Dipl.-Ing. Markus Hornauer
Outline
Simulink and Stateflow

Basics:
1) Simulink
   • Basics
   • Continuous Models
   • Discrete Models
   • Subsystems
   • Signals
2) Stateflow
   • Flow Charts
   • State Charts
   • Events

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
Your Expectations?
Introduction to Simulink, Stateflow and Code Generation

References to the book MATLAB – Simulink – Stateflow (Angermann, Beuschel, Rau, Wohlfarth, Oldenburg Verlag)  
- Supported by MathWorks -
Outline
Simulink and Stateflow

Basics:
1) Simulink
   • Basics
   • Continuous Models
   • Discrete Models
   • Subsystems
   • Signals
2) Stateflow
   • Flow Charts
   • State Charts
   • Events

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
Key Features

- Graphical editor for building and managing hierarchical block diagrams
- Libraries of predefined blocks for modeling continuous-time and discrete-time systems
- Simulation engine with fixed-step and variable-step ODE solvers for discrete and continuous time modelling
- Scopes and data displays for viewing simulation results
- Project and data management tools for managing model files and data
- Model analysis tools for refining model architecture and increasing simulation speed
- MATLAB Function block for importing MATLAB algorithms into models
- Legacy Code Tool for importing C and C++ code into models
- Automatic code generation capabilities for C, C++, Structured Text and HDL
- Multi domain modelling using signal flow diagrams, state machines and physical modelling
- Capabilities to directly interact with hardware and real time systems
Introduction

Application Examples

- Plant modelling
  - Modelling of nonlinear dynamic systems (continuous-time, discrete-time, hybrid)
  - Analyses of dynamic systems (pre-development)
  - Optimization of dynamic systems (system design)

- Design of embedded systems
  - Model-based software development
  - Automatic code generation (software and programmable hardware)

- Model-based testing
  - Open- and closed-loop testing of plant model and control software
  - Formal methods for software verification
  - Hardware in the loop testing

www.mathworks.com/company/user_stories/
Launching Simulink

Starting Simulink

```
>> simulink
```

MATLAB R2013a window with Simulink library open.
Finding Blocks

The image shows the Simulink Library Browser with the library search bar open. The term 'cos' has been entered, and the cosine block from the Simulink library is highlighted. The browser also displays matches from other libraries, such as the Aerospace Blockset and the Communications System Toolbox.

- **Simulink**
  - Cosine block

- **Aerospace Blockset**
  - Various blocks related to direction cosine, matrix, and quaternion operations

- **Communications System Toolbox**
  - Raised Cosine blocks

- **HDL Verifier**
  - HDL Cosimulation blocks

The interface is designed to help users quickly find and select blocks for their models.
Simulink – Basics

Getting Help

1. Using Simulink menu
2. Function Block Parameters: Gain dialog box
3. Configuration Parameters dialog box

Basics of Simulink
Demonstration of Model Elements
Adding Blocks:
- Drag and drop a block from the Simulink library into the block diagram
- Copy a block inside the block diagram by dragging it while holding the right mouse key
- Click into the block diagram and start to enter the name of the block (R14b)

Connecting Blocks:
- Draw a line from the outport of one block to the inport of a second block using the left mouse key
- To connect a block to a line, draw a line from the inport of the block backwards to the line to connect to.
- To quick connect two blocks, click on the outport of the first block and the inport of the second block while keeping the CRTL-key pressed
- Click the suggested connection lines (R14b)
Simulation Data Inspector

- Simulation Data Inspector
- Stream Selected Signals to Data Inspector
- Log Selected Signals to Workspace
- Send Logged Workspace Data to Data Inspector
- Configure Logging and Streaming...
- Help on Simulation Data Inspector...

