



Aylin EDGÜ

System Modeling and Control Engineer

aylin.edgu@figes.com.tr

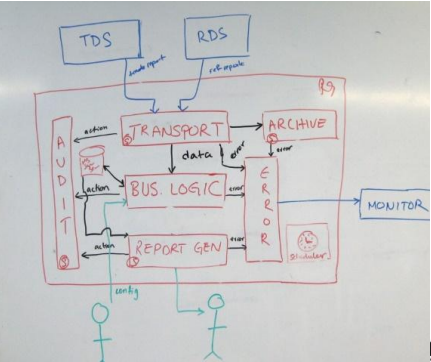
MathWorks Ekosisteminde Model Tabanlı Tasarım



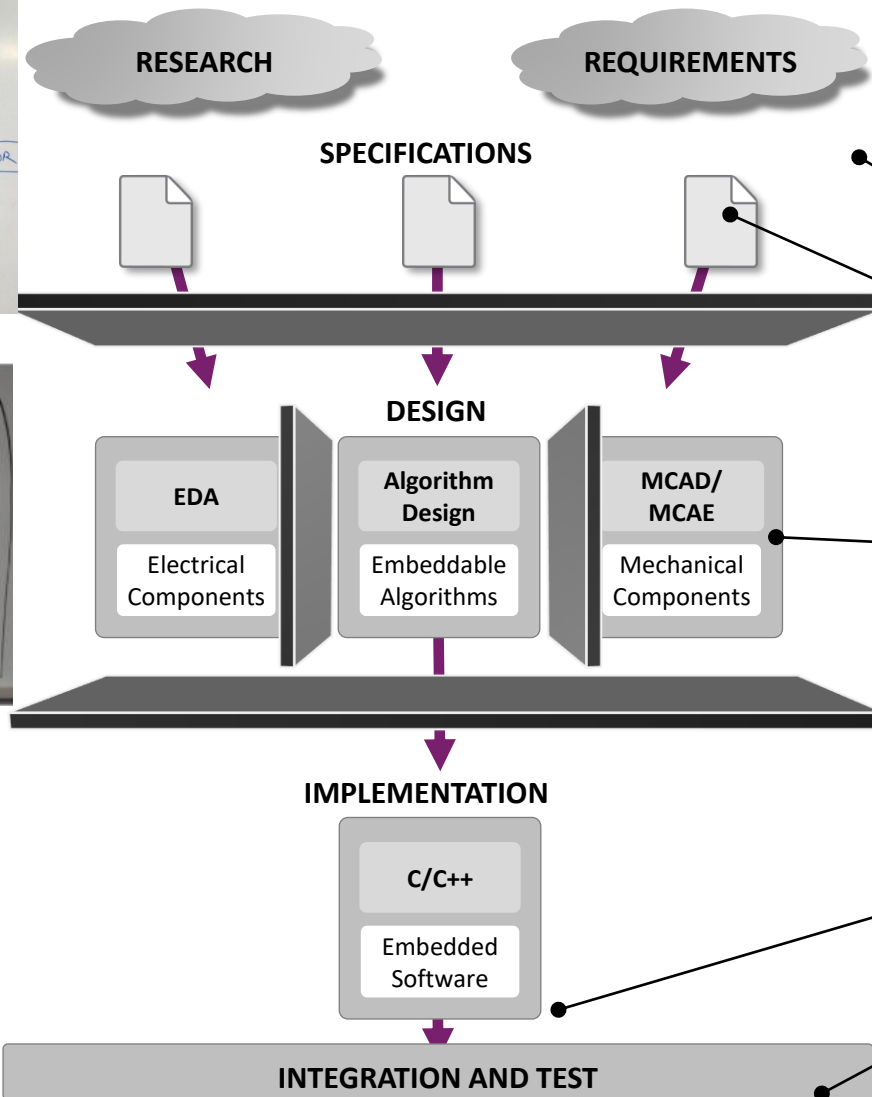
Agenda

- 1. Traditional vs. Model Based Design**
- 2. Model Based Design Overview**
- 3. Simulink Overview**
- 4. Simscape Overview**
- 5. Stateflow Overview**
- 6. Applications with Model Based Design**
- 7. Q&A**

Traditional Development Process



COMPONENT	MASS(kg)	POWER(W)
• COMMUNICATION SUBSYS.	→ 2.83	55
- ADSB	→ 0.05	5
- KV/KR RADIO	→ 0.05	2
- RADIO RX PPM/PWM	→ 2.5	50
• ELECTRICAL SUBSYS.	→ 0.01	0.95
- ACTUATOR POWER	→ 0.02	1
- POWER DISTRIBUTION	33.15	353000
- POWER MONITORING	8	300
- POWER SOURCE	10	1000
- PROPELLION POWER	0.1	1000
- VEHICLE POWER	→ 300	350000
- AUTOPLOT ESTIMATE	50	50
- CEMS ACUATOR	5	0.02
- CEMS ACUATOR	0.05	1.07
- MONITORING CONTROL LOGIC	0.05	2
- AUTOPLOT	3.85	1.09
	0.8	1.150
		1



Manual steps introduce **errors** and **slow down** the development process

Requirement Documents

- Difficult to analyze
- Difficult to manage as they change

Paper Specifications

- Easy to misinterpret
- Difficult to integrate with design

Physical Prototypes

- Incomplete and expensive
- Prevents rapid iteration
- No system-level testing

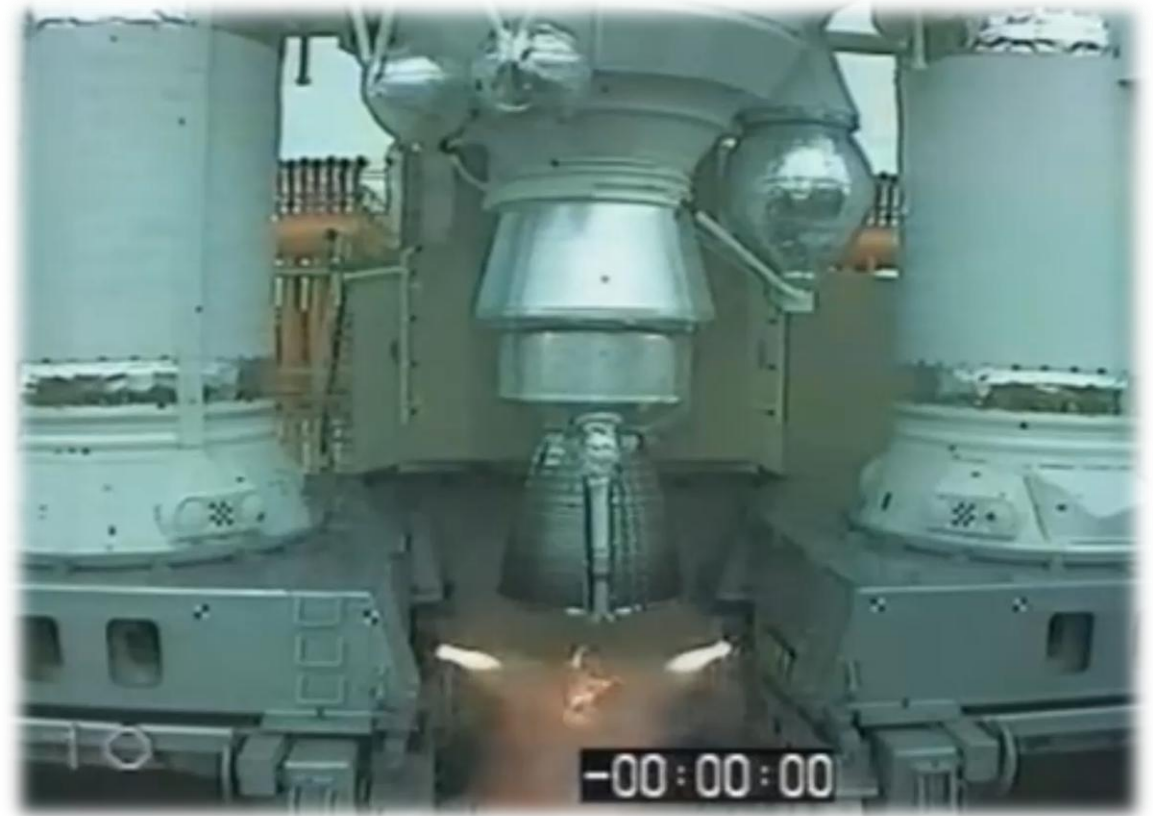
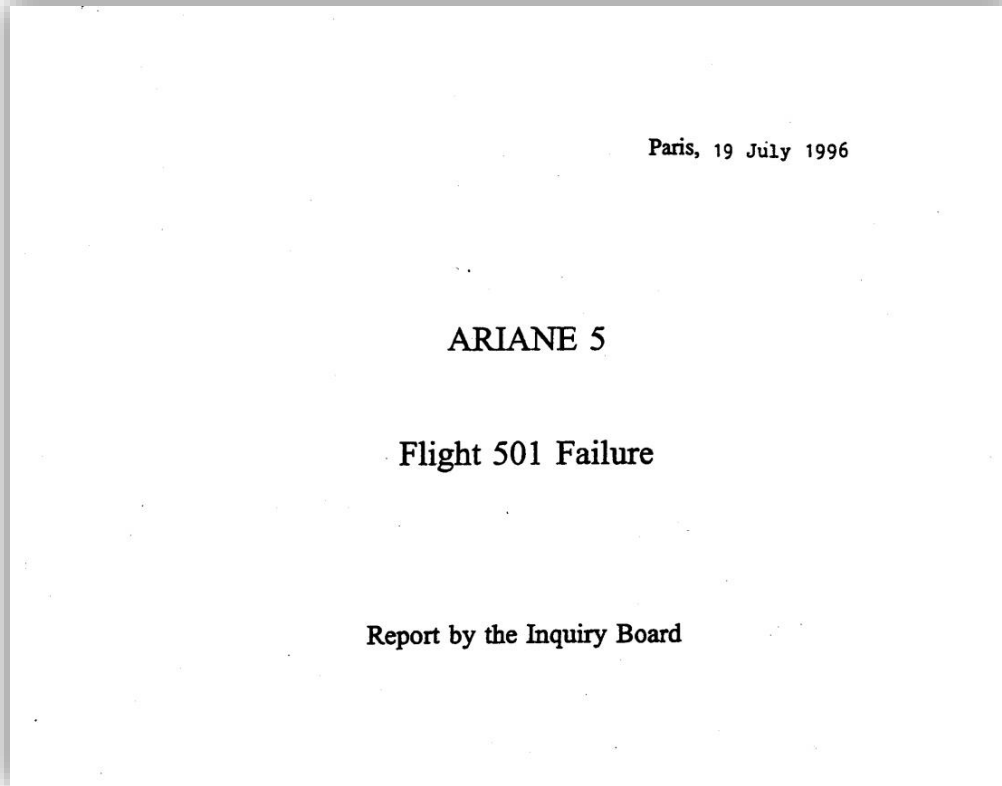
Manual Coding

- Time consuming
- Introduces defects and variance
- Difficult to reuse

Traditional Testing

- Design and integration issues found late
- Difficult to feed insights back into design process
- Traceability

Traditional vs. Model-Based Approach

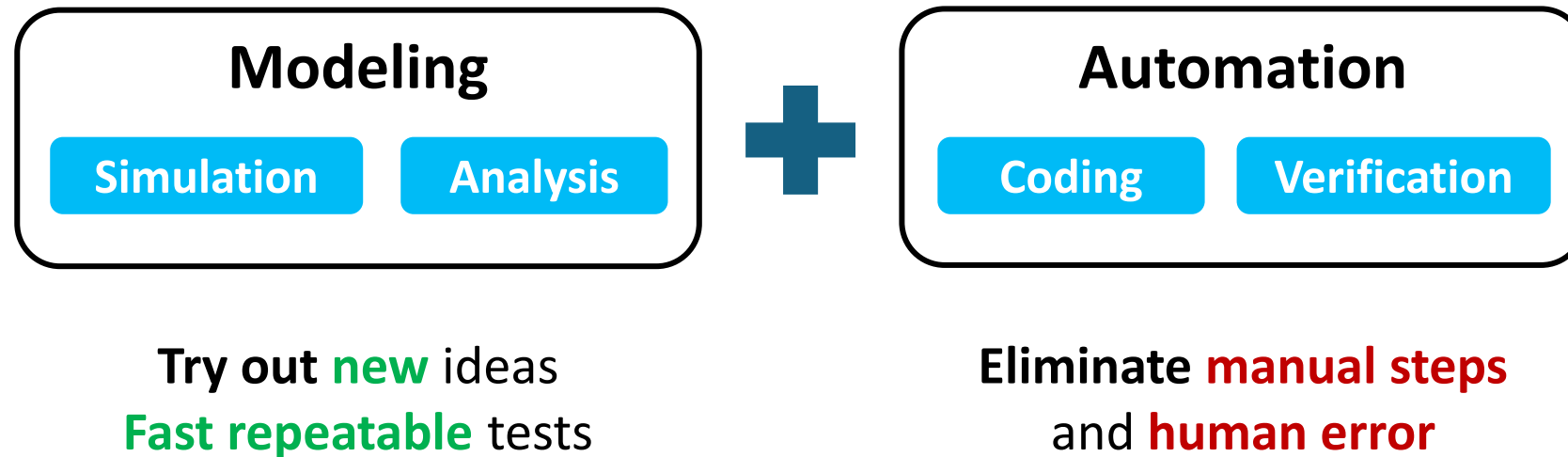


Overflow error

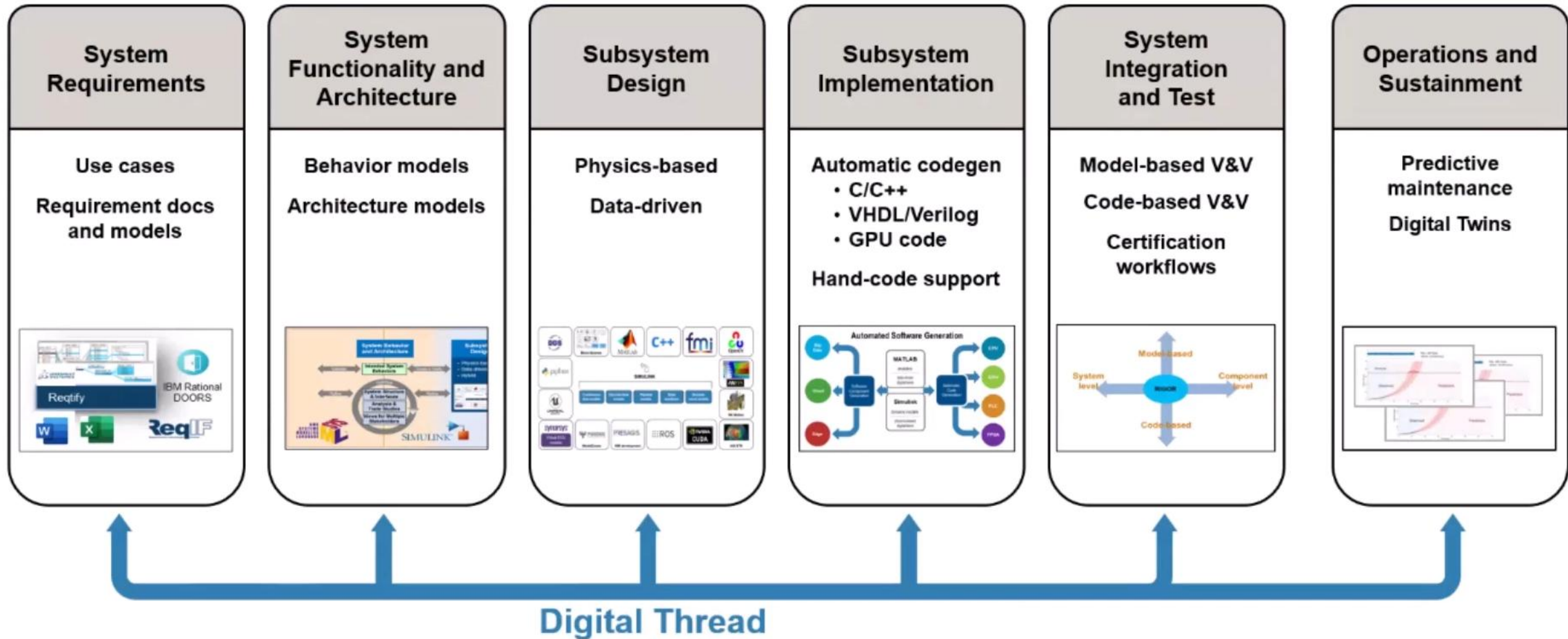


What is Model-Based Design?

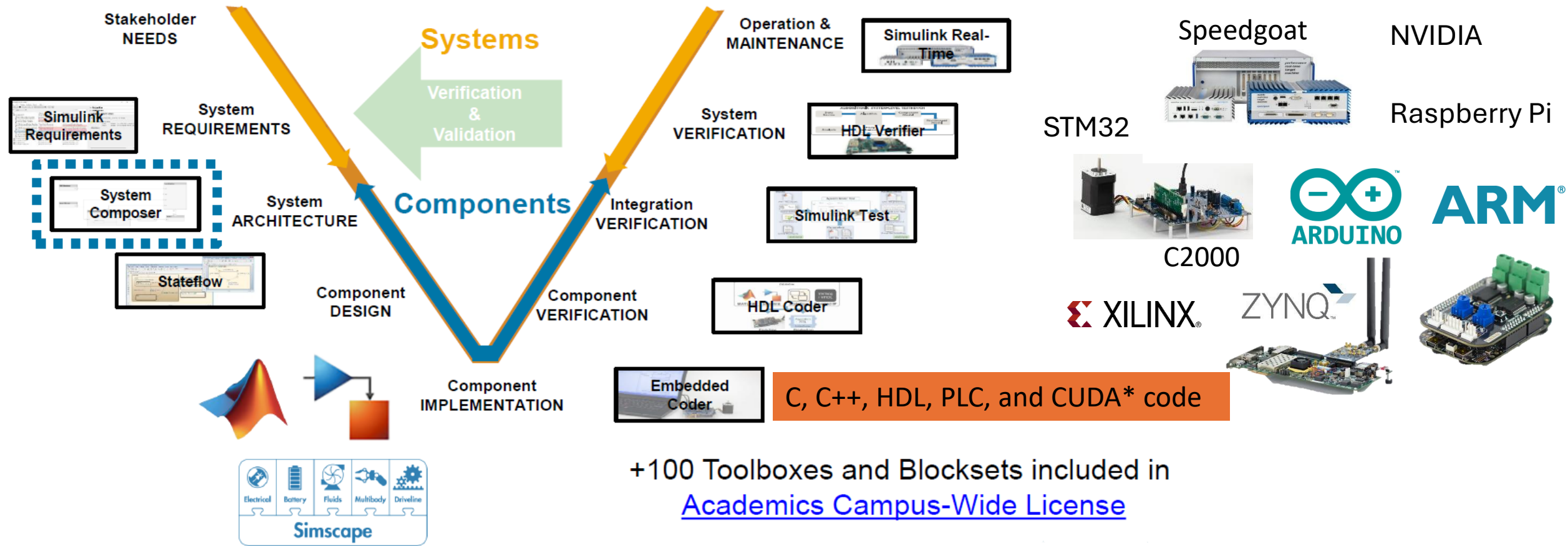
Systematic use of models throughout the development process



Why Model-Based Design?



Model-Based Design with MathWorks Tools



Data-Driven Versus Physics-Based Modeling

Data-driven

- Data sources
 - Experimental data
 - External tool (complexity-reduction exercise)
- Complex behaviour with many unknowns
- Examples
 - Combustion engine
 - Aerodynamics



- Machine Learning Algorithms
- System Identification
- Reduced Order Model (ROM)

Physics-based

- Mathematical paradigms
 - ODEs & DAEs
 - State charts
 - Physical networks
- Behaviour is adequately approximated by limited set of equations
- Examples
 - Algorithms/control
 - Gears & clutches

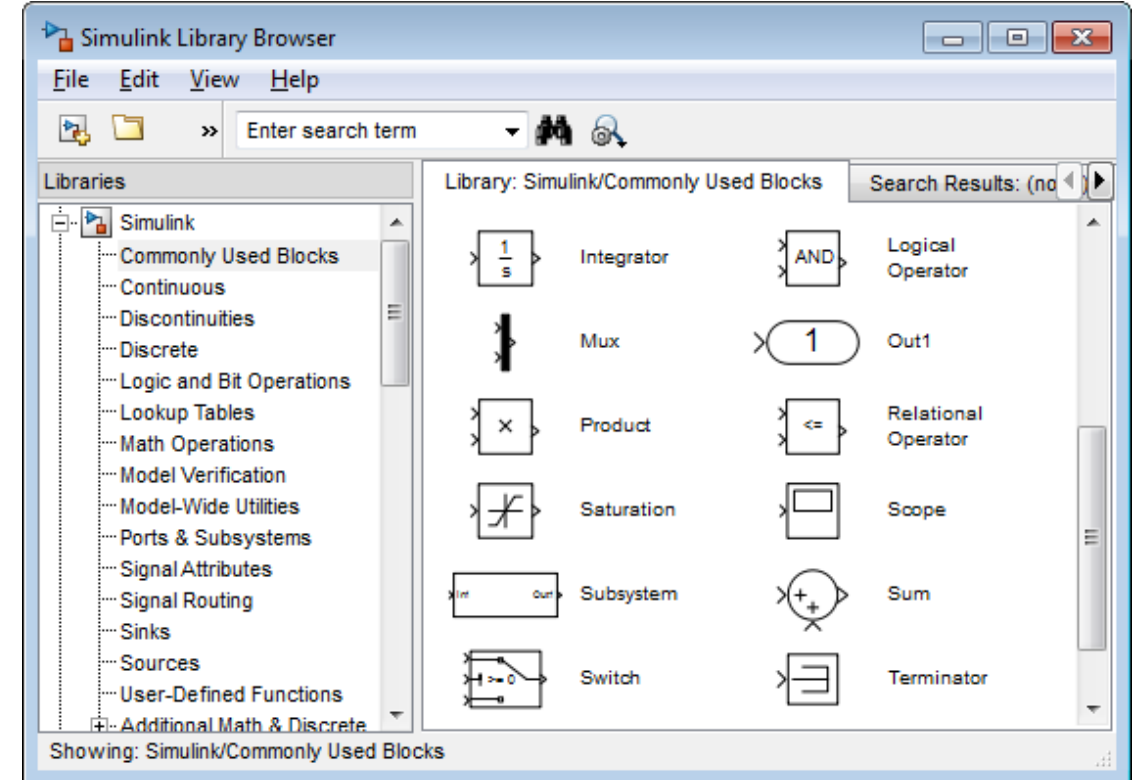


MathWorks Products Overview - Simulink

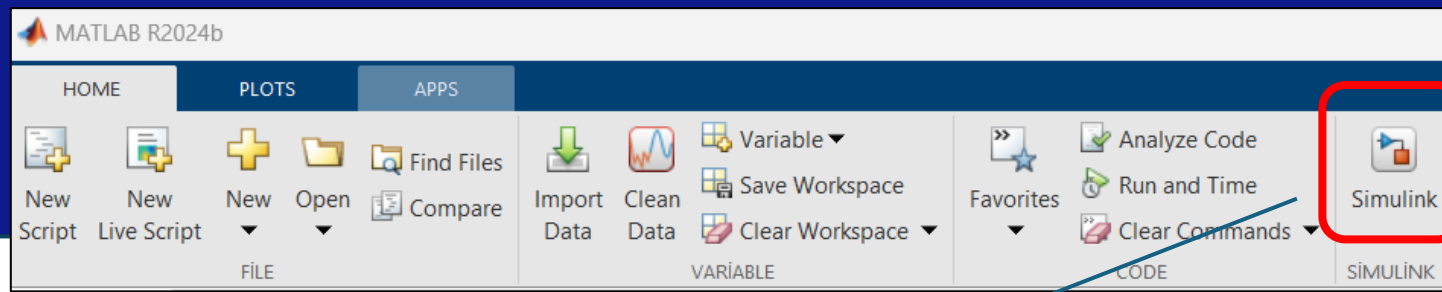


SIMULINK®

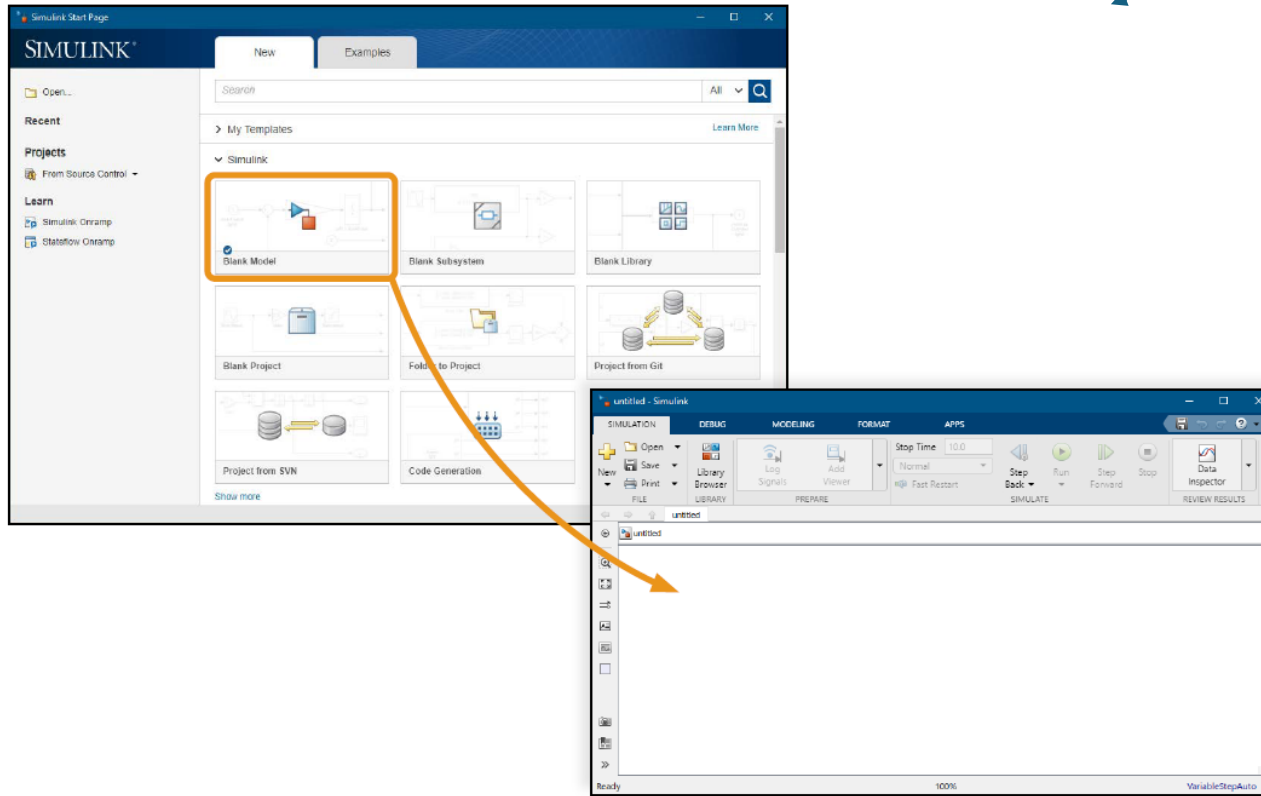
- **Block-diagram** environment(.slx)
- Model, simulate, and analyze multi-domain systems
- Design, implement, and test:
 - **Control systems**
 - **Signal processing systems**
 - **Communications systems**
 - **Other dynamic systems**
- Platform for **Model-Based Design**
- Open architecture with links to **third-party tools** and development boards, and instrumentation



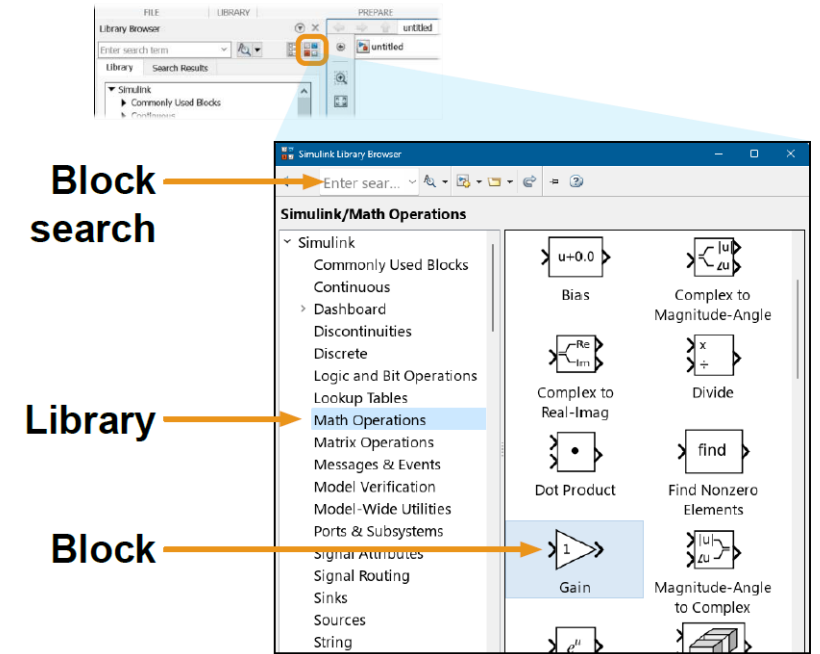
Simulink



Creating a New Simulink Model

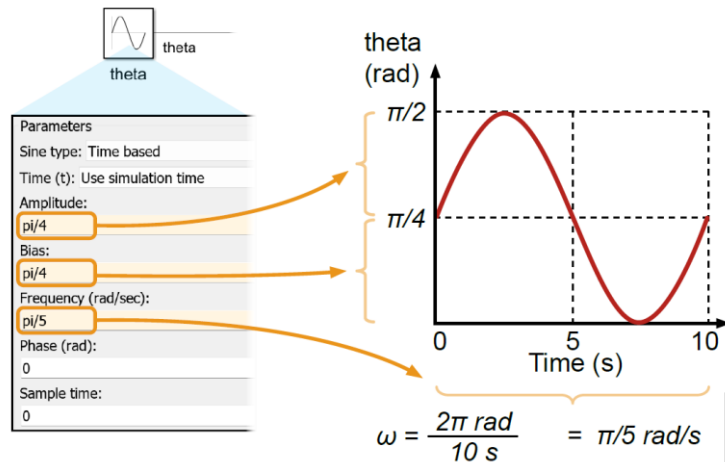


Using the Simulink Library Browser

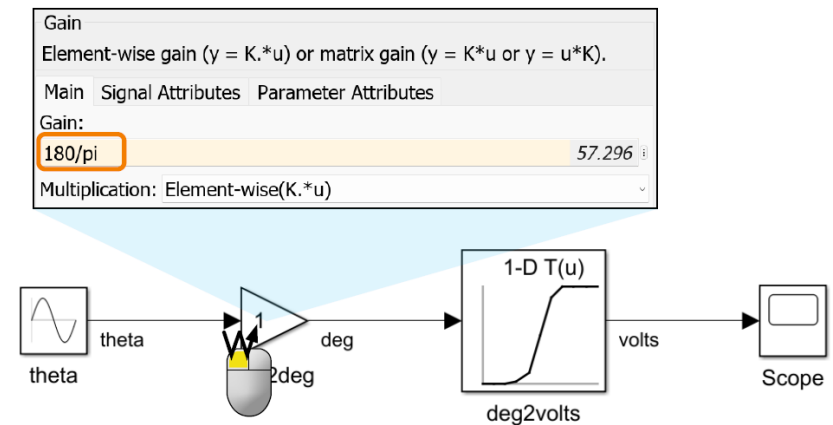


Define Simulink Block Parameters

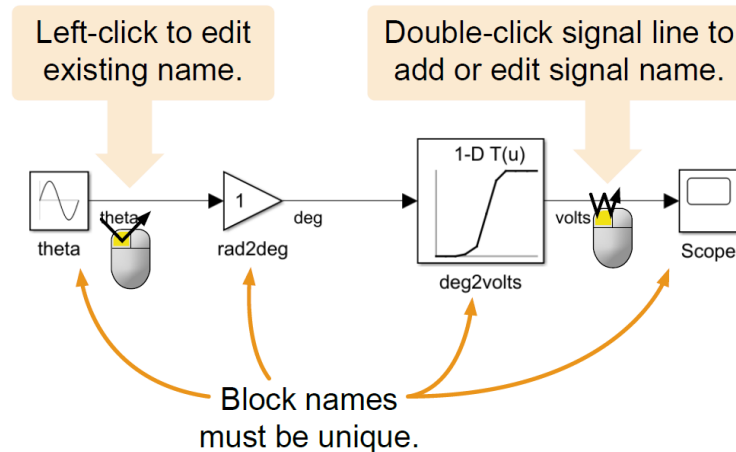
Defining Sine Wave Parameters



Defining Block Parameters



Labeling Blocks and Signals



Numeric value:

Expression:

