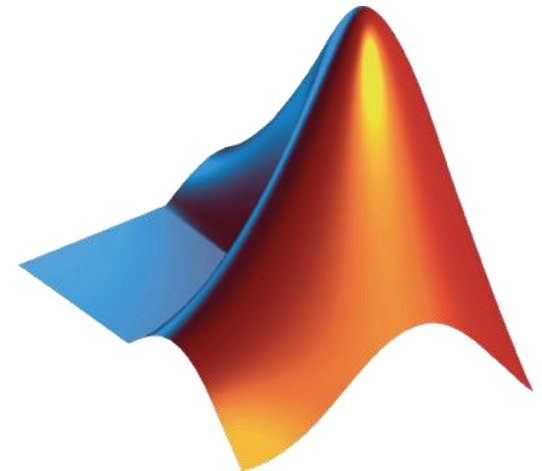


# Basics of Simulink

TUM Graduate School Training

Dipl.-Ing. Markus Hornauer



### 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 -**

### 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

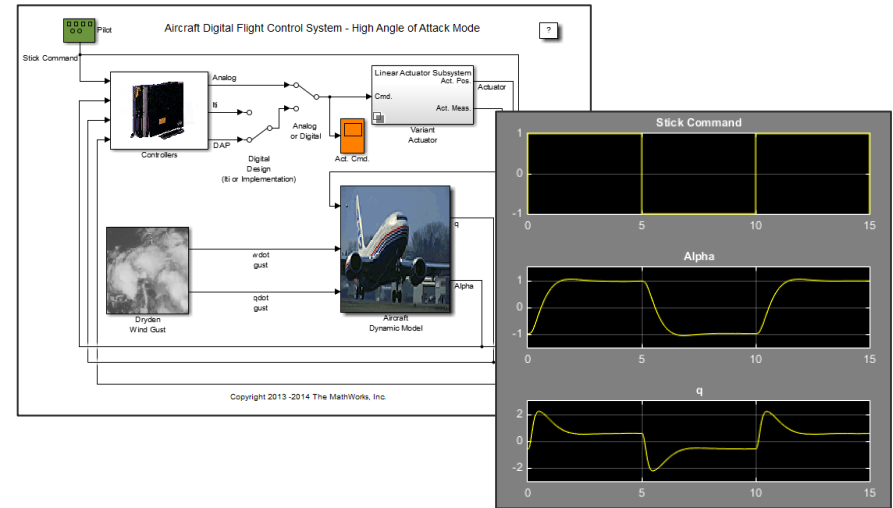
#### 4) Report Generator and Model Comparison

#### 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

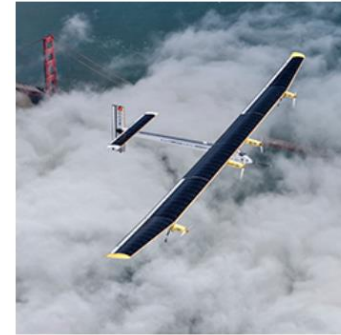


# Application Examples

- 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



The Festo Bionic Handling Assistant. Image © Festo AG.



The HB-SIA aircraft on a test flight over San Francisco Bay. Photo © Solar Impulse | Revillard | Rezo.ch



A Bosch eBike Systems drive unit.

[www.mathworks.com/company/user\\_stories/](http://www.mathworks.com/company/user_stories/)

# Simulink – Basics

## Launching Simulink

The image shows the MATLAB R2013a interface. The Simulink Library button in the ribbon is circled in red. The Command Window contains the command `>> simulink`, which is also circled in red. A red box with the text "Starting Simulink" has arrows pointing to both the button and the command. The Command History window shows a list of commands, with `simulink` at the bottom.

Name	Date Modified
File Folder	
daqwaterfall	16.11.2012 19:15
MATLAB_Ex	16.11.2012 19:15
MuPAD	16.11.2012 19:15
Simulink	14.01.2013 12:45
SL_Figure_Demo	16.11.2012 19:15
Stateflow	16.11.2012 19:15

```
>> simulink
fx >>
```

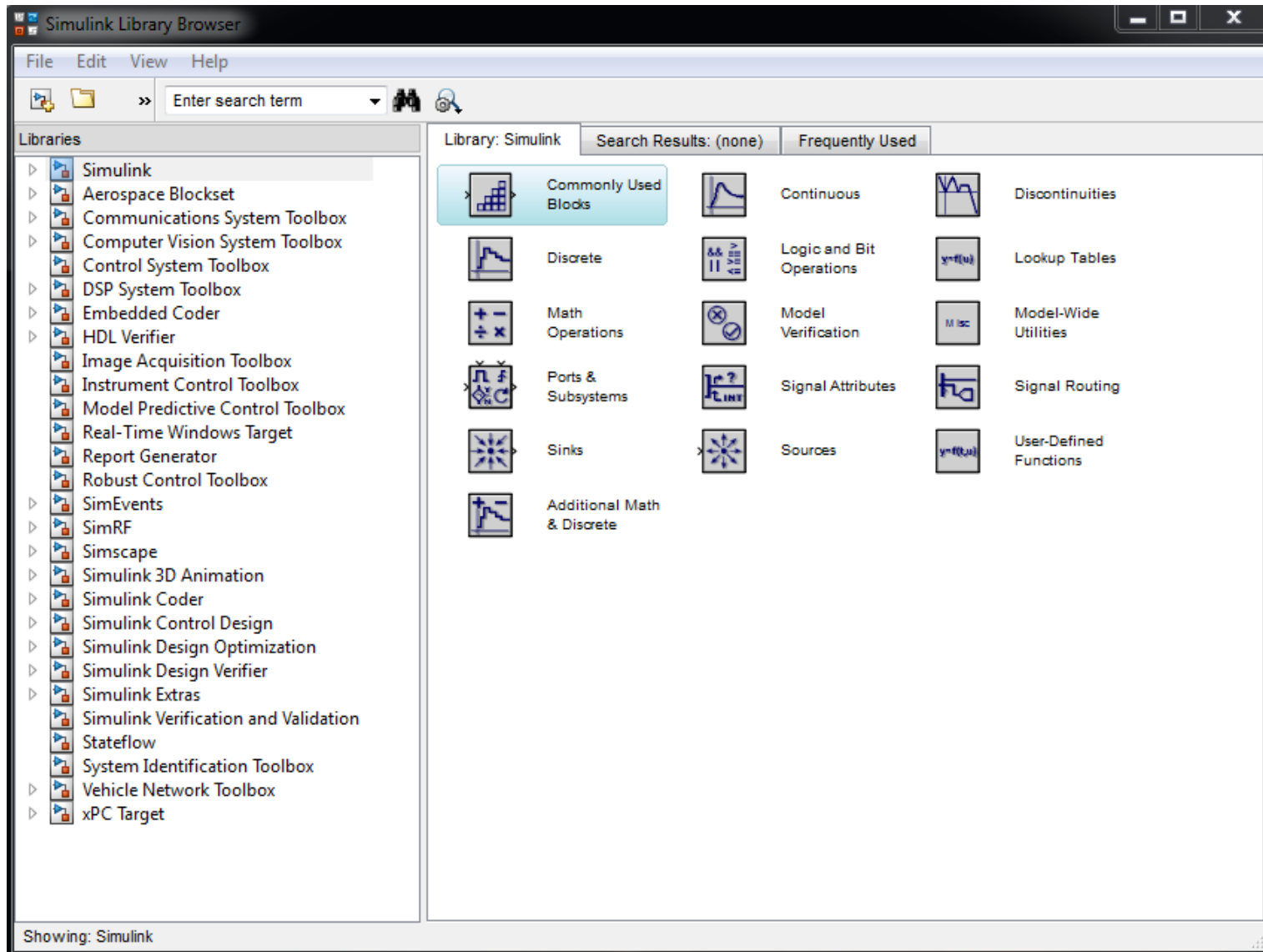
Command History

```
plot(t,y)
clear all
close all
clc
example_foo
clear all
close all
clc
graf3d
$-- 07.07.2013 19:31 --$
simulink
```



# Simulink – Basics

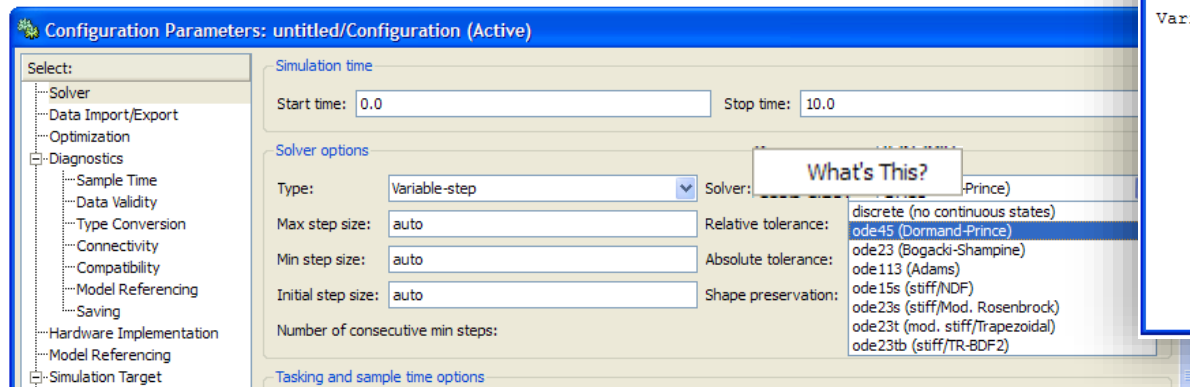
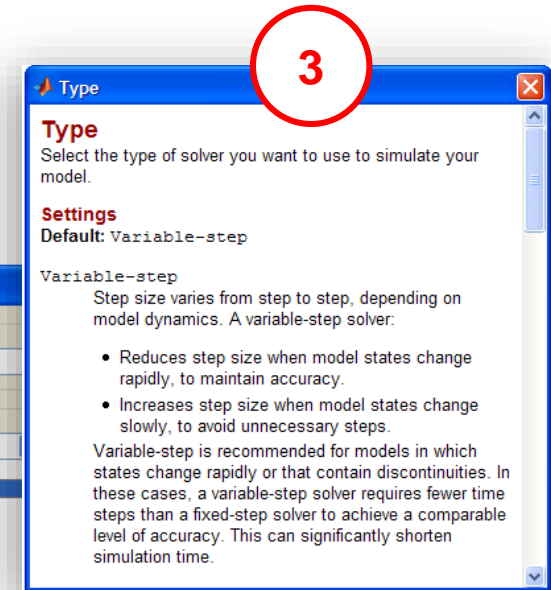
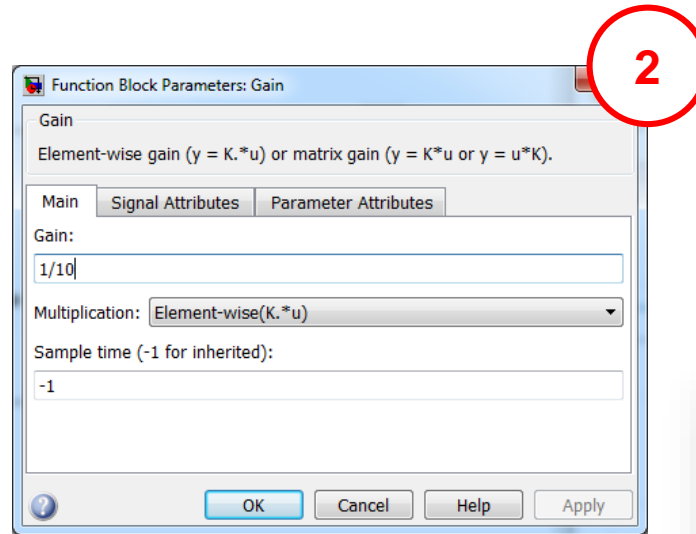
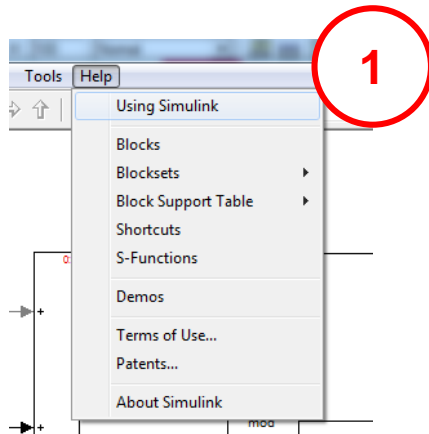
## Simulink Library Browser



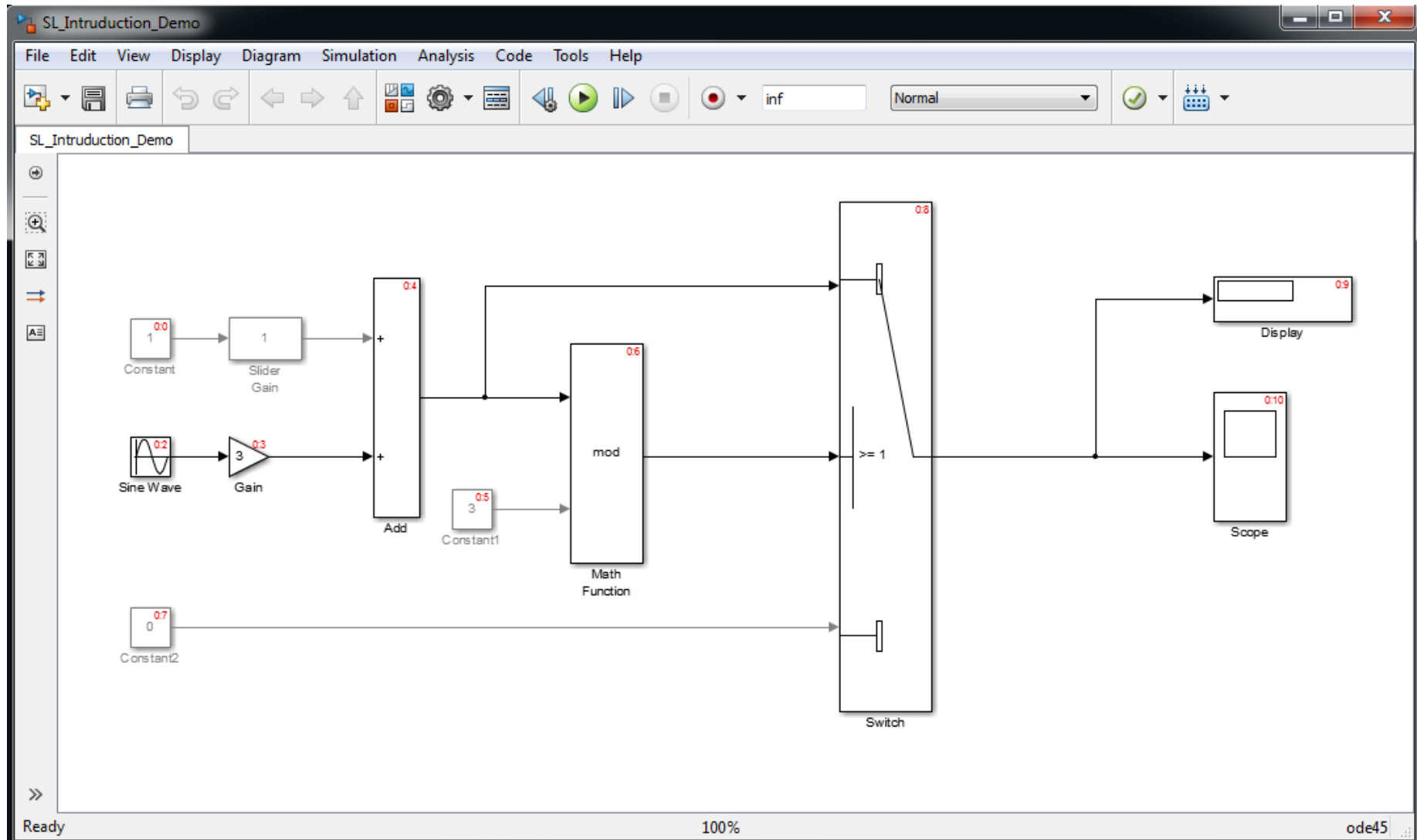
The screenshot displays the Simulink Library Browser interface. The search bar at the top contains the text 'cos', which is highlighted with a red circle. The left sidebar shows a tree view of libraries, with 'Lookup Tables' selected. The main area shows search results for 'cos' across four blocksets:

- Simulink** (1 block): Cosine
- Aerospace Blockset** (10 blocks):
  - Direction Cosine Matrix to Qua...
  - Direction Cosine Matrix Body to...
  - Direction Cosine Matrix ECEF t...
  - Direction Cosine Matrix ECEF t...
  - Direction Cosine Matrix to Wind...
  - Quaternions to Direction Cosi...
  - Rotation Angles to Direction C...
  - Wind Angles to Direction Cosi...
- Communications System Toolbox** (2 blocks):
  - Raised Cosine Receive Filter
  - Raised Cosine Transmit Filter
- HDL Verifier** (2 blocks):
  - HDL Cosimulation
  - HDL Cosimulation

At the bottom, a status bar indicates: Matches for 'cos' 4 blocksets 0 subsystems 15 blocks.