Graph showing simulation data with multiple curves and properties table.
Outline
Simulink and Stateflow

Basics:
1) Simulink
   • Basics
   • Continuous Models
   • Discrete Models
   • Subsystems
   • Signals
2) Stateflow
   • Flow Charts
   • State Charts
   • Events

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
Basics of Simulink

Modeling Continuous Systems

- Engine provides variable-step and fixed-step ODE solvers
- Block Diagram representation of dynamic systems
- Blocks define governing equations
- Signals are propagated between blocks over time

- Remember MuPad: \[ m\ddot{x} + b\dot{x} + kx = 0, \]
  \[ x(0) = 0, \dot{x}(0) = 1, m = 10, b = 1, k = 10 \]

- Exercise: Create a mass-spring-damper system in Simulink
Error because of Direct Feedthrough
- Block input depends directly on its own output
- E.g.: Gain, Product, Sum/Subtract, Transfer Fcn

Recommended solution: use Delay, Integrator or other history related block

Alternative, but bad solution: reduce diagnostics settings and leave solving up to Simulink engine (not recommended!!)
Simulink – Continuous Systems

Solver

- **Solver?**
  - Determines solution at current time step
  - Determines the next simulation time step

- **Solver options:**

  **Fixed-Step**
  - Ode1
  - Ode2
  - Ode3
  - Ode4
  - Ode5
  - Ode8

  **Variable-Step**
  - Ode45
  - Ode23
  - Ode113
  - Ode15s
  - Ode23s
  - Ode23t
  - Ode23tb
Fixed-step Solvers.

**Default**: `ode3 (Bogacki-Shampine)`

- Computes the model's state at the next time step as an explicit function of the current value of the state and the state derivatives, using the Bogacki-Shampine Formula integration technique to compute the state derivatives. In the following example, \( X \) is the state, \( DX \) is the state derivative, and \( h \) is the step size:

\[
X(n+1) = X(n) + h \times DX(n)
\]

**Discrete (no continuous states)**

- Computes the time of the next time step by adding a fixed step size to the current time.

Use this solver for models with no states or discrete states only, using a fixed step size. Relies on the model’s blocks to update discrete states.

The accuracy and length of time of the resulting simulation depends on the size of the steps taken by the simulation: the smaller the step size, the more accurate the results but the longer the simulation takes.

**Note**: The fixed-step discrete solver cannot be used to simulate models that have continuous states.

- `ode8 (Dormand-Prince RK8(7))`:
  Uses the eighth-order Dormand-Prince formula to compute the model state at the next time step as an explicit function of the current value of the state and the state derivatives approximated at intermediate points.

- `ode5 (Dormand-Prince)`:
  Uses the fifth-order Dormand-Prince formula to compute the model state at the next time step as an explicit function of the current value of the state and the state derivatives approximated at intermediate points.

- `ode4 (Runge-Kutta)`:
  Uses the fourth-order Runge-Kutta (RK4) formula to compute the model state at the next time step as an explicit function of the current value of the state and the state derivatives.

- `ode2 (Heun)`:
  Uses the Heun integration method to compute the model state at the next time step as an explicit function of the current value of the state and the state derivatives.

- `ode1 (Euler)`:
  Uses the Euler integration method to compute the model state at the next time step as an explicit function of the current value of the state and the state derivatives.

- `ode14x (extrapolation)`:
  Uses a combination of Newton’s method and extrapolation from the current value to compute the model’s state at the next time step, as an implicit function of the state and the state derivative at the next time step. In the following example, \( X \) is the state, \( DX \) is the state derivative, and \( h \) is the step size:

\[
X(n+1) = X(n) - h \times DX(n+1) = 0
\]

This solver requires more computation per step than an explicit solver, but is more accurate for a given step size.
Variable-step Solvers.

Default: ode45 (Dormand-Prince)

ode45 (Dormand-Prince)
Computes the model’s state at the next time step using an explicit Runge-Kutta (4,5) formula (the Dormand-Prince pair) for numerical integration.

ode45 is a one-step solver, and therefore only needs the solution at the preceding time point.

Use ode45 as a first try for most problems.

Discrete (no continuous states)
Computes the time of the next step by adding a step size that varies depending on the rate of change of the model’s states.

