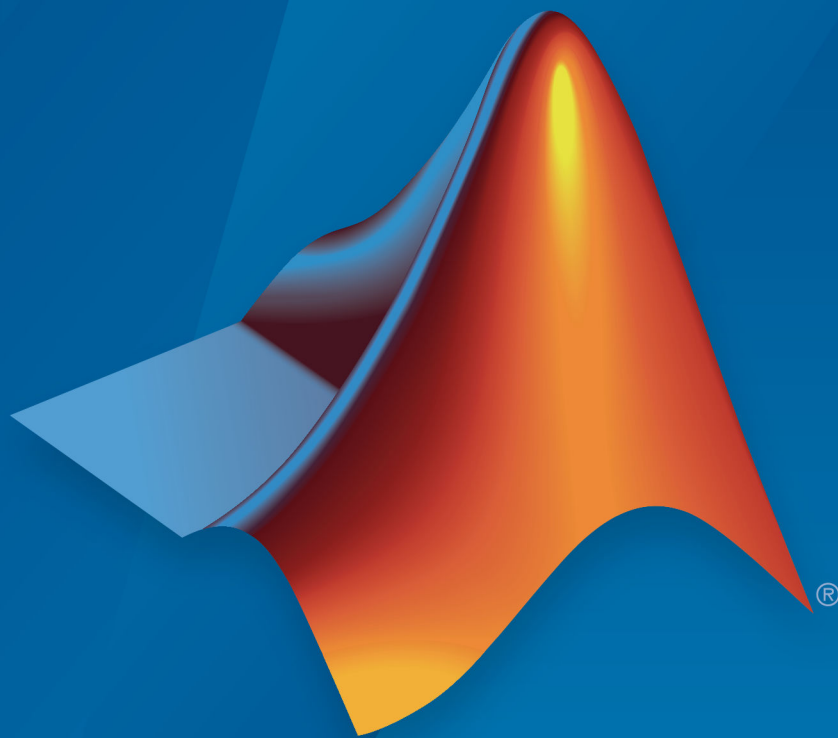


Simscape™ Electrical™

User's Guide



MATLAB® & SIMULINK®

R2019a



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Simscape™ Electrical™ User's Guide

© COPYRIGHT 2013–2019 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

September 2013	Online only	New for Version 6.0 (Release 2013b)
March 2014	Online only	Revised for Version 6.1 (Release 2014a) (Renamed from <i>SimPowerSystems™ User's Guide (Third Generation)</i>)
October 2014	Online only	Revised for Version 6.2 (Release 2014b)
March 2015	Online only	Revised for Version 6.3 (Release 2015a)
September 2015	Online only	Revised for Version 6.4 (Release 2015b)
March 2016	Online only	Revised for Version 6.5 (Release 2016a) (Renamed from <i>SimPowerSystems™ User's Guide (Simscape™ Components)</i>)
September 2016	Online only	Revised for Version 6.6 (Release 2016b)
March 2017	Online only	Revised for Version 6.7 (Release 2017a)
September 2017	Online only	Revised for Version 6.8 (Release 2017b)
March 2018	Online only	Revised for Version 6.9 (Release 2018a)
September 2018	Online only	Revised for Version 7.0 (Release 2018b) (Renamed from <i>Simscape™ Power Systems™ User's Guide (Simscape™ Components)</i> and <i>Simscape™ Electronics™ User's Guide</i>)
March 2019	Online only	Revised for Version 7.1 (Release 2019a) (Renamed from <i>Simscape™ Electrical™ User's Guide (Electronics, Mechatronics, and Power Systems)</i>)

1

Getting Started

Simscape Electrical Product Description	1-2
Key Features	1-2
Simscape Electrical Block Libraries	1-3
Libraries Compatible with Simscape Technology	1-3
Specialized Power Systems Library	1-4
Access the Simscape Electrical Block Libraries	1-5
Modeling Analog Circuit Architectures, Mechatronic Systems, and Electrical Power Systems Using Simscape Electrical ..	1-6
Simulink Templates for Modeling with Simscape Electrical ...	1-7
Simscape Electrical Blocks and Ports	1-7
Machine and Transformer Source Code Examples	1-8
Plotting and Display Options for Asynchronous and Synchronous Machines	1-8
Choosing the Right Simscape Electrical Technology	1-8
Assumptions and Limitations	1-8
Per-Unit System of Units	1-10
What Is the Per-Unit System?	1-10
Example 1: Three-Phase Transformer	1-12
Example 2: Asynchronous Machine	1-13
Base Values for Instantaneous Voltage and Current Waveforms	1-14
Why Use the Per-Unit System Instead of the Standard SI Units?	1-14

Build and Simulate Composite and Expanded Three-Phase Models	2-2
Select System Component Blocks and Build a Resistive Three-Phase Model	2-2
Specify Simulation Parameters	2-5
Load Impedance Parameters	2-6
Specify Display Parameters	2-6
Simulate and Analyze the Resistive Three-Phase Model	2-7
Simulate and Analyze a Reactive Three-Phase Model	2-7
Create an Expanded Balanced Three-Phase Model	2-8
Create an Expanded Unbalanced Three-Phase Model	2-10
Simulate the Expanded Balanced and Unbalanced Models and Analyze the Results	2-10
DC Motor Model	2-13
Select Blocks to Represent System Components	2-13
Build the Model	2-14
Specify Model Parameters	2-15
Configure the Solver Parameters	2-18
Run the Simulation and Analyze the Results	2-19
Triangle Wave Generator Model	2-21
Select Blocks to Represent System Components	2-21
Build the Model	2-22
Specify Model Parameters	2-24
Configure the Solver Parameters	2-27
Simulate Model and Analyze Results	2-28

Modeling and Simulation Basics

Essential Electrical Modeling Techniques	3-2
Overview of Modeling Rules	3-2
Required Blocks	3-3
Creating a New Model	3-4
Modeling Instantaneous Events	3-4
Using Simulink Blocks to Model Physical Components	3-5

Three-Phase Ports	3-7
About Three-Phase Ports	3-7
Expand and Collapse Three-Phase Ports on a Block	3-8
Switch Between Physical Signal and Electrical Ports	3-9
Simulating an Electronic, Mechatronic, or Electrical Power System	3-10
Selecting a Solver	3-10
Specifying Simulation Accuracy/Speed Tradeoff	3-10
Avoiding Simulation Issues	3-11
Running a Time-Domain Simulation	3-11
Running a Small-Signal Frequency-Domain Analysis	3-12
Selecting the Output Model for Logic Blocks	3-13
Available Output Models	3-13
Quadratic Model Output and Parameters	3-14
Parameterizing Blocks from Datasheets	3-17
Parameterize a Piecewise Linear Diode Model from a Datasheet	3-19
Parameterize an Exponential Diode from a Datasheet	3-23
Parameterize an Exponential Diode from SPICE Netlist	3-27
Parameterize an Op-Amp from a Datasheet	3-31
Additional Parameterization Workflows	3-33
Validation Using Data from SPICE Tool	3-33
Parameter Tuning Against External Data	3-33
Building an Equivalent Model of a SPICE Netlist	3-33
Simulating Thermal Effects in Rotational and Translational Actuators	3-35
Using the Thermal Ports	3-35
Thermal Model for Actuator Blocks	3-37
Simulating Thermal Effects in Semiconductors	3-38
Using the Thermal Ports	3-38
Thermal Model for Semiconductor Blocks	3-40
Thermal Mass Parameterization	3-41

Electrical Behavior Depending on Temperature	3-41
Improving Numerical Performance	3-42
Simulate Thermal Effects Using Simscape Electrical Thermal Blocks	3-43
Prerequisite	3-43
Thermal Variants	3-43
Thermal Blocks	3-43
Thermal Ports	3-44
Thermal-Modeling Parameters	3-45
Limitations	3-45
Model Thermal Losses for a Rectifier	3-46
Plot Basic Characteristics for Battery Blocks	3-54
Plot Basic Characteristics for Semiconductor Blocks	3-57
MOSFET Characteristics Viewer	3-60
Suggested Workflow	3-60
Add and Manage Characteristics	3-62
Choose Parameters and Generate Plots	3-65
Save the Results	3-67
Converting a SPICE Netlist to Simscape Blocks	3-69
Commands	3-69
Numeric Suffixes	3-70
Mathematical Functions	3-71
Symbols	3-73
Components	3-73
Performing Manual Conversions	3-76
Limitations	3-77

Modeling Machines

4

Machine Parameterization	4-2
Per-Unit Conversion for Machine Parameters	4-4
Impedance Conversion Equations	4-4
Magnetic Flux Linkage Conversion Equations	4-4

Machine Plotting and Display Options	4-6
Asynchronous Machine Options	4-6
Induction Machine Options	4-7
Machine Inertia Block Options	4-7
Initialize Synchronous Machines and Controllers	4-8

Customization

5

Build Custom Blocks Using the Three-Phase Electrical Domain	5-2
Custom Synchronous Machine	5-4

Control

6

Tune an Electric Drive	6-2
Cascade Control Structure	6-2
Equations for PI Tuning Using the Pole Placement Method ...	6-2
Equations for DC Motor Controller Tuning	6-5
Tune the Electric Drive in the Example Model	6-7

Simulation and Analysis of Power Engineering Systems

7

Optimize Block Settings for Simulating with the Partitioning Solver	7-2
Update Solver and Zero-Sequence Settings Using the ee_solverUpdate Function	7-3
Limitations of the ee_updateSolver Function	7-12

Phasor-Mode Simulation Using Simscape Components	7-13
Examine the Simulation Data Logging Configuration of a Model	7-19
Perform a Power-Loss Analysis	7-21
Prerequisite	7-21
Calculate Average Power Losses for the Simulation	7-21
Analyze Power Dissipation Differences Using Instantaneous Power Dissipation	7-23
Mitigate Transient Effects in Simulation Data	7-27
Choose a Simscape Electrical Function for an Offline Harmonic Analysis	7-30
Harmonic Distortion	7-30
Harmonic Analysis Functions	7-30
Evaluate Relative Overall Harmonic Distortion	7-31
Compare Harmonic Distortion to Standard Limits	7-32
Minimize Harmonic Distortion with Passive Filters	7-32
Verify the Results of an Online Harmonic Analysis	7-33
Perform an Online Harmonic Analysis Using the Simscape Spectrum Analyzer Block	7-34
Harmonic Distortion	7-34
Prerequisite	7-34
Perform an Offline Harmonic Analysis	7-35
Perform an Online Harmonic Analysis	7-38

Real-Time Simulation

8

Prepare Simscape Electrical Models for Real-Time Simulation Using Simscape Checks	8-2
--	------------

Generate HDL Code from Simscape Models	9-2
Deploy Simscape™ Plant Models to Speedgoat FPGA I/O Modules	9-13
Troubleshoot Conversion of Simscape DC Motor Control to HDL-Compatible Simulink Model	9-26
Troubleshoot Conversion of Simscape™ Permanent Magnet Synchronous Motor to HDL-Compatible Simulink Model .	9-36
Validate HDL Implementation Model to Simscape Algorithm	9-53
Bridge Rectifier Model	9-53
Increase Validation Logic Tolerance	9-55
Increase Number of Solver Iterations	9-56
Use Double-Precision Floating-Point	9-56
Improve Sampling Rate of HDL Implementation Model Generated from Simscape Algorithm	9-58
Sampling Frequency	9-58
Boost Converter Model	9-58
Reducing Number of Solver Iterations	9-60
Using Oversampling Factor and Latency Strategy	9-61

Getting Started

- “Simscape Electrical Product Description” on page 1-2
- “Simscape Electrical Block Libraries” on page 1-3
- “Modeling Analog Circuit Architectures, Mechatronic Systems, and Electrical Power Systems Using Simscape Electrical” on page 1-6
- “Per-Unit System of Units” on page 1-10

Simscape Electrical Product Description

Model and simulate electronic, mechatronic, and electrical power systems

Simscape Electrical (formerly SimPowerSystems™ and SimElectronics®) provides component libraries for modeling and simulating electronic, mechatronic, and electrical power systems. It includes models of semiconductors, motors, and components for applications such as electromechanical actuation, smart grids, and renewable energy systems. You can use these components to evaluate analog circuit architectures, develop mechatronic systems with electric drives, and analyze the generation, conversion, transmission, and consumption of electrical power at the grid level.

Simscape Electrical helps you develop control systems and test system-level performance. You can parameterize your models using MATLAB® variables and expressions, and design control systems for electrical systems in Simulink®. You can integrate mechanical, hydraulic, thermal, and other physical systems into your model using components from the Simscape family of products. To deploy models to other simulation environments, including hardware-in-the-loop (HIL) systems, Simscape Electrical supports C-code generation.

Simscape Electrical was developed in collaboration with Hydro-Québec of Montreal.

Key Features

- Libraries of electrical components including sensors, actuators, motors, machines, passive devices, and semiconductor devices
- Adjustable model fidelity, including nonlinear effects, operational limits, fault modeling, and temperature-dependent behavior
- SPICE netlist importer for converting SPICE subcircuits of discrete devices to Simscape models
- Application-specific models, including common AC and DC electric drives, smart grids, and renewable energy systems
- Ideal switching, discretization, and phasor simulation for faster model execution
- MATLAB based Simscape language for creating custom component models
- Support for C-code generation (with Simulink Coder™)

Simscape Electrical Block Libraries

Simscape Electrical software includes twelve different top-level libraries. These libraries allow you to model mechatronic systems, analog circuit architectures, and single- and multi-phase electrical power systems. You can also develop control algorithms for these systems within the Simulink environment by using these libraries. All of these libraries, except Specialized Power Systems, contain blocks developed specifically for extending the Simscape Foundation domains and are fully compatible with the Simscape technology. Blocks in the Specialized Power Systems library function in their own domain.

Libraries Compatible with Simscape Technology

All Simscape Electrical libraries, except Specialized Power Systems, contain blocks specifically developed to:

- Extend the Simscape Electrical domain, a single-phase electrical domain.
- Extend the Simscape Three-Phase Electrical domain, a three-phase electrical domain.

These library blocks are written in the Simscape language and are fully compatible with the Simscape technology, including local solvers, data logging, statistics and variable viewers, frequency analysis, and component and library customizations. To configure Simscape Electrical models composed of these library blocks for local-solver simulation, use the Solver Configuration block. Many of the blocks in these libraries also work with other Simscape Foundation domains, such as the Mechanical, Magnetic, and Thermal domains. When working with the Simscape technology compatible library blocks, you can use these capabilities:

- Partitioning Solver
- Simscape HDL Workflow Advisor
- Simscape Results Explorer

These libraries include models of high-fidelity, nonlinear, faultable, electrothermal power electronics. You can use these components to develop mechatronic systems and to build behavioral models for evaluating analog circuit architectures. The libraries also include low-fidelity models that are switched linear and optimized for fast simulation. There are also some models that contain optional ports for thermal analysis.

You can create single-line three-phase diagrams by using the three-phase blocks because the Three-Phase Electrical domain supports signals that contain all three phases as

individual elements in a single vector. You can also model each phase individually, for example, to inject a single-line-to-ground fault into your circuit, by expanding the three-phase ports on these blocks into three separate single-phase electrical ports.

The Control library contains Simulink blocks for signal generation, mathematical transformation, and machine control. You can use these components to develop control systems for single- and multi-phase electrical power systems.

Through conserving ports of the same domain, you can directly connect the blocks from these Simscape Electrical libraries to Simscape blocks from:

- Simscape Foundation libraries
- Simscape add-on products, such as Simscape Driveline™, Simscape Multibody™, and Simscape Fluids™

Through physical signal ports, you can connect the physical blocks from these Simscape Electrical libraries to:

- Simulink blocks, including blocks from the Control library, by using converter blocks from the Simscape Utilities library
- Blocks from the Physical Signals library, which is in the Simscape Foundation library.

Specialized Power Systems Library

The Simscape Electrical Specialized Power Systems library contains blocks that use their own, *specialized* electrical domain. The library contains models of typical power equipment such as transformers, electric machines and drives, and power electronics. It also contains control, measurement, and signal generation models that you can use for developing power system control algorithms. The Specialized Power Systems Fundamental Blocks library contains the powergui block, which provides tools for the steady-state analysis of electrical circuits. To configure Specialized Power Systems models for continuous-time, discrete-time, or phasor simulation, and to analyze simulation results, use the powergui block. The powergui block is in the Specialized Power Systems Fundamental Blocks library.

You can connect Specialized Power Systems blocks to Simulink blocks either:

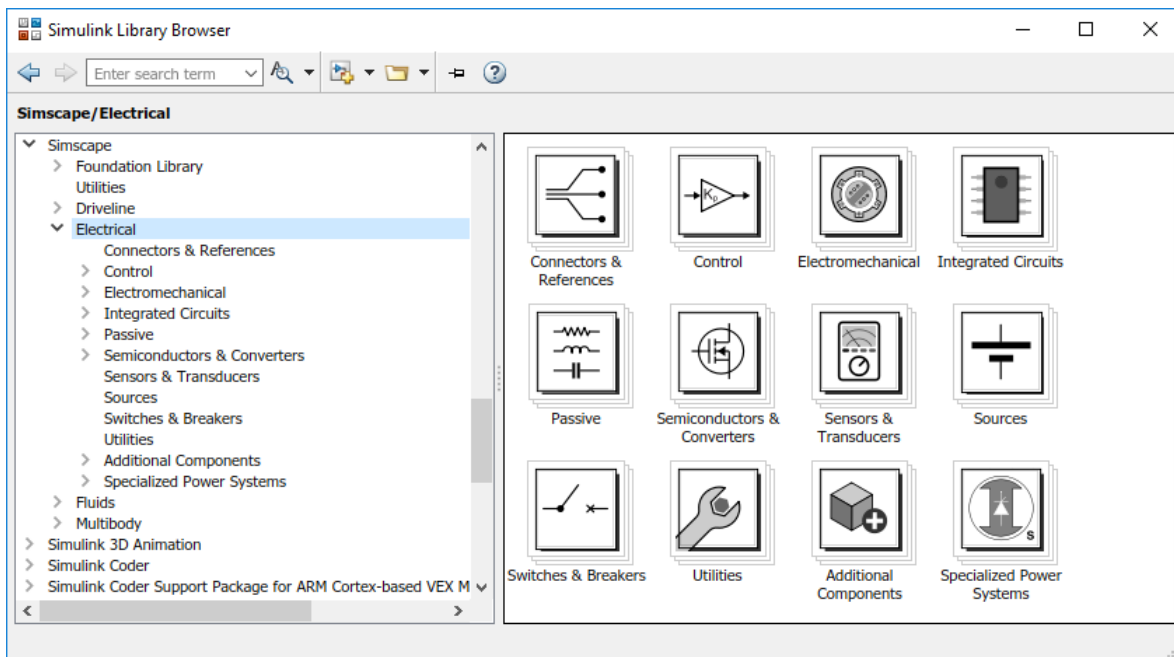
- Directly, through Simulink signal input and output ports.
- Through measurement blocks from the Measurements sublibrary of the Fundamental Blocks library.

Access the Simscape Electrical Block Libraries

You can access the Simscape Electrical libraries from the Simulink Library Browser or from the MATLAB command prompt.

To display the **Electrical** library in the Simulink Library Browser, scroll to the **Simscape** node. Expand the **Simscape** node and then the **Electrical** node. Alternately, at the MATLAB command prompt, enter this command.

```
ee_lib
```



To access the sublibraries in the twelve top-level Simscape Electrical libraries, further expand the nodes. Alternately, use the `open_system` command at the MATLAB command prompt. For example, to access the sublibraries in the **Connectors & References** library, enter the commands:

```
ee_lib;
open_system('ee_lib/Connectors & References')
```

Modeling Analog Circuit Architectures, Mechatronic Systems, and Electrical Power Systems Using Simscape Electrical

In this section...

“Simulink Templates for Modeling with Simscape Electrical” on page 1-7

“Simscape Electrical Blocks and Ports” on page 1-7

“Machine and Transformer Source Code Examples” on page 1-8

“Plotting and Display Options for Asynchronous and Synchronous Machines” on page 1-8

“Choosing the Right Simscape Electrical Technology” on page 1-8

“Assumptions and Limitations” on page 1-8

When you model and analyze mechatronic systems, analog circuit architectures, or electrical power systems using Simscape Electrical, your workflow might include the following tasks:

- 1** Create a Simulink model that includes components from Simscape Electrical libraries.

In most applications, it is most natural to model the physical system using Simscape Electrical blocks and other Simscape blocks, and then develop the controller or signal processing algorithm in Simulink.

For more information about modeling the physical system, see “Essential Electrical Modeling Techniques” on page 3-2.

- 2** Define component data by specifying electrical or mechanical properties as defined on a datasheet.

For more information about parametrization, see “Parameterizing Blocks from Datasheets” on page 3-17.

- 3** Configure the solver options.

For more information about the settings that most affect the solution of a physical system, see “Setting Up Solvers for Physical Models” (Simscape).

- 4** Run the simulation.

For more information on how to perform time-domain simulation of an electrical system, see “Simulating an Electronic, Mechatronic, or Electrical Power System” on page 3-10.

Simulink Templates for Modeling with Simscape Electrical

On the Simulink start page, the **Simscape** section has model templates that provide you with design patterns for modeling with Simscape Electrical:

- Electrical
- Electrical Three-Phase
- Mechanical Rotational
- Mechanical Translational

Models you create from these templates have the corresponding reference block, the required Solver Configuration block, and the frequently used Simscape-Simulink interfacing blocks already in the Simulink canvas. The models also contain links that you can double-click to access other blocks in the corresponding Simscape libraries.

To create a model using one of these **Simscape** templates:

- 1 Open the Simulink Start page. In the MATLAB Home tab, select the **Simulink** button. Alternatively, at the command line, enter:

```
simulink
```

- 2 In the **Simscape** section, locate the templates that are preconfigured for modeling with Simscape Electrical. Selecting a template opens a model in the Simulink Editor. To save the model, select **File > Save As**.

Simscape Electrical Blocks and Ports

Simscape Electrical blocks that are written in the Simscape language are fully compatible with Simscape technology, including the local solver, code generation, and data logging.

Simscape Electrical blocks have single-phase, composite three-phase, thermal, magnetic, mechanical translational conserving, and mechanical rotational conserving ports. You can use composite three-phase ports to build models corresponding to single-line diagrams of three-phase electrical systems. Composite three-phase ports connect to other composite three-phase ports. Electrical and mechanical rotational conserving ports connect directly

to Simscape Foundation library components and Simscape add-on products such as Simscape Driveline. You can use a Phase Splitter block to split a composite three-phase port into individual electrical conserving ports.

Machine and Transformer Source Code Examples

Simscape Electrical software provides Simscape language source code examples for machines and transformers, which you can view and customize. To access the example blocks, type `ThreePhaseExamples_lib` at the MATLAB command prompt.

Plotting and Display Options for Asynchronous and Synchronous Machines

For the Machine Inertia block and the asynchronous and synchronous machine blocks in Simscape Electrical software, you can perform some useful plotting and display actions using the **Electrical** menu on the block context menu. For example, to plot torque versus speed (both in SI units) for the Induction Machine Wound Rotor block, right-click the block. From the block context menu, select **Electrical > Plot Torque Speed (SI)**. The software plots the results in a figure window.

Using other options on the **Electrical** menu, you can plot values in per-unit or display base parameter values in the MATLAB Command Window. These options enable you to tune the performance of your three-phase machine quickly.

Choosing the Right Simscape Electrical Technology

Simscape Electrical software includes two different technologies and corresponding libraries. For a comparison of the two technologies, see “Simscape Electrical Block Libraries” on page 1-3. Choose the Simscape Electrical technology most appropriate for your modeling needs and, if possible, build your model using blocks exclusively from that technology. However, if necessary, you can build a model that uses blocks from both technologies. To do so, use blocks from the **Simscape > Electrical > Specialized Power Systems > Fundamental Blocks > Interface Elements** library to interface between them.

Assumptions and Limitations

The Simscape Electrical blocks let you perform tradeoff analyses to optimize system design, for example, by testing various algorithms with different circuit implementations.

The library contains blocks that use either high level or more detailed models to simulate components. Simscape Electrical does not have the capability to:

- Perform either layout (physical design) tasks, or the associated implementation tasks such as layout versus schematic (LVS), design rule checking (DRC), parasitic extraction, and back annotation.
- Model 3-D parasitic effects that are typically important for high-frequency applications.

For these types of requirements, you must use an EDA package specifically designed for the implementation of analog circuits.

See Also

Phase Splitter

More About

- “Simscape Electrical Block Libraries” on page 1-3
- “Essential Electrical Modeling Techniques” on page 3-2
- “Parameterizing Blocks from Datasheets” on page 3-17
- “Setting Up Solvers for Physical Models” (Simscape)
- “Simulating an Electronic, Mechatronic, or Electrical Power System” on page 3-10

Per-Unit System of Units

In this section...
“What Is the Per-Unit System?” on page 1-10
“Example 1: Three-Phase Transformer” on page 1-12
“Example 2: Asynchronous Machine” on page 1-13
“Base Values for Instantaneous Voltage and Current Waveforms” on page 1-14
“Why Use the Per-Unit System Instead of the Standard SI Units?” on page 1-14

What Is the Per-Unit System?

The per-unit system is widely used in the power system industry to express values of voltages, currents, powers, and impedances of various power equipment. It is typically used for transformers and AC machines.

For a given quantity (voltage, current, power, impedance, torque, etc.) the per-unit value is the value related to a base quantity.

$$\text{base value in p.u.} = \frac{\text{quantity expressed in SI units}}{\text{base value}}$$

Generally the following two base values are chosen:

- The base power = nominal power of the equipment
- The base voltage = nominal voltage of the equipment

All other base quantities are derived from these two base quantities. Once the base power and the base voltage are chosen, the base current and the base impedance are determined by the natural laws of electrical circuits.

$$\text{base current} = \frac{\text{base power}}{\text{base voltage}}$$

$$\text{base impedance} = \frac{\text{base voltage}}{\text{base current}} = \frac{(\text{base voltage})^2}{\text{base power}}$$

For a transformer with multiple windings, each having a different nominal voltage, the same base power is used for all windings (nominal power of the transformer). However,

according to the definitions, there are as many base values as windings for voltages, currents, and impedances.

The saturation characteristic of saturable transformer is given in the form of an instantaneous current versus instantaneous flux-linkage curve: [$i_1 \phi_1$; $i_2 \phi_2$; ..., in ϕ_{in}].

When the per-unit system is used to specify the transformer R L parameters, the flux linkage and current in the saturation characteristic must be also specified in pu. The corresponding base values are

$$\begin{aligned} \text{base instantaneous current} &= (\text{base rms current}) \times \sqrt{2} \\ \text{base flux linkage} &= \frac{(\text{base rms voltage}) \times \sqrt{2}}{2\pi \times (\text{base frequency})} \end{aligned}$$

where current, voltage, and flux linkage are expressed respectively in volts, amperes, and volt-seconds.

For AC machines, the torque and speed can be also expressed in pu. The following base quantities are chosen:

- The base speed = synchronous speed
- The base torque = torque corresponding at base power and synchronous speed

$$\text{base torque} = \frac{\text{base power (3 phases) in VA}}{\text{base speed in radians/second}}$$

Instead of specifying the rotor inertia in $\text{kg}\cdot\text{m}^2$, you would generally give the inertia constant H defined as

$$\begin{aligned} H &= \frac{\text{kinetic energy stored in the rotor at synchronous speed in joules}}{\text{machine nominal power in VA}} \\ H &= \frac{\frac{1}{2} \times J \cdot \omega^2}{P_{nom}} \end{aligned}$$

The inertia constant is expressed in seconds. For large machines, this constant is around 3-5 seconds. An inertia constant of 3 seconds means that the energy stored in the rotating part could supply the nominal load during 3 seconds. For small machines, H is lower. For example, for a 3-HP motor, it can be 0.5-0.7 seconds.

Example 1: Three-Phase Transformer

Consider, for example, a three-phase two-winding transformer with these manufacturer-provided, typical parameters:

- Nominal power = 300 kVA total for three phases
- Nominal frequency = 60 Hz
- Winding 1: connected in wye, nominal voltage = 25-kV RMS line-to-line
resistance 0.01 pu, leakage reactance = 0.02 pu
- Winding 2: connected in delta, nominal voltage = 600-V RMS line-to-line
resistance 0.01 pu, leakage reactance = 0.02 pu
- Magnetizing losses at nominal voltage in % of nominal current:
Resistive 1%, Inductive 1%

The base values for each single-phase transformer are first calculated:

- For winding 1:

Base power	$300 \text{ kVA}/3 = 100\text{e}3 \text{ VA/phase}$
Base voltage	$25 \text{ kV}/\sqrt{3} = 14434 \text{ V RMS}$
Base current	$100\text{e}3/14434 = 6.928 \text{ A RMS}$
Base impedance	$14434/6.928 = 2083 \Omega$
Base resistance	$14434/6.928 = 2083 \Omega$
Base inductance	$2083/(2\pi*60) = 5.525 \text{ H}$

- For winding 2:

Base power	$300 \text{ kVA}/3 = 100\text{e}3 \text{ VA}$
Base voltage	600 V RMS
Base current	$100\text{e}3/600 = 166.7 \text{ A RMS}$
Base impedance	$600/166.7 = 3.60 \Omega$
Base resistance	$600/166.7 = 3.60 \Omega$
Base inductance	$3.60/(2\pi*60) = 0.009549 \text{ H}$

The values of the winding resistances and leakage inductances expressed in SI units are therefore

- For winding 1: $R_1 = 0.01 * 2083 = 20.83 \Omega$; $L_1 = 0.02 * 5.525 = 0.1105 \text{ H}$
- For winding 2: $R_2 = 0.01 * 3.60 = 0.0360 \Omega$; $L_2 = 0.02 * 0.009549 = 0.191 \text{ mH}$

For the magnetizing branch, magnetizing losses of 1% resistive and 1% inductive mean a magnetizing resistance R_m of 100 pu and a magnetizing inductance L_m of 100 pu. Therefore, the values expressed in SI units referred to winding 1 are

- $R_m = 100 * 2083 = 208.3 \text{ k}\Omega$
- $L_m = 100 * 5.525 = 552.5 \text{ H}$

Example 2: Asynchronous Machine

Now consider a three-phase, four-pole Asynchronous Machine block in SI units. It is rated 3 HP, 220 V RMS line-to-line, 60 Hz.

The stator and rotor resistance and inductance referred to stator are

- $R_s = 0.435 \Omega$; $L_s = 2 \text{ mH}$
- $R_r = 0.816 \Omega$; $L_r = 2 \text{ mH}$

The mutual inductance is $L_m = 69.31 \text{ mH}$. The rotor inertia is $J = 0.089 \text{ kg}\cdot\text{m}^2$.

The base quantities for one phase are calculated as follows:

Base power	$3 \text{ HP} * 746 \text{ VA} / 3 = 746 \text{ VA/phase}$
Base voltage	$220 \text{ V} / \sqrt{3} = 127.0 \text{ V RMS}$
Base current	$746 / 127.0 = 5.874 \text{ A RMS}$
Base impedance	$127.0 / 5.874 = 21.62 \Omega$
Base resistance	$127.0 / 5.874 = 21.62 \Omega$
Base inductance	$21.62 / (2\pi * 60) = 0.05735 \text{ H} = 57.35 \text{ mH}$
Base speed	$1800 \text{ rpm} = 1800 * (2\pi) / 60 = 188.5 \text{ radians/second}$
Base torque (three-phase)	$746 * 3 / 188.5 = 11.87 \text{ newton-meters}$

Using the base values, you can compute the values in per-units.

$$\begin{aligned} R_s &= 0.435 / 21.62 = 0.0201 \text{ pu} & L_s &= 2 / 57.35 = 0.0349 \text{ pu} \\ R_r &= 0.816 / 21.62 = 0.0377 \text{ pu} & L_r &= 2 / 57.35 = 0.0349 \text{ pu} \\ L_m &= 69.31/57.35 = 1.208 \text{ pu} \end{aligned}$$

The inertia is calculated from inertia J , synchronous speed, and nominal power.

$$H = \frac{\frac{1}{2} \times J \cdot \omega^2}{P_{nom}} = \frac{\frac{1}{2} \times 0.089 \times (188.5)^2}{3 \times 746} = 0.7065 \text{ seconds}$$

If you open the dialog box of the Asynchronous Machine block in pu units provided in the Machines library of the Simscape Electrical Specialized Power Systems Fundamental Blocks library, you find that the parameters in pu are the ones calculated.

Base Values for Instantaneous Voltage and Current Waveforms

When displaying instantaneous voltage and current waveforms on graphs or oscilloscopes, you normally consider the peak value of the nominal sinusoidal voltage as 1 pu. In other words, the base values used for voltage and currents are the RMS values given multiplied by $\sqrt{2}$.

Why Use the Per-Unit System Instead of the Standard SI Units?

Here are the main reasons for using the per-unit system:

- When values are expressed in pu, the comparison of electrical quantities with their "normal" values is straightforward.

For example, a transient voltage reaching a maximum of 1.42 pu indicates immediately that this voltage exceeds the nominal value by 42%.

- The values of impedances expressed in pu stay fairly constant whatever the power and voltage ratings.

For example, for all transformers in the 3–300 kVA power range, the leakage reactance varies approximately 0.01–0.03 pu, whereas the winding resistances vary between 0.01 pu and 0.005 pu, whatever the nominal voltage. For transformers in the

300 kVA to 300 MVA range, the leakage reactance varies approximately 0.03-0.12 pu, whereas the winding resistances vary between 0.005-0.002 pu.

Similarly, for salient pole synchronous machines, the synchronous reactance X_d is generally 0.60-1.50 pu, whereas the subtransient reactance X'_d is generally 0.20-0.50 pu.

It means that if you do not know the parameters for a 10-kVA transformer, you are not making a major error by assuming an average value of 0.02 pu for leakage reactances and 0.0075 pu for winding resistances.

The calculations using the per-unit system are simplified. When all impedances in a multivoltage power system are expressed on a common power base and on the nominal voltages of the different subnetworks, the total impedance in pu seen at one bus is obtained by simply adding all impedances in pu, without considering the transformer ratios.

Tutorials

- “Build and Simulate Composite and Expanded Three-Phase Models” on page 2-2
- “DC Motor Model” on page 2-13
- “Triangle Wave Generator Model” on page 2-21

Build and Simulate Composite and Expanded Three-Phase Models

In this section...

“Select System Component Blocks and Build a Resistive Three-Phase Model” on page 2-2

“Specify Simulation Parameters” on page 2-5

“Load Impedance Parameters” on page 2-6

“Specify Display Parameters” on page 2-6

“Simulate and Analyze the Resistive Three-Phase Model” on page 2-7

“Simulate and Analyze a Reactive Three-Phase Model” on page 2-7

“Create an Expanded Balanced Three-Phase Model” on page 2-8

“Create an Expanded Unbalanced Three-Phase Model” on page 2-10

“Simulate the Expanded Balanced and Unbalanced Models and Analyze the Results” on page 2-10

In this example, you build and analyze a simple Simscape Electrical model that simulates the behavior of a three-phase AC voltage source driving a purely resistive three-phase load. You then modify the load in this model to change it to:

- A reactive three-phase load
- A resistive three-phase load expanded into individual phases
- An expanded three-phase load that does not have equal resistance in each phase

For the completed initial model, see Simple Three-Phase Model.

Select System Component Blocks and Build a Resistive Three-Phase Model

- 1 Open the Simulink Start page. In the MATLAB Home tab, select the **Simulink** button. Alternatively, at the command line, enter:

```
simulink
```

- 2 In the **Simscape** section, find the templates that are preconfigured for modeling with Simscape Electrical. Select the Electrical Three-Phase template. A model that contains these blocks opens in the Simulink canvas.

Block	Purpose	Library
Scope	Display phase voltages and currents for the three-phase system.	Simulink > Sinks
Electrical Reference	Provide the ground connection for electrical conserving ports.	Simscape > Foundation Library > Electrical > Electrical Elements
PS-Simulink Converter	Convert the physical signals to Simulink signals.	Simscape > Utilities
Simulink-PS Converter	Convert Simulink signals to physical signals.	Simscape > Utilities
Solver Configuration	Define solver settings that apply to all physical modeling blocks.	Simscape > Utilities
Grounded Neutral (Three-Phase)	Provide an electrical ground connection for each phase of the three-phase system.	Simscape > Electrical > Connectors & References
Line Voltage Sensor (Three-Phase)	Measure the line-line voltages of a three-phase system and output a three-element physical signal vector.	Simscape > Electrical > Sensors & Transducers

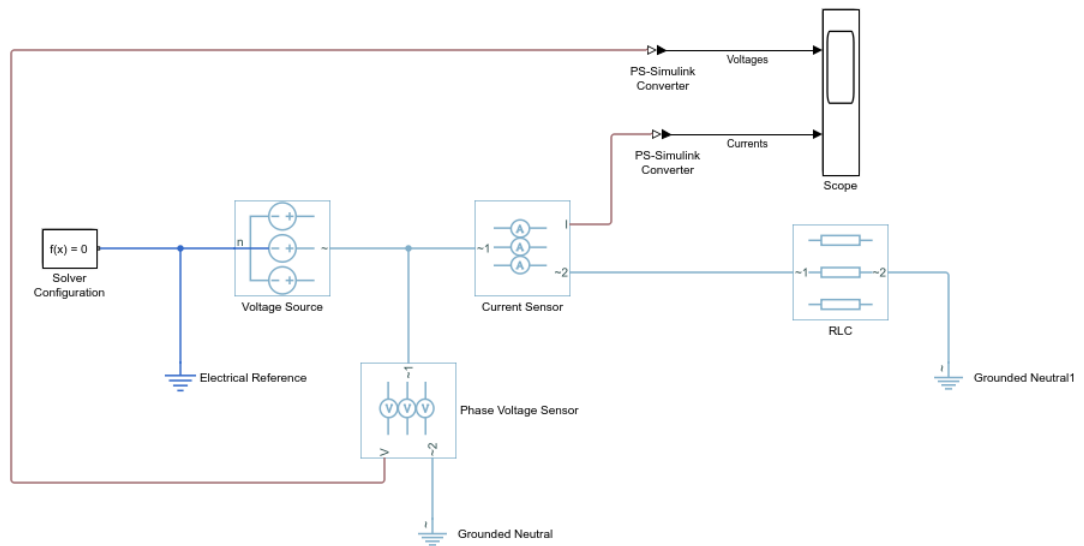
The model also contains two links that you can double-click to access blocks from Simscape and Simscape Electrical libraries. For more information on using templates for modeling with Simscape Electrical, see “Modeling Analog Circuit Architectures, Mechatronic Systems, and Electrical Power Systems Using Simscape Electrical” on page 1-6.

- 3 Delete the Simulink-PS Converter and Line Voltage Sensor (Three-Phase) blocks.
- 4 Add these blocks to the model.

Block	Purpose	Library
RLC (Three-Phase)	Model the resistive, inductive, and capacitive properties of the three-phase load.	Simscape > Electrical > Passive > RLC Assemblies

Block	Purpose	Library
Current Sensor (Three-Phase)	Convert the electrical current flowing in each phase of the three-phase load into a physical signal proportional to that current.	Simscape > Electrical > Sensors & Transducers
Phase Voltage Sensor (Three-Phase)	Convert the voltage across each phase of the three-phase system into a physical signal proportional to that voltage.	Simscape > Electrical > Sensors & Transducers
Voltage Source (Three-Phase)	Provide an ideal three-phase voltage source that maintains a sinusoidal voltage across its output terminals, regardless of the current flowing in the source.	Simscape > Electrical > Sources

- 5 Copy the PS-Simulink Converter and Grounded-Neutral (Three-Phase) blocks by right-clicking them and dragging them to new locations on canvas.
- 6 Add a second input port to the Scope block.
 - a Right-click the Scope block.
 - b From the context menu, select **Signals & Ports > Number of Input Ports > 2**
- 7 Connect the blocks as shown.



- 8 Remove the on-canvas annotations titled Open Simscape Library and Open Simscape Electrical Library. Save the model using the name `simplethreephasemodel`.

The blocks in this model use composite three-phase ports. For more information, see “Three-Phase Ports” on page 3-7.

Specify Simulation Parameters

As with Simscape models, you must include a Solver Configuration block in each topologically distinct physical network. This model has a single physical network, so use one Solver Configuration block.

- 1 In the Solver Configuration block, select **Use local solver** and set **Sample time** to `0.0001`.

In Simscape-based models, the local solver is a sample-based solver that represents physical network states as discrete states. For most Simscape Electrical models, the local solver is an appropriate first choice. The solver updates block states once per simulation time step, as determined by **Sample time**. For simulation of a 60-Hz AC system, an appropriate sample time is a value in the order of $1e-4$. For more information on solver options, see Solver Configuration.

If you prefer to use a continuous solver instead of a discrete solver, clear the **Use local solver** check box in the Solver Configuration block. The simulation then uses the Simulink solver specified in the model configuration parameters (**Simulation > Model Configuration Parameters**). For Simscape Electrical models, an appropriate solver choice is the moderately stiff solver ode23t. For a 60 Hz AC system, specify a value for **Max step size** in the order of $1e-4$. For more information, see “Variable-Step Continuous Explicit Solvers” (Simulink).

- 2 In the Simulink Editor, set the simulation **Stop time** to 0.1 .

Load Impedance Parameters

The RLC block models resistive, inductive, and capacitive characteristics of the three-phase load. Using the **Component structure** parameter, you can specify a series or parallel combination of resistance, inductance, and capacitance.

In the RLC block, the defaults are:

- **Component structure** — R.
- **Resistance** — 1Ω .

Using the default **Component structure** value, R, models a three-phase load that is purely resistive in nature. The resistance in *each* phase is 1Ω .


Specify Display Parameters

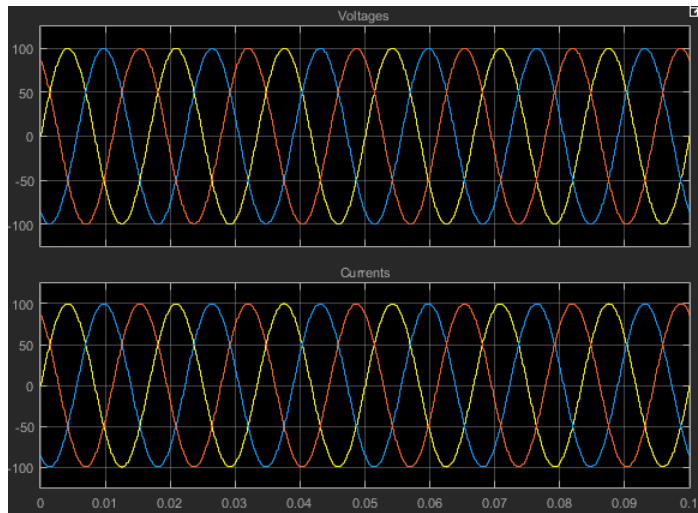
Sensor blocks in the model convert the current and voltage in each phase of the three-phase system to proportional physical signals. PS-Simulink Converter blocks convert the physical signals into Simulink signals for the Scope block to display.

- 1 Of these three types of blocks, only the converter blocks have parameters. For this example:
 - Set **Output signal unit** of the **PS-Simulink Converter1** block to V. This setting ensures that the block outputs a signal with the same magnitude as the voltage signal that enters it.
 - Set **Output signal unit** of the **PS-Simulink Converter2** block to A. This setting ensures that the block outputs a signal with the same magnitude as the current signal that enters it.
- 2 Label the input signals to the Scope block. Double-click each line, and type the appropriate label, **Voltages** or **Currents**, as shown in the model graphic.

You are ready to simulate the model and analyze the results.

Simulate and Analyze the Resistive Three-Phase Model


- 1 Save the model.
- 2 Simulate the model.
- 3 View the phase currents and voltages. Double-click the Scope block.
- 4 From the scope menu, select **View > Configuration Properties**. Set **Layout** to 1-by-2 display.
- 5 To scale the scope axes to the data, click the **Autoscale** button .

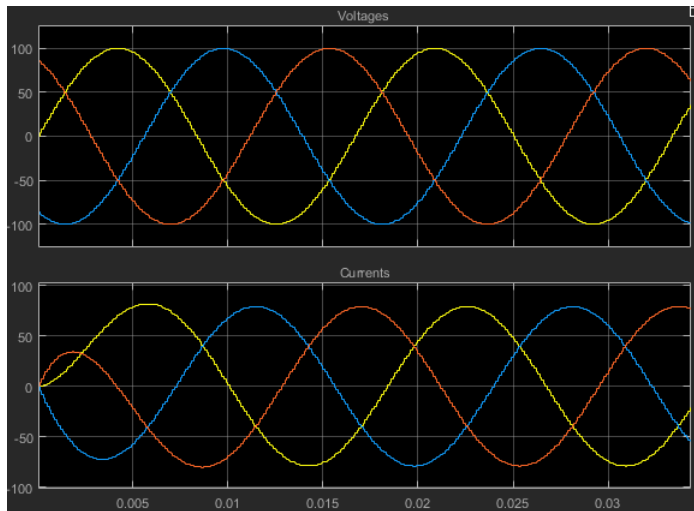


In this simulation, the **Component structure** parameter of the RLC (Three-Phase) block specifies that the electrical characteristics of the three-phase load are purely resistive. Therefore, for each phase of the three-phase system, the voltage and current remain in phase with each other. Because the resistance in each phase is 1Ω , the magnitude of the phase voltage is equal to the magnitude of the phase current.

Simulate and Analyze a Reactive Three-Phase Model

You can modify the model to create a reactive load. A reactive load has inductive or capacitive characteristics.

- 1 Save this version of the model using the name `simplethreephase_model_reactive`.
- 2 In the RLC (Three-Phase) block, set:
 - **Component structure** to Series RL
 - **Inductance** to 0.002
- 3 Simulate the model.
- 4 View the simulation results. Autoscale the scope axes.
- 5 Examine the results in closer detail. For example, click the Zoom button  and drag a box over the first third of one of the plots.

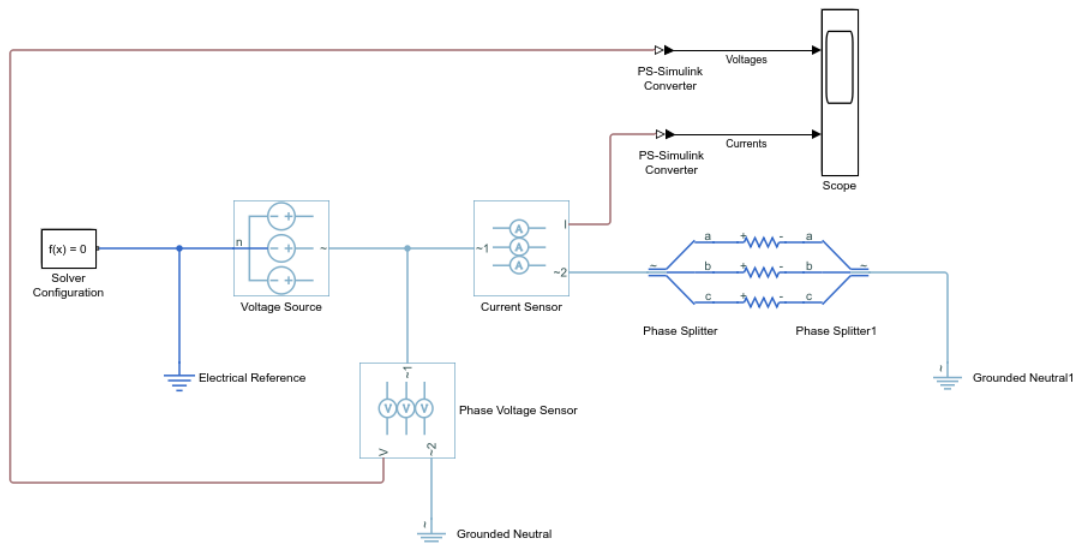


The electrical characteristics of three-phase load are no longer purely resistive. Because the load has an inductive characteristic, the current flowing in each phase lags the voltage.

Create an Expanded Balanced Three-Phase Model

- 1 Open the resistive three-phase model `simplethreephase_model` that you initially created.
- 2 Delete the RLC (Three-Phase) block.

- 3 Drag two copies of the Phase Splitter block into the model from the **Simscape > Electrical > Connections & References** library.
- 4 Flip one of the Phase Splitter blocks horizontally. Right-click the block and select **Rotate & Flip > Flip Block > Left-Right**.
- 5 Drag a Resistor element into the model from the **Simscape > Foundation Library > Electrical > Electrical Elements** library.
- 6 To create space for more components, hide the Resistor element label. Right-click the resistor and select **Format > Show Block Name** to clear this option.
- 7 Make two more copies of the Resistor element.
- 8 Connect the components as shown.



- 9 Save this version of the modified model using the name `simplethreephasemodel_expanded_balanced`.


This model name reflects that the load previously modeled by the RLC block is now expanded into individual phases. The load is still balanced, that is, there is equal resistance in each phase.

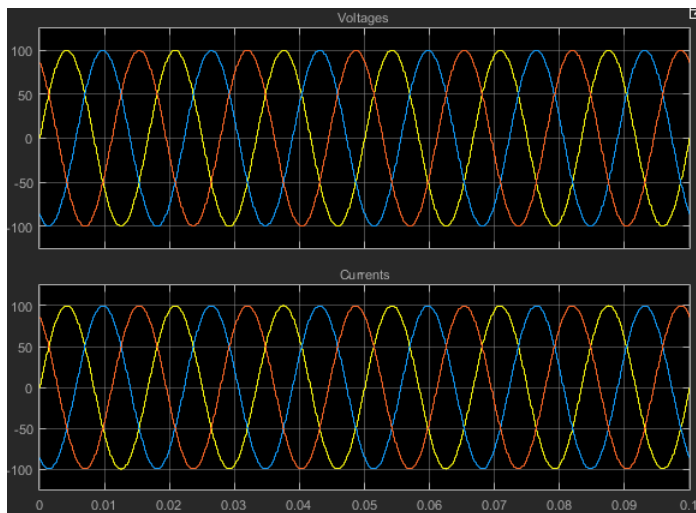
Create an Expanded Unbalanced Three-Phase Model

- 1 Unbalance the load in `simplethreephase_model_expanded_balanced` by changing the resistance in one phase. Double-click the phase-c resistor element. Change **Resistance** to 2.
- 2 Save this version of the modified model using the name `simplethreephase_model_expanded_unbalanced`.

This model name reflects that the three-phase load previously modeled by the RLC block is expanded into individual phases. The load is unbalanced, that is, the resistance in one of the phases is higher than in the other two.

Simulate the Expanded Balanced and Unbalanced Models and Analyze the Results

- 1 Simulate the `simplethreephase_model_expanded_balanced` model. In the menu bar of the Simulink Explorer, click the **Run** button.
- 2 View the simulation results. Double-click the Scope block.
- 3 To scale the scope axes to the data, click the **Autoscale** button .

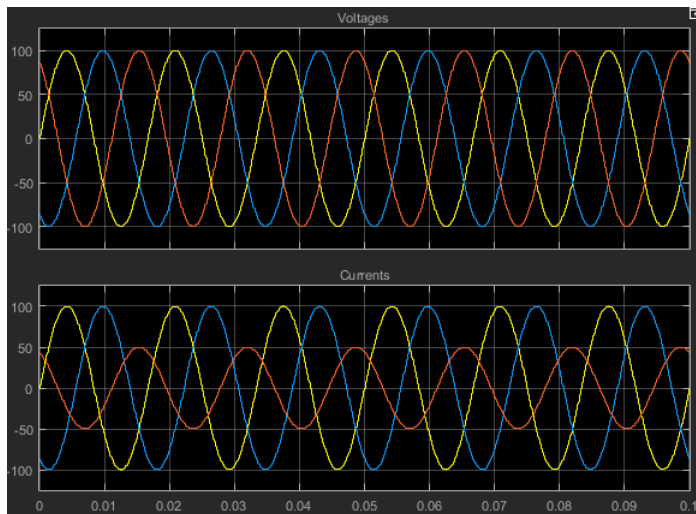


In the `simplethreephase_model`, the **Component structure** parameter of the RLC (Three-Phase) block specifies that the three-phase load is purely resistive. In this

version of the model, the load is expanded into an individual resistive element for each phase, but the resistance in each phase is unchanged. For each phase of the three-phase system, the voltage and current remain in phase with each other. Because the resistance in each phase is 1Ω , the magnitude of the phase voltage is equal to the magnitude of the phase current.

Comparing these results with the results for the three-phase resistive model shows that a block with composite three-phase ports, the RLC (Three-Phase) block in the original model, produces results with the same fidelity as that of expanded phases.

- 4 Open the `simplethreephase_model_expanded_unbalanced` model.
- 5 Simulate the model. Autoscale the scope axes.



In this version of the model, the c-phase of the three-phase load has twice the resistance of the other two. Therefore, half as much current flows in that phase, as the second plot shows. However, because the load remains purely resistive, the voltage and current remain in phase with each other.

See Also

Related Examples

- “Modeling Analog Circuit Architectures, Mechatronic Systems, and Electrical Power Systems Using Simscape Electrical” on page 1-6
- “Essential Electrical Modeling Techniques” on page 3-2

DC Motor Model

In this section...

“Select Blocks to Represent System Components” on page 2-13

“Build the Model” on page 2-14

“Specify Model Parameters” on page 2-15

“Configure the Solver Parameters” on page 2-18

“Run the Simulation and Analyze the Results” on page 2-19

In this example, you model a DC motor driven by a constant input signal that approximates a pulse-width modulated signal and look at the current and rotational motion at the motor output.

To see the completed model, open the PWM-Controlled DC Motor example.

Select Blocks to Represent System Components

Select the blocks to represent the input signal, the DC motor, and the motor output displays.

The following table describes the role of the blocks that represent the system components.