## Demonstration of Model Elements



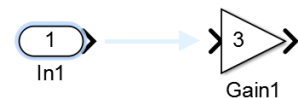
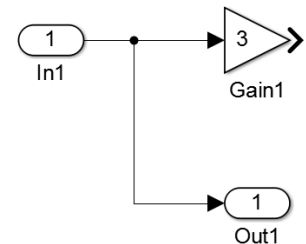
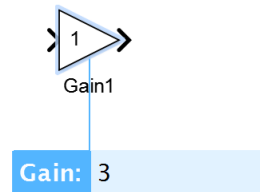
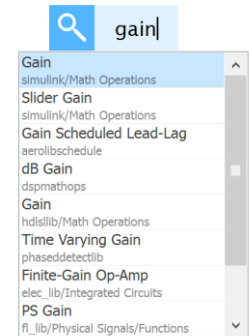
## Summary – Using Blocks and Signals

### 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)

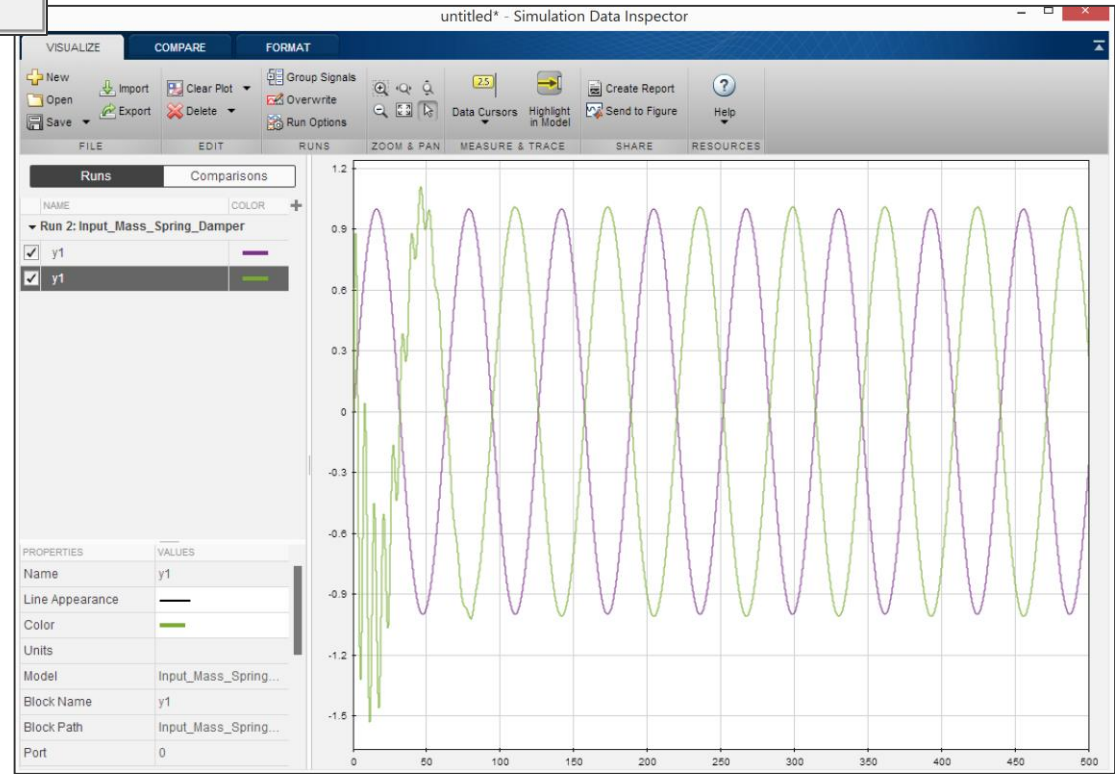
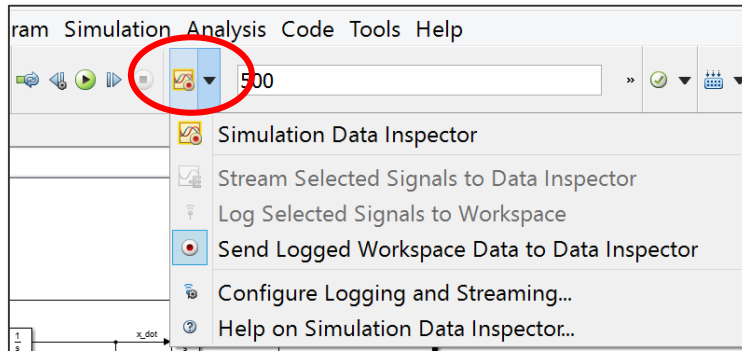
### Connecting Blocks:

- Draw a line from the output 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 output of the first block and the inport of the second block while keeping the CTRL-key pressed
- Click the suggested connection lines (R14b)



# Simulink – Basics

## Simulation Data Inspector



### 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

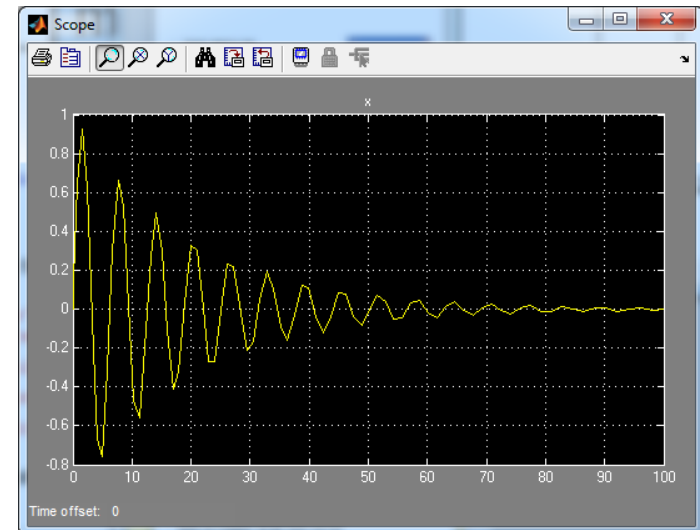
#### 4) Report Generator and Model Comparison

#### 5) Integrating C Code using the Legacy Code Tool

#### 6) MATLAB Coder, Simulink Coder, Embedded Coder

## Modeling Continuous Systems

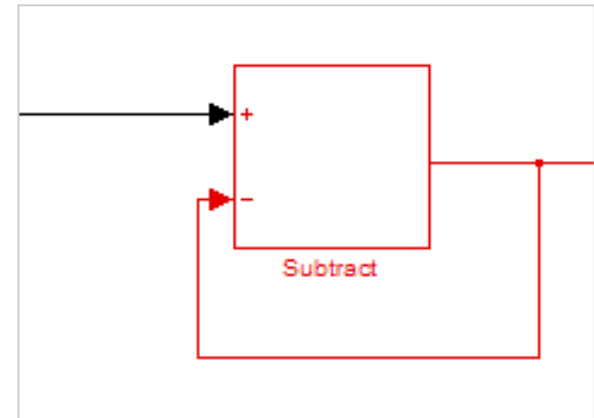
- Engine provides variable-step and fixed-step ODE solvers
  - Block Diagram representation of dynamic systems
  - Blocks define governing equations
  - Signals are propagated between blocks over time
- 
- Remember MuPad:  $m\ddot{x} + b\dot{x} + kx = 0,$   
 $x(0) = 0, \dot{x}(0) = 1, m = 10, b = 1, k = 10$
- 
- **Exercise:** Create a mass-spring-damper system in Simulink





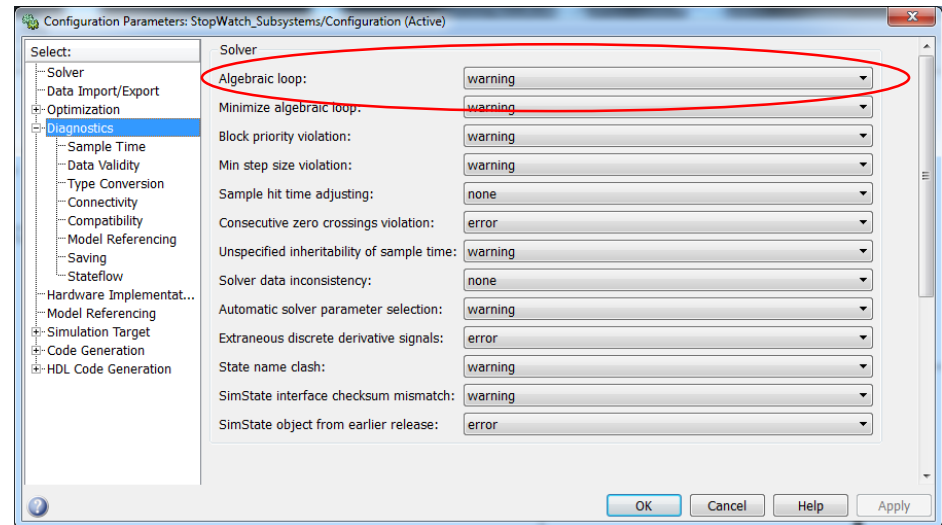
## Algebraic Loops

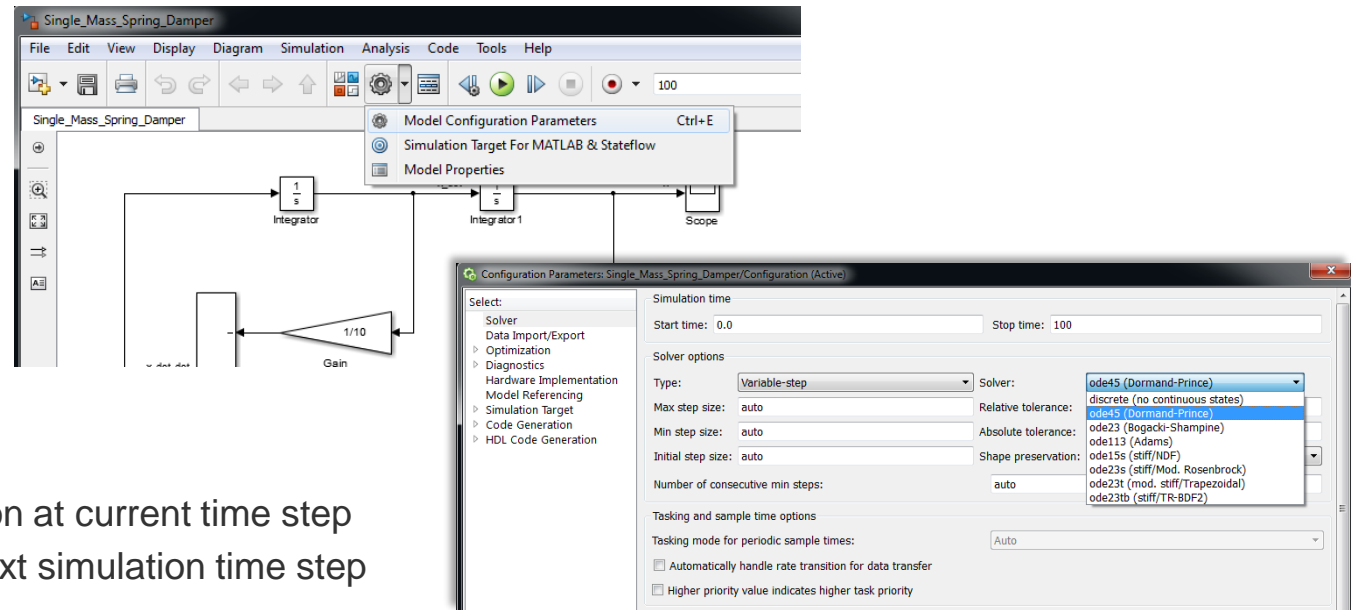
- Error because of *Direct Feedthrough*
  - Block input depends directly on its own output
  - E.g.: Gain, Product, Sum/Subtract, Transfer Fcn
- Recommended solution: use Delay, Integrator or other history related block



Block diagram 'untitled' contains an algebraic loop. The algebraic loop solver is disabled because of the current setting for Algebraic loop option in the Diagnostics page of the Configuration Parameters Dialog

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





- 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 Solver

## Fixed-step Solvers.

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

`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:

$$X(n+1) = X(n) + h * DX(n)$$

`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:

$$X(n+1) - X(n) - h * DX(n+1) = 0$$

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

# Variable-step Solver

## 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/NDF)

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)

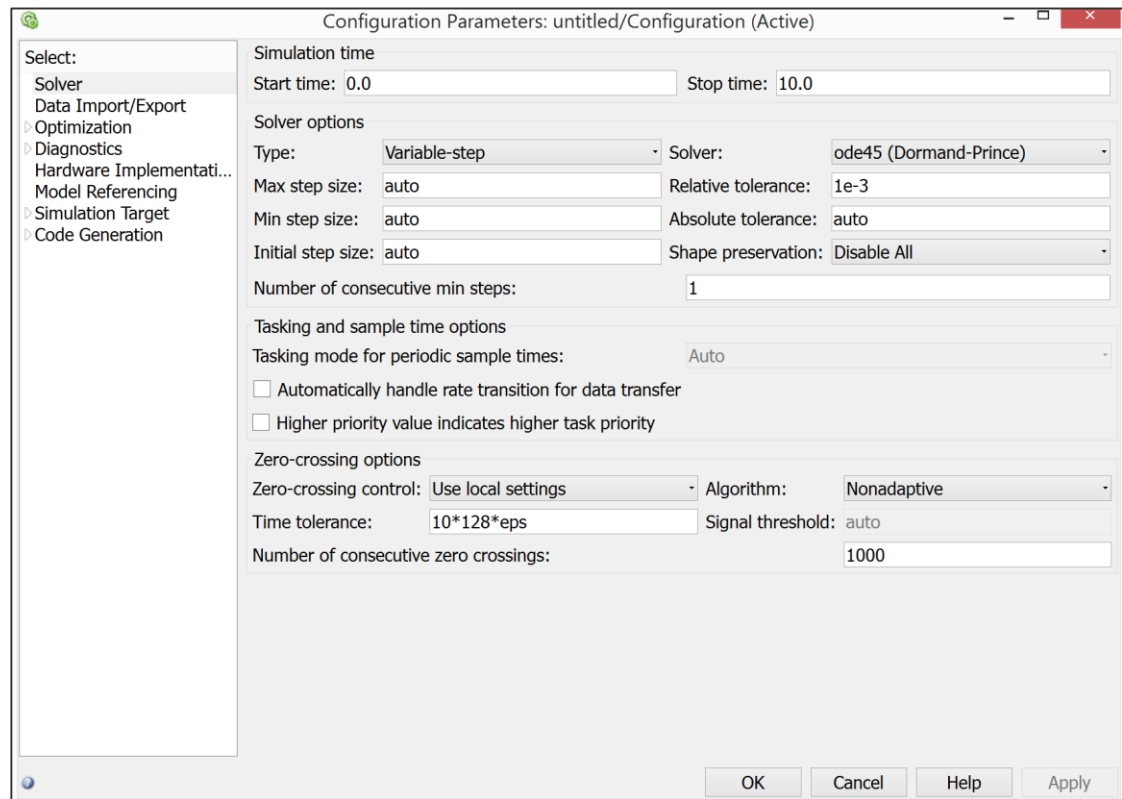
Computes the model's state at the next time step using a multistep implementation of TR-BDF2, an implicit Runge-Kutta formula with a trapezoidal rule first stage, and a second stage consisting of a backward differentiation formula of order two. By construction, the same iteration matrix is used in evaluating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

`ode23tb` is more efficient than `ode15s` at crude tolerances, and can solve stiff problems for which `ode15s` is ineffective.

## Variable-step Size

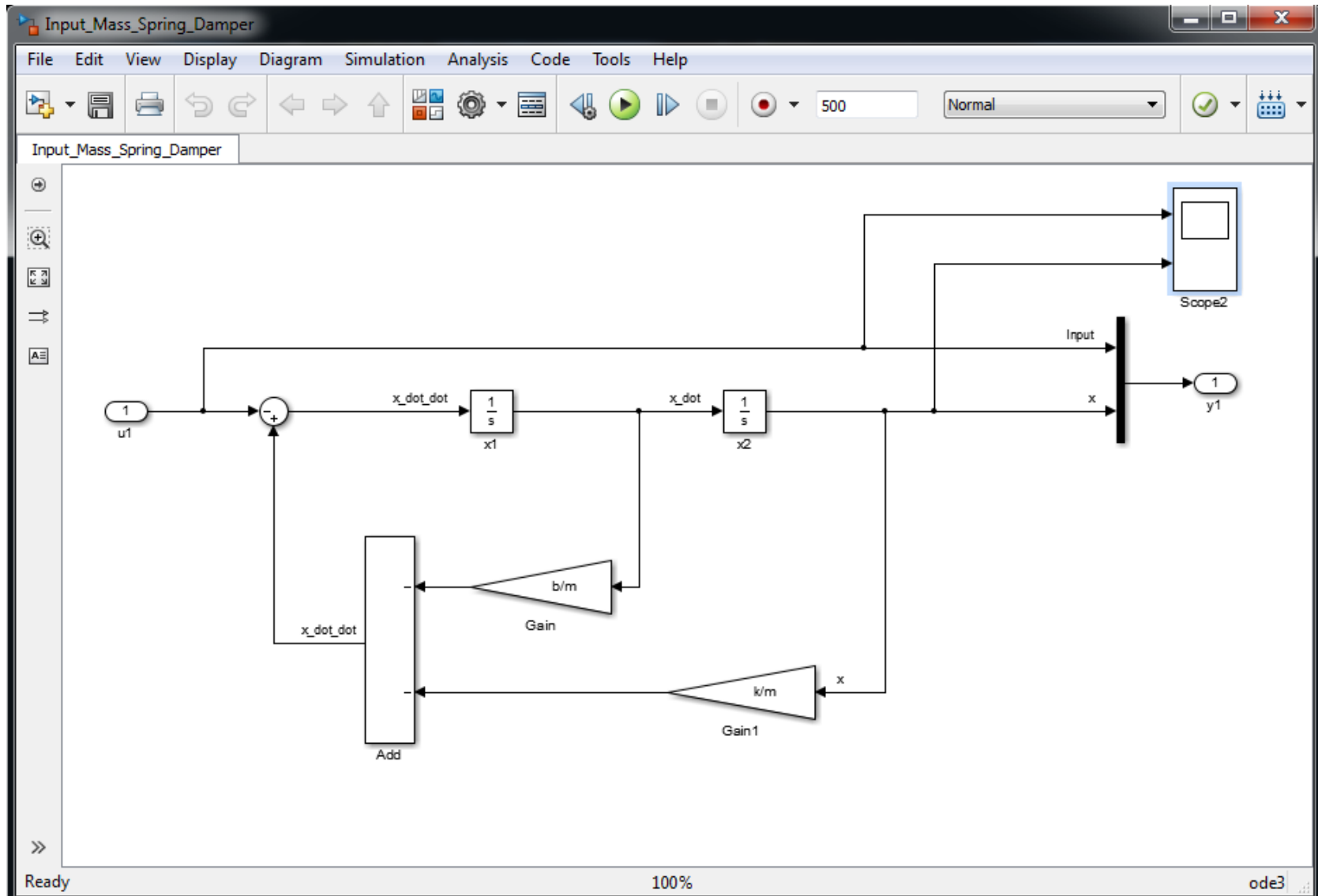
$$h_{\max} = \frac{t_{\text{stop}} - t_{\text{start}}}{50}$$

Max step size is the largest time step that the solver can take ('auto' means 50 steps)



# Simulink – Continuous Systems

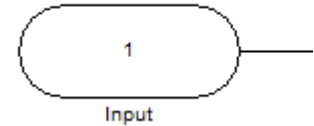
## Demo: Importing and Exporting Data



## Importing Data into Simulink

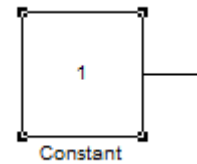
### IN

- Requires vector of time along with input values  
 $\text{input}(t, u_1 \dots u_n)$  defined configuration parameters



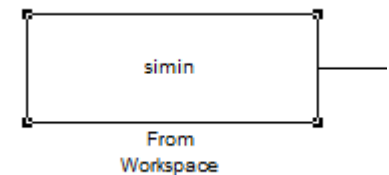
### CONSTANT

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



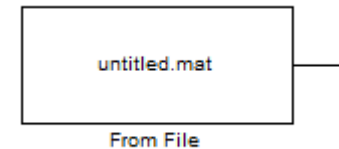
### FROM WORKSPACE

- Requires vector of time along with input values  
 $\text{input}(t, u_1 \dots u_n)$  defined a workspace variable



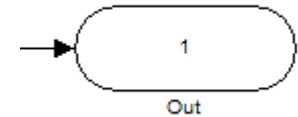
### FROM FILE

- Requires vector of time along with input values  
 $\text{input}(t, u_1 \dots u_n)$  defined in a given mat file