Use this solver for models with no states or discrete states only, using a variable step size.

ode23 (Bogacki-Shampine)
Computes the model’s state at the next time step using an explicit Runge-Kutta (2,3) formula (the Bogacki-Shampine pair) for numerical integration.

ode23 is a one-step solver, and therefore only needs the solution at the preceding time point.

ode23 is more efficient than ode45 at crude tolerances and in the presence of mild stiffness.

ode113 (Adams)
Computes the model’s state at the next time step using a variable-order Adams-Bashforth-Moulton PECE numerical integration technique.

ode113 is a multistep solver, and thus generally needs the solutions at several preceding time points to compute the current solution.

ode113 can be more efficient than ode45 at stringent tolerances.

ode15s (stiff/MD)
Computes the model’s state at the next time step using variable-order numerical differentiation formulas (NDFs). These are related to, but more efficient than the backward differentiation formulas (BDFs), also known as Gear’s method.

ode15s is a multistep solver, and thus generally needs the solutions at several preceding time points to compute the current solution.

ode15s is efficient for stiff problems. Try this solver if ode45 fails or is inefficient.

ode23s (stiff/Mod. Rosenbrock)
Computes the model’s state at the next time step using a modified Rosenbrock formula of order 2.

ode23s is a one-step solver, and therefore only needs the solution at the preceding time point.

ode23s is more efficient than ode15s at crude tolerances, and can solve stiff problems for which ode15s is ineffective.

ode23t (Mod. stiff/Trapezoidal)
Computes the model’s state at the next time step using an implementation of the trapezoidal rule with a “free” interpolant.

ode23t is a one-step solver, and therefore only needs the solution at the preceding time point.

Use ode23t if the problem is only moderately stiff and you need a solution with no numerical damping.

ode23tb (stiff/TR-BDF2)
Computes the model’s state at the next time step using a multistep implementation of TR-BDF2, an implicit Runge-Kutta formula with a trapezoidal rule first stage, and a second stage consisting of a backward differentiation formula of order two. By construction, the same iteration matrix is used in evaluating both stages.

ode23tb is more efficient than ode15s at crude tolerances, and can solve stiff problems for which ode15s is ineffective.
Max step size is the largest time step that the solver can take (‘auto’ means 50 steps)
Simulink – Continuous Systems
Demo: Importing and Exporting Data
Importing Data into Simulink

**IN**
- Requires vector of time along with input values
  \[ \text{input}(t, u_1 \ldots u_n) \]
  defined configuration parameters

**CONSTANT**
- Changeable on the highest hierarchy level
- Tunable with parameter objects

**FROM WORKSPACE**
- Requires vector of time along with input values
  \[ \text{input}(t, u_1 \ldots u_n) \]
  defined a workspace variable

**FROM FILE**
- Requires vector of time along with input values
  \[ \text{input}(t, u_1 \ldots u_n) \]
  defined in a given mat file
OUT
• Saves all outputs together in one variable, defined in the configuration parameters

TO WORKSPACE
• Saves the output data in a variable to the workspace, defined in the block parameters

TO FILE
• Saves the output data in a .mat file
For IN and OUT blocks variables must be defined in Configuration Parameters.
MATLAB Embedded

- Subset of MATLAB for code generation
- Can be used for direct generation of source code out of MATLAB as well as in Simulink MATLAB Function blocks
- Enables user to reuse his MATLAB code in Simulink
- To call unsupported functions use `eml.extrinsic` or `coder.extrinsic` (leads to significantly reduced performance!!!)
Simulink – Continuous Systems
Model and Block Callbacks
Common tasks you can achieve by using callback functions include:

- Loading variables into the MATLAB workspace automatically when you open your Simulink model
- Executing a MATLAB script by double-clicking on a block
- Executing a series of commands before starting a simulation
- Executing commands when a block diagram is closed
Outline
Simulink and Stateflow

Basics:
1) Simulink
   • Basics
   • Continuous Models
   • Discrete Models
   • Subsystems
   • Signals
2) Stateflow
   • Flow Charts
   • State Charts
   • Events

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
Exercise: Create a Stop Watch

- Combine counters to a stop watch
- Show tenth seconds, seconds, minutes and hours
- Reduce simulation speed to soft realtime
Systems with signals that are sampled at different rates

Use for discrete or hybrid systems

To connect systems use rate transition blocks

Specify specific sampling rate by variable at each in and out port

Different sample times need to be an integer multiple of the highest (global) sampling rate