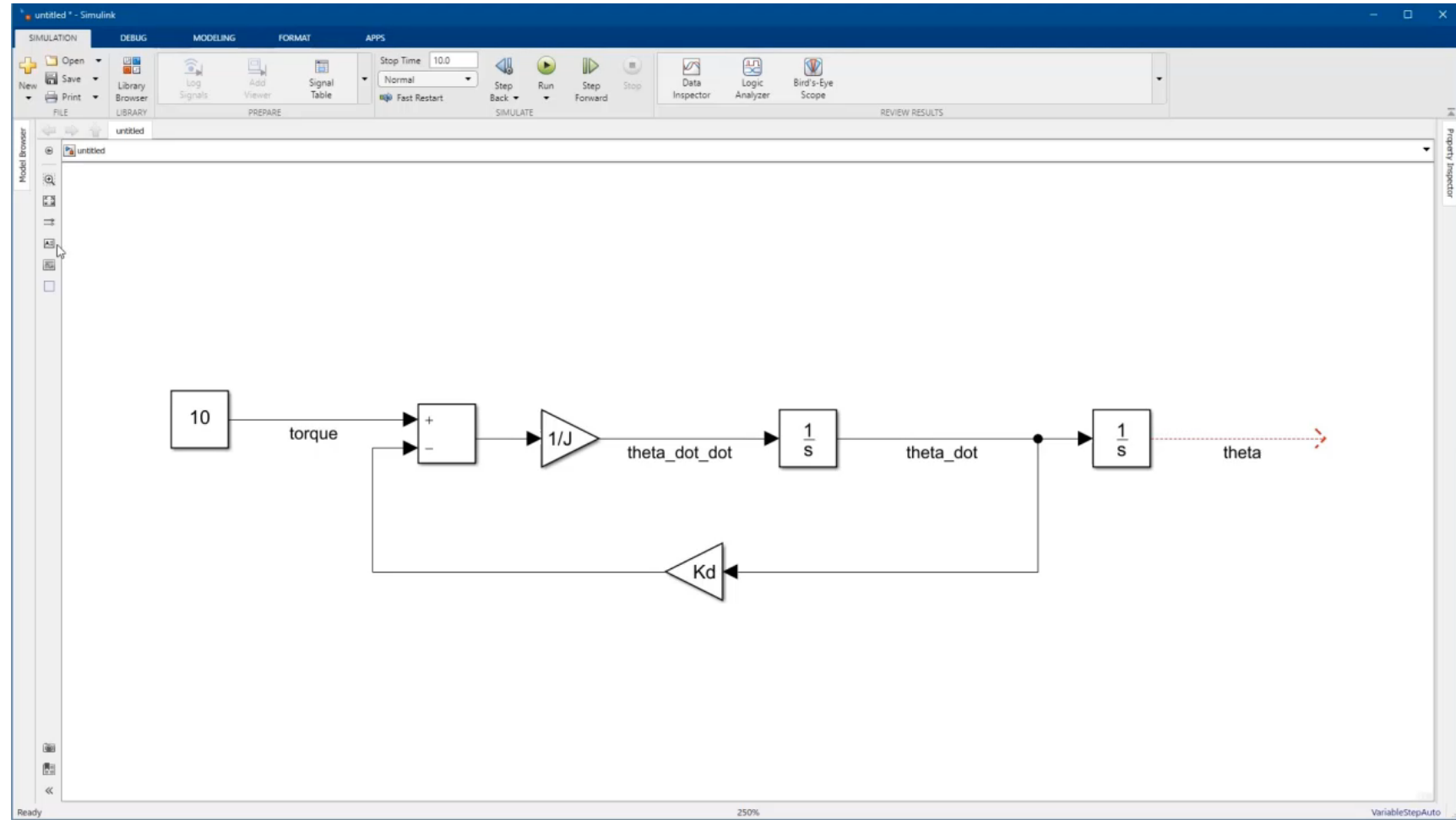
Variable:

Gain:
57.296

Gain:
180/pi

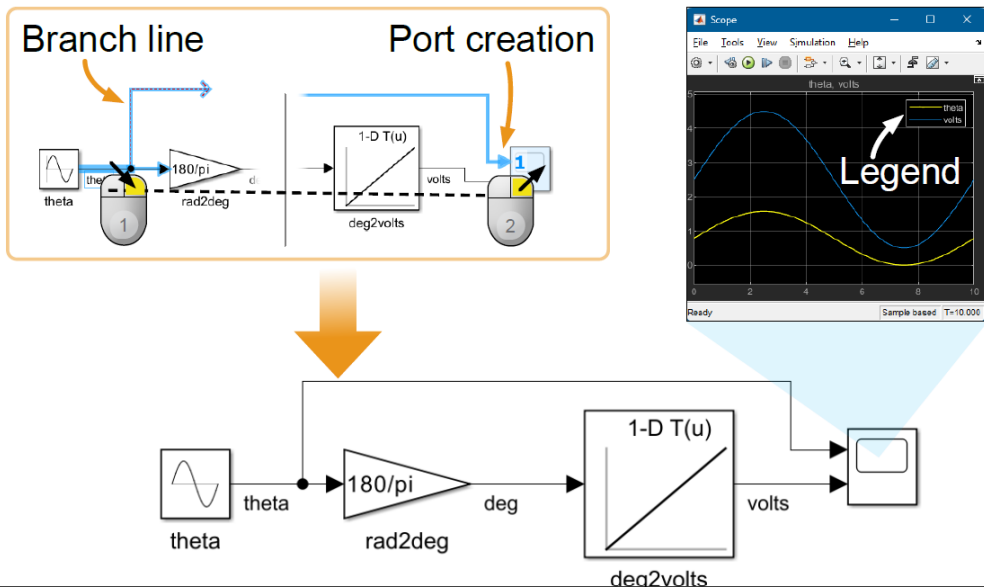
Gain:
degConv

Modeling in Simulink

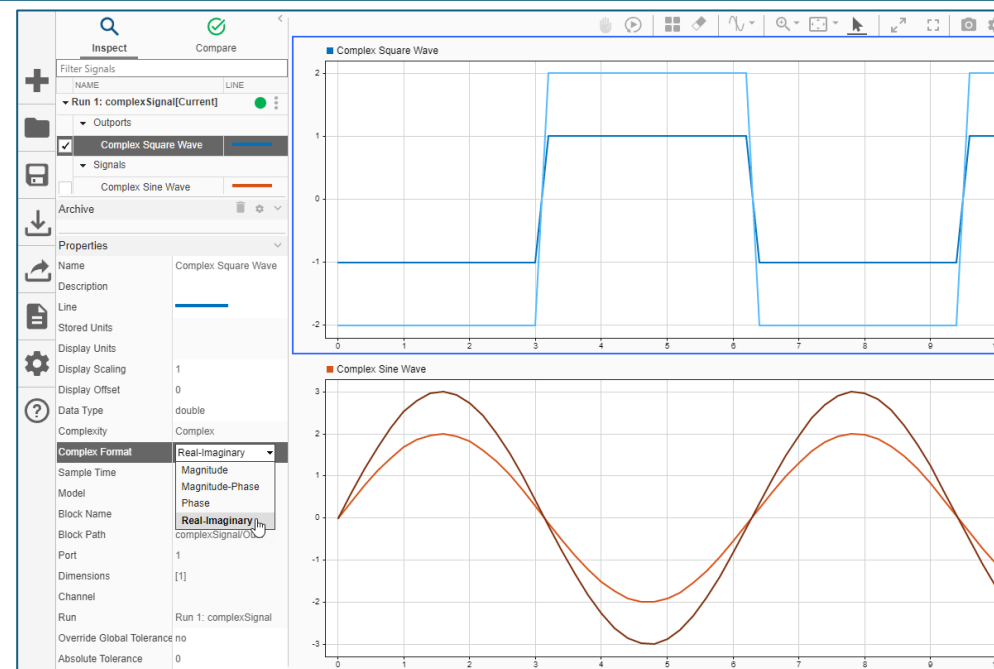
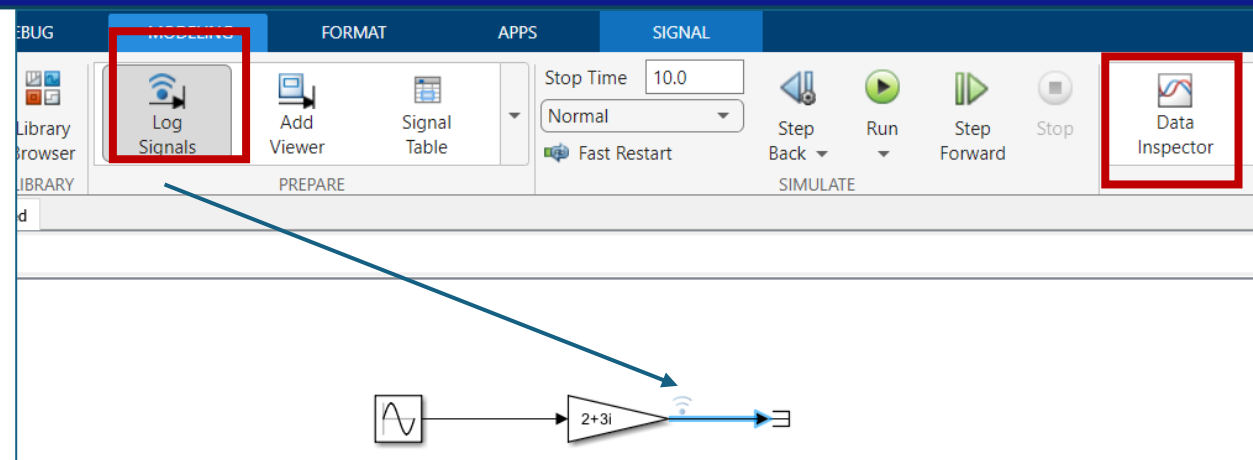
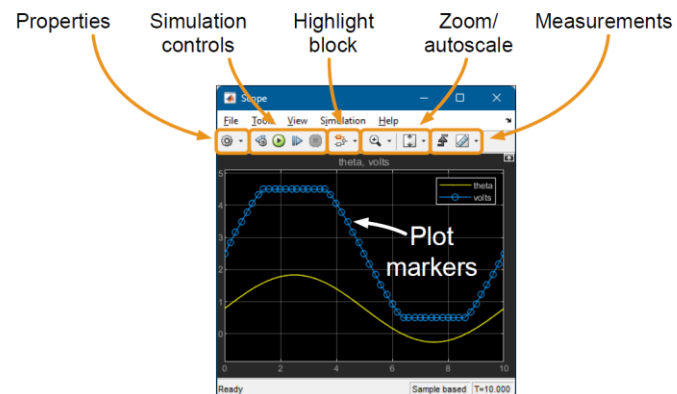


Scope & Data Inspector

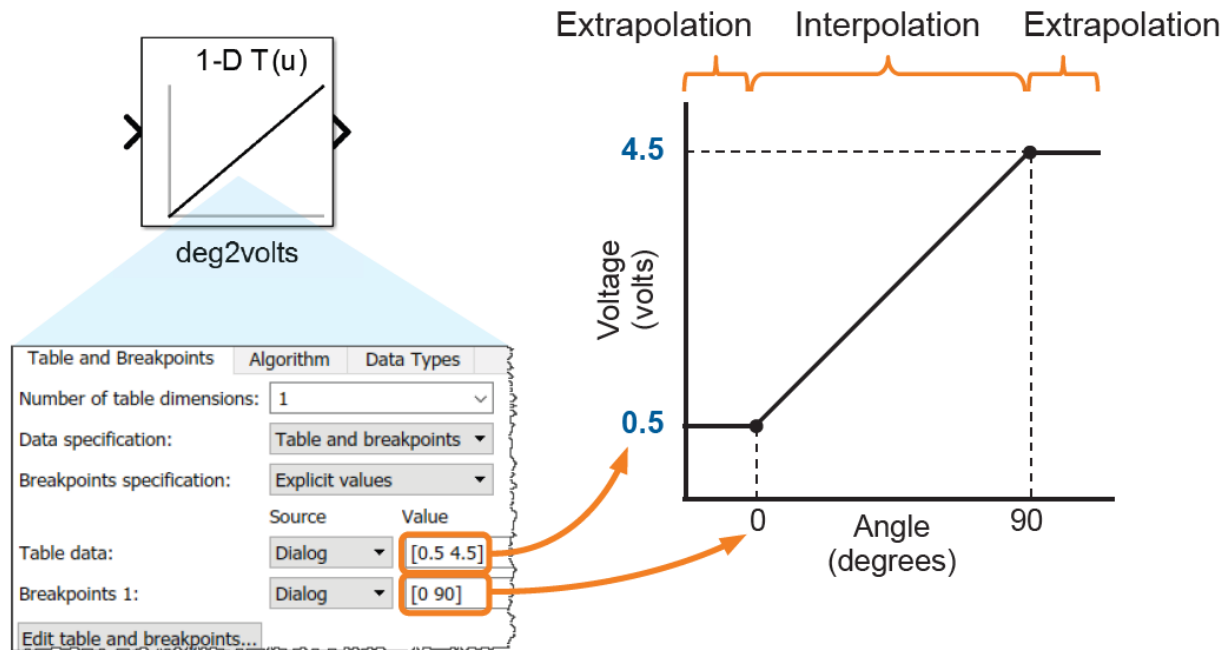
Plotting Multiple Signals



Customizing Scope Appearance



Defining Lookup Table Parameters



Interpolation.m

```
1 - clc;
2 - clear;
```

Lookup Tables

1-D T(u) 2-D T(u) n-D T(u)

1-D Lookup Table 2-D Lookup Table n-D Lookup Table

Prelookup Interpolation Using Prelookup Direct Lookup Table (n-D)

x xdat y u sin(2*pi*u) u cos(2*pi*u)

Lookup Table Sine Cosine

Dynamic

Categories

Interpolation

Functions

fx interp (signal) Interpolation — increase samplin...

fx interp1 1-D data interpolation (table look...

fx interp2 Interpolation for 2-D gridded data...

fx interp3 Interpolation for 3-D gridded data...

fx interpn Interpolation for 1-D, 2-D, 3-D, an...

fx intfilt (sigal) Interpolation FIR filter design

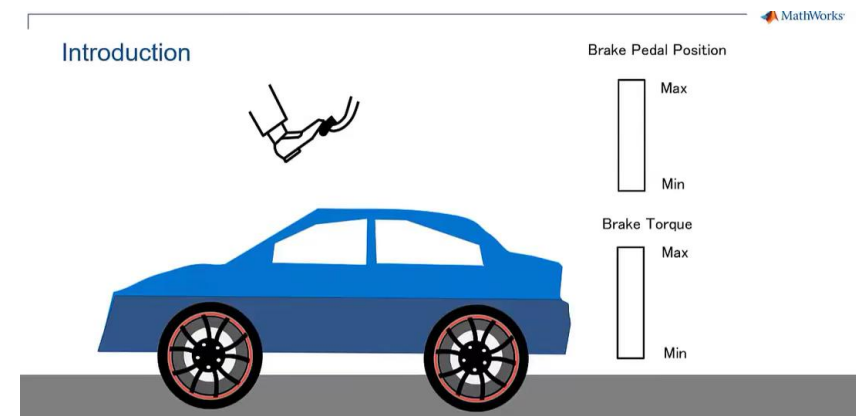
fx interp1q Quick 1-D linear interpolation

fx interpft 1-D interpolation using FFT meth...

fx dsp.Interpolator (dsp) Linear or FIR interpolation

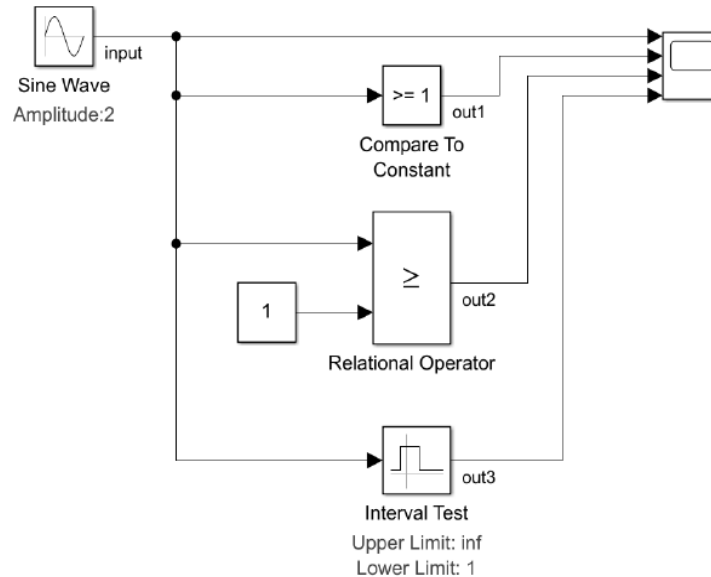
All installed products

script Ln 16 Col 7



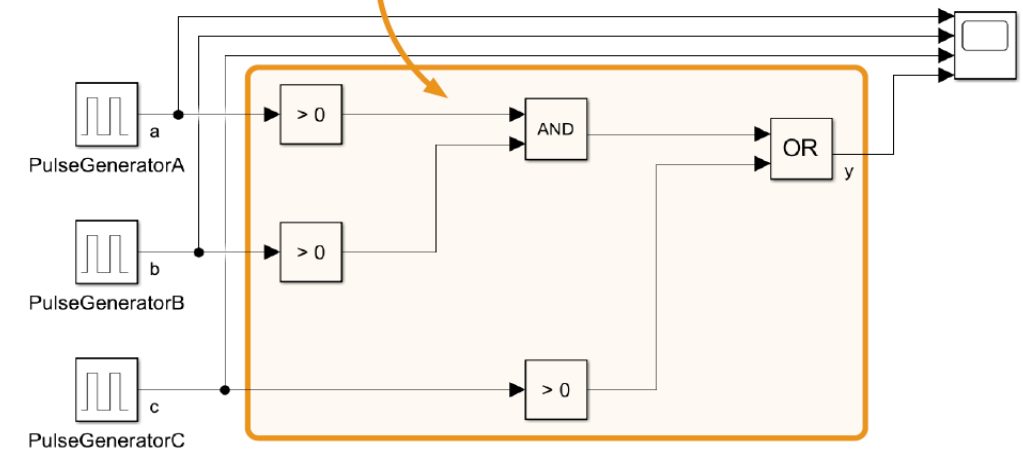
Modeling Programming Constructs

- Relational operators
($<$, $>$, $==$, etc.)
- Logical operators
(**AND**, **OR**, **NOT**, etc.)
- Decision statements
(**if-else**,
switch-case, etc.)



Modeling Comparisons

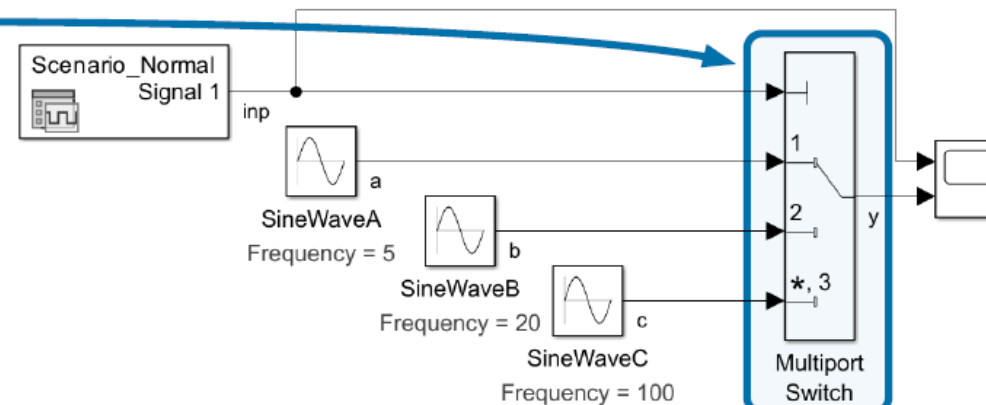
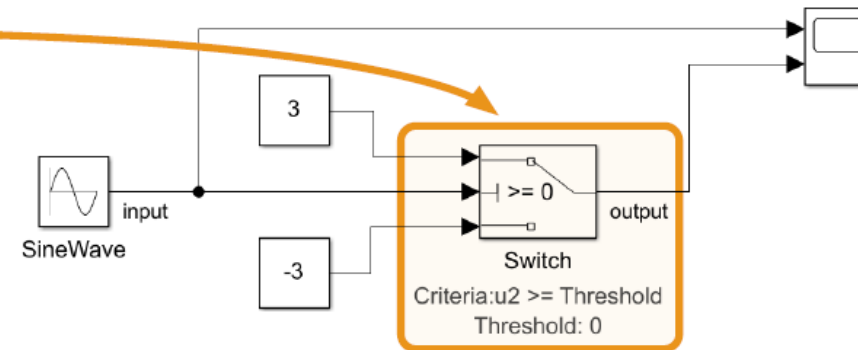
$$y = ((a > 0) \& (b > 0)) \mid (c > 0)$$



Modeling Decision Statements

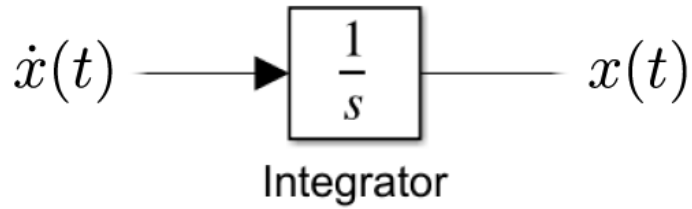
```
if input >= 0
    output = 3;
else
    output = -3;
end
```

```
switch inp
case 1
    y = a;
case 2
    y = b;
case 3
    y = c;
otherwise
    y = c;
end
```

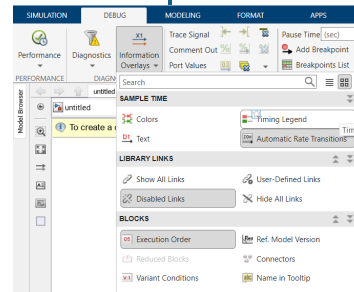


Continuous & Discrete Systems in Simulink

Continuous Systems



- $T_s = 0$ – Continuous block execution
- $T_s > 0$ – Discrete block execution
- $T_s = -1$ – Sample time is inherited



1

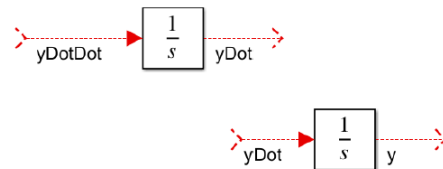
$$\ddot{y}(t) = u(t) - y(t) - \dot{y}(t)$$

Two Integrator blocks:

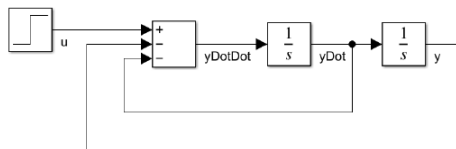
$$\ddot{y}(t) \rightarrow \dot{y}(t)$$

$$\dot{y}(t) \rightarrow y(t)$$

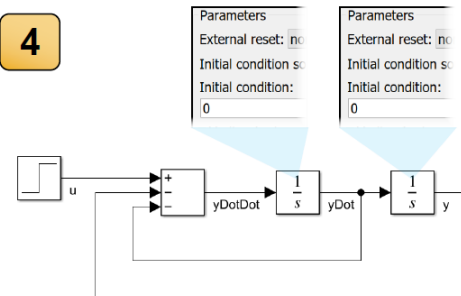
2



3

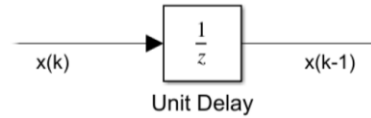


4



Discrete Systems

The Unit Delay block is the generic building block of difference equations.



Sample time 0.5

Time	$x(k)$	$x(k-1)$
0	0.48	0.00
0.5	0.84	0.48
1.0	1.00	0.84
1.5	0.91	1.00

Initial condition 0.0

1

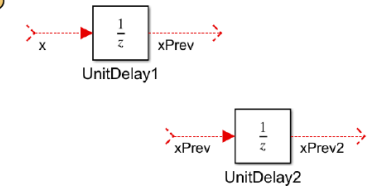
$$x(k) = x(k-1) - x(k-2) + u(k)$$

Two Unit Delay blocks:

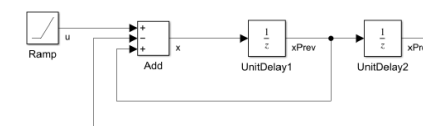
$$x(k) \rightarrow x(k-1)$$

$$x(k-1) \rightarrow x(k-2)$$

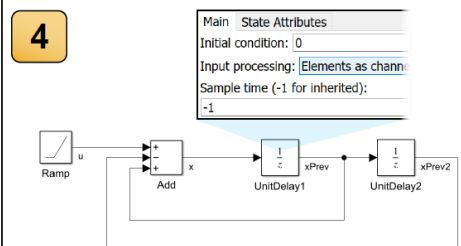
2



3

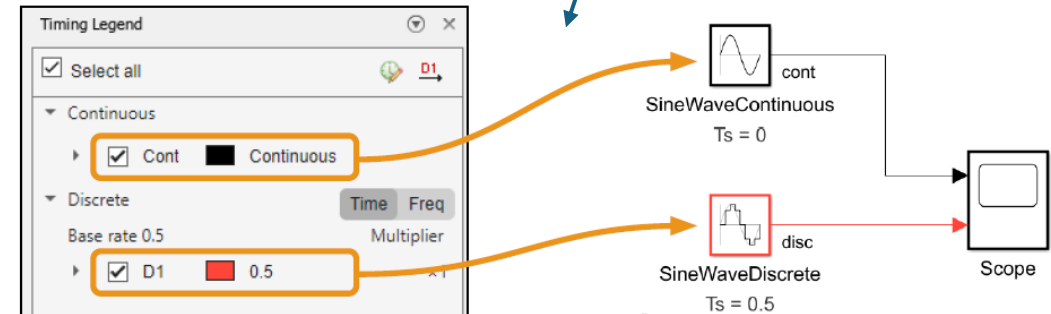
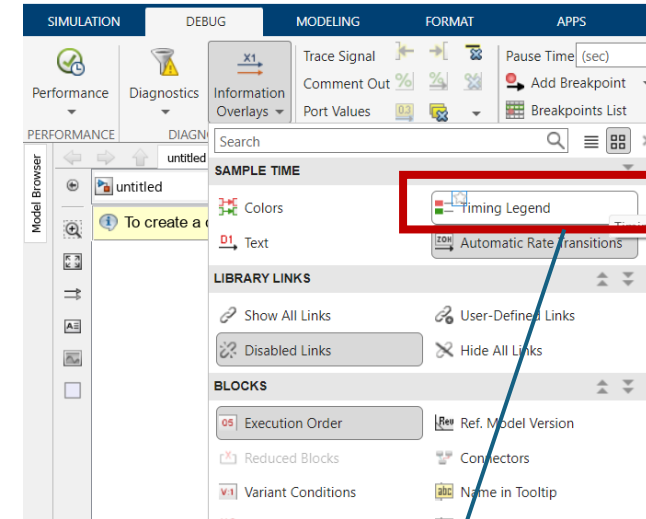
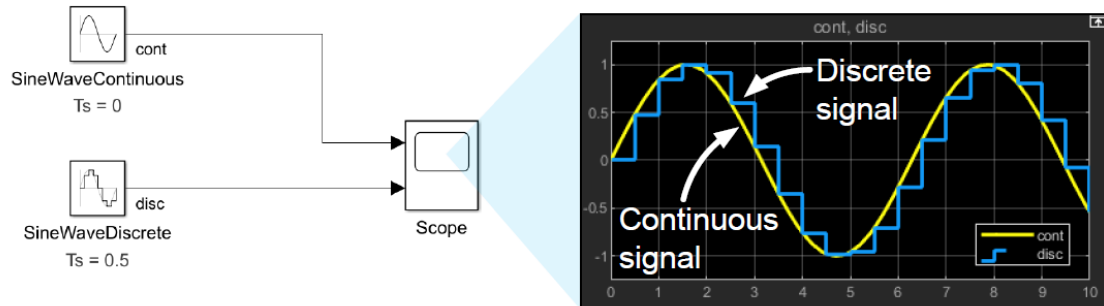


4

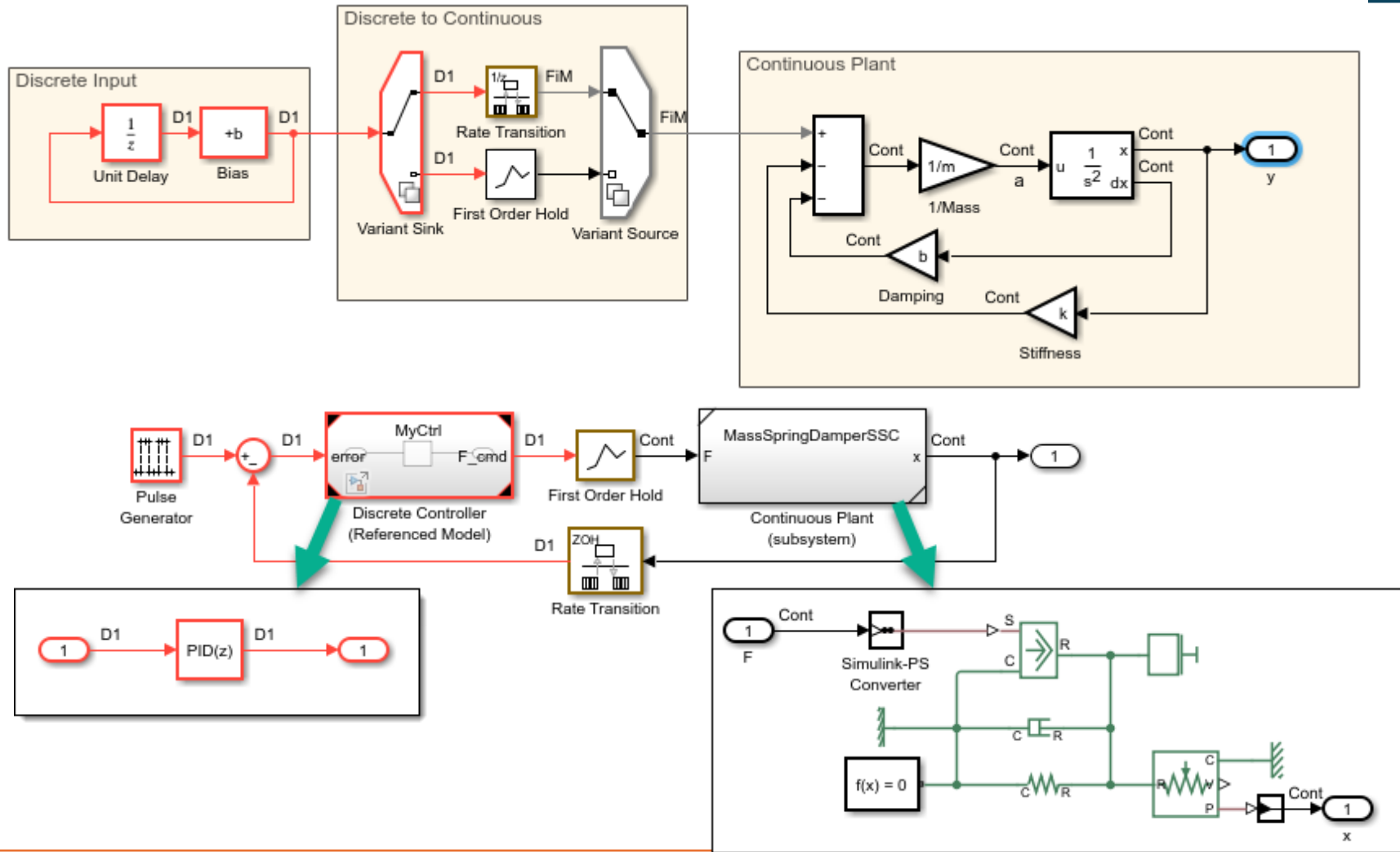


Sample Time

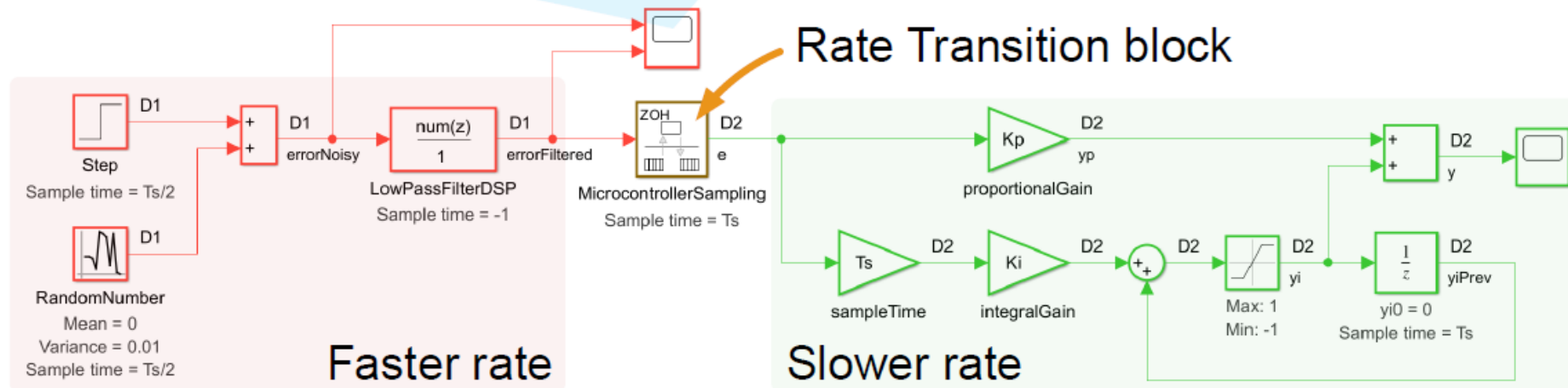
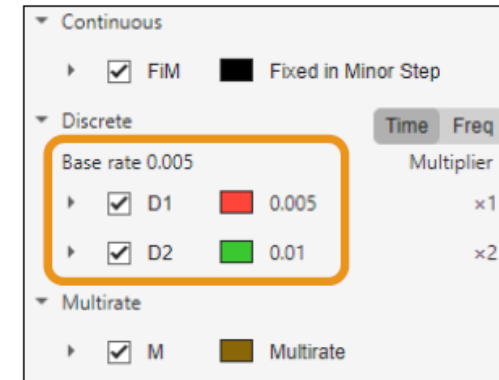
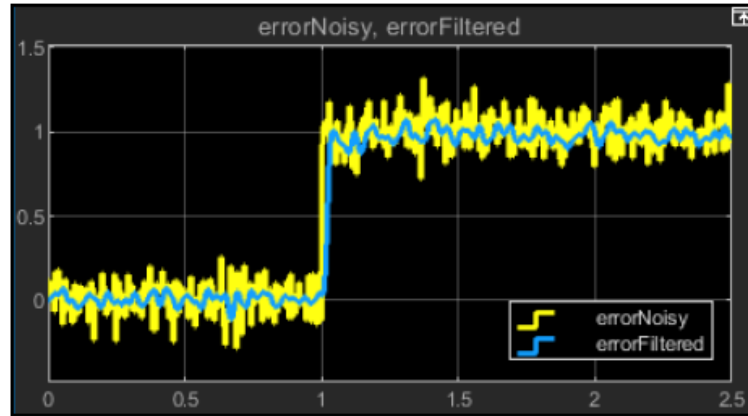
- $T_s = 0$ – Continuous block execution
- $T_s > 0$ – Discrete block execution
- $T_s = -1$ – Sample time is inherited