Block	Description
Solver Configuration	Defines solver settings that apply to all physical modeling blocks
PS-Simulink Converter	Converts the input physical signal to a Simulink signal
Controlled PWM Voltage	Generates the signal that approximates a pulse-width modulated motor input signal
H-Bridge	Drives the DC motor
DC Motor	Converts input electrical energy into mechanical motion
Current Sensor	Converts the electrical current that drives the motor into a measurable physical signal proportional to the current
DC Voltage Source	Generates a DC voltage

Block	Description
Electrical Reference	Provides the electrical ground
Mechanical Rotational Reference	Provides the mechanical ground
Ideal Rotational Motion Sensor	Converts the rotational motion of the motor into a measurable physical signal proportional to the motion
Scope	Displays motor current and rotational motion

Build the Model

- 1 Create a new model.
- 2 Add to the model the blocks listed in the following table. The Library column of the table specifies the hierarchical path to each block.

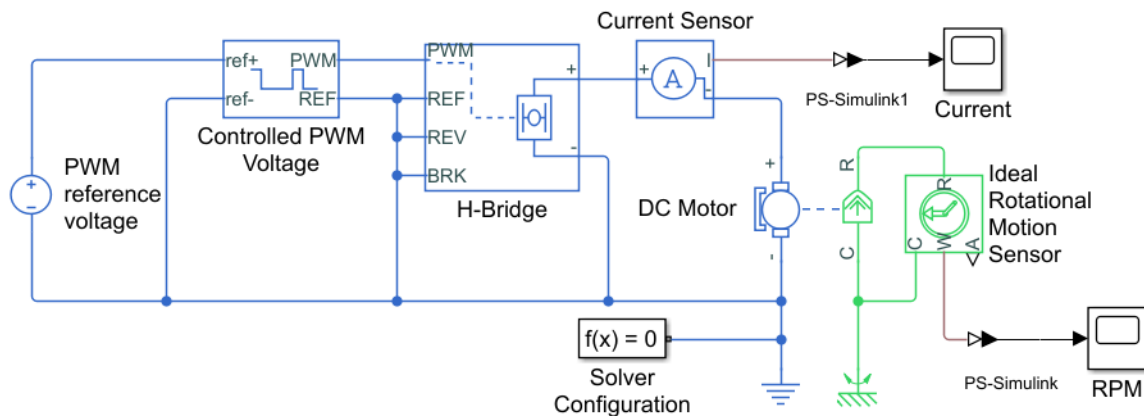
Block	Library	Quantity
Solver Configuration	Simscape > Utilities	1
PS-Simulink Converter	Simscape > Utilities	2
Controlled PWM Voltage	Simscape > Electrical > Integrated Circuits	1
H-Bridge	Simscape > Electrical > Semiconductors & Converters > Converters	1
DC Motor	Simscape > Electrical > Electromechanical > Brushed Motors	1
Current Sensor	Simscape > Foundation Library > Electrical > Electrical Sensors	1
DC Voltage Source	Simscape > Foundation Library > Electrical > Electrical Sources	1
Electrical Reference	Simscape > Foundation Library > Electrical > Electrical Elements	1
Mechanical Rotational Reference	Simscape > Foundation Library > Mechanical > Rotational Elements	1

Block	Library	Quantity
Ideal Rotational Motion Sensor	Simscape > Foundation Library > Mechanical > Mechanical Sensors	1
Scope	Simulink > Commonly Used Blocks	2

Note You can use the Simscape function `ssc_new` with domain type `electrical` to create a Simscape model that contains these blocks:

- **Simulink-PS Converter**
- **PS-Simulink Converter**
- **Scope**
- **Solver Configuration**
- **Electrical Reference**

3 Rename and connect the blocks as shown in the diagram.



Now you are ready to specify block parameters.

Specify Model Parameters

Specify the following parameters to represent the behavior of the system components:

Model Setup Parameters

The following blocks specify model information that is not specific to a particular block:

- Solver Configuration
- Electrical Reference
- Mechanical Rotational Reference

As with Simscape models, you must include a Solver Configuration block in each topologically distinct physical network. This example has a single physical network, so use one Solver Configuration block with the default parameter values.

You must include an Electrical Reference block in each Simscape Electrical network. You must include a Mechanical Rotational Reference block in each network that includes electromechanical blocks. These blocks do not have any parameters.

For more information about using reference blocks, see “Grounding Rules” (Simscape).

Motor Input Signal Parameters

You generate the motor input signal using these blocks:

- The DC Voltage Source block (PWM reference voltage) generates a constant signal.
- The Controlled PWM Voltage block generates a pulse-width modulated signal.
- The H-Bridge block drives the motor.

In this example, all input ports of the H-Bridge block except the PWM port are connected to ground. As a result, the H-Bridge block behaves as follows:

- When the motor is on, the H-Bridge block connects the motor terminals to the power supply.
- When the motor is off, the H-Bridge block acts as a freewheeling diode to maintain the motor current.

In this example, you simulate the motor with a constant current whose value is the average value of the PWM signal. By using this type of signal, you set up a fast simulation that estimates the motor behavior.

- 1 Set the DC Voltage Source block parameters as follows:
 - **Constant voltage** to 2.5

2 Set the Controlled PWM Voltage block parameters as follows:

- **PWM frequency** to 4000
- **Simulation mode** to Averaged

This value tells the block to generate an output signal whose value is the average value of the PWM signal. Simulating the motor with an averaged signal estimates the motor behavior in the presence of a PWM signal. To validate this approximation, use value of PWM for this parameter.

3 Set the H-Bridge block parameters as follows:

- **Simulation mode** to Averaged

This value tells the block to generate an output signal whose value is the average value of the PWM signal. Simulating the motor with an averaged signal estimates the motor behavior in the presence of a PWM signal. To validate this approximation, use value of PWM for this parameter.

Note The simulation mode for both the Controlled PWM Voltage and H-Bridge blocks must be the same.

Motor Parameters

Configure the block that models the motor.

Set the Motor block parameters as follows, leaving the unit settings at their default values where applicable:

- **Electrical Torque** tab:
 - **Model parameterization** to By rated power, rated speed & no-load speed
 - **Armature inductance** to 0.01
 - **No-load speed** to 4000
 - **Rated speed (at rated load)** to 2500
 - **Rated load (mechanical power)** to 10
 - **Rated DC supply voltage** to 12

- **Mechanical** tab:
 - **Rotor inertia** to 2000
 - **Rotor damping** to 1e-06

Current Display Parameters

Specify the parameters of the blocks that create the motor current display:

- Current Sensor block
- **PS-Simulink Converter1** block
- **Current** scope

Of the three blocks, only the PS-Simulink Converter1 block has parameters. Set the PS-Simulink Converter1 block **Output signal unit** parameter to A to indicate that the block input signal has units of amperes.

Torque Display Parameters

Specify the parameters of the blocks that create the motor torque display:

- Ideal Rotational Motion Sensor block
- PS-Simulink Converter block
- **RPM** scope

Of the three blocks, only the PS-Simulink Converter block has parameters you need to configure for this example. Set the PS-Simulink Converter block **Output signal unit** parameter to rpm to indicate that the block input signal has units of revolutions per minute.

Note You must type this parameter value. It is not available in the drop-down list.

Configure the Solver Parameters

Configure the solver parameters to use a continuous-time solver because Simscape Electrical models only run with a continuous-time solver. Increase the maximum step size the solver can take so the simulation runs faster.

- 1 In the model window, select **Simulation > Model Configuration Parameters** to open the Configuration Parameters dialog box.
- 2 Select **ode15s (Stiff/NDF)** from the **Solver** list.
- 3 Expand **Additional options** and enter 1 for the **Max step size** parameter value.
- 4 Click **OK**.

For more information about configuring solver parameters, see “Simulating an Electronic, Mechatronic, or Electrical Power System” on page 3-10.

Run the Simulation and Analyze the Results

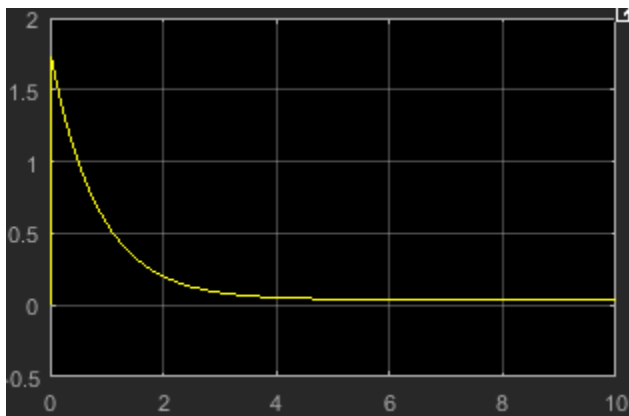
In this part of the example, you run the simulation and plot the results.

In the model window, select **Simulation > Run** to run the simulation.

To view the motor current and torque in the Scope windows, double-click the Scope blocks. You can do this before or after you run the simulation.

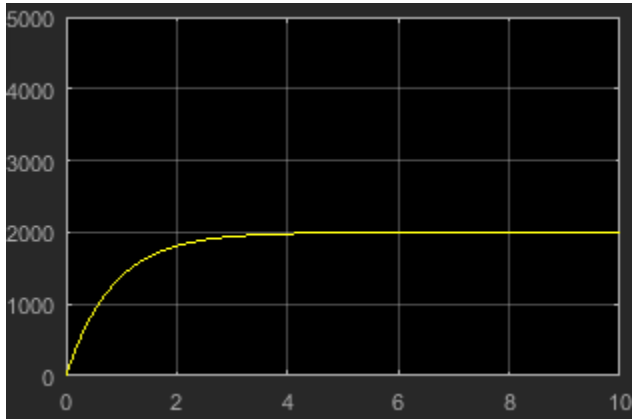
Note By default, the scope displays appear stacked on top of each other on the screen, so you can only see one of them. Click and drag the windows to reposition them.

The following plot shows the motor current.



Motor Current

The next plot shows the motor rpm.



Motor RPM

As expected, the motor runs at about 2000 rpm when the applied DC voltage is 2.5 V.

Triangle Wave Generator Model

In this section...

“Select Blocks to Represent System Components” on page 2-21

“Build the Model” on page 2-22

“Specify Model Parameters” on page 2-24

“Configure the Solver Parameters” on page 2-27

“Simulate Model and Analyze Results” on page 2-28

In this example, you model a triangle wave generator using Simscape Electrical blocks and custom Simscape Electrical blocks, and then look at the voltage at the wave generator output.

You use a classic circuit configuration consisting of an integrator and a noninverting amplifier to generate the triangle wave, and use datasheets to specify block parameters. For more information, see “Parameterizing Blocks from Datasheets” on page 3-17.

To see the completed model, open the Triangle Wave Generator example.

Select Blocks to Represent System Components

First, you select the blocks to represent the input signal, the triangle wave generator, and the output signal display.

You model the triangle wave generator with a set of physical blocks. The wave generator consists of:

- Two operational amplifier blocks
- Resistors and a capacitor that work with the operational amplifiers to create the integrator and noninverting amplifier
- Simulink-PS Converter and PS-Simulink Converter blocks whose function is to bridge the physical part of the model, which uses physical signals, and the rest of the model, which uses Simulink signals.

You have a manufacturer datasheet for the two operational amplifiers you want to model. Later in the example, you use the datasheet to parameterize the Simscape Electrical Band-Limited Op-Amp block.

The following table describes the role of the blocks that represent the system components.

Block	Description
Sine Wave	Generates a sinusoidal signal that controls the resistance of the Variable Resistor block
Scope	Displays the triangular output wave
Simulink-PS Converter	Converts the sinusoidal Simulink signal to a physical signal
Solver Configuration	Defines solver settings that apply to all physical modeling blocks
PS-Simulink Converter	Converts the output physical signal to a Simulink signal
Capacitor	Works with an operational amplifier and resistor block to create the integrator
Resistor	Works with the operational amplifier and capacitor blocks to create the integrator and noninverting amplifier
Variable Resistor	Supplies a time-varying resistance that adjusts the gain of the integrator, which in turn varies the frequency and amplitude of the generated triangular wave
DC Voltage Source	Generates a DC reference signal for the operational amplifier block of the noninverting amplifier
Voltage Sensor	Converts the electrical voltage at the output of the integrator into a physical signal proportional to the current
Electrical Reference	Provides the electrical ground
Band-Limited Op-Amp	Works with the capacitor and resistor to create an integrator and a noninverting amplifier
Diode	Limits the output of the Band-Limited Op-Amp block, to make the output waveform independent of supply voltage

Build the Model

Create a Simulink model, add blocks to the model, and connect the blocks.

- 1 Create a new model.

- 2 Add to the model the blocks listed in this table. The Library Path column of the table specifies the hierarchical path to each block.

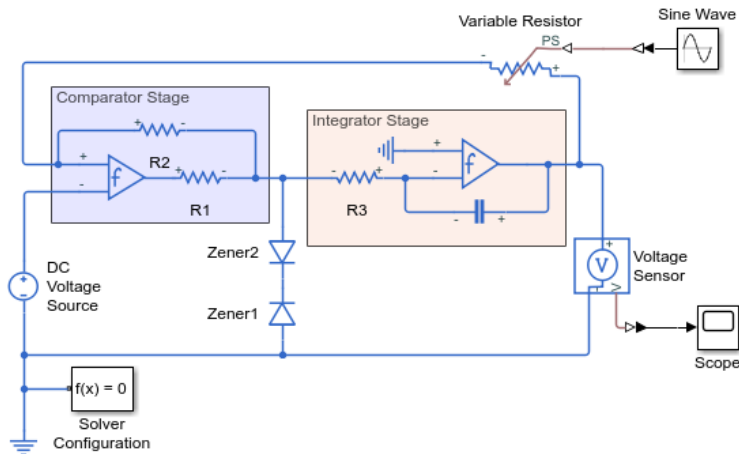
Block	Library Path	Quantity
Sine Wave	Simulink > Sources	1
Scope	Simulink > Commonly Used Blocks	1
Simulink-PS Converter	Simscape > Utilities	1
Solver Configuration	Simscape > Utilities	1
PS-Simulink Converter	Simscape > Utilities	1
Capacitor	Simscape > Foundation Library > Electrical > Electrical Elements	1
Resistor	Simscape > Foundation Library > Electrical > Electrical Elements	3
Variable Resistor	Simscape > Foundation Library > Electrical > Electrical Elements	1
Electrical Reference	Simscape > Foundation Library > Electrical > Electrical Elements	2
DC Voltage Source	Simscape > Foundation Library > Electrical > Electrical Sources	1
Voltage Sensor	Simscape > Foundation Library > Electrical > Electrical Sensors	1
Band-Limited Op-Amp	Simscape > Electrical > Integrated Circuits	2
Diode	Simscape > Electrical > Semiconductor & Converters	2

Note You can use the Simscape function `ssc_new` with a domain type of `electrical` to create a Simscape model that contains these blocks:

- **Simulink-PS Converter**
- **PS-Simulink Converter**

- **Scope**
- **Solver Configuration**
- **Electrical Reference**

- 3 Rename and connect the blocks as shown in the diagram. The blocks in the triangle wave generator circuit are organized in two stages. The Comparator Stage contains a comparator constructed from a Band-Limited Op-Amp block and two Resistor blocks. The Integrator Stage contains an integrator constructed from another Band-Limited Op-Amp block, a Resistor, a Capacitor, and Electrical Reference.



Specify Model Parameters

Specify these parameters to represent the behavior of the system components:

- “Model Setup Parameters” on page 2-24
- “Input Signal Parameters” on page 2-25
- “Triangle Wave Generator Parameters” on page 2-25
- “Signal Display Parameters” on page 2-27

Model Setup Parameters

These blocks specify model information that is not specific to a particular block:

- Solver Configuration
- Electrical Reference

As with Simscape models, you must include a Solver Configuration block in each topologically distinct physical network. This example has a single physical network, so use one Solver Configuration block with the default parameter values.

You must include an Electrical Reference block in each Simscape Electrical network. This block does not have any parameters.

Input Signal Parameters

Generate the sinusoidal control signal using the Sine Wave block.

Set the Sine Wave block parameters as follows:

- **Amplitude** — $0.5e4$
- **Bias** — $1e4$
- **Frequency** — $\pi/5e-4$

Triangle Wave Generator Parameters

Configure the blocks modeling the physical system that generates the triangle wave:

- Integrator stage — Band-Limited Op-Amp, Capacitor, and Resistor block R3
 - Comparator stage — Band-Limited Op-Amp1, Resistor blocks R1 and R2
 - Variable Resistor
 - Diode and Diode1
 - Simulink-PS Converter and PS-Simulink Converter blocks that bridge the physical part of the model and the Simulink part of the model.
- 1 Accept the default parameters for the Simulink-PS Converter block. These parameters establish the units of the physical signal at the block output such that they match the expected default units of the Variable Resistor block input.
 - 2 Set the two Band-Limited Op-Amp block parameters for the LM7301 device with a +-20V power supply:
 - The datasheet gives the gain as 97 dB, which is equivalent to $10^{(97/20)} = 7.1e4$. Set the **Gain, A** parameter to $7.1e4$.

- The datasheet gives input resistance as 39 Mohms. Set **Input resistance, Rin** to 39e6.
 - Set **Output resistance, Rout** to 0 ohms. The datasheet does not quote a value for Rout, but the term is insignificant compared to the output resistor that it drives.
 - Set minimum and maximum output voltages to -20 V and +20 V, respectively.
 - The datasheet gives the maximum slew rate as 1.25 V/ μ s. Set the **Maximum slew rate, Vdot** parameter to 1.25e6 V/s.
 - Set the bandwidth to 4e6.
- 3 Set the two Diode block parameters for a 4.3 V zener diode. To model a BZX384-B4V3, set block parameters as follows:
- On the **Main** tab, set **Diode model** to Piecewise Linear. This selects a simplified Zener diode model that is more than adequate to test the correct operation of this circuit.
 - Leave the **Forward voltage** as 0.6 V — this is a typical value for most diodes.
 - The datasheet gives the forward current as 250 mA when the forward voltage is 1V. So that the Diode block matches this, set the **On resistance** to $(1 \text{ V} - 0.6 \text{ V}) / 250 \text{ mA} = 1.6 \text{ ohms}$.
 - The datasheet gives the reverse leakage current as 3 μ A at a reverse voltage of 1 V. Therefore, set the **Off conductance** to $3 \mu\text{A} / 1 \text{ V} = 3\text{e-}6 \text{ S}$.
 - The datasheet gives the reverse voltage as 4.3 V. On the Breakdown tab, set the **Reverse breakdown voltage Vz** to 4.3 V.
 - Set the **Zener resistance Rz** to a suitably small number. The datasheet quotes the Zener voltage for a reverse current of 5 mA. For the Diode block to be representative of the real device, the simulated reverse voltage should be close to 4.3V at 5mA. As Rz tends to zero, the reverse breakdown voltage tends to Vz regardless of current, as the voltage-current gradient becomes infinite. However, for good numerical properties, Rz must not be made too small. If, say, you allow a 0.01 V error on the Zener voltage at 5 mA, then Rz is $0.01 \text{ V} / 5 \text{ mA} = 2 \text{ ohms}$. Set the **Zener resistance** parameter to this value.
- 4 The Voltage Sensor block does not have any parameters.
- 5 Accept the default parameters for the Variable Resistor block. These parameters establish the units of the physical signal at the block output such that they match the expected default units of the Variable Resistor block input.
- 6 Set the Capacitor block parameters as follows:

- **Capacitance** — $2.5e-9$
- **Capacitor voltage** — 0.08

This value starts the oscillation in the feedback loop. It is found in the **Variables** tab.

- **Series resistance** — 0
- 7 Set the DC Voltage Source block **Constant voltage** parameter to 0 .
 - 8 Set the Resistor R3 block **Resistance** parameter to 10000 .
 - 9 Set the Resistor R1 block **Resistance** parameter to 1000 .
 - 10 Set the Resistor R2 block **Resistance** parameter to 10000 .
 - 11 Accept the default parameters for the PS-Simulink Converter block. These parameters establish the units of the physical signal at the block output such that they match the expected default units of the Scope block input.

Signal Display Parameters

Specify the parameters of the Scope block to display the triangular output signal.

Double-click the Scope block and then click the **View > Configuration Properties** to open the Scope Configuration Properties dialog box. On the **Logging** tab, clear the **Limit data points to last** check box.

Configure the Solver Parameters

Configure the solver parameters to use a continuous-time solver. Simscape Electrical models only run with a continuous-time solver when the Simscape Solver Configuration block has its **Local Solver** parameter cleared. You also change the simulation end time, tighten the relative tolerance for a more accurate simulation, and remove the limit on the number of simulation data points Simulink saves.

- 1 In the model window, select **Simulation > Model Configuration Parameters** to open the Configuration Parameters dialog box.
- 2 In the **Solver** category in the **Select** tree on the left side of the dialog box:
 - Enter $2000e-6$ for the **Stop time** parameter value.
 - Select `ode23t (Mod. stiff/Trapezoidal)` from the **Solver** list.
 - Enter $4e-5$ for the **Max step size** parameter value.

- Enter $1e-6$ for the **Relative tolerance** parameter value.
- 3 In the **Data Import/Export** category in the **Select** tree, clear the **Limit data points to last** check box.
 - 4 Click **OK**.

For more information about configuring solver parameters, see “Simulating an Electronic, Mechatronic, or Electrical Power System” on page 3-10.

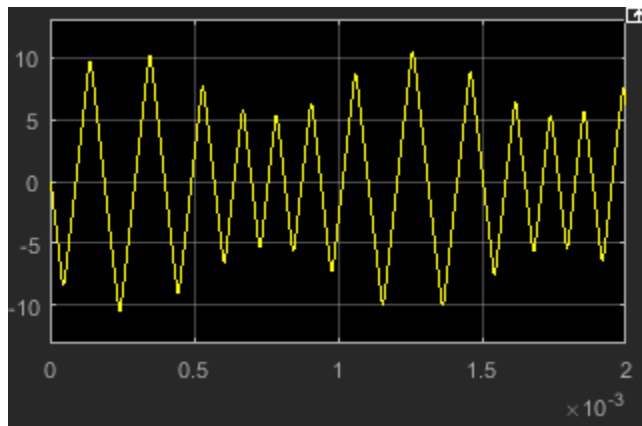
Simulate Model and Analyze Results

Run the simulation and plot the results.

In the model window, select **Simulation > Run** to run the simulation.

To view the triangle wave in the Scope window, double-click the Scope block. You can do this before or after you run the simulation.

The following plot shows the voltage waveform. As the resistance of the Variable Resistor block increases, the amplitude of the output waveform increases and the frequency decreases.



Triangle Waveform Voltage

Modeling and Simulation Basics

- “Essential Electrical Modeling Techniques” on page 3-2
- “Three-Phase Ports” on page 3-7
- “Switch Between Physical Signal and Electrical Ports” on page 3-9
- “Simulating an Electronic, Mechatronic, or Electrical Power System” on page 3-10
- “Selecting the Output Model for Logic Blocks” on page 3-13
- “Parameterizing Blocks from Datasheets” on page 3-17
- “Parameterize a Piecewise Linear Diode Model from a Datasheet” on page 3-19
- “Parameterize an Exponential Diode from a Datasheet” on page 3-23
- “Parameterize an Exponential Diode from SPICE Netlist” on page 3-27
- “Parameterize an Op-Amp from a Datasheet” on page 3-31
- “Additional Parameterization Workflows” on page 3-33
- “Simulating Thermal Effects in Rotational and Translational Actuators” on page 3-35
- “Simulating Thermal Effects in Semiconductors” on page 3-38
- “Simulate Thermal Effects Using Simscape Electrical Thermal Blocks” on page 3-43
- “Plot Basic Characteristics for Battery Blocks” on page 3-54
- “Plot Basic Characteristics for Semiconductor Blocks” on page 3-57
- “MOSFET Characteristics Viewer” on page 3-60
- “Converting a SPICE Netlist to Simscape Blocks” on page 3-69

Essential Electrical Modeling Techniques


In this section...
“Overview of Modeling Rules” on page 3-2
“Required Blocks” on page 3-3
“Creating a New Model” on page 3-4
“Modeling Instantaneous Events” on page 3-4
“Using Simulink Blocks to Model Physical Components” on page 3-5

Overview of Modeling Rules


Simscape Electrical models are essentially Simscape block diagrams refined for modeling single- and multi-phase electronic, mechatronic, and electrical power systems. Simscape Electrical blocks feature these port types:

- Three-phase ports, which connect the phases of a three-phase electrical system between Simscape Electrical blocks.

There are two three-phase port types in Simscape Electrical blocks, composite and expanded. You can connect a composite three-phase port only to another composite three-phase port. You can connect the individual electrical conserving ports of an expanded three-phase port only to other electrical conserving ports. For more information, see “Three-Phase Ports” on page 3-7.

- Electrical and mechanical rotational conserving ports , which connect directly to Simscape foundation blocks.

Each port type has specific Across and Through variables associated with it. To learn about the rules to follow when building an electromechanical model, see “Basic Principles of Modeling Physical Networks” (Simscape).

- Physical signal ports , which connect to Simulink blocks through Simulink-PS Converter and PS-Simulink Converter blocks from the Simscape Utilities library. These blocks convert physical signals to and from Simulink mathematical signals.

Keep these rules in mind when using each port type in Simscape Electrical blocks.

- You can connect physical conserving ports only to other conserving ports of the same type. Electrical conserving ports in Simscape Electrical blocks can connect directly to

Simscape electrical components. Mechanical rotational conserving ports in Simscape Electrical blocks can connect directly to Simscape mechanical rotational components.

- The physical connection lines that connect conserving ports are nondirectional lines that carry physical variables (Across and Through variables) rather than signals. You cannot connect physical conserving ports to Simulink ports or to physical signal ports.
- You can branch physical connection lines. When you do so, directly connected components share the same Across variables. The value of any Through variable (e.g., current or torque) transferred along the physical connection line is divided among the multiple components connected by the branches.

For each Through variable, the sum of the values flowing into a branch point equals the sum of the values flowing out.

- You can connect physical signal ports to other physical signal ports using regular connection lines, similar to Simulink signal connections. These connection lines carry physical signals between Simscape Electrical blocks.
- You can connect physical signal ports to Simulink ports through converter blocks. Use the Simulink-PS Converter block to connect Simulink outputs to physical signal inports. Use the PS-Simulink Converter block to connect physical signal outputs to Simulink inports.
- Unlike Simulink signals, physical signals can have units. In Simscape Electrical block dialog boxes, you can specify the units along with the parameter values, where appropriate. Use the converter blocks to associate units with an input signal and to specify the desired output signal units.

For an example of these rules applied to an electromechanical model, see *Three-Phase Asynchronous Machine Starting*.

Required Blocks

Each topologically distinct physical network in a diagram requires exactly one Solver Configuration block from the Simscape Utilities library. The Solver Configuration block specifies global environment information for simulation and provides parameters for the solver that your model needs for simulation.

Each electrical network requires an Electrical Reference block. This block establishes the electrical ground for the circuit. Networks with electromechanical blocks also require a Mechanical Rotational Reference block. For more information about using reference blocks, see “Grounding Rules” (Simscape).

Creating a New Model

An easy way to start a new Simscape Electrical model, prepopulated with the required blocks, is to use the Simscape function `ssc_new`. For more information, see “Creating a New Simscape Model” (Simscape).

Another way to start a new model is to use a Simscape template from the Simulink start page. The start page includes model templates that provide you with design patterns for modeling electrical, three-phase electrical, mechanical rotational, and mechanical translational networks using Simscape Electrical. For more information, see “Modeling Analog Circuit Architectures, Mechatronic Systems, and Electrical Power Systems Using Simscape Electrical” on page 1-6.

You can also use the Simscape “Creating A New Circuit” (Simscape) example as a template for a new electronic circuit model. This example opens a simple electrical model, prepopulated with some useful blocks, and also opens an Electrical Starter Palette, which contains links to the most often used electrical components. Open the example by typing `ssc_new_elec` in the MATLAB Command Window and use **File > Save As** to save the example model under the desired name. Then delete the unwanted blocks and add new ones from the Electrical Starter Palette and from the block libraries.

Modeling Instantaneous Events

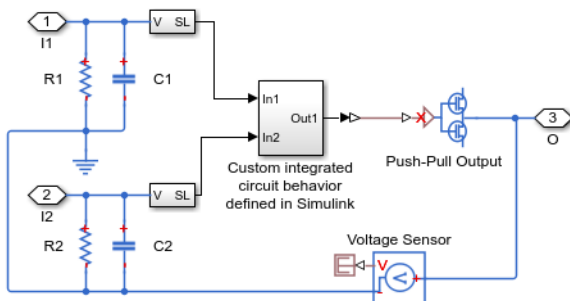
When working with Simscape Electrical, your model may include Simulink blocks that are associated with events or discrete sampling. Such blocks can create instantaneous changes to the physical system inputs through the Simulink-PS Converter block that connects them. When you build this type of model, make sure the corresponding zero crossings are generated.

Many blocks in the Simulink library generate these zero crossings by default. For example, the Pulse Generator block produces a discrete-time output by default, and generates the corresponding zero crossings. To generate zero crossings for all Simulink blocks that model instantaneous events, in the Solver Configuration Parameters for the model, expand **Solver details** and in the **Zero crossing options**, for the **Zero crossing control** option, select Use local settings or Enable all. For more information about zero crossing control, see “Zero-crossing control” (Simulink).

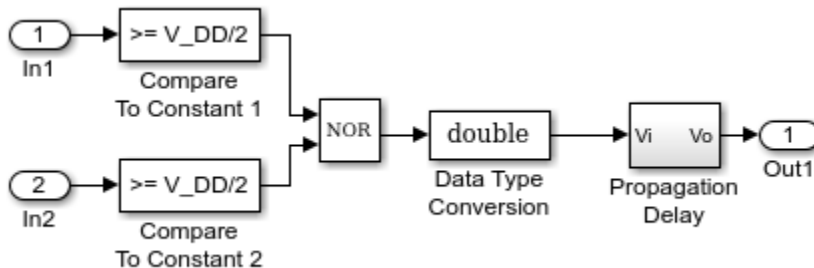
Using Simulink Blocks to Model Physical Components

To run a fast simulation that approximates the behavior of the physical components in a system, you may want to use Simulink blocks to model one or more physical components.

The Modeling an Integrated Circuit example uses Simulink to model a physical component. The 2-Input NOR (Behavioral Model) masked subsystem is a behavioral model, built using Simscape Foundation Library blocks.



This behavioral model contains a subsystem comprised of Simulink blocks, which implements the custom integrated circuit behavior.



The Simulink Logical Operator block implements the behavioral model of the two-input NOR gate. Using Simulink in this manner introduces algebraic loops, unless you place a lag somewhere between the physical signal inputs and outputs. In this case, a first-order lag is included in the Propagation Delay subsystem to represent the delay due to gate capacitances. For applications where no lag is required, use blocks from the Physical

Signals sublibrary in the Simscape Foundation Library to implement the desired functionality.

Three-Phase Ports

In this section...

“About Three-Phase Ports” on page 3-7

“Expand and Collapse Three-Phase Ports on a Block” on page 3-8

About Three-Phase Ports

In Simscape Electrical software, you can connect the phases of a three-phase system between blocks using two types of port.

- Composite three-phase port
- Expanded three-phase port

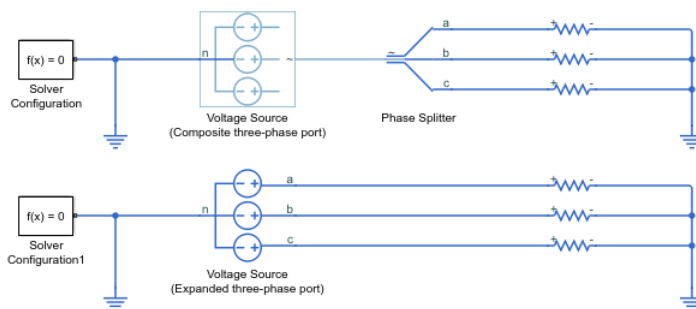
Composite three-phase ports represent three individual electrical conserving ports with a single block port. You can use composite three-phase ports to build models that correspond to single-line diagrams of three-phase electrical systems. Instead of explicitly connecting each phase of the three-phase system between blocks, you connect all three phases using a single port. You can connect composite three-phase ports only to other composite three-phase ports.

Expanded three-phase ports represent the individual phases of a three-phase system using three separate electrical conserving ports. You individually connect each phase of the three-phase system between blocks. Electrical conserving ports can connect directly to electrical components from the Simscape and Simscape Electrical libraries.

Composite three-phase ports produce results with the same fidelity as expanded three-phase ports. Both connection methods consider instantaneous phase voltages and currents and are suitable for modeling balanced and unbalanced three-phase electrical power systems. Each electrical conserving port in an expanded three-phase port has a Through variable of scalar current and an Across variable of scalar voltage. For a composite three-phase port, the Through variable is a three-element current, and the Across variable is a three-element voltage.

You can use the Phase Splitter block to expand a composite three-phase port into separate electrical conserving ports. The separate electrical ports can then connect to Simscape Electrical electrical components.

The figure shows two simple circuits that contrast the composite and expanded connection methods. The two circuits produce the same results.



The top circuit uses a Voltage Source block with a composite three-phase port \sim . The bottom circuit uses a Voltage Source block with expanded electrical conserving ports **a**, **b**, and **c**. In each circuit, the instantaneous phase voltages and currents are the same.

Expand and Collapse Three-Phase Ports on a Block



Simscape Electrical blocks that have composite three-phase ports have an option to switch between composite and expanded ports.

- Right-click the block. On the **Simscape block choices** context menu, select **Expanded three-phase ports** or **Composite three-phase ports**.

For blocks with a single composite port \sim , the expanded electrical ports are labeled **a**, **b**, and **c**. For blocks with more than one composite port ~ 1 and ~ 2 , the expanded electrical ports are labeled **a1**, **b1**, **c1** and **a2**, **b2**, **c2**.

Switch Between Physical Signal and Electrical Ports

Some Simscape Electrical blocks have an option to switch certain ports between physical signal and electrical conserving ports. An electrical conserving port is a Simscape physical conserving port that has a Through variable of current and an Across variable of voltage. For a comparison of Simscape physical signal and physical conserving ports, see “Connector Ports and Connection Lines” (Simscape).

To switch a block connection port between an electrical conserving port  and a physical signal port :

- 1 Right-click the block.
- 2 On the **Simscape > Block choices** context menu, select a variant that includes that includes the term **Electrical**, for an electrical conserving port, or **PS** for a physical signal port.

Simulating an Electronic, Mechatronic, or Electrical Power System

In this section...
“Selecting a Solver” on page 3-10
“Specifying Simulation Accuracy/Speed Tradeoff” on page 3-10
“Avoiding Simulation Issues” on page 3-11
“Running a Time-Domain Simulation” on page 3-11
“Running a Small-Signal Frequency-Domain Analysis” on page 3-12

Selecting a Solver

Simscape Electrical software supports all of the continuous-time solvers that Simscape supports. For more information, see “Setting Up Solvers for Physical Models” (Simscape).

You can select any of the supported solvers for running a simulation of an electronic model. The variable-step solvers, `ode23t` and `ode15s`, are recommended for most applications because they run faster and work better for systems with a range of both fast and slow dynamics. The `ode23t` solver is closest to the solver that SPICE traditionally uses.

To use Simulink Coder software to generate standalone C or C++ code from your model, you must use the `ode14x` solver. For more information about code generation, see “Code Generation” (Simscape).

Specifying Simulation Accuracy/Speed Tradeoff

To trade off accuracy and simulation time, adjust one or more of the following parameters:

- **Relative tolerance** in the Simulink Configuration Parameters dialog box
- **Absolute tolerance** in the Simulink Configuration Parameters dialog box
- **Max step size** in the Simulink Configuration Parameters dialog box
- **Consistency Tolerance** in the Solver Configuration block dialog box

In most cases, the default tolerance values produce accurate results without sacrificing unnecessary simulation time. The parameter value that is most likely to be inappropriate

for your simulation is **Max step size**, because the default value, `auto`, depends on the simulation start and stop times rather than on the amount by which the signals are changing during the simulation. If you are concerned about the solver missing significant behavior, change the parameter to prevent the solver from taking too large a step.

The Simulink documentation describes the following parameters in more detail and provides tips on how to adjust them:

- “Relative tolerance” (Simulink)
- “Absolute tolerance” (Simulink)
- “Max step size” (Simulink)

The Solver Configuration block reference page in the Simscape documentation explains when to adjust the **Consistency Tolerance** parameter value.

Avoiding Simulation Issues

If you experience a simulation issue, first read “Troubleshooting Simulation Errors” (Simscape) to learn about general troubleshooting techniques.

There are a few techniques you can apply to any Simscape Electrical model to overcome simulation issues:

- Add parasitic capacitors and/or resistors (specifically, junction capacitance and ohmic resistance) to the circuit to avoid numerical issues. The Astable Oscillator example uses these devices.
- Adjust the current and voltage sources so they start at zero and ramp up to their final values rather than starting at nonzero values.

To learn about avoiding simulation errors in the presence of specific Simscape Electrical model configurations, see “Modeling Instantaneous Events” on page 3-4 and “Using Simulink Blocks to Model Physical Components” on page 3-5.

Running a Time-Domain Simulation

When you run a time-domain simulation, Simscape Electrical software uses the Simscape solver to analyze the physical system in the Simulink environment. For more information, see “How Simscape Simulation Works” (Simscape).

Running a Small-Signal Frequency-Domain Analysis

You can perform small-signal analysis for Simscape and Simscape Electrical models using linearization capabilities of Simulink software. For more information, see “Linearize an Electronic Circuit” (Simscape).

See Also

Solver Configuration

More About

- “How Simscape Simulation Works” (Simscape)
- “Trimming and Linearization” (Simscape)
- “Troubleshooting” (Simscape)

Selecting the Output Model for Logic Blocks

In this section...

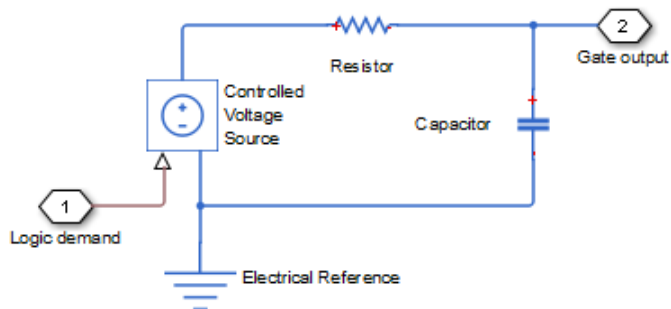
“Available Output Models” on page 3-13

“Quadratic Model Output and Parameters” on page 3-14

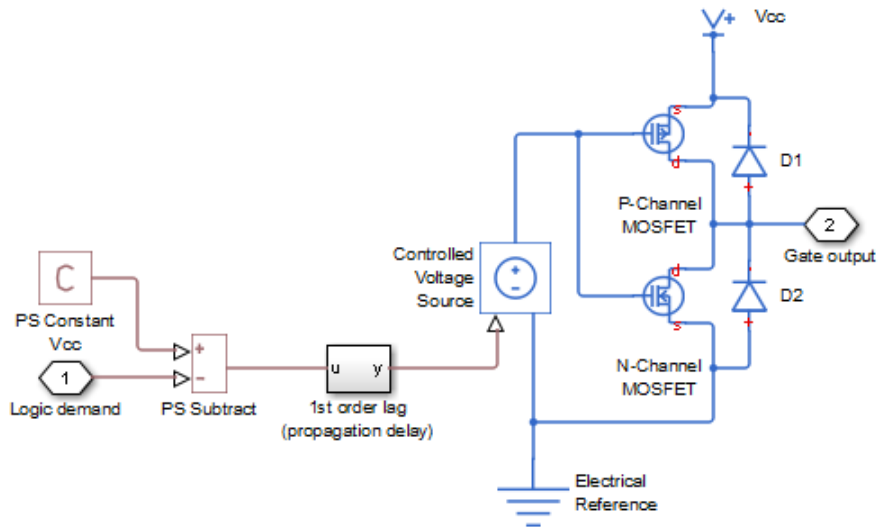
Available Output Models

The blocks in the Logic sublibrary of the Integrated Circuits library provide a choice of two output models:

- **Linear** — Models the gate output as a voltage source driving a series resistor and capacitor connected to ground. This is suitable for logic circuit operation under normal conditions and when the logic gate drives other high-impedance CMOS gates. The block sets the value of the gate output capacitor such that the resistor-capacitor time constant equals the **Propagation delay** parameter value. The linear output model is shown in the following illustration.



- **Quadratic** — Models the gate output in terms of a complementary N-channel and P-channel MOSFET pair. This adds more fidelity, which becomes relevant if drawing higher currents from the gate output, or if exercising the gate under fault conditions. In addition, the gate input demand is lagged to approximate the **Propagation delay** parameter value. Default parameters are representative of the 74HC logic gate family. The quadratic output model is shown in the next illustration.



Use the **Output current-voltage relationship** parameter on the **Outputs** tab of the block dialog box to specify the output model.

For most system models, MathWorks recommends selecting the linear option because it supports faster simulation. If necessary, you can use the more detailed output model to validate simulation results obtained from the simpler model.

Quadratic Model Output and Parameters

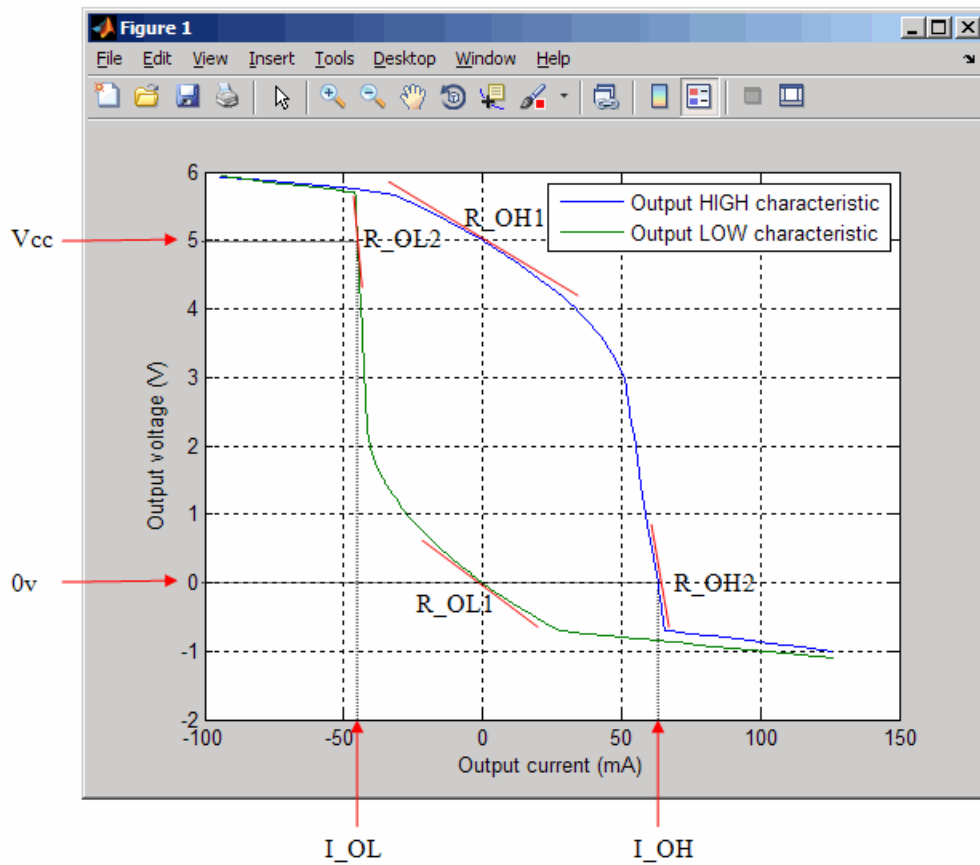
If you select the quadratic model, use the following parameters to control the block output:

- **Supply voltage** — Supply voltage value (Vcc) applied to the gate in your circuit. The default value is 5 V.
- **Measurement voltage** — The gate supply voltage for which mask data output resistances and currents are defined. The default value is 5 V.
- **Logic HIGH output resistance at zero current and at I_OH** — A row vector [R_{OH1} R_{OH2}] of two resistance values. The first value R_{OH1} is the gradient of the output voltage-current relationship when the gate is logic HIGH and there is no output current. The second value R_{OH2} is the gradient of the output voltage-current

relationship when the gate is logic HIGH and the output current is I_{OH} . The default value is [25 250] Ω .

- **Logic HIGH output current I_{OH} when shorted to ground** — The resulting current when the gate is in the logic HIGH state, but the load forces the output voltage to zero. The default value is 63 mA.
- **Logic LOW output resistance at zero current and at I_{OL}** — A row vector [R_{OL1} R_{OL2}] of two resistance values. The first value R_{OL1} is the gradient of the output voltage-current relationship when the gate is logic LOW and there is no output current. The second value R_{OL2} is the gradient of the output voltage-current relationship when the gate is logic LOW and the output current is I_{OL} . The default value is [30 800] Ω .
- **Logic LOW output current I_{OL} when shorted to Vcc** — The resulting current when the gate is in the logic LOW state, but the load forces the output voltage to the supply voltage Vcc. The default value is -45 mA.
- **Propagation delay** — Time it takes for the output to swing from LOW to HIGH or HIGH to LOW after the input logic levels change. For quadratic output, it is implemented by the lagged gate input demand. The default value is 25 ns.
- **Protection diode on resistance** — The gradient of the voltage-current relationship for the protection diodes when forward biased. The default value is 5 Ω .
- **Protection diode forward voltage** — The voltage above which the protection diode is turned on. The default value is 0.6 V.

The following graphic illustrates the quadratic output model parameterization, using the default parameter output characteristics for a +5V supply.



Parameterizing Blocks from Datasheets

Simscape Electrical is a system-level simulation tool that provides blocks with a commensurate level of fidelity. Block parameters are designed, where possible, to match the data found on manufacturer datasheets. For example, the bipolar transistor blocks support parameterization in terms of the small-signal quantities usually quoted on a datasheet, and the underlying model is simpler than that typically used by specialist EDA simulation tools. The smaller number of parameters and simpler underlying models can support MATLAB system performance analysis better, and thereby support design choices. Following system design, you can perform validation in hardware or more detailed modeling and validation using an EDA simulation tool.

The following parameterization examples illustrate various block parameterization techniques:

- Example 1: “Parameterize a Piecewise Linear Diode Model from a Datasheet” on page 3-19
- Example 2: “Parameterize an Exponential Diode from a Datasheet” on page 3-23
- Example 3: “Parameterize an Exponential Diode from SPICE Netlist” on page 3-27
- Example 4: “Parameterize an Op-Amp from a Datasheet” on page 3-31

Most of the time, datasheets should be a sufficient source of parameters for Simscape Electrical blocks (see Examples 1 on page 3-19, 2 on page 3-23, and 4 on page 3-31). Sometimes, there is need for more information than is available on the datasheet, and data can be augmented from a manufacturer SPICE netlist. For example, circuit performance may depend on one or two critical components, and increased accuracy is needed either for parameter values or the underlying model. Simscape Electrical libraries contain a SPICE-compatible sublibrary to support this case, and this is illustrated by Example 3 on page 3-27. If you have many components that need to be modeled to a high level of accuracy, then Simulink cosimulation with a specialist circuit simulator may be a better option.

You can also use the SPICE conversion assistant to convert SPICE components into Simscape equivalents. For more information, see “Converting a SPICE Netlist to Simscape Blocks” on page 3-69

In mechatronic applications in particular, you may need to model input-output behavior of integrated circuits, such as PWM waveform generators and H-bridges. For these two examples, Simscape Electrical libraries contain abstracted-behavior equivalent blocks that you can use. Where you need to model other devices, possible options include

creating your own abstracted model using the Simscape language, or using Simulink blocks. For an example of using Simulink blocks, see the Modeling an Integrated Circuit example.

When looking for a datasheet, make sure you have the originating manufacturer datasheet because some resellers abbreviate them.

For additional ways to parameterize and validate your model, see “Additional Parameterization Workflows” on page 3-33.

Parameterize a Piecewise Linear Diode Model from a Datasheet

The Triangle Wave Generator example model, also described in “Triangle Wave Generator Model” on page 2-21, contains two zener diodes that regulate the maximum output voltage from an op-amp amplifier circuit. Each of these diodes is implemented with the Simscape Electrical Diode block, parameterized using the **Piecewise Linear** option. This simple model is sufficient to check correct operation of the circuit, and requires fewer parameters than the **Exponential** option of the Diode block. However, when specifying the parameters, you need to take into account the bias condition that will be used in the circuit. This example explains how to do this.

The Phillips Semiconductors datasheet for a BZX384-B4V3 gives the following data:

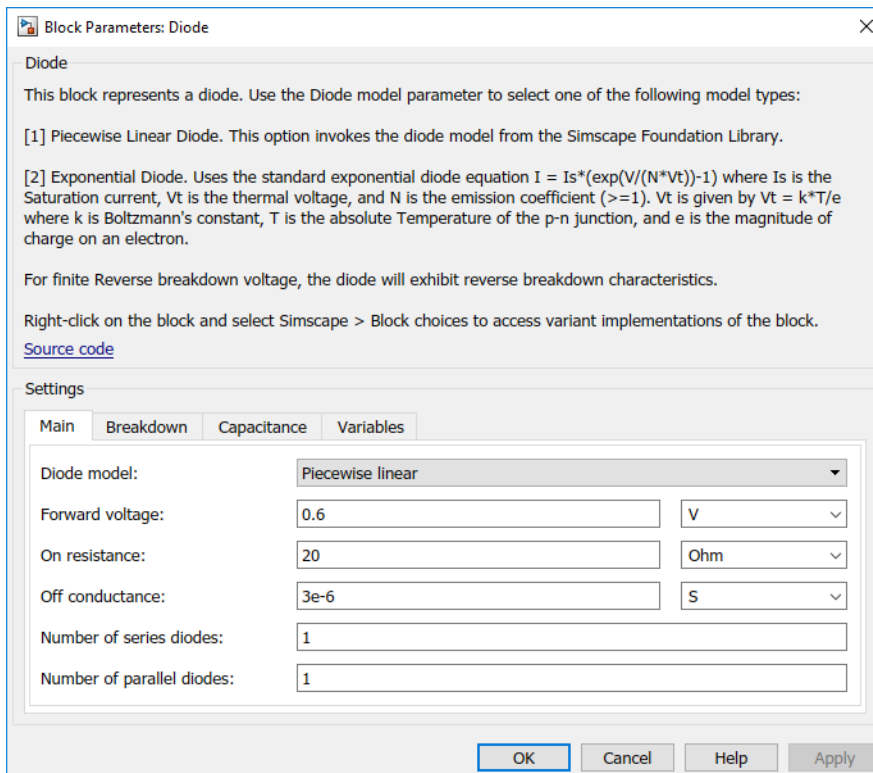
Working voltage, V_Z (V) at $I_{Z_{test}} = 5 \text{ mA}$	4.3
Diode capacitance, C_d (pF)	450
Reverse current, I_R (μA) at $V_R = 1 \text{ V}$	3
Forward voltage, V_F (V) at $I_F = 5 \text{ mA}$	0.7

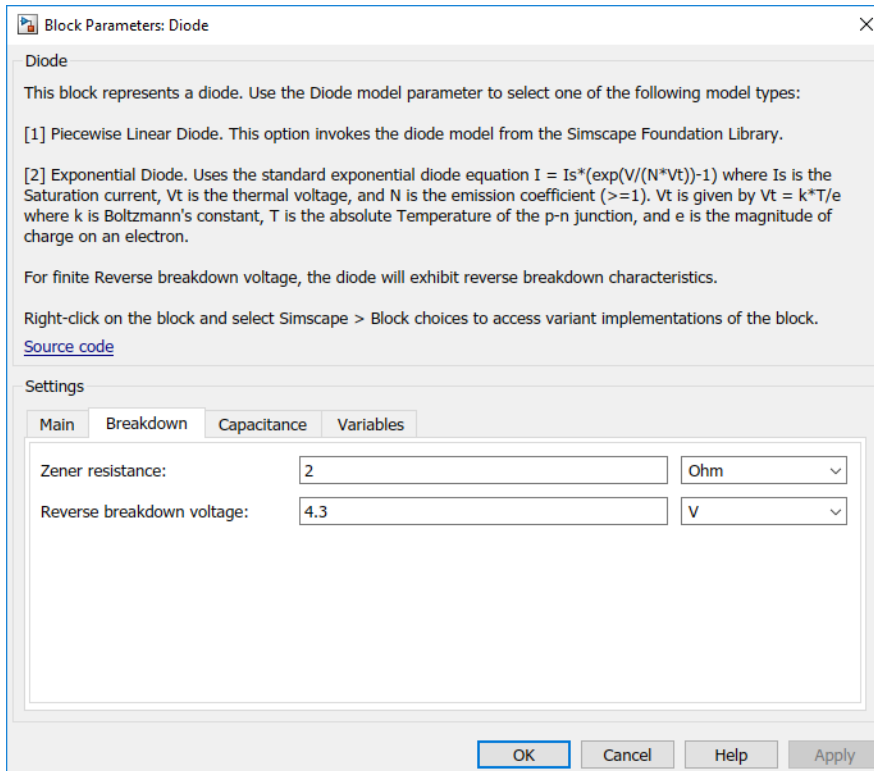
In the datasheet, the tabulated values for V_F are for higher forward currents. This value of 0.7V at 5mA is extracted from the datasheet current-voltage curve, and is chosen as it matches the zener current used when quoting the working voltage of 4.3V.