Sample Time Colors

-> fastest discrete sampling time is displayed in red
Outline
Simulink and Stateflow

**Basics:**
1) Simulink
   - Basics
   - Continuous Models
   - Discrete Models
   - **Subsystems**
   - Signals
2) Stateflow
   - Flow Charts
   - State Charts
   - Events

**Advanced:**
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
• Why?
  – Reduce blocks displayed in a model window
  – Keep functionally related block together
  – Establish hierarchical block diagram
Creating Subsystems

- Context menu -> Create Subsystem
- Subsystem ports
- Inside a subsystem
Atomic Subsystems

- Represent non-virtual systems within another system
- Have their own sampling rate
- Have their own code generating characteristics
- Have their own execution order number
Masking Subsystems

- Mask - Encapsulation with a UI
- Provides
  - Mask icon display
  - Block description
  - Parameter dialog prompt
  - Custom block help text
Outline
Simulink and Stateflow

Basics:
1) Simulink
2) Stateflow

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
User Libraries

- Collection of reusable blocks
- Prototype block vs Reference block
- Propagation of changes to Library
  - Discard
  - Push
- Library Links
  - Disable link
  - Restore link
  - Break link
- Other features
  - Display in Simulink Library Browser
  - Add documentation
Model Referencing

- One model in another- *parent and referenced model*

- Advantages:
  - Componentization/Modularization
  - IP protection
  - Multiple referencing
  - Acceleration
Outline
Simulink and Stateflow

Basics:
1) Simulink
   • Basics
   • Continuous Models
   • Discrete Models
   • Subsystems
   • Signals
2) Stateflow
   • Flow Charts
   • State Charts
   • Events

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
Vectors

- Matrix and Vector operations possible
- Mux block to compose vector
- Demux block to extract from signal
- Increase simulation performance
Simulink – Signals
Busses

• Graphical grouping of signals to a hierarchical bus signal
• Bus creators to create a bus from signal and busses
• Bus selector to select single signals or whole sub-busses
• Bus Objects can be specified
Simulink – Signals
Simulink Data Objects

• Simulink Parameter Object
  
  \[ \text{>> Var = Simulink.Parameter} \]

• Simulink Signal Object
  
  \[ \text{>> Var = Simulink.Signal} \]

• Simulink Bus Object
  
  – Use Bus editor
Simulink – Signals

Signal Logging

Configuration Parameters

Signal Context Menu
Outline
Simulink and Stateflow

Basics:
1) Simulink
   • Basics
   • Continuous Models
   • Discrete Models
   • Subsystems
   • Signals
2) Stateflow
   • Flow Charts
   • State Charts
   • Events

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
Stateflow

What is Stateflow?

- Stateflow is a blockset for Simulink
- Stateflow extends the signal flow paradigm of Simulink with state machines
- Stateflow supports flow charts and state charts
- Charts can be implemented using C or MATLAB as action language
- Stateflow supports Mealy and Moore charts
- Events are available for asynchronous communication
- Stateflow is fully integrated with Simulink
- Stateflow supports embedded code generation
na_0006: Guidelines for mixed use of Simulink and Stateflow

- If the function primarily involves complicated logical operations, use Stateflow diagrams. Use Stateflow diagrams to implement modal logic, where the control function to be performed at the current time depends on a combination of past and present logical conditions.
- If the function primarily involves numerical operations, use Simulink features.

na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines

- If the primary nature of the function segment is to calculate modes of operation or discrete-valued states, use state charts. Some examples are:- Diagnostic models with pass, fail, abort, and conflict states- Model that calculates different modes of operation for a control algorithm
- If the primary nature of the function segment involves if-then-else statements, use flowcharts or truth tables.
Outline
Simulink and Stateflow

Basics:
1) Simulink
   • Basics
   • Continuous Models
   • Discrete Models
   • Subsystems
   • Signals
2) Stateflow
   • Flow Charts
   • State Charts
   • Events

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
Flow Graphs have no action or information in state. Everything is done on transitions.