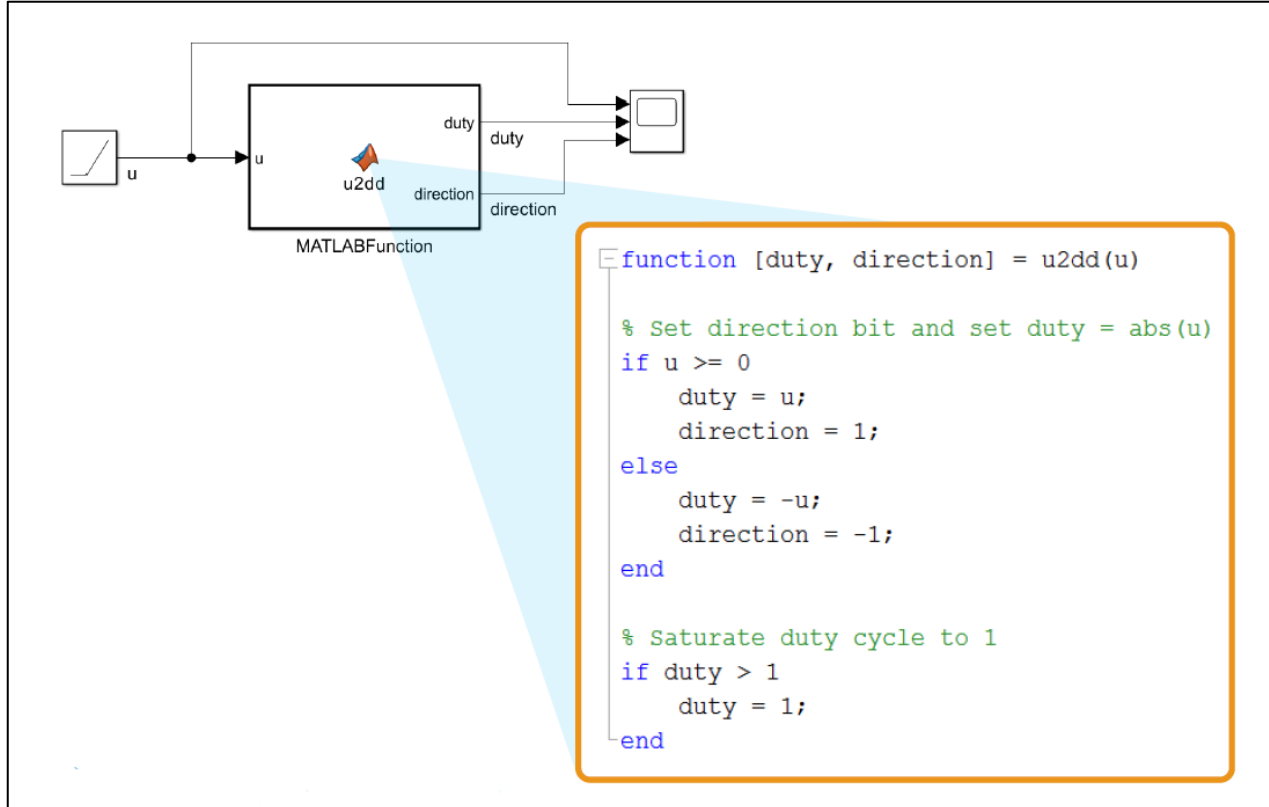
Continuous & Discrete Systems in Simulink



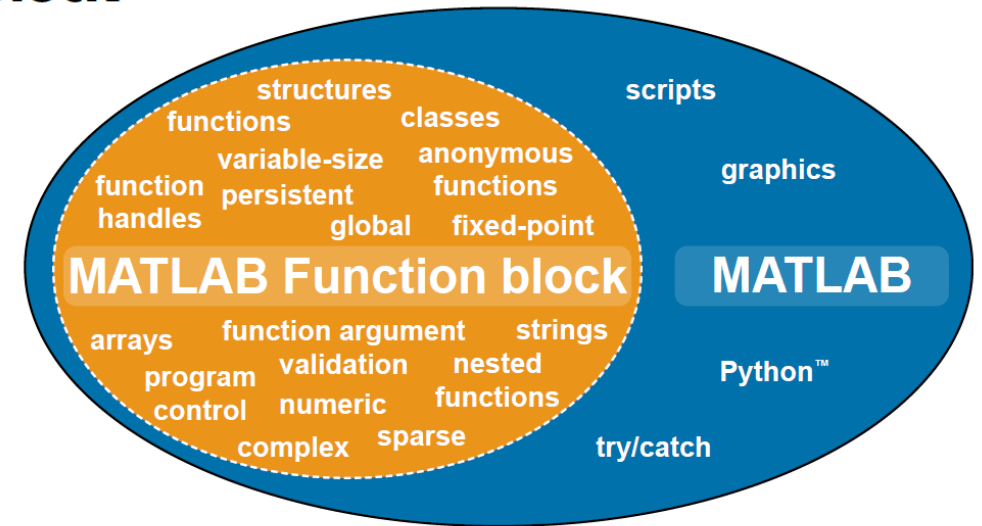
Multirate Systems



Modeling with MATLAB Function Blocks



Understanding the MATLAB Function Block



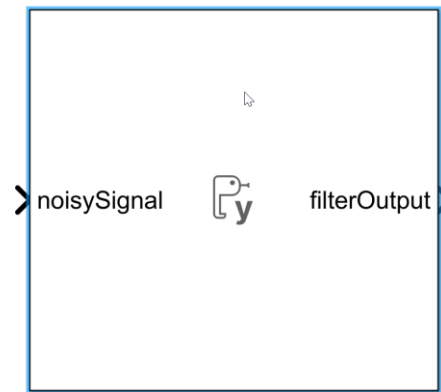
- Subset of supported functions and language features
- Unsupported functions call MATLAB interpreter



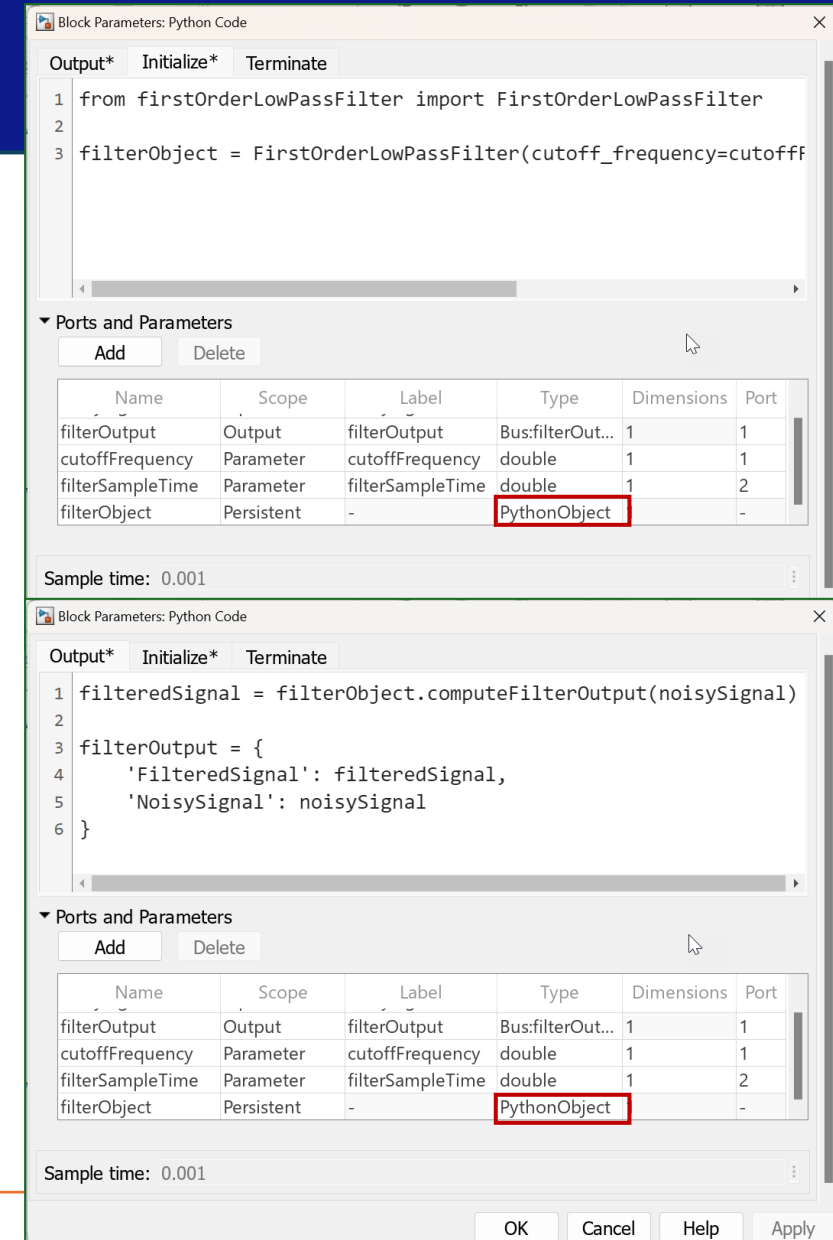
Python Code Block

- Easily integrate external Python code into Simulink
- Call external Python libraries, functions or methods from Simulink
- Write **native Python code** in the block dialog to support the interface between Simulink and Python
 - No need for “py.” prefix
- Supports all simulation modes
 - normal, accelerator, rapid accelerator, model reference accelerator
- **Python Code Block vs Python Importer**
 - Flexibility vs Reusability

R2025a



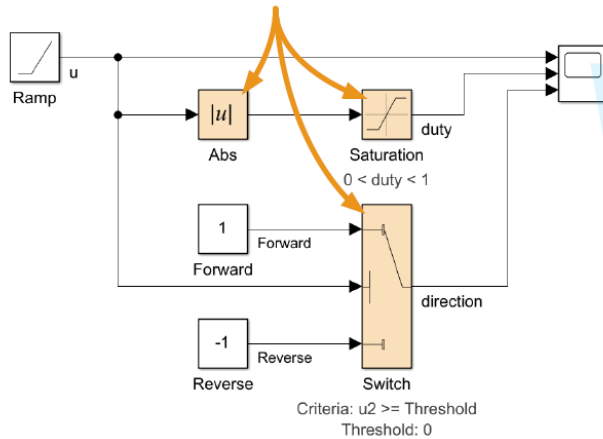
Python Code



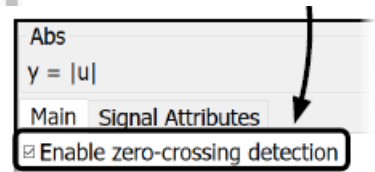
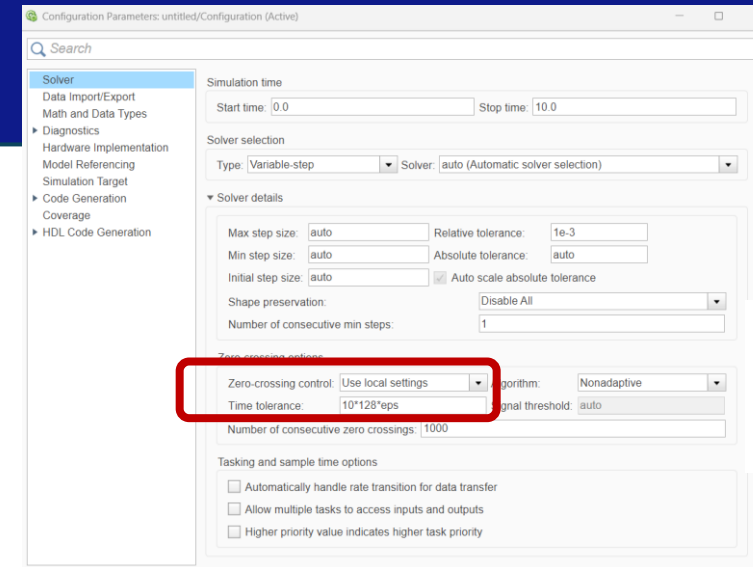
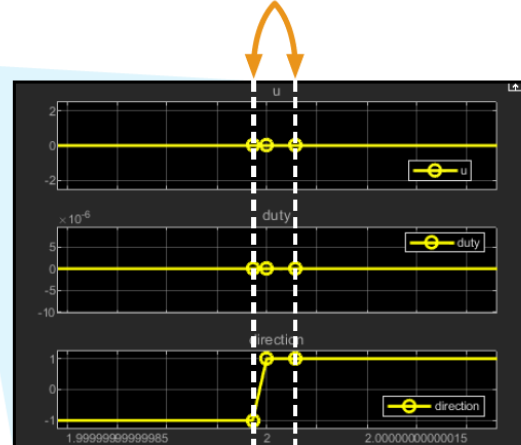
Zero Crossing Detection

Blocks with discontinuous behavior can cause the solver to take additional time steps before and after a discontinuity.

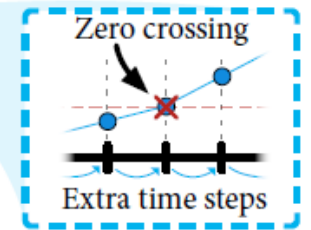
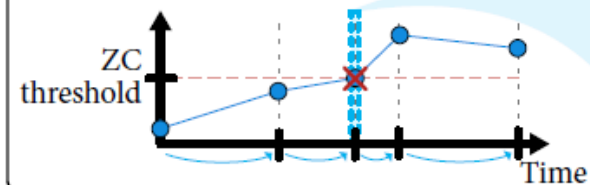
Blocks with zero-crossing detection



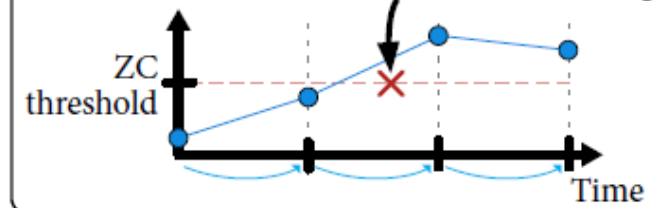
Extra time steps taken around a zero crossing



Variable-step solver



Fixed-step solver



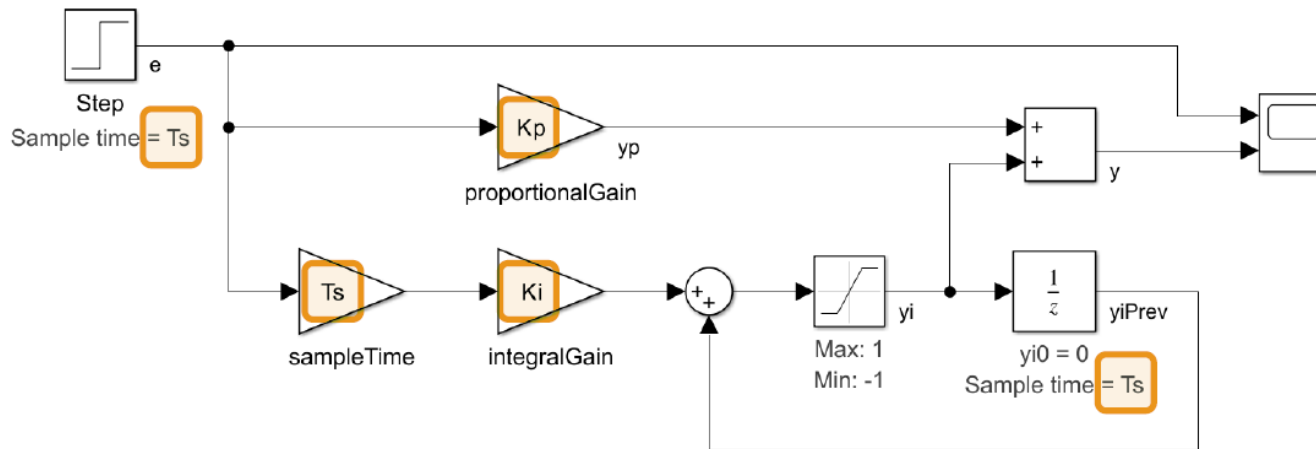
Parameterizing in Simulink

- Make model more readable
- Edit parameters from workspace

Workspace	
Name	Value
Ki	2
Kp	0.3
Ts	0.01

Workspace					
Name	Value	Size	Class	Min	Max
A	[1;2;3]	3×1	double	1	3
b	5	1×1	uint32	5	5
C	5×5 cell	5×5	cell		
str	"abc"	1×1	string		
S	0×0 struct	0×0	struct		
T	3×2 table	3×2	table		

Base Workspace



Model Workspace

The image illustrates the process of accessing the Model Workspace in Simulink. It consists of several screenshots:

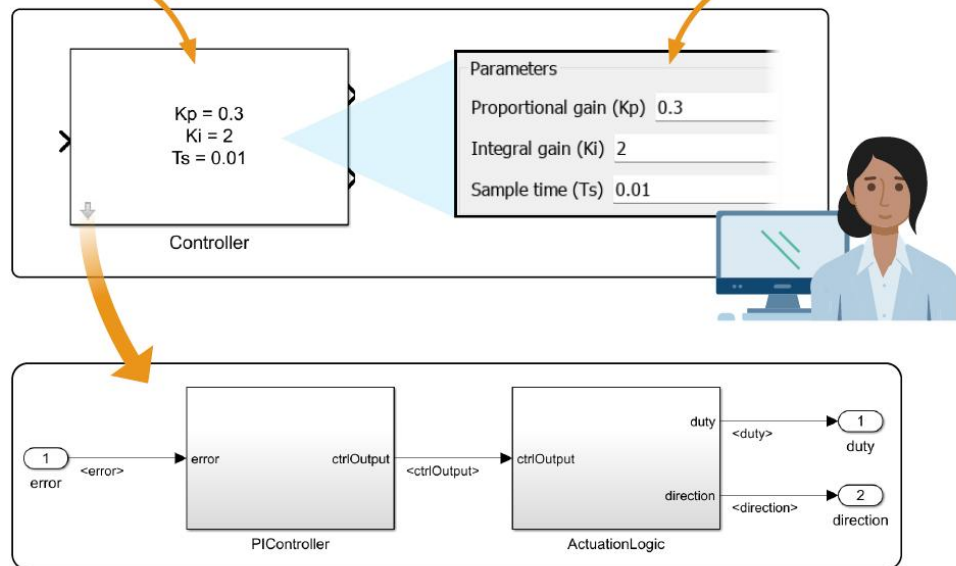
- Top Toolbar:** The 'Model Explorer' button is highlighted with a red box.
- Model Hierarchy:** The 'Model Workspace' is highlighted with a red box in the tree view.
- Model Explorer Window:** This window displays a table of data objects in the Model Workspace. The table has columns: Name, Value, DataType, Dimensions, Complexity, Min, Max, and Unit. The objects listed are 'a' and 'b'.
- Model Properties Window:** The 'Model Properties' window for 'etc_PI' is shown. The 'Main' tab is active, and the 'PreLoadFcn' callback is highlighted. The 'Model pre-load function' section shows the following code:

```
1 Kp = 0.3;  
2 Ki = 2;  
3 Ts = 0.01;
```
- Command Window:** The MATLAB Command Window shows the command `hws = get_param(bdroot, 'modelworkspace')` and the resulting handle `hws` with a value of `1x1 M...`.

Creating Mask

Masked subsystem

Custom mask dialog box



The screenshot shows the MATLAB/Simulink interface with the Mask Editor and Block Parameters dialog box. The Mask Editor is open, showing the "Parameters & Dialog" tab. The "Parameters & Dialog" tab contains a table with the following parameters:

Type	Prompt	Name
A	%<MaskType>	DescGro
A	%<MaskDescription>	DescText
Parameters	Parameters	Parameter
#1	Proportional gain (Kp)	Kp
#2	Integral gain (Ki)	Ki
#3	Sample time (Ts)	Ts

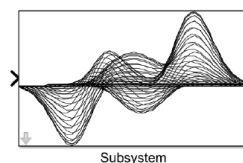
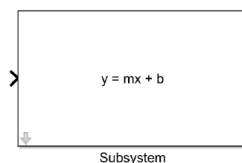
The Block Parameters dialog box is also open, showing the "Controller (mask)" parameters:

- Proportional gain (Kp) 0.3
- Integral gain (Ki) 2
- Sample time (Ts) 0.01

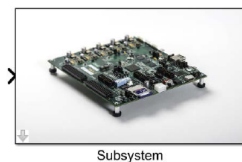
The "Help" button is highlighted in the dialog box.

Text: `disp("y = mx + b")`

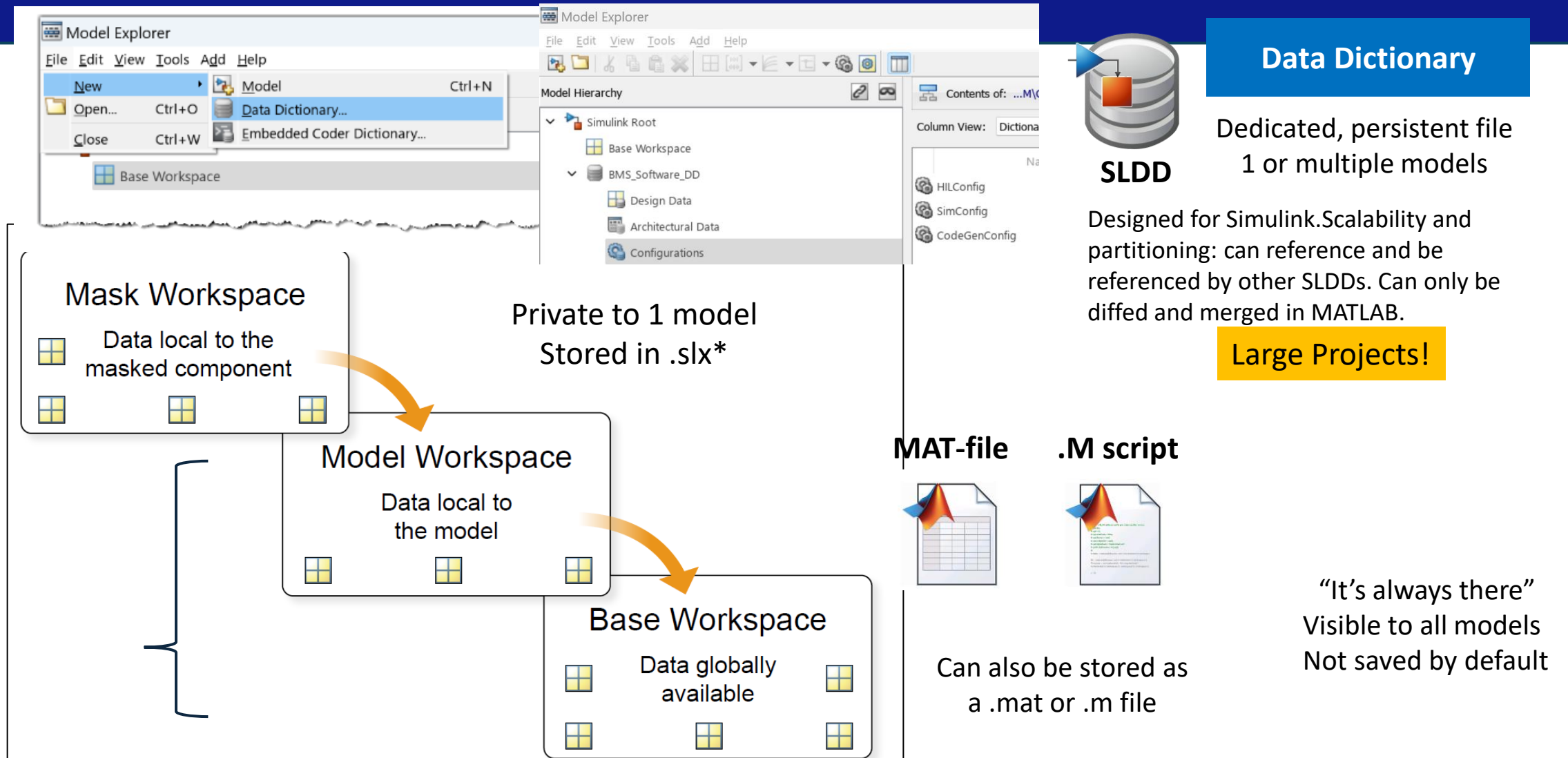
Graphics: `plot(peaks)`



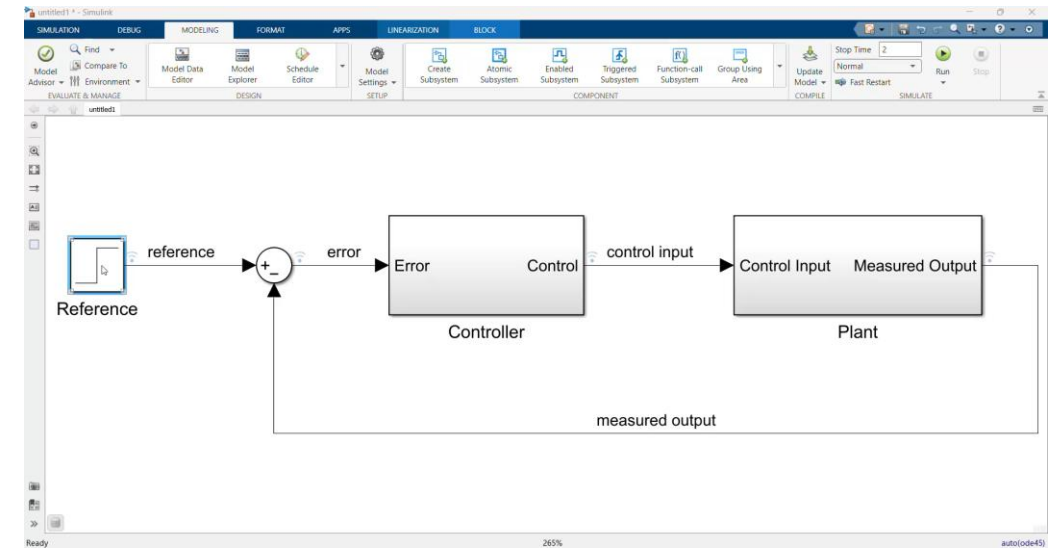
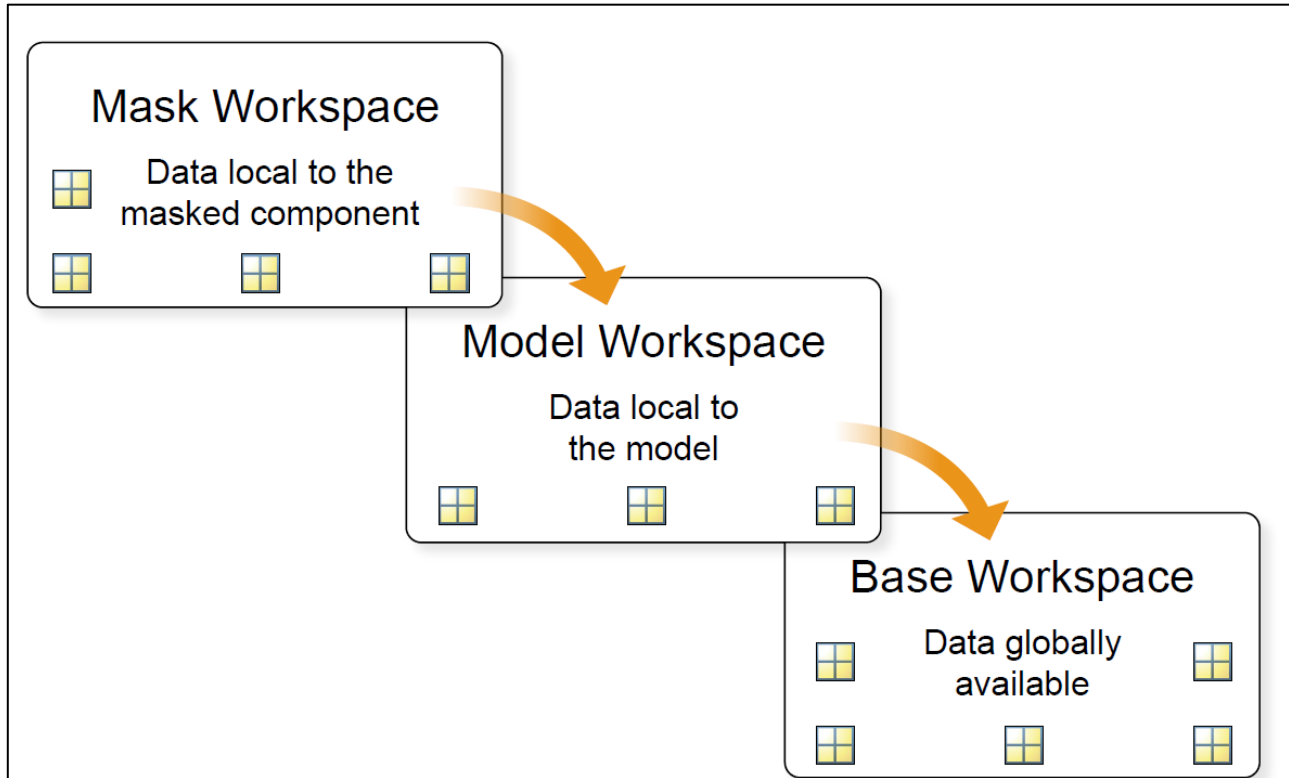
Images: `image("ecu.jpg")`



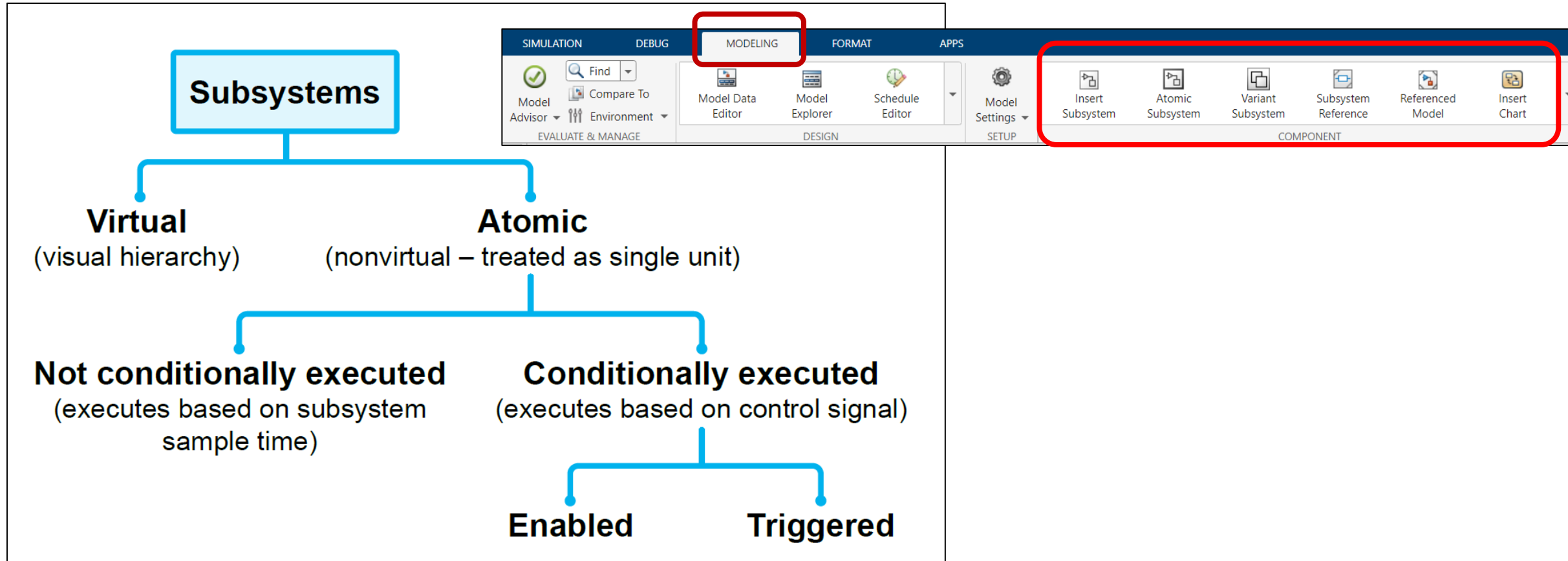
Simulink's options for data storage



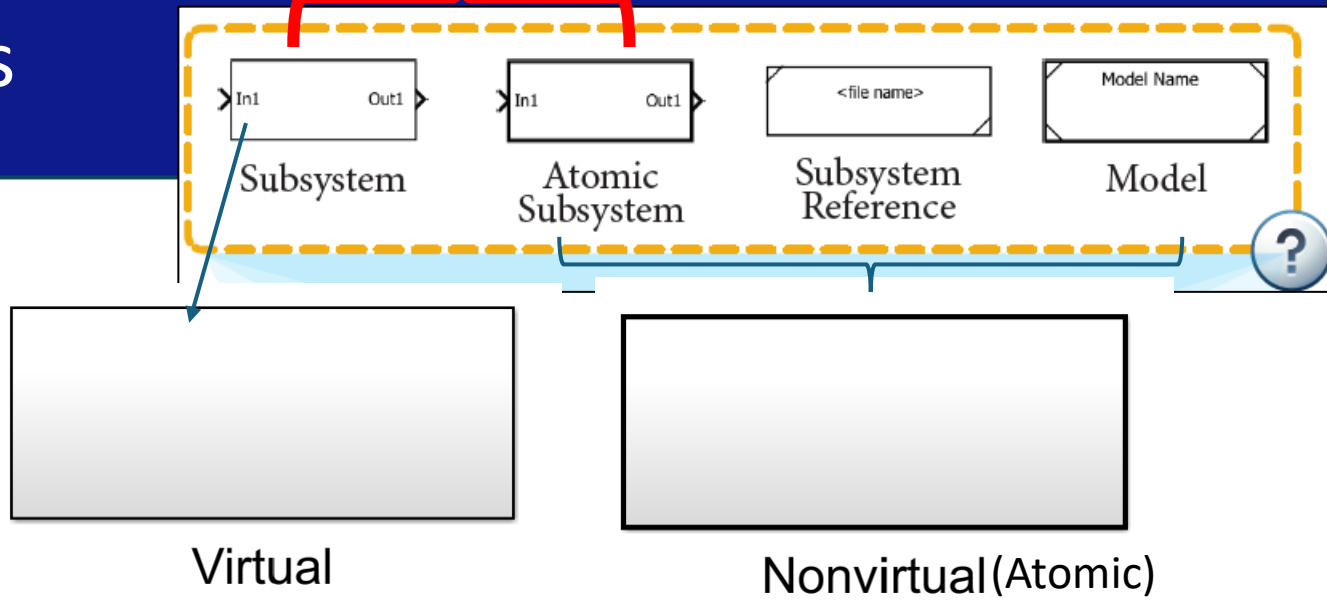
Simulink's options for data storage



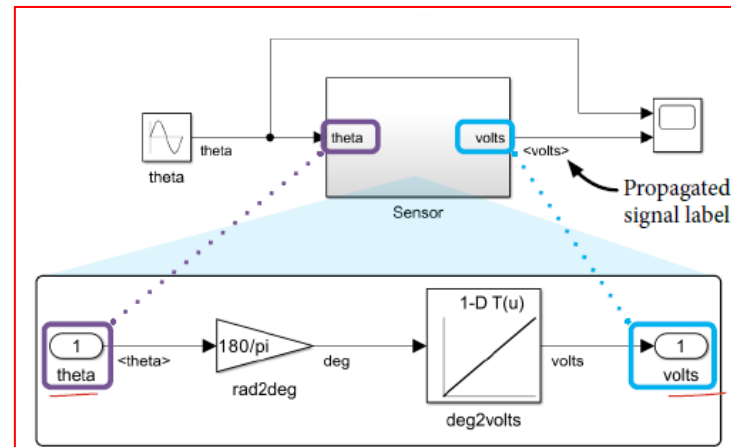
Types of Subsystems



Subsystems



- Atomic (nonvirtual) subsystems **functionally** encapsulate blocks.
 - No outside block will be executed in-between the first and last block of the atomic subsystem.
 - Virtual subsystems group blocks **visually**.
 - The virtual boundary is ignored at compile time.
 - Virtual subsystems are useful for:
 - Visually organizing and decluttering models.
 - Workflows involving physical connections (Simscape).
 - Workflows involving HDL code.
-
- The diagram illustrates two types of subsystems in Simulink:
- Functional Encapsulation (Top):** A block labeled "Sensor" contains two internal blocks: "theta" (highlighted with a purple border) and "volts" (highlighted with a blue border). An input signal "theta" enters the "Sensor" block from the left. An output signal "<volts>" exits the "Sensor" block to the right. This represents functional encapsulation where the internal logic is hidden behind a single interface.
 - Visual Encapsulation (Bottom):** A zoomed-in view of the "Sensor" block's internal structure. It shows a sequence of blocks: a constant block "1" (purple border), a gain block "180/pi" (triangle), and a transfer function block "1-D T(u)". The signal flow is from "1" to "180/pi" (labeled "<theta>") and then to "1-D T(u)" (labeled "deg"). The final output is "volts". This represents visual encapsulation where the internal details are visible but grouped together for organizational purposes.