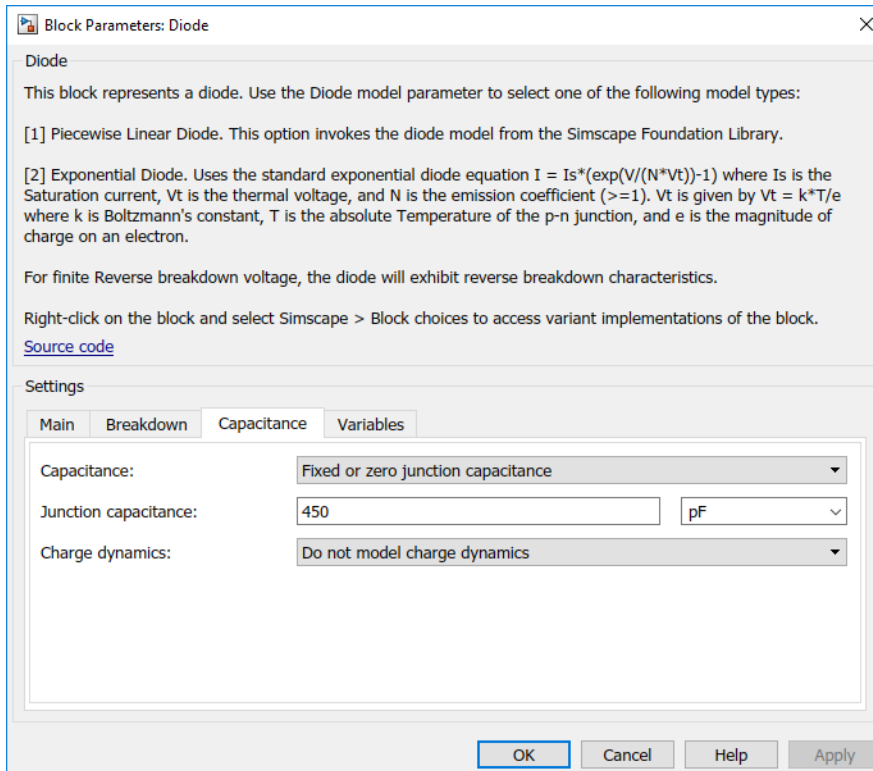
To match the datasheet values, the example sets the piecewise linear diode block parameters as follows:

- **Forward voltage.** Leave as default value of 0.6V. This is a typical value for most diodes, and the exact value is not critical. However, it is important that the value set is taken into account when calculating the **On resistance** parameter.
- **On resistance.** This is set using the datasheet information that the forward voltage is 0.7V when the current is 5mA. The voltage to be dropped by the **On resistance** parameter is 0.7V minus the **Forward voltage** parameter; that is 0.1V. Hence the **On resistance** is $0.1\text{V} / 5\text{mA} = 20 \Omega$.
- **Off conductance.** This is set using the datasheet information on reverse current. The reverse current is $3\mu\text{A}$ for a reverse voltage of 1V. Hence the **Off conductance** should be set to $3\mu\text{A} / 1\text{V} = 3\text{e-}6 \text{ S}$.
- **Reverse breakdown voltage.** This parameter should be set to the datasheet working voltage parameter, 4.3V.

- Zener resistance.** This needs to be set to a suitable small number. Too small, and the voltage-current relationship becomes very steep, and simulation convergence may not be as efficient. Too large, and the zener voltage will be incorrect. For the Diode block to be representative of the real device, the simulated reverse voltage should be close to 4.3V at 5mA (the reverse bias current provided by the circuit). Allowing a 0.01 V error on the zener voltage at 5mA, the zener resistance R_z will be $0.01V / 5mA = 2 \Omega$.
- Junction capacitance.** This parameter is set to the datasheet diode capacitance value, 450 pF.







Parameterize an Exponential Diode from a Datasheet

Example 1 on page 3-19 uses a piecewise linear approximation to the diode's exponential current-voltage relationship. This results in more efficient simulation, but requires some thought to go into the setting of block parameter values. An alternative is to use a more complex model that is valid for a wider range of voltage and current values. This example uses the `Exponential` parameterization option of the Diode block.

This model either requires two data points from the diode current-voltage relationship, or values for the underlying equation coefficients, namely the saturation current I_S and the emission coefficient N . The BZX384-B4V3 datasheet only provides values for the former case. Some datasheets do not give the necessary data for either case, and you must follow the processes in Example 1 on page 3-19 or Example 3 on page 3-27 instead.

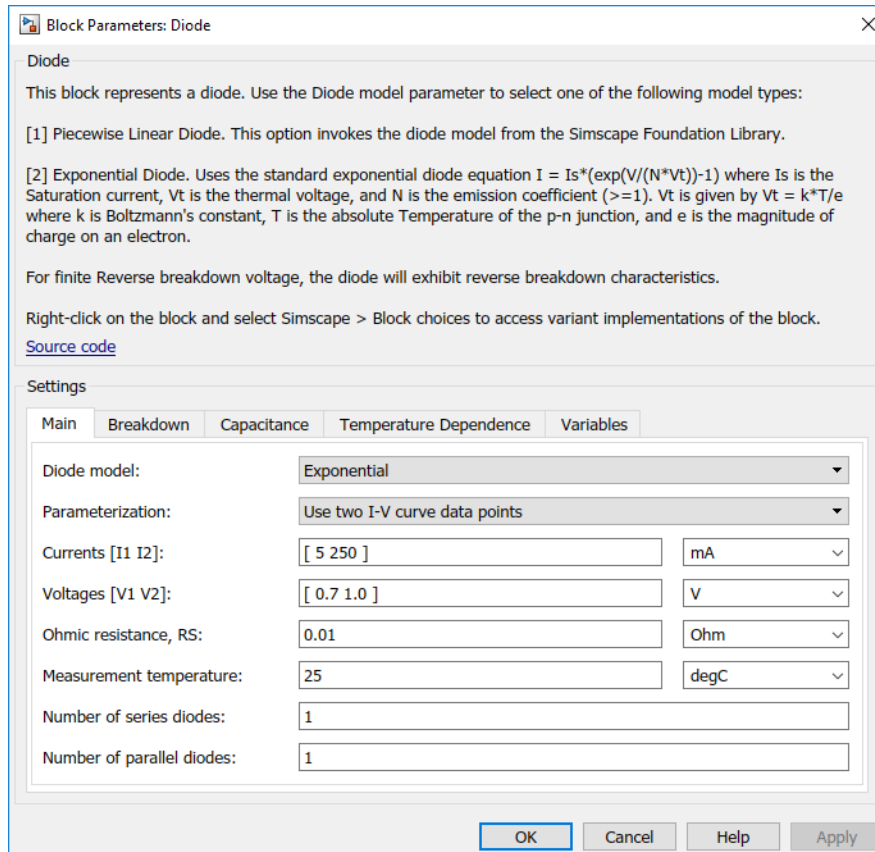
The two data points in the table below are from the BZX384-B4V3 datasheet current-voltage curve:

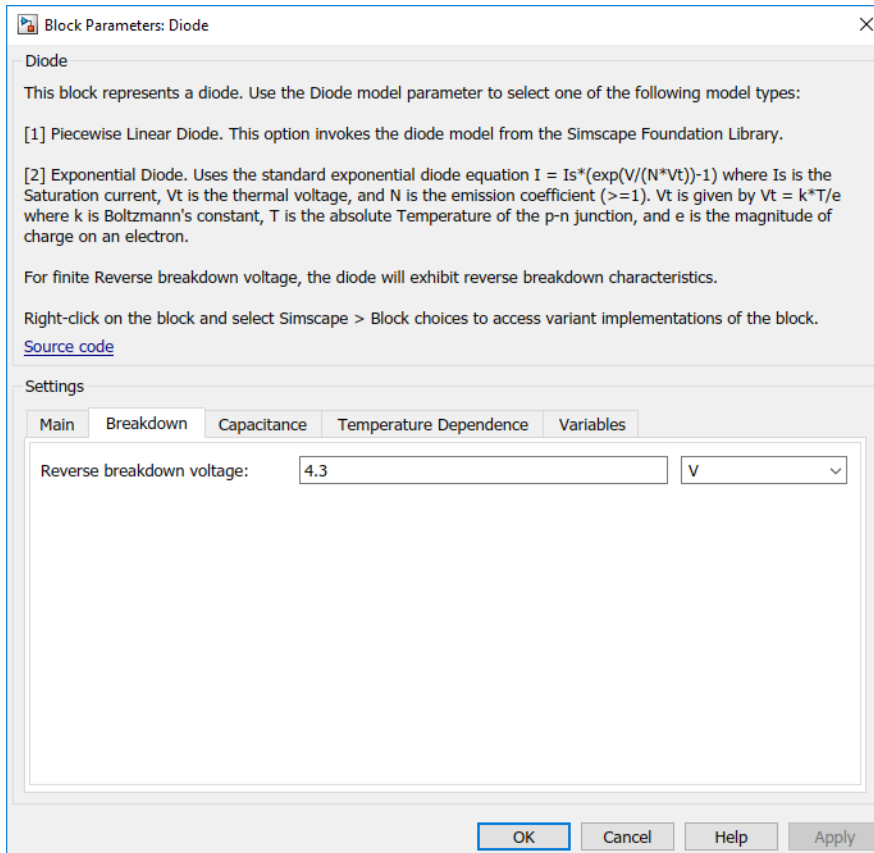
Diode forward voltage, V_F	0.7V	1V
Diode forward current, I_F	5mA	250mA

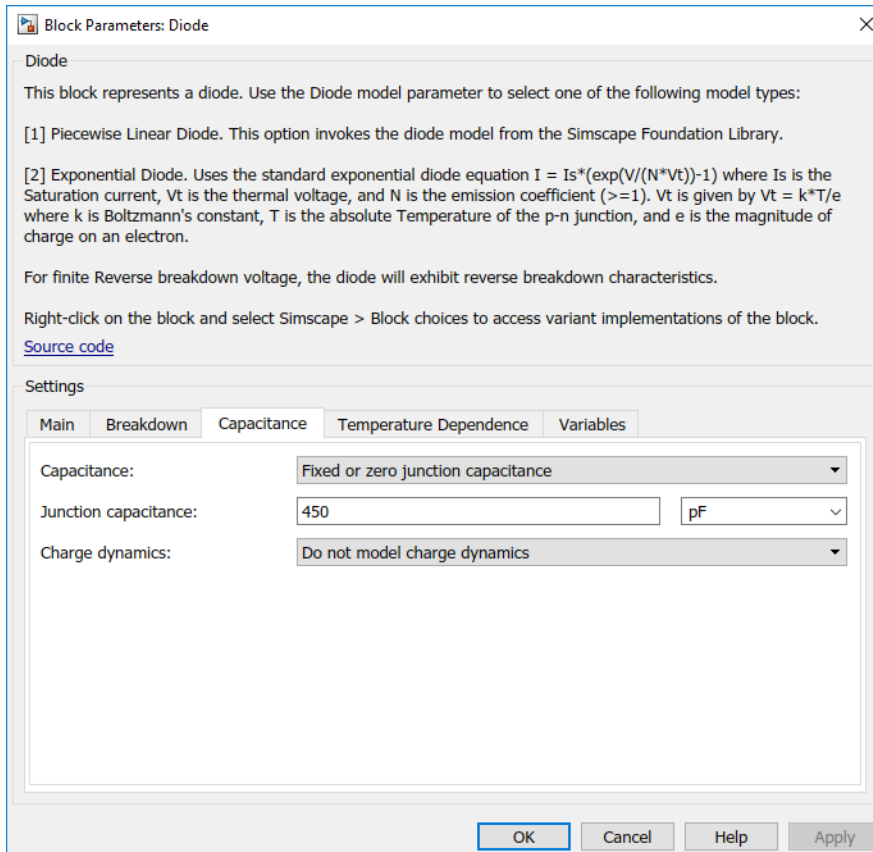
Set the exponential diode block parameters as follows:

- **Currents [I1 I2]**. Set to [5 250] mA.
- **Voltages [V1 V2]**. Set to [0.7 1.0] V.
- **Reverse breakdown voltage**. Set to the datasheet working voltage value, 4.3V.
- **Ohmic resistance, RS**. Set to 0.01 Ω . This is an example of a parameter that cannot be determined from the datasheet. However, setting its value to zero is not necessarily a good idea, because a small value can help simulation convergence for some circuit topologies. Physically, this term will not be zero because of the connection resistances.
- **Junction capacitance**. Set to the datasheet diode capacitance value, 450 pF.

A more complex capacitance model is also available for the Diode component with the exponential equation option. However, the datasheet does not provide the necessary data. Moreover, the operation of this circuit is not sufficiently sensitive to voltage-dependent capacitance effects to warrant the extra detail.







Parameterize an Exponential Diode from SPICE Netlist

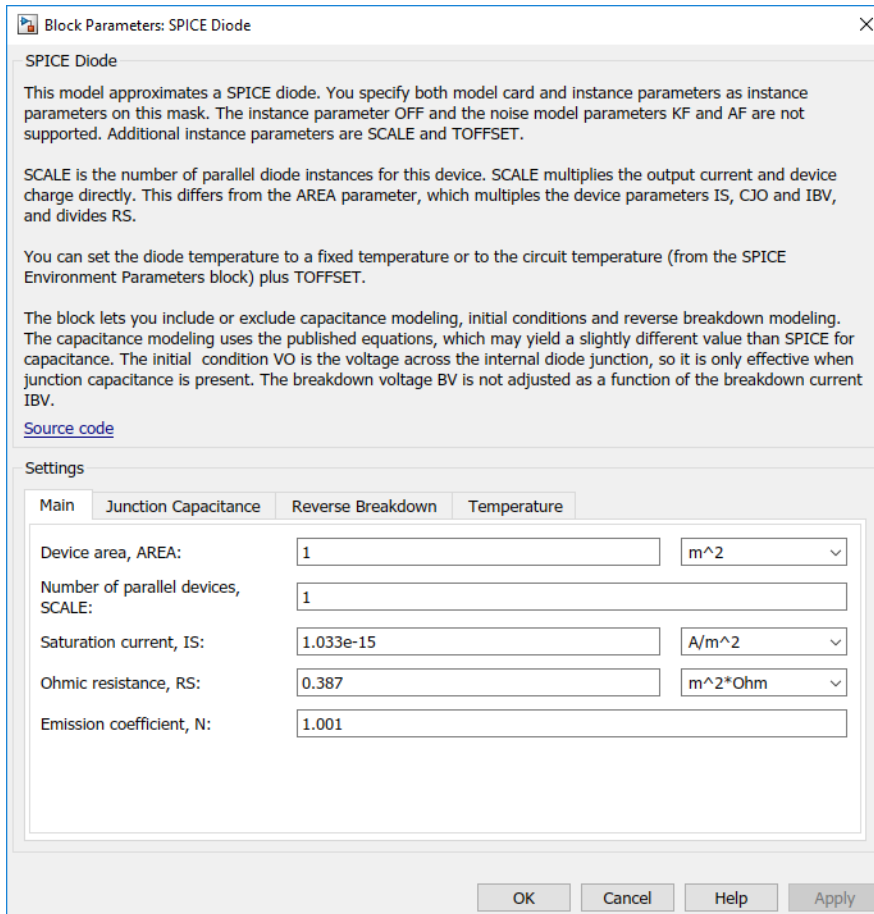
If a datasheet does not provide all of the data required by the component model, another source is a SPICE netlist for the component. Components are defined by a particular type of SPICE netlist called a subcircuit. The subcircuit defines the coefficients for the defining equations. Most component manufacturers make subcircuits available on their websites. The format is ASCII, and you can directly read off the parameters. The BZX384-B4V3 subcircuit can be obtained from Philips Semiconductors.

The subcircuit data can be used to parameterize the Simscape Electrical Diode block either in conjunction with the datasheet, or on its own. For example, the Ohmic resistance is defined in the subcircuit as $RS = 0.387$, thus providing the missing piece of information in Example 2 on page 3-23.

An alternative workflow is to use the Simscape Electrical Additional Components/SPICE Semiconductors sublibrary. The SPICE Diode block in this sublibrary can be directly parameterized from the subcircuit by setting:

- **Saturation current, IS** to $1.033e-15$
- **Ohmic resistance, RS** to 0.387
- **Emission coefficient, N** to 1.001
- **Zero-bias junction capacitance, CJO** to $2.715e-10$
- **Junction potential, VJ** to 0.7721
- **Grading coefficient, M** to 0.3557
- **Capacitance coefficient, FC** to 0.5
- **Reverse breakdown current, IBV** to 0.005
- **Reverse breakdown voltage, BV** to 4.3

Note that where there is a one-to-one correspondence between subcircuit parameters and datasheet values, the numbers often differ. One reason for this is that datasheet values are sometimes given for maximum values, whereas subcircuit values are normally for nominal values. In this example, the CJO value of 271.5 pF differs from the datasheet capacitance of 450 pF at zero bias for this reason.



Block Parameters: SPICE Diode [X]

SPICE Diode

This model approximates a SPICE diode. You specify both model card and instance parameters as instance parameters on this mask. The instance parameter OFF and the noise model parameters KF and AF are not supported. Additional instance parameters are SCALE and TOFFSET.

SCALE is the number of parallel diode instances for this device. SCALE multiplies the output current and device charge directly. This differs from the AREA parameter, which multiplies the device parameters IS, CJO and IBV, and divides RS.

You can set the diode temperature to a fixed temperature or to the circuit temperature (from the SPICE Environment Parameters block) plus TOFFSET.

The block lets you include or exclude capacitance modeling, initial conditions and reverse breakdown modeling. The capacitance modeling uses the published equations, which may yield a slightly different value than SPICE for capacitance. The initial condition VO is the voltage across the internal diode junction, so it is only effective when junction capacitance is present. The breakdown voltage BV is not adjusted as a function of the breakdown current IBV.

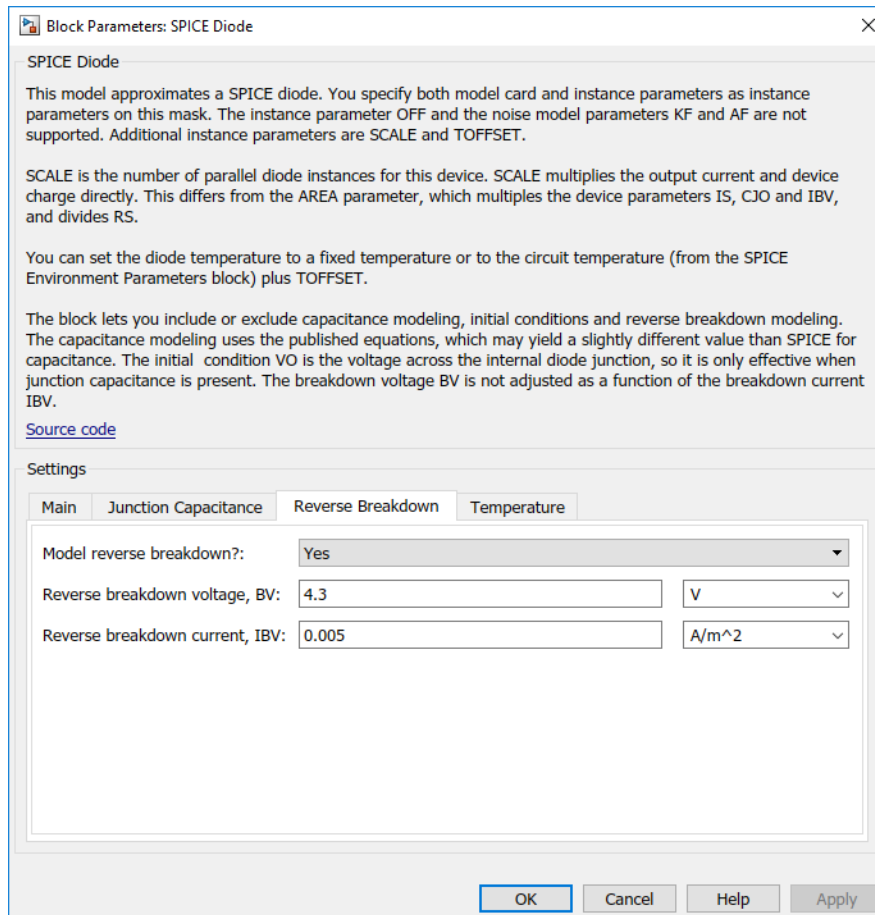
[Source code](#)

Settings

Main | **Junction Capacitance** | Reverse Breakdown | Temperature

Model junction capacitance?:	Yes	
Zero-bias junction capacitance, CJO:	2.715e-10	F/m ²
Junction potential, VJ:	0.7721	V
Grading coefficient, M:	0.3557	
Capacitance coefficient, FC:	0.5	
Transit time, TT:	0	s
Specify initial condition?:	No	

OK Cancel Help Apply



Parameterize an Op-Amp from a Datasheet

The Triangle Wave Generator example model, also described in “Triangle Wave Generator Model” on page 2-21, contains two op-amps, parameterized based on a datasheet for an LM7301. The National Semiconductor datasheet gives the following data for this device:

Gain	97dB = 7.1e4
Input resistance	39M Ω
Slew rate	1.25V/ μ s
Bandwidth	4MHz

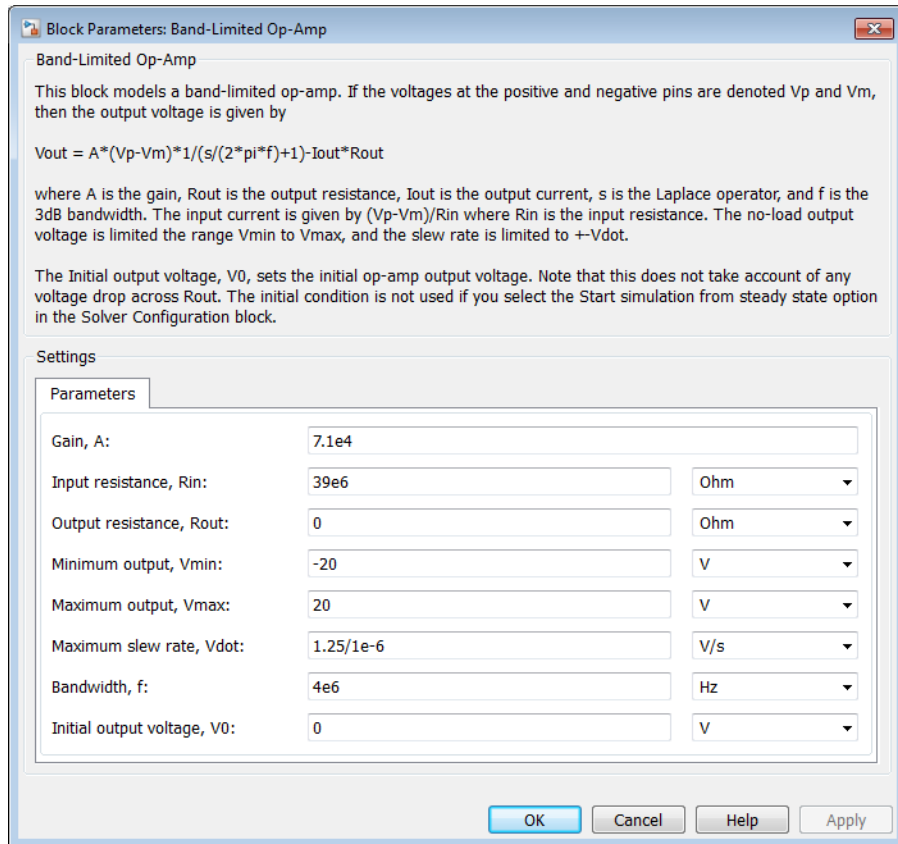
The Band-Limited Op-Amp and Finite-Gain Op-Amp blocks have been designed to work from manufacturer datasheets. Implementing detailed op-amp device models, derived from manufacturer SPICE netlist models, is not recommended, because it provides more accuracy than is typically warranted and slows down simulations. The simple parameterization of the Simscape Electrical op-amp blocks allows you to determine the sensitivity of your circuit to abstracted performance values, such as maximum slew rate and bandwidth. Because of this behavior-based parameterization, you can determine which specification of op-amp is required for a given application. A circuit designer can later match these behavioral parameters, determined from the model, against specific op-amp devices.

Based on the datasheet values above, set the Band-Limited Op-Amp block parameters as follows:

- **Gain** set to 7.1e4
- **Input resistance, Rin** set to 39e6 Ω
- **Output resistance, Rout** set to zero. The value is not defined, but will be small compared to the 1000 Ω load seen by the op-amp.
- **Minimum output, Vmin** set to the negative supply voltage, -20V in this model
- **Maximum output, Vmax** set to the positive supply voltage, 20V in this model
- **Maximum slew rate, Vdot** set to 1.25/1e-6 V/s
- **Bandwidth, f** set to 4e6 Hz

Note that these parameters correspond to the values for +-5 volt operation. The datasheet also gives values for +-2.2V and +-30V operation. It is usually better to pick values for a supply voltage below what your circuit uses, because performance is worse at

lower voltages; for example, the gain is less, and the input impedance is less. You can use the variation in op-amp parameters with supply voltage to suggest a typical range of parameter values for which you should check the operation of your circuit.



Additional Parameterization Workflows

In this section...

“Validation Using Data from SPICE Tool” on page 3-33

“Parameter Tuning Against External Data” on page 3-33

“Building an Equivalent Model of a SPICE Netlist” on page 3-33

Validation Using Data from SPICE Tool

You can validate a parameterized Simscape Electrical component by comparing its behavior to the data from a specialist circuit simulation tool that uses a manufacturer SPICE netlist. Make sure to create a test harness for the component that validates the data across relevant operating points and frequencies.

Parameter Tuning Against External Data

If you have lab measurements of the device, or data from another simulation environment, you can use this to tune the parameters of the equivalent Simscape Electrical component. For an example of parameter tuning, see “Solar Cell Parameter Extraction From Data”.

Building an Equivalent Model of a SPICE Netlist

In “Parameterize an Exponential Diode from SPICE Netlist” on page 3-27, parameterization from a SPICE netlist is relatively straightforward because the netlist defines a single device (the diode) plus the corresponding model card (the parameters). Conversely, a netlist for an op-amp may have more than ten devices, plus supporting model cards. In principle, it is possible to build your own equivalent model of a more complex device by using the SPICE-compatible blocks in the **Simscape > Electrical > Additional Components** library. Connect the components together using the information in the netlist. Before embarking on this, make sure that the Additional Components sublibraries contain all the SPICE-compatible component models you need.

If the device models you wish to model are complex (hundreds of components), then cosimulation with an external circuit simulator may be a better approach.

See Also

SPICE Diode

Simulating Thermal Effects in Rotational and Translational Actuators

In this section...

“Using the Thermal Ports” on page 3-35

“Thermal Model for Actuator Blocks” on page 3-37

Using the Thermal Ports

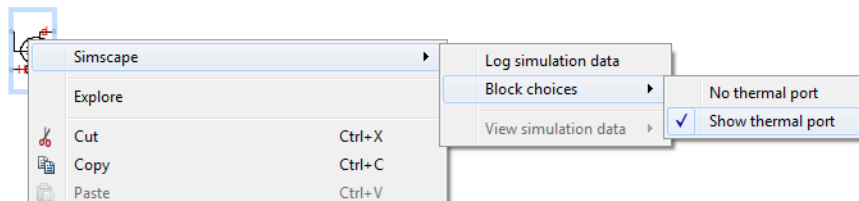
All blocks that represent rotational and translational actuators with electrical windings can optionally show a thermal port for each electrical winding. So, for example:

- A DC Motor block can optionally show a single thermal port corresponding to the armature
- A Shunt Motor block can optionally show two thermal ports, one for the stator winding and one for the field winding

The thermal port represents copper resistance losses which convert electrical power to heat. These losses are sometimes referred to as i^2R losses. The thermal ports do not represent iron losses due to, for example, Eddy currents and hysteresis.

The thermal ports are hidden by default. To expose the thermal port on a particular block instance in your block diagram:

- 1 Right-click the block where you want to show the thermal port.
- 2 From the context menu, select **Simscape > Block choices > Show thermal port**.



When the thermal port is exposed, the block dialog box contains two additional tabs, **Temperature Dependence** and **Thermal Port**. For actuator blocks with single winding, these tabs always contain the same set of parameters.

Settings

Electrical Torque Mechanical Temperature Dependence Thermal Port

Resistance temperature coefficient: 0.00393 1/K

Measurement temperature: 25 degC

- **Resistance temperature coefficient** — Parameter α in the equation defining resistance as a function of temperature, as described in “Thermal Model for Actuator Blocks” on page 3-37. The default value is for copper, and is 0.00393 1/K.
- **Measurement temperature** — The temperature for which motor parameters are defined. The default value is 25 °C.

Settings

Electrical Torque Mechanical Temperature Dependence Thermal Port

Thermal mass: 100 J/K

Initial temperature: 25 degC

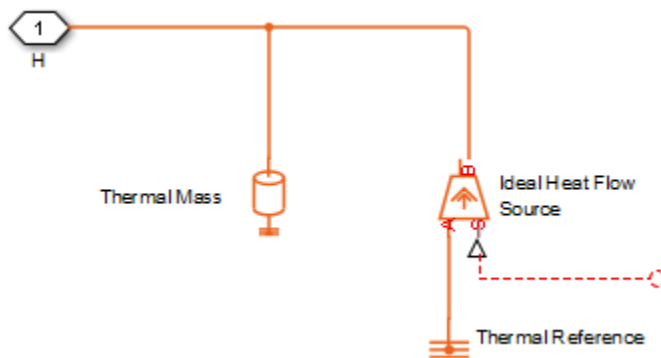
- **Thermal mass** — Thermal mass of the electrical winding, defined as the energy required to raise the temperature by one degree. The default value is 100 J/K.
- **Initial temperature** — The temperature of the thermal port at the start of simulation. The default value is 25 °C.

For more information on selecting the parameter values, see “Thermal Model for Actuator Blocks” on page 3-37.

Parameters for actuator blocks with two windings differ, and are described on the respective block reference pages.

Thermal Model for Actuator Blocks

The following illustration shows the thermal port model used by the actuator blocks. The heat generated by the copper windings is provided as an input to the S physical signal input port of the Ideal Heat Flow Source. The thermal mass represents the lumped thermal mass of the copper winding where thermal mass is defined as the energy required to raise its temperature by one degree. If the mass is denoted M and the specific heat capacity is c_p , then thermal mass is $M \cdot c_p$.



Winding resistance is assumed linearly dependent on temperature, and is given by:

$$R = R_0 (1 + \alpha (T - T_0))$$

where:

- R is the resistance at temperature T .
- R_0 is the resistance at the measurement (or reference) temperature T_0 .
- α is the resistance temperature coefficient. A typical value for copper is 0.00393/K.

Simulating Thermal Effects in Semiconductors

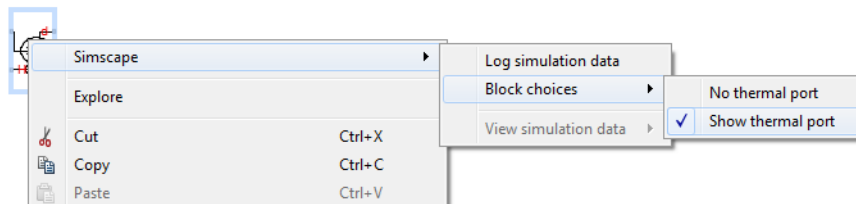
In this section...

- “Using the Thermal Ports” on page 3-38
- “Thermal Model for Semiconductor Blocks” on page 3-40
- “Thermal Mass Parameterization” on page 3-41
- “Electrical Behavior Depending on Temperature” on page 3-41
- “Improving Numerical Performance” on page 3-42

Using the Thermal Ports

Certain Simscape Electrical blocks, for example, the blocks in the Semiconductors & Converters library, contain an optional thermal port. This port is hidden by default. If you want to simulate the generated heat and device temperature, expose the thermal port on a particular block instance in your block diagram:

- 1 Right-click the block where you want to show the thermal port.
- 2 From the context menu, select **Simscape > Block choices > Show thermal port**.



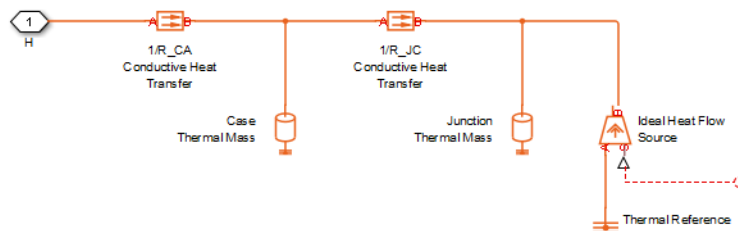
When the thermal port is exposed, the block dialog box contains an additional tab, **Thermal Port**. For semiconductor devices, the tab always contains the same set of parameters.

- Junction case and case-ambient (or case-heatsink) thermal resistances, $[R_{JC} R_{CA}]$** — A row vector $[R_{JC} R_{CA}]$ of two thermal resistance values, represented by the two Conductive Heat Transfer blocks in the “Thermal Model for Semiconductor Blocks” on page 3-40. The first value R_{JC} is the thermal resistance between the junction and case. The second value R_{CA} is the thermal resistance between port H and the device case. See “Thermal Model for Semiconductor Blocks” on page 3-40 for further details. The default value is $[0\ 10]$ K/W.
- Thermal mass parameterization** — Select whether you want to parameterize the thermal masses in terms of thermal time constants (By thermal time constants), or specify the thermal mass values directly (By thermal mass). For more information, see “Thermal Mass Parameterization” on page 3-41. The default is By thermal time constants.
- Junction and case thermal time constants, $[t_J t_C]$** — A row vector $[t_J t_C]$ of two thermal time constant values. The first value t_J is the junction time constant. The second value t_C is the case time constant. This parameter is only visible when you select By thermal time constants for the **Thermal mass parameterization** parameter. The default value is $[0\ 10]$ s.
- Junction and case thermal masses, $[M_J M_C]$** — A row vector $[M_J M_C]$ of two thermal mass values. The first value M_J is the junction thermal mass. The second value M_C is the case thermal mass. This parameter is only visible when you select By thermal mass for the **Thermal mass parameterization** parameter. The default value is $[0\ 1]$ J/K.
- Junction and case initial temperatures, $[T_J T_C]$** — A row vector $[T_J T_C]$ of two temperature values. The first value T_J is the junction initial temperature. The second value T_C is the case initial temperature. The default value is $[25\ 25]$ °C.

For more information on selecting the parameter values, see “Thermal Model for Semiconductor Blocks” on page 3-40 and “Improving Numerical Performance” on page 3-42. For explanation of the relationship between the **Thermal Port** and **Temperature Dependence** tabs in a block dialog box, see “Electrical Behavior Depending on Temperature” on page 3-41.

Thermal Model for Semiconductor Blocks

All blocks with optional thermal ports include an internal thermal model with thermal masses and resistances. The purpose of including this model internally is to keep your diagram uncluttered by the thermal model. The following figure shows an equivalent model of the internal thermal model for semiconductor devices.



The port H in the diagram corresponds to the thermal port H of the block. The two Thermal Mass blocks represent the thermal mass of the device case and the thermal mass of the semiconductor junction, respectively. The Controlled Heat Flow Rate Source block (called Ideal Heat Flow Source in the diagram) inputs heat to the model with value equal to the electrically generated heat from the device.

The two Conductive Heat Transfer blocks model the thermal resistances. Resistance R_{JC} (conductance $1/R_{JC}$) represents the thermal resistance between junction and case. Because of this resistance, under normal conditions the junction will be hotter than the case. Resistance R_{CA} represents the thermal resistance between port H and the device case. If the device has no heatsink, then in your model you should connect port H to an Ideal Temperature Source with its temperature set to ambient conditions. If your device does have an external heatsink, then you must model the heatsink externally to the device, and connect the heatsink thermal mass directly to port H.

If you wish to keep all or part of the thermal model of the device external to the model, you can set the necessary block parameters to zero. The following rules apply:

- Case thermal mass must be greater than zero.
- Junction thermal mass can only be set to zero if the junction-case resistance is also set to zero.
- If both case and junction thermal masses are defined, but junction-case resistance is zero, then the initial temperatures assigned to junction and case must be identical.

Thermal Mass Parameterization

Datasheets usually quote both of the thermal resistances, but rarely give values for thermal masses. There are two parameterization options for the thermal masses:

- By thermal time constants — Parameterize the thermal masses in terms of thermal time constants. This is the default.
- By thermal mass — Specify the thermal mass values directly.

The thermal time constants t_J and t_C are defined as follows:

$$t_J = M_J \cdot R_{JC}$$

$$t_C = M_C \cdot R_{CA}$$

where M_J and M_C are the junction and case thermal masses, respectively, R_{JC} is the thermal resistance between junction and case, and R_{CA} is the thermal resistance between port H and the device case.

You can determine the case time constant by experimental measurement. If data is not available for the junction time constant, you can either omit it and set the junction-case resistance to zero, or you can set the junction time constant to a typical value of one tenth of the case time constant. The alternative is to estimate thermal masses based on device dimensions and averaged material specific heats.

Electrical Behavior Depending on Temperature

For blocks with optional thermal ports, there are two simulation options:

- Simulate the generated heat, device temperature, and the effect of temperature on the electrical equations.
- Simulate the generated heat and device temperature, but do not include effect of temperature on the electrical equations. Use this option when the impact of

temperature on the electrical equations is small over the temperature range to be simulated, or where the primary task of the simulation is to capture the heat generated to support system-level design.

The thermal port and the **Thermal Port** tab of the block dialog box let you simulate the generated heat and device temperature. The **Thermal Dependence** tab of the block dialog box lets you model the effect of temperature of the semiconductor junction on the electrical equations. Therefore:

- To simulate all the temperature effects, show the block's thermal port and set the **Parameterization** parameter on the **Thermal Dependence** tab to **Model temperature dependence** (or, for blocks with a choice of options for modeling temperature dependence, select one of these options, for example, **Use an I-V data point at second measurement temperature**).
- To simulate just the generated heat and device temperature, show the block's thermal port but set the **Parameterization** parameter on the **Thermal Dependence** tab to **None – Simulate at parameter measurement temperature**.

Improving Numerical Performance

It is very important that you set realistic values for thermal masses and resistances. Otherwise, junction temperatures can become extreme, and out of range for valid results, which in turn may manifest itself as numerical difficulties when simulating. A simple test to see if numerical difficulties are a result of unrealistic thermal values is to turn off the temperature dependence for the electrical equations, by setting the **Parameterization** parameter on the **Thermal Dependence** tab to **None – Simulate at parameter measurement temperature**.

The thermal time constants are generally much slower than electrical time constants, so the thermal aspects of your model are unlikely to dictate the maximum fixed time step you can simulate at (for example, for hardware-in-the-loop simulations). However, if you need to remove detail (for example, to speed up simulation), the junction thermal mass time constant is typically an order of magnitude faster than the case thermal mass time constant. You can remove the effect of the junction thermal mass by setting the junction thermal mass to zero and also setting the junction-case thermal resistance to zero.

Simulate Thermal Effects Using Simscape Electrical Thermal Blocks

In this section...

“Prerequisite” on page 3-43

“Thermal Variants” on page 3-43

“Thermal Blocks” on page 3-43

“Thermal Ports” on page 3-44

“Thermal-Modeling Parameters” on page 3-45

“Limitations” on page 3-45

“Model Thermal Losses for a Rectifier” on page 3-46

Prerequisite

This example requires a simulation log variable in your MATLAB workspace. The model in this example is configured to log Simscape data for the whole model for the entire simulation time.

To learn how to determine if a model is configured to log simulation data, see “Examine the Simulation Data Logging Configuration of a Model” on page 7-19.

Thermal Variants

Thermal modeling provides data that helps you to estimate cooling requirements for your system. The nonideal blocks in the Simscape Electrical Semiconductors & Converters library has thermal variants that allow you to determine device temperatures by simulating heat generation. For example, the IGBT (Ideal, Switching) block, which models a three-terminal semiconductor device, has thermal variants that can simulate the heat generated by switching events and conduction losses. Selecting a thermal variant for a block adds a thermal port to the block and enables the associated thermal-modeling parameters.

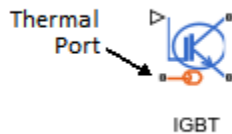
Thermal Blocks

In the Simscape Electrical Passive library, the Thermal sublibrary contains blocks that allow you to model heat transfer using thermal variants:

- Cauer Thermal Model Element — A thermal component that, in a series connection, models heat transfer as a function of the thermal characteristics of the individual physical components and materials, for example, a chip, solder, and base that make up a semiconductor.
- Foster Thermal Model — A thermal component that models heat transfer as a function of the thermal characteristics of a semiconductor.
- Thermal Resistor — A thermal interface resistance component that models conductive heat transfer through a layer of material. Use the Thermal Resistor block to parameterize heat transfer using the thermal resistance value of the material.

Thermal Ports

Thermal ports are physical conserving ports in the Simscape thermal domain. Thermal ports on Simscape Electrical semiconductors are associated with temperature and heat flow. The figure shows a thermal port on a thermal variant of the IGBT (Ideal, Switching) block.



Thermal ports are associated with temperature and heat flow which are the Across and Through variables of the Simscape thermal domain. To measure thermal variables, you can use one or both of these methods:

- 1 Log simulation data using a Simscape logging node. View the data using the `sscexplore` function.
- 2 Add a sensor from the **Simscape > Foundation Library > Thermal > Thermal Sensors** library to your model. To measure temperature, use a parallel-connected Ideal Temperature Sensor block. To measure heat flow, use a series-connected Ideal Heat Flow Sensor block.

There are several advantages to using data logging for desktop simulation. Data logging is less computationally costly than using a sensor block and it allows you to:

- View post-simulation results easily using the Simscape Results Explorer.
- Output data easily to the MATLAB Workspace for post-processing analysis.

However, if you use only data logging to measure a variable, you cannot output a feedback signal for that variable to a control system during simulation as you can when you use only a sensor to measure the variable. Also, because data logging is not supported for code generation, you cannot use Simscape data logging when you perform real-time simulation on target hardware.

Thermal-Modeling Parameters

Thermal-modeling parameters are device-specific characteristics that determine how much heat a block generates during simulation. When you select a thermal variant for a Diode, no additional parameters are enabled because the default variant includes all parameters necessary to model conduction loss. When you select a thermal variant for a three-terminal semiconductor block, additional thermal-modeling parameters are enabled because the default variant does not include parameters necessary to model switching losses.

Three-terminal semiconductors allow you to parameterize thermal losses based on **Voltage and current** or on **Voltage, current, and temperature**. If you parameterize thermal characteristics based only on voltage and current, use scalar values to specify these parameters:

- Output current
- Switch-on loss
- Switch-off loss
- On-state voltage

If you parameterize thermal losses based on **Voltage, current, and temperature**, use vectors to specify the temperature, output current, switching losses, and on-state voltage.

Limitations

Even though simulating thermal losses generates information about the thermal state of a block, thermal dynamics do not affect the electrical behavior of Simscape Electrical blocks during simulation.

Model Thermal Losses for a Rectifier

Model Heat Transfer for a Single Rectifier Diode

To model and measure heat transfer as a function of the thermal characteristics of a semiconductor, connect a Foster model-based thermal network and a temperature sensor to the thermal port on **Diode1**.

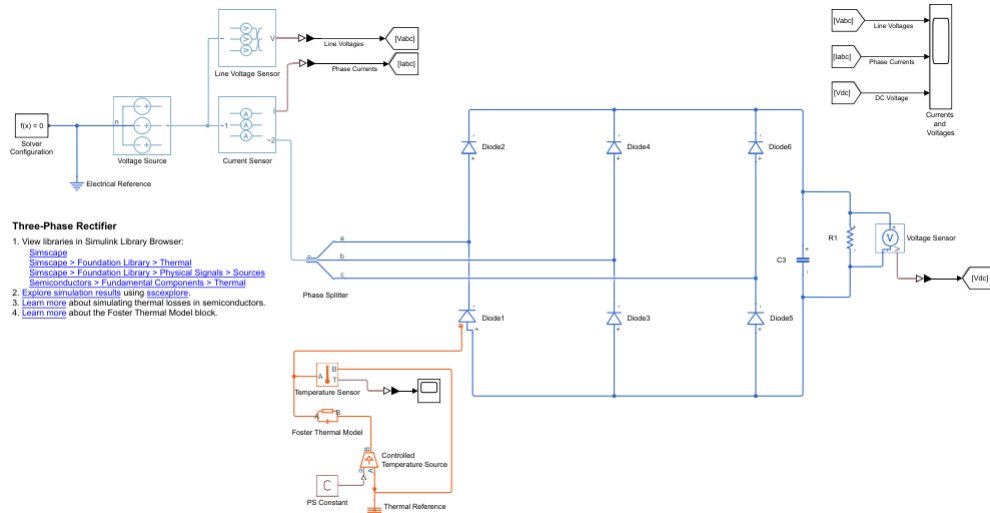
- 1 Open the model, at the MATLAB command prompt, enter

```
ee_rectifier_diodes
```

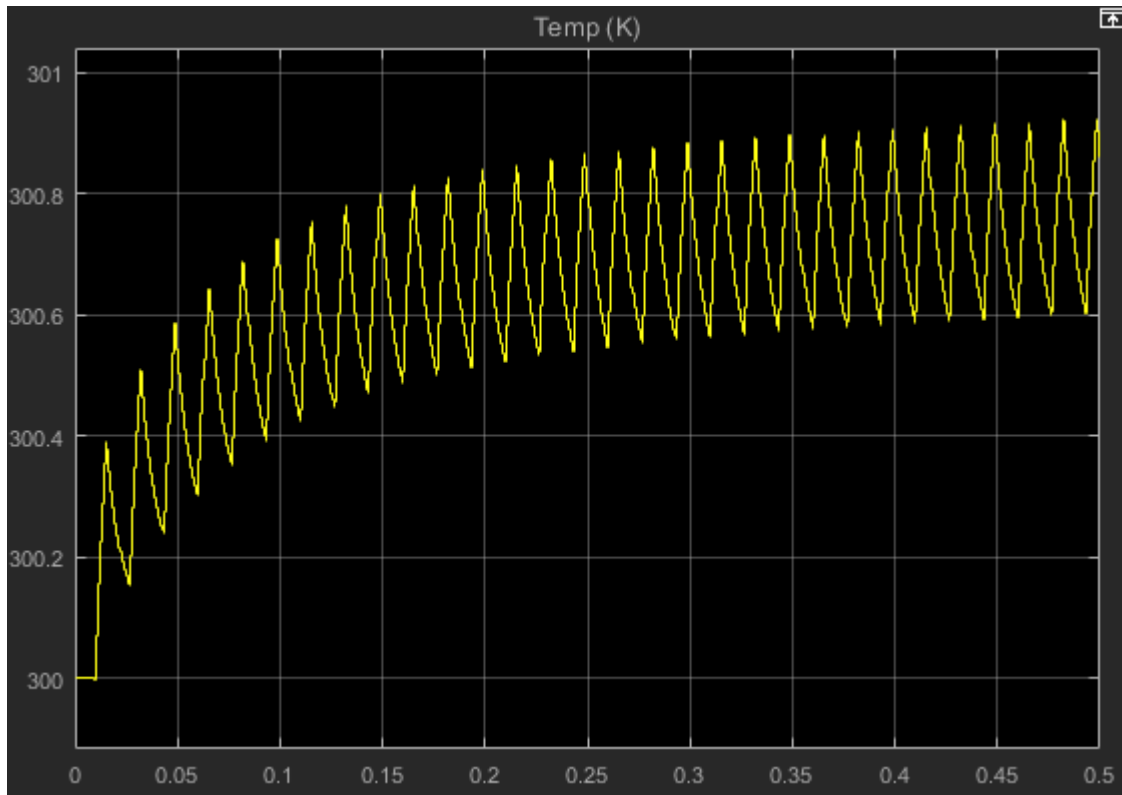
The model contains a three-phase rectifier that includes six Diode blocks.

- 2 Select a thermal variant for the **Diode1** block, right-click the block and, from the context menu, select **Simscape > Block choices**. Select **Show thermal port**.
- 3 Add a Simscape Electrical block that represents heat flow between the diode and the environment. In the model window, the text on the right, contains links that open the Simulink Library browser. Click **Simscape > Electrical > Passive > Thermal** and add a Foster Thermal Model block to the model.
- 4 Modify these Foster Thermal Model block parameters:
 - a **Thermal resistance data** — specify [0.00311 0.008493 0.00252 0.00288] K/W.
 - b **Thermal time constant data** — specify [0.0068 0.0642 0.3209 2.0212] s.
- 5 Represent the ambient temperature as constant using an ideal temperature source.
 - a From the Simulink Library browser, open the **Simscape > Foundation Library > Thermal > Thermal Sources** library and add an Ideal Temperature Source block.
 - b From the **Simscape > Foundation Library > Thermal > Thermal Elements** library, add a Thermal Reference block.
 - c From the **Simscape > Foundation Library > Physical Signals > Sources** library, add a PS Constant block. For the **Constant** parameter, specify a value of 300.
- 6 Measure and display the temperature of **Diode1**:
 - a From the Simulink Library browser, open the **Simscape > Foundation Library > Thermal > Thermal Sensors** library, add an Ideal Temperature Sensor block.

- b Make a copy of one of the PS-Simulink Converter blocks in the model window. For the **Output signal unit** parameter, select K.
 - c From the Simulink Library browser, open the **Simulink** > **Sinks** library and add a Scope block.
- 7 Arrange and connect the blocks as shown in the figure.



- 8 Label the signal from the PS-Simulink Converter block to the Scope block, double-click the line between the blocks and at the prompt, enter Temp (K).
- 9 Simulate the model.
- 10 To see the temperature data, open the Scope block.



The temperature of **Diode1** fluctuates over a temperature range of 0.3 K as it increases from the initial value of 300 K to a settling point of 300.6–300.9 K toward the end of the simulation.

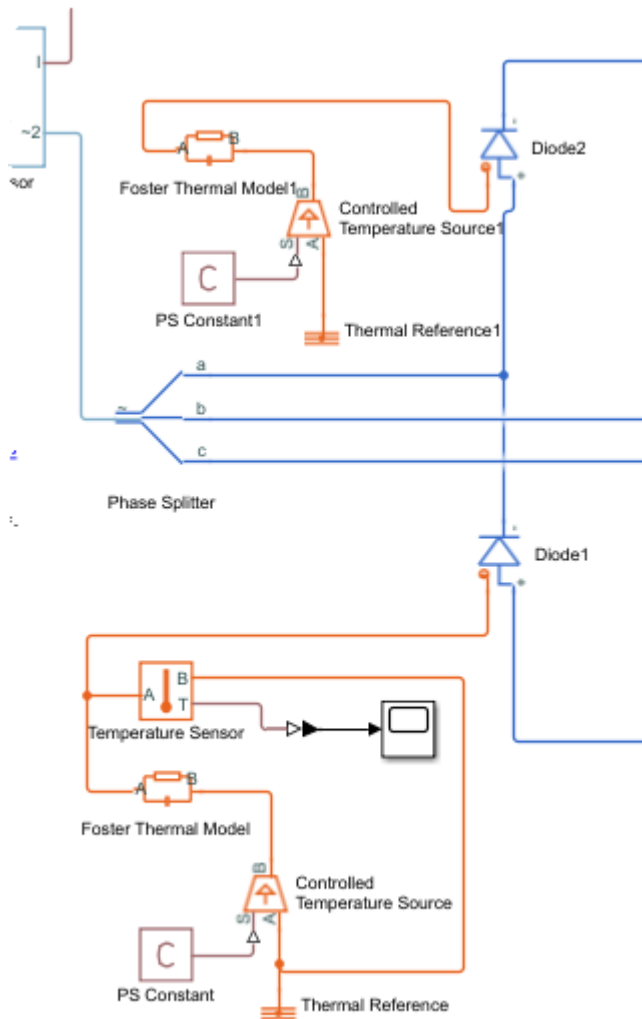
Model Heat Transfer for All Rectifier Diodes

To see the total heat generated by all the semiconductors in the rectifier, use data logging and the Simscape Results Explorer.

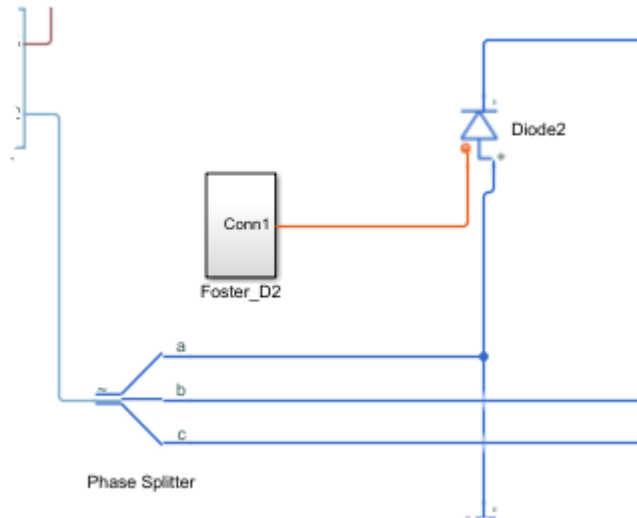
- 1 To enable the thermal ports on all the rectifier diodes, select thermal variants for the **Diode2**, **Diode3**, **Diode4**, **Diode5**, and **Diode6** blocks.
- 2 To measure heat transfer for each diode, create a Foster thermal model subsystem:
 - a Make a copy of this group of blocks:


- Foster Thermal Model
- Ideal Temperature Source
- PS Constant
- Thermal Reference

b Arrange and connect the copied blocks as shown in the figure.