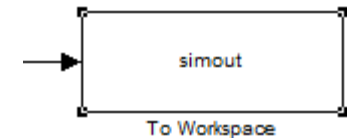
### OUT

- Saves all outputs together in one variable, defined in the configuration parameters



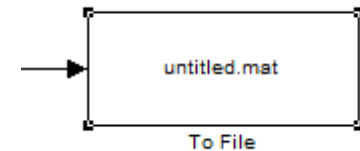
### TO WORKSPACE

- Saves the output data in a variable to the workspace, defined in the block parameters



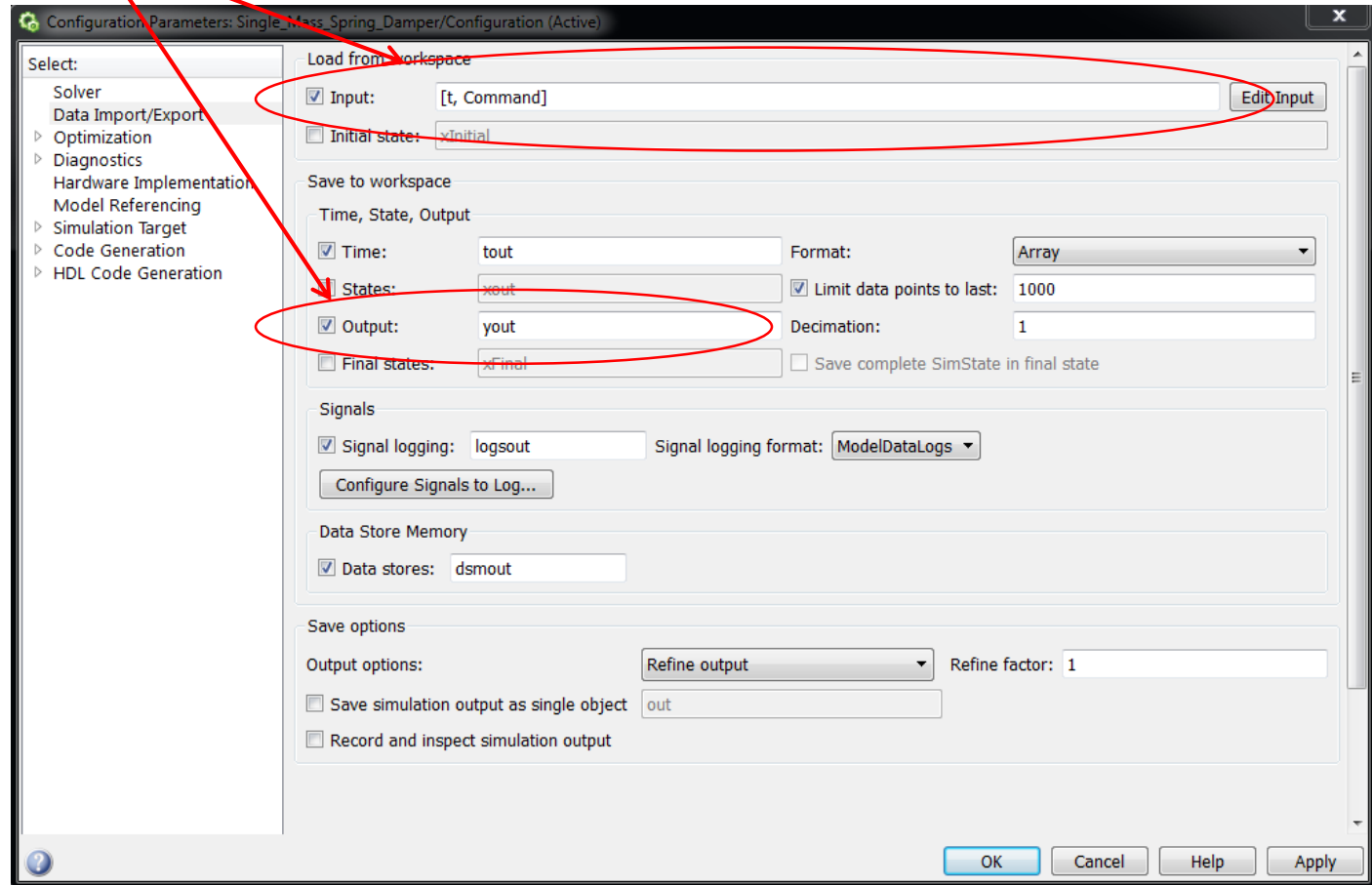
### TO FILE

- Saves the output data in a .mat file



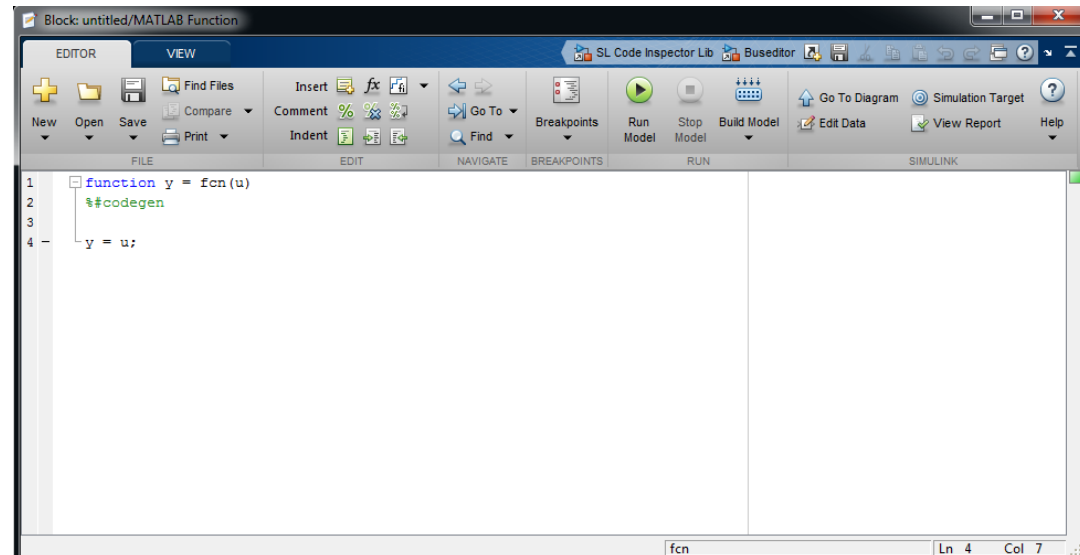
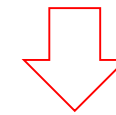
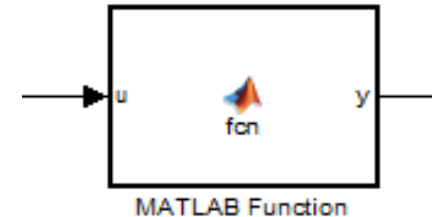


For IN and OUT blocks variables must be defined in Configuration Parameters



## MATLAB Embedded

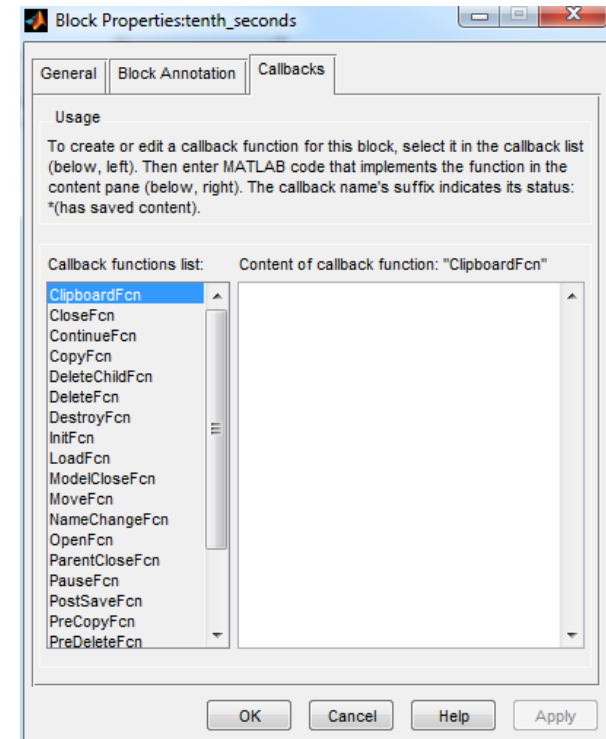
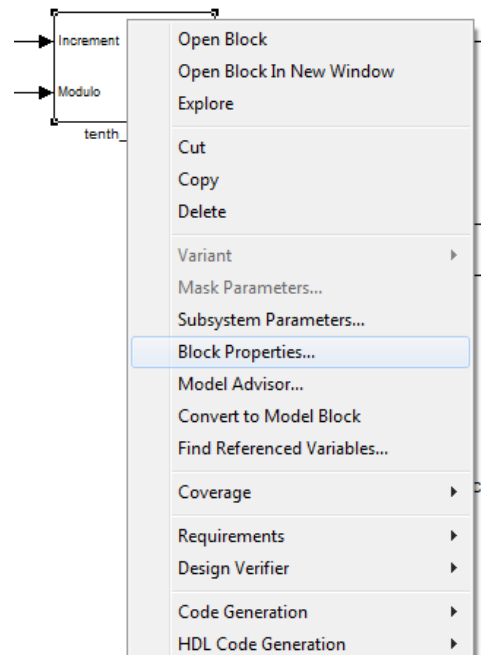
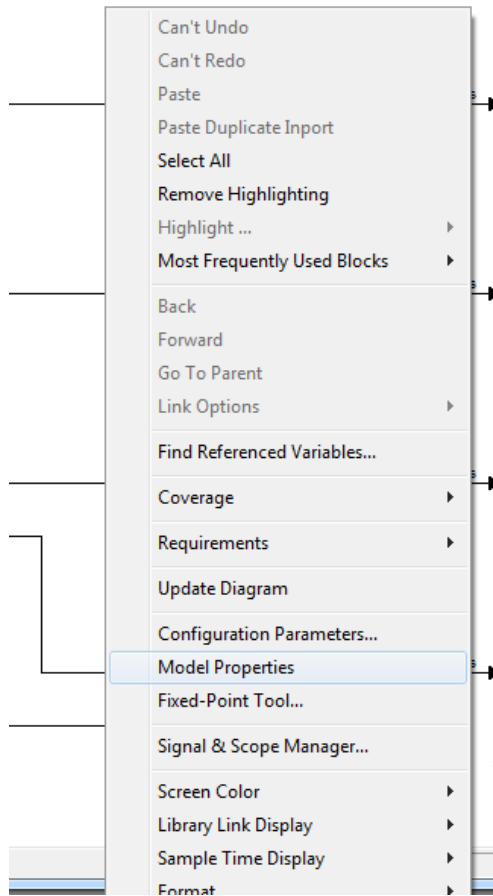
- Subset of MATLAB for code generation
- Can be used for direct generation of source code out of MATLAB as well as in Simulink MATLAB Function blocks
- Enables user to reuse his MATLAB code in Simulink
- To call unsupported functions use `eml.extrinsic` or `coder.extrinsic` (leads to significantly reduced performance!!!)



A screenshot of the MATLAB Function block code editor. The window title is 'Block: untitled/MATLAB Function'. The editor shows the following code:

```
1 function y = fcn(u)
2   %#codegen
3
4   y = u;
```

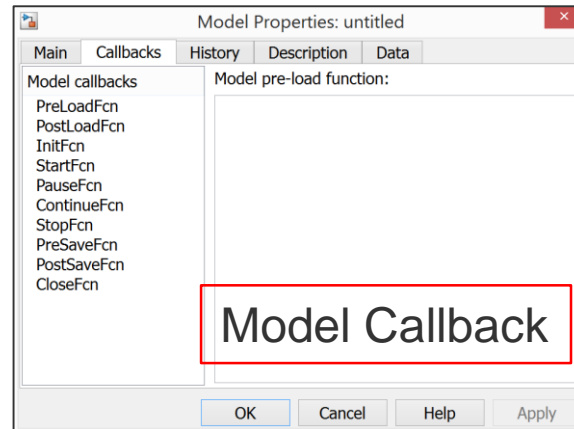
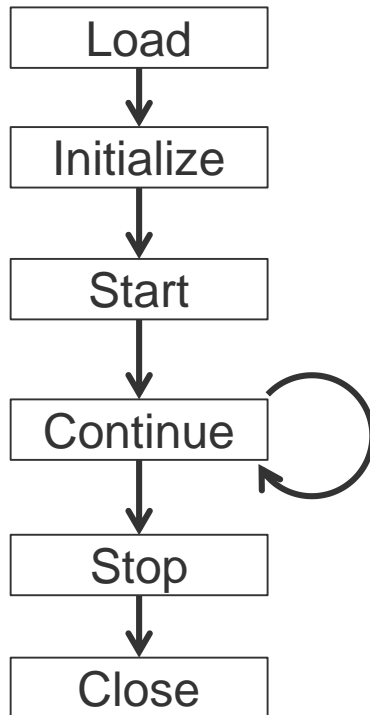
The editor interface includes a menu bar with 'EDITOR' and 'VIEW' tabs, and a toolbar with various icons for file operations, editing, navigation, and simulation. The status bar at the bottom indicates the file name 'fcn' and the current cursor position 'Ln 4 Col 7'.



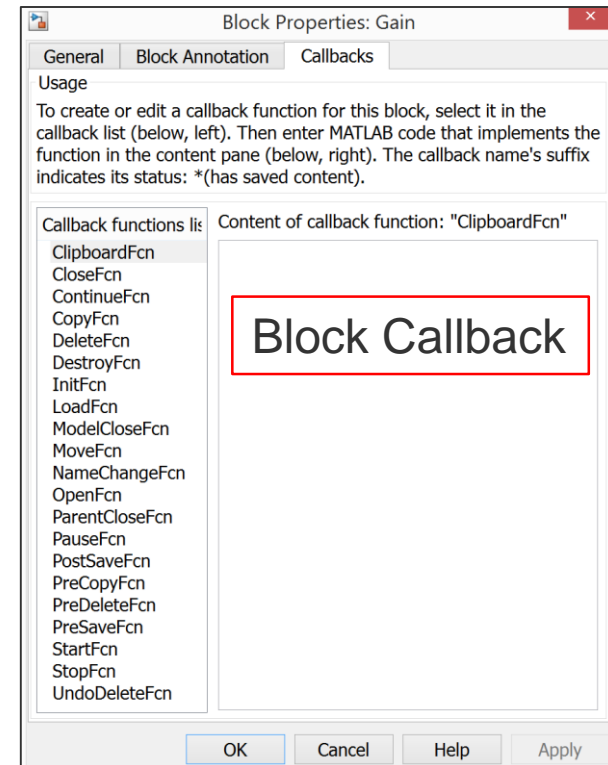
## Callback Functions

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



**Help => Callback Functions**



### 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

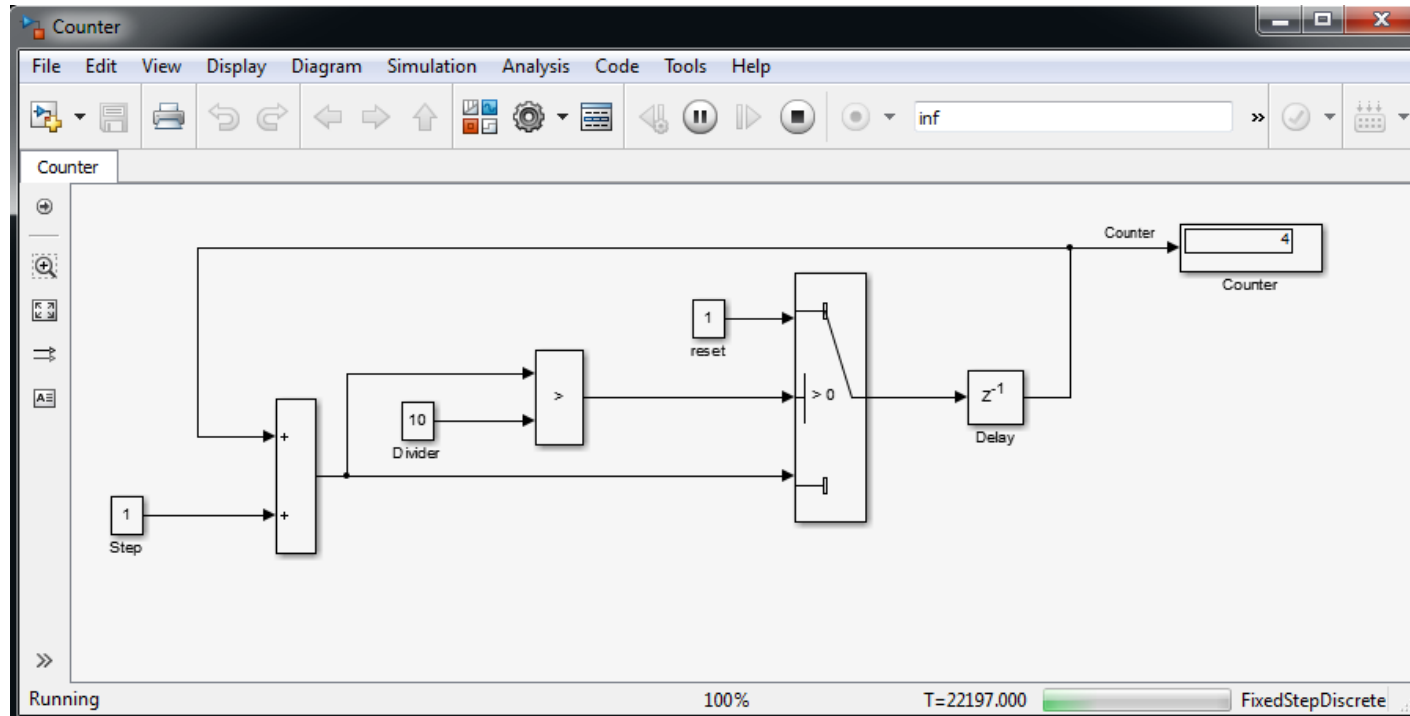
#### 3) Model Advisor

#### 4) Report Generator and Model Comparison

#### 5) Integrating C Code using the Legacy Code Tool

#### 6) MATLAB Coder, Simulink Coder, Embedded Coder

## Demo: Creating a counter

**Exercise: Create a Stop Watch**

- Combine counters to a stop watch
- Show tenth seconds, seconds, minutes and hours
- Reduce simulation speed to soft realtime

## Multirate Systems

- Systems with signals that are sampled at different rates
- Use for discrete or hybrid systems
- To connect system 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

**Sample Time Legend**

StopWatch\_multirate

**Sample Times for 'StopWatch\_multirate'**

Color	Description	Value
Red	Discrete 1	0.1
Green	Discrete 2	1
Blue	Discrete 3	60
Light Blue	Discrete 4	3600