First condition \([a > b]\), then action \(\{c = 0;\}\)

Demo: A soda machine provides coke, orange juice and water. The user enters the corresponding number. The machine puts out a can with the drink.
Stateflow – Flow Charts

Hints

• First Condition, then Action!

• Double Click keeps buttons pressed
Defining Chart Data
Guidelines for Creating Flow Charts

- The execution has only one entry point!
- The execution has only one termination point!
- The execution can always reach the termination point!
- The flow never backtracks!
Outline
Simulink and Stateflow

Basics:
1) Simulink
   • Basics
   • Continuous Models
   • Discrete Models
   • Subsystems
   • Signals
2) Stateflow
   • Flow Charts
   • State Charts
   • Events

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
State Charts

- State charts have an internal behavior and internal data.
- Actions can be performed on entry, during residence in the state, and on exit.
- A state can perform a self transition.
- A state can be either active or passive.

```
Normal State
/*Comment*/
entry: a=b;
during: count++;
exit: count = 1;
```
Mealy Charts, Moore Charts and Stateflow

- Mealy charts perform actions on transition
- Moore charts perform actions in states
- Using the Model Explorer, state charts can be configured to Mealy, Moore or Classic
- See sf_seqrec for example
- Choosing Mealy, Moore or Classic as chart type effects compatibility of other MathWorks tools (e.g. Simulink Code Inspector)

MATLAB help -> Stateflow -> Chart Programming -> Supported State Machines -> Concepts
Stateflow – State Charts

Action Language MATLAB vs. C

- Stateflow supports MATLAB and C as action language (selected via Model Explorer)

- MATLAB as action language supports auto correction

- For embedded code generation, C as action language is easier to review

MATLAB Help -> Stateflow -> Chart Programming -> Chart Programming Basics -> Concepts -> Differences Between MATLAB and C as Action Language Syntax

MATLAB Help -> Stateflow -> Chart Programming -> Chart Programming Basics -> Concepts -> Action Language Auto Correction
Stateflow – State Charts

Demo: Autopilot Mode Control
Execution Order

/*Comment*/
Event[Condition]{Condition Action}/Transition Action

true executes

false does not execute
Stateflow – State Charts

Parallel Charts and Hierarchical Charts
Exercise: Elevator

- States: Initial State, Stopped, Up, Down
- Doors may only open when elevator is stopped
- Inputs: Height, Request
- Outputs: Command, Doors
- Doors: close = false, open = true
- Command: up = 1, down = 2, stop = 3
- Height of each floor = 3
Outline
Simulink and Stateflow

Basics:
1) Simulink
   • Basics
   • Continuous Models
   • Discrete Models
   • Subsystems
   • Signals
2) Stateflow
   • Flow Charts
   • State Charts
   • Events

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
Stateflow – Events

Definition

• Events are used for asynchronous communication

• Events can be directed or broadcast

• Events in Stateflow can be defined as input, local or output using the Model Explorer

• Events interact with state charts (trigger actions in parallel states), Simulink Triggered Subsystems and Simulink Function-Call Subsystems

• Events can be used on transitions and within states
Stateflow – Events

Example: sf_car

Gear State

- First
- Second
- Third
- Fourth

Selection State

during: \[ \text{down_th, up_th} \] = \text{calc_th}(\text{gear, throttle});

Steady State

- [speed < down_th]
- [speed > up_th]

Downshifting

- after(TWAIT, tick)
- [speed <= down_th]
- {gear_state.DOWN}

Upshifting

- after(TWAIT, tick)
- [speed >= up_th]
- {gear_state.UP}

Simulink Function

\[ \text{[down_th, up_th]} = \text{calc_th}(\text{gear, throttle}) \]
Stateflow – Events

Direct and qualified event broadcast

Direct Broadcast

Qualified Broadcast

```
selection_state
during: [down_th, up_th] = calc_th(gear, throttle);
```

```
steady_state
[steady_state] = [speed < down_th, speed > up_th];
```

```
downshifting
[downshifting] = [speed <= down_th, speed >= up_th];
```

```
upshifting
after(TWAIT, tick) [speed <= down_th] /send(DOWN, gear_state) {gear_state.UP}
```

```
upshifting
after(TWAIT, tick) [speed >= up_th] {gear_state.UP}
```