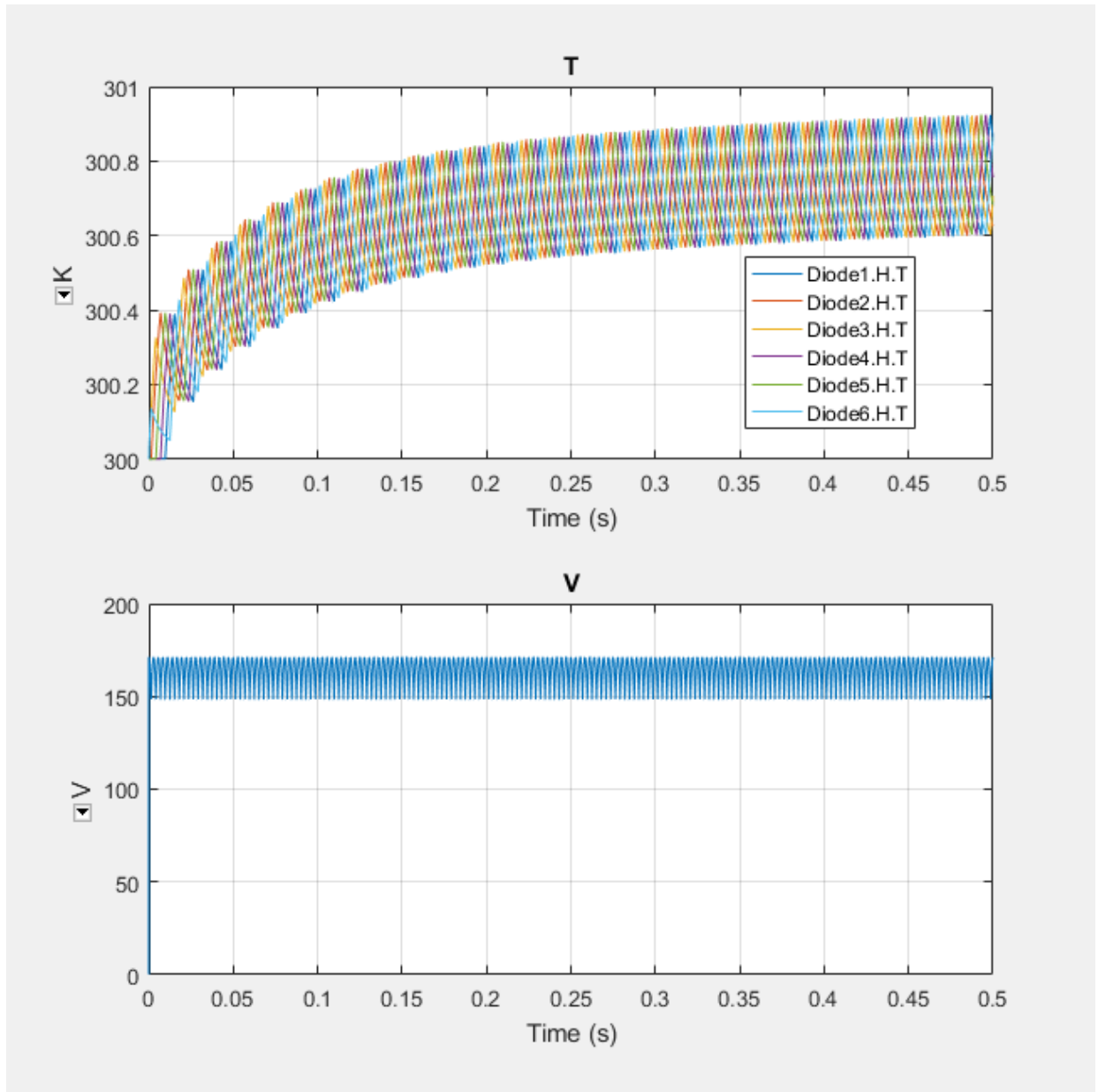


- c Create a subsystem from the copied blocks and rename the subsystem as **Foster_D2**. For information see, “Create a Subsystem” (Simulink).
- d Open the **Foster_D2** subsystem. For the **Conn1** block, for the **Port location on the parent subsystem** parameter, select Right.



- e Make four copies of the **Foster_D2** subsystem. Attach one subsystem to each of the remaining Diode blocks and rename the subsystems as **Foster_D3** through **Foster_D6** to match the **Diode3** through **Diode6** block names.
- 3 Simulate the model.
 - 4 View the results using the Simscape Results Explorer:
 - a In the model window, in the text under **Three-Phase Rectifier**, click **Explore simulation results**.
 - b To display the temperature data for **Diode1**, in the Simscape Results Explorer window, expand the **Diode1 > H** node and click **T**.
 - c To display the DC voltage in a separate plot, expand the **Voltage_Sensor** node and **CTRL**+click **V**.
 - d To display the temperature data for all the diodes, expand the **Diode2 > H** node and **CTRL**+click **T**. Repeat the process for **Diode3** through **Diode6**.
 - e To overlay the temperature data in single plot, in the Simscape Results Explorer window, above the tree-node window, click the options  button. In the Options

dialog box, for **Plot signals**, select **Overlay**. To accept the change, click **OK**. Click and drag the legend down to see the temperature data clearly.



The temperature profile for each diode lags, in succession, behind the temperature profile of **Diode1**. For each diode, the temperature also rises and settles along the same values as the temperature profile for **Diode1**. The data indicate that, because of the lagging behavior of the individual diode temperatures, the temperature of the rectifier rises and settles along the same temperature profile as the diodes, but with less fluctuation.

References

- [1] Schütze, T. *AN2008-03: Thermal equivalent circuit models*. Application Note. V1.0. Germany: Infineon Technologies AG, 2008.

See Also

Cauer Thermal Model Element | Diode | Foster Thermal Model | GTO | IGBT (Ideal, Switching) | MOSFET (Ideal, Switching) | Thermal Resistor | Thyristor (Piecewise Linear)

Related Examples

- “Quantifying IGBT Thermal Losses”

Plot Basic Characteristics for Battery Blocks

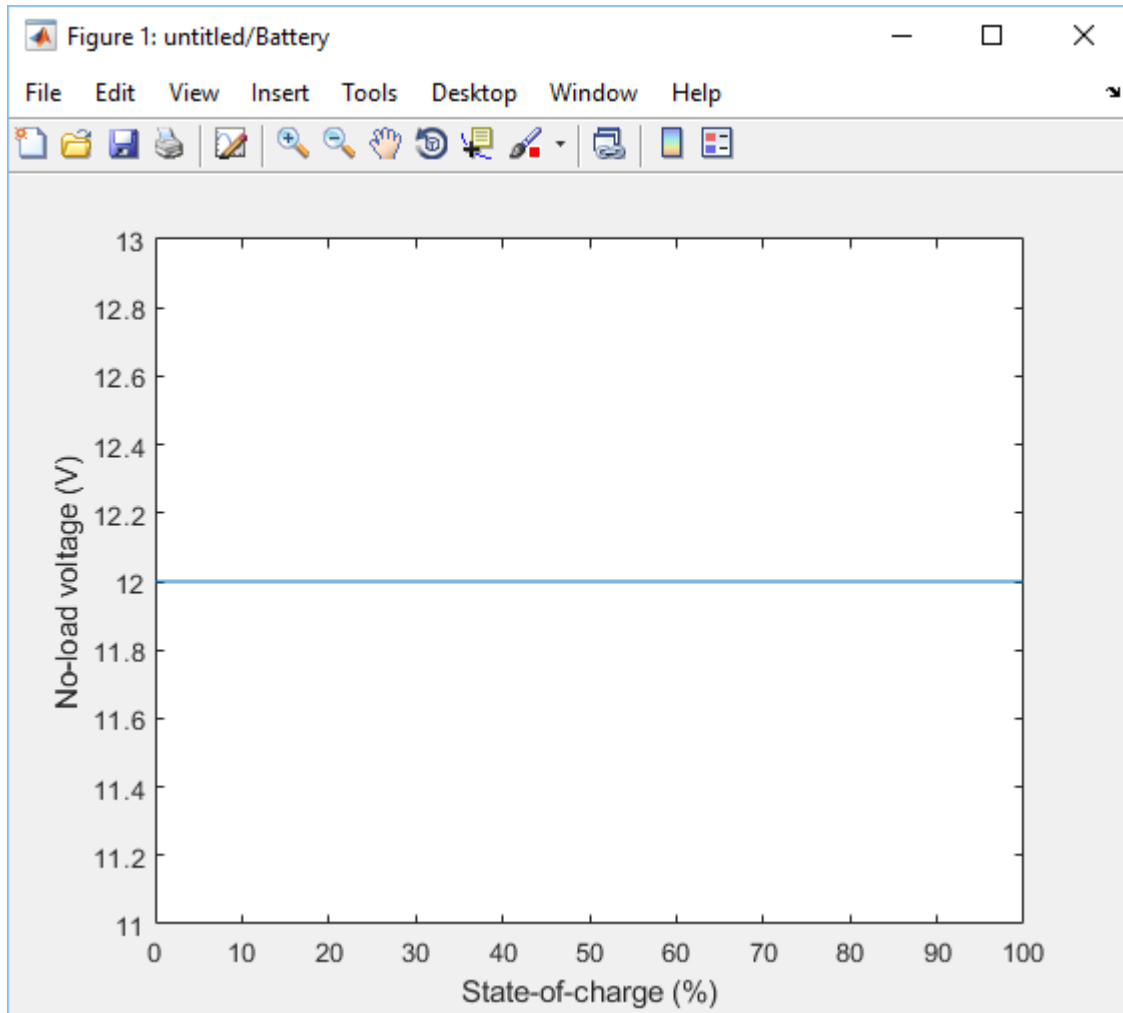
A quick plot feature lets you visualize the voltage-charge characteristic for battery blocks, based on the current block parameter values.

This feature is implemented for Battery and Battery (Table-Based) blocks.

To plot the battery voltage-charge characteristics:

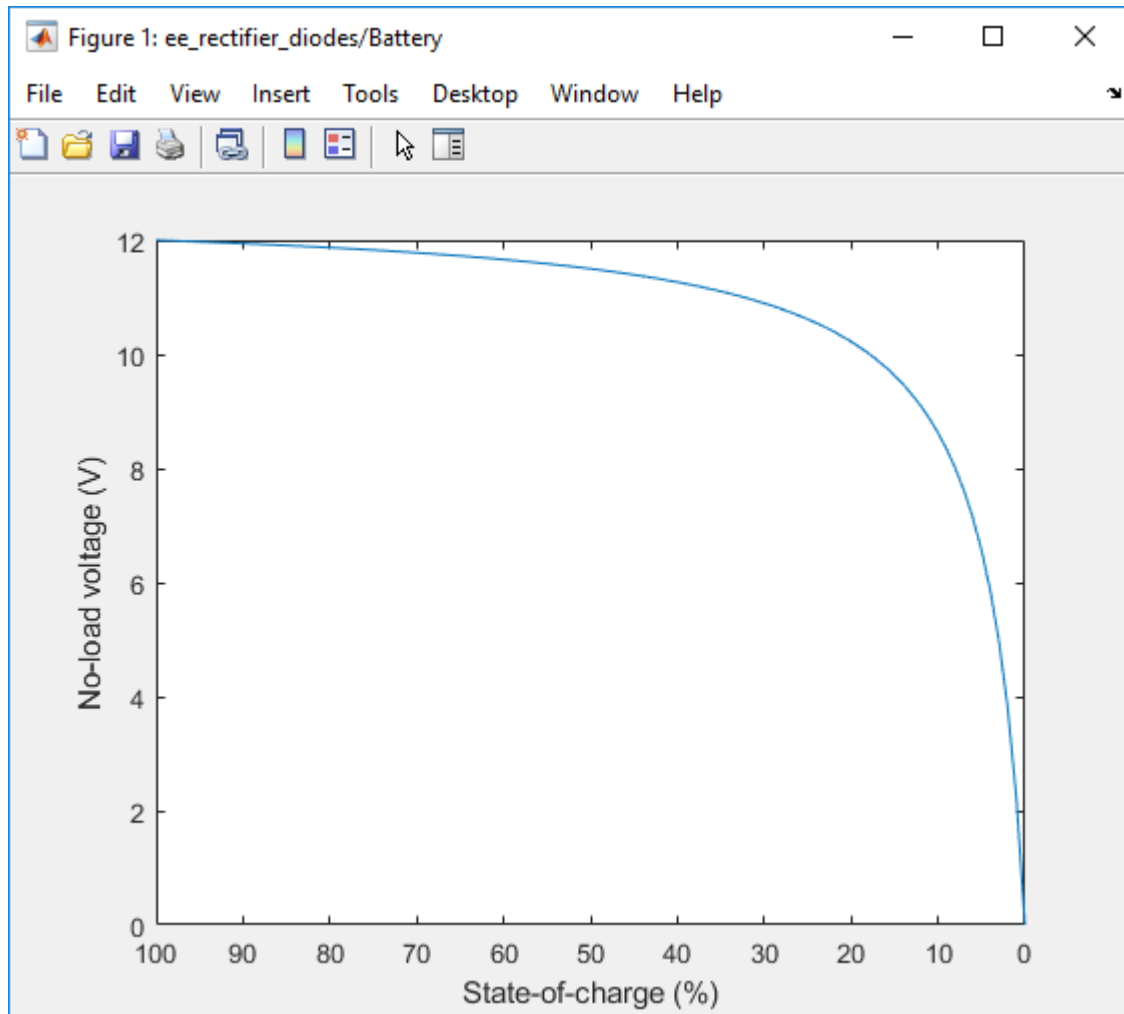
- 1 Right-click a battery block in your model and, from the context menu, select **Electrical > Basic characteristics**. The software automatically computes a set of bias conditions, based on the block parameter values, and opens a figure window containing a plot of no-load voltage versus the state-of-charge (SOC) for the block.

For example, the following plot corresponds to the default parameter values of a Battery block with infinite charge.



- 2 If you change the block parameter values and plot the characteristics again, the plot opens in a new window. This way, you can compare the plots side-by-side and see how the parameter values affect the resulting voltage-charge characteristics for the block.

For example, if you change the **Battery charge capacity** parameter value to Finite and **Self-discharge** to Enabled, the new plot looks like this.



See Also

Battery | Battery (Table-Based)

Plot Basic Characteristics for Semiconductor Blocks

A quick plot feature lets you visualize the basic I-V characteristics for semiconductor switching devices, based on the current block parameter values.

This feature is implemented for nonthermal variants of the following blocks in the Semiconductors library:

- N-Channel IGBT
- N-Channel MOSFET (both threshold-based and surface-potential-based variants)
- P-Channel MOSFET (both threshold-based and surface-potential-based variants)
- N-Channel LDMOS FET
- P-Channel LDMOS FET
- N-Channel JFET
- P-Channel JFET
- NPN Bipolar Transistor
- PNP Bipolar Transistor

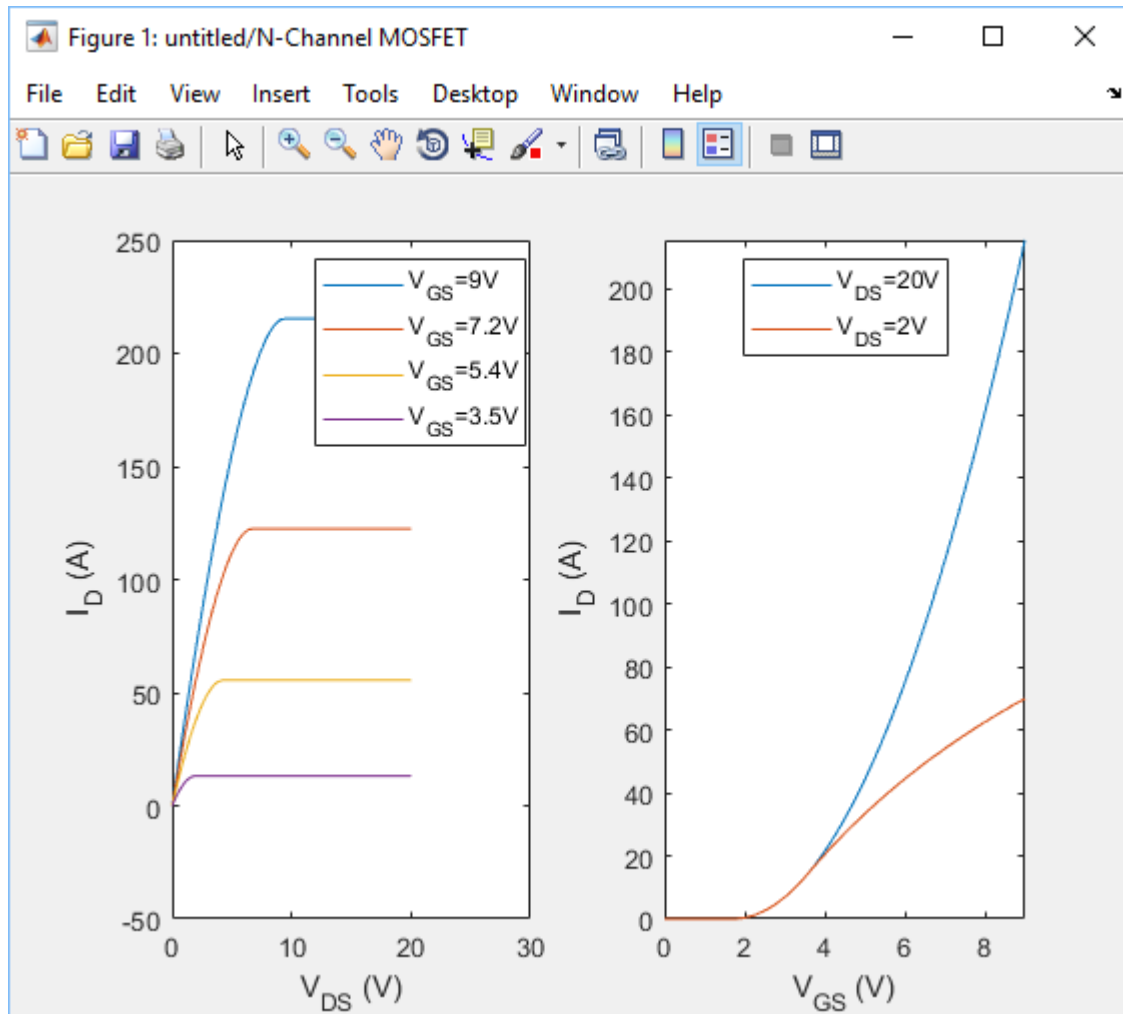
To plot the characteristics, right-click an appropriate semiconductor block in your model and, from the context menu, select **Electrical > Basic characteristics**.

Note For surface-potential-based N-Channel MOSFET and P-Channel MOSFET blocks, the **Electrical > Explore characteristics** option is also available. This option opens the Characteristics Viewer tool, which lets you perform an in-depth study of block characteristics and match the block behavior to a set of target characteristics. For more information, see “MOSFET Characteristics Viewer” on page 3-60.

To plot the basic characteristics:

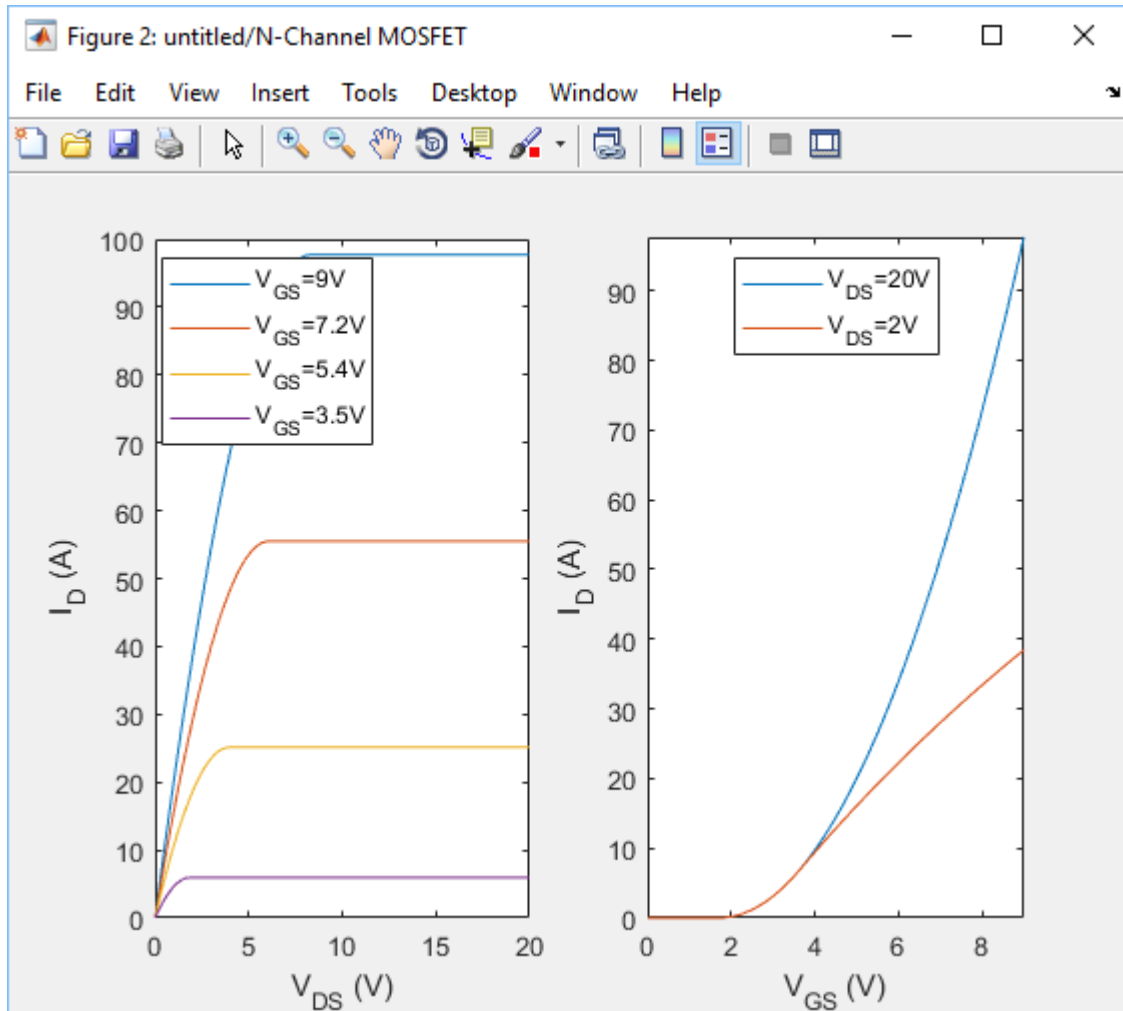
- 1 Right-click a semiconductor block in your model and, from the context menu, select **Electrical > Basic characteristics**. The software automatically computes a set of bias conditions, based on the block parameter values, and opens a figure window containing a plot of the DC I-V characteristics for the block.

For example, the following plot corresponds to the default parameter values of a threshold-based N-Channel MOSFET block.



- 2 If you change the block parameter values and plot the characteristics again, the plot opens in a new window. This way, you can compare the plots side-by-side and see how the parameter values affect the resulting DC I-V characteristics for the block.

For example, if you change the **Gate-source voltage, V_{GS} , for $R_{DS(on)}$** parameter value to 20 V, the new plot looks like this.



See Also

More About

- “MOSFET Characteristics Viewer” on page 3-60

MOSFET Characteristics Viewer

In this section...
“Suggested Workflow” on page 3-60
“Add and Manage Characteristics” on page 3-62
“Choose Parameters and Generate Plots” on page 3-65
“Save the Results” on page 3-67

The Characteristics Viewer tool lets you study characteristics of a particular parameterization of a surface-potential-based MOSFET block and match the block behavior to a set of target characteristics. The tool allows you to:

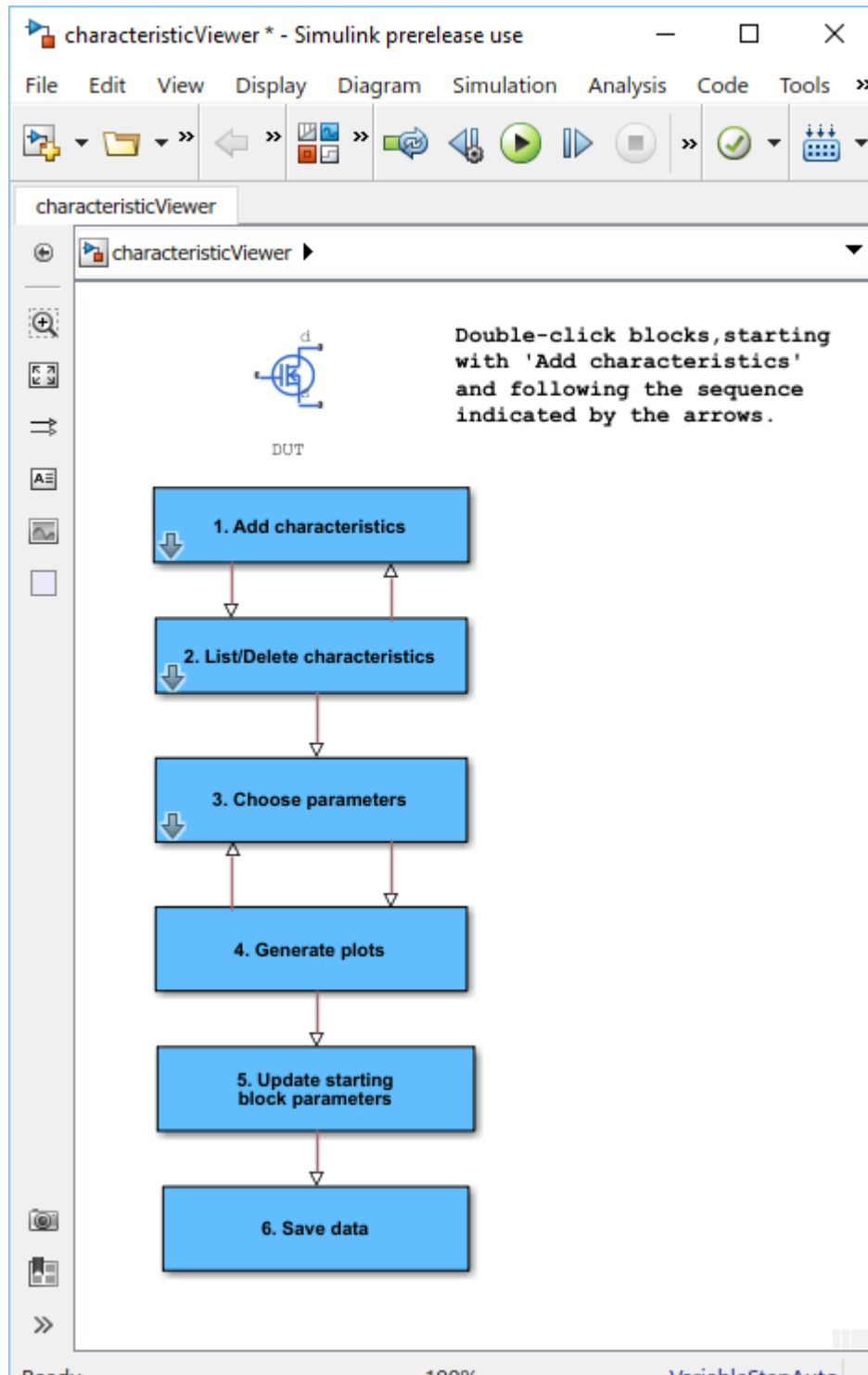
- Plot simulated data, using the current block parameters.
- Overlay simulated data plots over tabulated target data.
- Modify block parameters.
- When satisfied with the results of the parameters tuning in the Characteristics Viewer, update the block parameters in the model.
- Save generated parameter sets for future reuse in a different model.

Suggested Workflow

The Characteristics Viewer tool is available for surface-potential-based N-Channel MOSFET or P-Channel MOSFET blocks only. To switch to a surface-potential-based variant when you add an N-Channel MOSFET or P-Channel MOSFET block from the library, right-click the block in your model and, from the context menu, select **Simscape > Block choices > Surface-potential-based**. Then, when you right-click the block again, the context menu will contain the **Electrical** option, necessary to start the parameterization tool.

To use the MOSFET parameterization tool:

- 1 Right-click a surface-potential-based MOSFET block in your model and, from the context menu, select **Electrical > Explore characteristics**. A characteristicViewer window opens.



- 2 Double-click **Add characteristics**. Specify the characteristics type (target, simulated, or both), and the desired values. Click **Add to plot**.

Continue adding more characteristics, as needed. The **Replace plot** button lets you replace previously added plots. You can also use the **List/Delete characteristics** block, iteratively with **Add characteristics**, to configure your characteristics set.

- 3 Double-click **Choose parameters** and select the parameters of interest.
- 4 Double-click **Generate plots**.
- 5 Iterate between the previous two steps to tune the parameters by matching the simulation results to the target curves.
- 6 When satisfied with the results of the parameters tuning, double-click **Update starting block parameters** to update the block parameters in your model. Until you perform this step, the block in the original model is not affected.
- 7 You can double-click **Save data** to save the generated characteristics as a MAT-file, for future reuse in a different model.

Add and Manage Characteristics

You start the MOSFET parameters tuning process by specifying the desired set of target characteristics:

- 1 In the characteristicViewer window, double-click **Add characteristics**.

The Characteristics window opens.

Block Parameters: Characteristics

Parameters

Plot number

Characteristic type

Sweep type

Sweep values

Step type

Step values

Output type

Output values

- 2 Enter **Plot number**. This number defines the number of the figure that the characteristic will be plotted on. It allows you to add multiple characteristics to the same figure, for overlaying characteristics on top of each other. However, the figure will comprise one xy -axis only.
- 3 Specify the **Characteristic type**:
 - **Target only** — The plot will contain data that you specified, in terms of both input and output values. No simulation will be performed in this case. The data will simply be added to the appropriate plot.
 - **Simulated only** — The plot will contain data that is a result of a simulation over the input bias conditions that you specify.
 - **Target and simulated** — The plot will contain both types of data. This option is useful if you are trying to adjust parameters for the model to fit data that you have extracted from a datasheet.

- 4 Select **Sweep type**, which defines the x -axis variable for the resultant plot:
 - V_{GS} — Sweep over the gate-source voltage.
 - V_{DS} — Sweep over the drain-source voltage.
 - I_D — Sweep over the drain current. Normally, the drain current is not a typical input for a characteristic sweep.
- 5 If the **Characteristic type** is **Simulated only**, specify **Sweep range**. This is a vector of values indicating the range for the swept variable. Only the minimum and maximum values of this vector are utilized by the tool, since the exact sample points for the output data are determined by the variable-step simulation.
- 6 If the **Characteristic type** is **Target only** or **Target and simulated**, specify **Sweep values**. This is a vector of values for the swept variable at which the output is sampled for the target data. As an example, for an I_D - V_{DS} characteristic extracted from a datasheet, the vector would contain the V_{DS} values corresponding to the sampled I_D values in the target curve.
- 7 Select **Step type** to define the second independent input bias condition. The choices are the same as for **Sweep type**. For example, if an I_D - V_{DS} curve is defined as being at a constant V_{GS} , choose V_{GS} for **Step type**.
- 8 Use **Step values** to specify the values for the stepped variable. For example, if an I_D - V_{DS} curve is desired for V_{GS} values of 0 and 10V, set **Step type** to V_{GS} and **Step values** to [0 10].
- 9 Select **Output type**, which defines the output measurement for the characteristic. This is the y -axis variable for the resultant plot. The available values are: V_{GS} , V_{DS} , I_D , C_{GG} , C_{GD} , C_{DG} , and C_{DD} . The capacitances C_{GG} , C_{GD} , C_{DG} , and C_{DD} are defined according to their terminals. To relate these quantities to the datasheet parameters of C_{iss} , C_{rss} and C_{oss} , note that $C_{GG} = C_{iss}$, $C_{DD} = C_{oss}$, and $C_{GD} = C_{rss}$ at $V_{GS} = 0$.

V_{GS} is not a good choice as an output for the surface-potential-based MOSFET model. This value is provided in anticipation of using this tool for other device types.

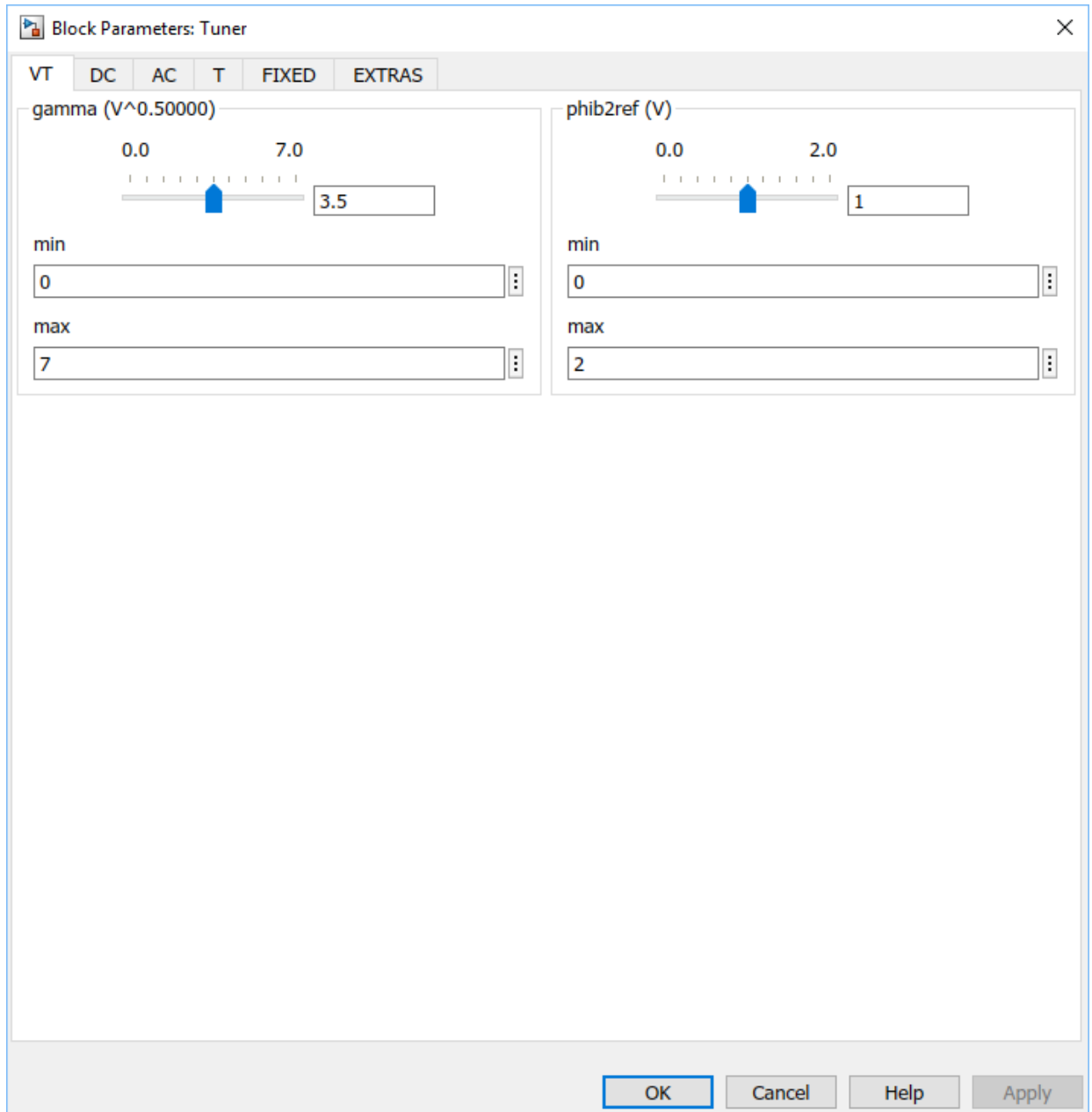
- 10 If the **Characteristic type** is **Target only** or **Target and simulated**, specify **Output values**. This is the target data that you want to plot in the figures. Provide this data as an m -by- n matrix, where m is the size of **Step values** and n is the size of **Sweep values**.
- 11 Click **Add to plot** to add the characteristic specification to the appropriate **Plot number**.
- 12 Continue adding more characteristics, as needed.

The **Replace plot** button lets you replace previously added plots. You can also use the **List/Delete characteristics** block, iteratively with **Add characteristics**, to configure your characteristics set.

Choose Parameters and Generate Plots

After you have specified the desired set of target characteristics, the next step is to define the parameters for the MOSFET block:

- 1 In the characteristicViewer window, double-click **Choose parameters**.



The Tuner window opens. It contains a series of sliders on different tabs, according to which feature of the MOSFET characteristics is most impacted by the specific parameter:

- The **VT** tab displays parameters that primarily impact the threshold voltage (**gamma** and **phib2ref**).
 - The parameters on the **DC** tab primarily affect the DC characteristics.
 - The parameters on the **AC** tab primarily affect the MOSFET dynamics.
 - The parameters on the **T** tab affect temperature scaling.
 - The parameters on the **FIXED** tab are generally fixed at some particular value that is not easy to derive from the displayed characteristics, such as the simulation temperature and the gate resistance (which is often indicated directly on datasheets).
 - The **EXTRAS** tab contains other parameters, which impact the characteristics in ways similar to parameters that already appear on other tabs. For example, **Rsref** (the series resistance associated with the source) operates similarly to **betaref** from the **DC** tab. As a result, it is not always possible to disentangle these two effects.
- 2 Use the sliders on the appropriate tabs of the Tuner dialog.

You can modify the **min** and **max** values, as needed, because they simply define the range over which the various sliders work. These values have no meaning for the underlying model parameters. Changing a **min** or **max** value automatically updates the slider range, without needing to click **OK** or **Apply**.

- 3 After adjusting the sliders, generate the plots to see how close the simulation data is to the target data. In the **charactericViewer** window, double-click **Generate plots**.
- 4 Iterate between tuning the parameters and generating plots until the simulation results match the target curves.

Save the Results

Once you are satisfied with the results of the parameters tuning:

- Double-click **Update starting block parameters** to update the block parameters in your model. Until you perform this step, the block in the original model is not affected.

Note For this step to work, the original model must stay open while you are tuning the parameters.

- You can also double-click **Save data** to save the generated characteristics as a MAT-file, for future reuse in a different model. Specify the file name for saving the data. Inside the file, all the data is saved in an object named `parameterHelper`.

To apply the parameters stored in this object to another MOSFET block, select the MOSFET block in a model and, in the MATLAB Command Window, type:

```
parameterHelper.parameters.updateBlockParameters(gcbh)
```

This command applies the parameter values to the block defined by the handle `gcbh`.

You can also use a string instead of the block handle, for example:

```
parameterHelper.parameters.updateBlockParameters(gcb)
```

To inspect the parameters directly, type `parameterHelper.parameters.values` for the values (stored as character vectors) or `parameterHelper.parameters.names` for the names.

See Also

More About

- Interactive Generation of MOSFET Characteristics
- “Plot Basic Characteristics for Semiconductor Blocks” on page 3-57

Converting a SPICE Netlist to Simscape Blocks

In this section...

“Commands” on page 3-69
“Numeric Suffixes” on page 3-70
“Mathematical Functions” on page 3-71
“Symbols” on page 3-73
“Components” on page 3-73
“Performing Manual Conversions” on page 3-76
“Limitations” on page 3-77

You can convert SPICE components into Simscape equivalents using the SPICE conversion assistant. Often this conversion is automatic. However, because SPICE is a rich language, it is not always possible to perform a full conversion without some manual intervention.

To convert SPICE subcircuits into equivalent Simscape components, follow these steps.

- 1 Use the `subcircuit2ssc` function to generate Simscape language component files from a SPICE netlist file. You can use the optional `subcircuit1,...,subcircuitN` input arguments to specify which subcircuits to convert.
- 2 Make any necessary manual conversions to the generated Simscape component files. To identify the required manual conversions, check the comments at the beginning of the generated Simscape component files. You can use the optional `unsupportedCommands` output argument to generate a `struct` array that lists unsupported SPICE commands for each subcircuit.
- 3 Build the library using `ssc_build` or add individual components to your model using Simscape Component blocks.

There are many different SPICE simulators with variations in syntax and syntax interpretation. The conversion assistant uses the same syntax as Cadence® PSpice and, where such differences exist, complies with PSpice.

Commands

The SPICE conversion assistant supports these commands:

- .FUNC — Reusable function
- .PARAM — Definable parameter
- .MODEL — Set of reusable component parameters
- .SUBCKT — Subcircuit
- .LIB — Directive to include models from an external netlist
- .INC — Directive to include contents of external netlist

The conversion assistant implements .FUNC SPICE commands using Simscape functions. These functions are placed inside a package sublibrary named `+subcircuit_name_simscape_functions`, where `subcircuit_name` is the name of the subcircuit being converted.

Specify the .MODEL syntax for resistors, capacitors, and inductors, as

```
.MODEL <model name> res(r=<value>)  
.MODEL <model name> cap(c=<value>)  
.MODEL <model name> ind(l=<value>)
```

where the `r`, `c`, and `l` values are scaling factors for the value specified on the component declaration. This behavior complies with PSpice, but is not consistent across all simulators.

The conversion assistant does not automatically convert initial conditions specified using the .IC statement. However, you can specify initial conditions for capacitors and inductors using the syntax `IC=<value>`. Also, you can manually convert any .IC statements from the generated Simscape component files.

Because the purpose of the conversion assistant is to help convert SPICE subcircuits into Simscape blocks, simulation commands, such as .TRAN, are ignored.

Numeric Suffixes

The conversion assistant supports these numeric SPICE suffixes:

Suffix	Name	Scale
T	Tera	1e12
G	Giga	1e9
MEG	Mega	1e6
K	Kilo	1e3

Suffix	Name	Scale
M	Milli	1e-3
MIL	--	25.4e-6
U	Micro	1e-6
N	Nano	1e-9
P	Pico	1e-12
F	Femto	1e-15

Mathematical Functions

The conversion assistant supports these basic mathematical functions used in SPICE and MATLAB.

Elementary Math

Name	SPICE Function	MATLAB Function
Absolute value	abs	abs
Smallest element	min	min
Largest element	max	max
Sign function	sgn	sign

Trigonometry

Name	SPICE Function	MATLAB Function
Sine	sin	sin
Inverse sine	asin	asin
Hyperbolic sine	sinh	sinh
Cosine	cos	cos
Inverse cosine	acos	acos
Hyperbolic cosine	cosh	cosh

Name	SPICE Function	MATLAB Function
Tangent	tan	tan
Inverse tangent	atan	atan
Four-quadrant inverse tangent	atan2	atan2
Hyperbolic tangent	tanh	tanh

Exponents and Logarithms

Name	SPICE Function	MATLAB Function
Power	** or pwr	^ or power
Exponential	exp	exp
Natural logarithm	ln or log	log
Base-10 logarithm	log10	log10
Square root	sqrt	sqrt

The conversion assistant interprets `log()` as the natural logarithm rather than the base-10 logarithm. Not all SPICE simulators are consistent in this regard, so ensure that this interpretation is congruent with your SPICE model.

Other

In addition, the conversion assistant supports these SPICE functions:

Name	SPICE Function
If condition	if
Saturation	limit
Current through device	i
Voltage across device	v
Step function	stp

Symbols

The conversion assistant recognizes these SPICE symbols:

- + at the start of a line indicates line continuation from the previous line
- * at the start of a line indicates that the entire line is a comment
- ; within a line indicates the beginning of an inline comment

Components

The notation for SPICE commands in this section follows these rules:

- <argument> refers to a required item in a command line
- <argument>* refers to a required item in a command line that occur one or more times
- [argument] refers to an optional item in a command line
- [argument]* refers to an optional item in a command line that occur zero or more times

This list shows the full set of supported SPICE components, and their supported SPICE netlist notations. You can specify only the .MODEL parameters that differ from SPICE default values.

Sources

- Independent voltage source

```
V<name> <+ node> <- node> [DC] <value>
V<name> <+ node> <- node> exp(<v1> <v2> <td1> <tc1> <td2> <tc2>)
V<name> <+ node> <- node> pulse(<v1> <v2> <td> <tr> <tf> <pw> <per>)
V<name> <+ node> <- node> pwl(<<tj> <vj>>*)
V<name> <+ node> <- node> sffm(<voff> <vampl> <fc> <mod> <fm>)
V<name> <+ node> <- node> sin(<voff> <vampl> <freq> <td> <df>)
```

- Independent current source

```
I<name> <+ node> <- node> [DC] <value>
I<name> <+ node> <- node> exp(<i1> <i2> <td1> <tc1> <td2> <tc2>)
I<name> <+ node> <- node> pulse(<i1> <i2> <td> <tr> <tf> <pw> <per>)
I<name> <+ node> <- node> pwl(<<tj> <ij>>*)
I<name> <+ node> <- node> sffm(<ioff> <iampl> <fc> <mod> <fm>)
I<name> <+ node> <- node> sin(<ioff> <iampl> <freq> <td> <df>)
```

- Current-controlled voltage source

```
H<name> <+ node> <- node> <voltage source name> <gain>
H<name> <+ node> <- node> VALUE={<expression>}
H<name> <+ node> <- node> POLY(<value>) <voltage source name>* <coefficient>*
H<name> <+ node> <- node> TABLE {<expression>}=< <input value>, <output value> >*
H<name> <+ node> <- node> <voltage source name> TABLE=< <input value>, <output value> >*
```

- Voltage-controlled voltage source

```
E<name> <+ node> <- node> <+ control node> <- control node> <gain>
E<name> <+ node> <- node> VALUE={<expression>}
E<name> <+ node> <- node> POLY(<value>) <<+ control node> <- control node>* <coefficient>*
E<name> <+ node> <- node> TABLE {<expression>}=< <input value>, <output value> >*
E<name> <+ node> <- node> <+ control node> <- control node> TABLE=< <input value>, <output value> >*
```

- Current-controlled current source

```
F<name> <+ node> <- node> <voltage source name> <gain>
F<name> <+ node> <- node> VALUE={<expression>}
F<name> <+ node> <- node> POLY(<value>) <voltage source name>* <coefficient>*
F<name> <+ node> <- node> TABLE {<expression>}=< <input value>, <output value> >*
F<name> <+ node> <- node> <voltage source name> TABLE=< <input value>, <output value> >*
```

- Voltage-controlled current source

```
G<name> <+ node> <- node> <+ control node> <- control node> <gain>
G<name> <+ node> <- node> VALUE={<expression>}
G<name> <+ node> <- node> POLY(<value>) <<+ control node> <- control node>* <coefficient>*
G<name> <+ node> <- node> TABLE {<expression>}=< <input value>, <output value> >*
G<name> <+ node> <- node> <+ control node> <- control node> TABLE=< <input value>, <output value> >*
```

- Behavioral source (The <expression> does not need to appear in braces {})

```
B<name> <+ node> <- node> V=<expression>
B<name> <+ node> <- node> I=<expression>
```

Passive Devices

- Resistor

```
R<name> <+ node> <- node> [model name] <value>
.MODEL <model name> res(r=<value>)
```

- Capacitor

```
C<name> <+ node> <- node> [model name] <value> [IC=<value>]
.MODEL <model name> cap(c=<value>)
```

- Inductor

```
L<name> <+ node> <- node> [model name] <value> [IC=<value>]
.MODEL <model name> ind(l=<value>)
```

- Inductor coupling

```
K<name> <inductor name> <inductor name>* <value>
```

Switches

- Voltage-controlled switch

```
S<name> <+ node> <- node> <+ control node> <- control node> <model name>
.MODEL <model name> sw(ron=<value>, roff=<value>, vt=<value>, vh=<value>)
```

- Current-controlled switch

```
W<name> <+ node> <- node> <voltage source name> <model name>
.MODEL <model name> csw(ron=<value>, roff=<value>, it=<value>, ih=<value>)
```

Semiconductor Devices

- Diode

```
D<name> <+ node> <- node> <model name> [area]
.MODEL <model name> d(is=<value>, rs=<value>, n=<value>, cjo=<value>, vj=<value>,
+m=<value>, fc=<value>, tt=<value>, revbrk=<value>, bv=<value>, ibv=<value>,
+xti=<value>, eg=<value>)
```

- Bipolar junction transistor (BJT)

NPN

```
Q<name> <collector node> <base node> <emitter node> [substrate node] <model name> <area>
.MODEL <model name> npn(bf=<value>, br=<value>, cjc=<value>, cje=<value>, cjs=<value>,
+eg=<value>, fc=<value>, ikf=<value>, ikr=<value>, irb=<value>, is=<value>, isc=<value>,
+ise=<value>, itf=<value>, mjc=<value>, mje=<value>, mjs=<value>, nc=<value>, ne=<value>,
+nf=<value>, nr=<value>, rb=<value>, rbm=<value>, rc=<value>, re=<value>, tf=<value>,
+tr=<value>, vaf=<value>, var=<value>, vjc=<value>, vje=<value>, vjs=<value>, vtf=<value>,
+xcjc=<value>, xtb=<value>, xtf=<value>, xti=<value>)
```

PNP

```
Q<name> <collector node> <base node> <emitter node> [substrate node] <model name> <area>
.MODEL <model name> pnp(bf=<value>, br=<value>, cjc=<value>, cje=<value>, cjs=<value>,
+eg=<value>, fc=<value>, ikf=<value>, ikr=<value>, irb=<value>, is=<value>, isc=<value>,
+ise=<value>, itf=<value>, mjc=<value>, mje=<value>, mjs=<value>, nc=<value>, ne=<value>,
+nf=<value>, nr=<value>, rb=<value>, rbm=<value>, rc=<value>, re=<value>, tf=<value>,
+tr=<value>, vaf=<value>, var=<value>, vjc=<value>, vje=<value>, vjs=<value>, vtf=<value>,
+xcjc=<value>, xtb=<value>, xtf=<value>, xti=<value>)
```

- Junction field-effect transistor (JFET)

N-Channel

```
J<name> <drain node> <gate node> <source node> <model name> [area]
.MODEL <model name> njf(beta=<value>, cgd=<value>, cgs=<value>, fc=<value>, is=<value>,
+lambda=<value>, m=<value>, n=<value>, rd=<value>, rs=<value>, vto=<value>, xti=<value>)
```

P-Channel

```
J<name> <drain node> <gate node> <source node> <model name> [area]
.MODEL <model name> pjf(beta=<value>, cgd=<value>, cgs=<value>, fc=<value>, is=<value>,
+lambda=<value>, m=<value>, n=<value>, rd=<value>, rs=<value>, vto=<value>, xti=<value>)
```

- Metal-oxide-semiconductor field-effect transistor (MOSFET)

N-Channel (only level-1 and level-3 are supported)

```
M<name> <drain node> <gate node> <source node> <bulk node> <model name>
+[L=<value>] [W=<value>] [AD=<value>] [AS=<value>] [PD=<value>] [PS=<value>] [NRD=<value>]
+[NRS=<value>] [M=<value>]
.MODEL <model name> nmos(cbd=<value>, cbs=<value>, cgbo=<value>, cgdo=<value>,
+cgso=<value>, cj=<value>, cjsw=<value>, delta=<value>, eta=<value>, fc=<value>,
+gamma=<value>, is=<value>, js=<value>, kappa=<value>, kp=<value>, lambda=<value>,
+ld=<value>, level=<value>, mj=<value>, mjsw=<value>, n=<value>, neff=<value>, nfs=<value>,
+nss=<value>, nsub=<value>, nrd=<value>, nrs=<value>, pb=<value>, phi=<value>, rd=<value>,
+rs=<value>, rsh=<value>, theta=<value>, tox=<value>, tpg=<value>, ucrit=<value>,
+uexp=<value>, uo=<value>, vmax=<value>, vto=<value>, xj=<value>)
```

P-Channel (only level-1 and level-3 are supported)

```
M<name> <drain node> <gate node> <source node> <bulk node> <model name>
+[L=<value>] [W=<value>] [AD=<value>] [AS=<value>] [PD=<value>] [PS=<value>] [NRD=<value>]
+[NRS=<value>] [M=<value>]
.MODEL <model name> pmos(cbd=<value>, cbs=<value>, cgbo=<value>, cgdo=<value>,
+cgso=<value>, cj=<value>, cjsw=<value>, delta=<value>, eta=<value>, fc=<value>,
+gamma=<value>, is=<value>, js=<value>, kappa=<value>, kp=<value>, lambda=<value>,
+ld=<value>, level=<value>, mj=<value>, mjsw=<value>, n=<value>, neff=<value>, nfs=<value>,
+nss=<value>, nsub=<value>, nrd=<value>, nrs=<value>, pb=<value>, phi=<value>, rd=<value>,
+rs=<value>, rsh=<value>, theta=<value>, tox=<value>, tpg=<value>, ucrit=<value>,
+uexp=<value>, uo=<value>, vmax=<value>, vto=<value>, xj=<value>)
```

Subsystems

- Subcircuit

```
X<name> [node]* <subcircuit name> [PARAMS: < <name>=<value> >*]
```

Performing Manual Conversions

After you generate the Simscape component files, inspect each file header for messages regarding unsupported SPICE commands. For example, the conversion assistant does not support the implementation of temperature coefficients for resistors:

```
R1 p n 1k TC=0.01, -0.002
```

The generated Simscape component file contains all the supported conversions, and this header, which identifies the temperature coefficients of the resistor for manual conversion:

```
component test
% test
```

```
% Component automatically generated from a SPICE netlist (11-Dec-2018 09:34:57).
% Users should manually implement the following SPICE commands in order to
% achieve a complete implementation:
%   R1: tc 0.01 -0.002
```

To complete the conversion, modify the Simscape component file to implement the missing components. For more information about performing manual conversions and this particular scenario, see `subcircuit2ssc`.

Limitations

- The netlist must be written in PSpice format and be syntactically correct. The conversion assistant does not check for proper PSpice syntax.
- Only a subset of the PSpice netlist language is supported. However, unsupported PSpice commands are identified at the top of the corresponding Simscape component file to facilitate manual conversion.
- To build generated Simscape components into Simscape blocks, parameter values must conform to Simscape constraints. For example, capacitance of a fundamental capacitor and inductance of a fundamental inductor must be nonzero.

See Also

`ssc_build` | `subcircuit2ssc`

More About

- “Building Custom Block Libraries” (Simscape)
- “Composite Components” (Simscape)

Modeling Machines

- “Machine Parameterization” on page 4-2
- “Per-Unit Conversion for Machine Parameters” on page 4-4
- “Machine Plotting and Display Options” on page 4-6
- “Initialize Synchronous Machines and Controllers” on page 4-8

Machine Parameterization

In Simscape Electrical software, induction machines are parameterized using fundamental parameters. Each synchronous machine is parameterized using standard or fundamental parameters.