Print

StopWatch\_multirate

File Edit View Display Diagram Simulation Analysis Code Tools Help

StopWatch\_multirate

StopWatch\_multirate

StopWatch\_multirate

StopWatch\_multirate

Running 100% T=18.600 FixedStepDiscrete

### 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

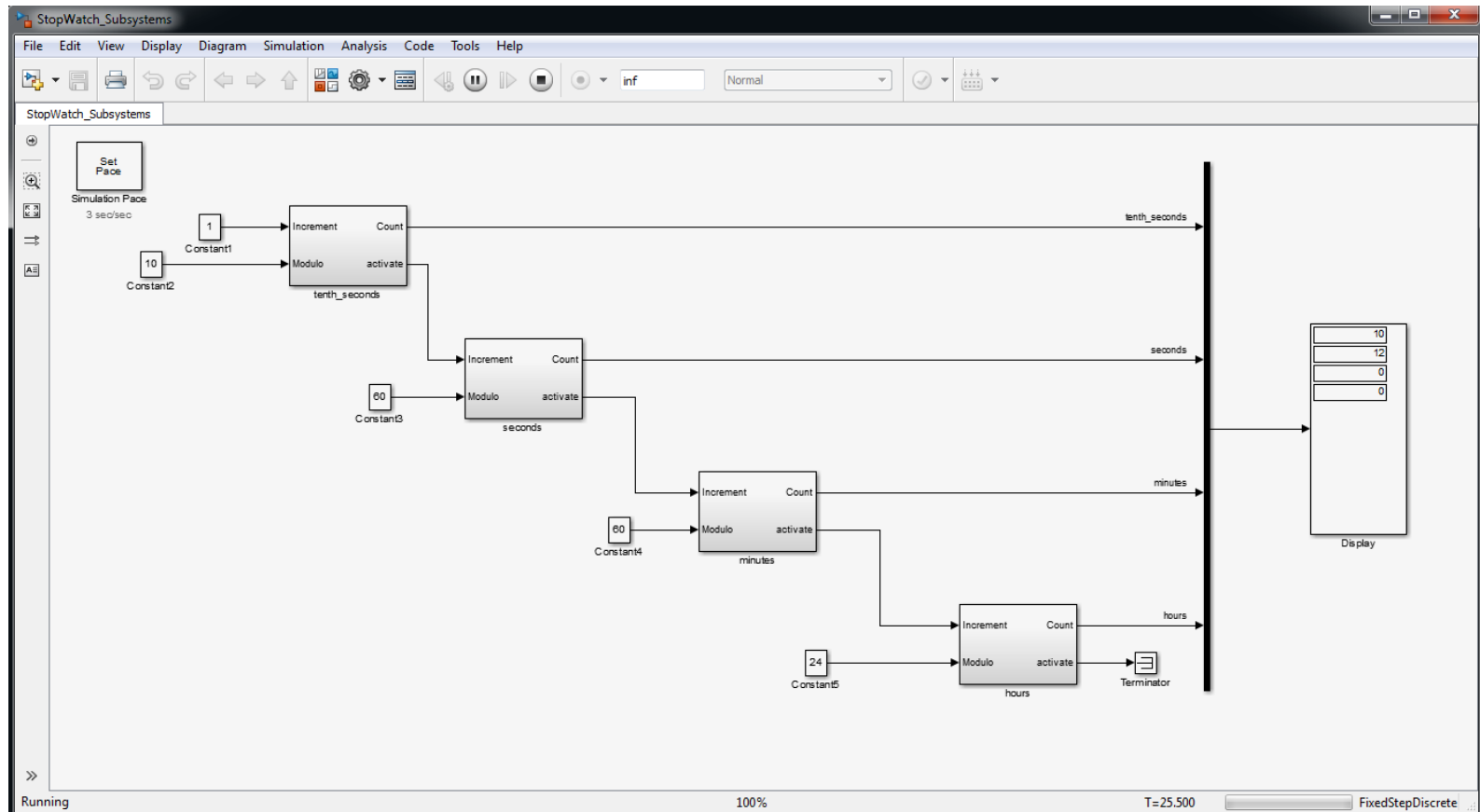
#### 4) Report Generator and Model Comparison

#### 5) Integrating C Code using the Legacy Code Tool

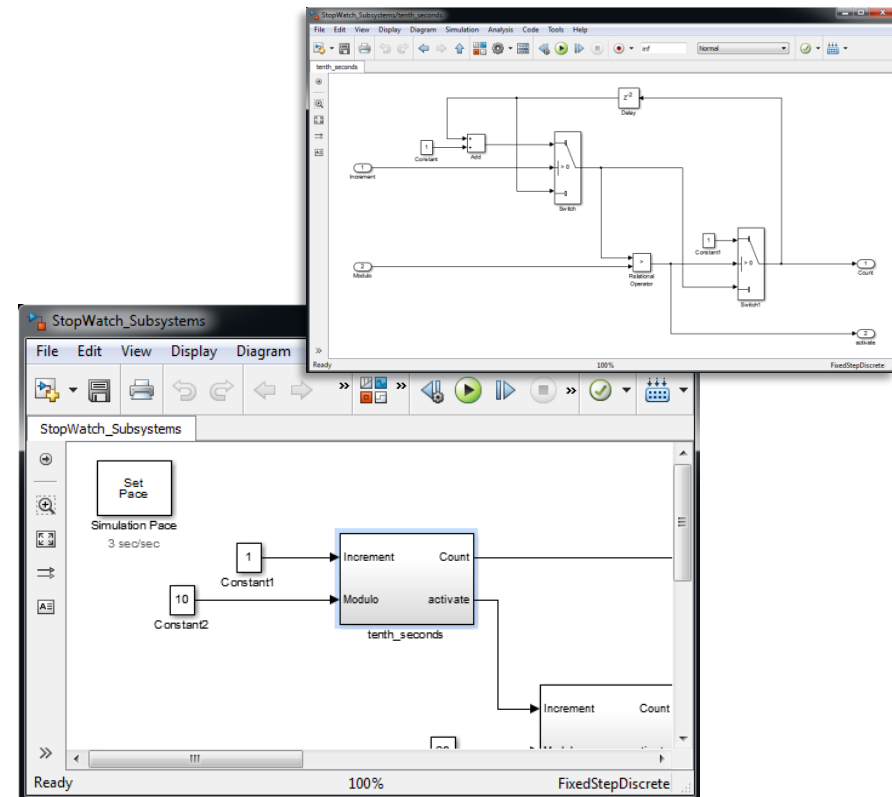
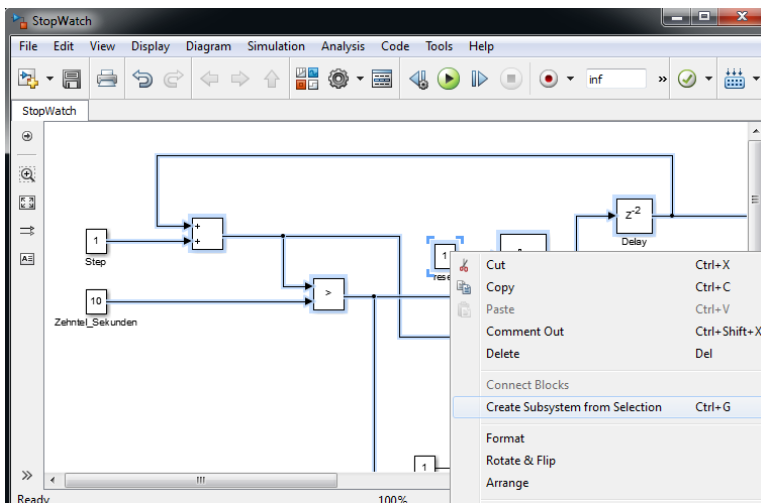
#### 6) MATLAB Coder, Simulink Coder, Embedded Coder



- Why?
  - Reduce blocks displayed in a model window
  - Keep functionally related block together
  - Establish hierarchical block diagram

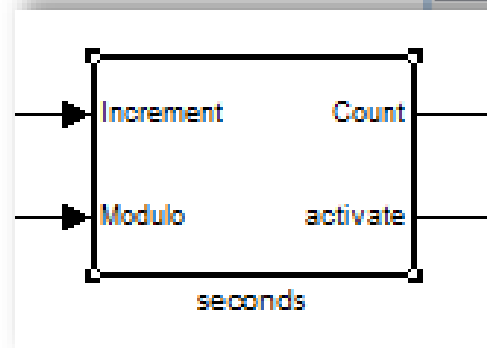
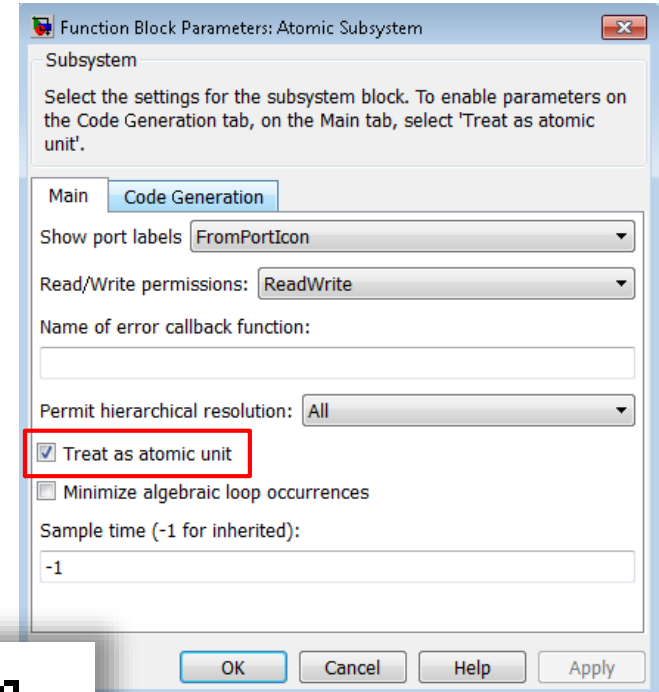


- Context menu -> Create Subsystem
- Subsystem ports
- Inside a subsystem

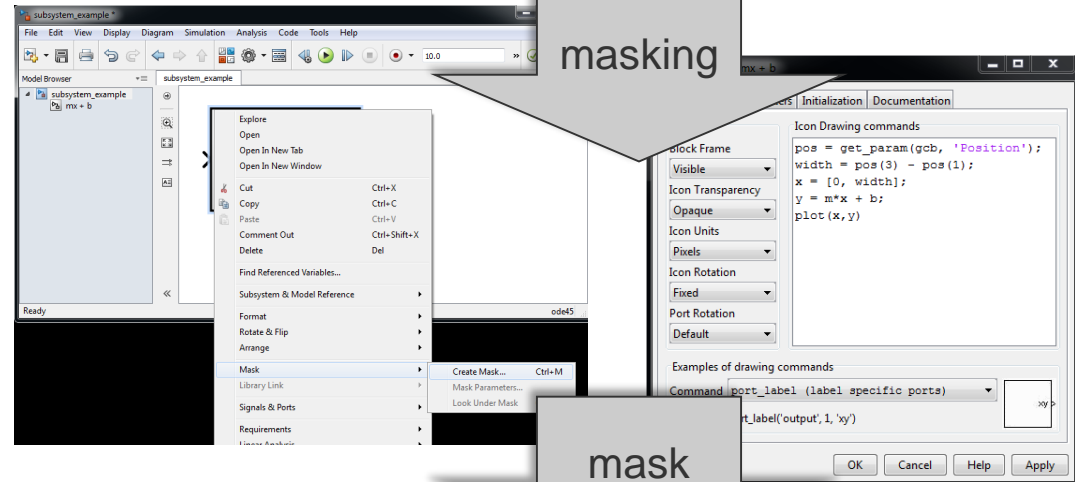
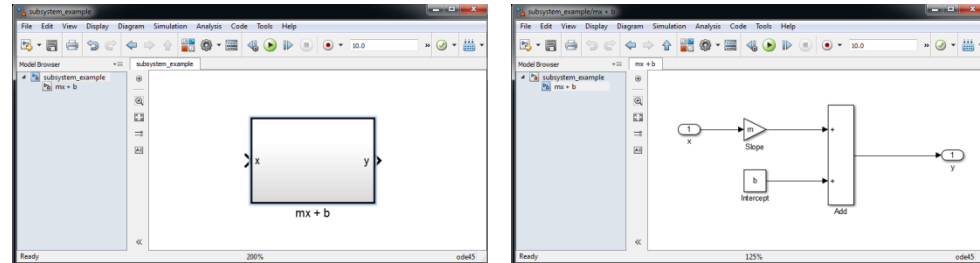


# Atomic Subsystems

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



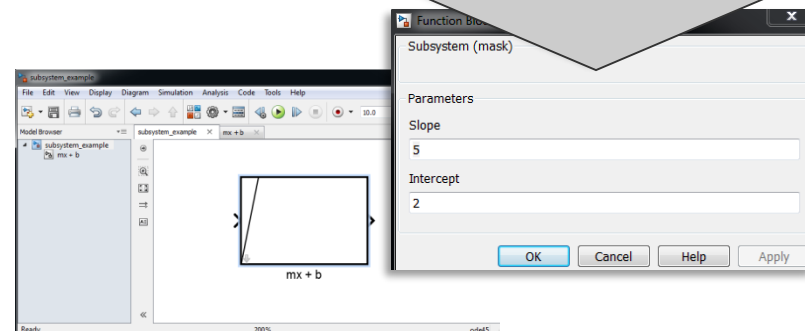
- **Mask - Encapsulation with a UI**
- **Provides**
  - Mask icon display
  - Block description
  - Parameter dialog prompt
  - Custom block help text



masking

mask

```
Icon Drawing commands
pos = get_param(gcf, 'Position');
width = pos(3) - pos(1);
x = [0, width];
y = m*x + b;
plot(x, y)
```



Subsystem (mask)

Parameters

Slope  
5

Intercept  
2

OK Cancel Help Apply

### Basics:

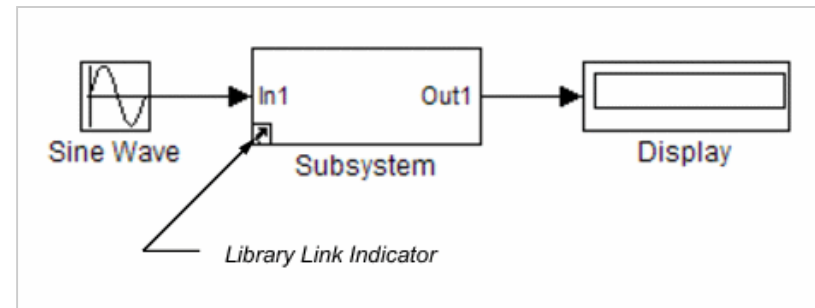
- 1) Simulink
- 2) Stateflow

### Advanced:

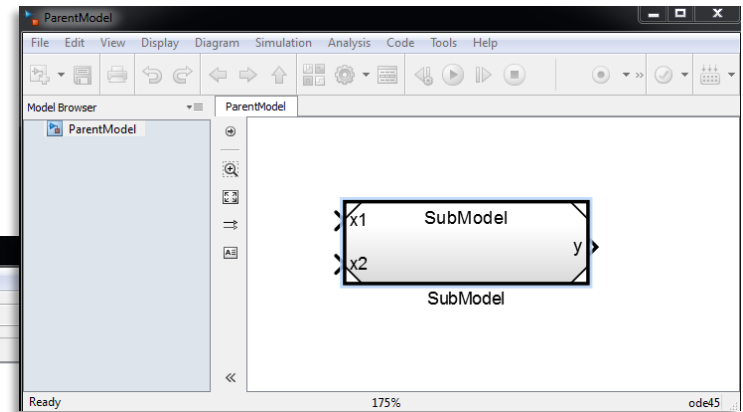
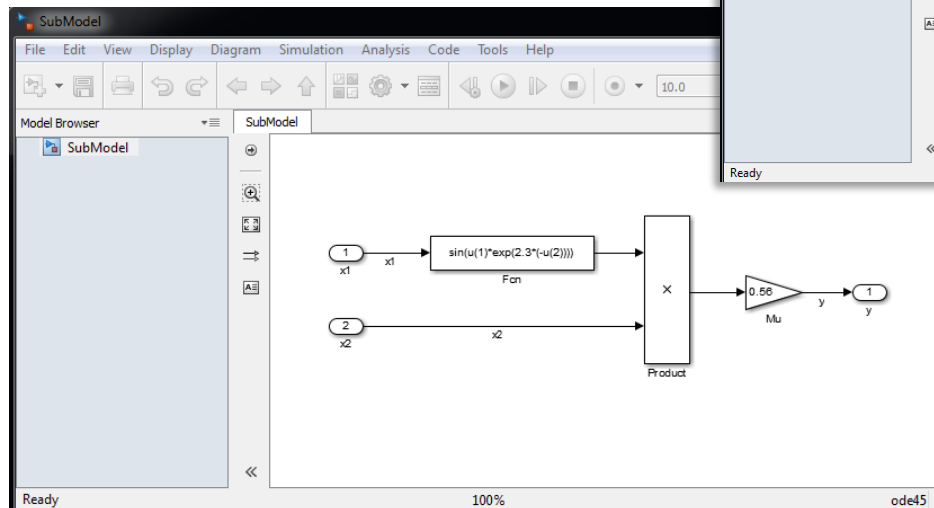
- 1) Libraries and Model Reference
- 2) Style Guidelines
- 3) Model Advisor
- 4) Report Generator and Model Comparison
- 5) Integrating C Code using the Legacy Code Tool
- 6) MATLAB Coder, Simulink Coder, Embedded Coder

## 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



- One model in another- *parent and referenced model*
- Advantages:
  - Componentization/Modularization
  - IP protection
  - Multiple referencing
  - Acceleration



### 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

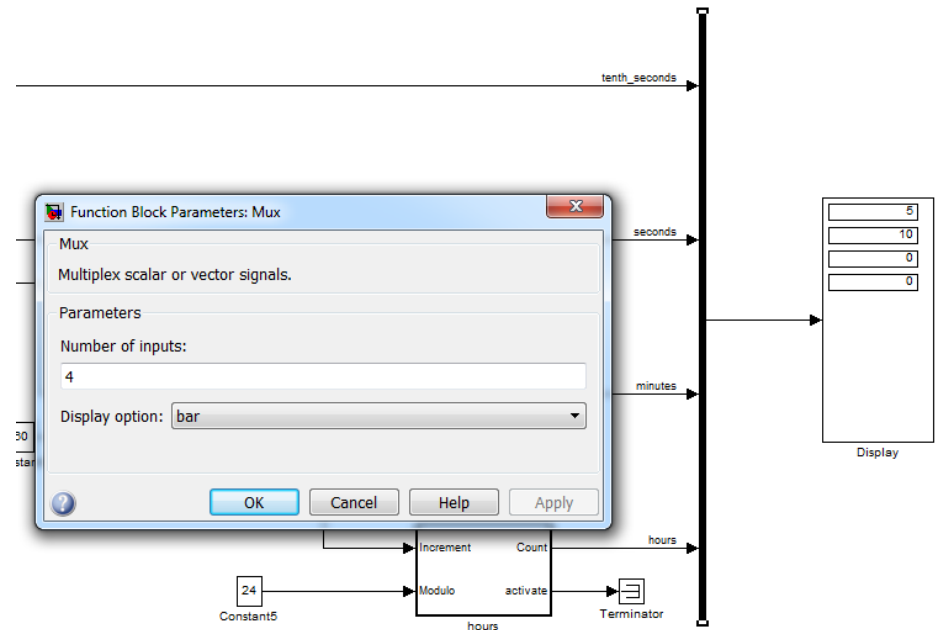
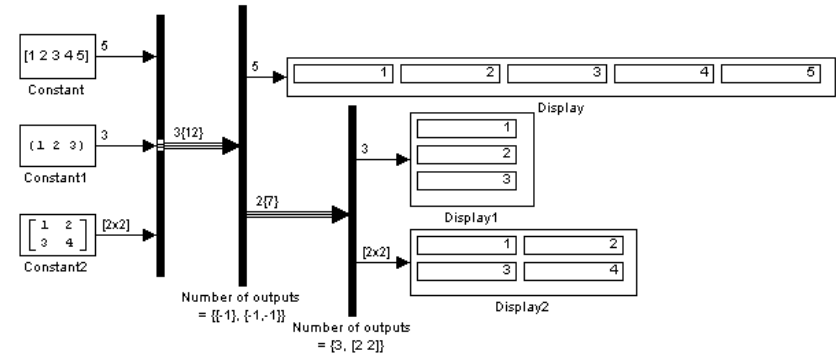
#### 5) Integrating C Code using the Legacy Code Tool