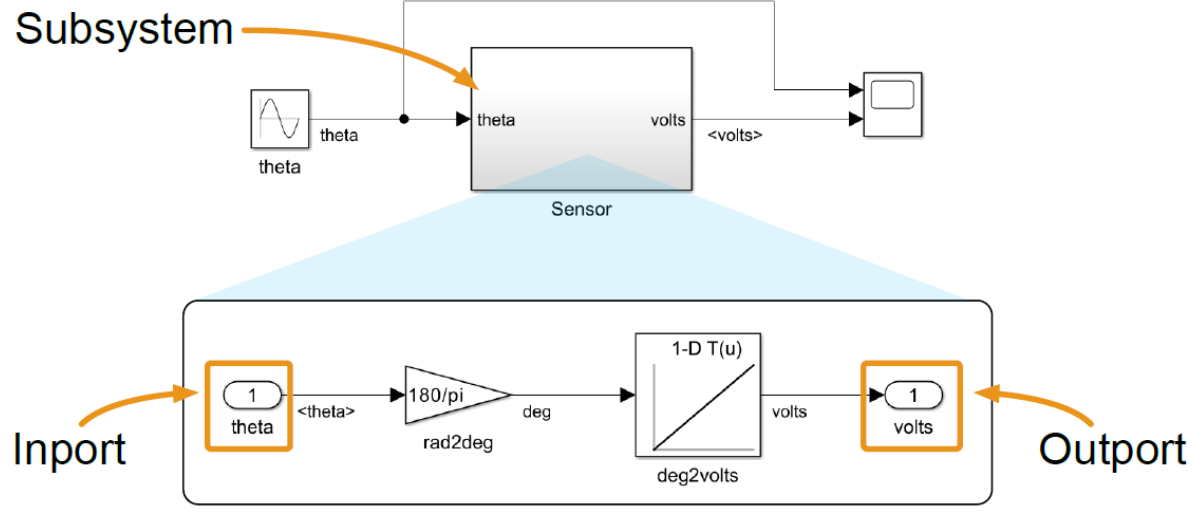


Benefits	Virtual subsystem	Atomic subsystem
Ease of use		
Readability		
Traceability		
Reusability		
Concurrent development		
Unit testing		
Performance		



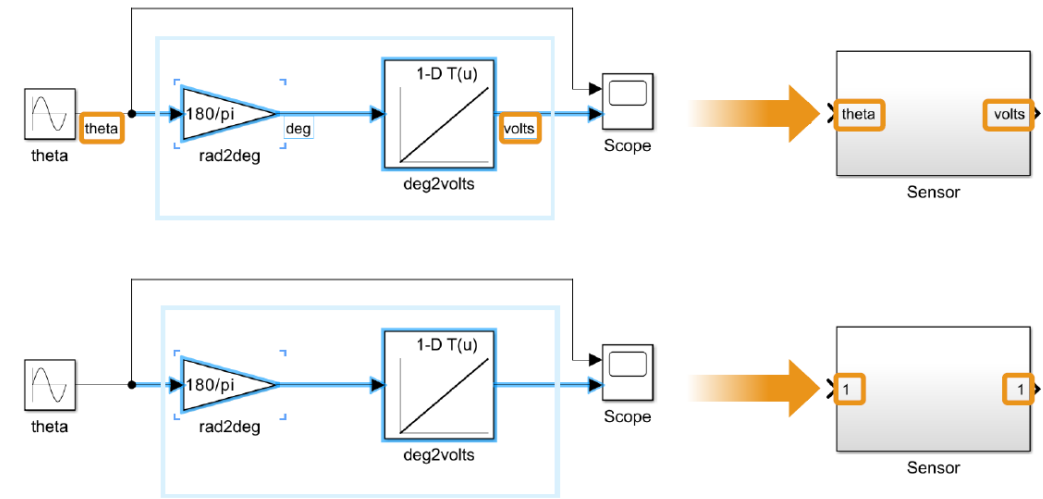
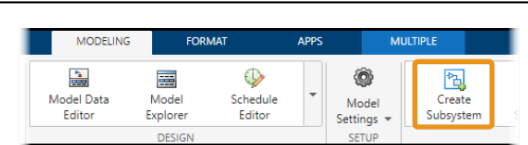
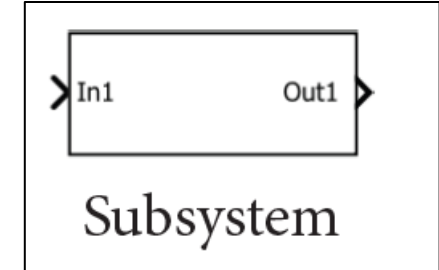
Virtual Subsystem

- **Scalability** – Add hierarchy to reduce the number of blocks at a given level
- **Readability** – Group functionally related blocks

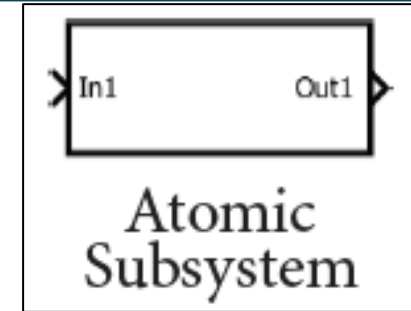
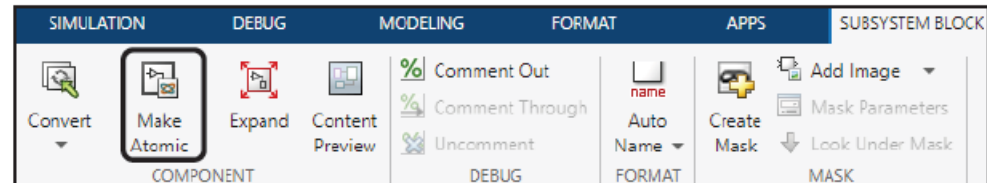


Benefits	Virtual subsystem
Ease of use	+
Readability	+
Traceability	-
Reusability	-
Concurrent development	-
Unit testing	- *
Performance	

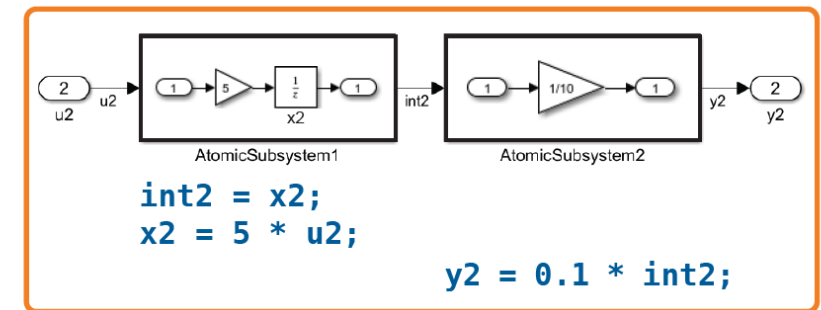
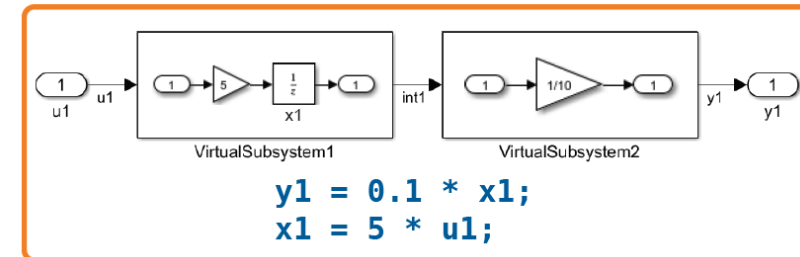
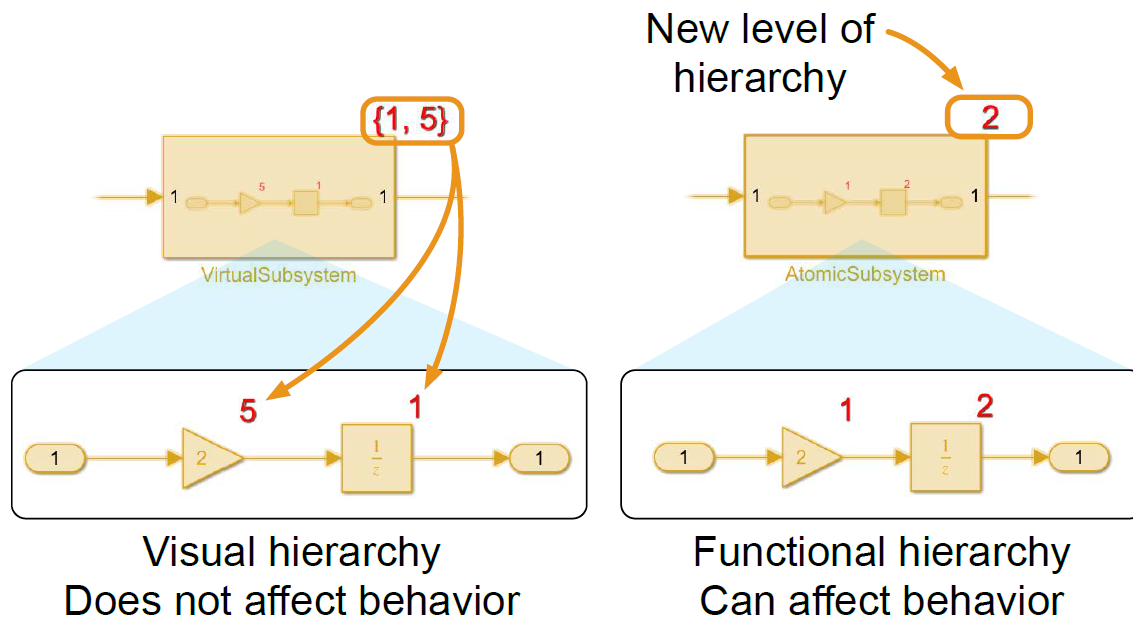
* Simulink Test is needed for unit testing



Atomic (Nonvirtual) Subsystems

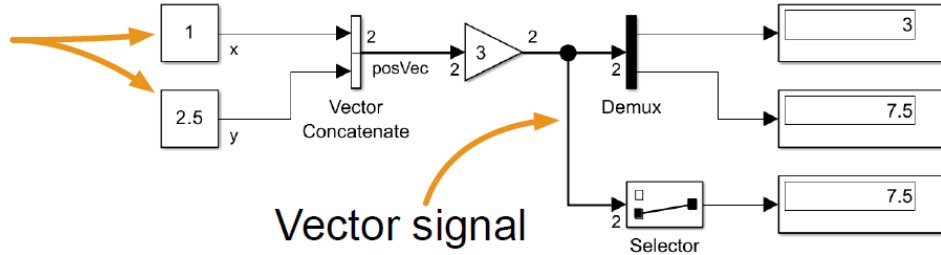


Creating Functional Hierarchy



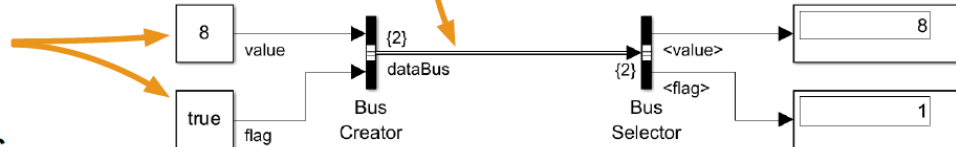
Vector & Bus Signals

Vectors must have **same** data type



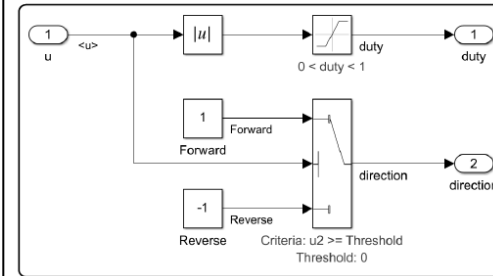
Bus signal

Buses can have **different** data types

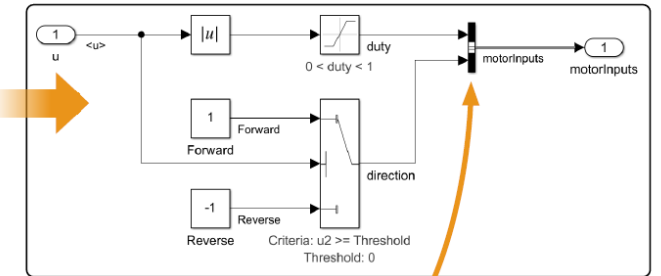


Use the Bus Creator block to group signals into a bus signal.

Multiple output signals:

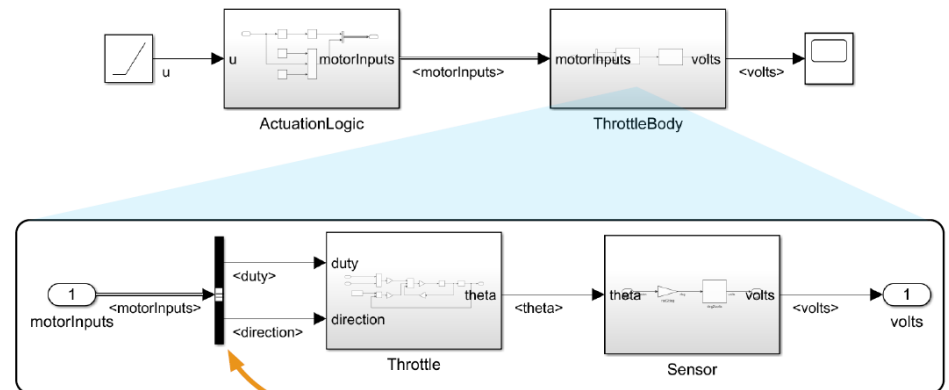


Single bus output signal:



Bus Creator

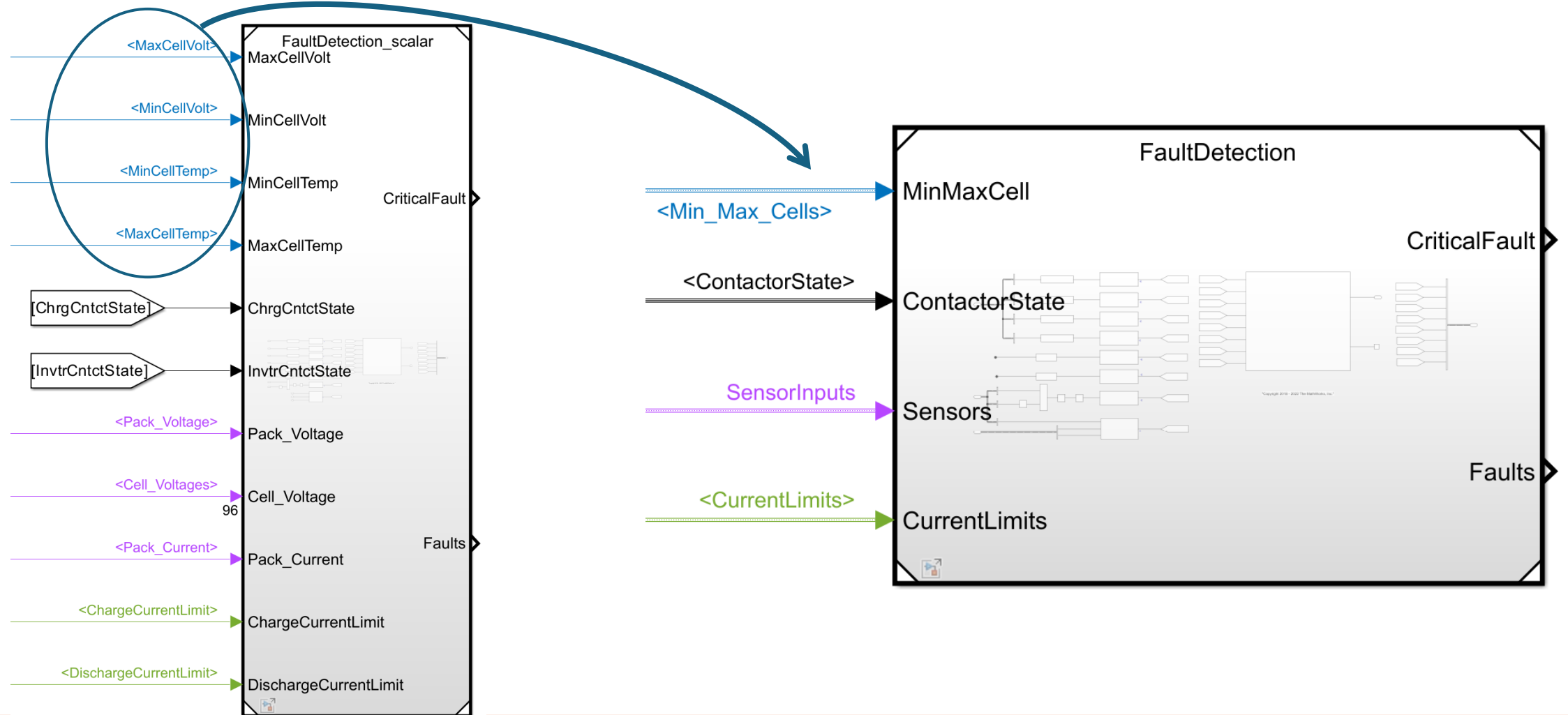
Use the Bus Selector block to extract signals from a bus.



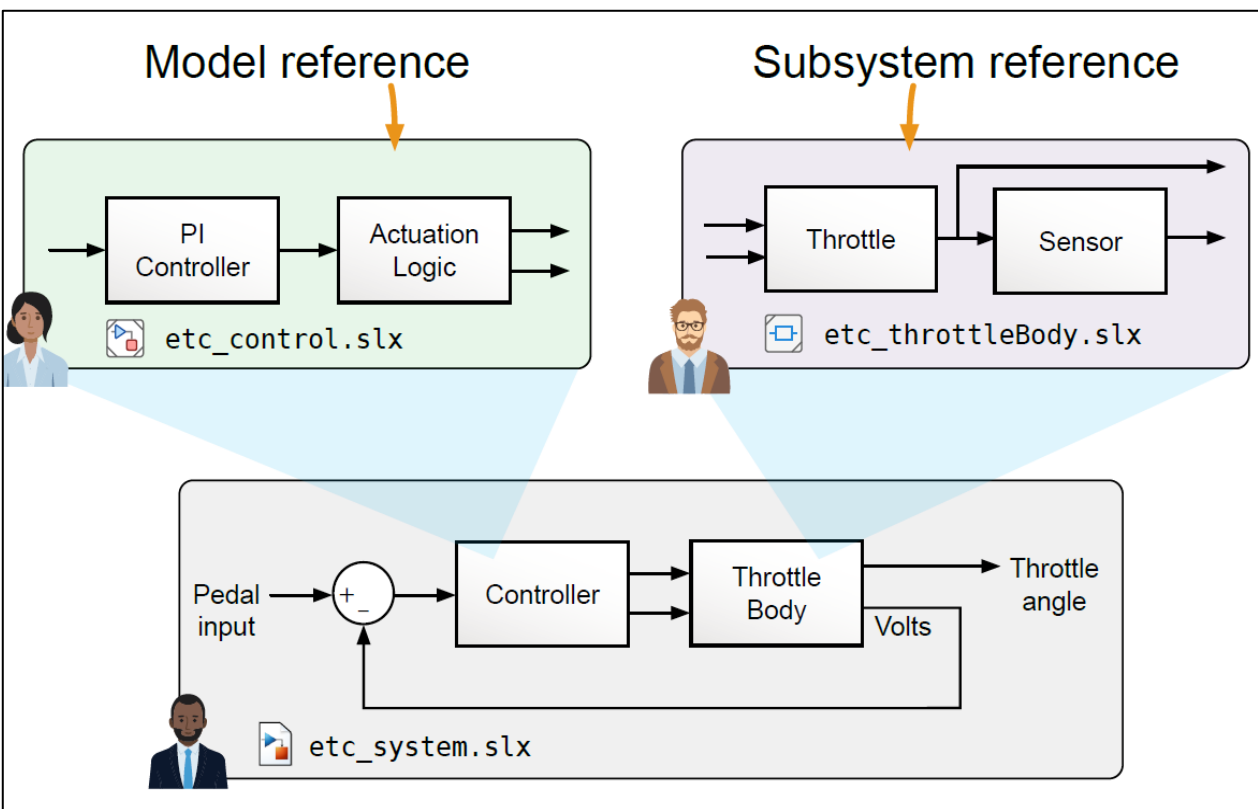
Bus Selector



Simplifying Interfaces with Buses

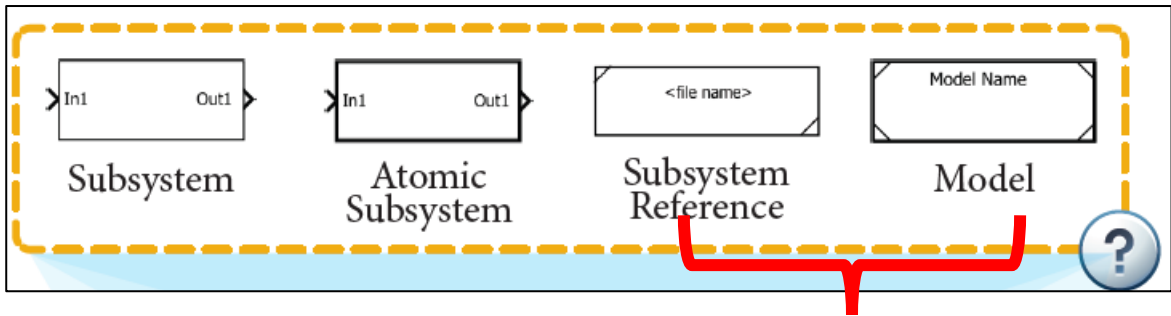


Reference Component Simulink

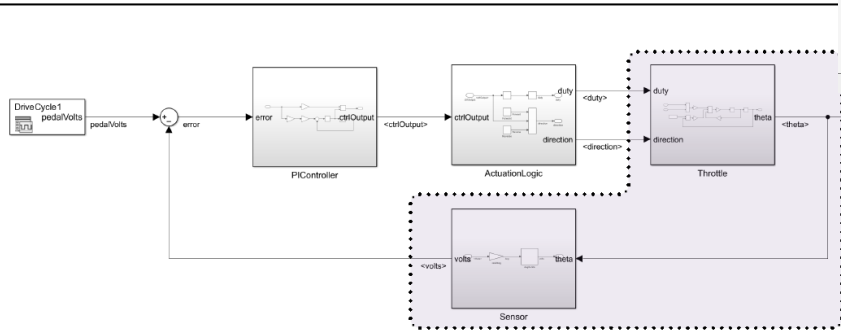


Benefits	Virtual subsystem	Atomic subsystem	Subsystem reference	Model reference
Ease of use	+	+		-
Readability	+	+	+	+
Traceability		+	+ *	+
Reusability	-	-	+	+
Concurrent development	-	-	+	+
Unit testing	-	-	+	+
Performance		-	- *	+, -

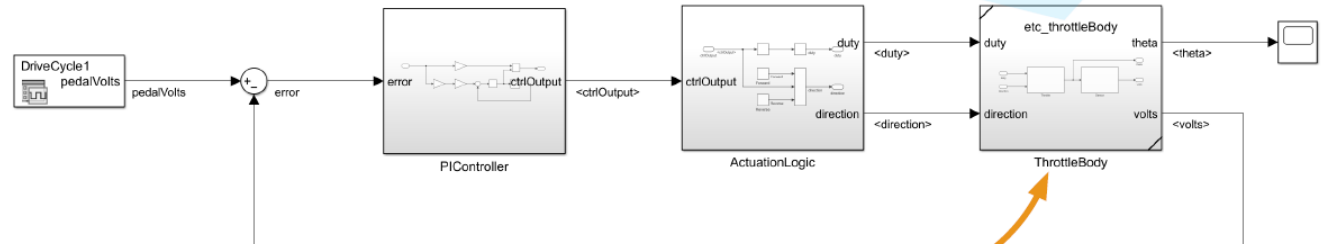
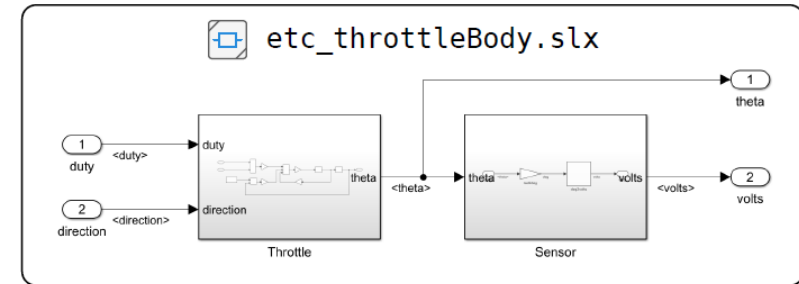
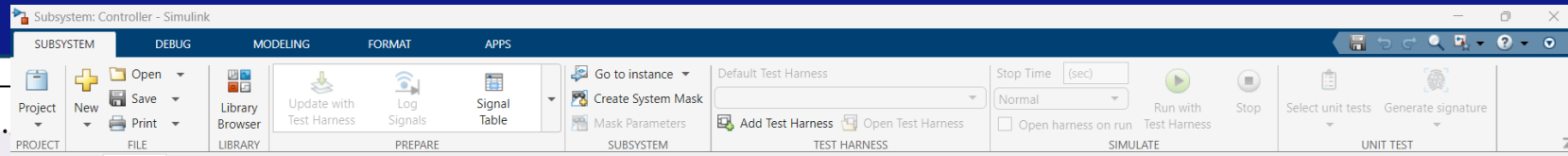
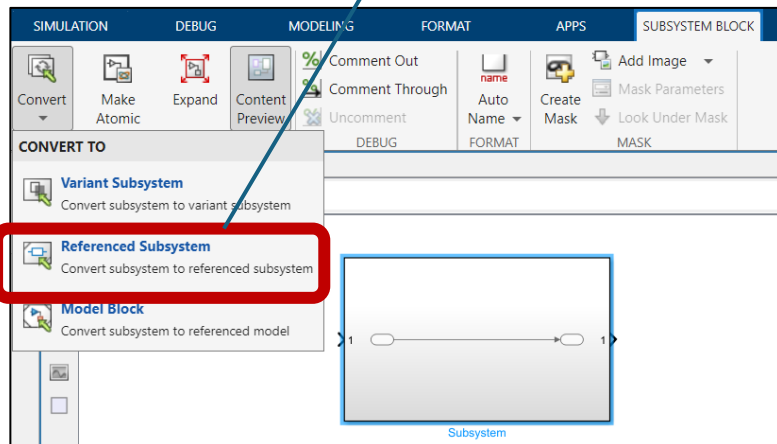
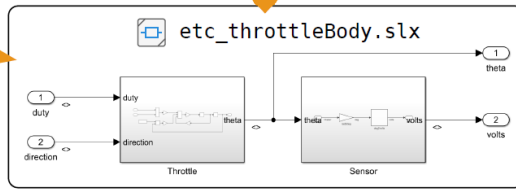
* Depends whether subsystem reference instance is virtual or atomic



Subsystem References



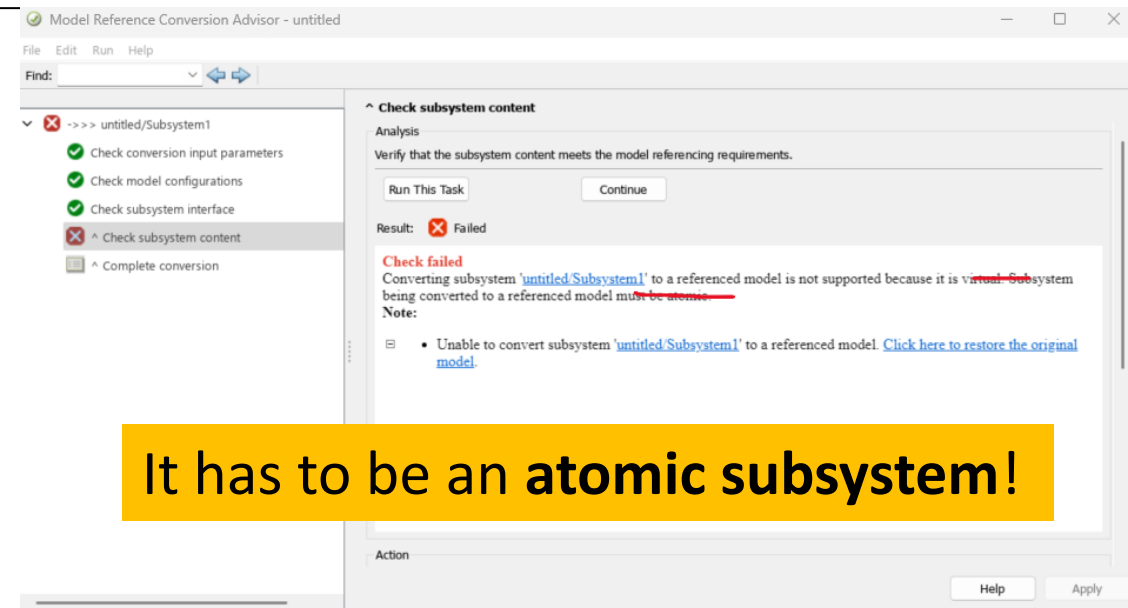
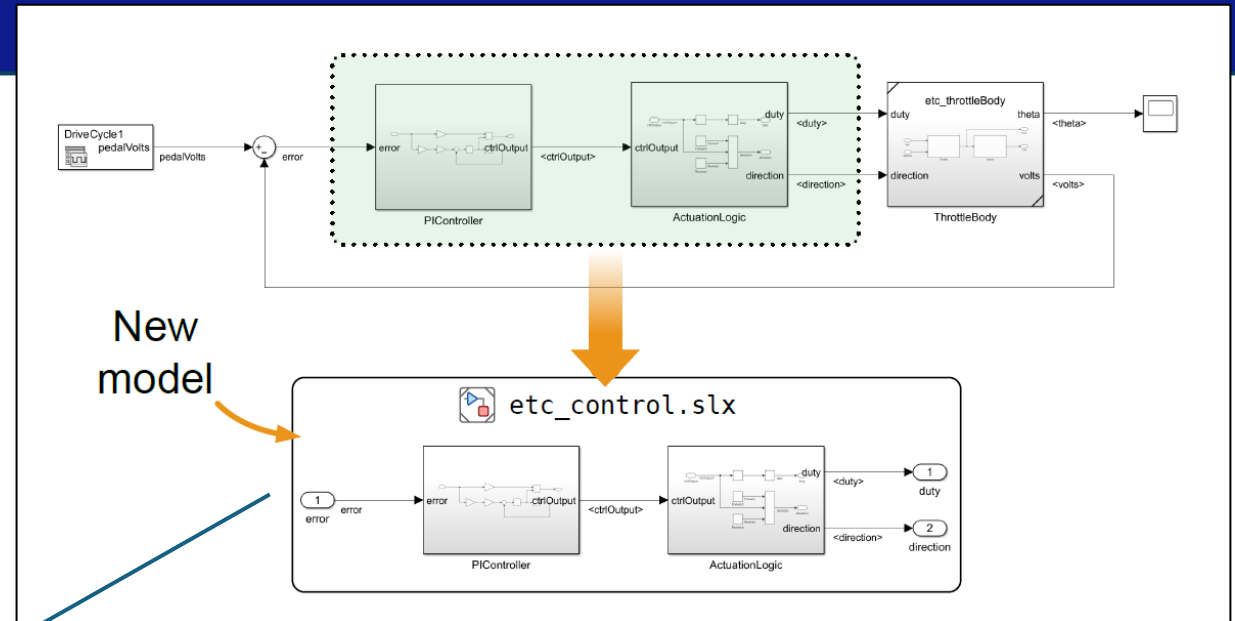
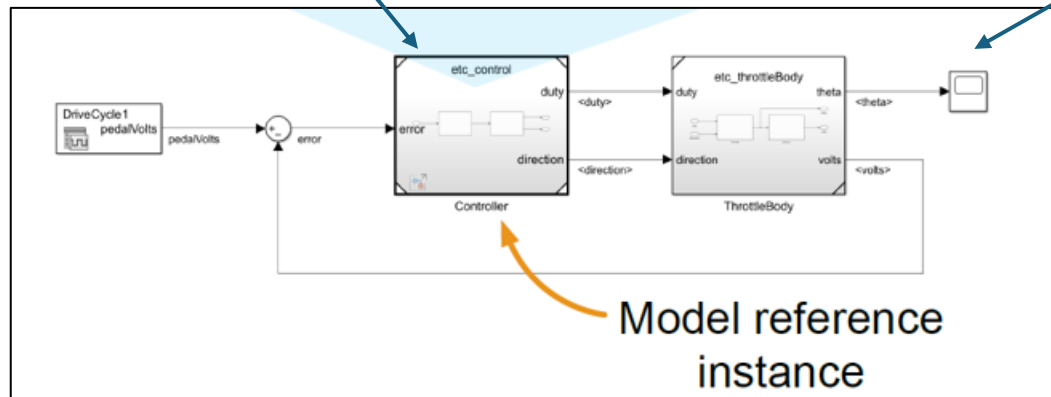
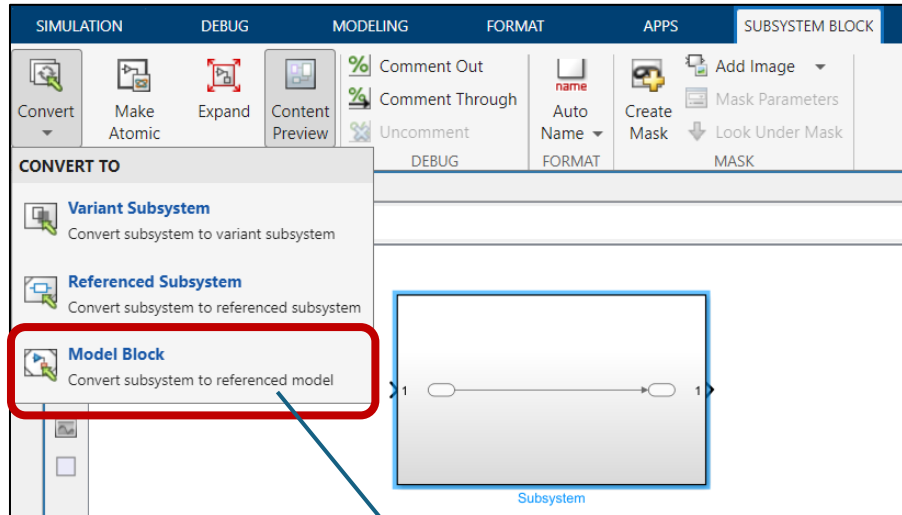
New subsystem reference



Subsystem reference instance

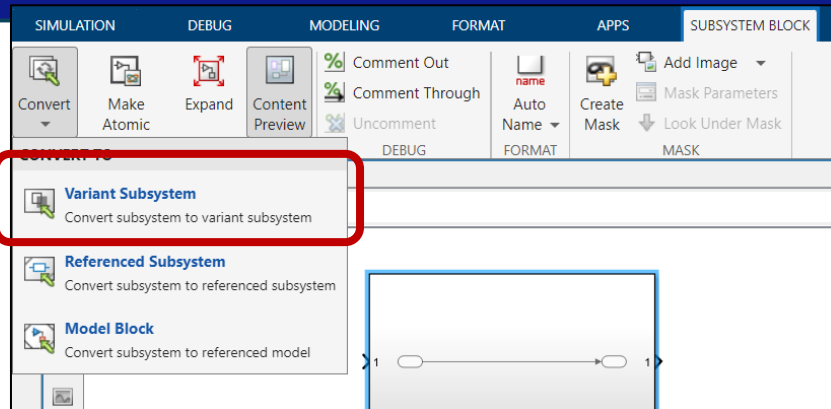
Need not be an atomic subsystem!