Machine fundamental parameters include the values of inductances and resistances of the stator and rotor d - and q -axis equivalent circuits. These parameters fully specify the electrical characteristics of the machine, but you cannot determine them directly from machine test responses. Hence, it is more common to parameterize a synchronous machine using a standard parameter set. You can obtain the standard parameters by observing responses at the machine terminals with suitable tests scenarios.

You can tell the parameter set a block uses because the block name includes the parameter set name, e.g. Induction Machine Squirrel Cage. The parameters you can set in the block dialog box correspond to the parameterization type.

If a machine block has standard and fundamental variants, base your block choice on the parameters you are most familiar with or you have available. Standard block variants use classical equations to convert standard parameter values that you enter to fundamental parameter values for use at run time.

If a machine block has an SI and a per-unit variant, base your block choice on the parameters you have available. For machine blocks that are SI variants, you enter the number of pole pairs and the SI values for the nominal voltage, power, and frequency on the main tab of the dialog box. You also enter SI values for the resistance and reactance parameters on the impedance tab, and for the magnetic flux linkage parameters on the initial condition tab. The block uses classical equations to calculate per-unit base values from the parameters on the main tab. It expresses the resistance, inductance, and magnetic flux linkage parameters as per-unit ratios of the SI values (resistance, reactance, and magnetic flux linkage) and the base values for use at run time.

The field circuit and rotational ports of machine blocks use SI units. However, the pu measurement port of machine blocks outputs a vector of physical signals in per-unit.

See Also

More About

- “Per-Unit System of Units” on page 1-10
- “Per-Unit Conversion for Machine Parameters” on page 4-4

Per-Unit Conversion for Machine Parameters

In this section...
“Impedance Conversion Equations” on page 4-4
“Magnetic Flux Linkage Conversion Equations” on page 4-4

Impedance Conversion Equations

For machine impedance parameters (resistance, inductance, and reactance), the relationships between SI and per-unit values are defined by these equations:

$$R = \frac{R_{(SI)}}{R_{base}}$$

$$L = X = \frac{X_{(SI)}}{X_{base}}$$

where:

- $R_{(SI)}$ is the resistance, expressed in Ω .
- R_{base} is the per-unit base resistance, expressed in Ω .
- R is the per-unit resistance.
- $X_{(SI)}$ is the reactance, expressed in Ω .
- X_{base} is the per-unit base reactance, expressed in Ω .
- X is the per-unit reactance.
- L is the per-unit inductance.

Magnetic Flux Linkage Conversion Equations

For machine magnetic flux linkage parameters, the relationship between SI and per-unit values is defined by

$$\psi = \frac{\psi_{(SI)}}{\psi_{base}}$$

where:

- $\psi_{(SI)}$ is the magnetic flux linkage, expressed in Wb.
- ψ_{base} is the per-unit base magnetic flux linkage, expressed in Wb.
- ψ is the per-unit magnetic flux linkage.

See Also

More About

- “Per-Unit System of Units” on page 1-10

Machine Plotting and Display Options

Use the **Electrical** menu on the block context menu to perform plotting and display actions for certain blocks in the Simscape Electrical/Electromechanical sublibrary. For example, you can plot torque versus speed for the Induction Machine Wound Rotor block, either in SI or per-unit units.

Using other options on the **Electrical** menu, you can display values in per-unit or display base parameter values in the MATLAB Command Window. These options enable you to initialize and tune your three-phase machine quickly.

Asynchronous Machine Options

The context menus of certain asynchronous machine blocks contain some or all of these options:

- **Display Base Values** — Displays the machine per-unit base values in the MATLAB Command Window.
- **Plot Torque Speed (SI)** — Plots torque versus speed, both measured in SI units, in a MATLAB figure window using the present machine parameters.
- **Plot Torque Speed (pu)** — Plots torque versus speed, both measured in per-unit, in a MATLAB figure window using the present machine parameters.
- **Plot Open-Circuit Saturation** — Plots terminal voltage versus no-load stator current, both in per-unit, or, for SI blocks, in V and A, respectively, in a MATLAB figure window. The plot contains three traces:
 - Unsaturated
 - Saturated
 - Derived
- **Plot Saturation Factor** — Plots saturation factor applied to magnetic inductance versus magnetic flux linkage in per-unit, or for SI blocks, in Wb, in a MATLAB figure window.
- **Plot Saturated Inductance** — Plots magnetizing inductance versus per-unit magnetic flux linkage, both in per-unit, or, for SI blocks, in H and Wb, respectively, in a MATLAB figure window.

Induction Machine Options

The context menus of certain induction machine blocks contain some or all of these options for displaying the associated values in the MATLAB Command Window:

- **Display Base Values** — Displays the machine per-unit base values in the MATLAB Command Window
- **Display Associated Base Values** — Displays the associated per-unit base values in the MATLAB Command Window.
- **Display Associated Initial Conditions** — Displays the associated initial condition values in the MATLAB Command Window.
- **Plot Open-Circuit Saturation (pu)** — Plots air-gap voltage, V_{ag} , versus field current, i_{fd} , both measured in per-unit, in a MATLAB figure window. The plot contains three traces:
 - Unsaturated — **Stator d-axis mutual inductance (unsaturated), L_{ad}** you specify
 - Saturated — **Per-unit open-circuit lookup table (Vag versus ifd)** you specify
 - Derived — Open-circuit lookup table (per-unit) derived from the **Per-unit open-circuit lookup table (Vag versus ifd)** you specify. This data is used to calculate the saturation factor, K_s , versus magnetic flux linkage, ψ_{at} , characteristic.
- **Plot Saturation Factor (pu)** — Plots saturation factor, K_s , versus magnetic flux linkage, ψ_{at} , both measured in per-unit, in a MATLAB figure window using the present machine parameters. This value is derived from parameters you specify:
 - **Stator d-axis mutual inductance (unsaturated), L_{ad}**
 - **Per-unit field current saturation data, ifd**
 - **Per-unit air-gap voltage saturation data, Vag**

Machine Inertia Block Options

For the Machine Inertia block, you can display the inertia parameters and base values using the **Electrical** menu on the block context menu. The block displays parameter values in the MATLAB Command Window.

Initialize Synchronous Machines and Controllers

In Simscape Electrical software, you can specify steady-state power and voltage values for a synchronous machine. Based on the values you specify, the machine block calculates the initial field circuit and rotational input values required to achieve this steady state. Starting a machine at steady state prevents undesired transient effects in your simulation.

- 1 Calculate the required power and voltage characteristics of your load circuit.
- 2 In the **Initial Conditions** tab of the dialog box, set **Specify initialization by** to **Electrical power and voltage output**.
- 3 Enter the required power and voltage values and click **OK**.
- 4 Right-click the machine block and select **Electrical > Display Associated Initial Conditions**.

Simscape Electrical Power Systems calculates the field circuit and rotational port values required to start the machine in steady state and displays them in the MATLAB Command Window.

- 5 Use these values to input parameters to the blocks connected to the field circuit and rotational ports of the synchronous machine.

Note If you set **Specify initialization by** to **Mechanical** and **magnetic states**, Simulink does not calculate the associated initial conditions for the machine.

Customization

- “Build Custom Blocks Using the Three-Phase Electrical Domain” on page 5-2
- “Custom Synchronous Machine” on page 5-4

Build Custom Blocks Using the Three-Phase Electrical Domain

Simscape Foundation domains include a three-phase electrical domain. You can use this domain to develop your own custom three-phase blocks using Simscape language. To refer to this domain in your custom component declarations, use the following syntax:

```
foundation.electrical.three_phase
```

Additionally, the ThreePhaseExamples library, included in the Simscape Electrical product examples, contains a Fundamental library, a Transforms library, and the following custom three-phase components:

- Permanent Magnet Synchronous Motor
- Synchronous Machine
- Synchronous Machine (simplified)
- Zigzag Transformer

You can use these simplified example models to write your own custom component files.

To open the custom library, at the MATLAB command prompt, type `ThreePhaseExamples_lib`. Double-click any block in the library to open its dialog box, and then click the **Source code** link in the block dialog box to open the Simscape source file for this block in the MATLAB Editor.

To customize the block for your application, edit the source file and save it under another name.

For example, you can create a folder called `+MyMachines` and save the source files for your customized machines in this folder. Create this folder in your working directory, or in another directory that is on the MATLAB path. Running the `ssc_build` command on this package generates the `MyMachines_lib` library model. This library contains all your custom machine blocks and is located in the same directory where you have created the `+MyMachines` folder. Open the `MyMachines_lib` library by double-clicking it or by typing its name at the MATLAB command prompt.

For more information on packaging and deploying Simscape component files, see “Building Custom Block Libraries” (Simscape).

Things to keep in mind when writing component files:

- If you create a custom component by modifying an existing one, do not forget to change the name of the component and the name of the resulting block.
- The component name must be the same as the name of the Simscape file. For example, if you plan to save your component in a file called `MyComponent.ssc`, change the declaration line in the file:

```
component MyComponent
```

- The comment line immediately following the component declaration (that is, the first line beginning with the % character) defines the name of the block, as it appears in the custom library next to the block icon and at the top of the block dialog box. If you do not specify this comment, then the component name serves as the block name. The block name must be unique within the subpackage (sublibrary) where it resides.
- Additional comments, below the line specifying the block name, are interpreted as the block description. You do not have to modify them when copying an existing file, but if you change the way the component works, it makes sense to reflect the change in the block description. The block description is for informational purposes only.
- When modifying component equations, if you introduce additional terms, make sure to add the appropriate variables or parameters to the component declaration section. For example, if you add zero-sequence dynamics to the component equations, declare an additional parameter for stator zero-sequence inductance, L_0 , and an additional variable for the initial stator zero-sequence magnetic flux linkage.

The “Custom Synchronous Machine” on page 5-4 tutorial shows how you can modify the Synchronous Machine component file and customize it for use in your applications. For more information on writing customized component files, see “Custom Components” (Simscape).

See Also

More About

- “Custom Synchronous Machine” on page 5-4
- “Custom Components” (Simscape)
- “Foundation Domains” (Simscape)

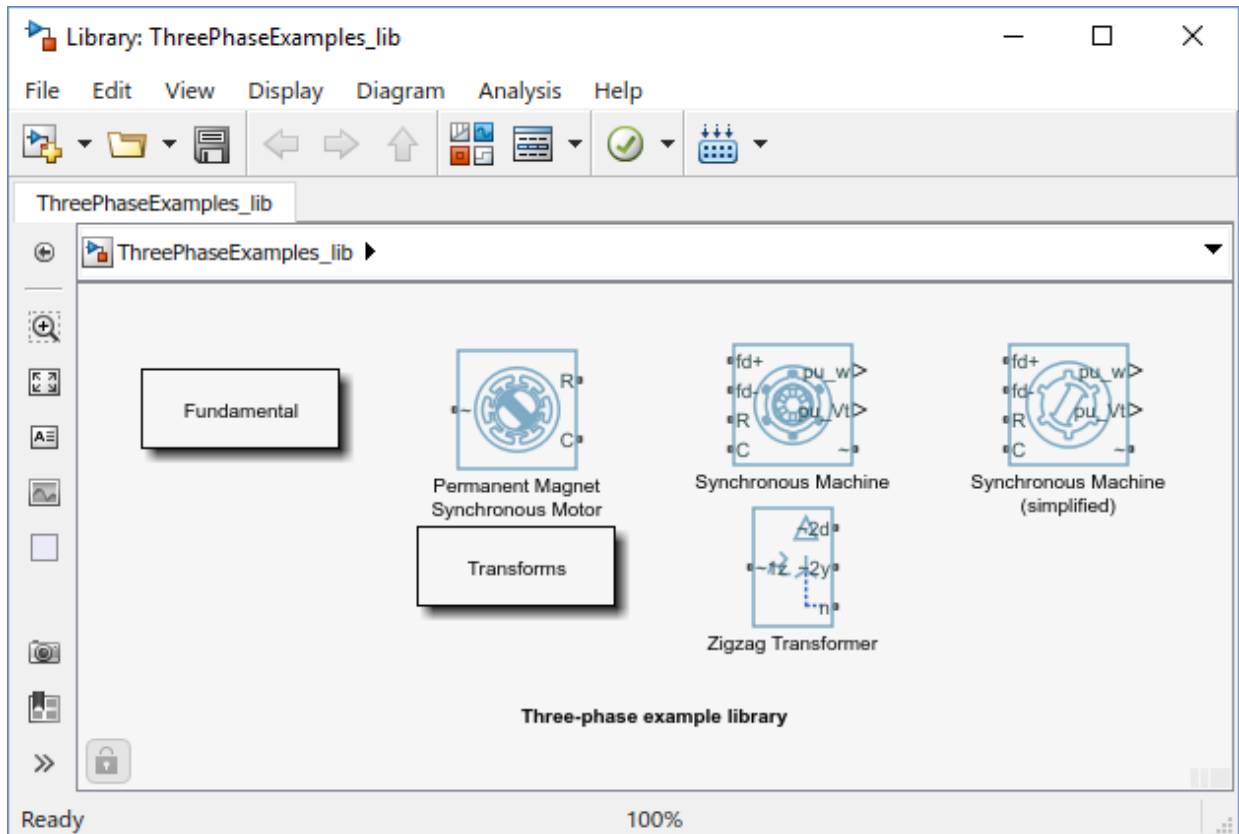
Custom Synchronous Machine

The ThreePhaseExamples library, included in the product examples, contains simplified example models that you can use to write your own machine and transformer component files. The Synchronous Machine component in the ThreePhaseExamples library is similar to the Synchronous Machine Round Rotor block, but its equations have been simplified to omit zero-sequence dynamics. The Synchronous Machine block is therefore suitable for balanced operation only.

This example shows how you can further simplify the component file and make a custom machine block that does not account for the stator rate of change of flux.

- 1** In your working directory, create a folder called `+MyMachines`. This folder will contain the source files for your customized machines.
- 2** To open the library of simplified component examples, at the MATLAB command prompt, type:

```
ThreePhaseExamples_lib
```



- 3 Double-click the Synchronous Machine block.
- 4 In the block dialog box, click the **Source code** link.

The Simscape source file for this block opens in the MATLAB Editor.

- 5 Change the name of the component, the name of the block, and the block description by replacing these lines of the file:

```

component sm
% Synchronous Machine :1.5
% Synchronous machine (SM) with a round rotor parameterized
% using fundamental per-unit parameters. The defining equations are
% simplified by omitting the zero-sequence dynamics: the model is suitable
% for balanced operation.
% The model contains effect of rate of change of magnetic flux linkages
% on stator voltages, effect of speed variation on stator voltages, one
% damper winding on the d-axis and two damper windings on the q-axis.

```

```
% Copyright 2012-2018 The MathWorks, Inc.
```

with:

```
component sm1
% Simplified Synchronous Machine
% This synchronous machine does not include the stator d.psi/dt terms.
```

- 6 To remove the stator rate of change of flux terms, scroll down to the equations section and modify the stator voltage equations from:

```
% Per unit stator voltage equations
pu_ed == oneOverOmega*pu_psid.der - pu_psiq*pu_velocity - Ra*pu_id;
pu_eq == oneOverOmega*pu_psiq.der + pu_psid*pu_velocity - Ra*pu_iq;
```

to:

```
% Per unit stator voltage equations
pu_ed == -pu_psiq*pu_velocity - Ra*pu_id;
pu_eq == pu_psid*pu_velocity - Ra*pu_iq;
```

- 7 Save the file in the +MyMachines folder as sm1.ssc. The name of the Simscape file must match the component name.
- 8 To generate the custom library containing the new block, at the MATLAB command prompt, type:

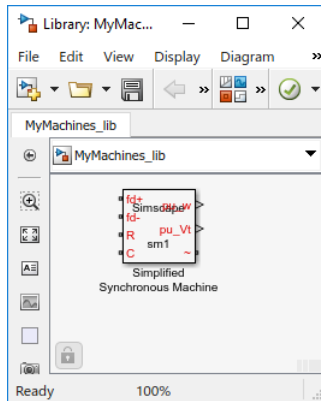
```
ssc_build(MyMachines)
```

This command generates the MyMachines_lib library model in your working directory.

- 9 To open the custom library, at the MATLAB command prompt, type:

```
MyMachines_lib
```

The library contains the Simplified Synchronous Machine block, which you can now use in your models.



See Also

`ssc_build`

More About

- “Build Custom Blocks Using the Three-Phase Electrical Domain” on page 5-2
- “Custom Components” (Simscape)
- “Customizing the Block Name and Appearance” (Simscape)
- “Component Equations” (Simscape)

Control

Tune an Electric Drive

In this section...

“Cascade Control Structure” on page 6-2

“Equations for PI Tuning Using the Pole Placement Method” on page 6-2

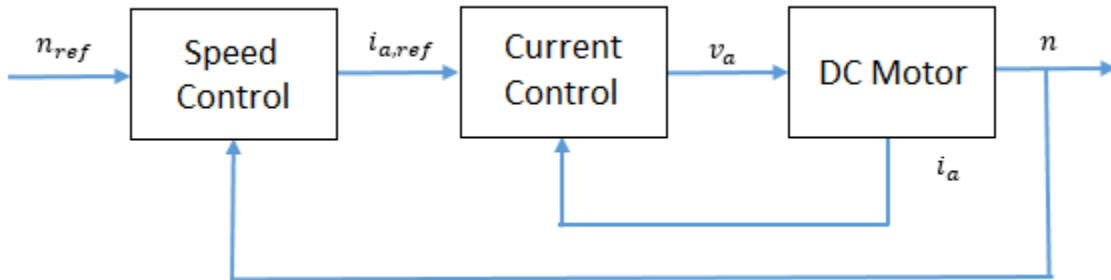
“Equations for DC Motor Controller Tuning” on page 6-5

“Tune the Electric Drive in the Example Model” on page 6-7

This example shows how to tune an electric drive using a cascade control structure.

Cascade Control Structure

The figure shows a feedback control loop that uses a cascade control structure. The outer speed-control loop is slower acting than the inner current-control loop.



Equations for PI Tuning Using the Pole Placement Method

To satisfy the required control performance for a simple discrete plant model, $G_f(z^{-1})$, use a closed loop PI control system $G_{PI}(z^{-1})$. The transient performance can be expressed in terms of the overshoot. The overshoot decreases relative to the damping factor:

$$\sigma = e^{\frac{-\pi\xi}{\sqrt{1-\xi^2}}}$$

where,

- σ is overshoot.
- ξ is the damping factor.

The response time, t_r , depends on the damping and the natural frequency, ω_n , such that:

- If $\xi < 0.7$,

$$t_r \cong \frac{4}{\omega_n \xi}.$$

- If $\xi \geq 0.7$,

$$t_r \cong \frac{6\xi}{\omega_n}.$$

The general workflow for designing a PI controller for a first-order system is:

- 1 Discretize the plant model using the zero-order hold (ZOH) discretization method. That is, given that the first-order equation representing the plant is

$$G(s) = \frac{K_m}{T_m s + 1},$$

where,

- K_m is the first-order gain.
- T_m is time constant of the first-order system.

Setting

$$s = \frac{1 - z^{-1}}{z^{-1}T_s},$$

yields the discrete plant model,

$$G(z^{-1}) = \frac{K_m \left(\frac{T_s}{T_m} \right) z^{-1}}{1 + \left(\frac{T_s - T_m}{T_m} \right) z^{-1}} = \frac{b_1 z^{-1}}{1 + a_1 z^{-1}},$$

where T_s is sample time for the discrete-time controller.

- 2 Write a discrete-time representation for the PI controller using the same transform. For

$$G_{PI}(s) = K_P + K_I \left(\frac{1}{s} \right),$$

setting

$$s = \frac{1 - z^{-1}}{z^{-1}T_s},$$

yields the discrete controller model,

$$G_{PI}(z^{-1}) = \frac{K_P + (K_I T_s - K_P)z^{-1}}{1 - z^{-1}} = \frac{q_0 + q_1 z^{-1}}{1 - z^{-1}}.$$

Combining the discrete equations for the plant and the controller yields the closed loop transfer function for the system,

$$G_0(z^{-1}) = \frac{q_0 b_1 z^{-1} + q_1 b_1 z^{-2}}{1 + (a_1 - 1 + q_0 b_1)z^{-1} + (-a_1 + q_1 b_1)z^{-2}},$$

The denominator of the transfer function is the characteristic polynomial. That is,

$$P_{C0}(z^{-1}) = 1 + (a_1 - 1 + q_0 b_1)z^{-1} + (-a_1 + q_1 b_1)z^{-2}.$$

- 3** The characteristic polynomial for achieving the required performance is defined as

$$P_{cd}(z^{-1}) = 1 + \alpha_1 z^{-1} + \alpha_2 z^{-2},$$

where,

- $\alpha_1 = -2e^{-\xi\omega_n T_s} \cos(\omega_n T_s \sqrt{1 - \xi^2})$.
- $\alpha_2 = e^{-2\xi\omega_n T_s}$.

- 4** To determine the controller parameters, set the characteristic polynomial for the system equal to the characteristic polynomial for the required performance. If

$$P_{C0}(z^{-1}) = P_{cd}(z^{-1}),$$

then

$$\alpha_1 = a_1 - 1 + q_0 b_1$$

and

$$\alpha_2 = -a_1 + q_1 b_1.$$

Solving for q_0 and q_1 yields

$$q_0 = \frac{\alpha_1 - a_1 + 1}{b_1}$$

and

$$q_1 = \frac{\alpha_2 + a_1}{b_1}.$$

Therefore, the general equations for the proportional and integral control parameters for the first-order system are

$$K_P = q_0$$

and

$$K_I = \frac{q_1 + K_P}{T_s}.$$

Equations for DC Motor Controller Tuning

Assuming that, for the system in the example model, $K_b = K_t$, the simplified mathematical equations for voltage and torque of the DC motor are

$$v_a = L_a \frac{di_a}{dt} + R_a i_a + K_b \omega$$

and

$$T_e = J_m \frac{d\omega}{dt} + B_m \omega + T_{load} = K_b i_a,$$

where:

- v_a is the armature voltage.
- i_a is the armature current.

- L_a is the armature inductance.
- R_a is the armature resistance.
- ω is the rotor angular velocity
- T_e is the motor torque.
- T_{load} is the load torque.
- J_m is the rotor moment of inertia.
- B_m is the viscous friction coefficient.
- K_b is a constant of proportionality.

To tune the current controller, assume that the model is linear, that is, that the back electromotive force, as represented by $K_b\omega$, is negligible. This assumption allows for an approximation of the plant model using this first-order Laplace equation:

$$G_i(s) = \frac{\frac{1}{R_a}}{\left(\frac{L_a}{R_a}\right)s + 1}.$$

Given the system requirements, you can now solve for K_p and K_I . The requirements for the current controller in the example model are:

- Sample time, $T_s = 1$ ms.
- Overshoot, $\sigma = 5\%$.
- Response time, $t_r = 0.11$ s.

Therefore, the proportional and integral parameters for the current controller are:

- $K_p = 7.7099$.
- $K_I = 455.1491$.

To tune the speed controller, approximate the plant model with a simple model. First assume that the inner loop is much faster than the outer loop. Also assume that there is no steady-state error. These assumptions allow for the use a first-order system by considering a transfer function of 1 for the inner current loop.

To output rotational velocity in revolutions per minute, the transfer function is multiplied by a factor of $30/\pi$. To take as control input the armature current instead of the motor torque, the transfer function is multiplied by the proportionality constant, K_b . The resulting approximation for the outer-loop plant model is

$$G_n(s) = \frac{\frac{30K_b}{nB_m}}{\left(\frac{J_m}{B_m}\right)s + 1}.$$

The speed controller has the same sample time and overshoot requirements as the current controller, but the response time is slower, such that:

- Sample time $T_s = 1$ ms.
- Overshoot $\sigma = 5\%$.
- Response time $t_r = 0.50$ s.

Therefore, the proportional and integral parameters for the speed controller are:

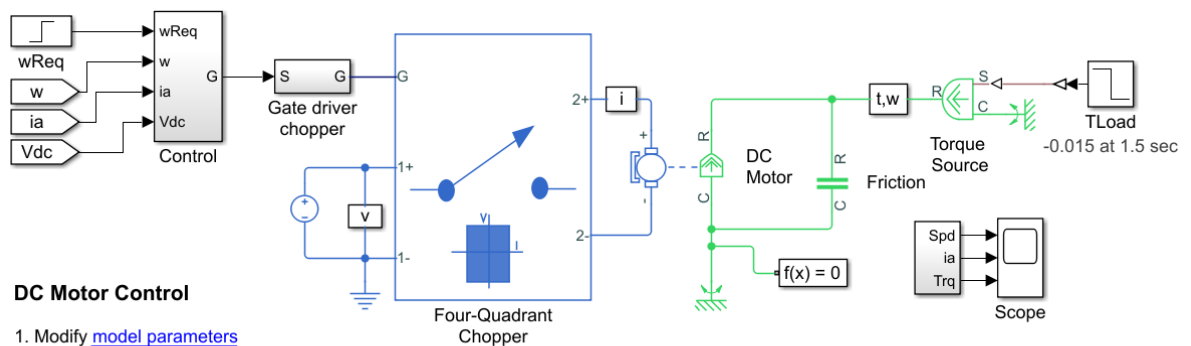
- $K_P = 0.0045$
- $K_I = 0.0405$

Tune the Electric Drive in the Example Model

1 Explore the components of the DC motor and the cascaded controller.

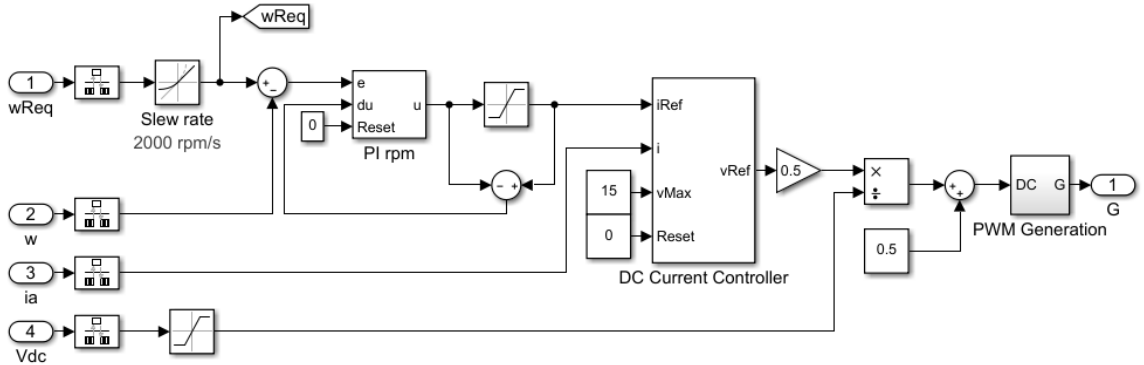
a Open the model. At the MATLAB command prompt, enter

```
model = 'ee_dc_motor_control'
open_system(model)
```

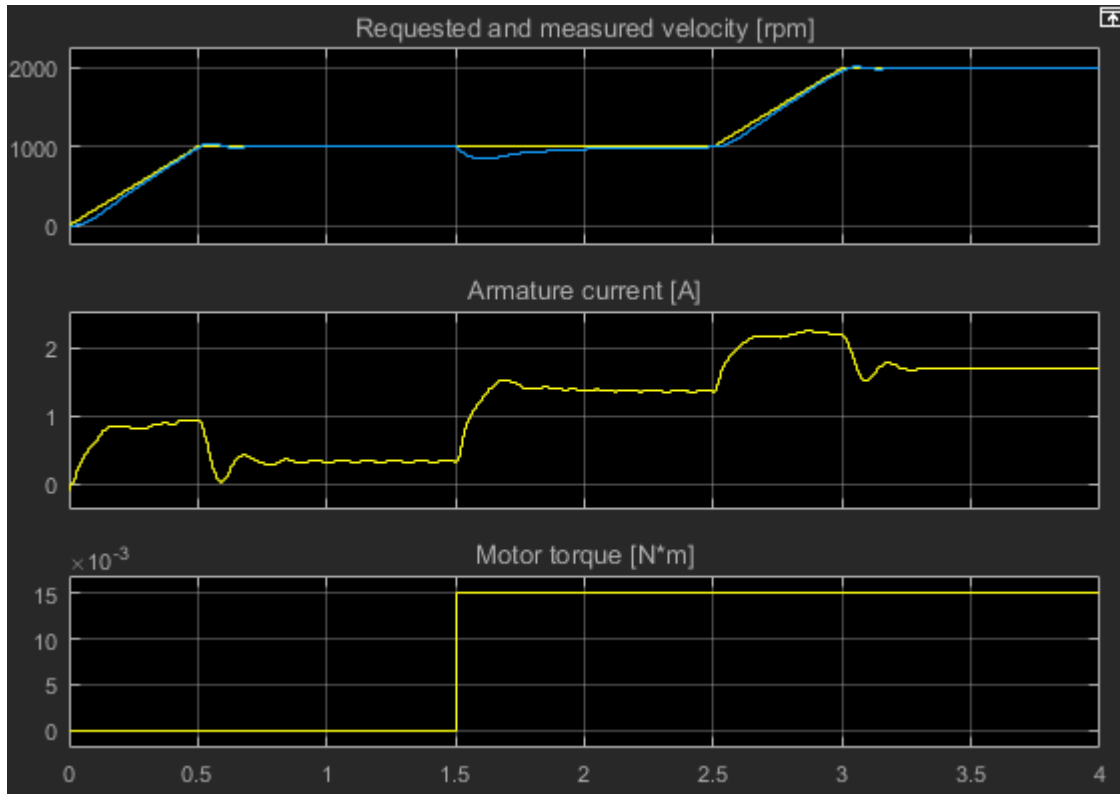


1. Modify [model parameters](#)
2. [Explore simulation results](#) using `sscexplore`
3. [Learn more](#) about this example

- b** The **Control** subsystem contains the model of the cascaded control system built using blocks from the Simulink library.



- c** The Four Quadrant Chopper block represents a four-quadrant DC-DC chopper that contains two bridge arms, each of which has two IGBT (Ideal, Switching) blocks. When the input voltage exceeds the threshold of 0.5 V, the IGBT (Ideal, Switching) blocks behave like linear diodes with a forward-voltage of 0.8 V and a resistance of $1e-4$ ohm. When the threshold voltage is not exceeded, the IGBT (Ideal, Switching) blocks act like linear resistors with an off-state conductance of $1e-5$ 1/ohm.
- 2 Simulate the model.
`sim(model)`
 - 3 View the results. Open the **Scope** block.



At 1.5 seconds, there is a load torque that results in a steady-state error.

- 4 Tune the DC motor controller. The `ee_getDCMotorFirstOrderPIParams` function calculates the proportional gain, K_p , and the integral gain, K_i , for the first-order system in this example.

The function syntax is `[Kp, Ki] = getParamPI(Km,Tm,Ts,sigma,tr)`.

The input arguments for the function are the system parameters and the requirements for the controller:

- K_m is the first-order gain.
- T_m is the time constant of the first-order system.
- T_s is the sample time for the discrete-time controller.

- σ is the desired maximum overshoot, σ .
 - t_r is the desired response time.
- a** To examine the equations in the function, enter
- edit `ee_getDCMotorFirstOrderPIParams`
- b** To calculate the controller parameters using the function, save these system parameters to the workspace:

```
Ra=4.67;           % [Ohm]
La=170e-3;        % [H]
Bm=47.3e-6;       % [N*m/(rad/s)]
Jm=42.6e-6;       % [Kg*m^2]
Kb=14.7e-3;       % [V/(rad/s)]
Tsc=1e-3;         % [s]
```

- c** Calculate the parameters for tuning the current controller as a function of the parameters and requirements for the inner controller:

- $K_m = 1/R_a$.
- $T_m = L_a/R_a$.
- $T_s = T_{sc}$.
- $\sigma = 0.05$.
- $T_r = 0.11$.

```
[Kp_i, Ki_i] = ee_getDCMotorFirstOrderPIParams(1/Ra,La/Ra,Tsc,0.05,0.11)
```

```
Kp_i =
    7.7099
```

```
Ki_i =
   455.1491
```

The gain parameters for the current controller are saved to the workspace.

- d** Calculate the parameters for tuning the speed controller based on the parameters and requirements for the outer controller:
- $K_m = K_b \cdot (30/\pi)$.

- $T_m = J_m/R_a$.
- $T_s = T_{sc}$.
- $\sigma = 0.05$.
- $T_r = 0.5$.

```
[Kp_n, Ki_n] = ee_getDCMotorFirstOrderPIParams((Kb*(30/pi))/Bm, Jm/Bm, Tsc, 0.05, 0.5)
```

```
Kp_n =
```

```
0.0045
```

```
Ki_n =
```

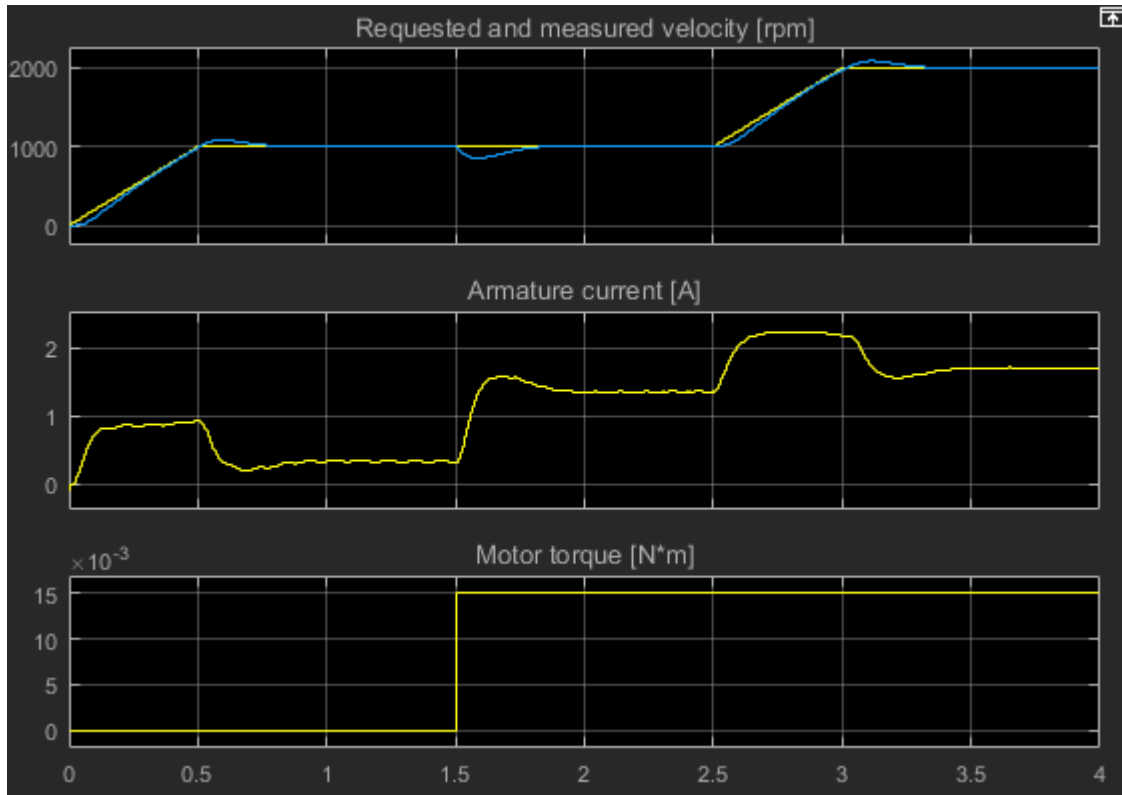
```
0.0405
```

The gain parameters for the speed controller are saved to the workspace.

- 5 Simulate the model using the saved gain parameters for the speed and controllers.

```
sim(model)
```

- 6 View the results. Open the **Scope** block.



There is slightly more overshoot, however, the controller responds much faster to the load torque change.

See Also

Inertia | Rotational Electromechanical Converter | Rotational Friction

Related Examples

- "DC Motor Control"

Simulation and Analysis of Power Engineering Systems

- “Optimize Block Settings for Simulating with the Partitioning Solver” on page 7-2
- “Phasor-Mode Simulation Using Simscape Components” on page 7-13
- “Examine the Simulation Data Logging Configuration of a Model” on page 7-19
- “Perform a Power-Loss Analysis” on page 7-21
- “Choose a Simscape Electrical Function for an Offline Harmonic Analysis” on page 7-30
- “Perform an Online Harmonic Analysis Using the Simscape Spectrum Analyzer Block” on page 7-34

Optimize Block Settings for Simulating with the Partitioning Solver

In this section...

“Update Solver and Zero-Sequence Settings Using the ee_solverUpdate Function” on page 7-3

“Limitations of the ee_updateSolver Function” on page 7-12

The Partitioning solver is a Simscape fixed-step local solver that improves performance for certain models. However, not all networks can simulate with the Partitioning solver. Some models that use the Partitioning solver can produce errors and fail to initialize due to numerical difficulties. To resolve numerical difficulties preventing initialization with asynchronous, synchronous, and permanent magnet rotor machine blocks, you can exclude zero-sequence terms. Excluding parasitic conductance resolves numerical difficulties with the Floating Neutral (Three-Phase) and Neutral Connection block, which include such conductance by default.

To determine the best solver choice for your model, use the ee_updateSolver helper function, which is useful for iterating with various solvers. The function updates certain parameter values for every instance of these blocks in your model:

- Solver Configuration blocks
- Machine blocks that have a **Zero sequence** parameter
- Connection blocks that have a **Parasitic conductance to ground** parameter

The function syntax is ee_updateSolver(solver, system). Specify both input arguments using character vectors. The table shows how the function updates the values, depending on the solver that you specify.

Input Argument	Solver Configuration Block (<i>Solver type</i>)	Solver Configuration Block (<i>Use local solver and Use fixed-cost runtime consistency iterations</i>)	Asynchronous, Synchronous, and Permanent Magnet Rotor Machine Blocks (<i>Zero sequence</i>)	Floating Neutral (Three-Phase) Block and Neutral Connection Block (<i>Parasitic conductance to ground</i>)
'Partitioning'	Partitioning	Selected	Exclude	0
'Backward Euler' or 'BackwardEuler'	Backward Euler	Selected	Include	1e-12
'Trapezoidal'	Trapezoidal	Selected	Include	1e-12
'Global' or 'Nonlocal'	No change	Cleared	Include	1e-12

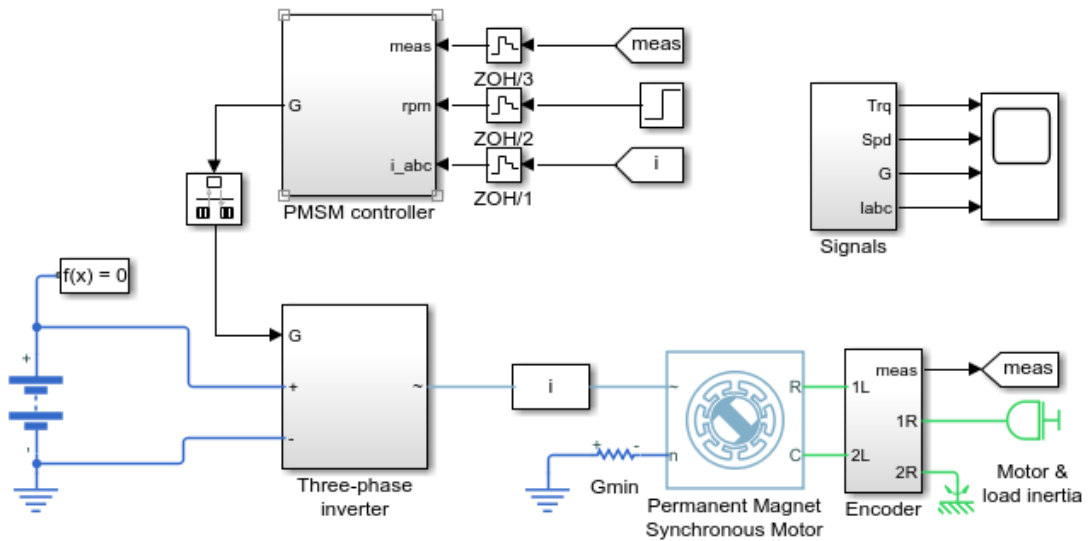
Update Solver and Zero-Sequence Settings Using the ee_solverUpdate Function

This example shows how to use the ee_solverUpdate function to configure the Solver Configuration and PMSM blocks in a model for simulation with the Partitioning solver and the Backward Euler solver. It also shows how to compare the simulation duration times and the results.

- 1 Open the model. At the MATLAB command prompt, enter this code.

See Code

```
model = 'ee_pmsm_drive';
open_system(model)
```



Three-Phase PMSM Drive

1. [Explore simulation results](#) using [sscexplore](#)
2. [Learn more](#) about this example

Two blocks that the `ee_solverUpdate` function can update are the Solver Configuration block and PMSM block.

- 2 Save the parameter settings for the two blocks.

See Code

```
% Define the Solver Configuration block and the path
% to it as variables
solvConfig = 'Solver Configuration';
solvConfigPath = [model, '/', solvConfig];

% Define the machine block and the path
% to it as variables
machine = 'Permanent Magnet Synchronous Motor';
machinePath = [model, '/', machine];

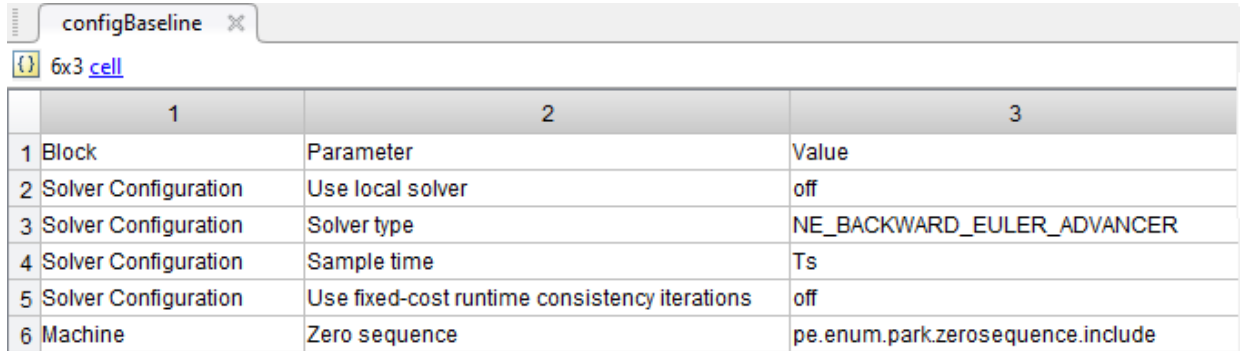
% Create a cell array that contains configuration data
configBaseline = {'Block', 'Parameter', 'Value'}
```

```

'Solver Configuration','Use local solver',...
get_param(solvConfigPath,'UseLocalSolver');
'Solver Configuration','Solver type',...
'Solver Configuration','LocalSolverChoice');
get_param(solvConfigPath,'LocalSolverChoice');
'Solver Configuration','Sample time',...
get_param(solvConfigPath,'LocalSolverSampleTime');
'Solver Configuration',...
'Use fixed-cost runtime consistency iterations',...
get_param(solvConfigPath,'DoFixedCost');
'Machine','Zero sequence',...
get_param(machinePath,'zero_sequence')});

```

The settings are saved to configBaseline array in the MATLAB workspace.



	1	2	3
1	Block	Parameter	Value
2	Solver Configuration	Use local solver	off
3	Solver Configuration	Solver type	NE_BACKWARD_EULER_ADVANCER
4	Solver Configuration	Sample time	Ts
5	Solver Configuration	Use fixed-cost runtime consistency iterations	off
6	Machine	Zero sequence	pe.enum.park.zerosequence.include

The settings of interest for the Solver Configuration block are:

- **Use local solver** — The option to use a local Simscape solver is cleared.
- **Solver type** — Backward Euler, a Simscape local fixed- cost solver, is specified. However, if you open the block dialog box, you can see that it is not enabled because the option to use a local solver is cleared.
- **Use fixed-cost runtime consistency iterations** — The option to use fixed-cost is cleared. This option is also disabled when the option to use a local solver is cleared.

For the machine, the **Zero sequence** parameter is set to Include. Zero-sequence equations can cause numerical difficulty when you simulate with the Partitioning solver.

- 3 To return all simulation outputs within a single `Simulink.SimulationOutput` object so that you can later compare simulation times, enable the single-output format of the `sim` command.

```
% Enable single-output format
set_param(model, 'ReturnWorkspaceOutputs', 'on')
```

- 4 Mark the rotor torque signal, which connects the **trqMotor** From block to a Mux block, for Simulink data logging and viewing with the Simulink Data Inspector.

See Code

```
% Define the trqMotor From block and the path
%   to it as variables
torqueSensor = 'From6';
signalSubsystem = 'Signals';
torqueSensorPath = [model, '/', signalSubsystem, '/', torqueSensor];

% Mark the output signal from the trqMotor From block
%   for Simulink(R) data logging
phTorqueSensor = get_param(torqueSensorPath, 'PortHandles');
set_param(phTorqueSensor.Outport(1), 'DataLogging', 'on')
```

The logging badge  marks the signal in the model.

- 5 Determine the results and how long it takes to simulate with the baseline settings.

See Code

```
% Run a timed simulation using the Baseline solver configuration
out = sim(model);
tBaseline = out.SimulationMetadata.TimingInfo.ExecutionElapsedWallTime;
```

- 6 Use `ee_updateSolver` function to change to the Backward Euler solver configuration. Save the configuration settings, and compare the settings to the baseline settings.

See Code

```
% Configure for Backward Euler solver simulation
ee_updateSolver('Backward Euler', model)

% Save the new parameter settings and compare them to the baseline
%   configuration.

% Create a cell array that contains configuration data
configBackEuler = {'Block', 'Parameter', 'Value';
                  'Solver Configuration', 'Use local solver', ...
```



```

get_param(solvConfigPath,'UseLocalSolver');
'Solver Configuration','Solver type',...
get_param(solvConfigPath,'LocalSolverChoice');
'Solver Configuration','Sample time',...
get_param(solvConfigPath,'LocalSolverSampleTime');
'Solver Configuration',...
'Use fixed-cost runtime consistency iterations',...
get_param(solvConfigPath,'DoFixedCost');
'Machine','Zero sequence',...
get_param(machinePath,'zero_sequence'));

% Compare the Partitioning solver block settings to the Baseline settings
configDiff = setdiff(configBackEuler,configBaseline)

configDiff =

    1x1 cell array

    {'on'}

```

	1	2	3
1	Block	Parameter	Value
2	Solver Configuration	Use local solver	on
3	Solver Configuration	Solver type	NE_BACKWARD_EULER_ADVANCER
4	Solver Configuration	Sample time	Ts
5	Solver Configuration	Use fixed-cost runtime consistency iterations	on
6	Machine	Zero sequence	pe.enum.park.zerosequence.include

The option to use the local solver, which is set to Backward Euler by default, and the option to use fixed-cost runtime consistency iterations are now both selected.

- 7 Run a timed simulation using the Backward Euler solver.

See Code

```

out = sim(model);
tBackEuler = out.SimulationMetadata.TimingInfo.ExecutionElapsedWallTime;

```

- 8 If you change the local solver to the Partitioning solver and simulate the model now, an error occurs because of the zero-sequence terms. Use the `ee_updateSolver` function to configure the model for simulating with the Partitioning solver without generating an error. Save the configuration settings, compare the settings to baseline settings, and run a timed simulation.

See Code

```
% Configure for Partitioning solver simulation
ee_updateSolver('Partitioning', model)

% Create a cell array that contains configuration data
configPartitioning = {'Block','Parameter','Value';
    'Solver Configuration','Use local solver',...
    get_param(solvConfigPath,'UseLocalSolver');
    'Solver Configuration','Solver type',...
    get_param(solvConfigPath,'LocalSolverChoice');
    'Solver Configuration','Sample time',...
    get_param(solvConfigPath,'LocalSolverSampleTime');
    'Solver Configuration',...
    'Use fixed-cost runtime consistency iterations',...
    get_param(solvConfigPath,'DoFixedCost');
    'Machine','Zero sequence',...
    get_param(machinePath,'zero_sequence')};

% Compare the Partitioning solver block settings to the Baseline settings
configDiff = setdiff(configPartitioning,configBaseline)

% Run a timed simulation using the Partitioning solver
out = sim(model);
tPartitioning = out.SimulationMetadata.TimingInfo.ExecutionElapsedWallTime;

configDiff =

3x1 cell array

    {'NE_PARTITIONING_ADVANCER'          }
    {'ee.enum.park.zerosequence.exclude'}
    {'on'                                }
```

Warning: Initial conditions for nondifferential variables not supported. The following states may deviate from requested initial conditions:

```
['<a href="matlab:open_and_hilite_system('ee_pmsm_drive/Battery')"...
    >ee_pmsm_drive/Battery</a>']
Battery.num_cycles
    o In ee.sources.battery_base
['<a href="matlab:open_and_hilite_system('ee_pmsm_drive/Permanent
Magnet Synchronous Motor')">ee_pmsm_drive/Permanent
Magnet Synchronous Motor</a>']
Permanent_Magnet_Synchronous_Motor.angular_position
```

configPartitioning			
6x3 cell			
	1	2	3
1	Block	Parameter	Value
2	Solver Configuration	Use local solver	on
3	Solver Configuration	Solver type	NE_PARTITIONING_ADVANCER
4	Solver Configuration	Sample time	Ts
5	Solver Configuration	Use fixed-cost runtime consistency iterations	on
6	Machine	Zero sequence	pe.enum.park.zerosequence.exclude

The solver type is now set to the Partitioning solver and the machine is configured to exclude zero-sequence terms.

The simulation runs without generating an error. It does generate a warning because initial conditions for nondifferential variables are not supported for the Partitioning solver.

9 Print tables that show:

- Simulation time for each solver
- Percent differences in speed for the local solvers versus the baseline global solver.

See Code

```
% Display the simulation times
compTimeDiffTable = table({'Baseline';...
    'Backward Euler';...
    'Partitioning'},...
    {tBaseline;tBackEuler;tPartitioning},...
    'VariableNames', {'Solver', 'Sim_Duration'});

display(compTimeDiffTable);

% Compute and display the percent difference for the simulation times
spdBackEulerVsBaseline = 100*(tBaseline - tBackEuler)/tBaseline;
spdPartitionVsBaseline = 100*(tBaseline - tPartitioning)/tBaseline;

compPctDiffTable = table({'Backward Euler versus Baseline';...
    'Partitioning versus Baseline'},...
    {spdBackEulerVsBaseline;...
    spdPartitionVsBaseline},...
```

```
'VariableNames', {'Comparison', 'Percent_Difference'});
```

```
display(compPctDiffTable);
```

```
3x2 table
```

Solver	Sim_Duration
'Baseline'	[38.0255]
'Backward Euler'	[23.4011]
'Partitioning'	[9.2042]

```
compPctDiffTable =
```

```
2x2 table
```

Comparison	Percent_Difference
'Backward Euler versus Baseline'	[38.4594]
'Partitioning versus Baseline'	[75.7946]

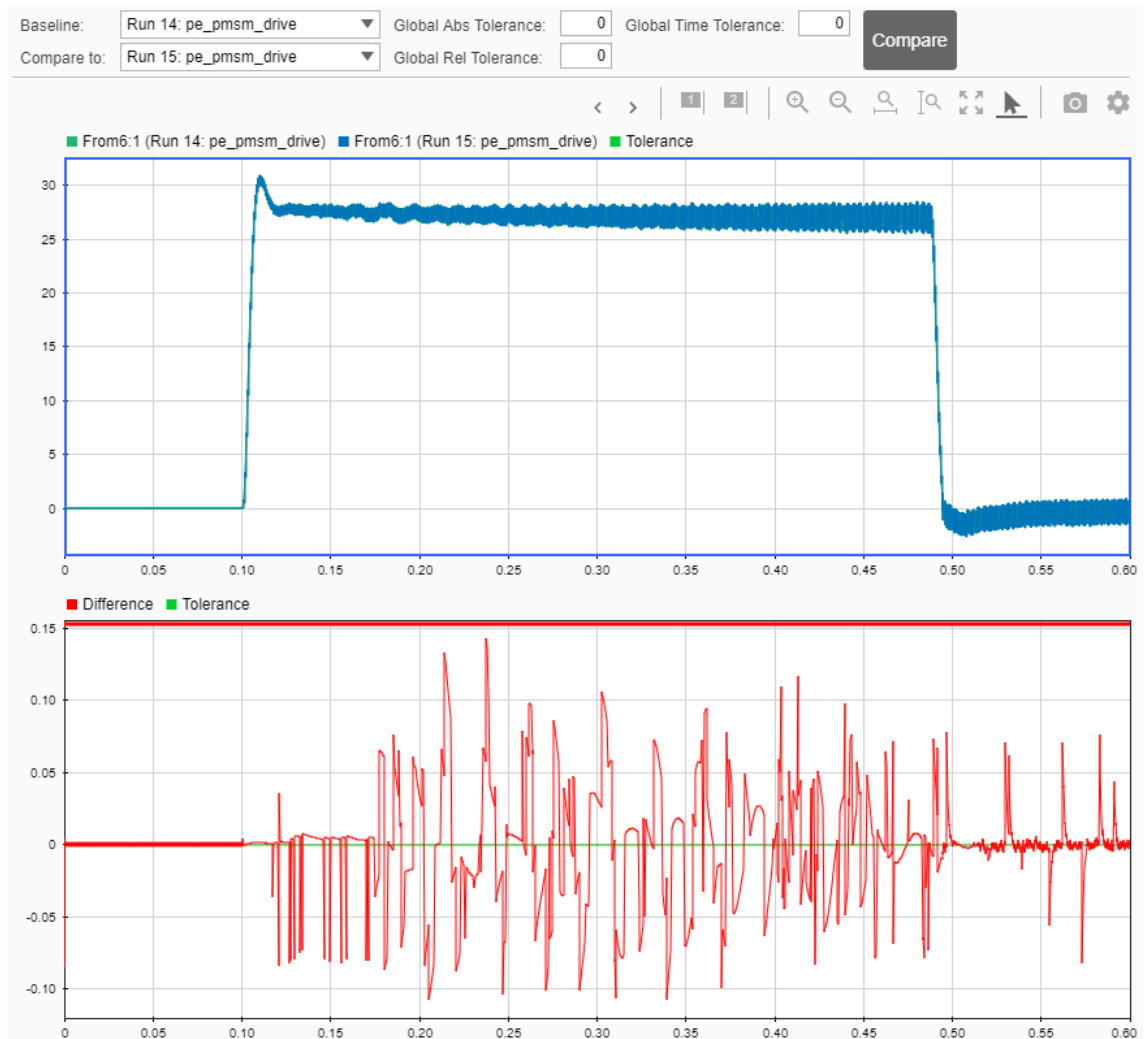
Simulation time on your machine may differ because simulation speed depends on machine processing power and the computational cost of concurrent processes. The local fixed-step Partitioning and Backward Euler solvers are faster than the baseline solver, which is a global, variable-step solver. The Partitioning solver is faster than the Backward Euler solver.