#### 6) MATLAB Coder, Simulink Coder, Embedded Coder

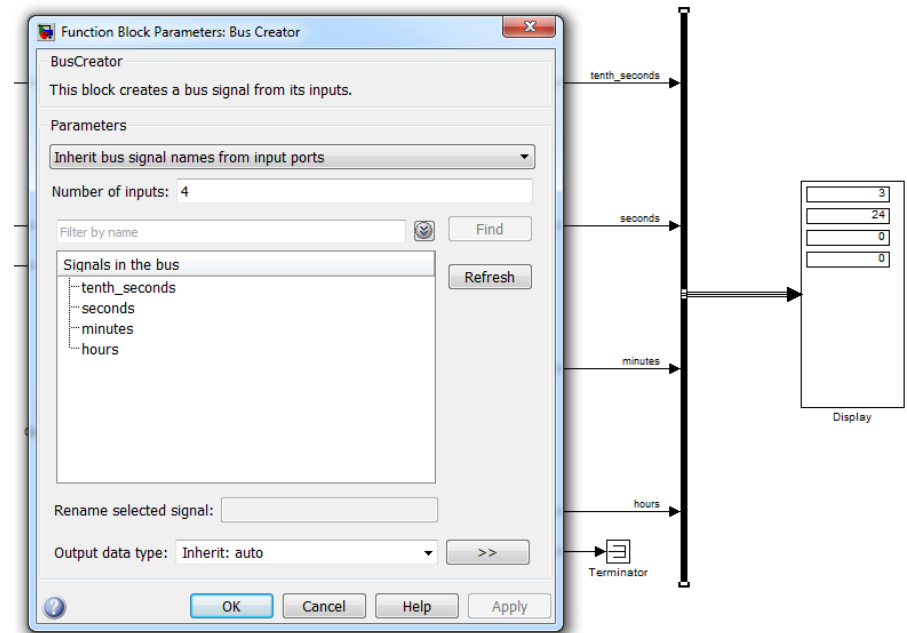


## Vectors

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



- 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



The image displays the Simulink Model Explorer interface. The top window shows a block diagram with two input signals,  $x_1$  and  $x_2$ , entering a function block labeled 'Fcn' with the expression  $\sin(u(1)) \cdot \exp(2.3 \cdot (-u(2)))$ . The output of the 'Fcn' block is multiplied by a gain block labeled 'Mu' with a value of 0.56, resulting in the output signal  $y$ . The 'Model Explorer' window is overlaid on the diagram, showing the model hierarchy and a table of contents.

**Model Hierarchy:**

- Simulink Root
  - Base Workspace
  - SubModel
    - Model Workspace
    - Configuration (Active)
    - Code for SubModel
    - Advice for SubModel
    - Simulink Design Verifier results

**Table of Contents:**

Name	BlockType	SampleTime	OutDataTypeStr	OutMir
Model Workspace				
Configuration (Active)				
Code for SubModel				
Advice for SubModel				
Simulink Design Verifier results				
y	Output	-1	Inherit: auto	0
x1	Inport	-1	Inherit: auto	0
x2	Inport	-1	Inherit: auto	0
Product	Product	-1	Inherit: Inherit via internal rule	0
Mu	Gain	-1	Inherit: Inherit via internal rule	0
Fcn	Fcn	-1		
x1				
x2				
y				

**Model Properties: SubModel**

**Model Information for: SubModel**

- Source File: C:\Users\MHORN1\Documents\MATLAB\Exercises\_TU
- Last Saved: Mon Jul 08 19:50:16 2013
- Created On: Mon Jul 08 19:47:21 2013
- Is Modified: no
- Model Version: 1.1

- Simulink Parameter Object

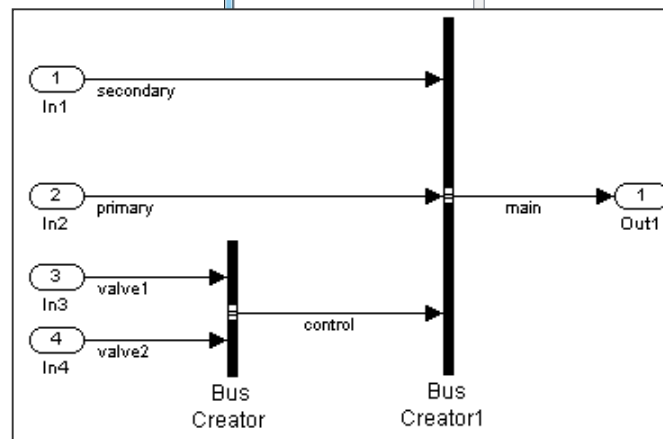
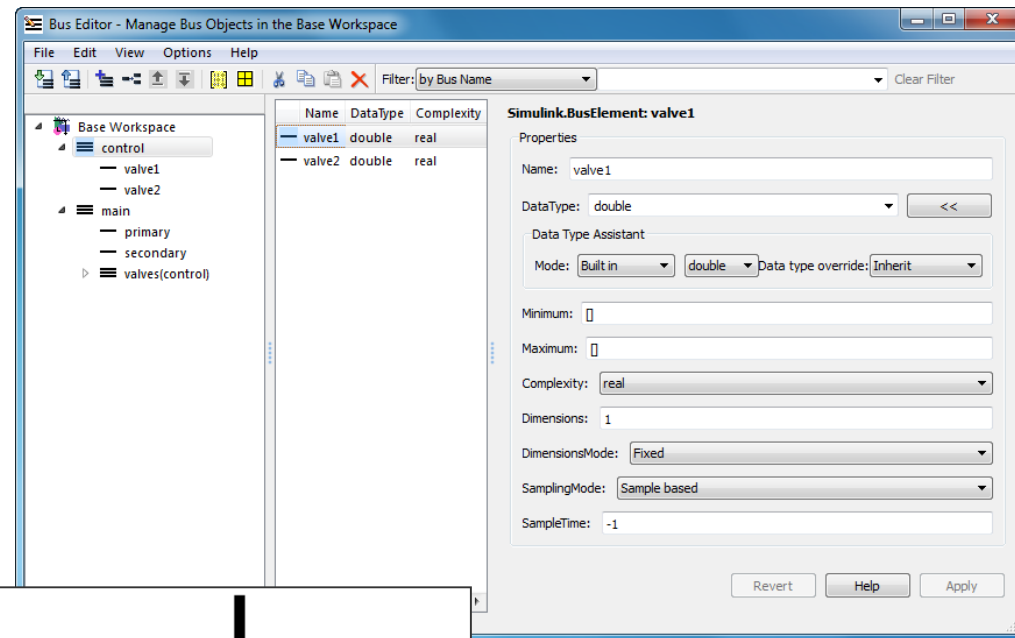
```
>> Var = Simulink.Parameter
```

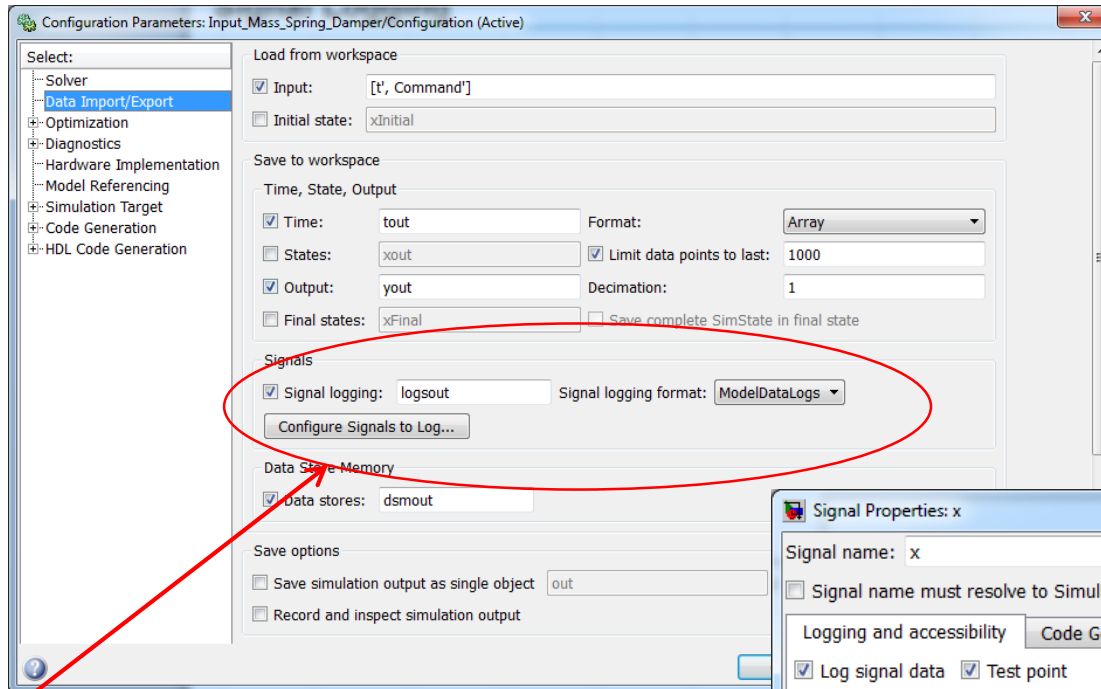
- Simulink Signal Object

```
>> Var = Simulink.Signal
```

- Simulink Bus Object

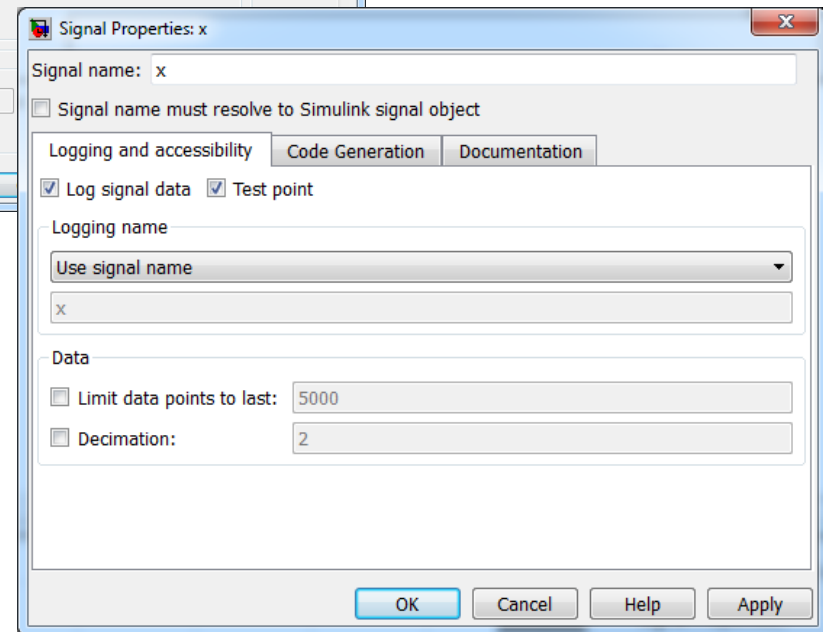
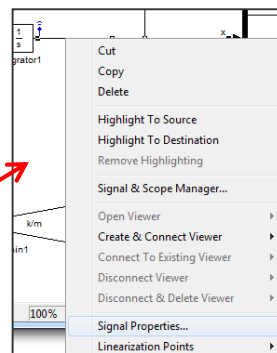
- Use Bus editor





Configuration Parameters

Signal Context Menu



### 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

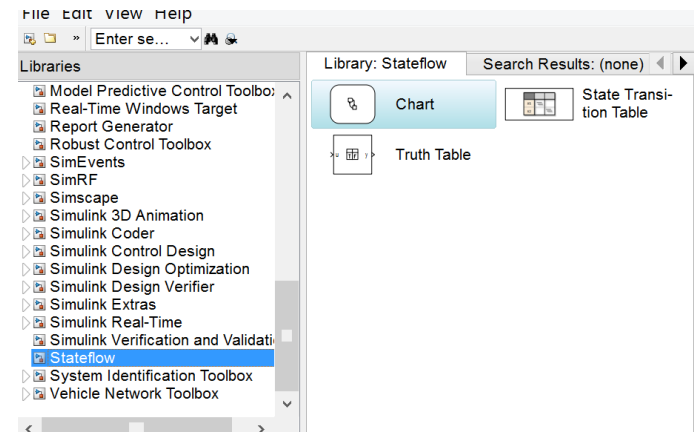
#### 4) Report Generator and Model Comparison

#### 5) Integrating C Code using the Legacy Code Tool

#### 6) MATLAB Coder, Simulink Coder, Embedded Coder

## 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



## When to use Stateflow?

### 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.



### 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

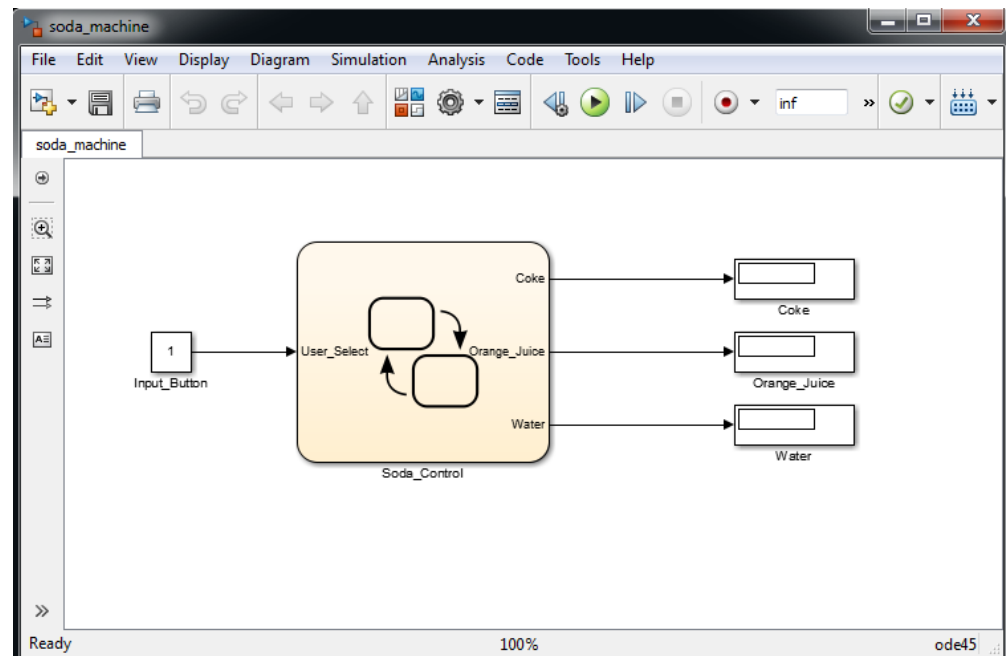
#### 4) Report Generator and Model Comparison

#### 5) Integrating C Code using the Legacy Code Tool

#### 6) MATLAB Coder, Simulink Coder, Embedded Coder

## Demo: Flow Graph Soda Machine

- 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.