Simulink Function

```
[down_th, up_th] = calc_th(gear, throttle);
```
Temporal Logic

Enables time-dependent logic based on event counts

Temporal logic operators:
- \( at(n, event) \): true at the nth trigger of event
- \( every(n, event) \): true at every nth trigger of event
- \( after(n, event) \): true after the nth trigger of event
- \( before(n, event) \): true before the nth trigger of event

Can be applied as an event or condition
- \( after(5, tick) \)
- \( [after(5, tick)] \)

MATLAB Help -> Stateflow -> Chart Programming -> Syntax for States and Transitions -> Control Chart Execution Using Temporal Logic
Stateflow
Functions and Keywords – Summary

State action keywords
- **entry / en** – Perform actions upon state entry
- **during / du** – Perform actions when staying in state
- **exit / ex** – Perform actions upon state exit
- **on** – Perform actions upon specified event
- **bind** – Bind events to a state

- *Note, you can combine entry, during, and exit actions with the syntax
  - **en, du:**

Temporal logic operators
- **at(n,event)** – true at the n\(^{th}\) trigger of event
- **every(n,event)** – true at every n\(^{th}\) trigger of event
- **after(n,event)** – true after the n\(^{th}\) trigger of event
- **before(n,event)** – true before the n\(^{th}\) trigger of event
- **temporalCount(event)** – returns n at the n\(^{th}\) trigger of event, otherwise returns 0

In these operators, you can also use the keyword **sec** in place of an event. This keyword makes these operators count elapsed simulation time instead of the number of events.

State detection
- **enter(state)** – Event occurs when the specified state is entered.
- **exit(state)** – Event occurs when the specified state is exited.
- **in(state)** – Returns true when state is active

Data change detection
- **change(data)** – Event occurs when the specified data is written.
- **hasChanged(data)** – True when changes have been made to data since the last time step
- **hasChangedFrom(data,x)** – True when changes have been made to data since the last time step, and last time step value was x
- **hasChangedTo(data,x)** – True when changes have been made to data since the last time step, and current time step value is x

Built-in temporal events
- **tick** – Event occurs whenever the Stateflow chart is updated.
- **wakeup** – Same as tick

Local state data
- **StateA.a** – Accesses local data a defined in state StateA from outside of StateA
- **StateA.e** – Broadcasts local event e defined in state StateA from outside of StateA

Event broadcast
- **StateA.e** – Qualified event broadcast
- **send(e,StateA)** – Directed event broadcast
- **e** – Unqualified event broadcast
Outline

Simulink and Stateflow

Basics:
1) Simulink
   • Basics
   • Continuous Models
   • Discrete Models
   • Subsystems
   • Signals
2) Stateflow Flow
   • Charts
   • State Charts
   • Events

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
Modeling Guidelines
Increasing Model Quality

Help => Simulink => Modeling Guidelines
Outline
Simulink and Stateflow

Basics:
1) Simulink
   • Basics
   • Continuous Models
   • Discrete Models
   • Subsystems
   • Signals
2) Stateflow
   • Flow Charts
   • State Charts
   • Events

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
Simulink Model Advisor

Model Advisor
- Model Advisor
- Model Advisor Dashboard
- Upgrade Advisor
- Performance Advisor
- Code Generation Advisor

Verify model complies with modeling guidelines.

Tips
To enable or disable a check, select or clear the check box next to the check name. To enable or disable all checks within a folder, right-click the folder and then click "Select All" or "Deselect All". To run checks, select a folder or check in the left pane. For a list of all possible actions, right-click an object in the left pane.

To show or hide a Product folder, select or clear "Show By Product Folder" in the Settings > Preferences dialog box. To show or hide a Task folder, select or clear "Show By Task Folder" in the Settings > Preferences dialog box.