- 10 To compare the results, open the Simulink Data Inspector.

See Code

```
% Get Simulink Data Inspector run IDs for  
% the last three runs  
runIDs = Simulink.sdi.getAllRunIDs;  
runBackEuler = runIDs(end - 1);  
runPartition = runIDs(end);  
  
% Open the Simulink Data Inspector  
Simulink.sdi.view  
  
compBaselinePartition = Simulink.sdi.compareRuns(runBackEuler, ...  
runPartition);
```

To see the comparison, click **Compare** and then click **From6**.



The first plot shows the overlay of the Backward Euler and Partitioning solver simulation results. The second plot shows how they differ. The default tolerance for differences is 0. To determine if the accuracy of the results meets your requirements, you can adjust the relative, absolute, and time tolerances. For more information, see “Compare Simulation Data” (Simulink).

You can also use the `ee_updateSolver` function to reset the model for simulation with a global solver.

See Code

```
% Configure for Global/Nonlocal solver simulation  
ee_updateSolver('Global',model)
```

Limitations of the `ee_updateSolver` Function

Using the `ee_updateSolver` function does not guarantee that a simulation does not generate an error or that a simulation produces accurate results. To ensure that simulation accuracy meets your requirements, it is a recommended practice to compare simulation results to baseline results whenever you change model or block settings.

See Also

PMSM | Solver Configuration

Related Examples

- “Increase Simulation Speed Using the Partitioning Solver” (Simscape)

Phasor-Mode Simulation Using Simscape Components

You can run your model in phasor mode to speed up simulation. In Simscape, phasor mode is known as frequency-time equation formulation. In general, this formulation leads to accurate simulation of AC models using larger time steps than the traditional time formulation.

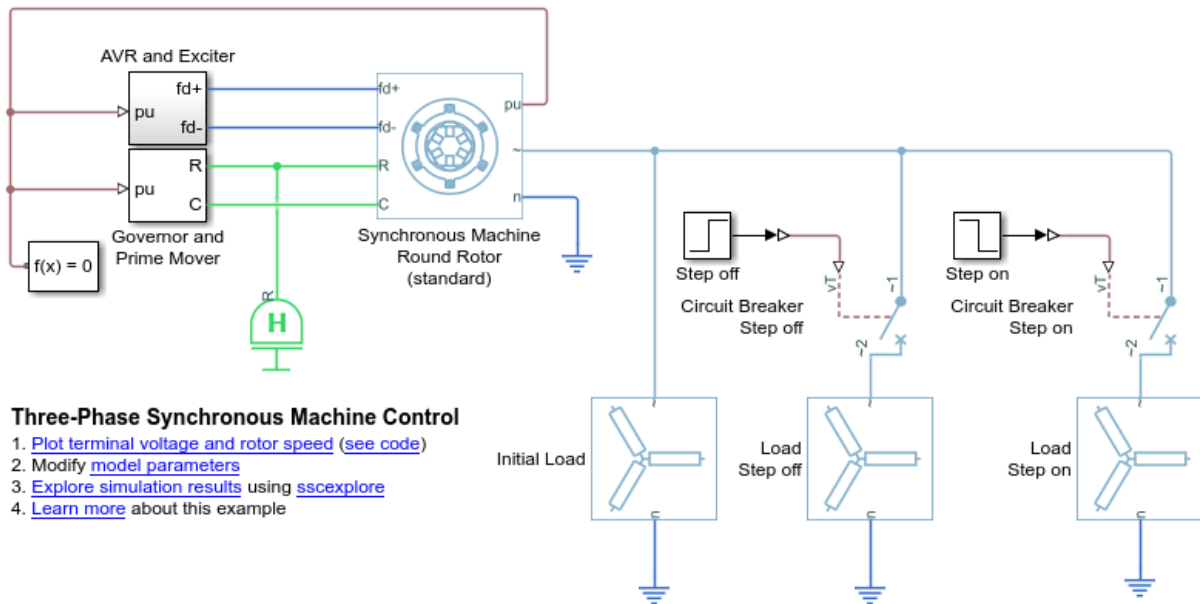
Use frequency-time equation formulation to speed up your simulation when:

- Your simulation contains periodic AC signals with a common fundamental frequency
- You are interested in the slow-moving AC-related quantities, such as amplitude or phase, and the DC output signals

Set up the model

To measure the time required to run a simulation, open the model *ee_sm_control* and create a model callback.

```
mdl = load_system('ee_sm_control');  
open_system(mdl);  
set_param(mdl, 'StartFcn', 'tic;');  
set_param(mdl, 'StopFcn', 'tsim=toc;');
```



Run a time-based simulation

Double-click the Solver Configuration block and apply the following configuration:

- Enable the local solver by checking the **Use local solver** check box
- Set the Sample time parameter to $1e-3$
- Set the Equation formulation parameter to Time

You can also run this code to configure the block.

```
blk = find_system mdl, 'MaskType', 'Solver Configuration';
set_param(blk, 'UseLocalSolver', 'on');
set_param(blk, 'LocalSolverSampleTime', '1e-3');
set_param(blk, 'EquationFormulation', 'NE_TIME_EF');
```

Simulate the model and save the run time and logging variable.

```
sim(get_param(mdl, 'Name'));
tsim_time = round(tsim, 2);
simlog_ee_sm_control_time = simlog_ee_sm_control;
```


Run a phasor-mode simulation

Double-click the Solver Configuration block and apply the following configuration:

- Enable the local solver by checking the `Use local solver` check box
- Set the `Sample time` parameter to `1e-2`
- Set the `Equation formulation` parameter to `Frequency and time`

You can also run this code to configure the block.

```
blk = find_system mdl, 'name', 'Solver Configuration');
set_param(blk, 'UseLocalSolver', 'on');
set_param(blk, 'LocalSolverSampleTime', '1e-2');
set_param(blk, 'EquationFormulation', 'NE_FREQUENCY_TIME_EF');
```

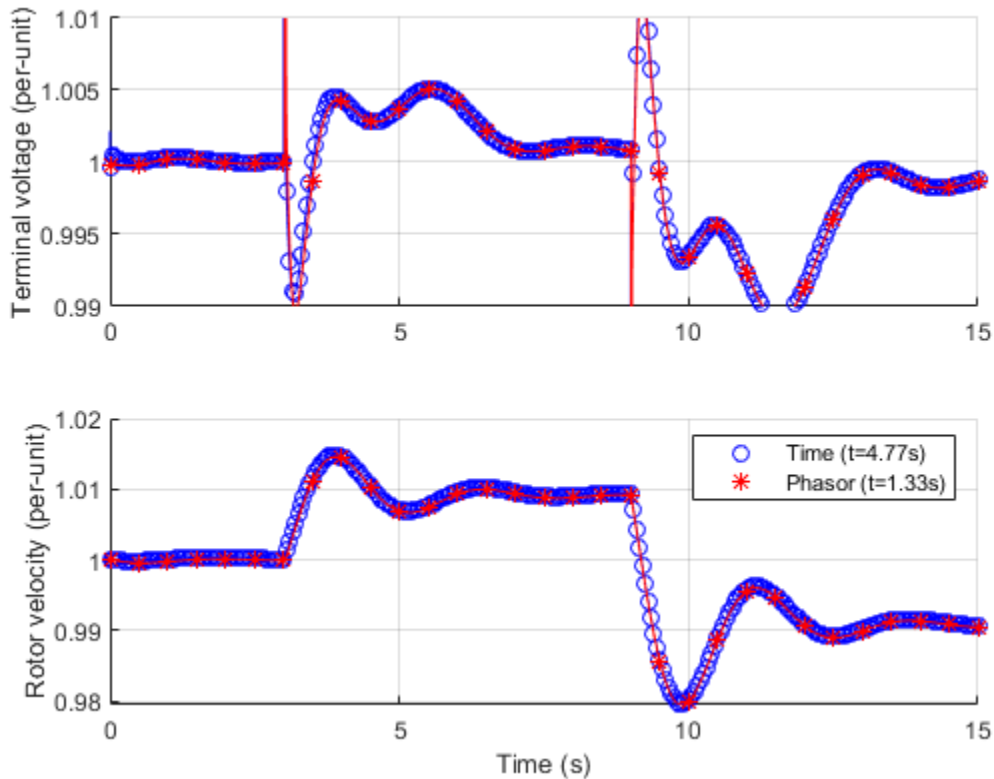
Simulate the model and save the run time and logging variable.

```
sim(get_param mdl, 'Name');
tsim_phasor = round(tsim, 2);
simlog_ee_sm_control_phasor = simlog_ee_sm_control;
```

Compare DC results

Plot the field voltage and rotor speed for both the time and frequency-time simulations. For each simulation mode, display markers at every 50 data points.

```
[hTime, hPhasor] = setup_figure(simlog_ee_sm_control_time, simlog_ee_sm_control_phasor, 'dc');
legend([hTime, hPhasor], {'Time (t=', num2str(tsim_time), 's)'}, ['Phasor (t=', num2str(tsim_time), 's)']);
```

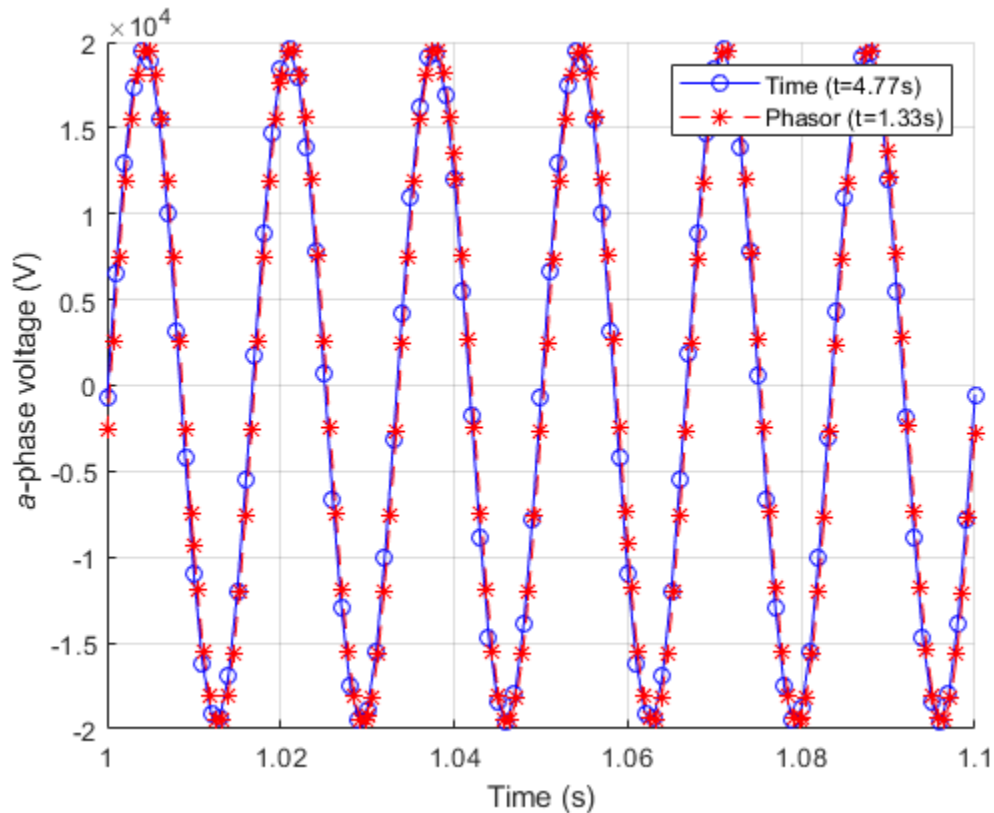


The phasor simulation reproduces near-identical results as the time-based simulation, despite using a time step that is 10 times larger. The measured simulation time is also shown for each of the simulation modes in the plot legend. This performance indicator is different on different machines, but the frequency-time simulation should be about two times faster than the time simulation. Note that the actual time required per step is higher in the frequency-time case, but the overall time is reduced.

Compare AC results

Plot the a -phase voltage of the synchronous machine over the time period 1s to 1.1s. Because of the larger time steps in the frequency-time formulation, the resolution of the AC quantity is too small to make out the sine wave. The points that are available are undersampled, but still accurate.

```
[hTime,hPhasor]=setup_figure(simlog_ee_sm_control_time,simlog_ee_sm_control_phasor,'ac'  
legend([hTime,hPhasor],{'Time (t=',num2str(tsim_time),'s)'},['Phasor (t=',num2str(tsim_time),'s)']
```



In general, use frequency-time formulation to speed up simulations where the outputs of interest are DC or slow-moving AC quantities. You can use periodic sensors to measure slow-moving properties of AC signals such as amplitude and phase in both time and frequency time formulations. For more information, see the PS Harmonic Estimator (Amplitude, Phase) block.

Sometimes there are small phase offsets between time- and frequency-time-generated AC signals. This difference is caused by the accumulated integration error of a slightly different signal frequency over time.

See Also

Solver Configuration

More About

- “Frequency and Time Simulation Mode” (Simscape)

Examine the Simulation Data Logging Configuration of a Model

Many analyses that you can perform using Simscape Electrical require a simulation log variable in your MATLAB workspace. The model in this example is configured to log Simscape data for the whole model for the entire simulation time. To examine the data logging configuration of a model:

- 1 Open the model. At the MATLAB command prompt, enter

```
model = 'ee_rectifier_power_dissipated';  
open(model)
```

- 2 Open the model configuration parameters and then, in the left pane, select **Simscape**. Relevant parameters are:

- **Log simulation data** — Data logging is enabled for the whole model because this parameter is set to All so you can calculate the power dissipated by any of the semiconductors in the model.
- **Workspace variable name** — This parameter, which is also referred to as the name of the simulation log variable, is specified as `simlog_ee_rectifier_power_dissipated`.
- **Limit data points** — You can calculate the power dissipated for the entire simulation time because the option is not selected.

Alternatively, you can determine the Simscape data logging configuration without opening the model configuration parameters, by using the `get_param` function. For example, for the `ee_rectifier_power_dissipated` model, to determine:

- If all, some, or no data is logged, at the MATLAB command prompt, enter

```
get_param(model, 'SimscapeLogType')
```

```
ans =
```

```
    'all'
```

- The name of the Simscape logging variable

```
get_param(model, 'SimscapeLogName')
```

```
ans =  
    'simlog_ee_rectifier_power_dissipated'  
• If the option to limit data-points is on or off  
get_param(model, 'SimscapeLogLimitData')  
ans =  
    'off'
```

See Also

Functions

get_param

Related Examples

- “Data Logging” (Simscape)

Perform a Power-Loss Analysis

In this section...

“Prerequisite” on page 7-21

“Calculate Average Power Losses for the Simulation” on page 7-21

“Analyze Power Dissipation Differences Using Instantaneous Power Dissipation” on page 7-23

“Mitigate Transient Effects in Simulation Data” on page 7-27

This example shows how to analyze power loss and how to mitigate transient power dissipation behavior. Analyzing power loss, with and without transients, is useful for determining if components are operating within safety and efficiency guidelines.

Prerequisite

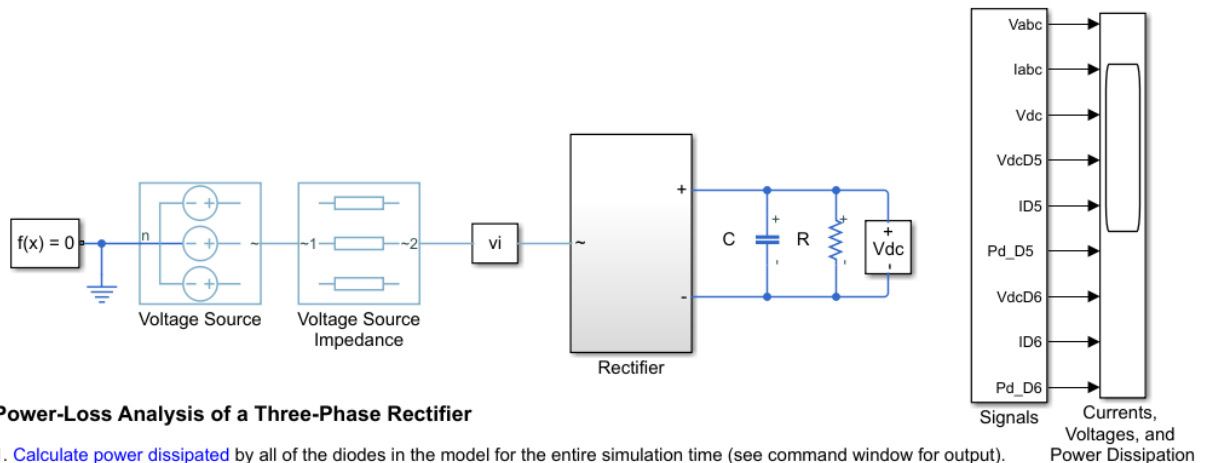
This example requires a simulation log variable in your MATLAB workspace. The model in this example is configured to log Simscape data for the whole model for the entire simulation time.

To learn how to determine if a model is configured to log simulation data, see “Examine the Simulation Data Logging Configuration of a Model” on page 7-19.

Calculate Average Power Losses for the Simulation

- 1 Open the model. At the MATLAB command prompt, enter

```
model = 'ee_rectifier_power_dissipated';  
open(model)
```



Power-Loss Analysis of a Three-Phase Rectifier

1. [Calculate power dissipated](#) by all of the diodes in the model for the entire simulation time (see command window for output).
2. [Calculate power dissipated](#) by diode D6 for the entire simulation time (see command window for output).
3. [Explore simulation results](#) using `sscexplore`.
4. [Calculate the maximum power dissipated](#) by diode D6. (see command window for output).
5. [Calculate power dissipated](#) by diode D6 between simulation time, $t = 1e-3$ to 0.5 s (see command window for output).
6. [Plot power dissipated](#) by the diodes ([see code](#)).
7. [Calculate power dissipated](#) by all of the diodes between simulation time, $t = 0.4$ to 0.5 s (see command window for output).
8. [Learn more](#) about the `ee_getPowerLossSummary` function.

2 Simulate the model.

```
sim(model)
```

The simulation log variable, which is named `simlog_ee_rectifier_power_dissipated`, appears in the workspace.

3 Calculate the average losses for the entire simulation for each of the diodes in the model.

```
rectifierLosses = ee_getPowerLossSummary(simlog_ee_rectifier_power_dissipated.Rectifier)
```

```
rectifierLosses =
```

```
6x2 table
```

LoggingNode	Power
'ee_rectifier_power_dissipated.Rectifier.D6'	52.222
'ee_rectifier_power_dissipated.Rectifier.D3'	52.222
'ee_rectifier_power_dissipated.Rectifier.D4'	52.194
'ee_rectifier_power_dissipated.Rectifier.D5'	52.194
'ee_rectifier_power_dissipated.Rectifier.D1'	52.194
'ee_rectifier_power_dissipated.Rectifier.D2'	52.194

On average, diodes D3 and D6 dissipate more power than the other diodes in the rectifier.


Analyze Power Dissipation Differences Using Instantaneous Power Dissipation

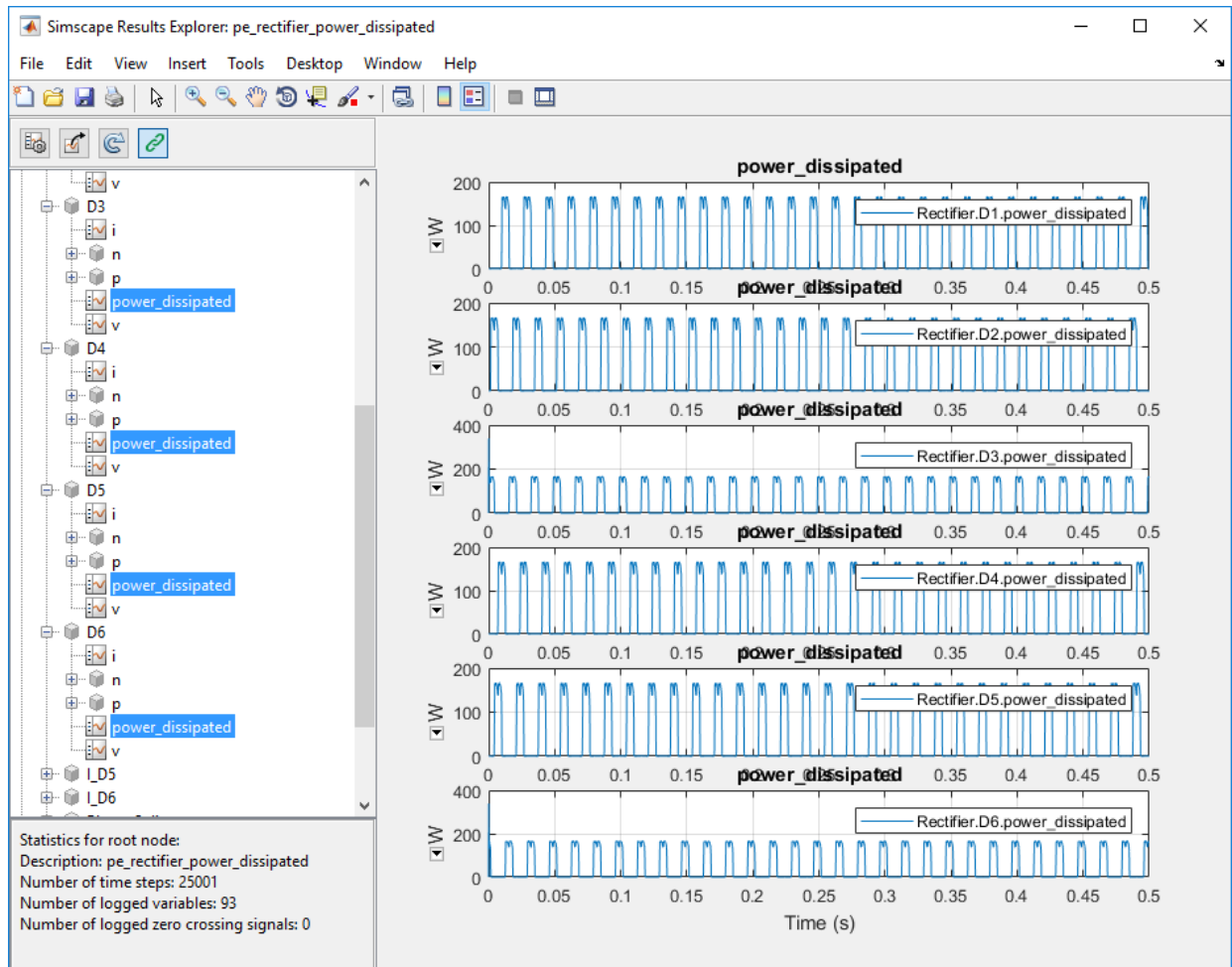
The Diode blocks each have a *power_dissipated* variable, which measures instantaneous power dissipation. To investigate the differences in the average power dissipated by the diodes, view the simulation data using the Simscape Results Explorer.

- 1 Open the simulation data using the Results Explorer.


```
sscexplore(simlog_ee_rectifier_power_dissipated)
```

- 2 View the instantaneous power dissipated by the diodes.

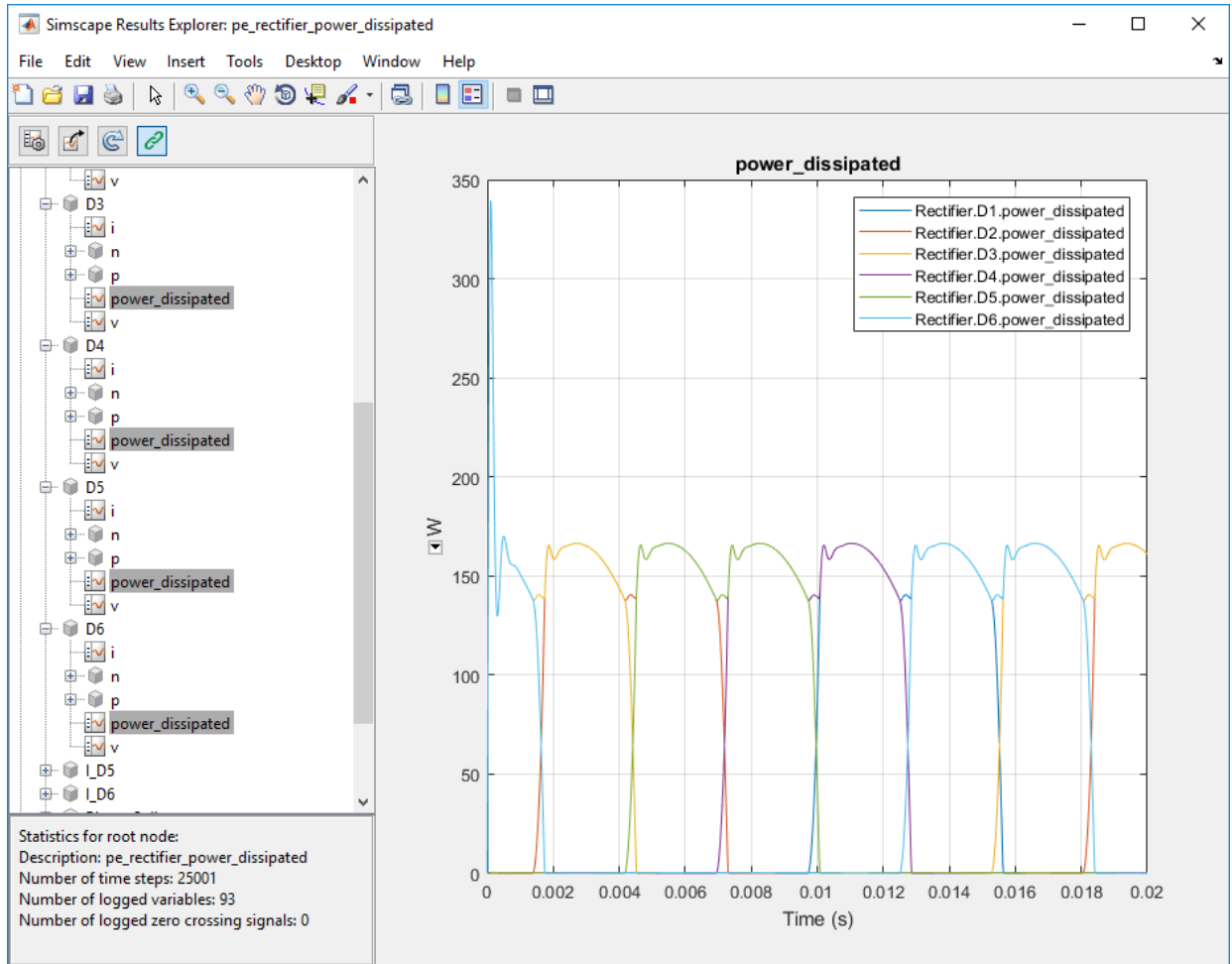
- a Expand the **Rectifier** node
- b Expand the **D1** through **D6** nodes
- c Click the *power_dissipated* nodes for diode **D1**, and then **Ctrl+click** the *power_dissipated* nodes for the other five diodes.
- d In the Results Explorer window, click the plot options  button and set **Plot signals** to Separate.



At the beginning of the simulation, there is a difference in the power dissipation for each diode.

- 3 Take a closer look at the differences. Overlay the plots and zoom to the beginning of the simulation.
 - a In the Results Explorer window, click the plot options  button.
 - b Enable the **Limit time axis** option.

- c For **Stop time**, specify 0.02.
- d Set **Plot signals** to Overlay.
- e Click **OK**.



The variation in power dissipation is due to transient behavior at the beginning of the simulation. The model reaches steady state at simulation time, $t \approx 0.001$ seconds.

- 4 Determine the average power dissipation for only the diodes during the interval that contains transient behavior.

```
rectifierLosses = ee_getPowerLossSummary(simlog_ee_rectifier_power_dissipated.Rectifier,0,1e-3)
```

```
rectifierLosses =
```

```
6x2 table
```

LoggingNode	Power
'Rectifier.D3'	174.88
'Rectifier.D6'	174.88
'Rectifier.D4'	0.27539
'Rectifier.D5'	0.27539
'Rectifier.D1'	0.12482
'Rectifier.D2'	0.032017

The average power dissipated by diodes D3 and D6 exceeds the average for the other diodes.

- 5 Output a table of the maximum power dissipation for each diode, for the entire simulation time.

```
pd_D1_max = max(simlog_ee_rectifier_power_dissipated.Rectifier.D1.power_dissipated.series.values);
pd_D2_max = max(simlog_ee_rectifier_power_dissipated.Rectifier.D2.power_dissipated.series.values);
pd_D3_max = max(simlog_ee_rectifier_power_dissipated.Rectifier.D3.power_dissipated.series.values);
pd_D4_max = max(simlog_ee_rectifier_power_dissipated.Rectifier.D4.power_dissipated.series.values);
pd_D5_max = max(simlog_ee_rectifier_power_dissipated.Rectifier.D5.power_dissipated.series.values);
pd_D6_max = max(simlog_ee_rectifier_power_dissipated.Rectifier.D6.power_dissipated.series.values);
```

```
diodes = {'D1';'D2';'D3';'D4';'D5';'D6'};
PowerMax = [pd_D1_max;pd_D2_max;pd_D3_max;pd_D4_max;pd_D5_max;pd_D6_max];
```

```
T = table(PowerMax,'RowNames', diodes)
```

```
T =
```

```
6x1 table
```

	PowerMax
D1	166.45
D2	166.45
D3	339.54
D4	166.45
D5	166.45
D6	339.54

The maximum instantaneous power dissipation for diodes D3 and D6 is almost double the maximum instantaneous power dissipation for the other diodes.


Mitigate Transient Effects in Simulation Data

To mitigate the transient power dissipation at the beginning of the simulation, use the final simulation state to initialize a new simulation at steady-state conditions.

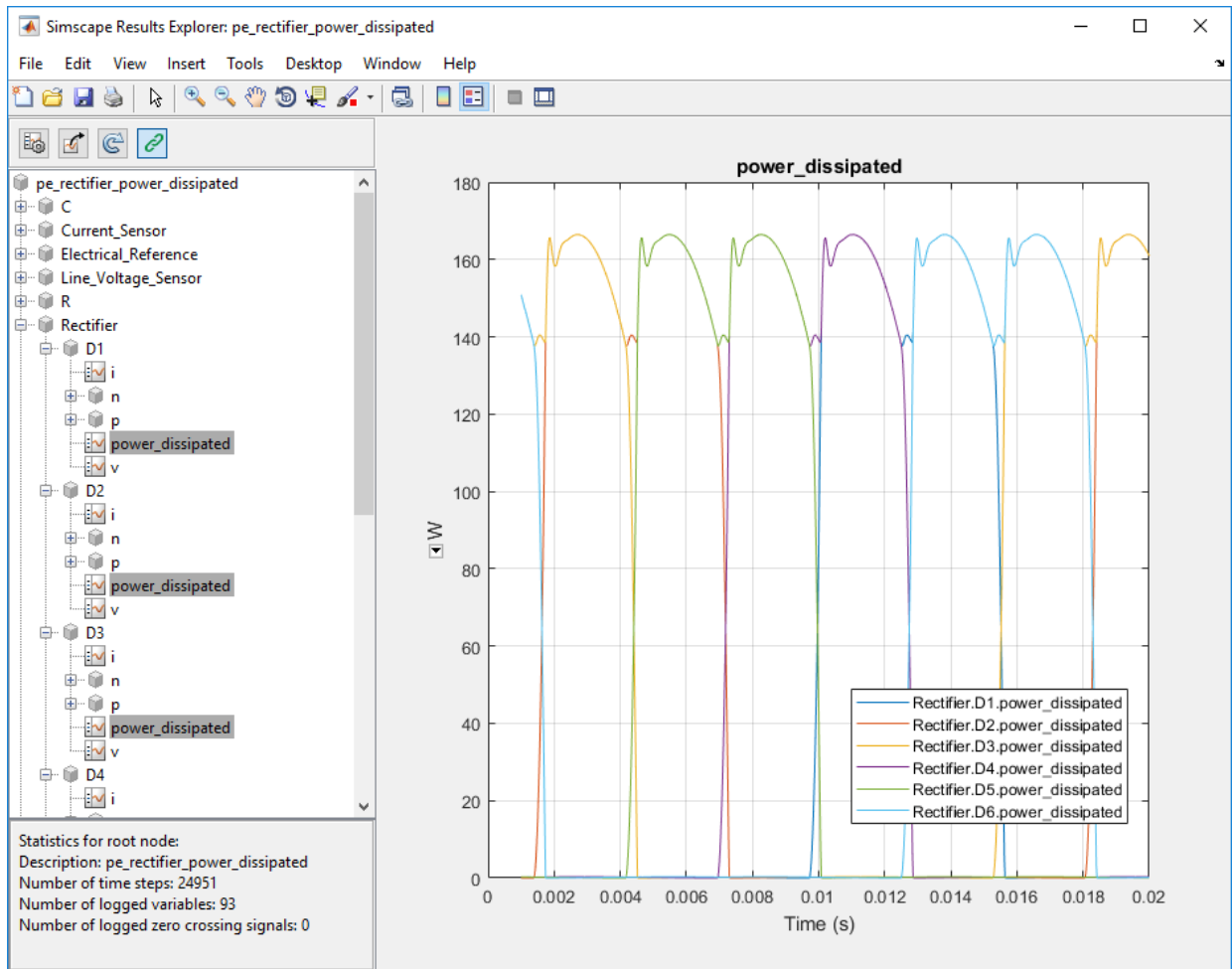
- 1 Configure the model to save the final state.
 - a Open the model configuration parameters.
 - b In the **Solver** pane, change the **Stop time** from 0.5 to 1e-3.
 - c In the **Data Import/Export** pane, select these options:
 - **Final States**
 - **Save final operating point**
 - d Click **Apply**.

- 2 Run the simulation.

The final state is saved as the variable `xFinal` in the MATLAB workspace.

- 3 Configure the model to initialize using `xFinal`, in the model configuration parameters.
 - a In the Data Import/Export pane:
 - Select the **Initial state** option.
 - Change the **Initial state** parameter value from `xInitial` to `xFinal`.
 - Clear the **Final states** option.
 - b In the **Solver** pane, change the **Stop time** to 0.5.
 - c Click **OK**.
- 4 Run the simulation.
- 5 View the data from the new simulation.
 - a Click the **Reload logged data**  button in the Simscape Results Explorer.
 - b Click **OK** to confirm that `simlog_ee_rectifier_power_dissipated` is the variable name that contains the logged data.

- c To see the data more clearly, click and drag the legend away from the peak amplitudes.



The plot shows that the simulation no longer contains the transient.

- 6 Output a table of the maximum power dissipation for each diode, for the modified simulation.

```
pd_D1_max = max(simlog_ee_rectifier_power_dissipated.Rectifier.D1.power_dissipated.series.values);
pd_D2_max = max(simlog_ee_rectifier_power_dissipated.Rectifier.D2.power_dissipated.series.values);
pd_D3_max = max(simlog_ee_rectifier_power_dissipated.Rectifier.D3.power_dissipated.series.values);
```

```

pd_D4_max = max(simlog_ee_rectifier_power_dissipated.Rectifier.D4.power_dissipated.series.values);
pd_D5_max = max(simlog_ee_rectifier_power_dissipated.Rectifier.D5.power_dissipated.series.values);
pd_D6_max = max(simlog_ee_rectifier_power_dissipated.Rectifier.D6.power_dissipated.series.values);

diodes = {'D1'; 'D2'; 'D3'; 'D4'; 'D5'; 'D6'};
PowerMax = [pd_D1_max;pd_D2_max;pd_D3_max;pd_D4_max;pd_D5_max;pd_D6_max];

T = table(PowerMax, 'RowNames', diodes)

T =

```

6×1 table

	PowerMax
D1	166.45
D2	166.45
D3	166.45
D4	166.45
D5	166.45
D6	166.45

The maximum instantaneous power dissipation for diodes D3 and D6 is the same as the maximum instantaneous power dissipation for the other diodes.

See Also

Functions

[ee_getEfficiency](#) | [ee_getPowerLossSummary](#) | [ee_getPowerLossTimeSeries](#)

Related Examples

- “Power-Loss Analysis of a Three-Phase Rectifier”
- “Examine the Simulation Data Logging Configuration of a Model” on page 7-19
- “Data Logging” (Simscape)
- “About the Simscape Results Explorer” (Simscape)

Choose a Simscape Electrical Function for an Offline Harmonic Analysis

In this section...
“Harmonic Distortion” on page 7-30
“Harmonic Analysis Functions” on page 7-30
“Evaluate Relative Overall Harmonic Distortion” on page 7-31
“Compare Harmonic Distortion to Standard Limits” on page 7-32
“Minimize Harmonic Distortion with Passive Filters” on page 7-32
“Verify the Results of an Online Harmonic Analysis” on page 7-33

Harmonic Distortion

Nonlinear loads create power distortion in the form of harmonics, that is, voltages and currents that are multiples of the fundamental frequency. Harmonic waveforms can result in energy losses through heat dissipation and in reduced power quality. They can also cause equipment to malfunction or to become damaged. Standards development organizations such as the Institute of Electrical and Electronics Engineers (IEEE) and the International Electrotechnical Commission (IEC) define the recommended limits for harmonic content in electric power systems.

Harmonic Analysis Functions

You can use the simulation and analysis functions in Simscape Electrical to perform an offline, that is post-simulation, analysis to examine harmonic distortion in your model. The `ee_plotHarmonics` function generates a bar chart. The `ee_getHarmonics` and `ee_calculateThdPercent` functions provide harmonic data in numerical form.

To decide which functions and workflows to use for your harmonic analysis, consider your goals. The table cross-references the harmonic functions with common harmonic analysis according to the data the function outputs and the task requires.

Goal	<code>ee_plotHarmonics</code>	<code>ee_getHarmonics</code>	<code>ee_calculateThdPercent</code>
Evaluate the relative overall harmonic distortion	<ul style="list-style-type: none"> • Bar chart of the percentage of fundamental magnitude • Fundamental peak value • Total harmonic distortion (THD) percentage 		
Compare the harmonic distortion to standard limits		<ul style="list-style-type: none"> • Fundamental frequency • Harmonic orders • Harmonic magnitudes 	Total harmonic distortion (THD) percentage
Determine the parameters for filtering harmonic distortion		<ul style="list-style-type: none"> • Fundamental frequency • Harmonic orders • Harmonic magnitudes 	

Evaluate Relative Overall Harmonic Distortion

Use this workflow for a high-level understanding of the waveform distortion in your power system.

- 1 Enable Simscape data logging.
- 2 Save the logged voltage or current data to a variable.
- 3 Use the `ee_plotHarmonics` function to generate a bar chart of harmonic percentages with the peak fundamental magnitude and the total harmonic distortion (THD) percentage displayed in the plot title.

Compare Harmonic Distortion to Standard Limits

Use this workflow to obtain values for evaluating the IEEE or IEC suitability of your power system.

- 1 Enable Simscape data logging.
- 2 Save the logged voltage or current data to a variable.
- 3 Use the `ee_getHarmonics` function to obtain the harmonic orders, the magnitude for each order, and the fundamental frequency.
- 4 Save the fundamental peak to a new variable.
- 5 Calculate the RMS voltage or current for each order.
- 6 Calculate the harmonic distortion percentage for individual harmonics.
- 7 Use the `ee_calculateThdPercent` function to obtain the total harmonic distortion (THD).
- 8 Compare the percentage data for each order and the THD percentage to the standard limits.

Minimize Harmonic Distortion with Passive Filters

Use this workflow to determine the parameters for filtering the distorted waveforms with passive filters. Use individual, series-tuned filters for specific harmonic orders. Use a single high-pass filter to filter higher orders.

- 1 Enable Simscape data logging.
- 2 Save the logged voltage or current data in a variable.
- 3 Use the `ee_getHarmonics` function to obtain the harmonic orders, the magnitude for each order, and the fundamental frequency.
- 4 Identify the harmonic orders that you want to filter.
- 5 For each filter:
 - a Specify the filter size, in terms of reactive power compensation, and specify the filter quality.
 - b Calculate the capacitor reactance at the tuned harmonic order.
 - c Calculate the filter capacitance.
 - d Calculate the inductor reactance at the tuned harmonic order.

- e Calculate the filter inductance.
- f Calculate the filter resistance.

Verify the Results of an Online Harmonic Analysis

You can examine harmonic distortion in your model online, that is during simulation, using the Simscape Spectrum Analyzer block. To verify the results from the Spectrum Analyzer block:

- 1 To determine the THD in your model, perform an online analysis. For information, see “Perform an Online Harmonic Analysis Using the Simscape Spectrum Analyzer Block” on page 7-34.
- 2 Use the `ee_getHarmonics` and `ee_calculateThdPercent` functions to determine the THD in your model.
- 3 Compare the THD values for the online and offline analyses. If the results differ, reconfigure the Spectrum Analyzer block.

See Also

Blocks

Spectrum Analyzer

Functions

`ee_calculateThdPercent` | `ee_getHarmonics` | `ee_plotHarmonics`

Related Examples

- “Harmonic Analysis of a Three-Phase Rectifier”
- “Perform an Online Harmonic Analysis Using the Simscape Spectrum Analyzer Block” on page 7-34
- “Data Logging” (Simscape)

Perform an Online Harmonic Analysis Using the Simscape Spectrum Analyzer Block

In this section...
“Harmonic Distortion” on page 7-34
“Prerequisite” on page 7-34
“Perform an Offline Harmonic Analysis” on page 7-35
“Perform an Online Harmonic Analysis” on page 7-38

Harmonic Distortion

Nonlinear loads create power distortion in the form of harmonics, that is, voltages and currents that are multiples of the fundamental frequency. Harmonic waveforms can result in energy losses through heat dissipation and in reduced power quality. They can also cause equipment to malfunction or to become damaged. Standards development organizations such as the Institute of Electrical and Electronics Engineers (IEEE) and the International Electrotechnical Commission (IEC) define the recommended limits for harmonic content in electric power systems.

This example shows how to examine harmonic distortion in your model using offline, that is after simulation, and online, that is during simulation, analyses. The offline analysis uses the Simscape Electrical harmonic analysis functions and helps you to determine configuration settings for, and verify the results of, the online analysis. The online analysis uses the Simscape Spectrum Analyzer block.

Prerequisite

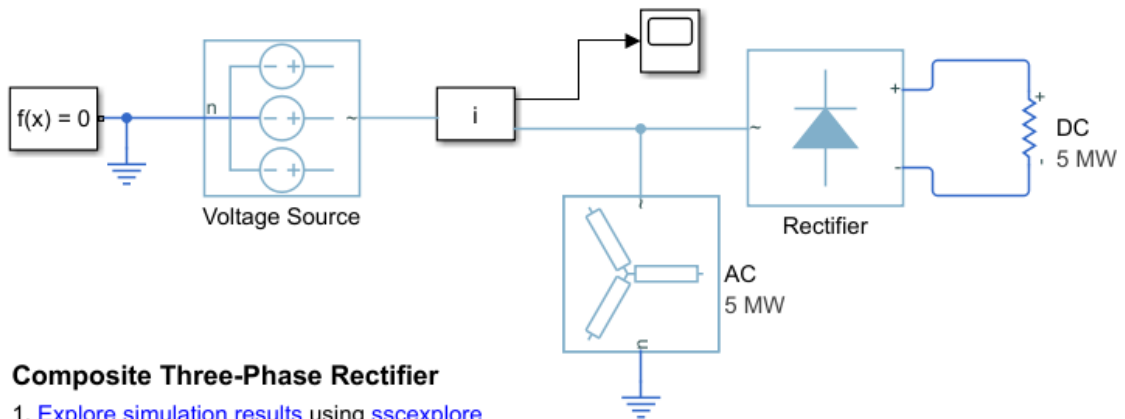
This example requires a simulation log variable in your MATLAB workspace. The model in this example is configured to log Simscape data for the whole model for the entire simulation time.

To learn how to determine if a model is configured to log simulation data, see “Examine the Simulation Data Logging Configuration of a Model” on page 7-19.

Perform an Offline Harmonic Analysis

- 1 Open the model. At the MATLAB command prompt, enter:

```
model = 'ee_composite_rectifier';
open_system(model)
```



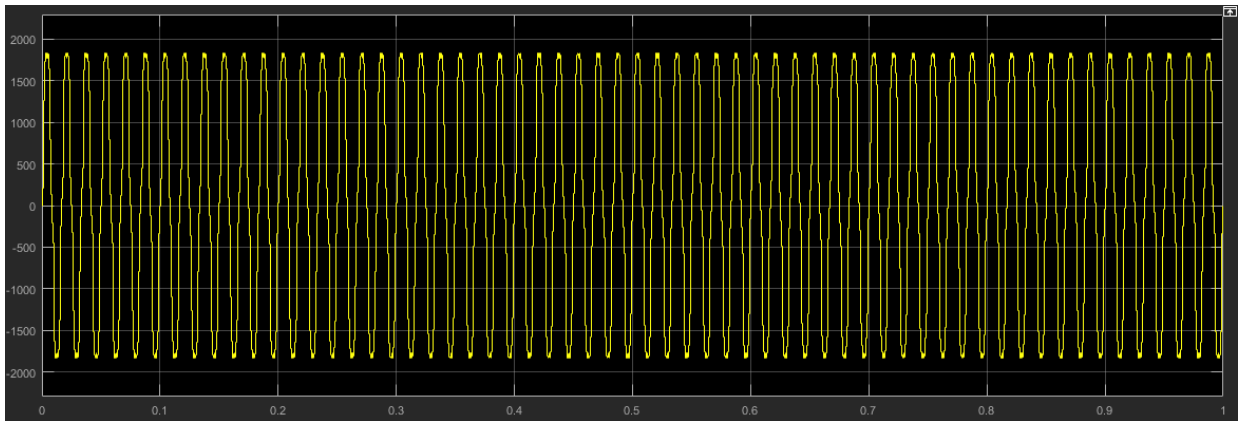
Composite Three-Phase Rectifier

1. [Explore simulation results](#) using [sscexplore](#)
2. [Learn more](#) about this example

The example model contains a three-phase rectifier. The model also contains a Selector block that outputs only the a -phase from three-phase current signal that it receives from the PS-Simulink Converter block.

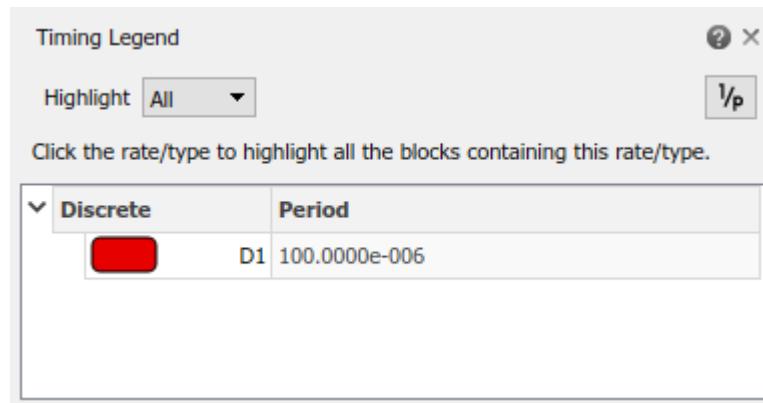
- 2 Simulate the model.


```
sim(model)
```
- 3 View the time-domain results. Open the Scope block.



The time domain analysis shows that the rectifier is converting the voltage, but it does not include any information about the frequencies in the signal.

- 4 Determine configuration settings and calculate the expected results for an online harmonic analysis. Perform an offline harmonic analysis.
 - a The Simscape Electrical harmonic analysis functions require that you use a fixed-step solver. Determine the solver type and sample time for the model. To turn on sample-time highlighting, in the Simulink editor menu bar, select **Display > Sample Time > All**.



The model is running at a discrete rate, therefore it is using a fixed-step solver, with a sample time of $1e-4$ s.

- b** Use the `ee_getHarmonics` function to calculate the harmonic order, the harmonic magnitude, and the fundamental frequency based on the voltage source currents.

```
[harmonicOrder,harmonicMagnitude,fundamentalFrequency] = ...
ee_getHarmonics(simlog_ee_composite_rectifier.Voltage_Source.I);
```

- c** Performing an online harmonic analysis using the Spectrum Analyzer block requires that you specify a value for maximum harmonic order and the resolution bandwidth (RBW). The RBW depends on the fundamental frequency.

Extract and display the maximum harmonic order and the fundamental frequency:

```
disp(['Maximum Harmonic Order = ', num2str(max(harmonicOrder))]
disp(['Fundamental Frequency = ', num2str(fundamentalFrequency)])
```

```
Maximum Harmonic Order = 30
Fundamental Frequency = 60
```

- d** Determine the peak value of the fundamental frequency. This value is useful for filtering out negligible harmonics and for verifying the results of the offline analyses.

```
fundamentalPeak = harmonicMagnitude(harmonicOrder==1);
disp(['Peak value of fundamental = ', num2str(fundamentalPeak), ' A']);
```

```
Peak value of fundamental = 1945.806 A
```

- e** Filter out small harmonics by identifying and keeping harmonics that are greater than one thousandth of the fundamental peak frequency.

```
threshold = fundamentalPeak ./ 1e3;
aboveThresold = harmonicMagnitude > threshold;
harmonicOrder = harmonicOrder(aboveThresold)';
harmonicMagnitude = harmonicMagnitude(aboveThresold)';
```

- f** Display the harmonic data in a MATLAB table.

```
harmonicRms = harmonicMagnitude./sqrt(2);
harmonicPct = 100.*harmonicMagnitude./harmonicMagnitude(harmonicOrder == 1);
harmonicTable = table(harmonicOrder,...
    harmonicMagnitude,...
    harmonicRms,...
    harmonicPct,...
    'VariableNames',{'Order','Magnitude','RMS','Percentage'});
display(harmonicTable);
```

```
harmonicTable =
```

10×4 table

Order	Magnitude	RMS	Percentage
1	1945.8	1375.9	100
5	218.86	154.75	11.248
7	105.83	74.835	5.439
11	85.135	60.2	4.3753
13	57.599	40.729	2.9602
17	50.417	35.65	2.5911
19	37.612	26.596	1.933
23	33.859	23.942	1.7401
25	26.507	18.743	1.3622
29	23.979	16.955	1.2323

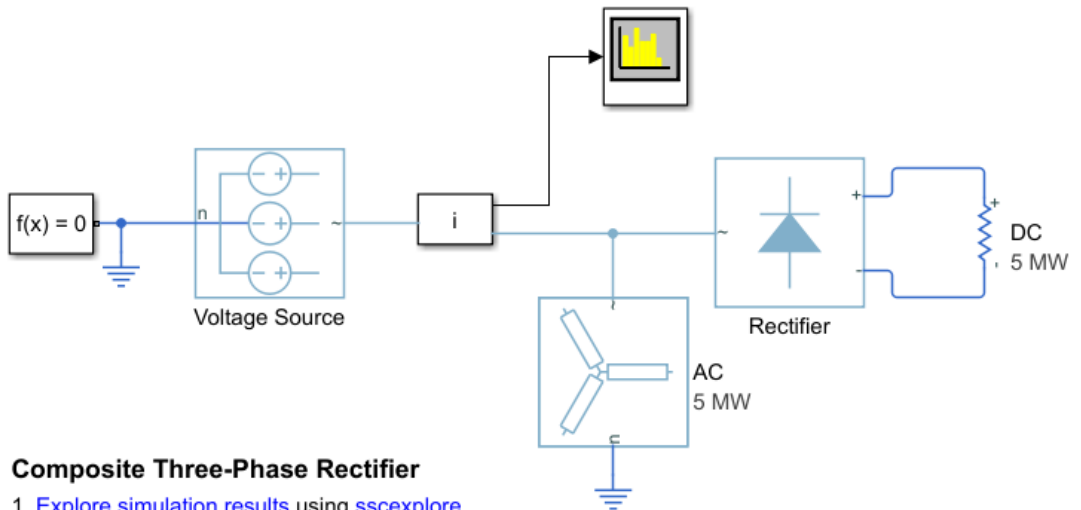
- g** Calculate the total harmonic distortion (THD) percentage using the `ee_calculate_ThdPercent` function.

```
thdPercent = ee_calculateThdPercent(harmonicOrder,harmonicMagnitude);
disp(['Total Harmonic Distortion Percentage = ' num2str(thdPercent), '%']);
```

Total Harmonic Distortion percentage = 14.1721 %


Perform an Online Harmonic Analysis

- 1** In the Simulink editor that contains the `ee_composite_rectifier` model, replace the Scope block with a Spectrum Analyzer block from the Simscape Utilities Library:
 - a** Delete the Scope block.
 - b** Left-click within the block diagram.
 - c** After the search icon appears, type `spec`, and then from the list, select the Spectrum Analyzer from the Utilities library.
 - d** Connect the Spectrum Analyzer block to the output signal from the Subsystem i.