Model Reference

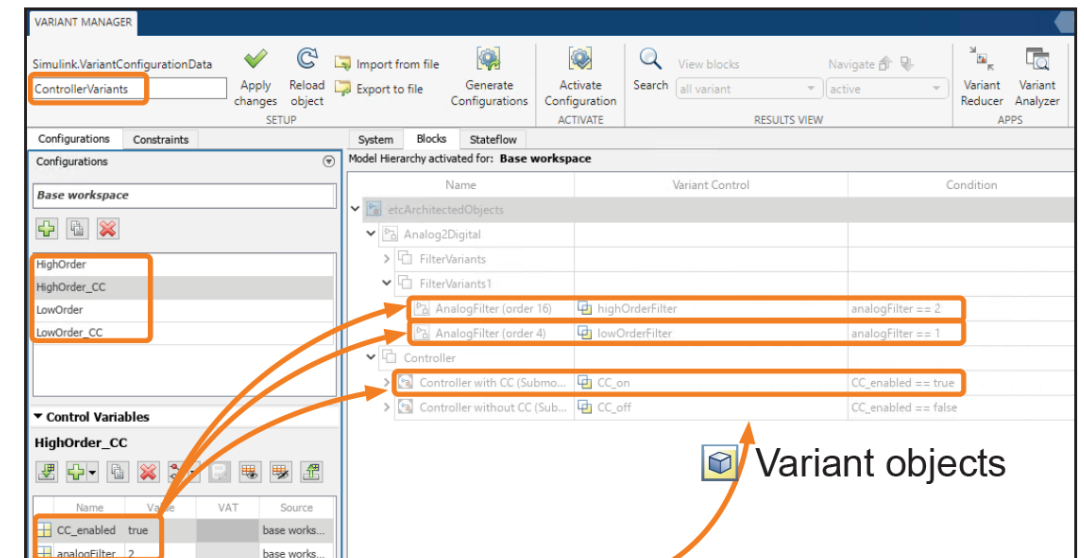
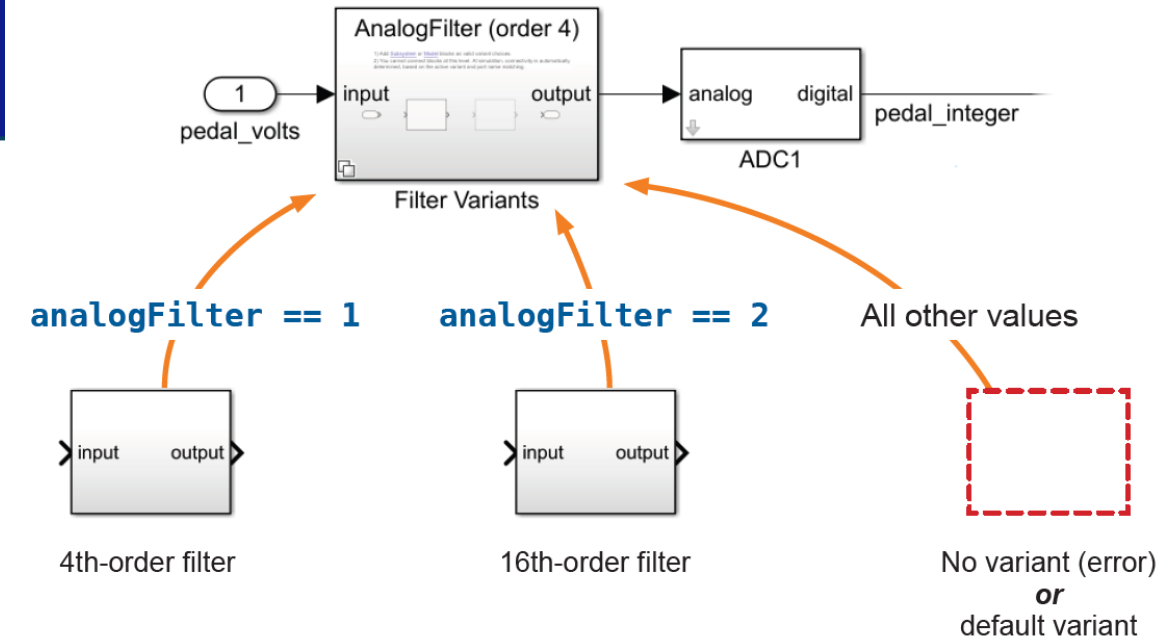
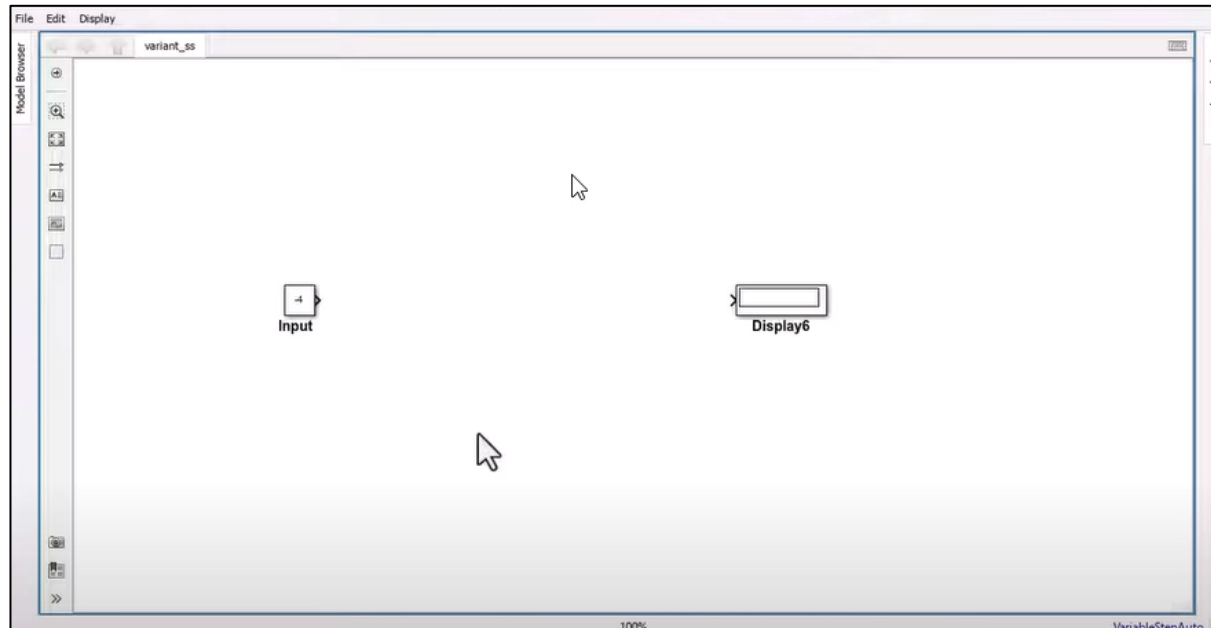


It has to be an **atomic** subsystem!

Variant Model



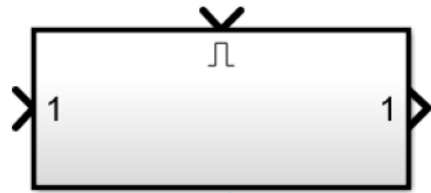
Need not be an atomic subsystem!



```
>> CC_on = Simulink.Variant;
>> CC_on.Condition = 'CC_enabled == true';
```

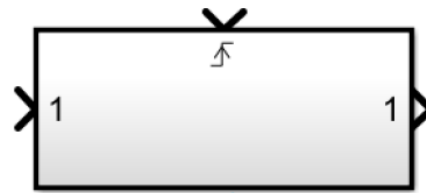
Conditionally Executed Subsystems

Analogy: Automatic Pilot



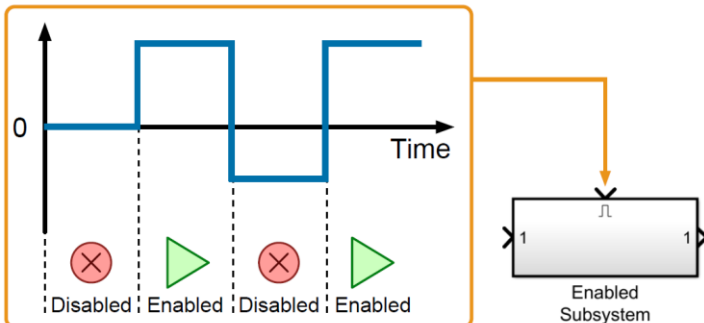
Enabled Subsystem

Analogy: Airbag System

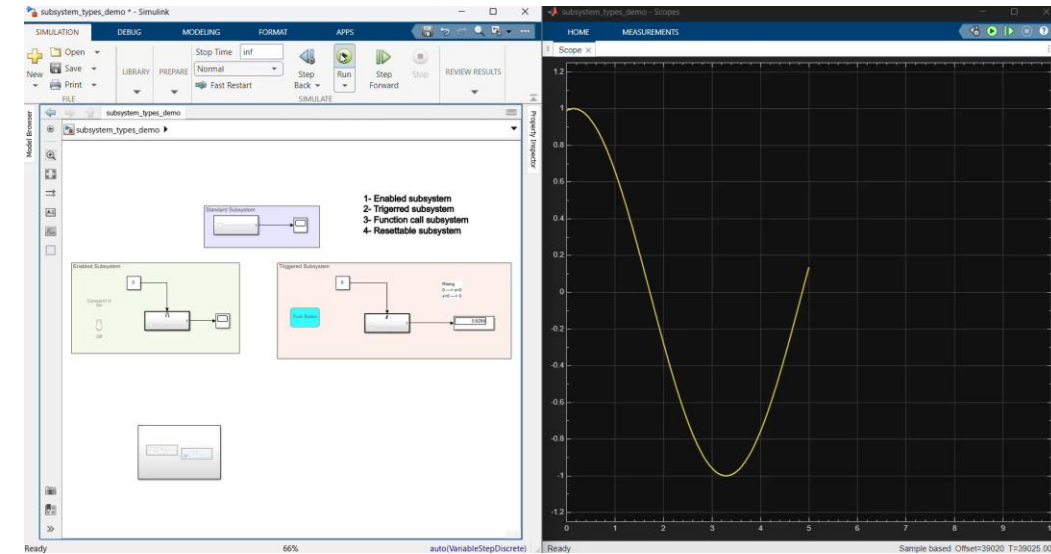
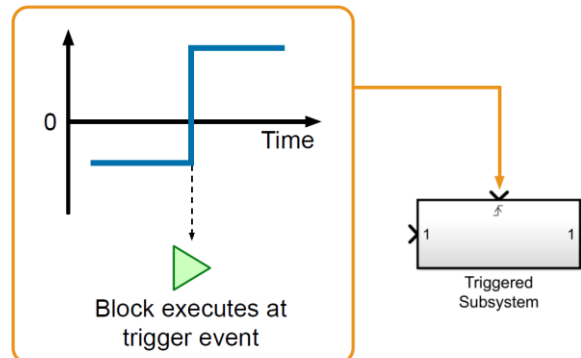


Triggered Subsystem

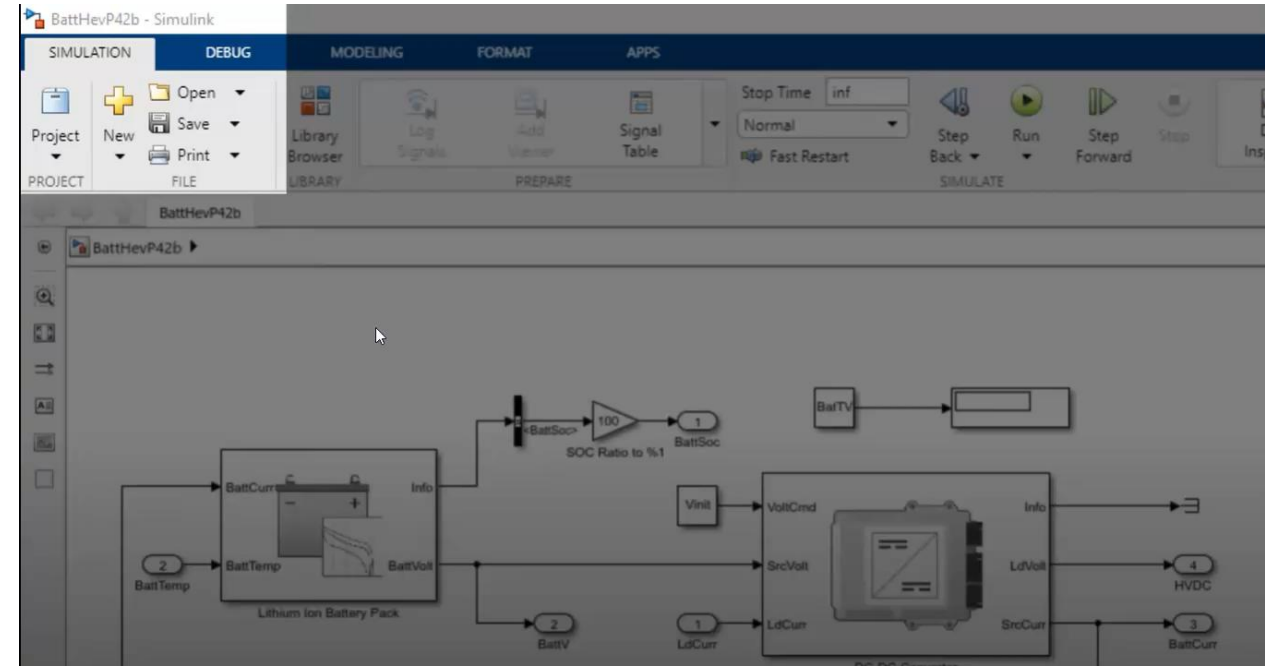
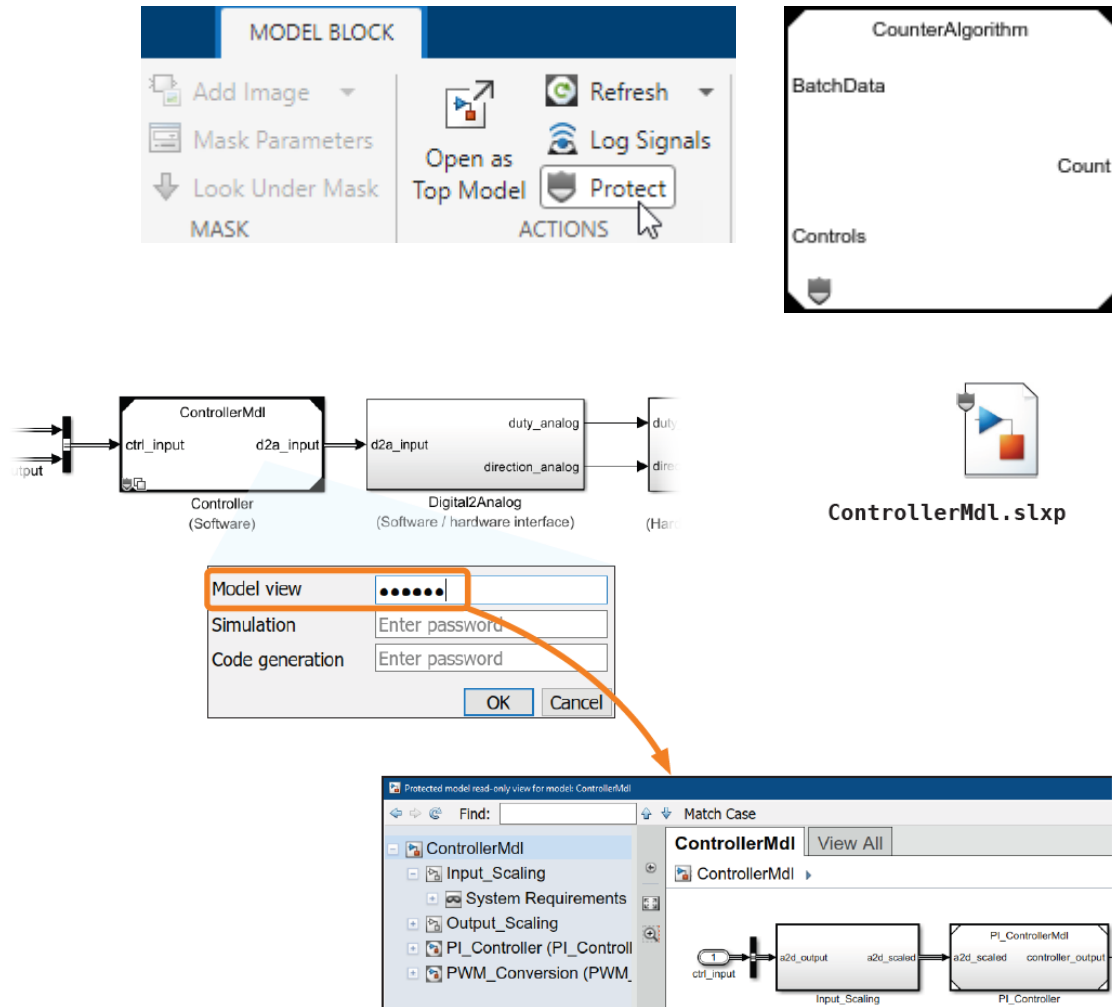
Enabled Subsystems



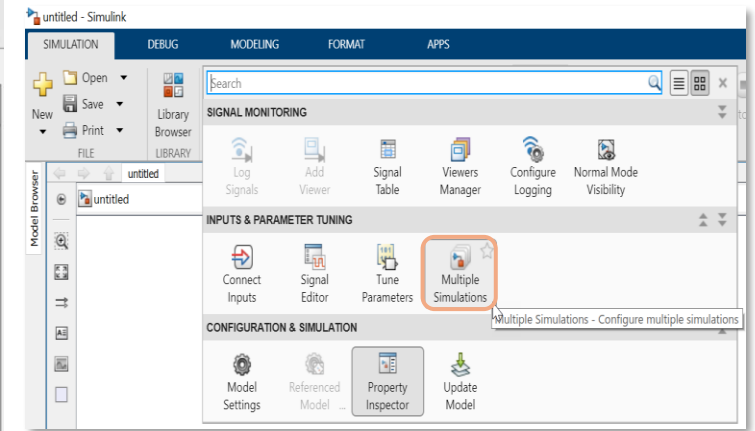
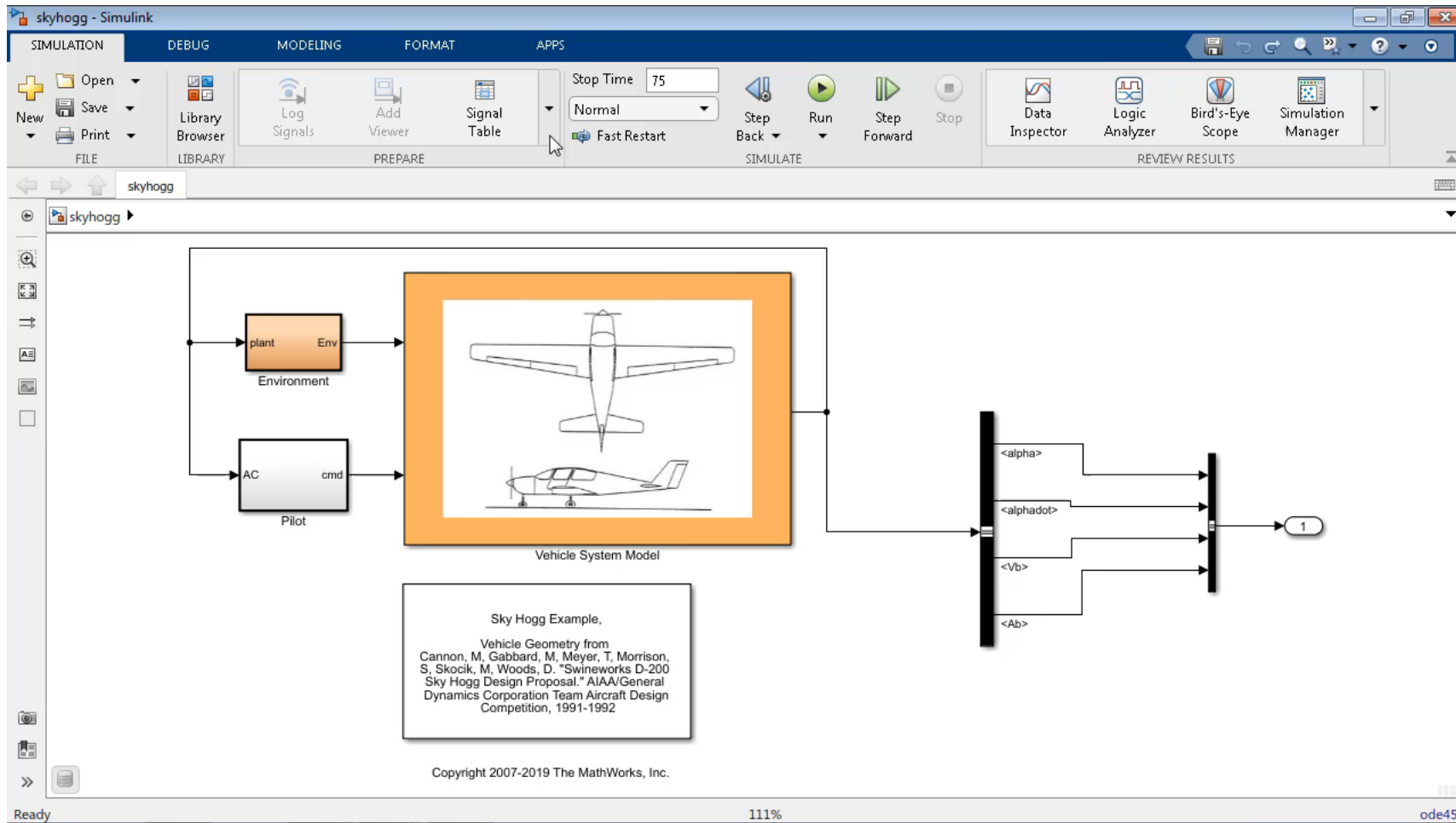
Triggered Subsystems



Protected Model



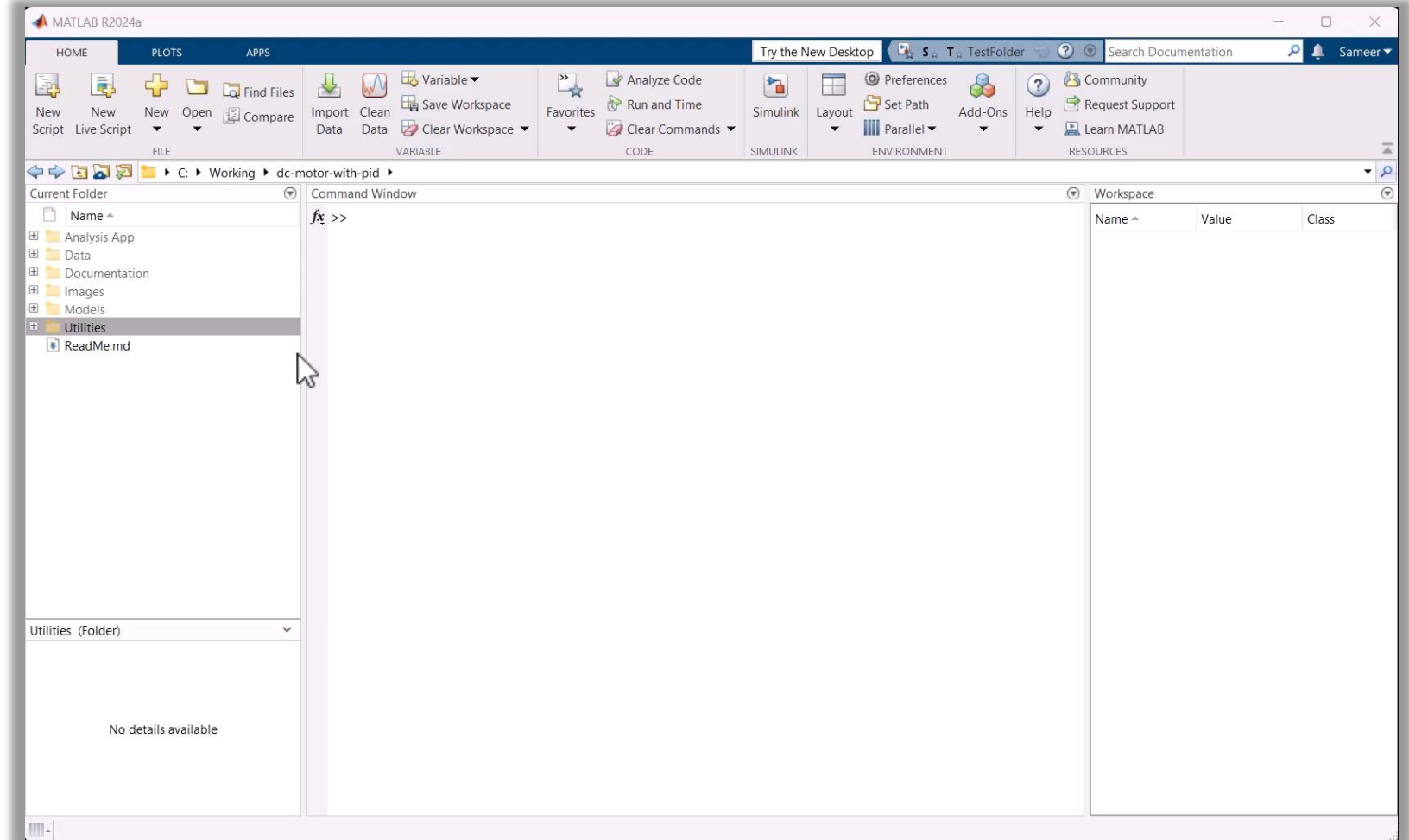
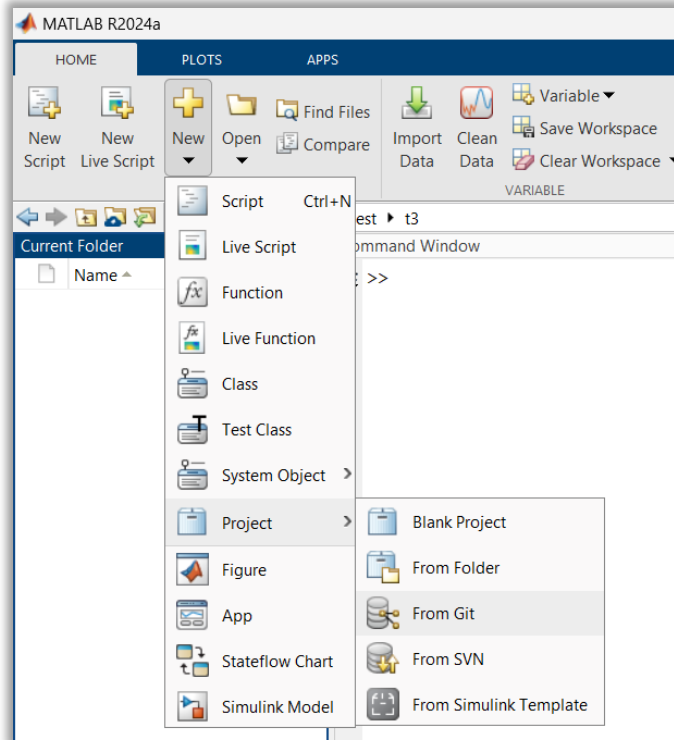
Multiple Simulations



Simulink Projects



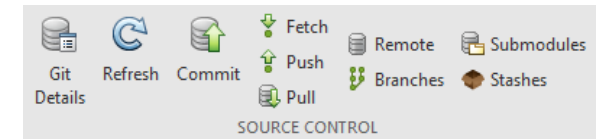
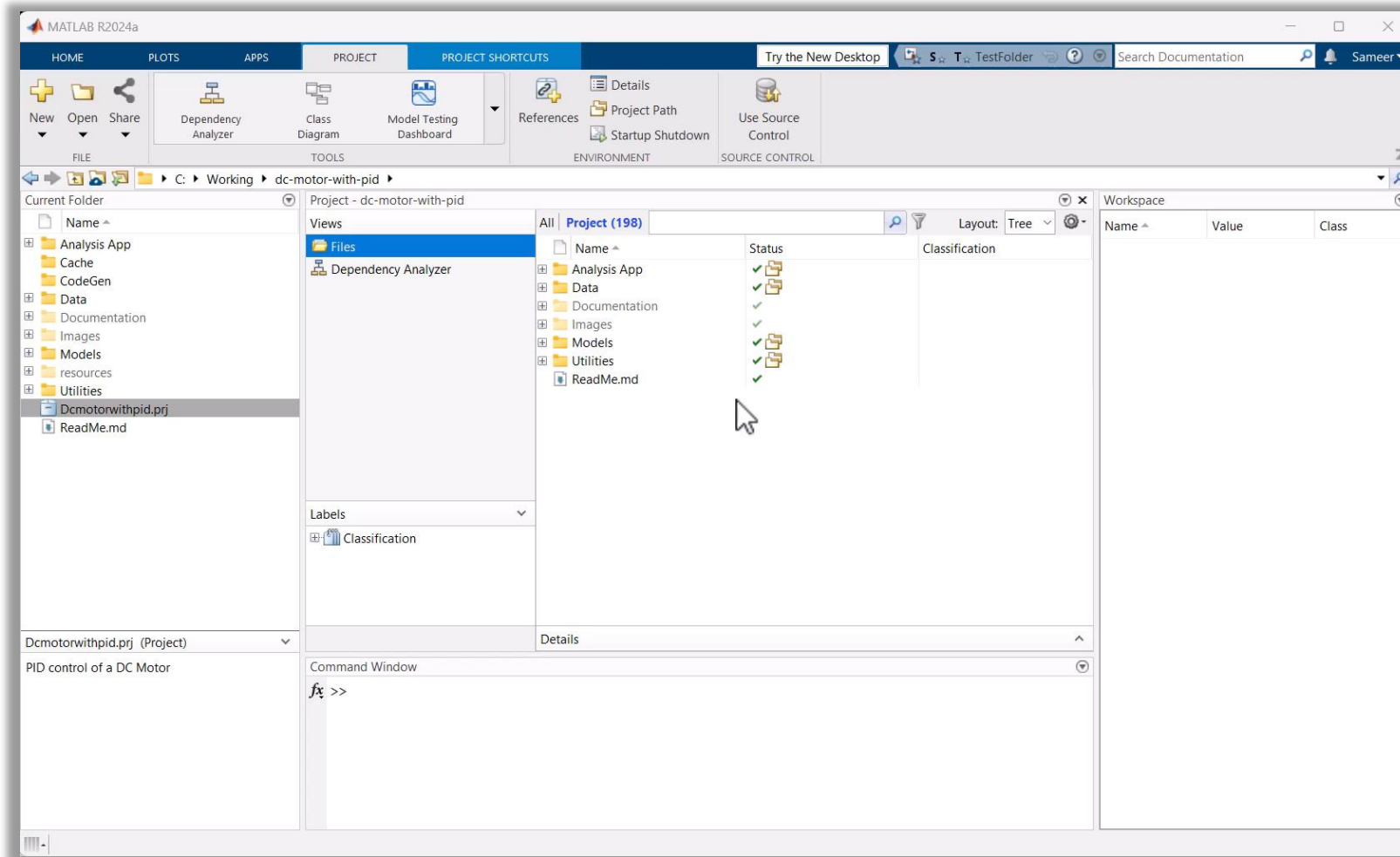
.prj