Legend
- Not Run
- Passed
- Failed
- Warning

Running this check triggers an update diagram.
Outline
Simulink and Stateflow

Basics:
1) Simulink
   • Basics
   • Continuous Models
   • Discrete Models
   • Subsystems
   • Signals
2) Stateflow
   • Flow Charts
   • State Charts
   • Events

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
Automatic Report Generation
Compare XML Files

- Model Advisor
- Model Dependencies
- Compare Simulink XML Files
- Performance Tools
- Requirements
- Control Design
- Parameter Estimation...
- Response Optimization...
- Design Verifier
- Coverage
- Fixed Point Tool...

Gain

File: C:\Users\MHORN1\Documents\MATLAB\Exercises_TUM_GS_Training\Simulink\Input_Mass_Spring_Damper.mdl
Saved in Simulink version: R2021b

File: C:\Users\MHORN1\Documents\MATLAB\Exercises_TUM_GS_Training\Simulink\mass_Spring_Damper_Control.mdl
Saved in Simulink version: R2012a

Parameter

MathWorks
Outline

Simulink and Stateflow

**Basics:**
1) Simulink
   - Basics
   - Continuous Models
   - Discrete Models
   - Subsystems
   - Signals
2) Stateflow
   - Flow Charts
   - State Charts
   - Events

**Advanced:**
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
• S-Functions are used for:
  – Hiding information about a models content (IPR)
  – Speeding up simulation
  – Integrating external functions written in C

• S-Functions can be created by Block Context Menu, by Legacy Code Tool, by S-Function Builder or they can be written by hand (template available)

• S-Functions always consist of two elements:
  • A .mexw32 file containing the compiled model
  • A S-Function Block calling the .mexw32 file

• In most cases S-Function blocks are masked to increase usability
Integrating C Code using the Legacy Code Tool

Demo: Legacy Code Tool

- LCT only creates a wrapper, which will be removed at code generation.
- Simple way to integrate C code in Simulink.
- In MATLAB use `ceval` to integrate code.

Help => „Integrating Existing C Functions into Simulink Models with the Legacy Code Tool“
Outline
Simulink and Stateflow

Basics:
1) Simulink
   • Basics
   • Continuous Models
   • Discrete Models
   • Subsystems
   • Signals
2) Stateflow
   • Flow Charts
   • State Charts
   • Events

Advanced:
1) Libraries and Model Reference
2) Style Guidelines
3) Model Advisor
4) Report Generator and Model Comparison
5) Integrating C Code using the Legacy Code Tool
6) MATLAB Coder, Simulink Coder, Embedded Coder
MATLAB Coder, Simulink Coder, Embedded Coder

MATLAB Coder

- MATLAB Coder
- Simulink Coder
- Embedded Coder

**Basics of Simulink**

MATLAB Coder

- Control System Design
- Linear System Design
- MPC Design
- PID Design
- System Identification

**SIGNAL PROCESSING AND COMMUNICATIONS**

- Bit Error Rate Analysis
- Eye Diagram Scope
- Filter Design & Analysis
- Radar Equation Calculator
- Radar Waveform Analysis
- RF Design & Analysis

**IMAGE PROCESSING AND COMPUTER VISION**

- Image Acquisition
- Image Viewer
- Map Viewer
- Video Viewer

**TEST AND MEASUREMENT**

- Instrument Control
- Vehicle CAN Bus Monitor

**CODE GENERATION**

- HDL Coder
- MATLAB Coder

**APPLICATION DEPLOYMENT**

- Generate C code or MEX function from MATLAB code (coder)
- MATLAB Coder 2.4
Summary

- Simulink is a **graphical modeling** environment **based on MATLAB**
- Simulink is **fully integrated** in MATLAB environment
- Simulink can be used to model **continuous, discrete and hybrid systems**
- In addition, Simulink is a **graphical programming** language for embedded systems
- Simulink interacts with real hardware for Hardware In The Loop or Processor in the Loop setups, as well as for test beds and laboratory setups
Contact for further information or feedback about this course:

Dipl.-Ing. Markus Hornauer  
Institute of Flight Systems Dynamics  
Boltzmannstr. 15  
85748 Garching, Germany  
Tel: +49 (0)89 289 16047  
Fax: +49 (0)89 289 16058  
Email: markus.hornauer@tum.de