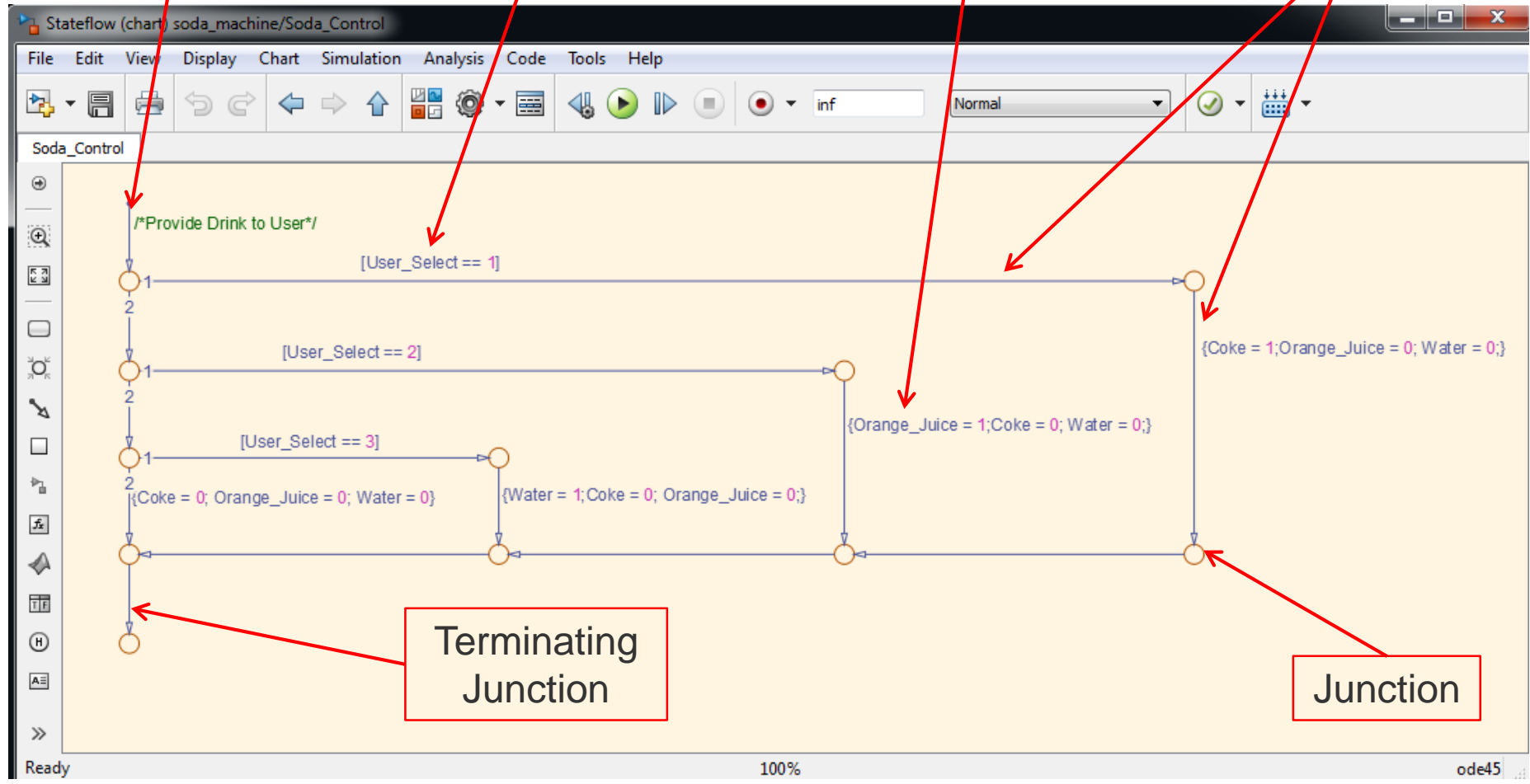


Default Transition

Condition

Action

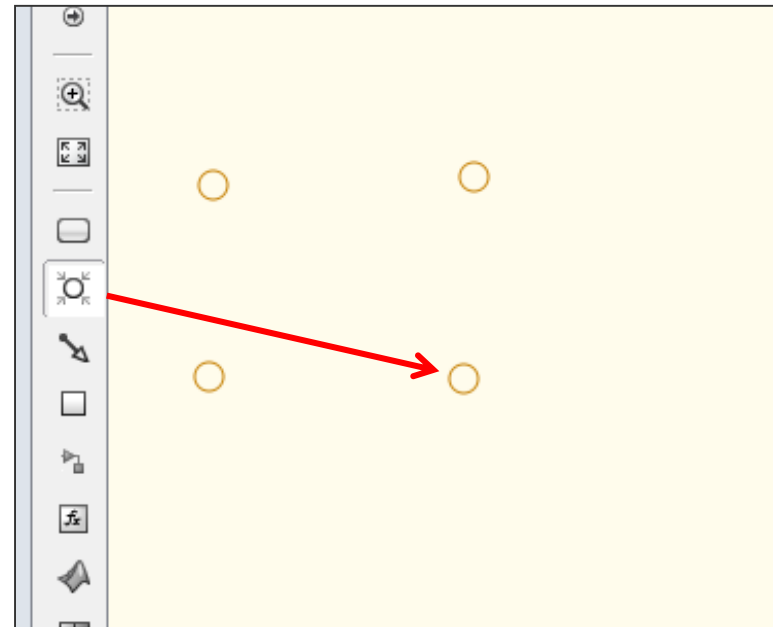
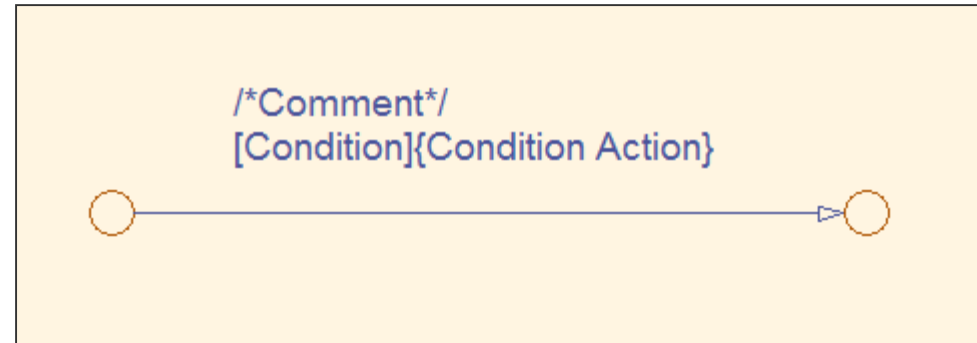
Transitions



Terminating Junction

Junction

- First Condition, then Action!
- Double Click keeps buttons pressed



The screenshot displays the Simulink Model Explorer interface. On the left, the Model Hierarchy shows the project structure, including 'Soda\_Control' under 'soda\_machine'. The main pane shows the contents of 'Soda\_Control' in Stateflow mode, listing four objects: 'User\_Select' (Input, int8), 'Coke' (Output, boolean), 'Orange\_Juice' (Output, boolean), and 'Water' (Output, boolean). The right pane shows the configuration for the 'Data User\_Select' object. The 'Scope' dropdown menu is open, showing options: 'Input', 'Local', 'Constant', 'Parameter', and 'Data Store Memory'. The 'Input' option is selected and highlighted. A red circle highlights the 'Scope' dropdown and the 'Complexity' dropdown, which is set to 'Input'. Other configuration options include 'Name' (User\_Select), 'Port' (1), 'Type' (int8), and 'Watch in debugger' (unchecked).

Name	Scope	Port	Resolve Signal	DataType	Size	Initial
User_Select	Input	1		int8		
Coke	Output	1	<input type="checkbox"/>	boolean		
Orange_Juice	Output	2	<input type="checkbox"/>	boolean		
Water	Output	3	<input type="checkbox"/>	boolean		

## 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!**

### 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

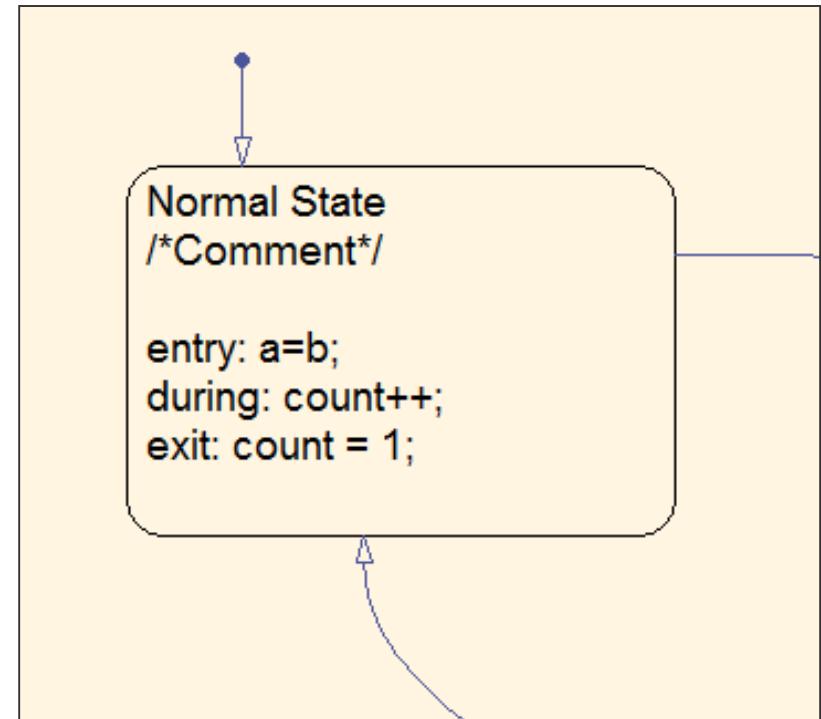
#### 4) Report Generator and Model Comparison

#### 5) Integrating C Code using the Legacy Code Tool

#### 6) MATLAB Coder, Simulink Coder, Embedded Coder

## State Charts

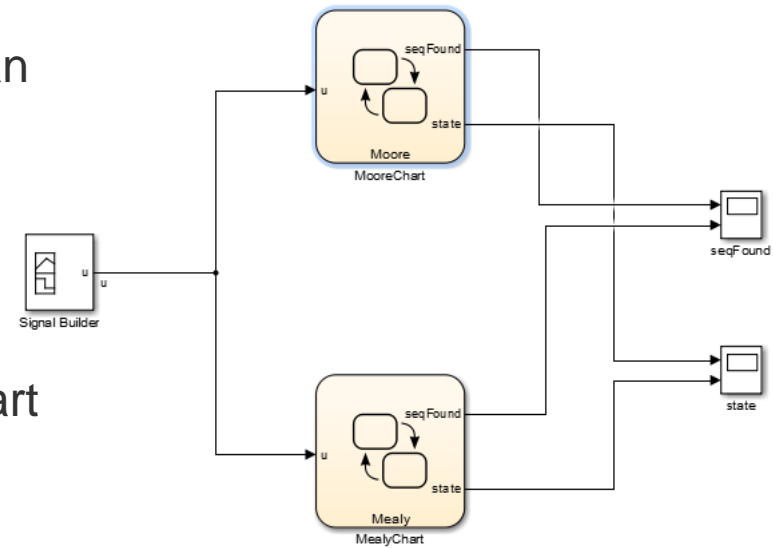
- State charts have a internal behavior and internal data
- Actions can be performed on entry, during residence in the state and on exit
- A state can perform a self transition
- A state can be either active or passive





## 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)



Copyright 2006-2009 The MathWorks, Inc.

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

## 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

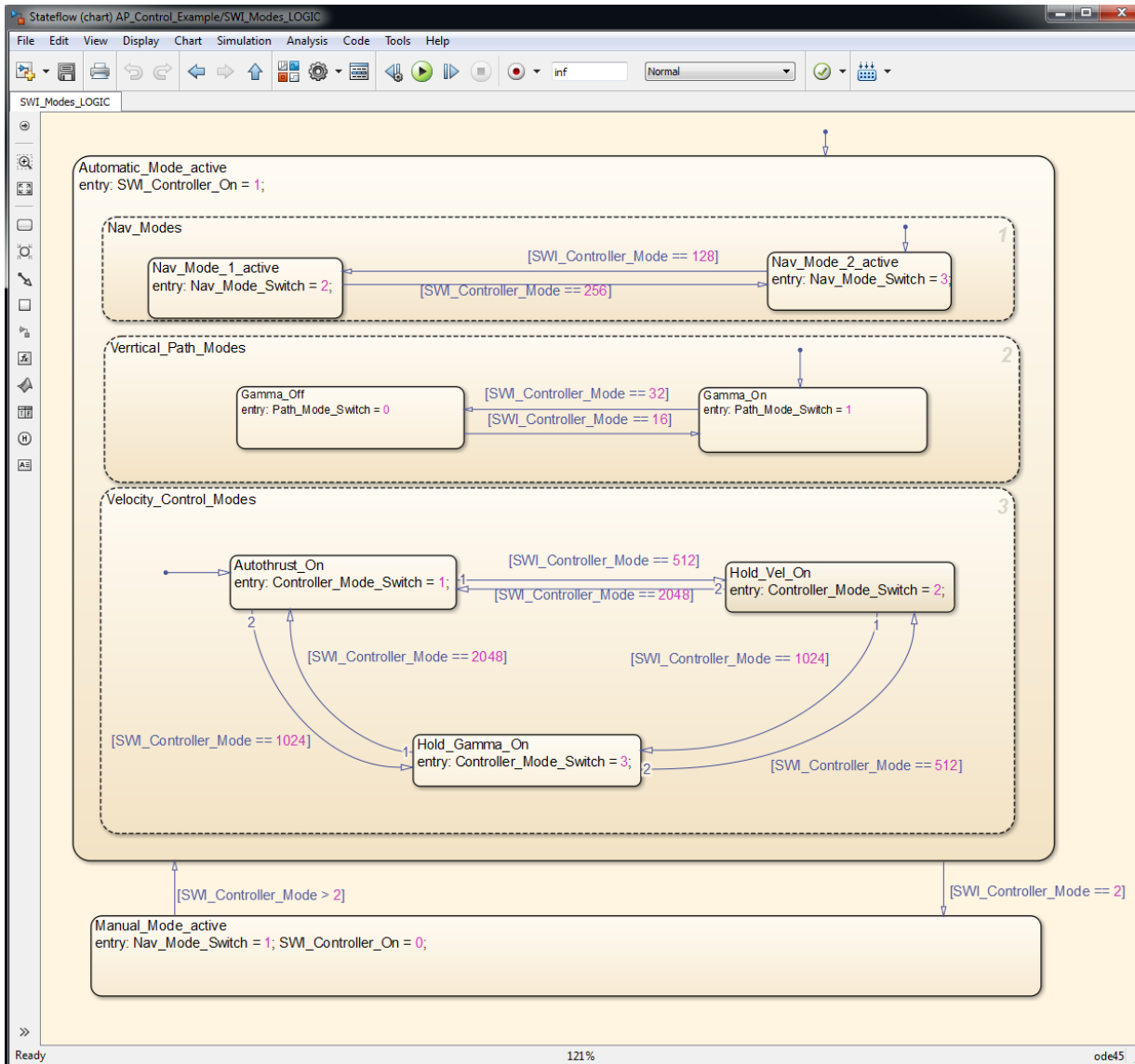
MATLAB Help -> Stateflow -> Chart Programming -> Chart Programming Basics -> Concepts -> Action Language Auto Correction

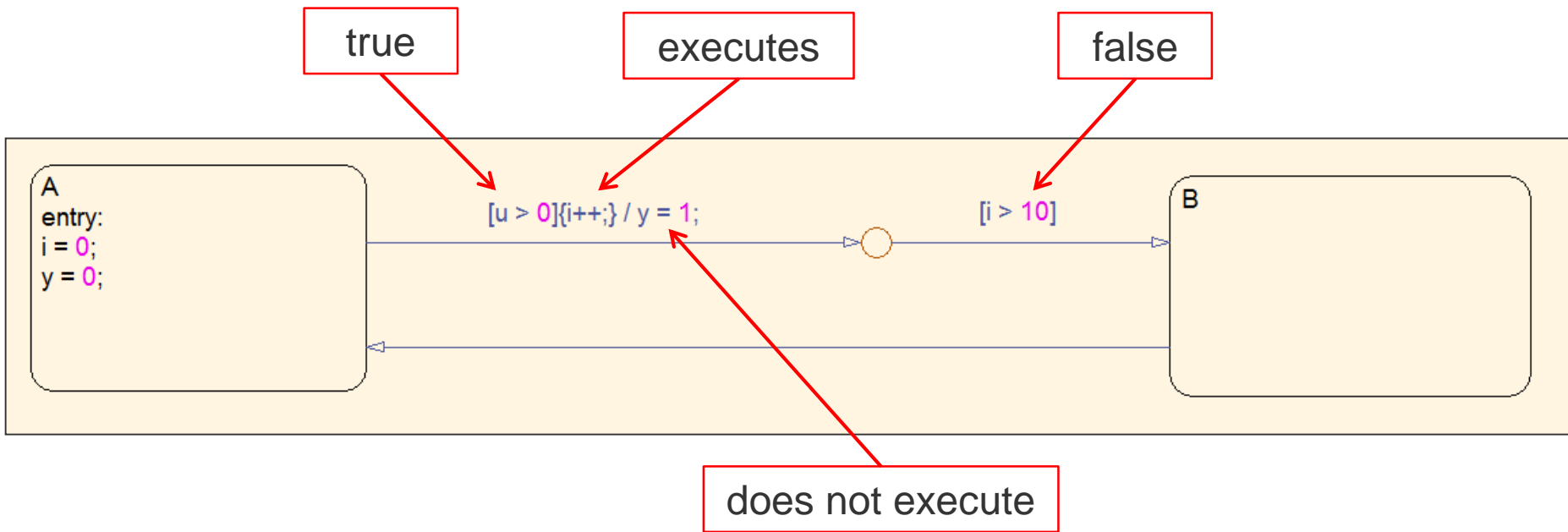
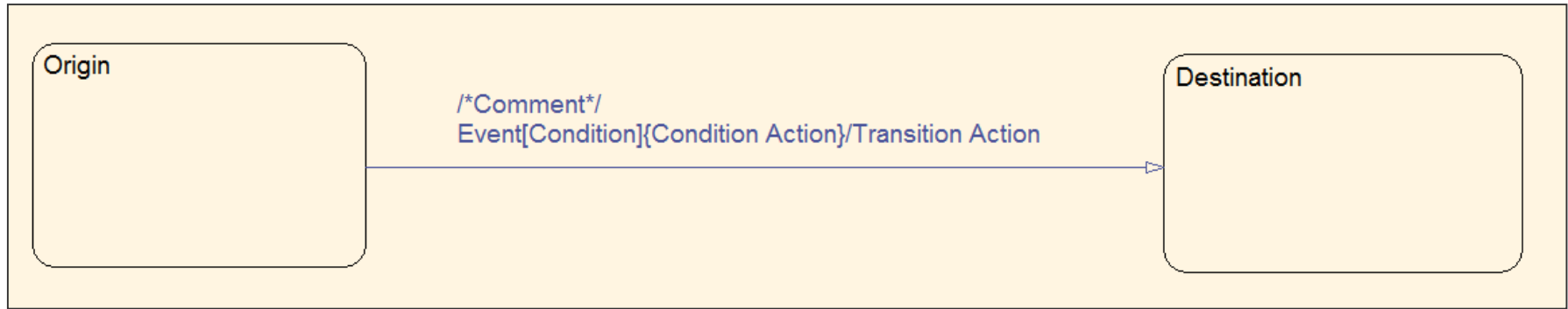
The screenshot shows the configuration dialog for a Stateflow chart named 'shift\_logic'. The 'General' tab is active, displaying the following settings:

- Name: `shift_logic`
- Machine: `(machine) sf_car`
- Action Language: `MATLAB` (selected from a dropdown menu)
- State Machine Type: `Classic` (selected from a dropdown menu)
- Update method: `Discrete` (selected from a dropdown menu)
- Sample Time: `0.04`
- User specified state/transition execution order
- Export Chart Level Functions (Make Global)
- Execute (enter) Chart At Initialization
- Initialize Outputs Every Time Chart Wakes Up
- Enable Super Step Semantics
- Support variable-size arrays
- Saturate on integer overflow