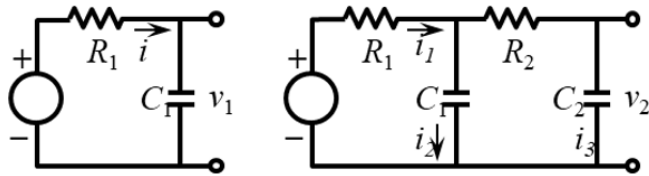
Simulink Projects



.prj



MathWorks Products Overview



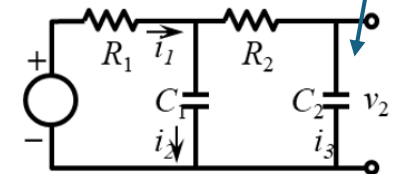
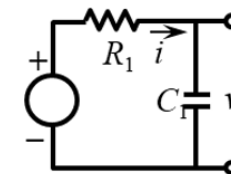
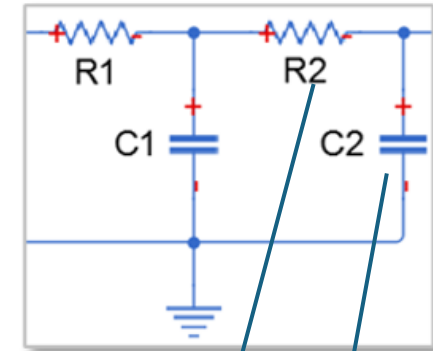
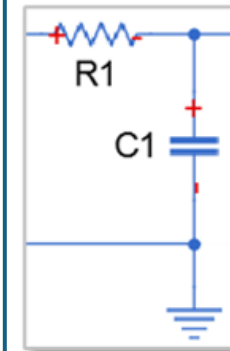
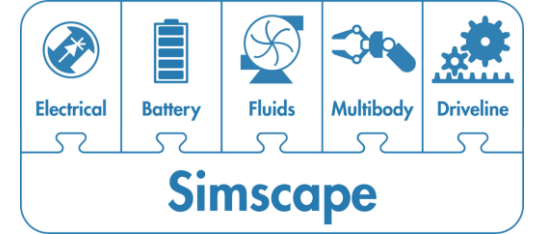
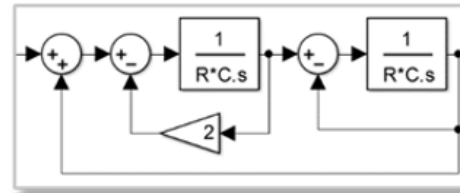
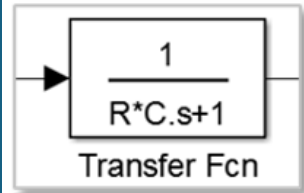
```
rc_circuit.m
% RC Circuit Parameters
R = 1e3; % Resistance in ohms (1 kOhm)
C = 1e-6; % Capacitance in farads (1 uF)
Vin = 5; % Input voltage (V)
tspan = [0 0.01]; % Time span for the simulation (10 ms)

% Differential Equation for the RC Circuit
% Vc' + (1/RC) * Vc = (1/RC) * Vin
% Define the ODE function
odefun = @(t, Vc) (Vin - Vc) / (R * C);

% Initial Condition (Vc(0) = 0)
Vc0 = 0;

% Solve the ODE
[t, Vc] = ode45(odefun, tspan, Vc0);

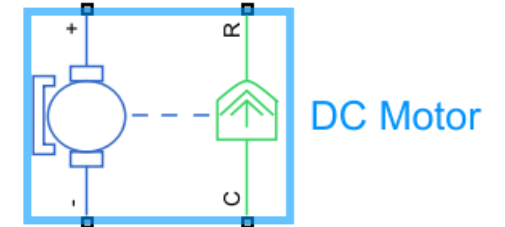
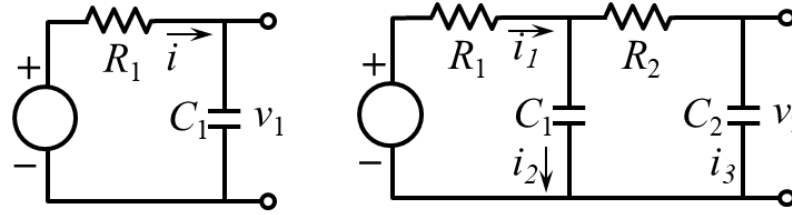
% Plot the results
figure;
plot(t, Vc, 'LineWidth', 2);
xlabel('Time (s)');
ylabel('Capacitor Voltage Vc (V)');
title('RC Circuit: Capacitor Voltage Over Time');
grid on;
```



Modeling Differences Between Simulink and Simscape

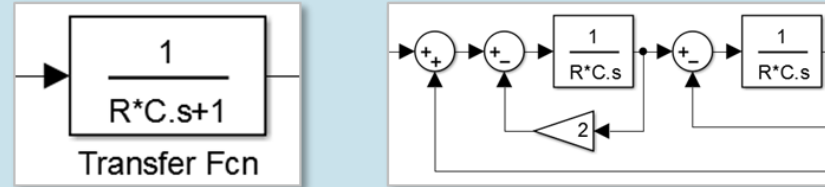
Simscape models are easy to build, simple to scale, and straightforward to understand

Systems To Be Modeled



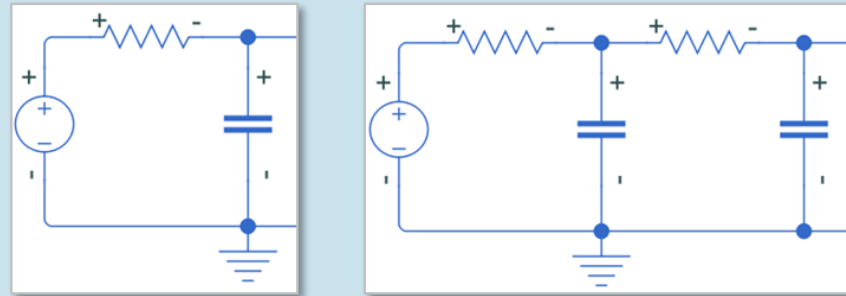
Simulink: Inputs → Outputs

- Model data flow from inputs to outputs
- Model behavior via equations



Simscape: Physical Networks

- Assemble schematics that directly represent the physical system
- Underlying equations are automatically derived
- Model bidirectional flow of energy



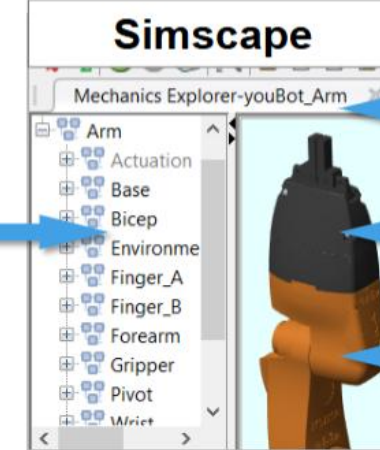
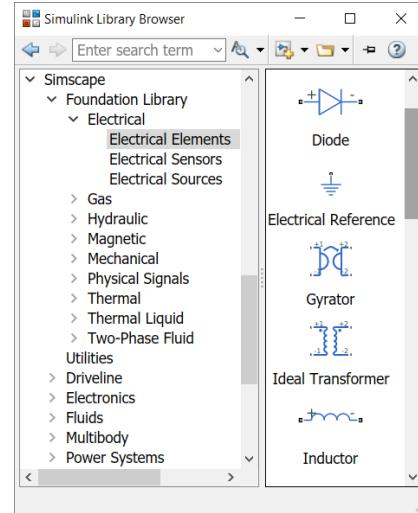
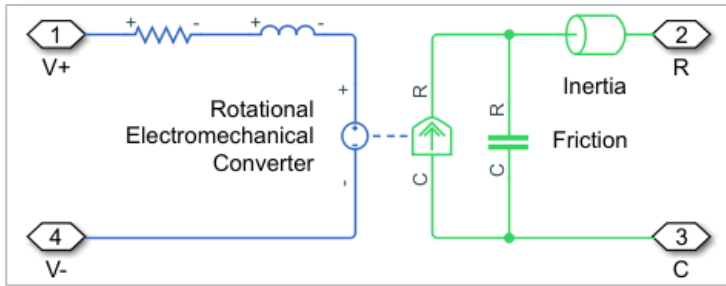
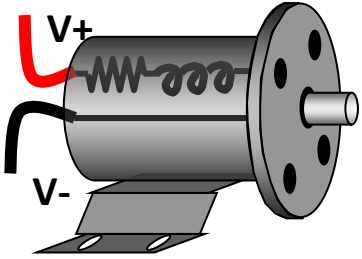
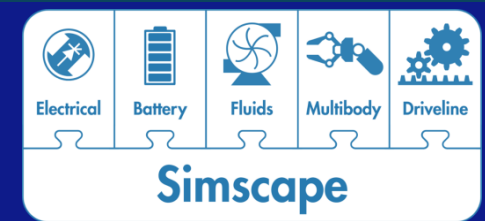
Block Parameters: DC Motor

DC Motor ☒ Auto Apply

Settings	Description
NAME	VALUE
Modeling option	No thermal port
Selected part	<click to select>
Electrical Torque	
Field type	Permanent magnet
Model parameterization	By equivalent circuit parameters
> Armature resistance	3.9 Ohm
Armature inductance	12e-6 H
Define back-emf or torque constant	Specify back-emf constant
> Back-emf constant	0.072e-3 V/rpm
Rotor damping parameterization	By damping value
Mechanical	
Rotor inertia	0.01 cm^2*g
Rotor damping	0 N*m*s/rad
> Initial rotor speed	0 rpm
Faults	
Armature winding fault	Add fault



MathWorks Products Overview - Simscape



SPICE

REFPROP

Maxwell

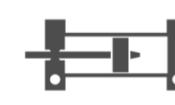
Simscape Fluid Component Models



Pumps



Valves



Actuators



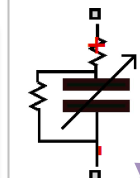
Pipes, Tanks



Heat Exchangers



$$i = (C_0 + C_v v) \frac{dv}{dt} + \frac{v}{r_d}$$



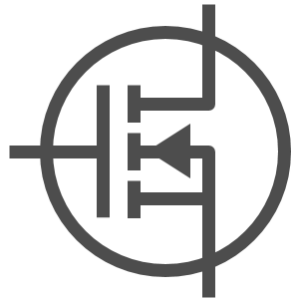
Lossy Ultracapacitor

```

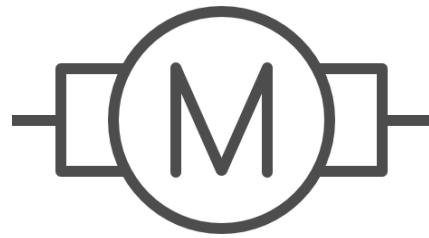
Editor - C:\MyComponents\LossyUltraCapacitor
40 equations
41 i == (C0 + Cv*vc)*vc.der + vc/Rd;
42 v == vc + i*R;
43 end
    
```



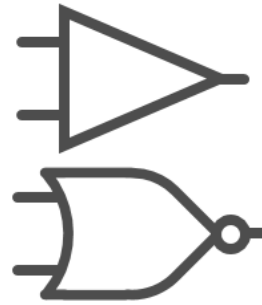
Simscape Electrical Component Models



**Semi-
conductors**



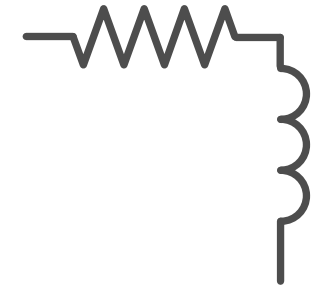
**Motors,
Actuators**



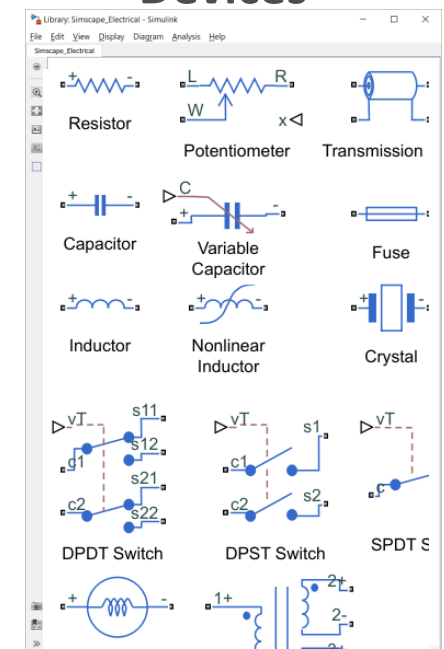
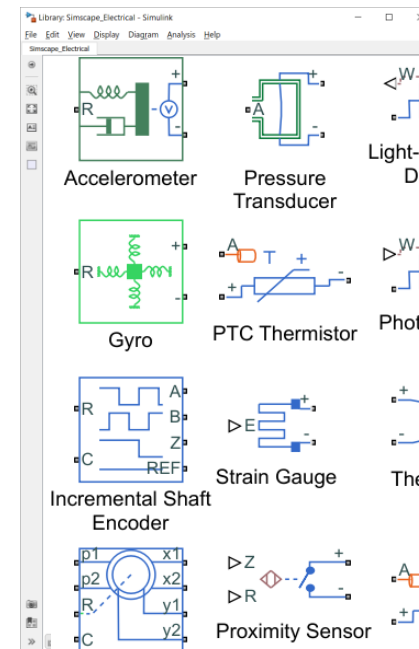
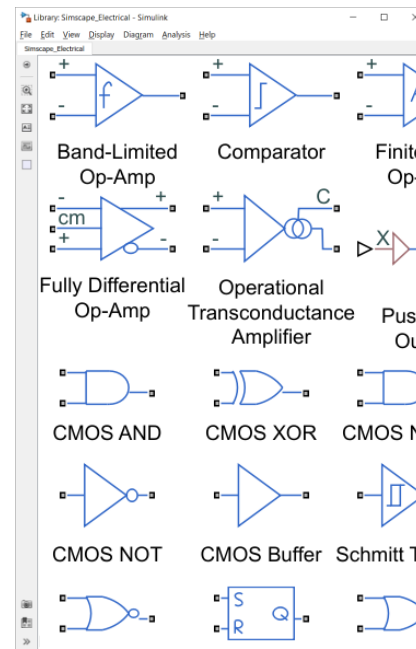
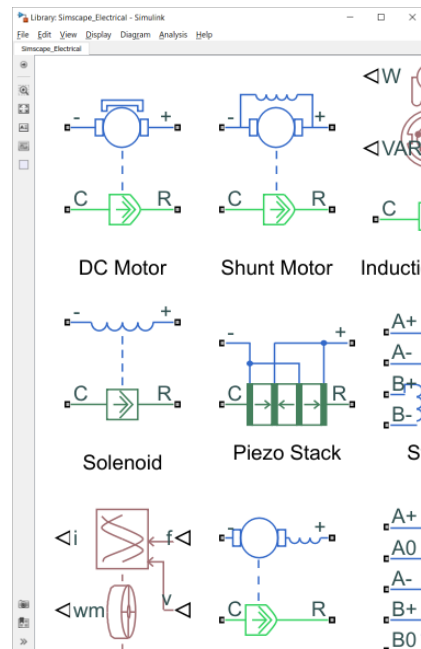
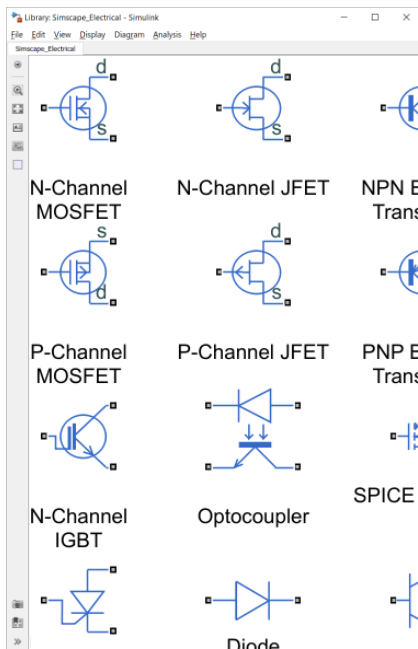
**Op-Amps,
Logic Gates**



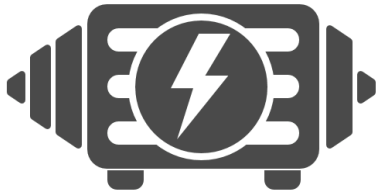
Sensors



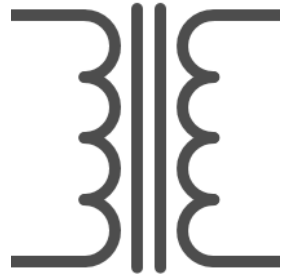
**Passive
Devices**



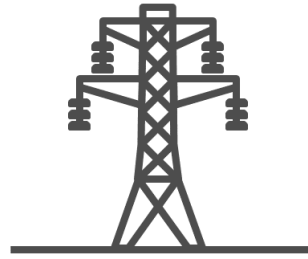
Simscape Electrical Component Models



Generators, Motors



Transformers



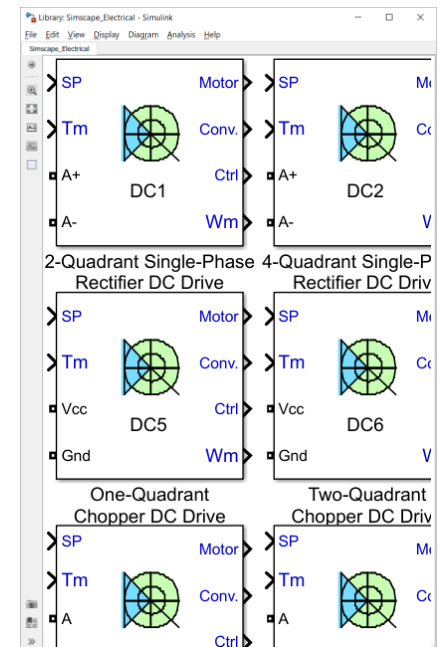
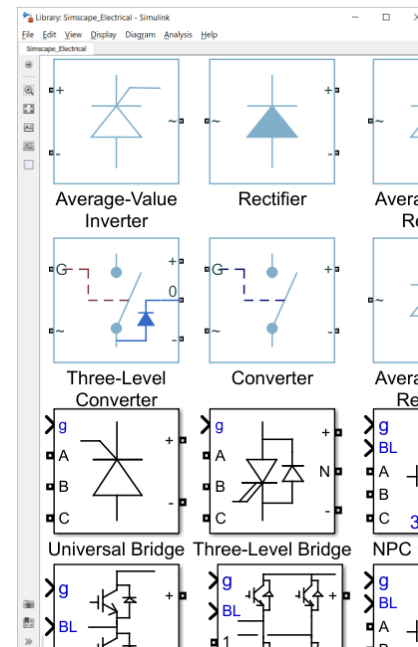
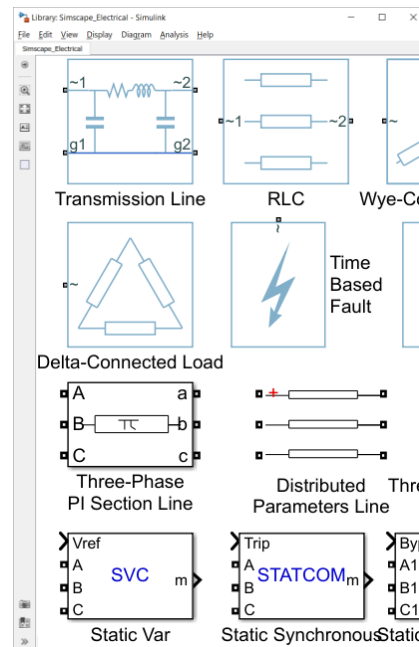
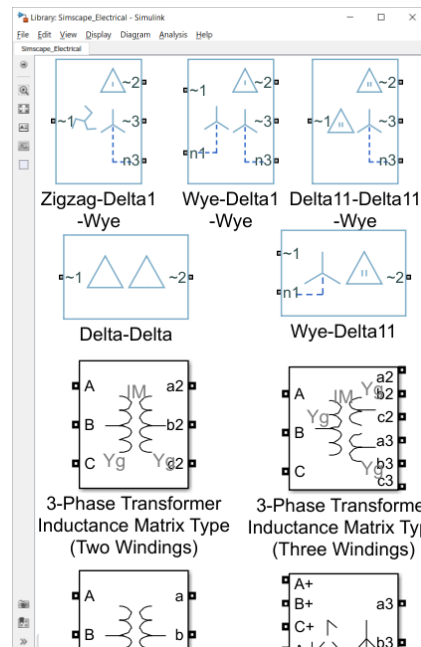
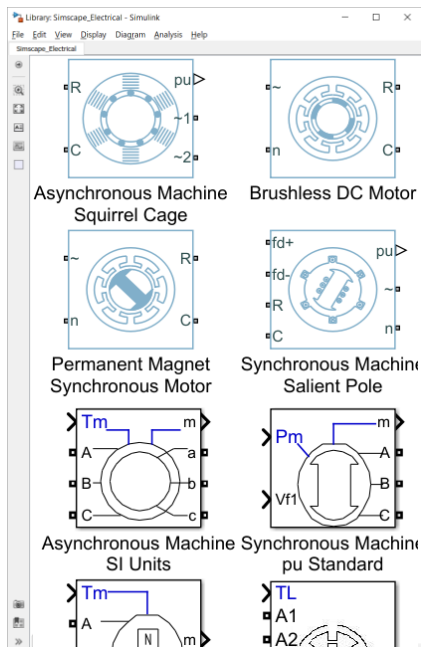
Lines,
FACTS



Converters

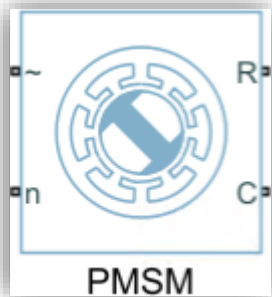


Electric
Drives



Manufacturer Specific Electric Motors

- Database with parameter values to match **vendor**-specific motors

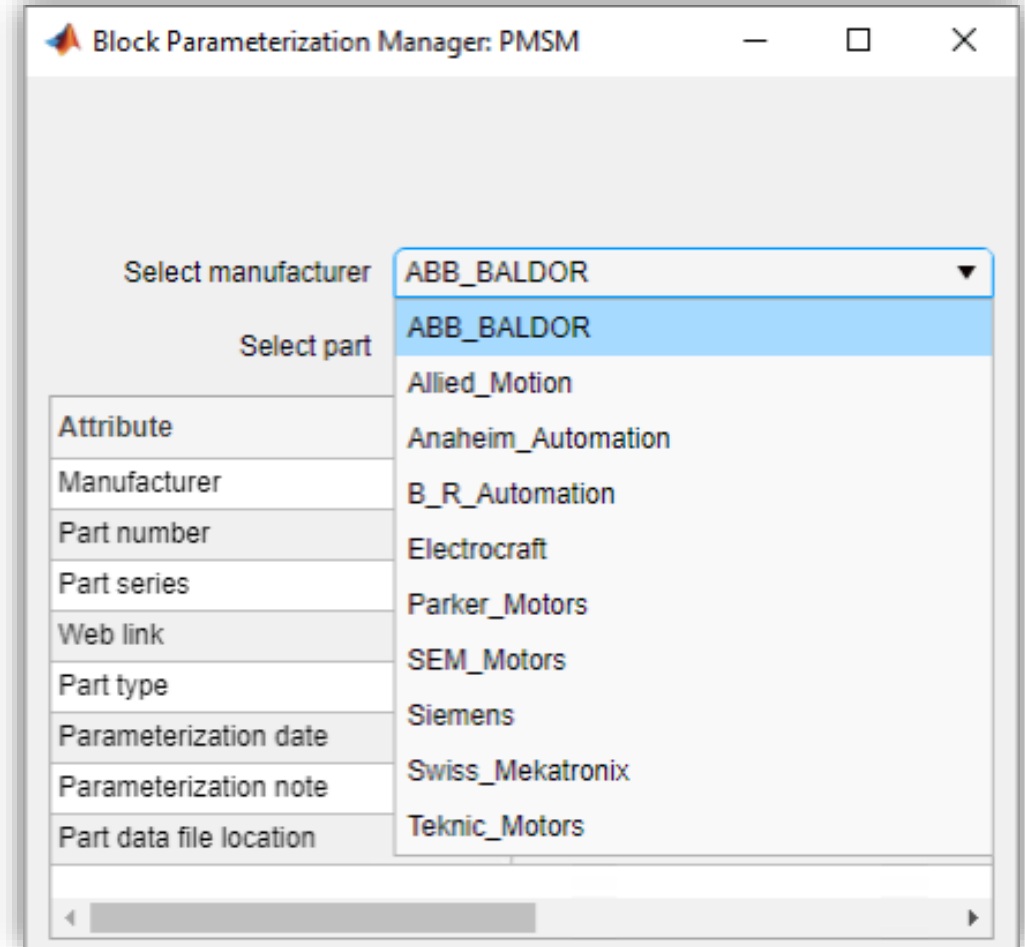


PMSM

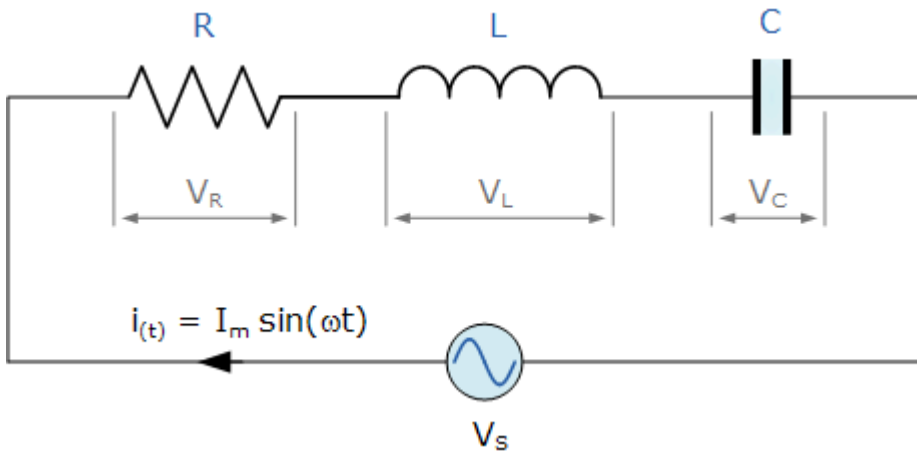
This block represents a permanent magnet synchr

Right-click on the block and select Simscape block

[Select a predefined parameterization](#)



RLC Circuit Modeling



Transfer functions form:

$$I(s) = \frac{1}{R + sL + \frac{1}{sC}} V_{in}(s)$$

$$H(s) = \frac{I(s)}{V_{in}(s)} = \frac{sC}{LCs^2 + RCs + 1}$$

$$\text{Total empedans: } Z(s) = R + sL + \frac{1}{sC}$$

$$\text{Kirchoff voltage law : } v_{in}(t) = v_R(t) + v_L(t) + v_C(t)$$

$$v_R = Ri(t)$$

$$v_L = L \frac{di(t)}{dt}$$

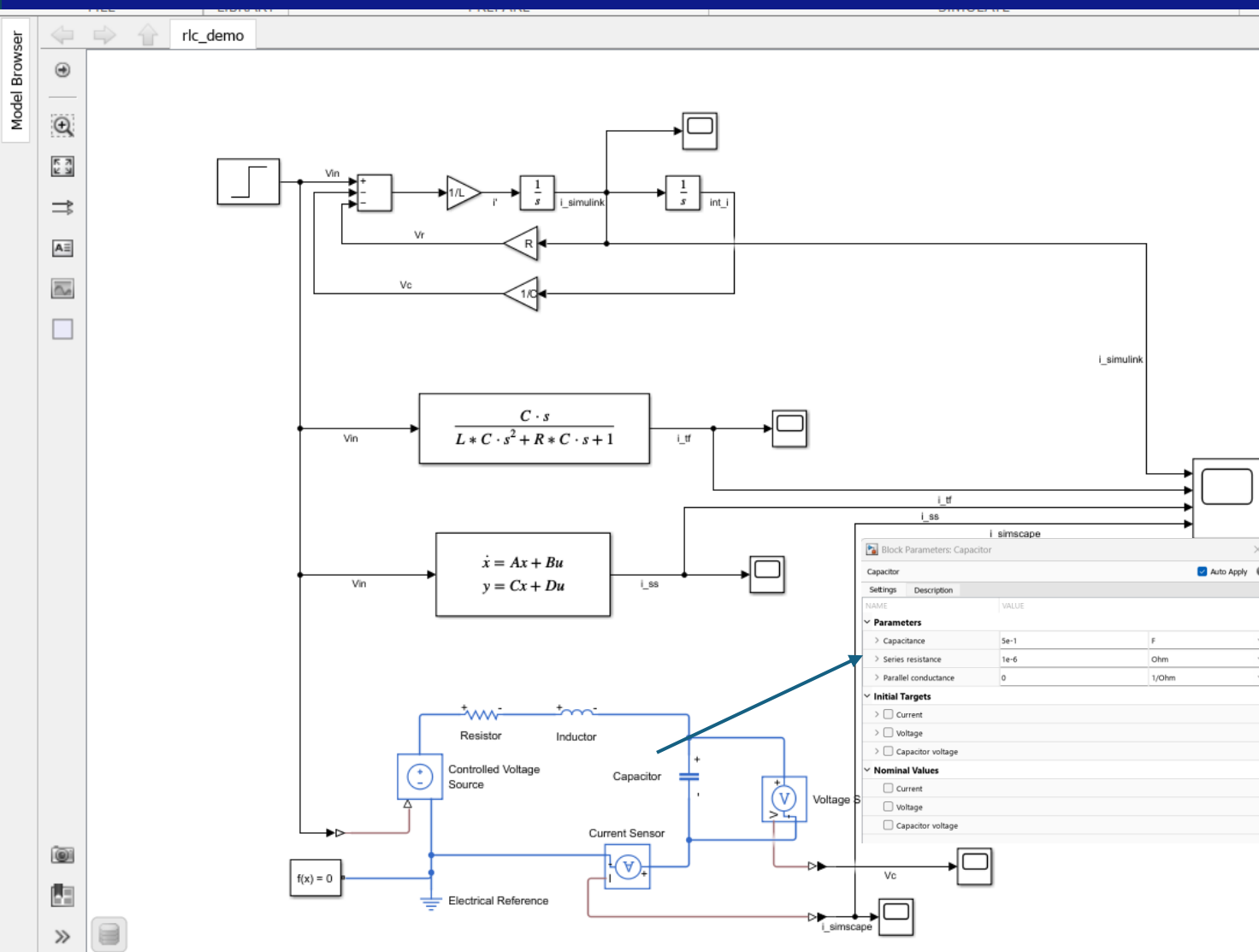
$$v_C = \frac{1}{C} \int i(t) dt$$

$$\text{State-space form: } x_1 = i(t) \quad x_2 = v_C(t)$$

$$\begin{aligned} \dot{x}_1 &= \frac{1}{L} (v_{in}(t) - Rx_1 - x_2) \\ \dot{x}_2 &= \frac{x_1}{C} \end{aligned} \quad y(t) = i(t)$$