Composite Three-Phase Rectifier

1. [Explore simulation results](#) using [sscexplore](#)
2. [Learn more](#) about this example

- 2 Configure the Spectrum Analyzer block using the Spectrum Settings panel.
 - a Open the Spectrum Analyzer.
 - b Open the Spectrum Settings panel. On the Spectrum Analyzer toolbar, click the **Spectrum Settings**  button.
 - c Configure the parameters on the **Main Options** pane.
 - i Configure the block to display the root mean square (RMS) of the frequency. From the **Type** dropdown menu, select RMS.
 - ii Determine the value to specify for the resolution bandwidth (RBW) using this equation:

$$RBW = \frac{NENBW * f}{N},$$

where,



- $NENBW$ is the normalized effective noise bandwidth, a factor of the windowing method used. The Hanning (Hann) window has an $NENBW$ value of approximately 1.5.

- f is the fundamental frequency.
- N is the number of periods.
- RBW is the resolution bandwidth in Hz.

For a fundamental frequency of 60 Hz over 10 periods, using a Hann window,

$$RBW = \frac{1.5 * 60Hz}{10} = 9Hz$$

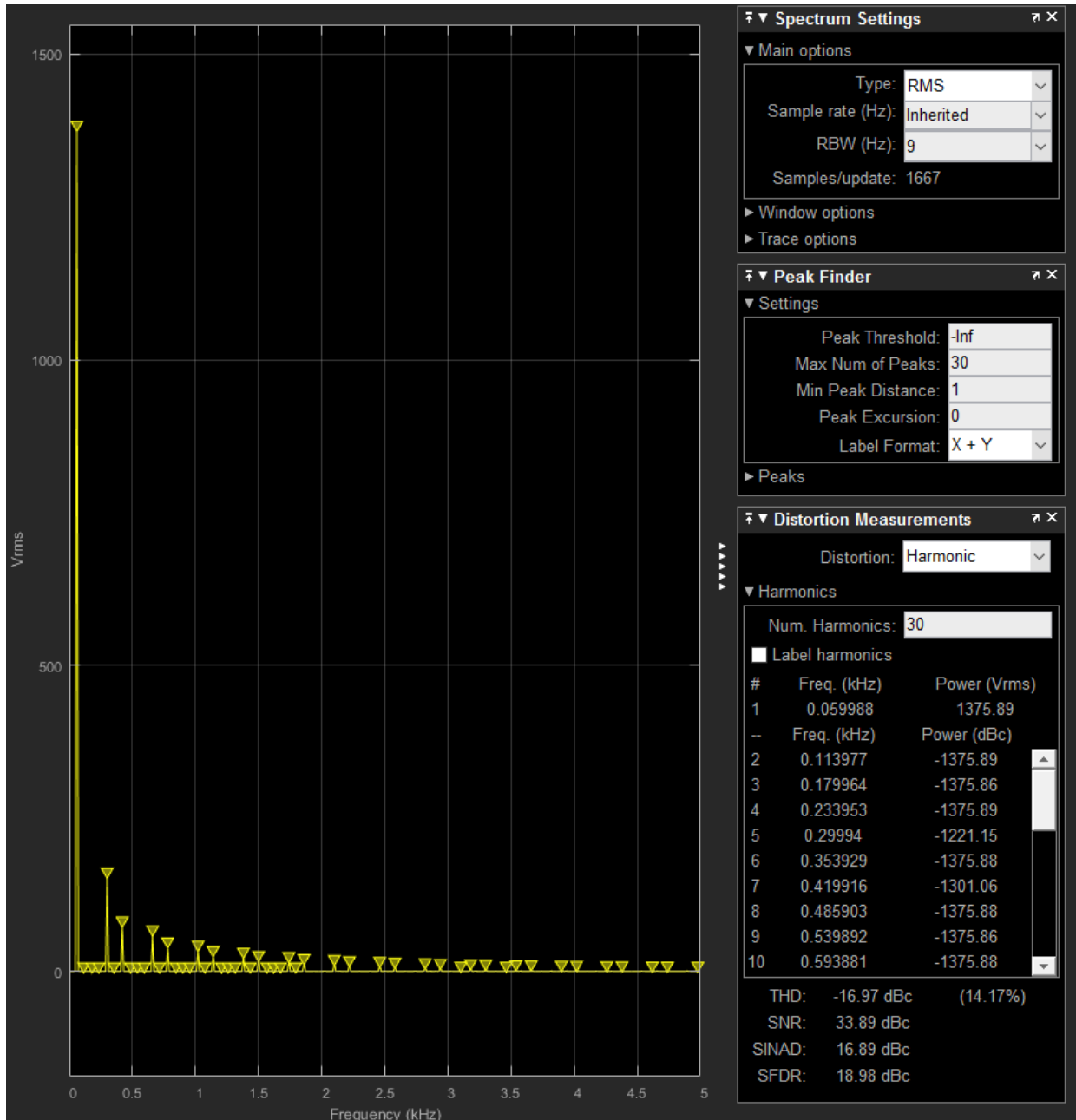
For **RBW (Hz)**, specify 9.

- d** Expand the **Windows Options** pane and specify an **Overlap (%)** of 90.
- e** Specify the maximum number of peaks for the analyzer to display. In the menu bar, select **Tools > Measurements > Peak Finder**. Alternatively, in the Spectrum Analyzer toolbar, select the Peak Finder  button. In the **Peakfinder** pane, in the **Settings** section, for **Max Num of Peaks**, enter 30. This value is based on the maximum harmonic order as indicated by the offline analysis.
- f** Set the number of harmonics to use for measuring harmonic distortion. Specify a number that captures the largest harmonic order that the offline analysis captures. In the menu bar, select **Tools > Measurements > Distortion Measurements**. Alternatively, in the Scope toolbar, click the Distortion Measurements  button. Scroll as required to see the **Distortion Measurements** pane.

In the **Distortion Measurements** pane, for **Num Harmonics**, again enter 30.

- 3** Simulate the model.

```
sim(model)
```



The THD percentage is 14.17% and the fundamental peak power is 1375.89 Vrms at 0.06 kHz (60 Hz). These results agree with the results from the offline harmonic analysis.

See Also

Blocks

PS-Simulink Converter | Selector | Spectrum Analyzer

Functions

ee_calculateThdPercent | ee_getHarmonics | ee_plotHarmonics

Related Examples

- “Harmonic Analysis of a Three-Phase Rectifier”
- “Choose a Simscape Electrical Function for an Offline Harmonic Analysis” on page 7-30
- “Data Logging” (Simscape)

Real-Time Simulation

Prepare Simscape Electrical Models for Real-Time Simulation Using Simscape Checks

If you have a Simulink Real-Time license, you can optimize your model for real-time execution using the `Execute real-time application` activity mode in the Simulink Performance Advisor. This mode includes several checks specific to physical models. For example, the Simulink Performance Advisor identifies Simscape Solver Configuration blocks with settings that are suboptimal for real-time simulation. For optimal results, Solver Configuration blocks should have the **Use local solver** and **Use fixed-cost runtime consistency iterations** options selected.

The checks are organized into folders. You can use the checks in the **Simscape checks** folder for all physical models. Subfolders contain checks that target blocks from Simscape Electrical and other add-on products such as Simscape Driveline and Simscape Multibody.

Before you run the checks, use the processes described in “Real-Time Model Preparation Workflow” (Simscape), “Real-Time Simulation Workflow” (Simscape), and “Hardware-In-The-Loop Simulation Workflow” (Simscape).

To run the Simulink Real-Time Performance Advisor Checks:

- 1 In the Simulink Editor menu bar, select **Analysis > Performance Tools > Performance Advisor**.
- 2 In the Performance Advisor window, under **Activity**, select `Execute real-time application`.
- 3 In the left pane, expand the **Real-Time** folder, and then the **Simscape checks** folder.
- 4 Run the top-level Simscape checks and the Simscape Electrical checks. If your model contains blocks from other add-on products, also run the checks in the subfolder corresponding to that product.

See Also

More About

- “Model Preparation Objectives” (Simscape)
- “Real-Time Model Preparation Workflow” (Simscape)

- “Real-Time Simulation Workflow” (Simscape)
- “Use Performance Advisor to Improve Simulation Efficiency” (Simulink)
- “Create and Use Code Generation Reports” (HDL Coder)

Simscape to HDL Workflow

- “Generate HDL Code from Simscape Models” on page 9-2
- “Deploy Simscape™ Plant Models to Speedgoat FPGA I/O Modules” on page 9-13
- “Troubleshoot Conversion of Simscape DC Motor Control to HDL-Compatible Simulink Model” on page 9-26
- “Troubleshoot Conversion of Simscape™ Permanent Magnet Synchronous Motor to HDL-Compatible Simulink Model” on page 9-36
- “Validate HDL Implementation Model to Simscape Algorithm” on page 9-53
- “Improve Sampling Rate of HDL Implementation Model Generated from Simscape Algorithm” on page 9-58

Generate HDL Code from Simscape Models

This example uses a halfwave rectifier model to illustrate how you can develop your plant model in Simscape™ and use Simscape HDL Workflow Advisor to generate HDL code for your model.

Why Generate HDL Code?

To perform hardware-in-the-loop (HIL) simulation with smaller timesteps and increased accuracy, you can deploy the plant models to the FPGAs on board the Speedgoat I/O modules. By using the Simscape HDL Workflow Advisor, you can generate an HDL implementation model. You can then generate HDL code for the implementation model and deploy the generated code onto the FPGA platforms. Using this capability, you can model and deploy complex physical systems in Simscape that previously took long time to model by using Simulink™ blocks.

Simscape Example models for HDL Code generation

For HDL code generation, you can design your own Simscape algorithm or choose from a list of example models that are created in Simscape. The example models include:

- Boost Converter
- Bridge Rectifier
- Buck Converter
- Halfwave Rectifier
- Three Phase Rectifier
- Two Level Converter Ideal
- Two Level Converter Igbt

All examples are prefixed with `sschdlex` and post fixed with `Example`. For example, to open the Boost Converter model, in the MATLAB™ command window, enter:

```
load_system('sschdlexBoostConverterExample')  
open_system('sschdlexBoostConverterExample/Simscape_system')
```

Restrictions for HDL Code Generation from Simscape Models

HDL Coder™ does not support code generation from Simscape networks that contain:

- Events

- Mode charts
- Delays
- Runtime parameters
- Periodic sources
- Simscape™ Multibody™ blocks
- Simscape Electrical Specialized Power Systems blocks

Guidelines for Modeling Simscape for HDL Compatibility

1. Create a Simscape model by using switched linear blocks. Add Simulink-PS Converter blocks at the input ports and PS-Simulink Converter blocks at the output ports.
2. Configure the solver options for HDL code generation by using a Solver Configuration block. In the block parameters of this block:
 - Select **Use local solver**.
 - Use **Backward Euler** as the **Solver type**.
 - Specify a discrete sample time, T_s .
3. Enclose the blocks inside a Subsystem and provide the test inputs.
4. Configure the model for HDL code generation by running the `hdlsetup` function. `hdlsetup` configures the solver settings such as using a fixed-step solver, specifies the simulation start and stop times, and so on. To run the command for your `current_model`:

```
hdlsetup('current_model')
```

5. Verify Simscape model compatibility by using the `simscape.findNonLinearBlocks` function. This function detects the nonlinear blocks in your Simscape model. Provide the path to your Simscape model as an argument to this function. It returns the names of nonlinear blocks.

For example: To verify presence of nonlinear blocks in Half Wave Rectifier Model, in the MATLAB command window, enter:

```
simscape.findNonlinearBlocks('sschdlexHalfWaveRectifierExample')
```

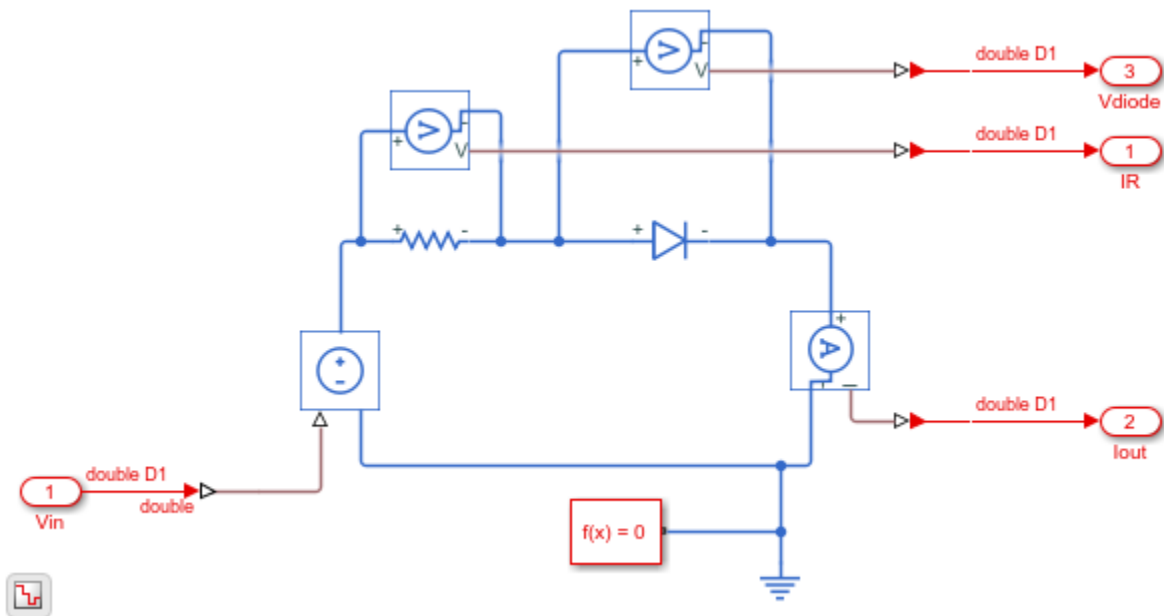
The Halfwave Rectifier Model

To open the half-wave rectifier model, in the MATLAB Command Window, enter:

```
open_system('sschdlexHalfWaveRectifierExample')
```

Save this model locally as `HalfWaveRectifier_HDL` to run the workflow.

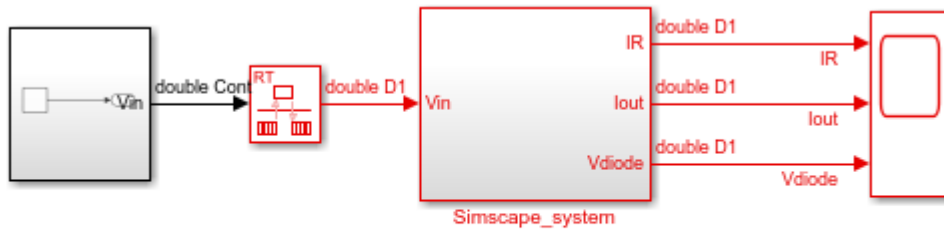
```
load_system('HalfWaveRectifier_HDL')
open_system('HalfWaveRectifier_HDL/Simscape_system')
set_param('HalfWaveRectifier_HDL', 'SimulationCommand', 'update');
```



The Simscape model uses switched linear blocks such as Diode and Resistor to model the design. The model has Simulink-PS Converter blocks at the input port and PS-Simulink converter blocks at the output ports. To verify that the solver settings are configured correctly, open the Solver Configuration block.

At the top level of the model, you see a `Simscape_system` block that models the half-wave rectifier algorithm. The model accepts a Sine Wave input, uses a Rate Transition block to discretize the continuous time input, and has a Scope block that calculates the output. To see the input stimulus and the output from the model, connect the Sine Wave input to the Scope block.

```
open_system('HalfWaveRectifier_HDL')
```



Copyright 2018 The MathWorks, Inc.

To configure the half-wave rectifier model for HDL compatibility, in the MATLAB command window, enter:

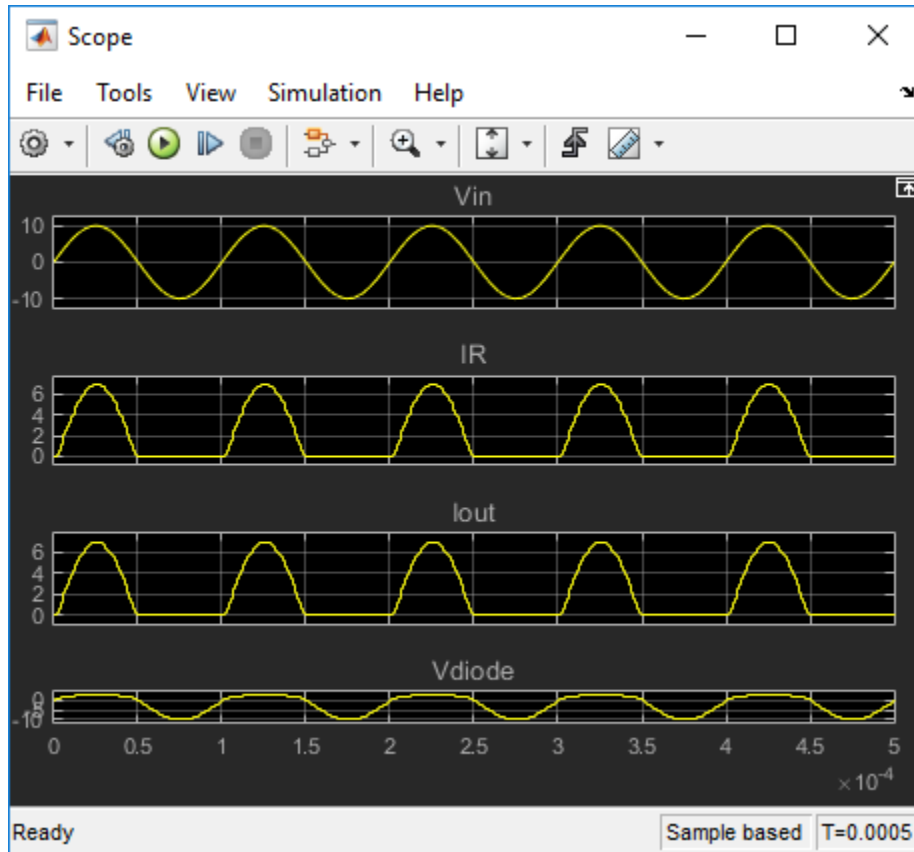
```
hdlsetup('HalfWaveRectifier_HDL')
```

Simulate and Verify Functionality of Simscape Algorithm

To see the simulation results, simulate the model and then open the Scope block.

```
sim('HalfWaveRectifier_HDL')
```

This figure shows simulation results with the sine wave input and the outputs from Simscape_system.



Open Simscape HDL Workflow Advisor

To generate an HDL implementation model from which you can generate code, use the Simscape HDL Workflow Advisor. To open the Advisor, run this command:

```
sschdladvisor('HalfWaveRectifier_HDL')
```

Updating Model Advisor cache...

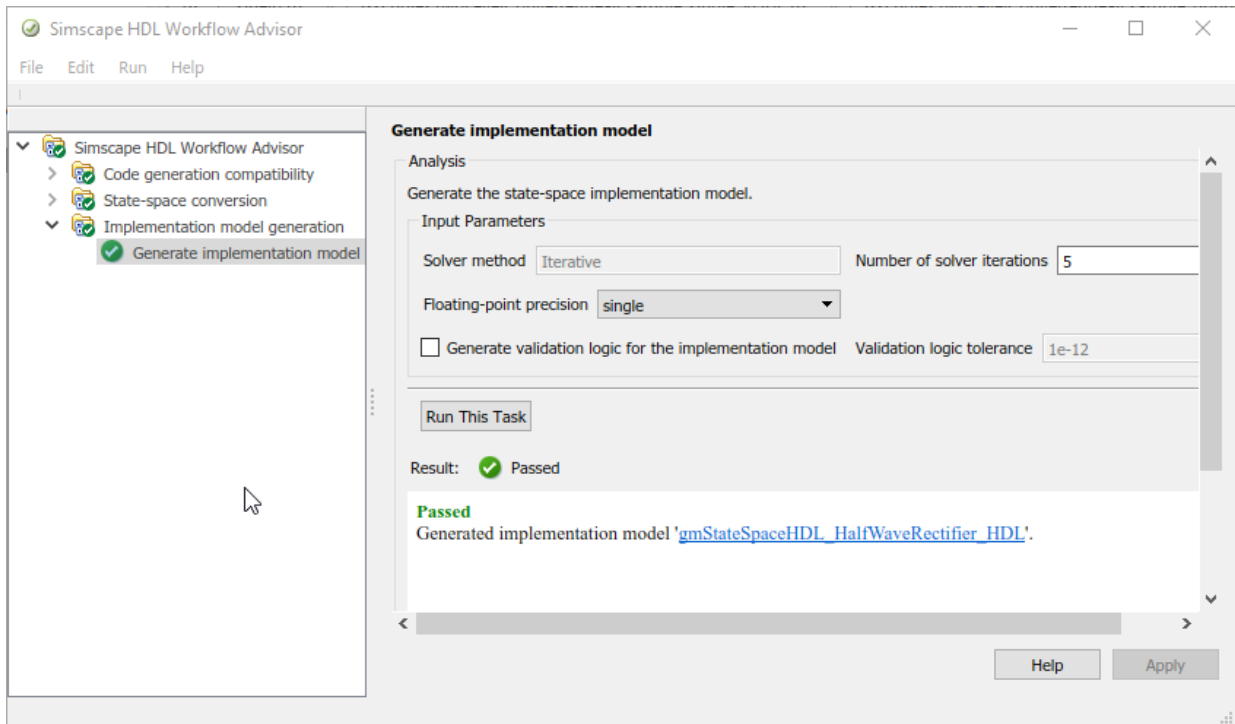
Model Advisor cache updated. For new customizations, to update the cache, use the Advisor...

This updates the model advisor cache and opens the Simscape HDL Workflow Advisor. To learn more about the Simscape HDL Workflow Advisor and the various tasks, right-click that folder or task, and select **What's This?**.

See also Simscape HDL Workflow Advisor Tasks.

Run Simscape HDL Workflow Advisor

To run the workflow, in the Simscape HDL Workflow Advisor, right-click the **Generate implementation model** task and select **Run to Selected Task**.



If the task passes, you see a link to the implementation model.

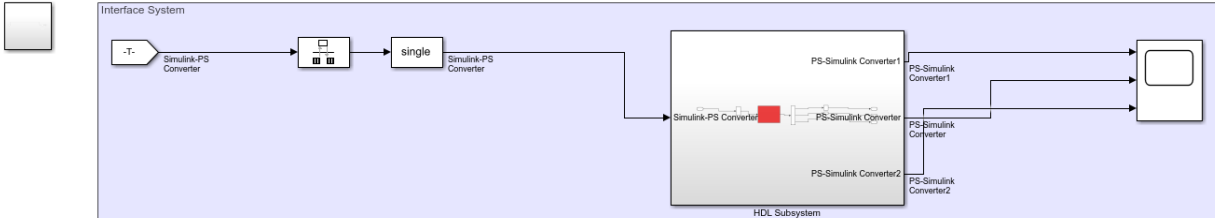
In some cases, your Simscape algorithm may not be compatible for generating an implementation model using the Simscape HDL Workflow Advisor. In such cases, running certain tasks in the Advisor can result in the task to fail. To learn how you can make it HDL compatible, see

- Troubleshoot Conversion of Simscape DC Motor Control to HDL-Compatible Simulink Model.
- Troubleshoot Conversion of Simscape Permanent Magnet Synchronous Motor to HDL-Compatible Simulink Model.

Open HDL Implementation Model

To see the implementation model, in the **Generate implementation model task**, click the link.

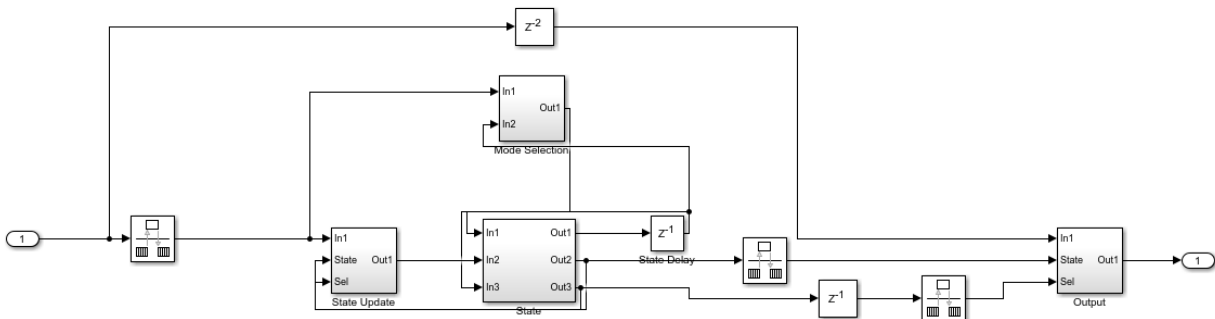
```
open_system('gmStateSpaceHDL_HalfWaveRectifier_HDL')
```



The model contains two subsystems. The Subsystem block contains the Simscape algorithm that you modeled. From and Goto blocks inside this Subsystem provide the same Sine Wave input to the HDL Subsystem.

The HDL Subsystem models the state-space representation that you generated from the Simscape model. The ports of this Subsystem use the same name as the Simulink-PS Converter and PS-Simulink Converter blocks in your original Simscape model. If you navigate inside this Subsystem, you see several delays, adders, and Matrix Multiply blocks that model the state-space equations.

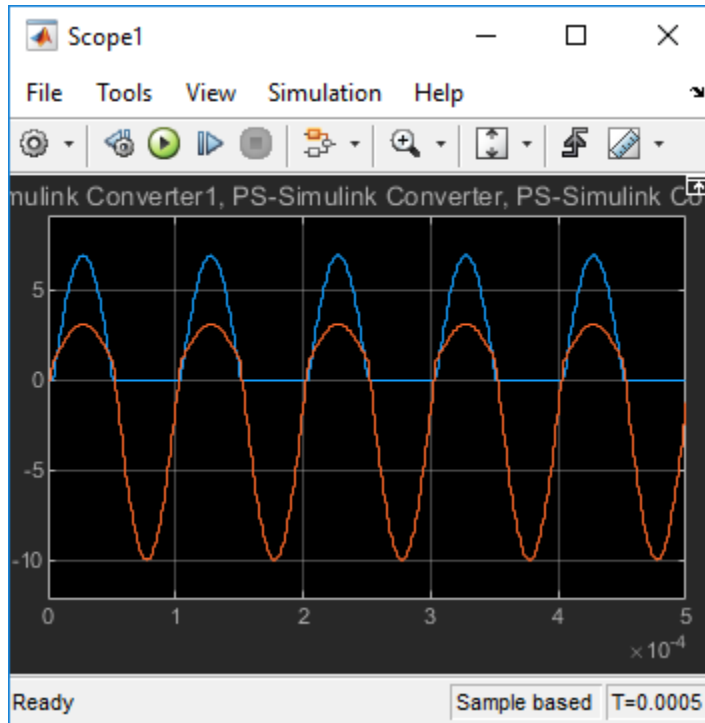
```
open_system('gmStateSpaceHDL_HalfWaveRectifier_HDL/HDL Subsystem/HDL Algorithm')
```



To simulate the HDL Implementation model, enter this command:

```
sim('gmStateSpaceHDL_HalfWaveRectifier_HDL')
```


Open the scope block to view results.



The simulation results from the HDL implementation model matches that of the original plant model. Therefore, we can verify that the plant simulation model is correctly transformed into an HDL implementation model.

HDL code is generated for the HDL Subsystem block inside this model.

Generate HDL Code and Validation Model

The HDL model and subsystem parameter settings are saved using this command:

```
hdlsaveparams('gmStateSpaceHDL_HalfWaveRectifier_HDL');
```

```
%% Set Model 'gmStateSpaceHDL_HalfWaveRectifier_HDL' HDL parameters
hdlset_param('gmStateSpaceHDL_HalfWaveRectifier_HDL', 'FloatingPointTargetConfiguration', 'on');
hdlset_param('gmStateSpaceHDL_HalfWaveRectifier_HDL', 'MaskParameterAsGeneric', 'on');
hdlset_param('gmStateSpaceHDL_HalfWaveRectifier_HDL', 'Oversampling', 100);
```

```
% Set SubSystem HDL parameters
hdlset_param('gmStateSpaceHDL_HalfWaveRectifier_HDL/HDL Subsystem', 'FlattenHierarchy')
```

The model uses single data types and generates HDL code in native floating-point mode. Floating-point operators can introduce delays. Because the design contains feedback loops, for the model transformation advisor to allocate enough delays for the operators inside the feedback loops, the model uses clock-rate pipelining in conjunction with a large value for the **Oversampling factor**. An **Oversampling factor** of 100 and the clock-rate pipelining optimization is saved on this model.

For more information, see:

- Clock-Rate Pipelining
- Oversampling Factor
- Allocate Sufficient Delays for Floating-Point Operations

Before you generate HDL code, it is recommended to enable generation of the validation model. The validation model compares the output of the generated model after code generation and the original model. To learn more, see [Generated Model and Validation Model](#).

Run these commands to save validation model generation settings on your Simulink model:

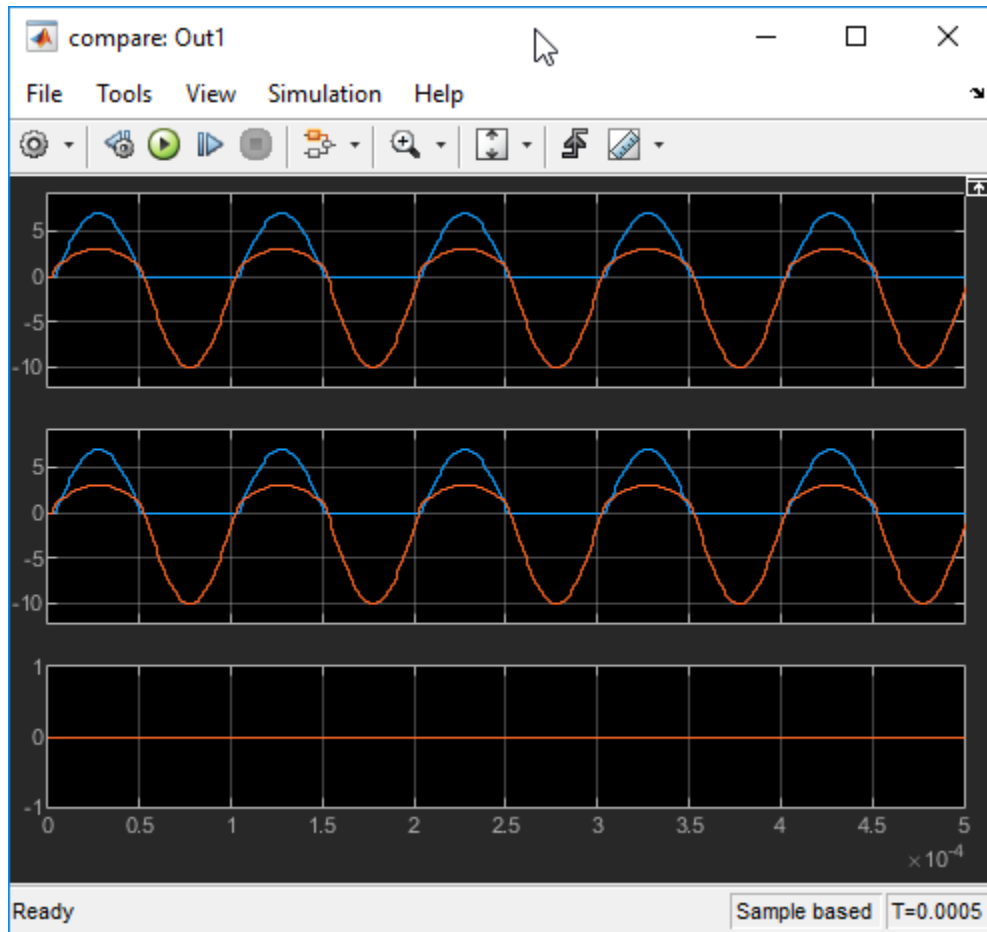
```
HDLmodelName = 'gmStateSpaceHDL_HalfWaveRectifier_HDL';
hdlset_param(HDLmodelName, 'TargetDirectory', 'C:/Temp/hdlsrc');
hdlset_param(HDLmodelName, 'GenerateValidationModel', 'on');
```

To generate HDL code, run this command:

```
makehdl('gmStateSpaceHDL_HalfWaveRectifier_HDL/HDL Subsystem');
```

The generated HDL code and validation model is saved in C:/Temp/hdlsrc directory. The generated code is saved as HDL_Subsystem_tc.vhd. To open the validation model, click the link to gm_gmStateSpaceHDL_HalfWaveRectifier_HDL_vnl.slx in the code generation logs in the Command Window.

Open the Compare block at the output of HDL_Subsystem_vnl Subsystem of the validation model. Then, open the Assert_Out1 block. To see the simulation results after HDL code generation, open the Compare: Out1 Scope block:



The top graph represents the output of our generated model, and the middle graph represents the output of the implementation model. Since the output generated by both the models are exactly matching, the error between them is zero, which is represented in the last graph.

Optionally, you can deploy the HDL code on a Hardware platform. For more information, see [Deploy Simscape plant models to Speedgoat FPGA IO modules](#).

See Also

Functions

`checkhdl` | `makehdl`

More About

- [“Getting Started with Simscape Electrical”](#)
- [“Troubleshoot Conversion of Simscape DC Motor Control to HDL-Compatible Simulink Model”](#) (HDL Coder)
- [“Validate HDL Implementation Model to Simscape Algorithm”](#) (HDL Coder)

Deploy Simscape™ Plant Models to Speedgoat FPGA I/O Modules

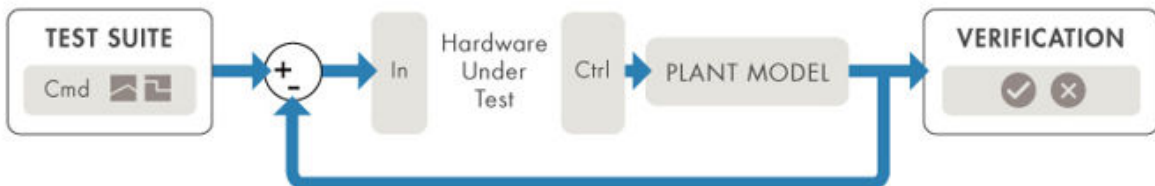
This example shows how to deploy the Simscape plant models on Speedgoat FPGA I/O modules by using the HDL Workflow Advisor. This is a two-step process.

- 1 Develop the Simscape model and convert it into an implementation model by using the Simscape HDL Workflow Advisor. HDL code is generated from this implementation model. For more information, see [Generate HDL Code from Simscape Models](#).
- 2 Deploy HDL code to a Speedgoat I/O module by using the HDL Workflow Advisor.

Why Deploy a Simulink Model to Speedgoat FPGA Modules

You can use the HDL Workflow Advisor to deploy the Simulink™ model to Speedgoat FPGA I/O modules. Simulating the plant model on the FPGA provides:

- **Real-time Simulation:** Hardware-in-the-loop provides real-time simulation of your Simscape plant model.



- **Hardware Acceleration:** The speed of simulating physical systems increases by implementing it on hardware as reconfigurable FPGAs provide rapid hardware prototyping. You can use this capability to model complex physical systems.

Set Up and Configuration

To deploy the Simscape plant models on Speedgoat FPGA modules:

1. **Install Xilinx Vivado®**

Speedgoat FPGA IO333-325K uses Xilinx Vivado. If it is not already present, install Xilinx Vivado v2018.2. Then, set the tool path to the installed Xilinx Vivado 2018.2 executable. To set the tool path, use the `hdlsetuptoolpath` function.

```
hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath','C:\Xilinx\Vivado\2018.2\bin\vi
```

2. Set Up I/O Module

To run the simulation of the Simscape plant model in real time on hardware, you must set up the I/O module. For information on setting up the I/O module, see Xilinx HDL Software for Speedgoat I/O Hardware.

HDL Workflow Advisor

The HDL Workflow Advisor guides you through the stages of generating HDL code for a Simulink subsystem and the FPGA design process, such as:

- Checking the model for HDL code generation compatibility and automatically fixing incompatible settings.
- Generating HDL code, a test bench, and scripts to build and run the code and test bench.
- Synthesis and timing analysis through integration with third-party synthesis tools.
- Completing the automated workflows for deployment on hardware platforms such as System-on-Chip(SoC), FPGAs, and Speedgoat I/O modules.

This example shows how to use the HDL Workflow Advisor to deploy HDL code on Speedgoat IO333-325K module that uses Xilinx Vivado. For example, to open the HDL Workflow Advisor for a Subsystem inside the model, enter:

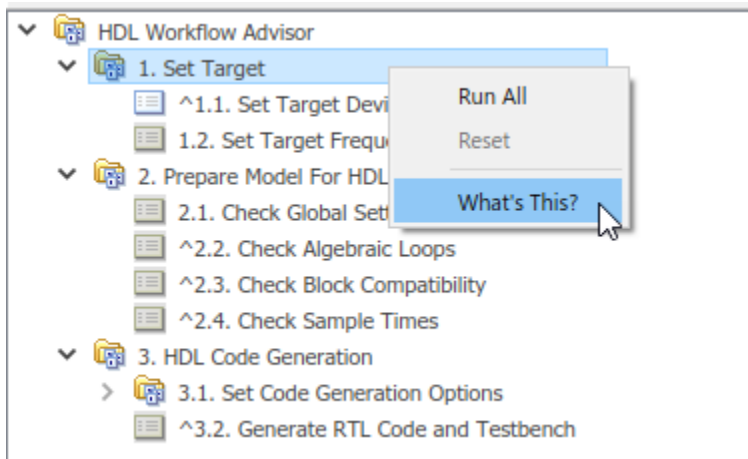
```
load_system('sschdlexTwoLevelConverterIgbtExample')  
hdladvisor('sschdlexTwoLevelConverterIgbtExample/Simscape_system')
```

For more information, see `hdladvisor`.

In the HDL Workflow Advisor, the left pane lists the folders in the hierarchy. Each folder represents a group or category of related tasks. Expanding the folders shows the available tasks in each folder. From the left pane, you can select a folder or an individual task. The HDL Workflow Advisor displays information about the selected folder or task in the right pane. The contents of the right pane depends on the selected folder or task. For some tasks, the right pane contains simple controls for running the task and a display area for status messages and other task results. For other tasks that involve setting code

or test bench generation parameters, the right pane displays several parameter and option settings.

To learn more about each individual task, right-click that task, and select **What's This?**.



For more information, see Getting Started with the HDL Workflow Advisor.

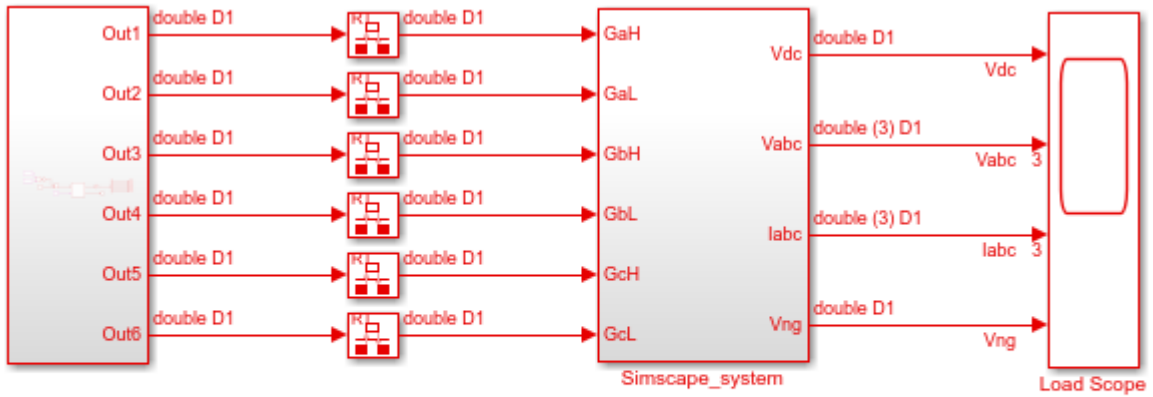
Two Level Ideal Converter Model

This example uses a Two-Level Ideal converter Simscape plant model. To open this model, enter:

```
open_system('sschdlexTwoLevelConverterIdealExample')
```

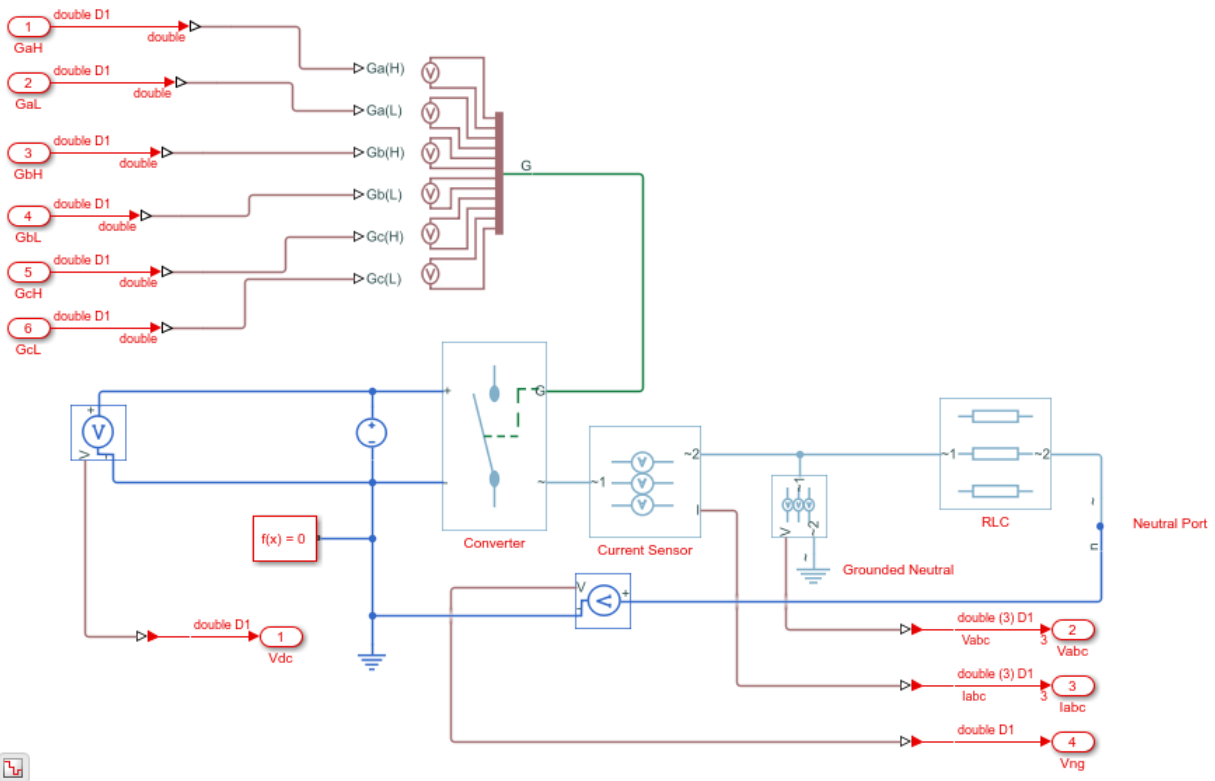
Save this model locally as TwoLevelConverter_HDL.slx to run this workflow.

```
open_system('TwoLevelConverter_HDL')
set_param('TwoLevelConverter_HDL', 'SimulationCommand', 'update')
```



Copyright 2018 The MathWorks, Inc.

```
open_system('TwoLevelConverter_HDL/Simscape_system')
```

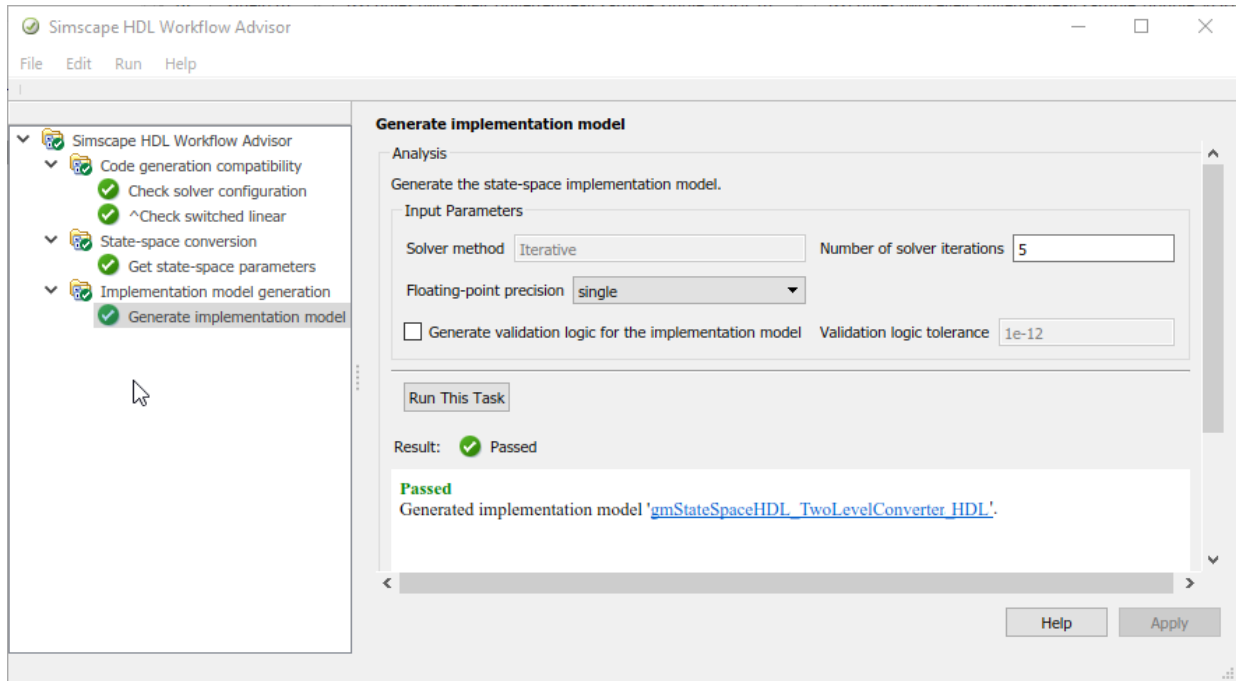
The Simscape subsystem receives six-switch controlling pulses as input. The Simscape subsystem acts as a generator that uses a two-level, carrier-based PWM method to:

- 1 Sample a reference wave.
- 2 Compare the sample to a triangular carrier wave.
- 3 Generate a switch-on pulse if a sample is higher than the carrier signal or a switch-off pulse if a sample is lower than the carrier wave.

Generate HDL Implementation Model

To generate an implementation model, use the Simscape HDL Workflow Advisor. Enter:

```
sschdladvisor('TwoLevelConverter_HDL')
```



To generate the implementation model, in the Simscape HDL Workflow Advisor, keep the default settings for all the tasks, and then run the tasks. You see a link to the model in the **Generate implementation model** task.

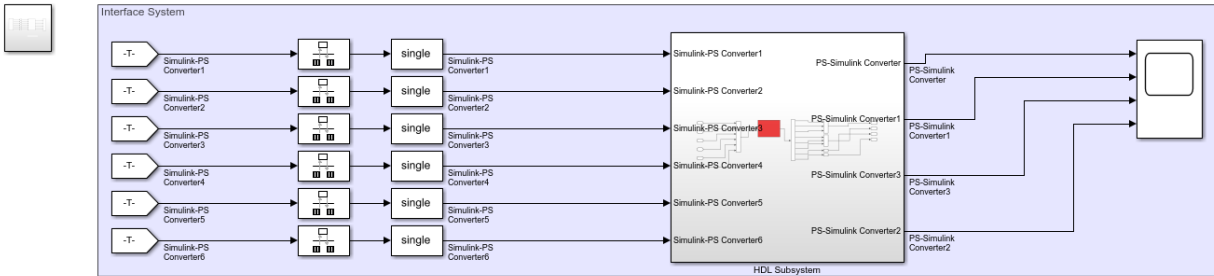
To learn more about the Simscape HDL Workflow Advisor, see:

- `sschldadvisor`
- Generate HDL code from Simscape Models

The Implementation Model

To open the implementation model, enter:

```
open_system('gmStateSpaceHDL_TwoLevelConverter_HDL')
```



The model contains two subsystems. The HDL Subsystem models the state-space representation that you generated from the Simscape model. The ports of this subsystem use the same name as the Simulink-PS Converter and PS-Simulink Converter blocks that you use in your original Simscape model. If you navigate inside this Subsystem, you see several delays, adders, and Matrix Multiply blocks that model the state-space equations. From and Goto blocks inside this subsystem provide the same input as that of the original model to the HDL Subsystem.

Deploy Two Level IGBT Converter Model to Speedgoat IO333-325K Module

This example shows how to deploy the implementation model of Two Level IGBT Converter to Speedgoat IO333-325K FPGA module by using the HDL Workflow Advisor. The Speedgoat IO333 FPGA module uses Xilinx Vivado and IP Core Generation Infrastructure. Before you run the Workflow Advisor, make sure that you have specified the path to the installed Xilinx Vivado executable.

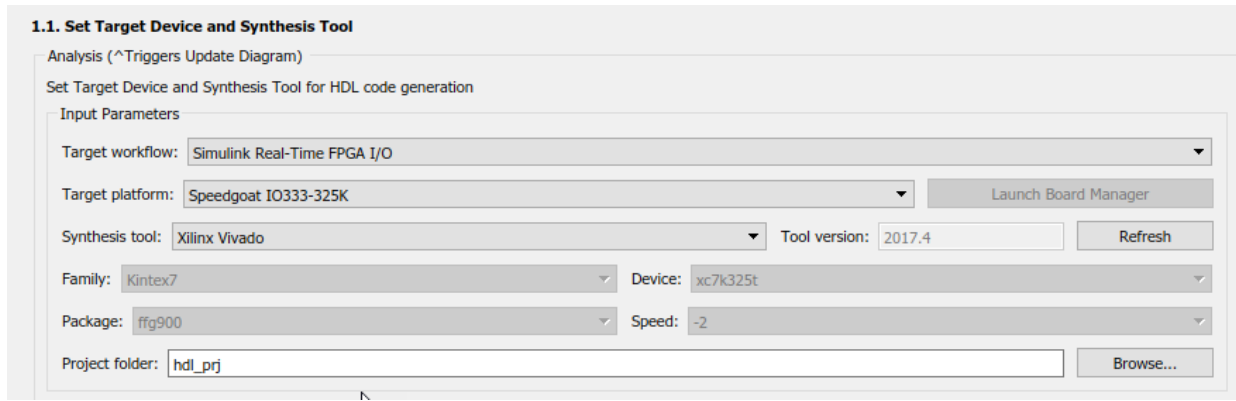
1. Open HDL Workflow Advisor

To open the HDL Workflow Advisor for the Implementation model, enter:

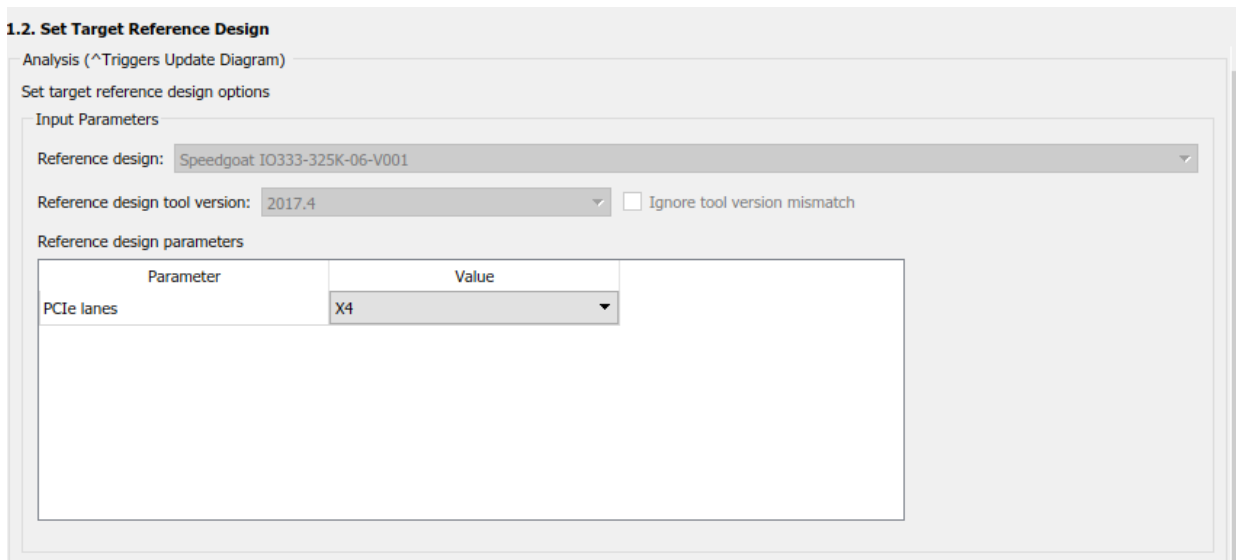
```
hdladvisor('gmStateSpaceHDL_TwoLevelConverter_HDL/HDL Subsystem')
```

2. In **Set Target Device and Synthesis Tool** task, set these parameters and select **Run This Task**:

- **Target workflow** as Simulink Real-Time FPGA I/O
- **Target platform** as Speedgoat IO333-325K
- **Synthesis tool** as Xilinx Vivado



3. In **Set Target Reference Design** task, select a value of x4 for the parameter PCIe lanes, and select **Run This Task**.



4. In **Set Target Interface** task, map the Input and Output single data type ports to PCIe Interface and select **Run This Task**.