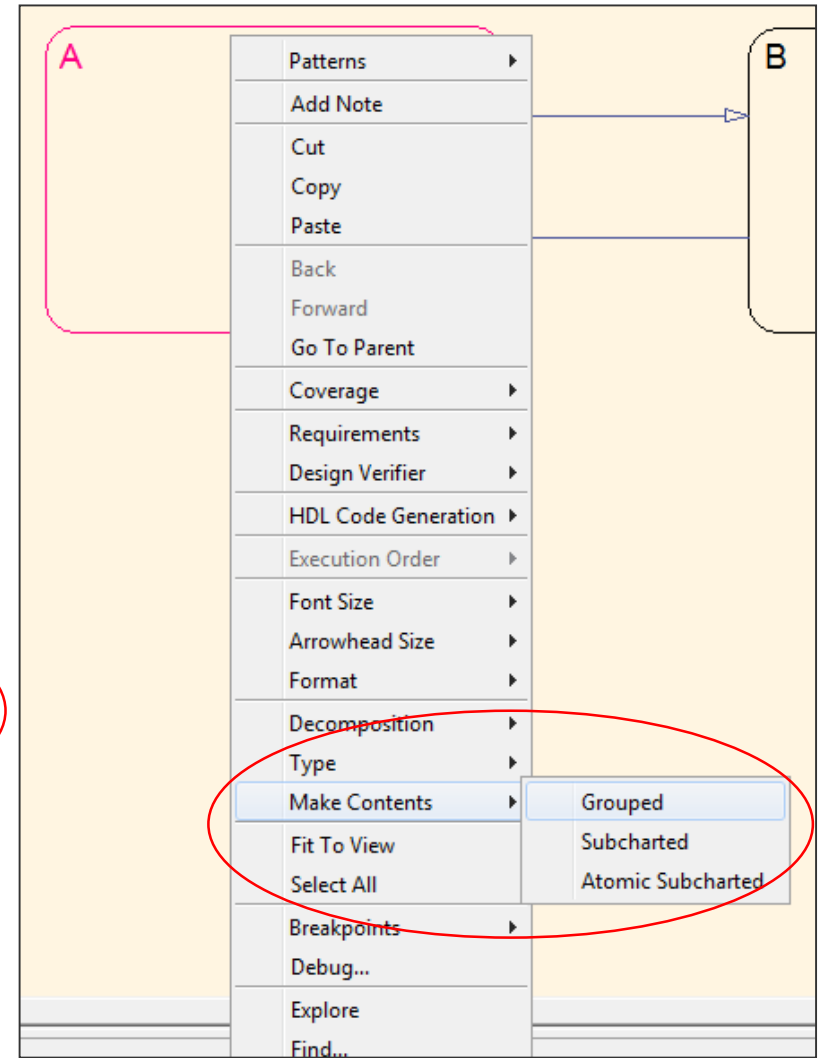
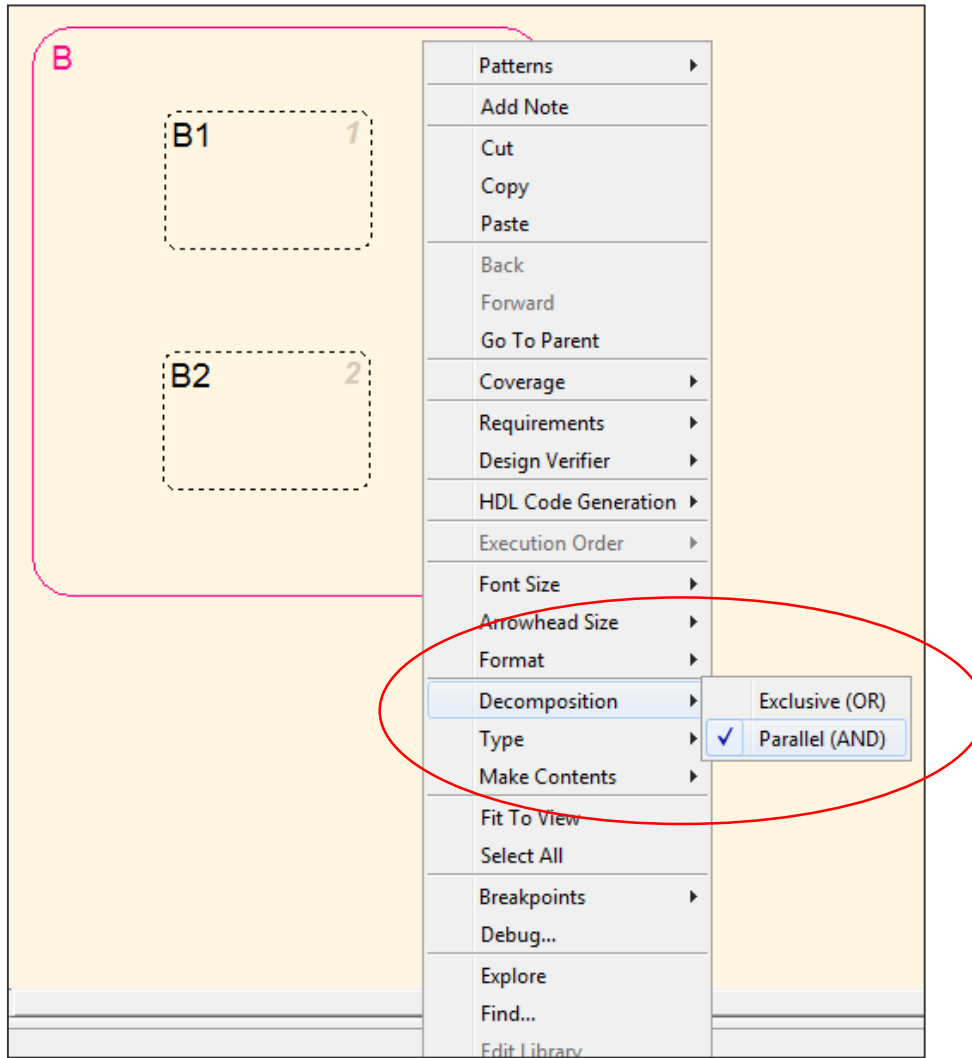
# Stateflow – State Charts

## Demo: Autopilot Mode Control



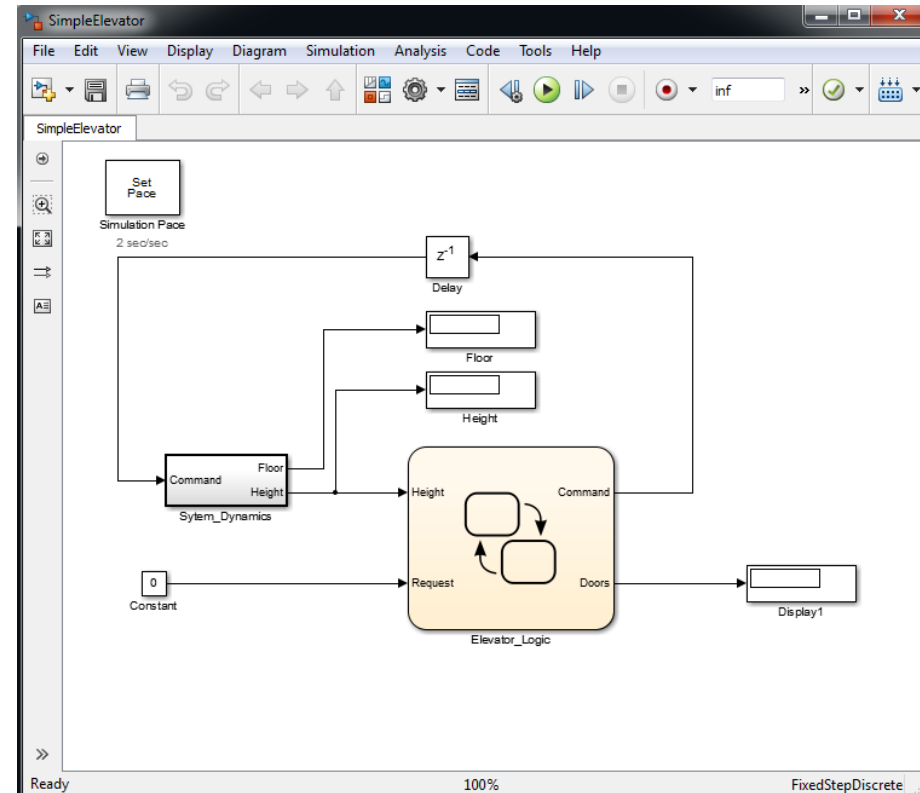


## Parallel Charts and Hierarchical Charts



## Exercise: Elevator

- States: InitialState, 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



### 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

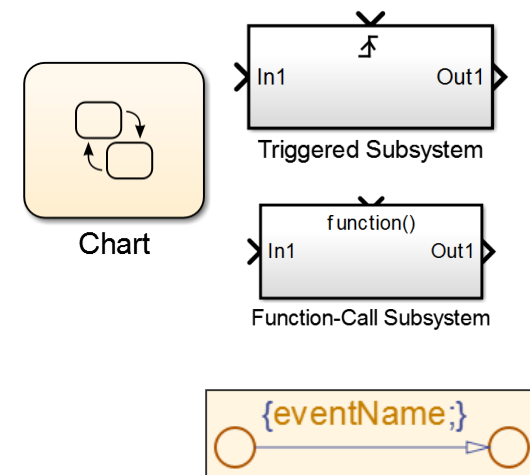
#### 3) Model Advisor

#### 4) Report Generator and Model Comparison

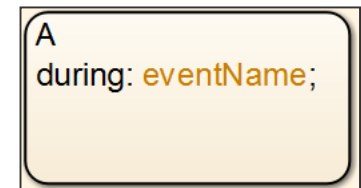
#### 5) Integrating C Code using the Legacy Code Tool

#### 6) MATLAB Coder, Simulink Coder, Embedded Coder

- 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

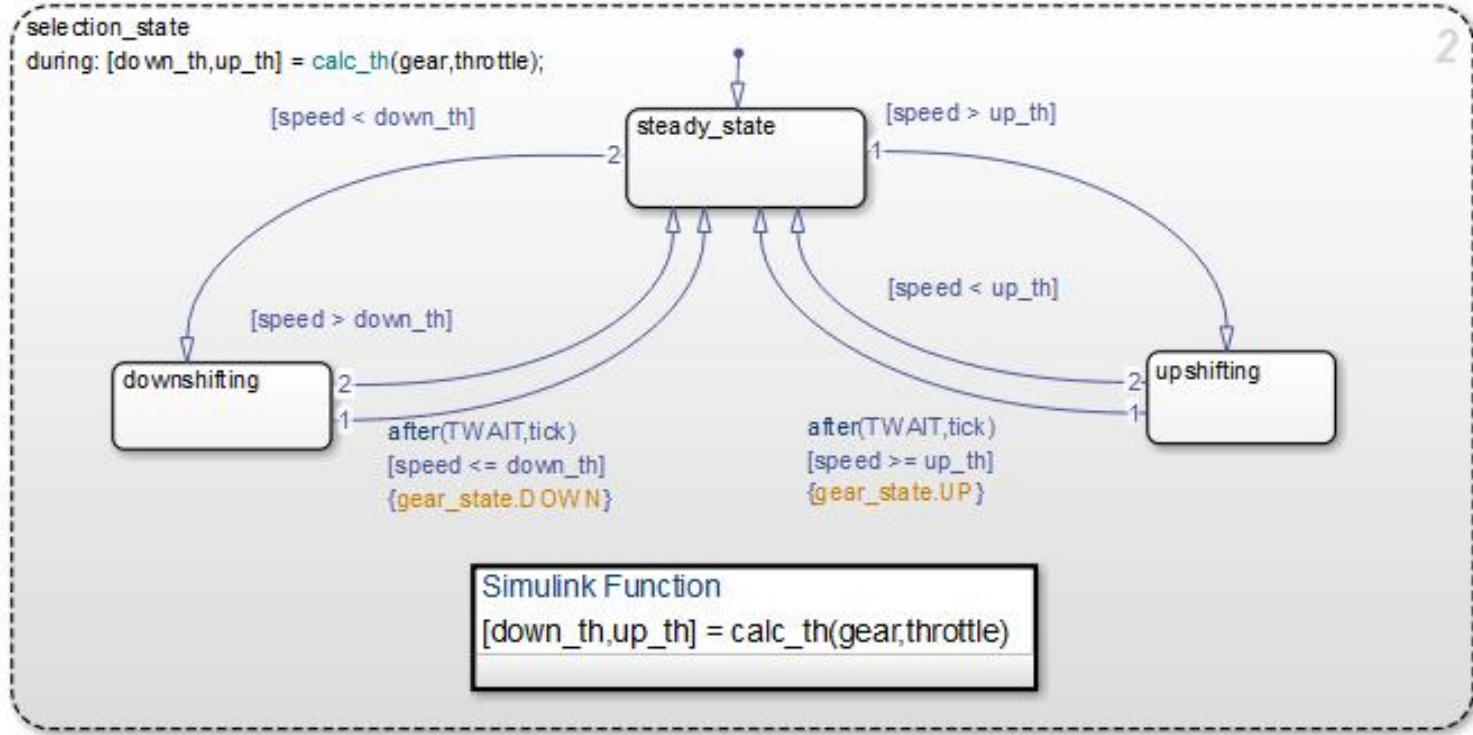
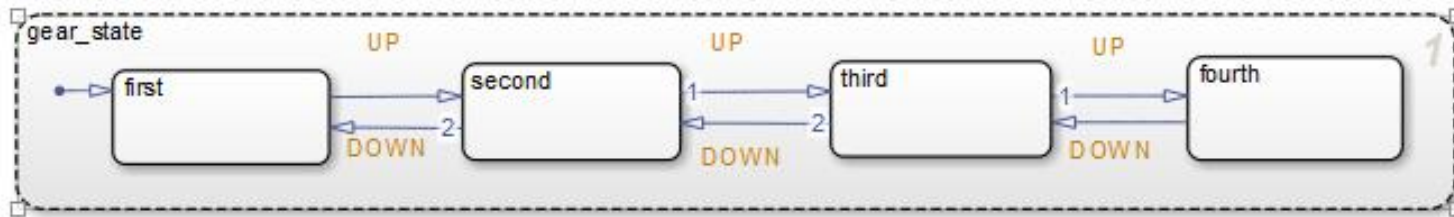


MATLAB Help -> Stateflow -> Chart Programming -> Chart Simulation Semantics -> Concepts -> How Events Drive Chart Execution





## Example: sf\_car



MATLAB Help -> Stateflow->Getting Started with Stateflow -> About Event-Driven System Modeling -> Anatomy of a Stateflow Chart

## Direct and qualified event broadcast

### Direct Broadcast

### Qualified Broadcast

The diagram illustrates a Stateflow state machine with three states: **steady\_state**, **downshifting**, and **upshifting**. The initial state is **steady\_state**.

- Transitions from steady\_state:**
  - To **downshifting**: `[speed < down_th]`
  - To **upshifting**: `[speed > up_th]`
- Transitions to steady\_state:**
  - From **downshifting**: `[speed > down_th]`
  - From **upshifting**: `[speed < up_th]`
- Transitions from downshifting to steady\_state:**
  - Event: `after(TWAIT,tick)`
  - Condition: `[speed <= down_th]`
  - Action: `{/send(DOWN, gear_state)}`
- Transitions from upshifting to steady\_state:**
  - Event: `after(TWAIT,tick)`
  - Condition: `[speed >= up_th]`
  - Action: `{gear_state.UP}`

A **Simulink Function** block is defined as: `[down_th, up_th] = calc_th(gear, throttle)`.

**Model Hierarchy (Left):**

- Simulink Root
  - Base Workspace
  - sf\_car\*
    - Model Workspace
    - Code for sf\_car
    - Simulink Design Verifier results
    - Advice for sf\_car
    - Configuration (Active)
    - Engine
    - Vehicle
    - shift\_logic
      - gear\_state
      - selection\_state
      - transmission

**Contents of: sf\_car/shift\_logic (only):**

Name	Scope	Port	Resolve Signal	DataType
gear_state				
selection_state				
speed	Input	1		double
TWAIT	Parameter			uint8
up_th	Local			double
down_th	Local			double
throttle	Input	2		Inherit: Same as parent
gear	Output	1		Enum: gearType
DOWN	Local			

**Contents of: gear\_state:**

Name	Value
UP	
fourth	
third	
second	
first	
?	
UP	
UP	
UP	
DOWN	
DOWN	
DOWN	

Enables time-dependent logic based on event counts

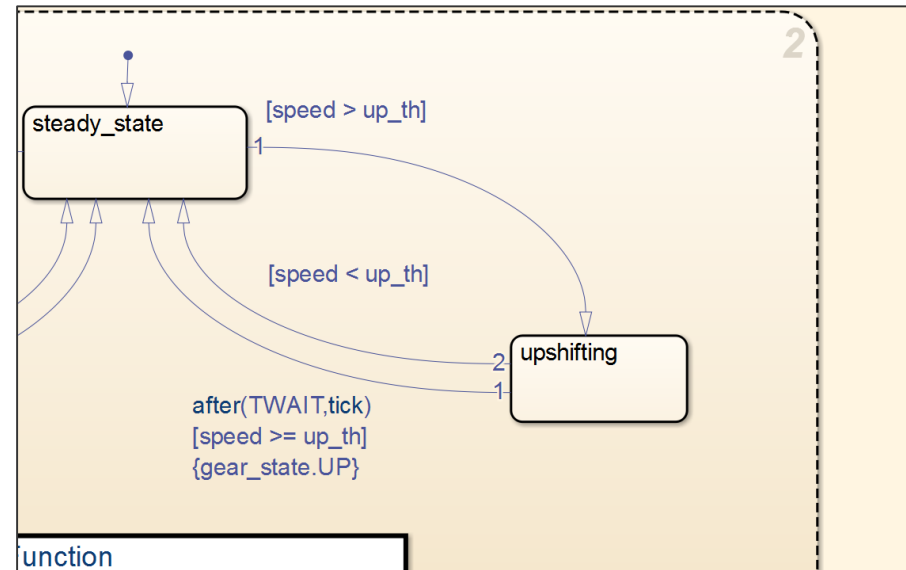
Temporal logic operators:

- $at(n, event)$ : 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

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



# 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  $n^{\text{th}}$  trigger of **event**
- **every(n,event)** – true at every  $n^{\text{th}}$  trigger of **event**
- **after(n,event)** – true after the  $n^{\text{th}}$  trigger of **event**
- **before(n,event)** – true before the  $n^{\text{th}}$  trigger of **event**
- **temporalCount(event)** – returns  $n$  at the  $n^{\text{th}}$  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

### Basics:

- 1) Simulink
  - Basics
  - Continuous Models
  - Discrete Models
  - Subsystems
  - Signals
- 2) StateflowFlow
  - 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

The screenshot shows the Simulink Help documentation interface. On the left is a 'Contents' sidebar with a tree view under 'Documentation Center' > 'Simulink'. The 'Modeling Guidelines' section is expanded, showing sub-items like 'Large-Scale Modeling', 'MAAB Control Algorithm Modeling', 'High-Integrity System Modeling', 'Code Generation', 'Complex Logic', 'Physical Modeling', 'Signal Processing', 'Model Upgrades', 'Block Creation', and 'Target Hardware'. The 'High-Integrity System Modeling' item is selected. The main content area features a search bar at the top, followed by the 'Simulink' title and navigation tabs for 'Getting Started', 'Examples', and 'Release Notes'. Below these are several expandable sections: 'Modeling' (Design models of time-varying systems), 'Simulation' (Run systems, review results, validate system behavior), 'Performance' (Optimize performance for specific goals, accelerate simulation speed), 'Component-Based Modeling' (Model architecture for large-scale modeling, component reuse, and team-based projects), 'Modeling Guidelines' (Application-specific guidelines for model architecture, design, and configuration), 'Block Creation' (Create new types of blocks to extend modeling functionality using MATLAB®, C/C++, and Fortran code), and 'Target Hardware' (Run Simulink® models on single-board computers and educational hardware). At the bottom of the main area are tabs for 'Simulink Blocks', 'MATLAB Functions', 'Classes', 'Model Checks', and 'PDF Documentation'. The footer contains copyright information '© 1994-2013 The MathWorks, Inc.' and links for 'Terms of Use', 'Patents', 'Trademarks', and 'Acknowledgments'. A blue-bordered box at the bottom right contains the text 'Help => Simulink => Modeling Guidelines'. The address bar at the bottom left shows the file path: 'file:///C:/Program Files/MATLAB/R2013a/help/simulink/high-integrity-systems.html'.