RLC Circuit Modeling



$$v_{in}(t) = Ri(t) + L \frac{di(t)}{dt} + \frac{1}{C} \int i(t) dt$$

$$H(s) = \frac{I(s)}{V_{in}(s)} = \frac{sC}{LCs^2 + RCs + 1}$$

Durum değişkenleri

$$x_1 = i(t), \quad x_2 = v_C(t)$$

State-space modeli

$$\dot{x}_1 = \frac{1}{L} (v_{in}(t) - Rx_1 - x_2)$$

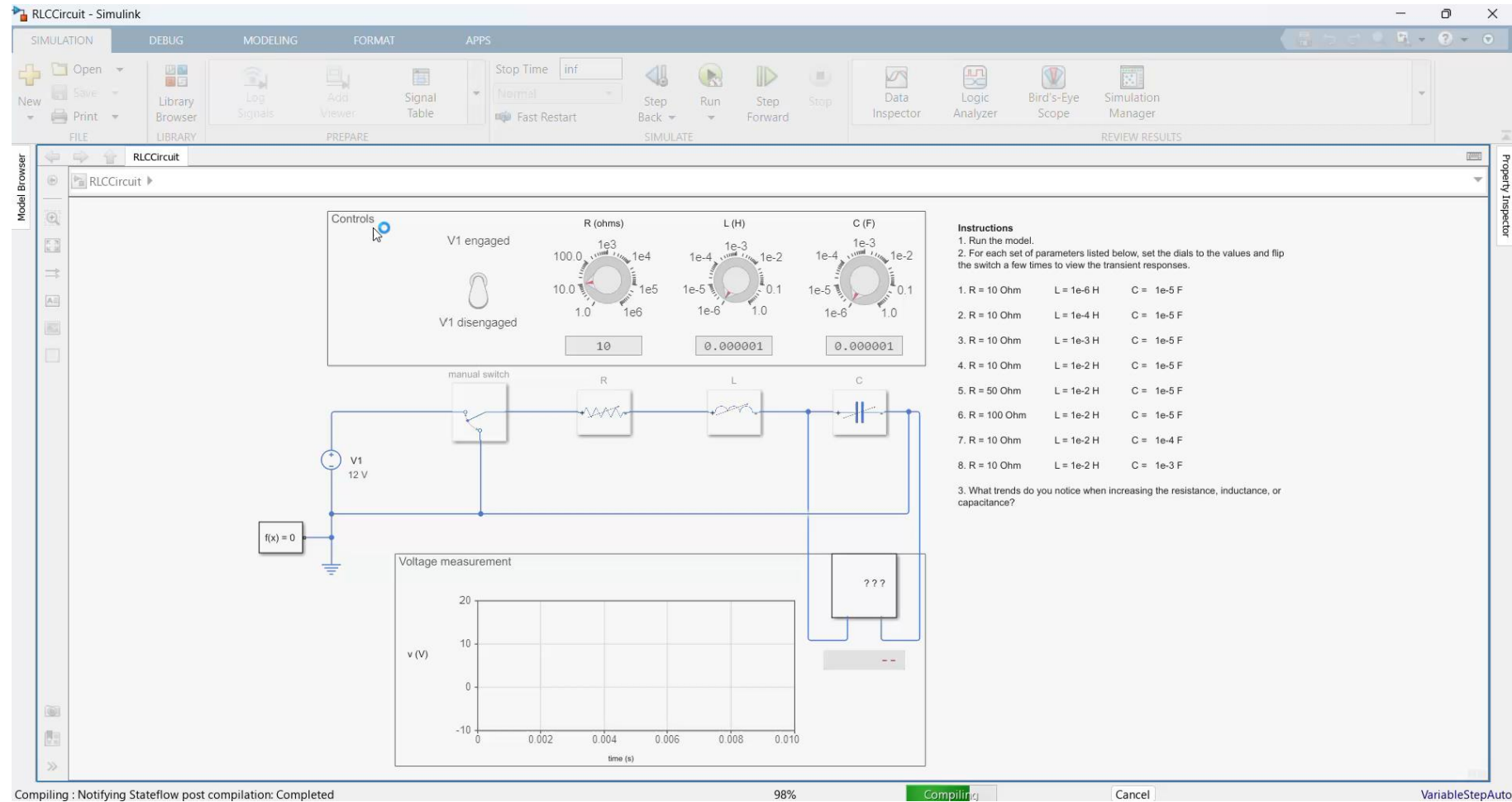
$$\dot{x}_2 = \frac{x_1}{C}$$

Çıkış

$$y(t) = i(t) = x_1$$



RLC Circuit Modeling



Can you Find the Mistakes?

```
function Logic = fcn(GoodComms, GCSCmds, IsBall, AngularState, ...
    TranslationalState, IMUStatus, BATT_SENS)
persistent LogicOut count;

if isempty(LogicOut)
    LogicOut.CalibrateSensors = false;
    LogicOut.Mode = uint8(1);
    count = 0;
end

TORampTime = 2;
ts_comms = 0.2;
TM_SettleTime = 2;

switch LogicOut.Mode
case 1 %WaitForComms
    if GoodComms && IMUStatus.GoodIMUData && BATT_SENS > 3.3
        LogicOut.Mode = uint8(2);
    end
case 2 %Init
    if GCSCmds.GCS_CalibrateCmd
        LogicOut.Mode = uint8(3);
        LogicOut.CalibrateSensors = true;
    end
case 3 %Calibration
    if IMUStatus.CalibrationDone
        LogicOut.Mode = uint8(4);
    elseif GCSCmds.GCS_EMERGENCY_OFF || ~GoodComms
        LogicOut.Mode = uint8(10);
        count = 0;
    end
    LogicOut.CalibrateSensors = false;
case 4 %Standby
    if CrashCheck(GoodComms, GCSCmds, AngularState)
        LogicOut.Mode = uint8(10);
        count = 0;
    elseif GCSCmds.GCS_MissionMode == 1
        LogicOut.Mode = uint8(6);
        count = 0;
    elseif GCSCmds.GCS_CalibrateCmd
        LogicOut.Mode = uint8(3);
        LogicOut.CalibrateSensors = true;
    elseif BATT_VOL > 2 && GCSCmds.GCS_TAKEOFF
        LogicOut.Mode = uint8(6);
    end
case 5 %Takeoff
```

```
    if CrashCheck(GoodComms, GCSCmds, AngularState)
        LogicOut.Mode = uint8(10);
        count = 0;
    elseif ~GoodComms
        LogicOut.Mode = uint8(9);
    elseif count >= uint8((TORampTime+2)/ts_comms) && IsBall
        LogicOut.Mode = uint8(7);
    elseif GCSCmds.GCS_TestCmd
        LogicOut.Mode = uint8(6);
        count = 0;
    elseif BATT_VOL <= 2
        LogicOut.Mode = uint8(4);
    end
    count = count + 1;
case 6 %TestManeuvers
    if CrashCheck(GoodComms, GCSCmds, AngularState)
        LogicOut.Mode = uint8(10);
        count = 0;
    elseif count >= uint8(TM_SettleTime/ts_comms - 1)
        if IsBall
            LogicOut.Mode = uint8(7);
        else
            LogicOut.Mode = uint8(8);
            count = 0;
        end
    end
    count = count + 1;
case 7 %Track3D
    if GCSCmds.GCS_MissionMode == 2 || ~GoodComms || ...
        BATT_SENS <= 2
        LogicOut.Mode = uint8(9);
    elseif CrashCheck(GoodComms, GCSCmds, AngularState)
        LogicOut.Mode = uint8(10);
        count = 0;
    elseif ~IsBall
        LogicOut.Mode = uint8(8);
        count = 0;
    elseif GCSCmds.GCS_TestCmd
        LogicOut.Mode = uint8(6);
        count = 0;
    end
case 8 %LostBall
```

```
    if CrashCheck(GoodComms, GCSCmds, AngularState)
        LogicOut.Mode = uint8(10);
        count = 0;
    elseif count >= (3/ts_comms - 1) || ~GoodComms
        LogicOut.Mode = uint8(9);
    elseif IsBall
        LogicOut.Mode = uint8(7);
    elseif GCSCmds.GCS_TestCmd
        LogicOut.Mode = uint8(6);
        count = 0;
    end
    count = count + 1;
case 9 %Land
    if abs(AngularState.Angles(1)) < 0.08727 && ...
        abs(AngularState.Angles(2)) < 0.08727 && ...
        TranslationalState.Acc_scaled(3) < -15
        LogicOut.Mode = uint8(1);
        count = 0;
    elseif CrashCheck(GoodComms, GCSCmds, AngularState)
        LogicOut.Mode = uint8(10);
        count = 0;
    end
case 10 %EmergencyOff
    count = count + 1;
end %End Switch

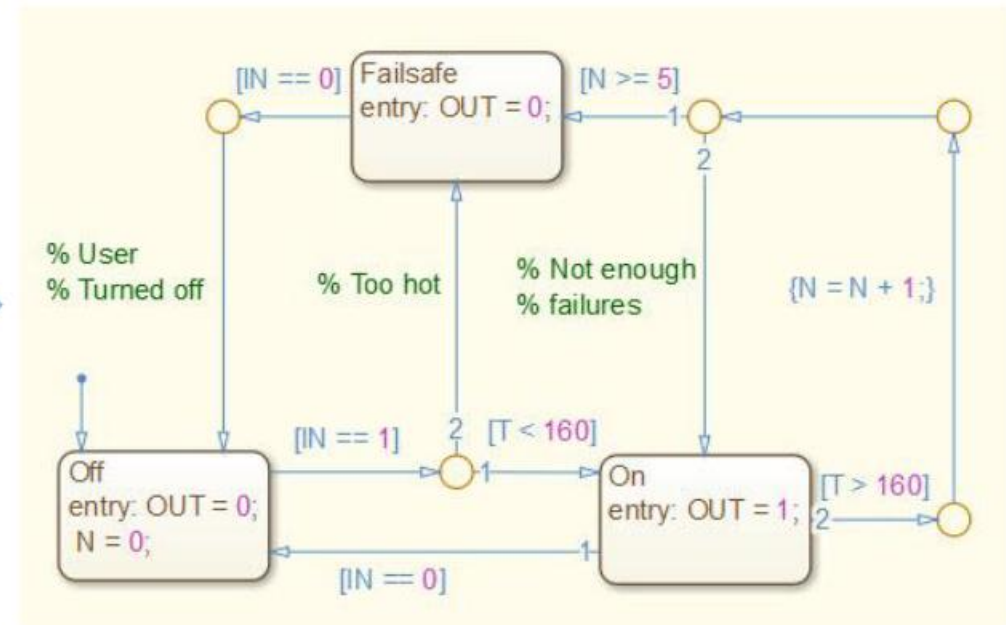
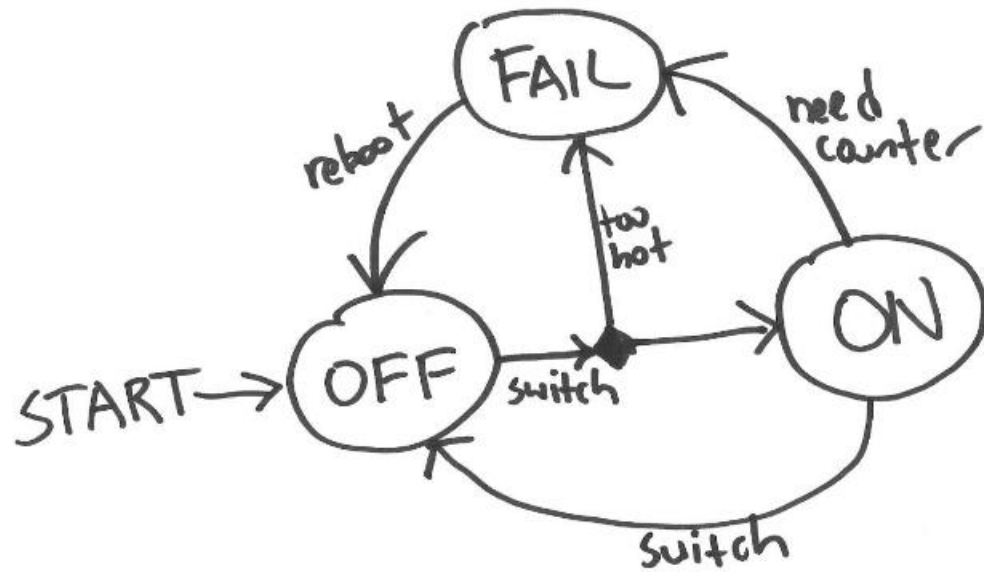
Logic = LogicOut;
end %End Function
```

```
function crash = CrashCheck(GoodComms, GCSCmds, AngularState)
crash = false;
if GCSCmds.GCS_EMERGENCY_OFF || ~GoodComms || ...
    AngularState.Angles(1) > 0.5235 || ...
    AngularState.Angles(2) > 0.5235
    crash = true;
end
end
```



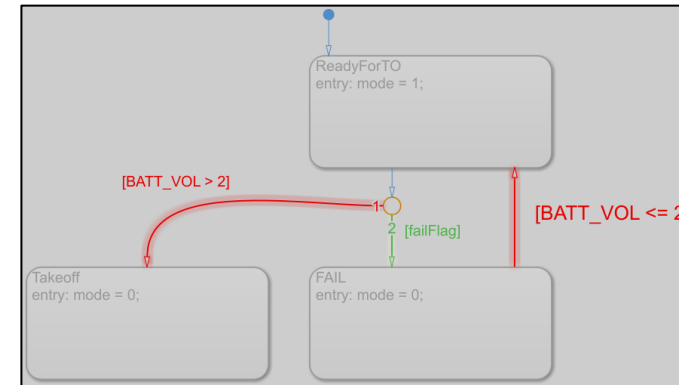
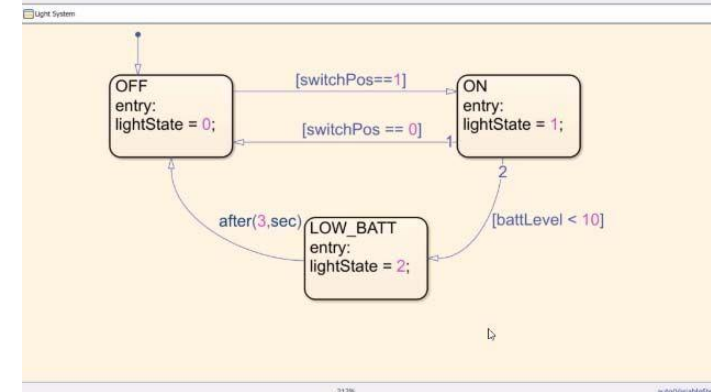
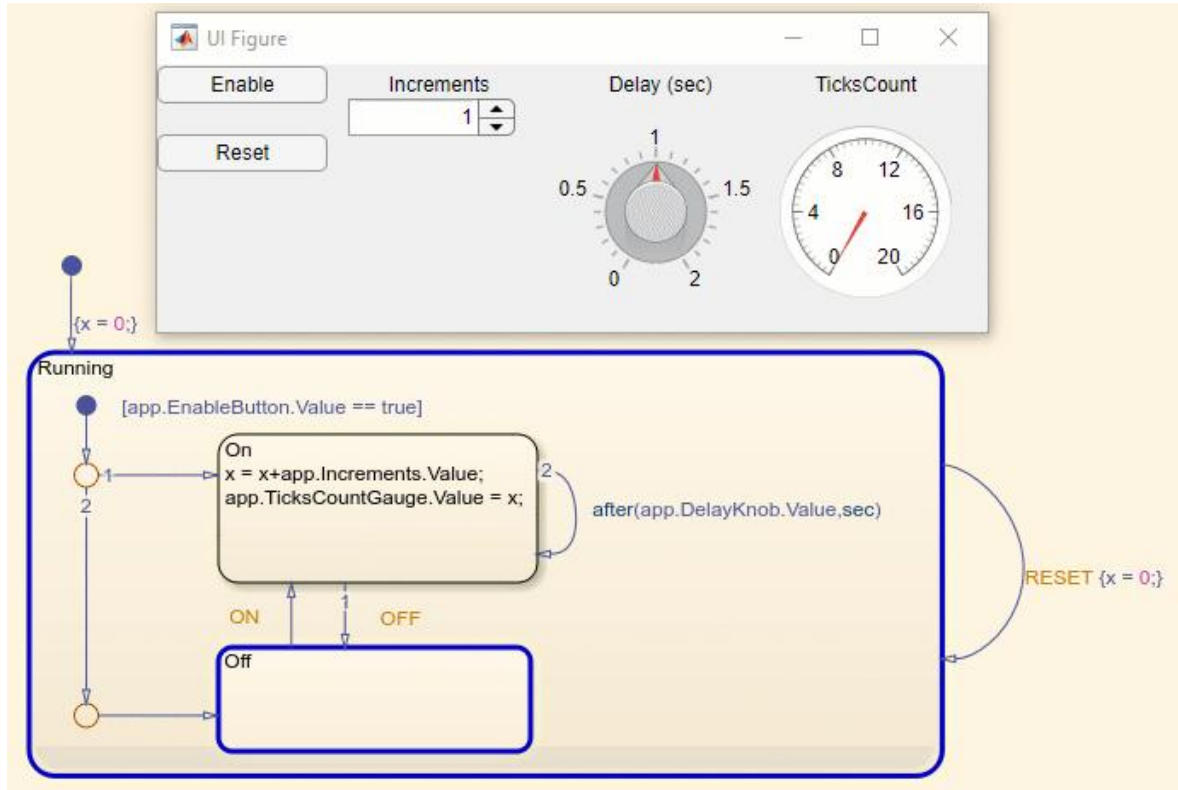
When should I switch from Simulink to Stateflow?

- A graphical state machine design environment that sits on top of Simulink
- What is a state machine?



MathWorks Products Overview - Stateflow

- Easily design and maintain **complex logic**
- Animations help **visually debug systems**



```
if (untitled_DW.is_active_c3_untitled == 0U) {
    /* Entry: Chart */
    untitled_DW.is_active_c3_untitled = 1U;

    /* Entry Internal: Chart */
    /* Transition: '<S1>:5' */
    untitled_DW.is_c3_untitled = untitled_IN_ReadyForTO;

    /* Entry 'ReadyForTO': '<S1>:3' */
} else {
    switch (untitled_DW.is_c3_untitled) {
        case untitled_IN_FAIL:
            /* During 'FAIL': '<S1>:1' */
            /* Transition: '<S1>:2' */
            untitled_DW.is_c3_untitled = untitled_IN_ReadyForTO;

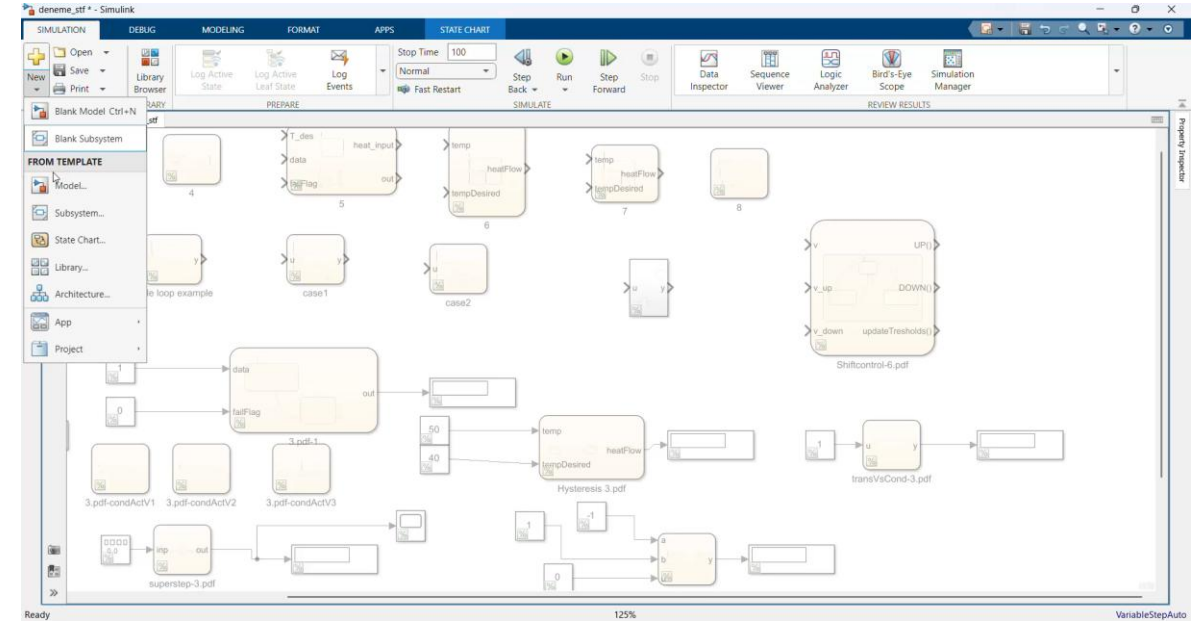
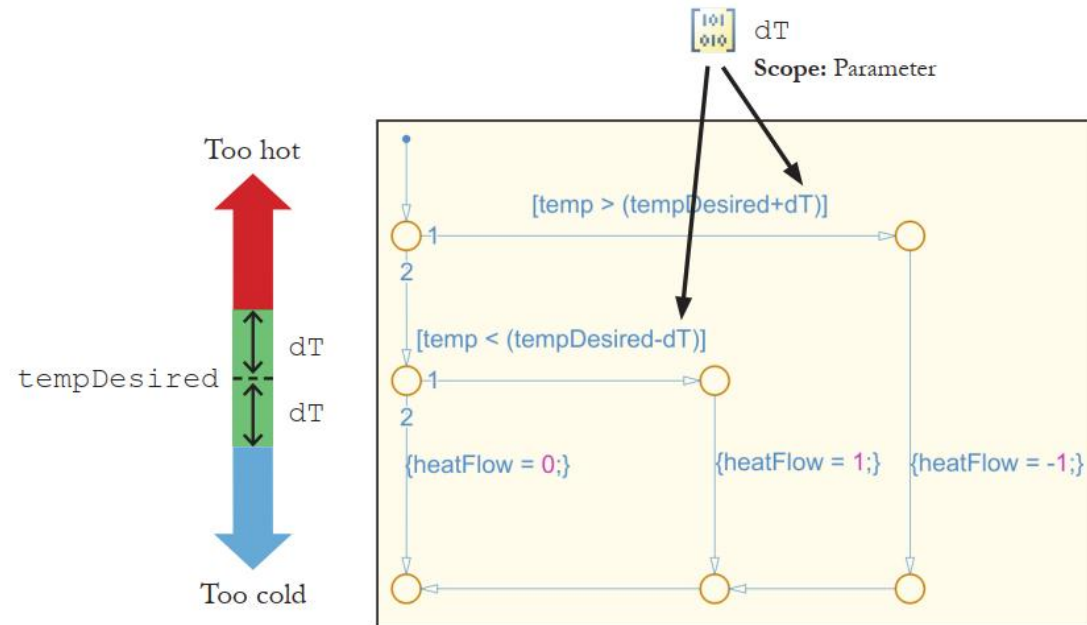
            /* Entry 'ReadyForTO': '<S1>:3' */
            break;

        case untitled_IN_ReadyForTO:
            /* During 'ReadyForTO': '<S1>:3' */
            /* Transition: '<S1>:4' */
            break;

        default:
            /* During 'Takeoff': '<S1>:8' */
            break;
    }
}
```

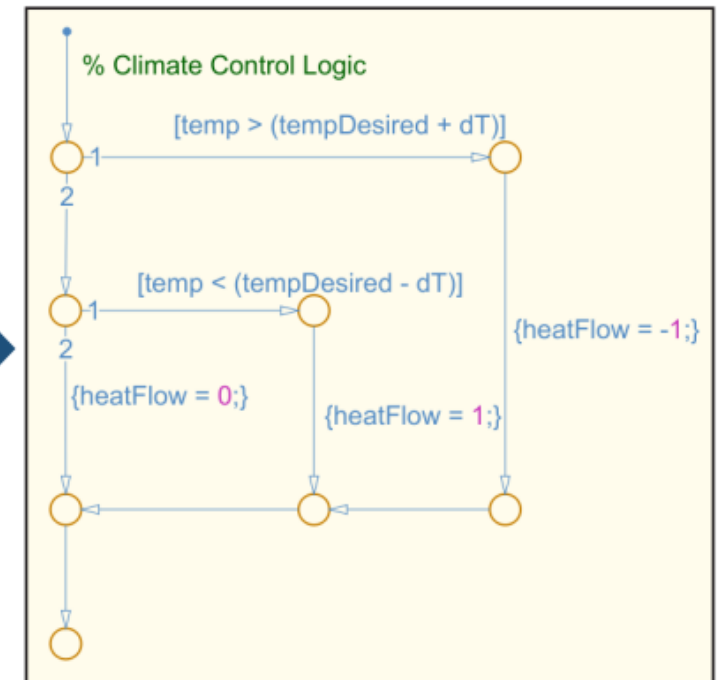
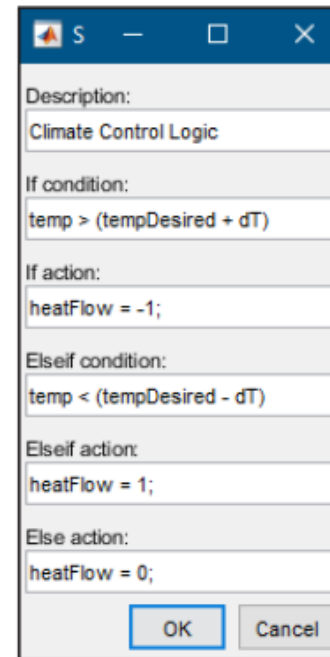
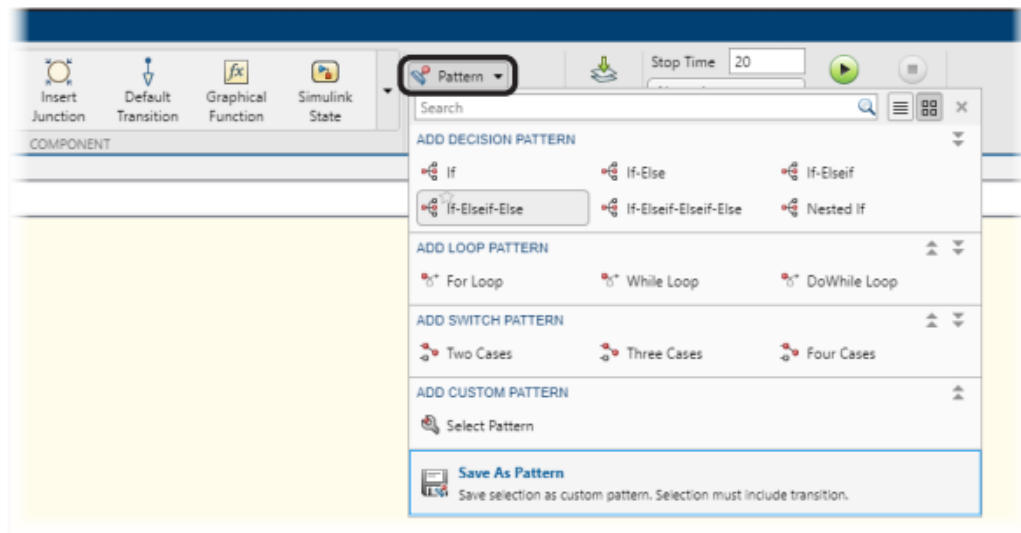


Modeling Flow Charts



Pattern Wizard

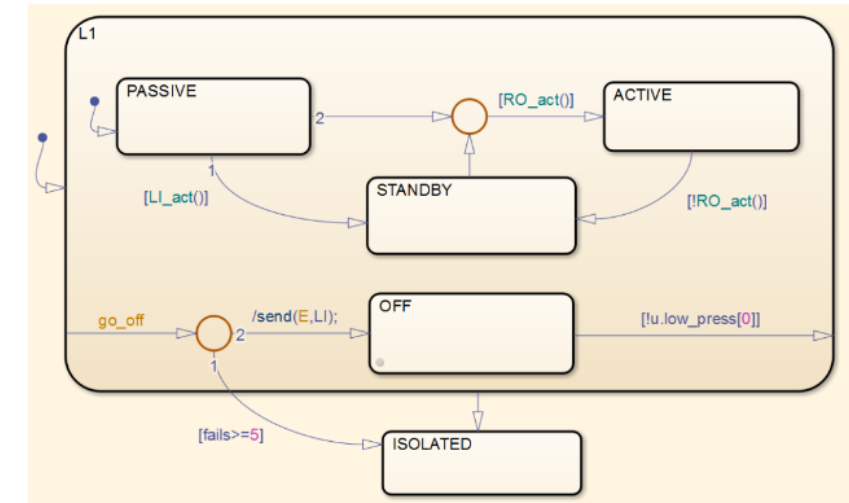
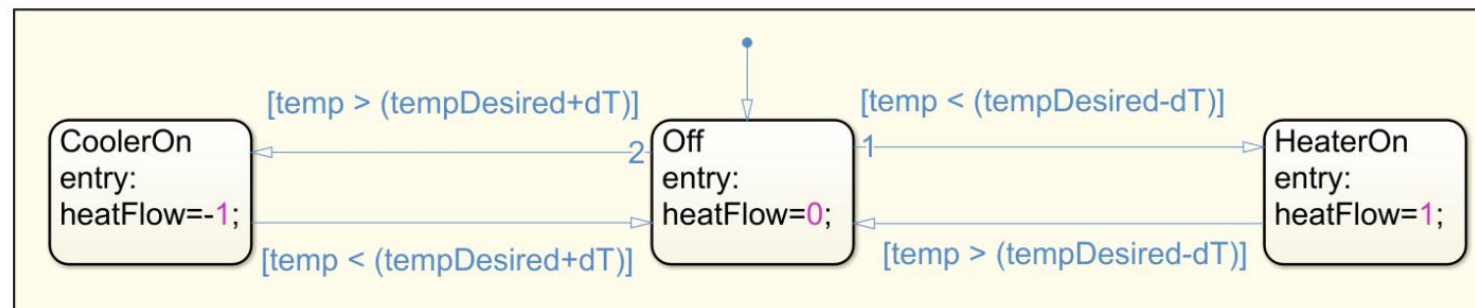
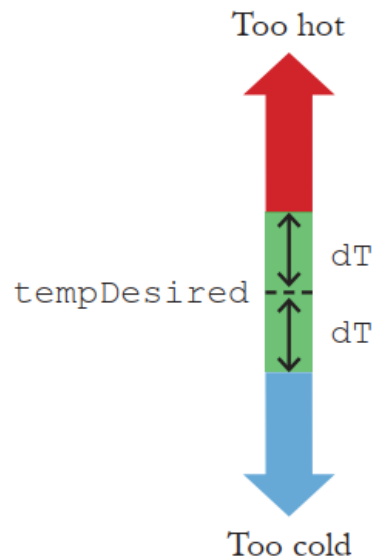
- There are standard patterns that exist for modeling common programming constructs (such as if-elseif and for loops).
- You can automatically generate these patterns using the Stateflow Pattern Wizard



Modeling State Machines

A state transition diagram contains:

- The possible states within the state machine
- The logic required to transition from one state to another
- The actions taken by the system for each particular state



E.g. Fault Management

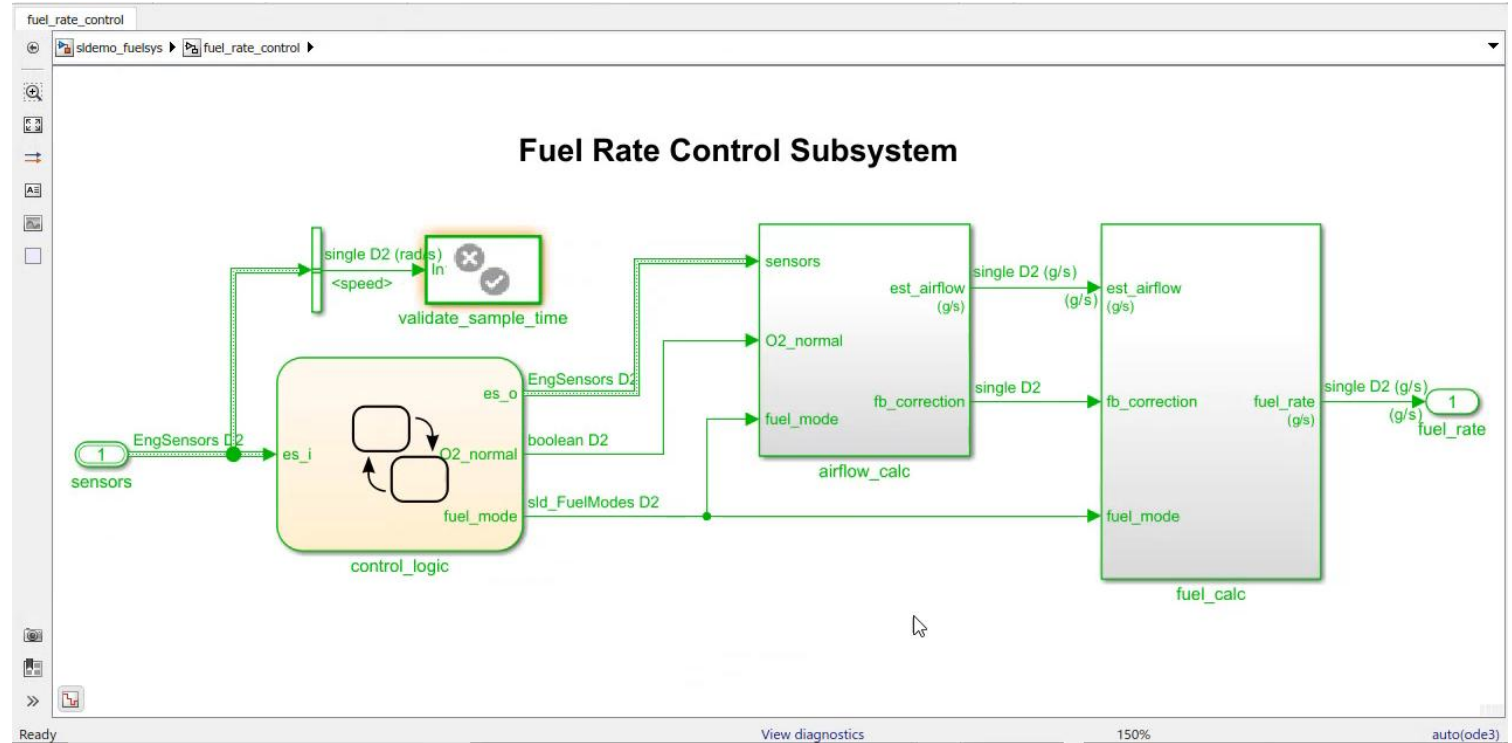


Automatically Generated Code Behaves The Same As The System Model

Integrate generated code

Reduce development time

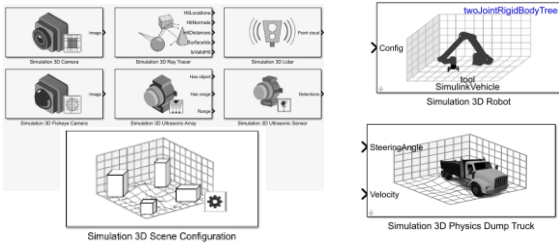
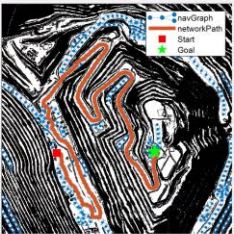
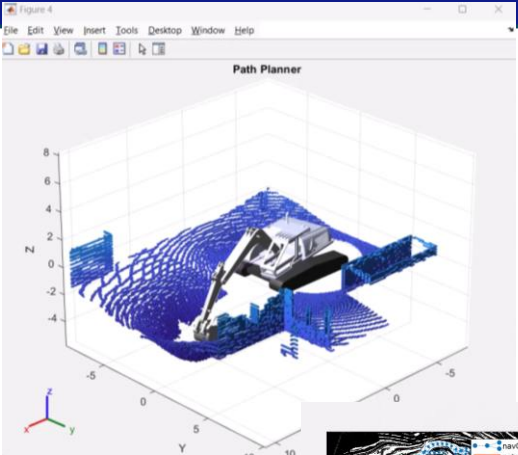
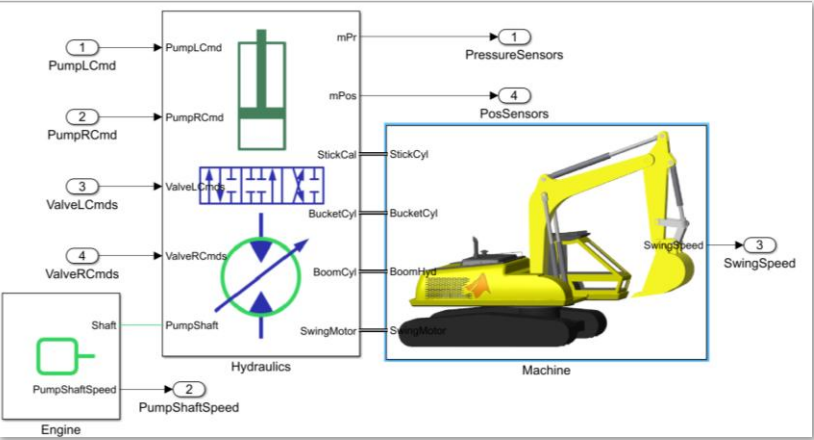
Eliminate hand-coding errors