file:///C:/Program Files/MATLAB/R2013a/help/simulink/high-integrity-systems.html

Help => Simulink => Modeling Guidelines

### Basics:

#### 1) Simulink

- Basics
- Continuous Models
- Discrete Models
- Subsystems
- Signals

#### 2) Stateflow

- Flow Charts
- State Charts
- Events

### Advanced:

#### 1) Libraries and Model Reference

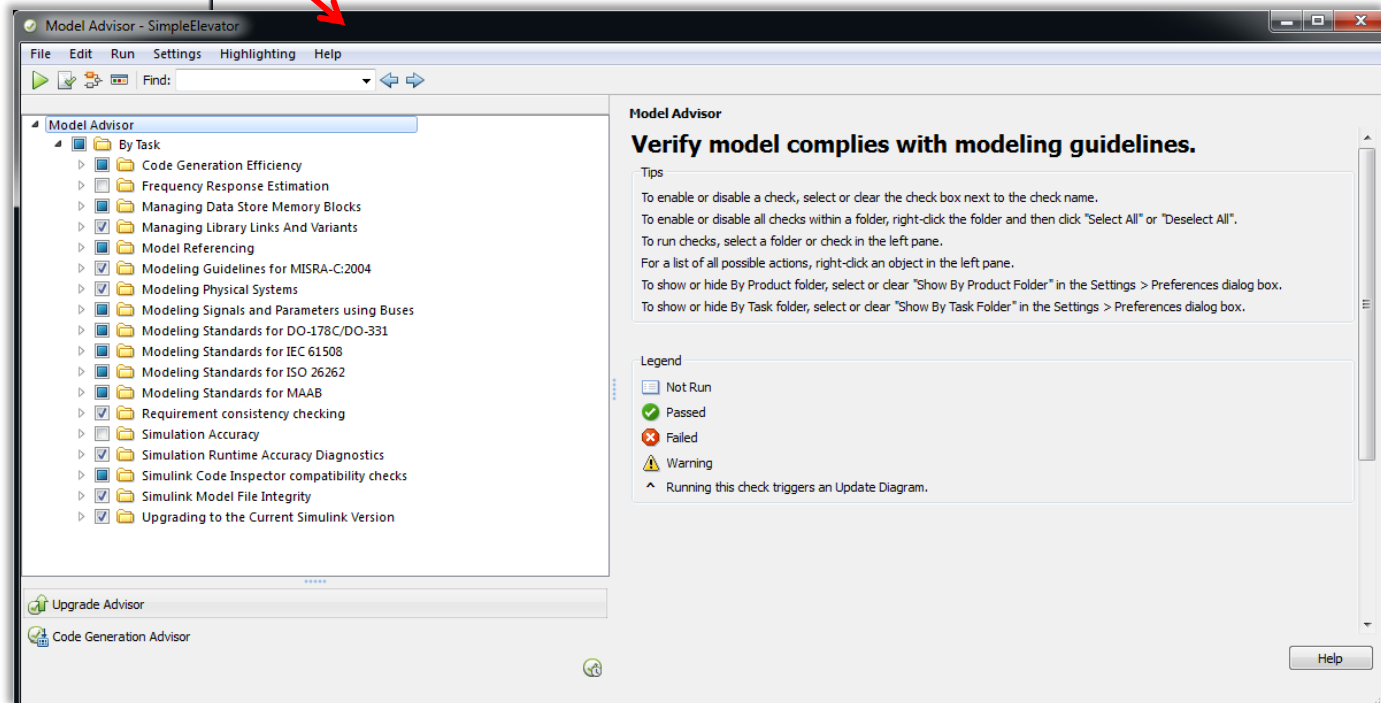
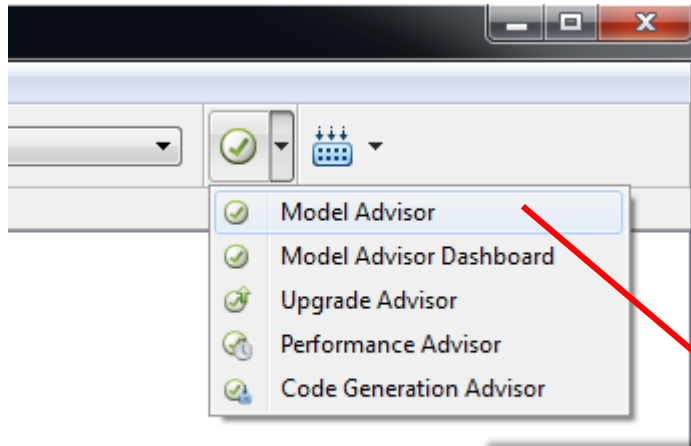
#### 2) Style Guidelines

#### 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



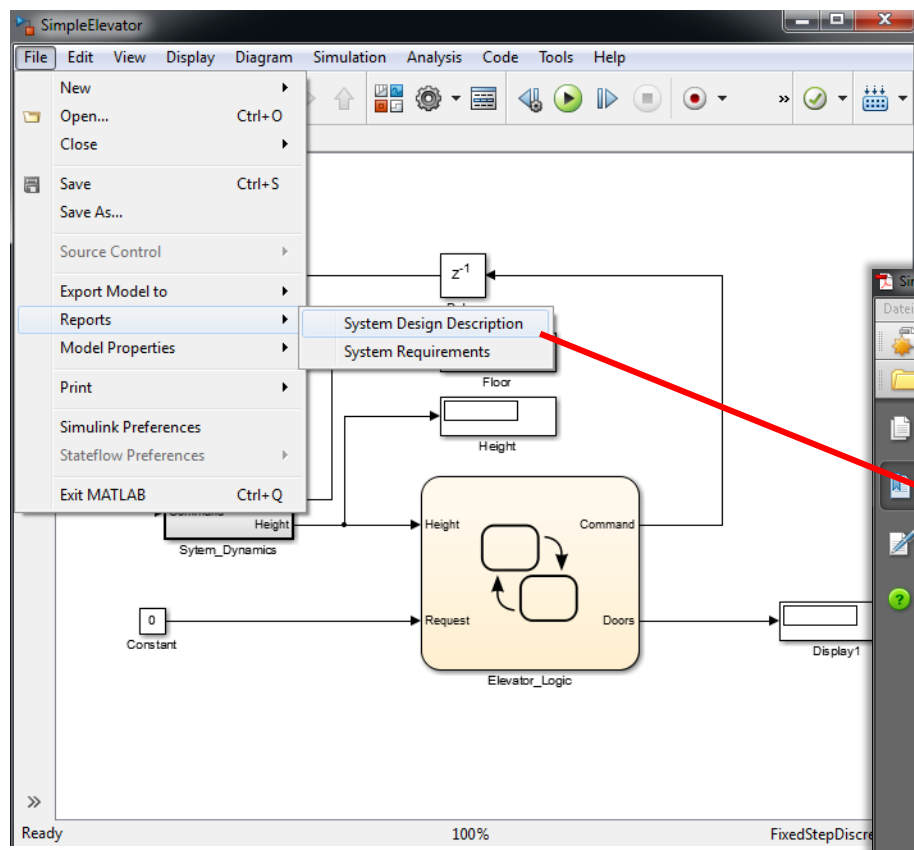


### Basics:

- 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



**Kapitel 2. Root System**

**Inhaltsverzeichnis**

- 2.1. Blocks ..... 2
  - 2.1.1. Parameters ..... 2
- 2.2. State Charts ..... 5
  - 2.2.1. Chart ..... 5
  - 2.2.2. States ..... 6
  - 2.2.3. Data ..... 6

**Abbildung 2.1. SimpleElevator**

**2.1. Blocks**

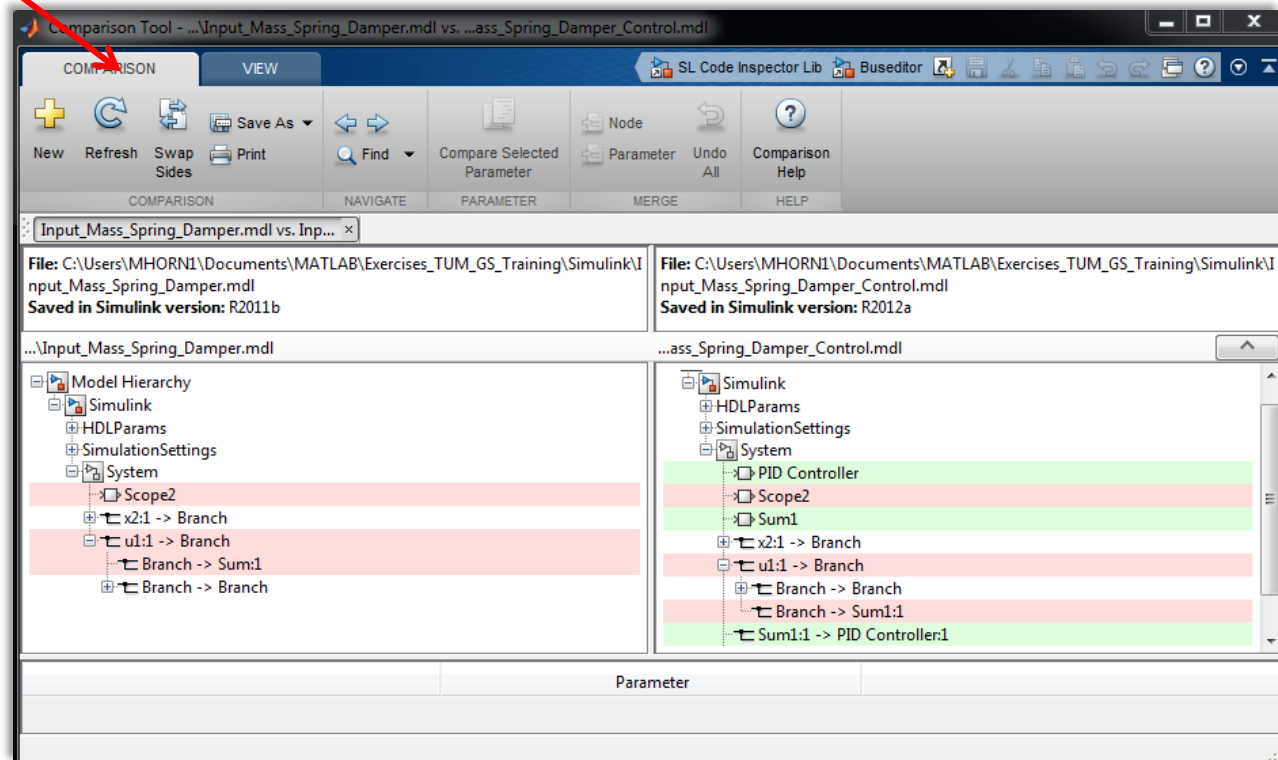
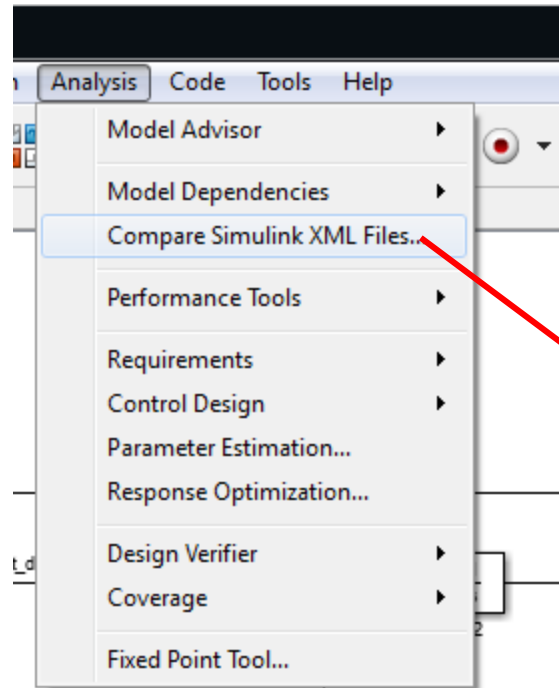
**2.1.1. Parameters**

**2.1.1.1. "Constant" (Constant)**

**Tabelle 2.1. "Constant" Parameters**

Parameter	Value

## Compare XML Files



### 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

## Introducing S-Functions

- 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

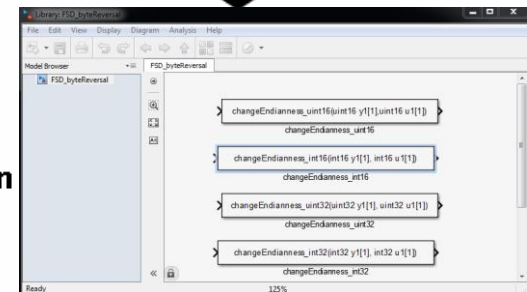
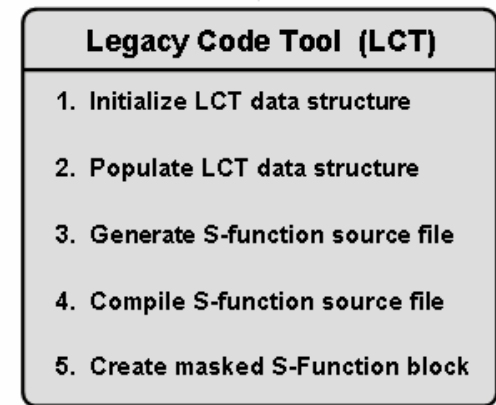
## Demo: Legacy Code Tool

- 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

**Legacy C code**

```
function.c
#include "function.h"
int function(int val)
{
    return(val);
}

function.h
#define _FUNCTION_H_
int function(int val);
#endif
```



**C MEX S-function**

**S-Function block**

**Help => „Integrating Existing C Functions into Simulink Models with the Legacy Code Tool”**

### Basics:

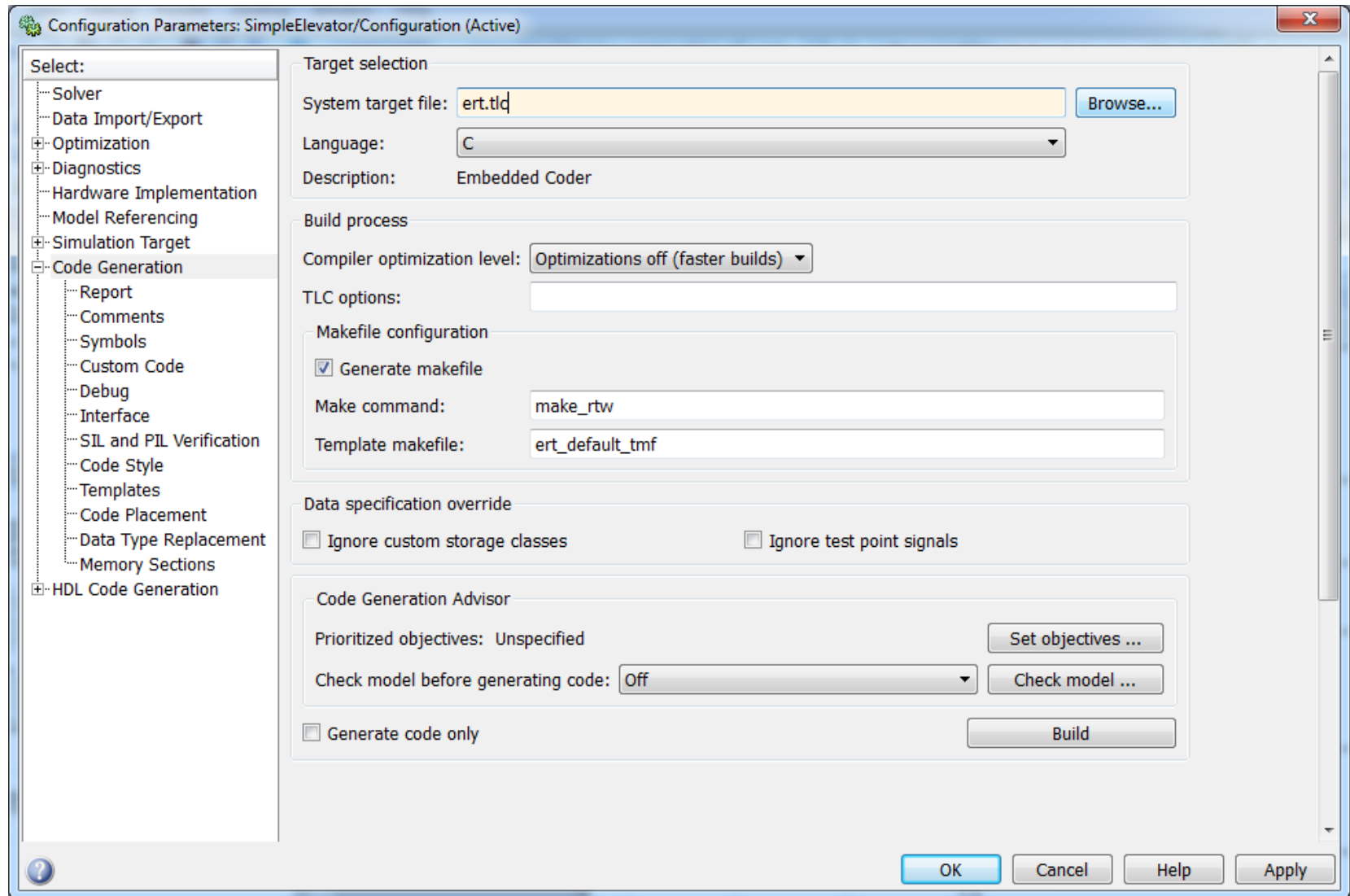
- 1) Simulink
  - Basics
  - Continuous Models
  - Discrete Models
  - Subsystems
  - Signals
- 2) Stateflow
  - Flow Charts
  - State Charts
  - Events

### 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

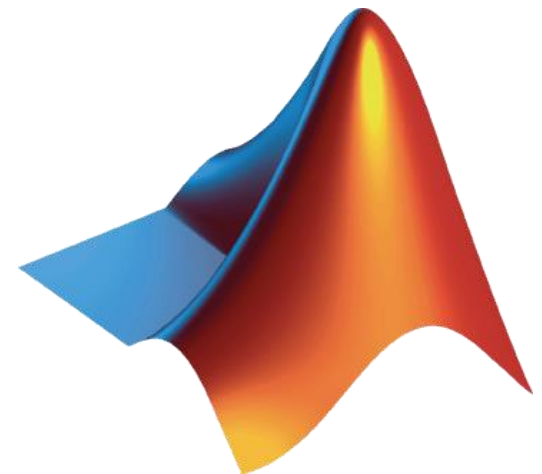






# Summary


- 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

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



**samoconsult GmbH**  
safety | modeling | consulting

**Markus Hornauer**  
High Integrity Systems Engineer

Französische Str. 13-14  
D – 10117 Berlin

+49 151 23506683  
markus.hornauer@samoconsult.de

www.samoconsult.de