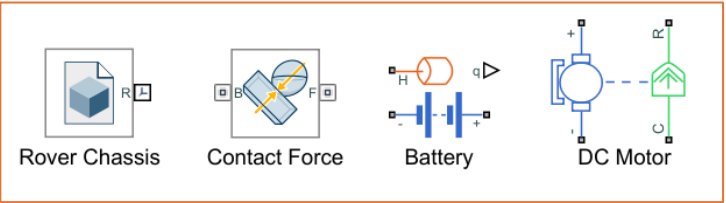
Physical Modeling of Offroad Vehicles



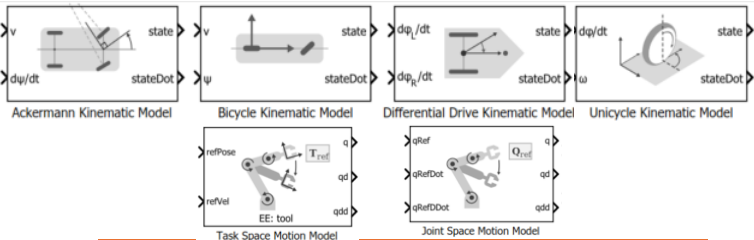
ISO 6015:2006
Earth-moving machinery —
Hydraulic excavators and
backhoe loaders — Methods
of determining tool forces



Scene, Scenario, and Agent
Models



Dynamic Models

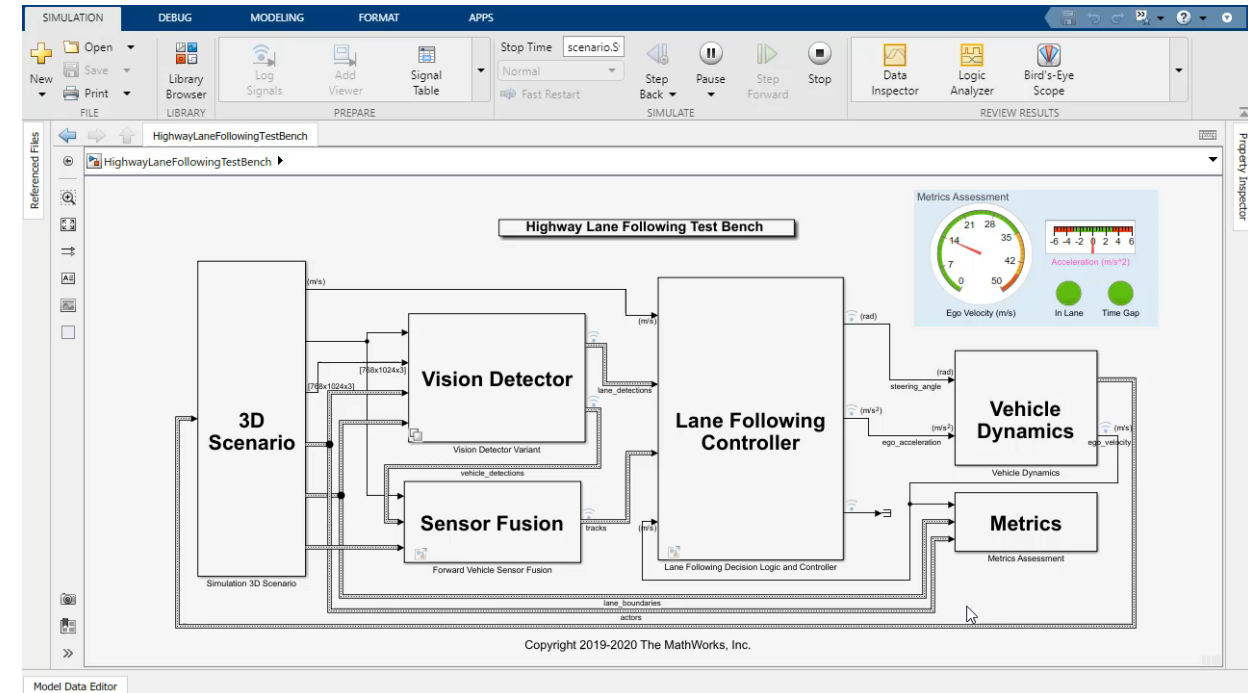
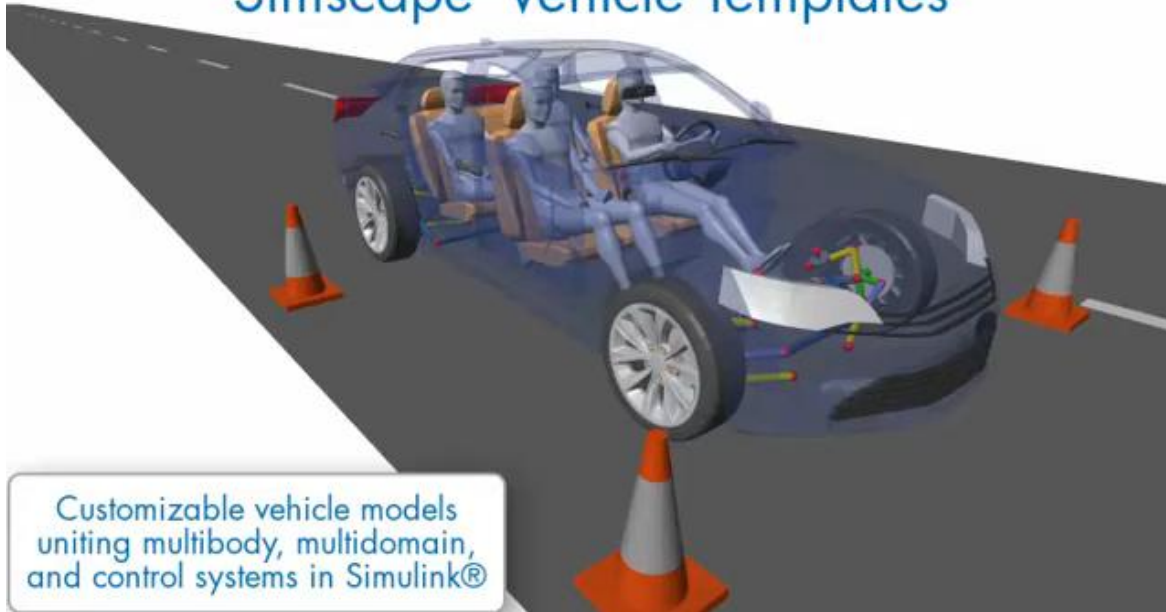


Kinematic Models



Simscape Applications

Simscape™ Vehicle Templates



<https://www.mathworks.com/matlabcentral/fileexchange/79484-simscape-vehicle-templates>



Aerospace Applications



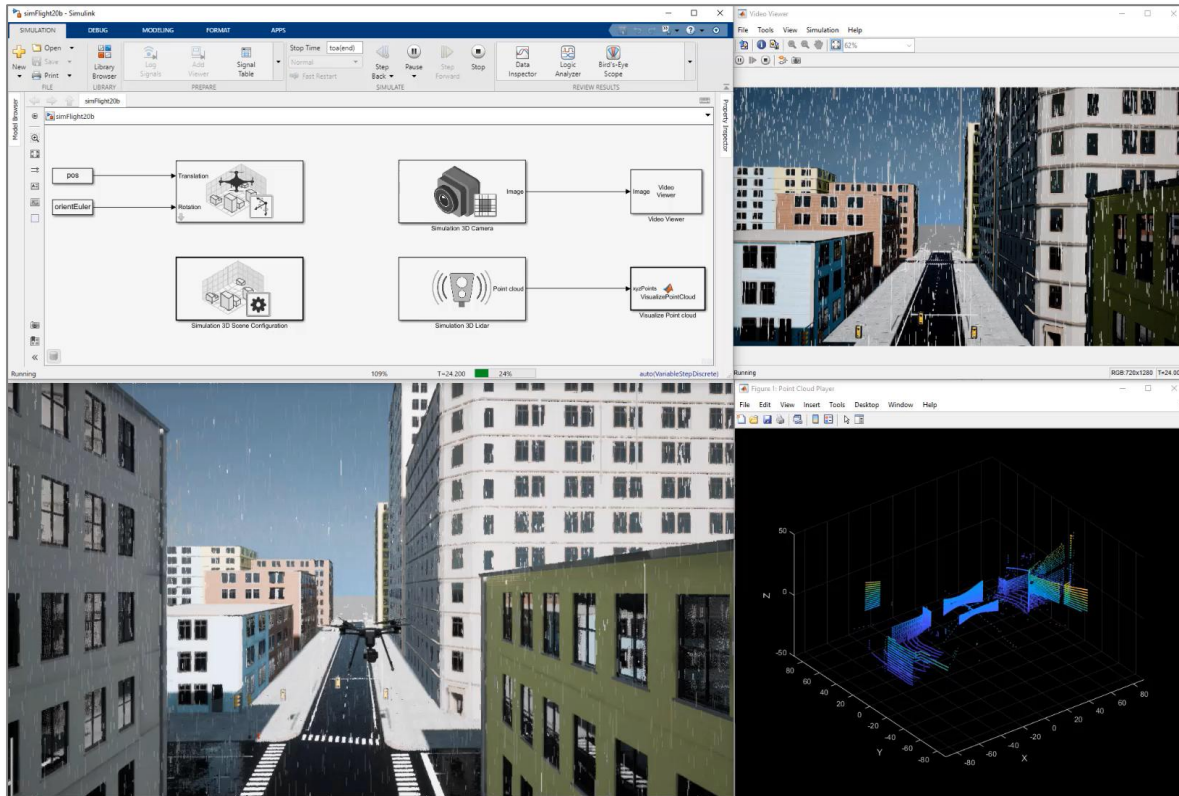
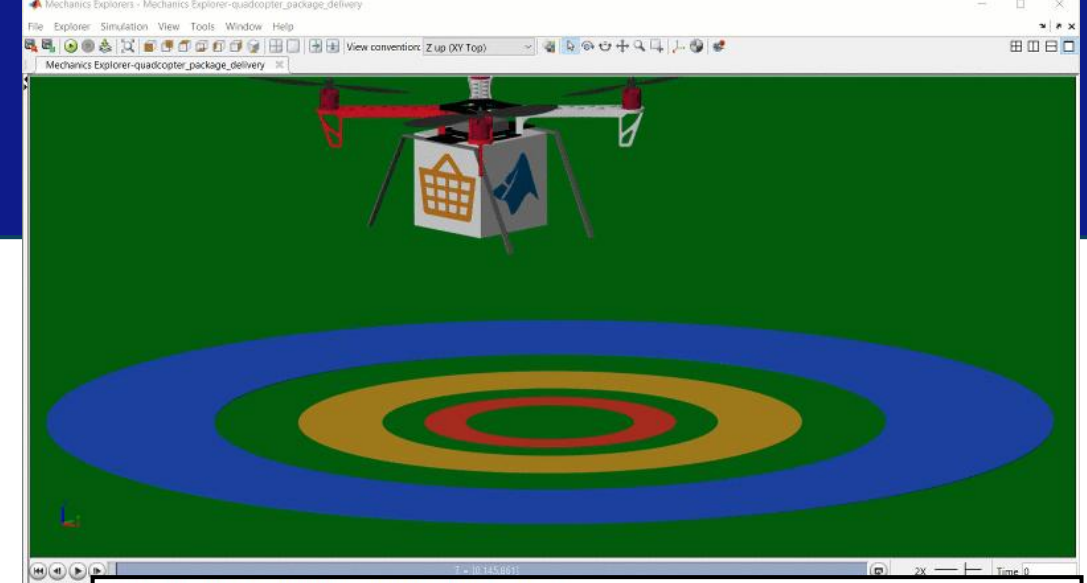
Fly the De Havilland Beaver with Unreal Engine®
Visualization

Cesium Ion®

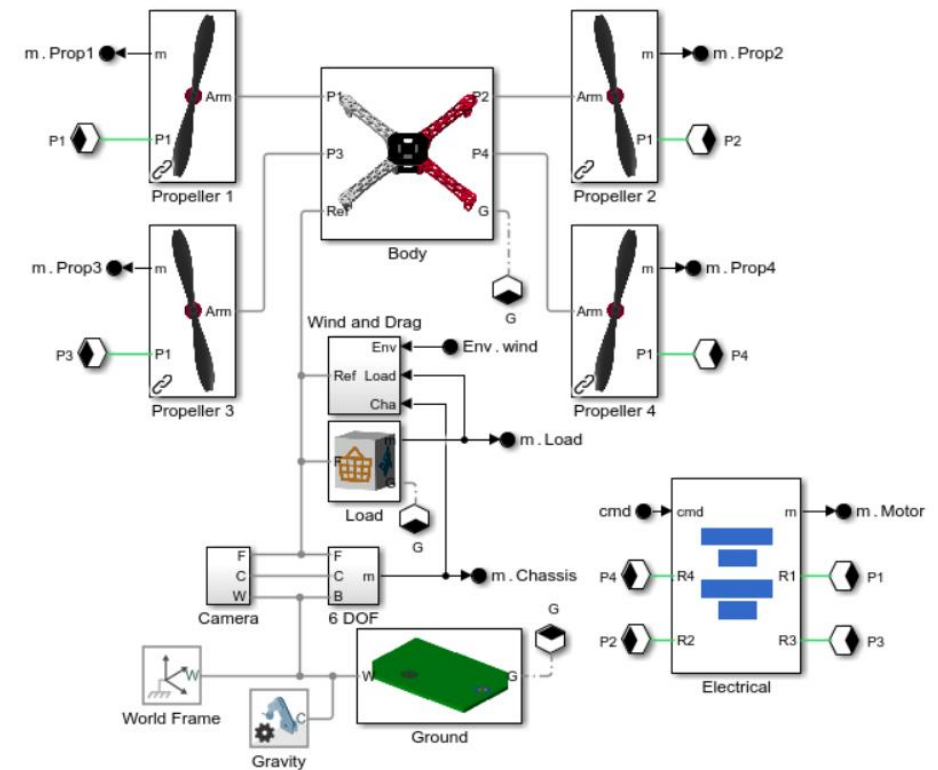


Rotorcraft
Unreal Engine Visualization Requires Simulink® 3D Animation™

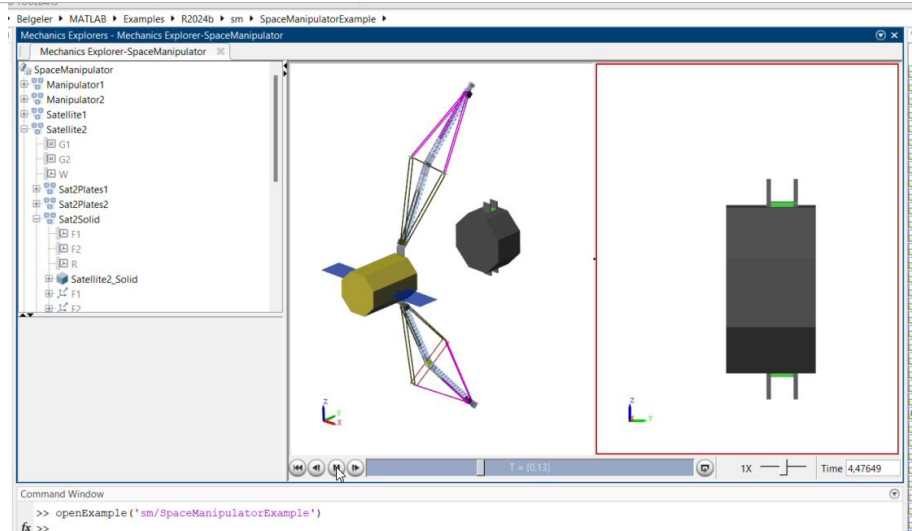
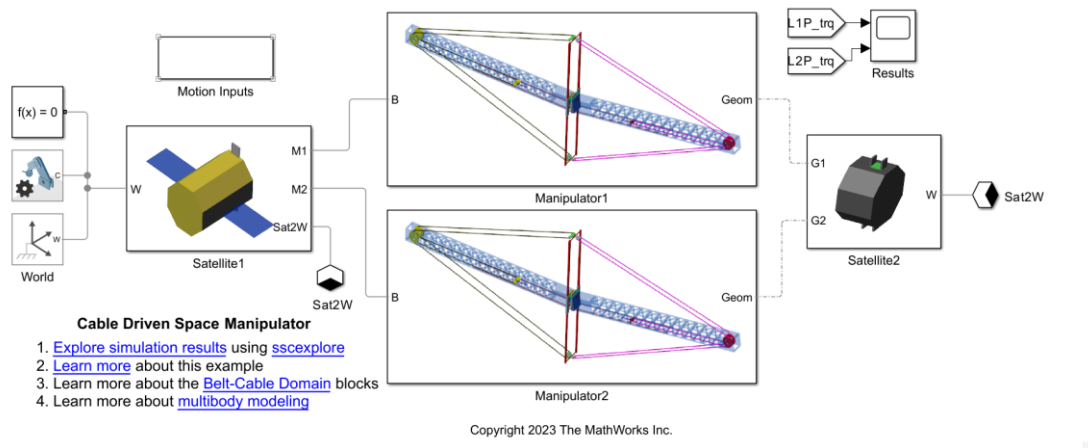
Aerospace Applications



Quadcopter Mechanical and Electrical Systems



Simscape Multibody Applications

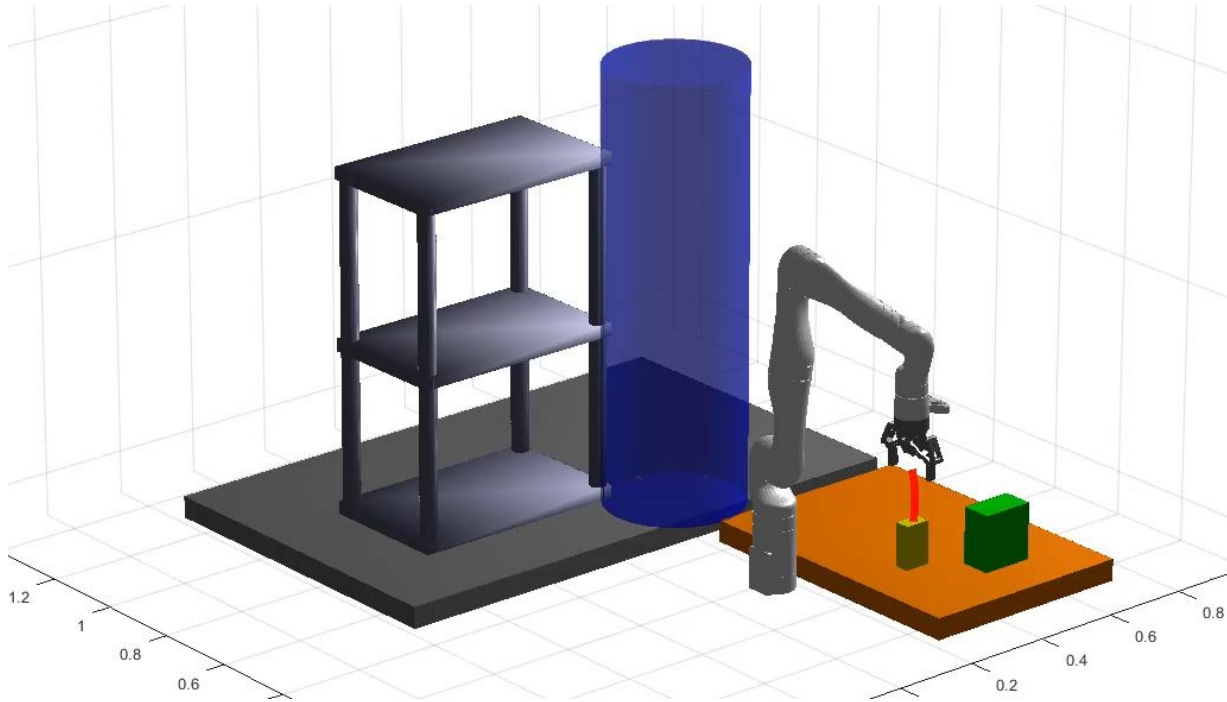


Cable Driven Space Manipulator



**Modeling and Control of a Mars Rover
in Simscape Multibody**

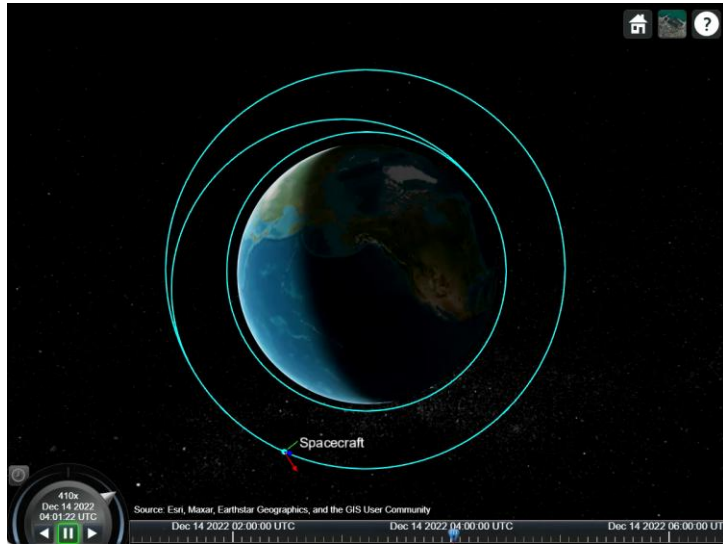
Robotic Applications



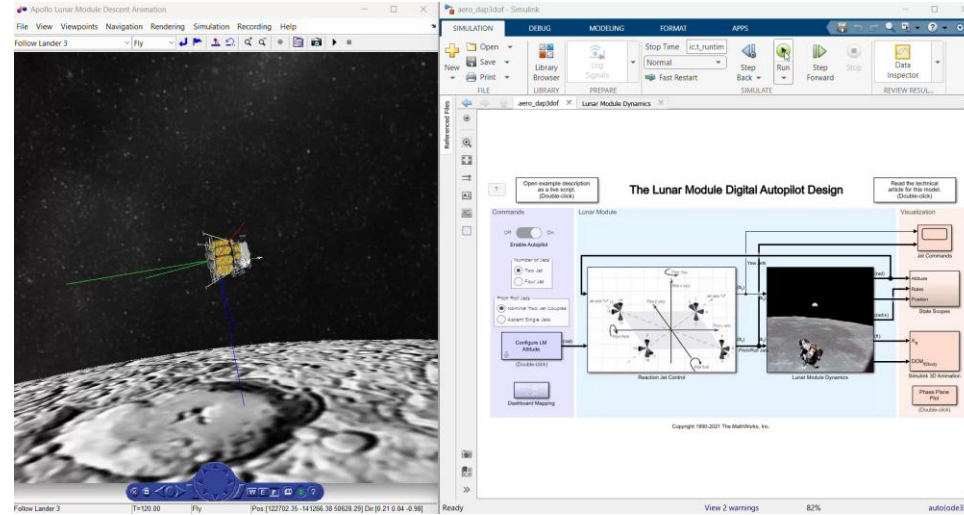
Robotic System Toolbox
Stateflow



Space Applications



Hohman Transfer Model



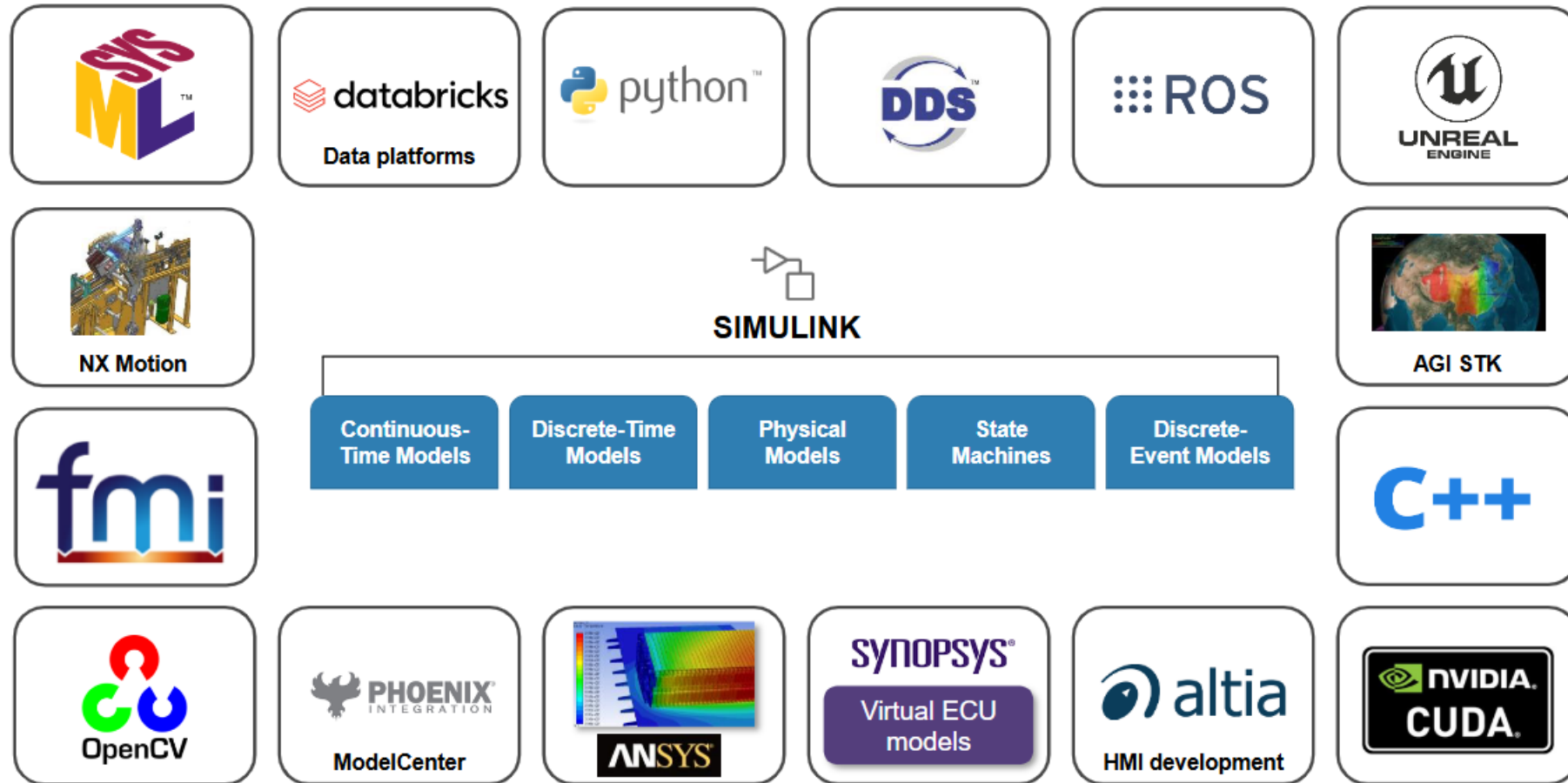
Apollo Lunar Module Digital Autopilot



**Satellite Conjunction Finder
Analysis Model**



Ecosystem with 100+ Third-Party Tools and Languages

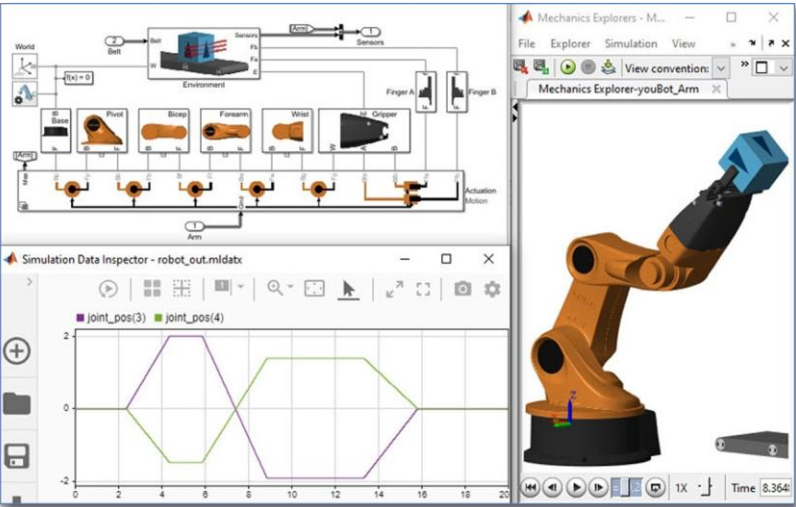


There are three key pieces to Model-Based Design

Modeling & Simulation

Code Generation

Test & Verification



```
604      /* End of Saturate: '<S210>/Saturation' */
605
606      /* RelationalOperator: '<S196>/NotEqual' */
607      NotEqual_n = (0.0F != Switch_f);
608
609      /* Signum: '<S196>/SignPreSat' */
610      if (Switch_f <= 0.0F) {
611          Switch_f = -1.0F;
612      } else {
613          if (Switch_f >= 0.0F) {
614              Switch_f = 1.0F;
615          }
616      }
```

Conditions analyzed

Description	True	False
Condition 1, "alt>10000"	4 U1.1	185 U1.1
Condition 2, "anomaly"	0 U1.1	4 U1.1

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition	True Out	False Out
Transition trigger expression		
Condition 1, "alt>10000"	TF U1.1	Fx U1.1

SIMULINK
Simulation and Model-Based Design



Simscape

Simscape Onramp

- Self-paced, interactive tutorial for getting started with Simscape

The screenshot shows the Simscape Onramp Simulink environment. The top menu bar includes SIMULATION, DEBUG, MODELING, FORMAT, APPS, and SIMSCAPE BLOCK. The left sidebar shows the Model Browser with the current task: 2.2 RC Circuit: (1/2) Free response. The main workspace displays a circuit diagram with a resistor labeled $R = 100 \Omega$ and a capacitor labeled $C = 1 \mu F$. A task box at the bottom left instructs the user to find the Resistor in the Electrical > Electrical Elements library and add it to the model. The bottom status bar shows 'Ready', '100%', and 'VariableStepAuto'.

<https://www.mathworks.com/services/training.html>

The screenshot shows the Simscape Onramp course overview window. The title bar says 'Simscape Onramp'. The main content area shows the course progress: 0% complete. The course is divided into nine sections, each with a progress indicator of 0%:

- 1. Course Overview 0%
- 2. Building Simscape Models 0%
 - Simscape Basics
 - RC Circuit (Free response)
 - RC Circuit (Driven response)
 - Rotational Mass Damper
- 3. Exploring Results 0%
- 4. Understanding Physical Signals 0%
- 5. Initial Values in Simscape 0%
- 6. Multidomain Modeling 0%
- 7. Simscape and Simulink 0%
- 8. Project - Electronic Valve 0%
- 9. Conclusion 0%

Resources and Training

Learn MATLAB for Free

Hands-on practice sessions and demonstrations

FREE



MATLAB Onramp

Get started quickly with the basics of MATLAB.

[Launch](#) [Details](#)

FREE



Simulink Onramp

Get started quickly with the basics of Simulink.

[Details and launch](#)

FREE



Machine Learning Onramp

Learn the basics of practical machine learning methods for classification problems.

[Launch](#) [Details](#)

FREE



Deep Learning Onramp

Get started quickly using deep learning methods to perform image recognition.

[Launch](#) [Details](#)

NEW



Reinforcement Learning Onramp

Master the basics of creating intelligent controllers that learn from experience.

[Launch](#) [Details](#)

NEW



Image Processing Onramp

Learn the basics of practical image processing techniques in MATLAB.

[Launch](#) [Details](#)

NEW



Signal Processing Onramp

An interactive introduction to signal processing methods for spectral analysis.

[Launch](#) [Details](#)

NEW



Simscape Onramp

Learn the basics of simulating physical systems in Simscape.

[Details and launch](#)

FREE



Stateflow Onramp

Learn the basics of creating, editing, and simulating state machines in Stateflow.

[Details and launch](#)

NEW



Control Design Onramp with Simulink

Get started quickly with the basics of feedback control design in Simulink.

[Details and launch](#)



Q & A

aylin.edgu@figes.com.tr



TEŞEKKÜRLER