1.3. Set Target Interface

Analysis (^Triggers Update Diagram)

Set target interface for HDL code generation

Input Parameters

Processor/FPGA synchronization: Free running

Target platform interface table

Port Name	Port Type	Data Type	Target Platform Interfaces	Bit Range / Address / FPGA Pin
Simulink-PS Converte...	Inport	single	PCIe Interface	x"100"
Simulink-PS Converte...	Inport	single	PCIe Interface	x"108"
Simulink-PS Converte...	Inport	single	PCIe Interface	x"104"
Simulink-PS Converte...	Inport	single	PCIe Interface	x"10C"
Simulink-PS Converte...	Inport	single	PCIe Interface	x"110"
Simulink-PS Converte...	Inport	single	PCIe Interface	x"114"
PS-Simulink Converter	Output	single (3)	PCIe Interface	x"120"
PS-Simulink Converte...	Output	single (3)	PCIe Interface	x"140"
PS-Simulink Converte...	Output	single	PCIe Interface	x"118"

5. In **Set Target Frequency** task, select a target frequency that is within the range. If the target frequency is set to higher values, it results in a failure to generate the bitstream when you run task **Build FPGA Bitstream**. This example has **Target Frequency** set to 50 MHz.

1.4. Set Target Frequency

Analysis

Set Target Frequency

Input Parameters

Target Frequency (MHz): 50

Default (MHz): 100

Frequency Range (MHz): 10-200

6. Right-click **Generate RTL Code and IP Core** task and select **Run to Selected Task**. This step generates a warning if the model uses vector data types. Click the link in the warning, select **Scalarize vector ports**, and rerun the task.
7. Run the workflow to the **Generate Simulink Real-Time interface** task. In **Create Project** task, you can open the Vivado project and see the implemented design.

4.1. Create Project

Analysis

Create project for embedded system tool

Input Parameters

Embedded system tool:

Project folder:

Synthesis objective:

Enable IP caching

Result: ✔ Passed

Passed Create Project.

```

Task "Create Project" successful.
Generated logfile: hdl\_prj\hdlsrc\qmStateSpaceHDLTwoLevelConverter\HDL\workflow\_task\_CreateProject.log
Generating Xilinx Vivado with IP Integrator project: hdl\_prj\vivado\_ip\_prj\vivado\_prj.xpr
***** Vivado v2017.4 (64-bit)
**** SW Build 2086221 on Fri Dec 15 20:55:39 MST 2017
**** IP Build 2085800 on Fri Dec 15 22:25:07 MST 2017
** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.

source vivado_create_prj.tcl
# create_project vivado_prj {} -part xc7k325tffg900-2 -force
# set_property target_language VHDL [current_project]
# set defaultRepoPath ./ipcore

```

8. When the **Generate Simulink Real-Time interface** task passes, you see a link to open the Simulink Real-Time Interface Model. Select this link.

5.1. Generate Simulink Real-Time interface

Analysis

Generate Simulink Real-Time interface

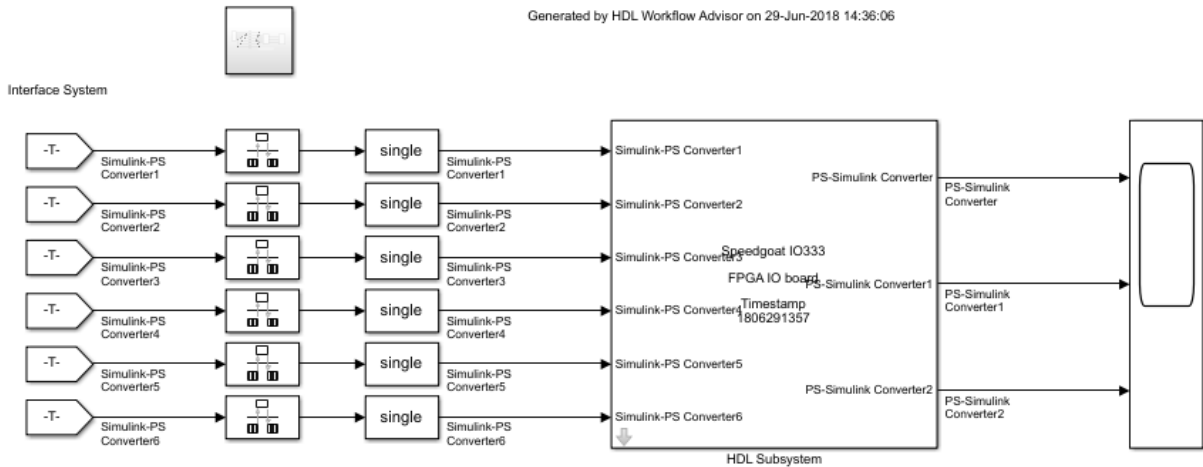
Run This Task

Result: ✔ Passed

Passed Generate Simulink Real-Time Interface.

Generating new Simulink Real-Time Interface model: [gm_gmStateSpaceHDL.TwoLevelConverter HDL. slrt](#)

Simulink Real-Time Interface model generation complete.



Export HDL Workflow to Script

Optionally, you can:

- Save the HDL Workflow Advisor settings to script and run the script using command line.
- Import the settings to modify it and rerun it using the HDL Workflow Advisor User Interface.

Export an HDL Workflow Script

- 1** In the HDL Workflow Advisor, configure and run all the tasks.
- 2** Select **File > Export to Script**.
- 3** In the Export Workflow Configuration dialog box, enter a file name and save the script.

The script is a MATLAB® file that you can run from the command line.

Import an HDL Workflow Script

- 1** In the HDL Workflow Advisor, select **File > Import from Script**.
- 2** In the Import Workflow configuration dialog box, select the script file and click **Open**.

The HDL Workflow Advisor updates the tasks with the imported script settings.

Simulink Real-Time FPGA I/O Workflow Example

This example shows how to configure and run an exported HDL Workflow script.

To generate an HDL Workflow script, configure and run the HDL Workflow Advisor with your Simulink design, then export the script.

This script is a Simulink Real-Time FPGA I/O workflow script that targets the Speedgoat I0333-325K module, which uses the Xilinx Vivado synthesis tool.

To edit the exported script in MATLAB command window, enter:

```
edit('hdlworkflow_slrt.m')
```

For more information, see Run HDL Workflow with a Script

See Also

Functions

checkhdl | makehdl

More About

- “IP Core Generation Workflow for Speedgoat I/O Modules” (HDL Coder)
- “FPGA Programming and Configuration” (HDL Coder)
- “Troubleshoot Conversion of Simscape DC Motor Control to HDL-Compatible Simulink Model” (HDL Coder)
- “Simscape HDL Workflow Advisor Tasks” (HDL Coder)
- “Validate HDL Implementation Model to Simscape Algorithm” (HDL Coder)

Troubleshoot Conversion of Simscape DC Motor Control to HDL-Compatible Simulink Model

This example shows how to modify a Simscape™ plant model to generate an HDL-compatible Simulink™ model with HDL Coder™. HDL code is then generated from this Simulink model.

Introduction

The Simscape plant model is converted to an HDL compatible Simulink model by using the Simscape HDL Workflow Advisor. To run the Advisor, you invoke the `sschdladvisor` function for the model.

The Simscape HDL Workflow Advisor generates an HDL Implementation model from which you can generate HDL code. Before you generate the implementation model, make sure make sure that the Simscape plant model is compatible for generation of the implementation model using the Simscape HDL Workflow Advisor. For more information, see [Generate HDL Code from Simscape Models](#).

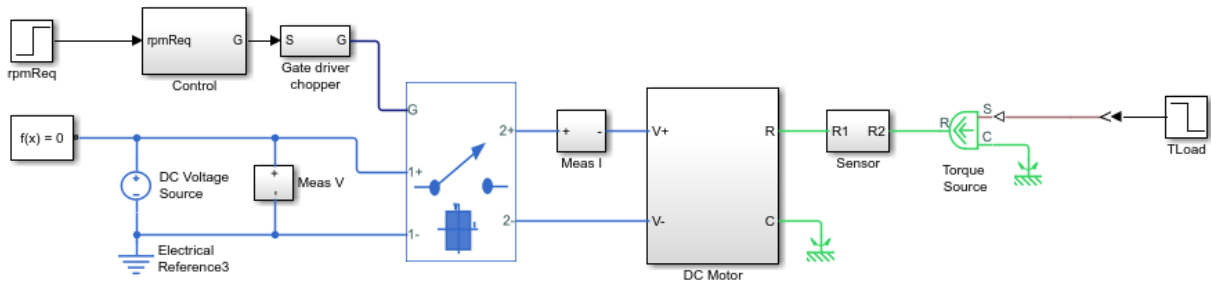
In some cases, the Simscape plant model may not be compatible for generation of the implementation model using the Simscape HDL Workflow Advisor. For HDL compatibility, you can modify the Simscape plant model and then run the Simscape HDL Workflow Advisor.

This example illustrates the DC Motor Control plant model. The model contains a nonlinear Friction block. You can use the approach in this example to convert Simscape models with few nonlinear blocks to a HDL-compatible Simulink model.

DC Motor Control Model

This is a physical model developed in Simscape. The model contains nonlinear elements and needs certain modifications for generating the implementation model.

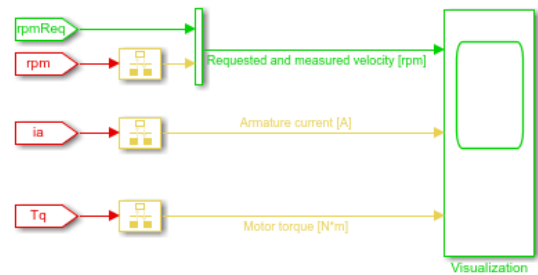
```
load('dc_motor_control_params.mat')
open_system('ee_dc_motor_control_original')
```



DC Motor Control

This example shows a cascade speed-control structure for a DC motor. A PWM controlled four-quadrant Chopper is used to feed the DC motor. The Control subsystem includes the outer speed-control loop, the inner current-control loop, and the PWM generation. The total simulation time (t) is 4 seconds. At t = 1.5 seconds, the load torque increases. At t = 2.5 seconds, the reference speed is changed from 1000 rpm to 2000 rpm.

1. [Edit associated parameterization script.](#)
2. [Explore simulation results using sscxplorer.](#)
3. [Learn more](#) about this example.



DC motor control is used as a speed control structure. A PWM controlled four-quadrant Chopper is used to feed the DC motor. The DC motor consists of Rotational Electromechanical Converter, Resistor, Inductance, Friction block and an Inertia block. The control subsystem includes the outer speed-control loop, the inner current-control loop and the PWM generation. To see how the models work, simulate the model.

```
sim('ee_dc_motor_control_original')
```

To convert Simscape plant models into HDL-compatible implementation model, make sure that the model does not contain nonlinear components or blocks.

Make DC Motor Model HDL-Compatible

To make the Simscape plant model HDL compatible:

1. Detect presence of nonlinear components or blocks in the model. To verify the presence of nonlinear blocks in Simscape plant model, enter:

```
simscape.findNonlinearBlocks('ee_dc_motor_control_original')
```

Found network that contains nonlinear equations in the following blocks:
'ee_dc_motor_control_original/DC Motor/Friction'

The number of linear or switched linear networks in the model is 0.
The number of nonlinear networks in the model is 1.

ans =

1x1 cell array

```
{'ee_dc_motor_control_original/DC Motor/Friction'}
```

The Simscape plant model has a nonlinear block, which is the Friction block. For HDL compatibility, remove the Friction block.

```
delete_block('ee_dc_motor_control_original/DC Motor/Friction')
```

2. Reduce the stop time of this model in Model configuration Parameters to 1s.

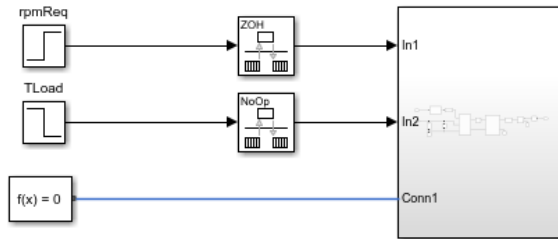
```
set_param('ee_dc_motor_control_original', 'Solver', 'ode15s', 'StopTime', '1')
```

3. Excluding the inputs and outputs, enclose all other blocks at the top level of the DC motor control model inside a subsystem. Attach Rate Transition blocks at the inputs TLoad and rpmReq. Save the changes into a new model as ee_dc_motor_control_modif.

```
load('dc_motor_control_params.mat')
```

```
open_system('ee_dc_motor_control_modif')
```

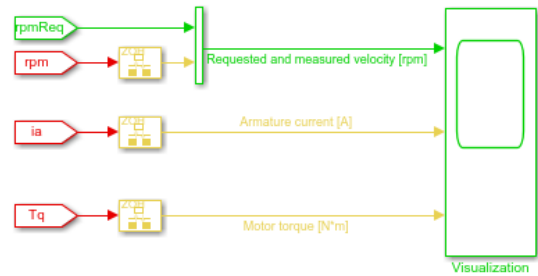
```
set_param('ee_dc_motor_control_modif', 'SimulationCommand', 'update')
```



DC Motor Control

This example shows a cascade speed-control structure for a DC motor. A PWM controlled four-quadrant Chopper is used to feed the DC motor. The Control subsystem includes the outer speed-control loop, the inner current-control loop, and the PWM generation. The total simulation time (t) is 4 seconds. At t = 1.5 seconds, the load torque increases. At t = 2.5 seconds, the reference speed is changed from 1000 rpm to 2000 rpm.

1. [Edit associated parameterization script.](#)
2. [Explore simulation results](#) using [sscexplore](#).
3. [Learn more](#) about this example.

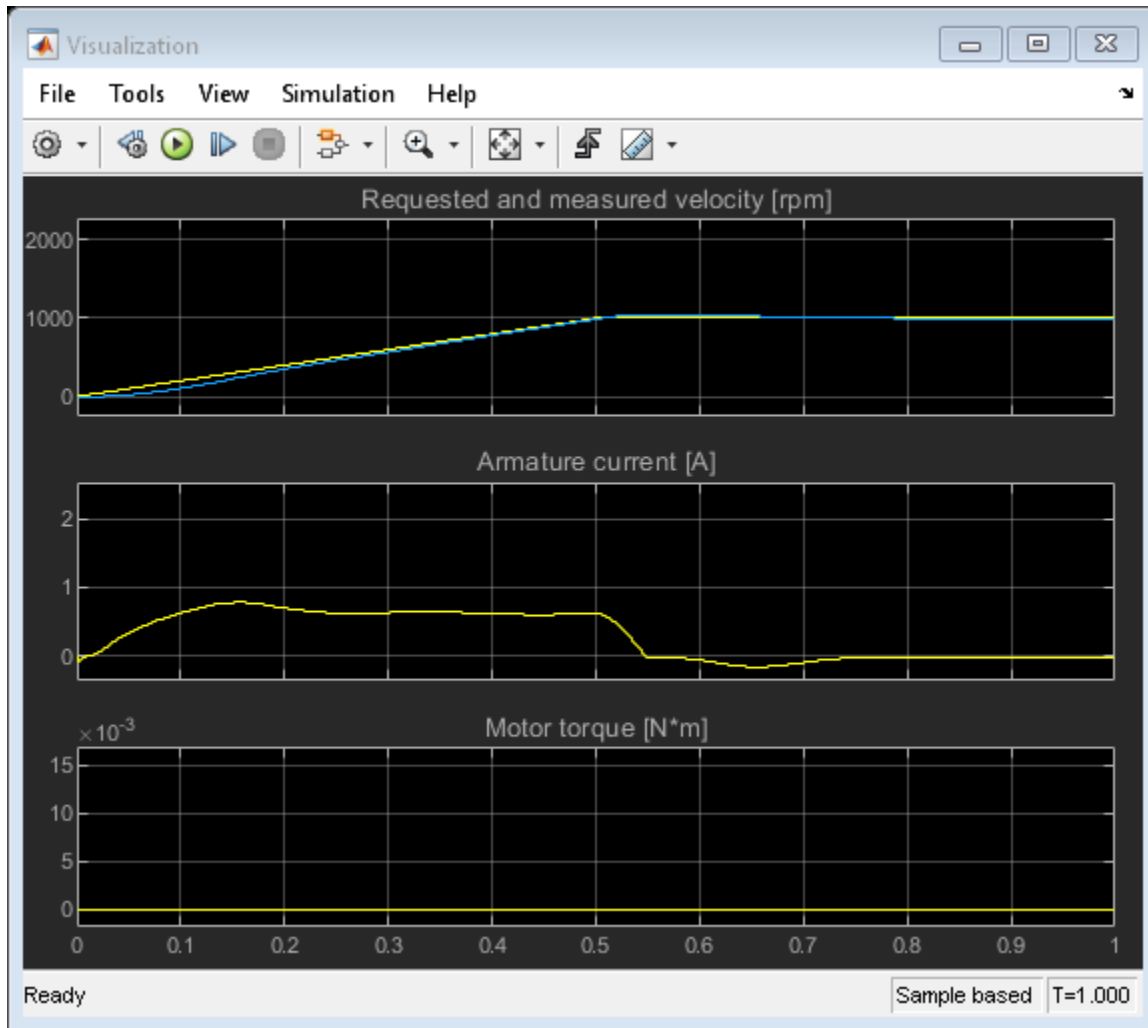


4. To see the simulation results after you modify the model, run these commands:

```
sim('ee_dc_motor_control_modif')
```

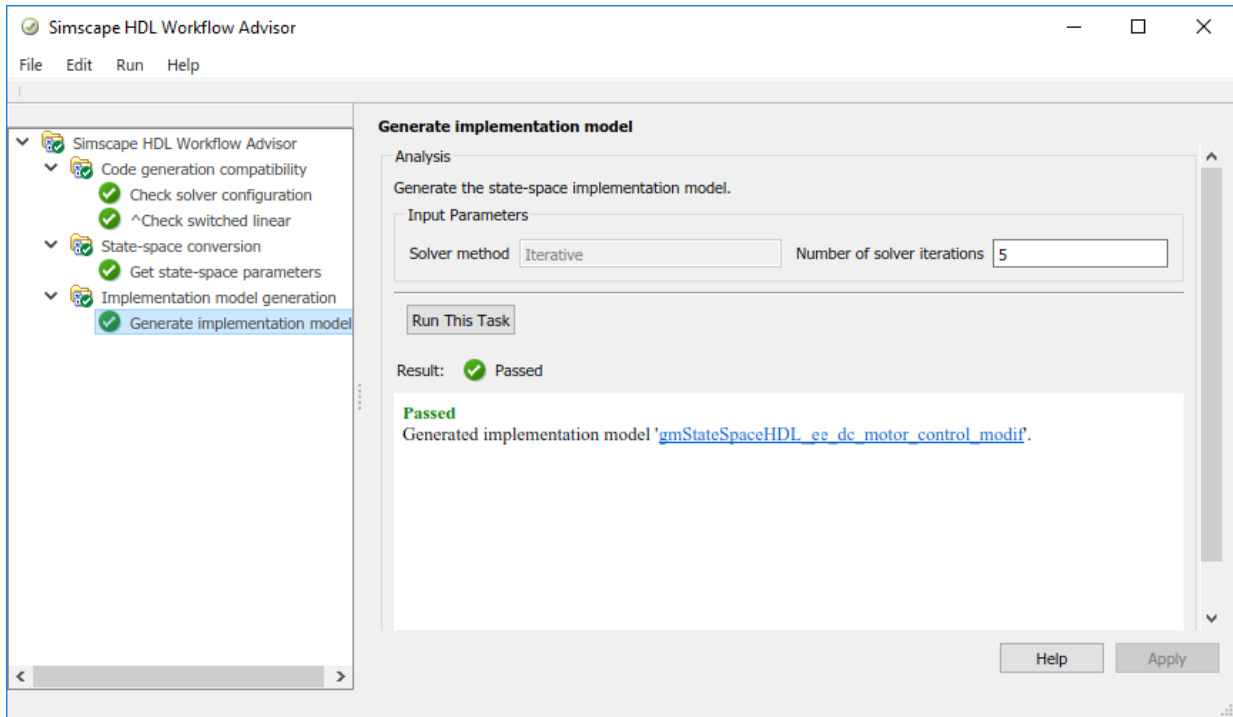
5. To view results, open the Scope block:

```
open_system('ee_dc_motor_control_modif/Visualization')
```



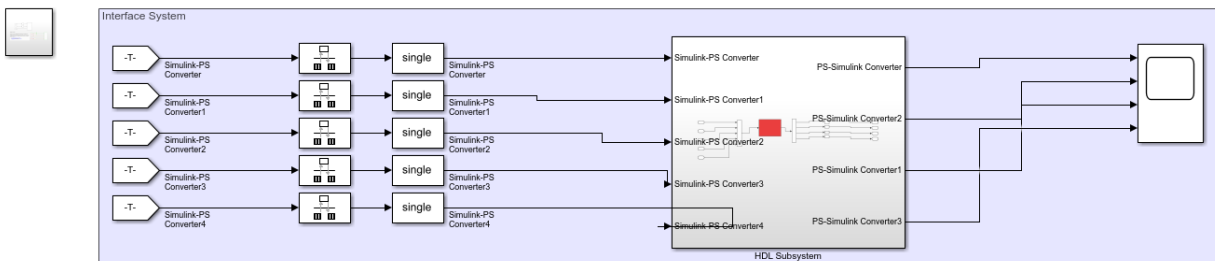
Run Simscape HDL Workflow Advisor and Verify Simulation Results

To open the Simscape HDL Workflow Advisor, run the `sschdladvisor` for your model.
`sschdladvisor('ee_dc_motor_control_modif')`



To generate the implementation model, in the Simscape HDL Workflow Advisor, leave all tasks to the default settings and then run the tasks. You see a link to the model in the **Generate implementation model** task. To open the implementation model, enter:

```
open_system('gmStateSpaceHDL_ee_dc_motor_control_modif')
```



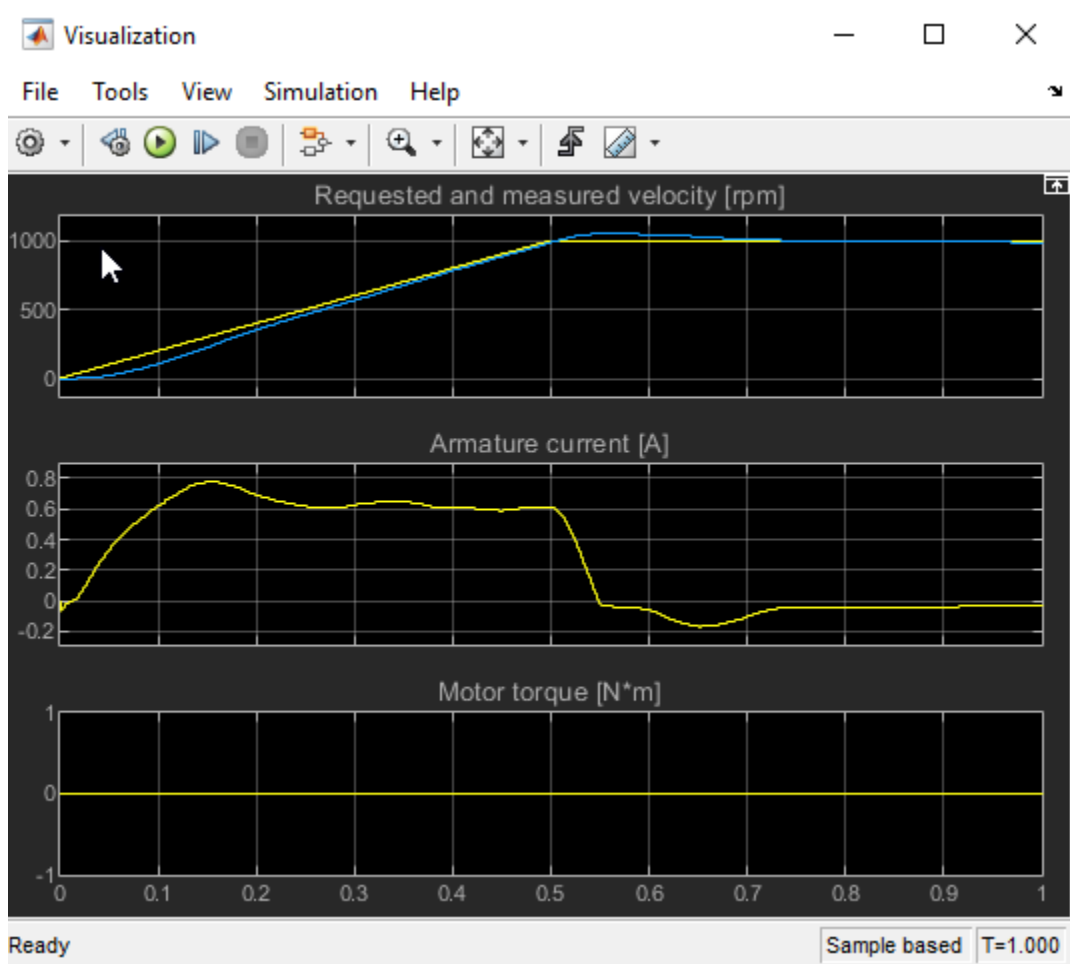
Simulate Implementation model and Generate HDL code

The sample time of Implementation model is related to the sample time of modified Simscape plant model and the number of iterations specified in Simscape HDL Workflow Advisor. Incorrect setting of sample time can result in failure to simulate the implementation model. The sample time of modified plant model is T_s , and the number of iterations are five. So the sample time of Implementation model should be $T_s/5$. To set the sample time, enter:

```
set_param('gmStateSpaceHDL_ee_dc_motor_control_modif','SolverType', ...  
          'Fixed-step','FixedStep','Ts/5')
```

To simulate the model, run this command and then open the Scope block to see the results:

```
sim('gmStateSpaceHDL_ee_dc_motor_control_modif')
```

From the Scope block, you can verify that the output generated by the modified Simscape plant model matches the output generated by the implementation model.

Generate HDL Code and Validation Model

You can now generate HDL code for the implementation model. Before you can generate HDL code, you must select the **Treat each discrete rate as a separate task** check box and set **Single task rate transition** to error.

```
set_param('gmStateSpaceHDL_ee_dc_motor_control_modif','EnableMultiTasking','on', ...  
          'SingleTaskRateTransMsg','error')
```

It is recommended to enable generation of the validation model. The validation model compares the output of the generated model after code generation and the modified Simscape plant model. To learn more, see [Generated Model and Validation Model](#).

To save validation model generation settings on your Simulink model, run these commands:

```
modelName = 'gmStateSpaceHDL_ee_dc_motor_control_modif';  
hdlset_param(modelName, 'TargetDirectory', 'C:/Temp/hdlsrc')  
hdlset_param(modelName, 'GenerateValidationModel', 'on');
```

To generate HDL code, run this command:

```
makehdl('gmStateSpaceHDL_ee_dc_motor_control_modif/HDL Subsystem')
```

By default, HDL Coder generates VHDL code. To generate Verilog code, run this command:

```
makehdl('gmStateSpaceHDL_ee_dc_motor_control_modif/HDL Subsystem', 'TargetLanguage', 'V')
```

The generated HDL code and the validation model is saved in C:/Temp/hdlsrc directory. The generated code is saved as HDL_Subsystem_tc.vhd.

Optionally, you can:

- 1 Verify the model generated after HDL code generation by using the validation model. To open the validation model, click the link to [gm_gmStateSpaceHDL_ee_dc_motor_control_modif_vnl.slx](#). when you generate code.
- 2 Deploy the generated HDL code on Speedgoat FPGA I/O boards or other target platforms. For more information, see [Deploy Simscape plant models to Speedgoat FPGA IO modules](#).

See Also

Functions

[checkhdl](#) | [makehdl](#)

More About

- [“Generate HDL Code from Simscape Models” \(HDL Coder\)](#)
- [“Getting Started with Simscape Electrical”](#)
- [“Simscape HDL Workflow Advisor Tasks” \(HDL Coder\)](#)
- [“Validate HDL Implementation Model to Simscape Algorithm” \(HDL Coder\)](#)

Troubleshoot Conversion of Simscape™ Permanent Magnet Synchronous Motor to HDL-Compatible Simulink Model

This example shows how to modify a Simscape™ plant model to generate an HDL-compatible Simulink™ model with HDL Coder™. HDL code is then generated from this Simulink model.

Introduction

The Simscape plant model is converted to an HDL-Compatible Simulink model by using the Simscape HDL Workflow Advisor. To run the Advisor, call the `sschdladvisor` function for the model.

The Simscape HDL Workflow Advisor generates an HDL implementation model from which you can generate HDL code. Before you generate the implementation model, make sure that the Simscape plant model is compatible for generation of the implementation model by using the Simscape HDL Workflow Advisor. For more information, see [Generate HDL Code from Simscape Models](#).

In some cases, the Simscape plant model might not be compatible for generation of the implementation model using the Simscape HDL Workflow Advisor. For HDL compatibility, you can modify the Simscape plant model and then run the Simscape HDL Workflow Advisor.

This example illustrates how to modify Permanent Magnet Synchronous Motor model in Simscape for HDL compatibility. The model is nonlinear and not compatible for generation of implementation model by using the Simscape HDL Workflow Advisor. This example illustrates how you can modify a continuous-time nonlinear model to a discrete-time switched linear model. The model becomes compatible for generation of the implementation model from which you can generate HDL code.

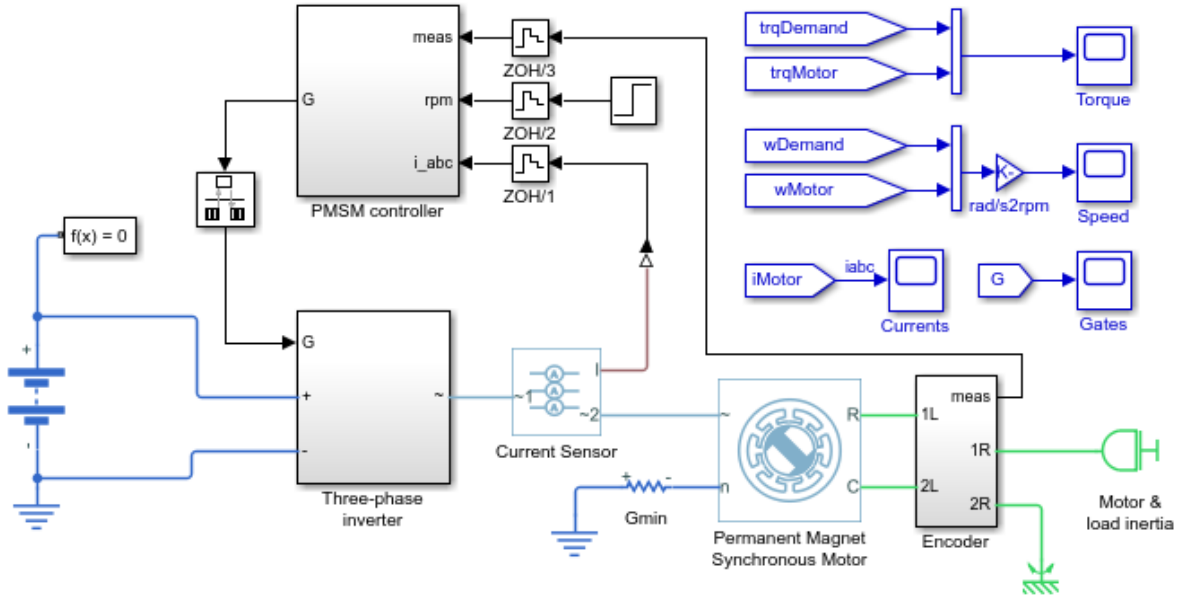
Permanent Magnet Synchronous Motor Model

This model is a physical system developed in Simscape. The model contains nonlinear elements and needs modifications for generating the implementation model. To open the model, run this command:

```
open_system('ee_pmsm_drive')
```

Save this model as `ee_pmsm_drive_original.slx`

```
open_system('ee_pmsm_drive_original')
```



Three-Phase PMSM Drive

This example shows a Permanent Magnet Synchronous Machine (PMSM) and inverter sized for use in a typical hybrid vehicle. Here the inverter is connected directly to the vehicle battery, but often there is also a DC-DC converter stage in between. The model can be used to design the PMSM controller, selecting architecture and gains to achieve desired performance. To check the timing of IGBT turn-on and turn-off, the IGBT devices can be directly replaced by the more detailed N-Channel IGBT block. For complete vehicle modeling, the Servomotor block can be used to abstract the PMSM, inverter and controller with an energy-based model. The Gmin resistor provides a very small conductance to ground that improves the numerical properties of the model when using a variable-step solver.

This model shows a Permanent Magnet Synchronous Machine (PMSM) and inverter sized for use in a typical hybrid vehicle. The inverter is connected directly to the vehicle battery. You can use the model to design the PMSM controller, selecting architecture and gains to achieve performance that you want. The Gmin resistor provides a very small conductance to ground that improves the numerical properties of the model when using a variable-step solver. To see how the model works, simulate the model.

```
sim('ee_pmsm_drive_original')
```

This model is a continuous time system. To work with Simscape HDL Workflow Advisor, you must convert the model into a discrete system.

Convert Continuous-Time Model to Fixed-Step Discrete Model

1. Configure the solver options for HDL code generation by using a Solver Configuration block. In the block parameters of this block:
 - Select **Use local solver**.
 - Use **Backward Euler** as the **Solver type**.
 - Specify a discrete **Sample time**, T_s .
2. Modify Solver settings in Configuration Parameters dialog box, on the **Solver** pane:
 - Set **Solver selection type** to Fixed-Step.
 - Set **Solver** to discrete (no continuous states).
 - Set **Fixed-step size (fundamental sample time)** to T_s .
 - Clear **Treat each discrete rate as a separate task** in the section **Tasking and sample time options**.
3. Modify the display settings of your model in the Simulink Editor.
 - On the **Display** tab, set the **Sample Time** to All. Review the generated report for any blocks that have a sample time other than T_s , or which are running on a continuous time scale.
4. Double-click the Step block in the model and set its **Sample time** to T_s .
5. For faster simulation, ignore the zero-sequence parameters of the PMSM. Double-click the Permanent Magnet Synchronous Motor block and set **Zero Sequence** to **Exclude**.

The model is now a fixed-step discrete system. Simulate the model and compare signals in the Simulation Data Inspector.



Some Pulse Width Modulation (PWM) signals transition earlier or later between simulations, but they are well within the step-size of the controller. The only signals that differ beyond tolerance are the **Torque Demand** and **Motor Torque**, toward the end of simulation, within acceptable limits.

Make Three Phase PMSM Drive Compatible with HDL Advisor

To make the Simscape plant model HDL-Compatible, identify the presence of any nonlinear components or blocks in the model. Enter:

```
simulink.findNonlinearBlocks('ee_pmsm_drive_original')
```

Found network that contains nonlinear equations in the following blocks:
'ee_pmsm_drive_original/Permanent Magnet Synchronous Motor'

The number of linear or switched linear networks in the model is 0.
The number of nonlinear networks in the model is 1.

ans =

```
1x1 cell array  
  
{'ee_pmsm_drive_original/Permanent Magnet Synchronous Motor'}
```

The Simscape plant model has a nonlinear block, which is the PMSM block.

Remove the PMSM block, Encoder block, Gmin resistor, and, Motor & load inertia block. To delete the Simscape elements and blocks, enter:

```
delete_block('ee_pmsm_drive_original/Permanent Magnet Synchronous Motor')  
delete_block('ee_pmsm_drive_original/Encoder')  
delete_block('ee_pmsm_drive_original/Gmin')  
delete_block('ee_pmsm_drive_original/Motor & load inertia')
```

For ease of implementation, the Simscape plant model is converted into an HDL Advisor compatible model by using a two-step process:

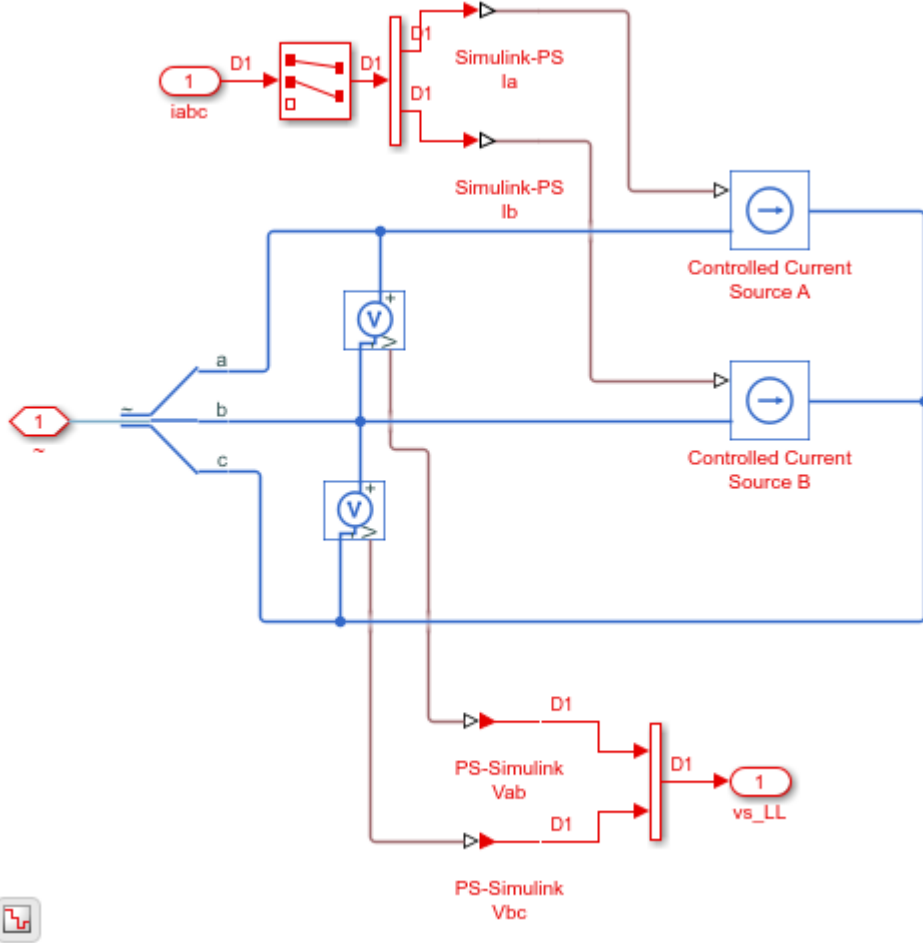
1. Implement an initial nonlinear Simulink model by using double-precision data type, and blocks that are not compatible with HDL coder.
2. Modify this nonlinear model to make it suitable for deployment with HDL Coder by using single-precision data types and blocks that are compatible with HDL Coder for more efficient HDL code generation.

Step 1: Convert Simscape Plant Model to Initial Simulink Model

Replace the PMSM block, Encoder block, Gmin resistor and, Motor & load inertia block with Electrical Interface Simulink block and Permanent Magnet Synchronous Motor Simulink block.

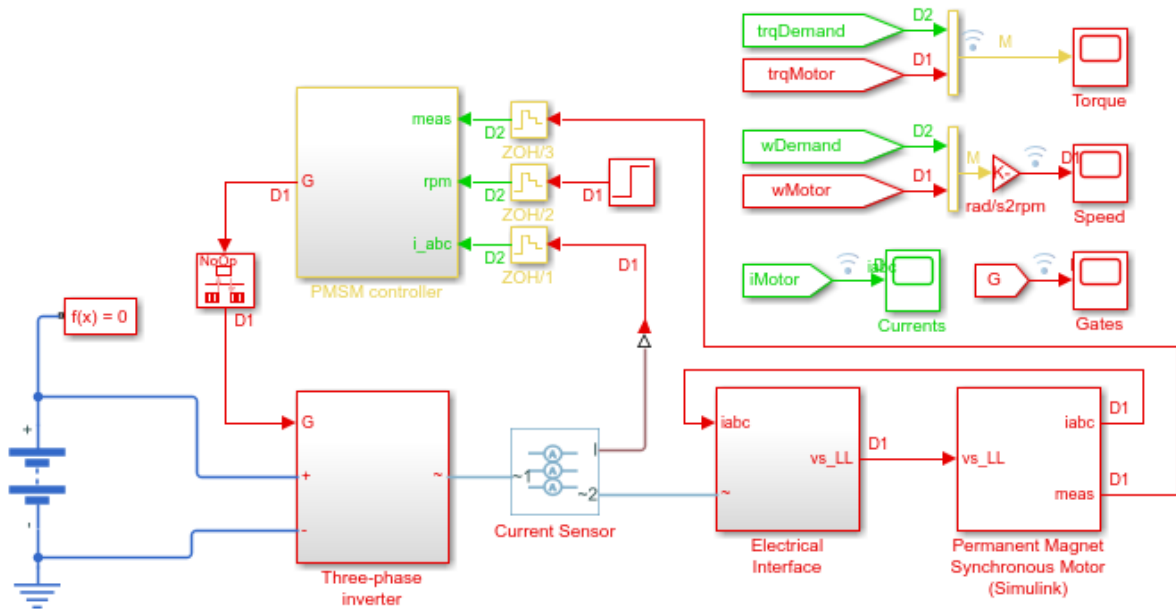
The Electrical Interface Simulink block is implemented by using Controlled Current Sources. The interface to the PMSM is isolated from the implementation.

```
load_system('ee_pmsm_drive_initialSL')  
set_param('ee_pmsm_drive_initialSL','SimulationCommand','update')  
open_system('ee_pmsm_drive_initialSL/Electrical Interface')
```

The PMSM Simulink block is implemented by using Electrical Equations and Mechanical Equations. For more information, see PMSM.

```
open_system('ee_pmsm_drive_initialSL/Permanent Magnet Synchronous Motor (Simulink)')
```



Three-Phase PMSM Drive

This example shows a Permanent Magnet Synchronous Machine (PMSM) and inverter sized for use in a typical hybrid vehicle. Here the inverter is connected directly to the vehicle battery, but often there is also a DC-DC converter stage in between. The model can be used to design the PMSM controller, selecting architecture and gains to achieve desired performance.



Save the updated model as ee_pmsm_drive_singleSL.

Step 2: Convert Simulink Machine Model to Single-Precision HDL-Compatible Simulink Model

1. Update Output data type in **Signal Attributes** tab of Gain and Product blocks to Single.

To search the Gain blocks in the Simulink model, enter:

```
load_system('ee_pmsm_drive_singleSL')
set_param('ee_pmsm_drive_singleSL','SimulationCommand','update')
blockconstant=find_system('ee_pmsm_drive_singleSL','blocktype','Gain')
```

```
blockconstant =  
  
    2x1 cell array  
  
    {'ee_pmsm_drive_singleSL/PMSM controller/rpm ->...'}  
    {'ee_pmsm_drive_singleSL/rad//s2rpm'}
```

The array `blockconstant` stores the exact location of Gain blocks as variables. There are two Gain blocks in the Simulink model.

To set the output data type of these Gain blocks to single, enter:

```
set_param(blockconstant{1}, 'OutDataTypeStr', 'single')  
set_param(blockconstant{2}, 'OutDataTypeStr', 'single')
```

To search the Product blocks in the Simulink model, enter:

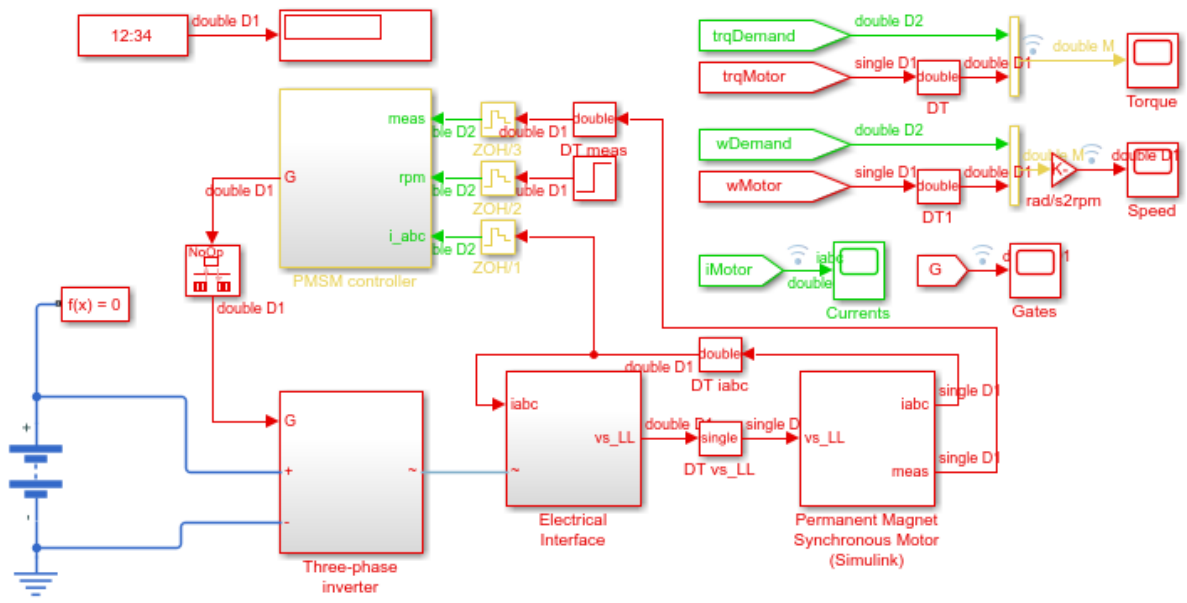
```
find_system('ee_pmsm_drive_singleSL', 'blocktype', 'Product')
```

```
ans =  
  
    0x1 empty cell array
```

There are no Product blocks in the Simulink model.

2. The Park Transform and Inverse Park Transform blocks inside the PMSM Simulink block are modified to eliminate the 6 Sine and 6 Cosine blocks.

```
open_system('ee_pmsm_drive_singleSL/Permanent Magnet Synchronous Motor (Simulink)')
```



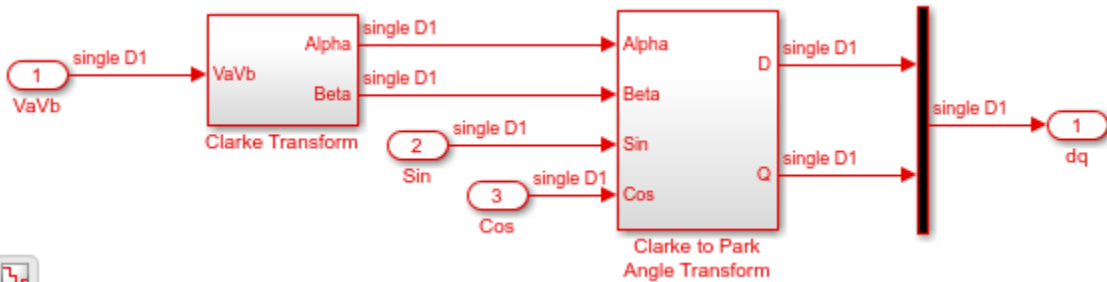
Three-Phase PMSM Drive

This example shows a Permanent Magnet Synchronous Machine (PMSM) and inverter sized for use in a typical hybrid vehicle. Here the inverter is connected directly to the vehicle battery, but often there is also a DC-DC converter stage in between. The model can be used to design the PMSM controller, selecting architecture and gains to achieve desired performance.



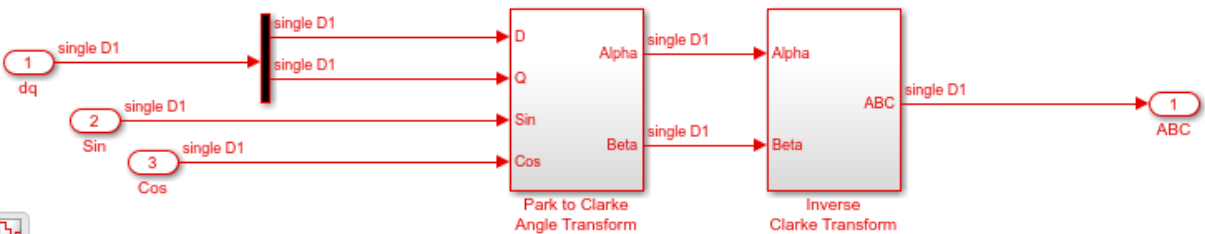
3. Reduce the hardware required for the Park Transform block by adding Clarke Transform and Clarke to Park Angle Transform blocks.

`open_system('ee_pmsm_drive_singleSL/Permanent Magnet Synchronous Motor (Simulink)/Park`



4. Reduce the hardware required for the Inverse Park Transform block by adding Inverse Park to Clarke Angle Transform and Inverse Clarke Transform blocks.

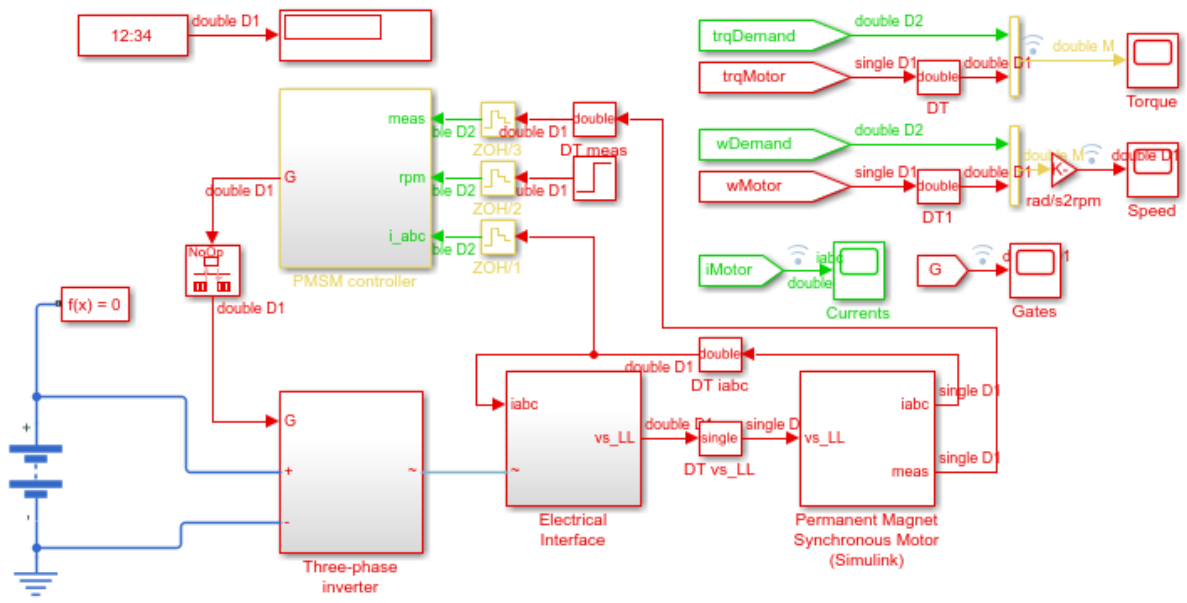
```
open_system('ee_pmsm_drive_singleSL/Permanent Magnet Synchronous Motor (Simulink)/Inverse Park to Clarke Angle Transform and Inverse Clarke Transform')
```



5. Replace the Three-Phase Current Sensor Simscape block by feeding the controller with three-phase currents coming from the PMSM model.

6. Add a Digital Clock with **Sample time** Ts. Connect the clock to a Display block.

```
open_system('ee_pmsm_drive_singleSL')
```



Three-Phase PMSM Drive

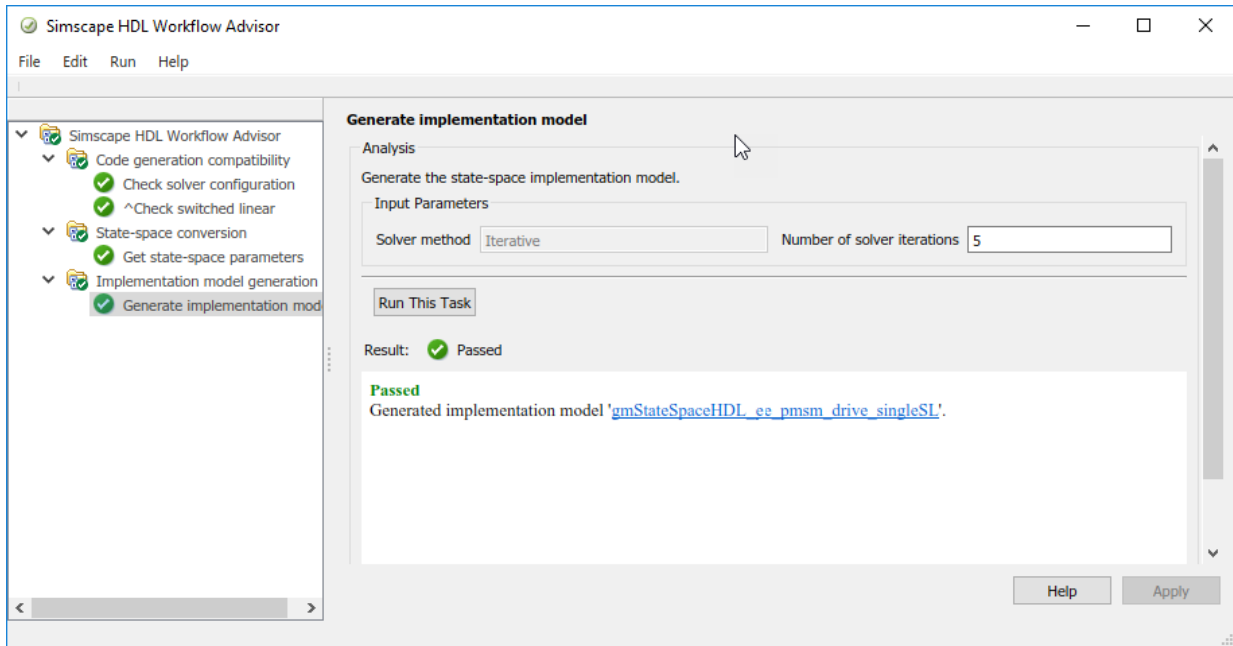
This example shows a Permanent Magnet Synchronous Machine (PMSM) and inverter sized for use in a typical hybrid vehicle. Here the inverter is connected directly to the vehicle battery, but often there is also a DC-DC converter stage in between. The model can be used to design the PMSM controller, selecting architecture and gains to achieve desired performance.



Run Simscape HDL Workflow Advisor

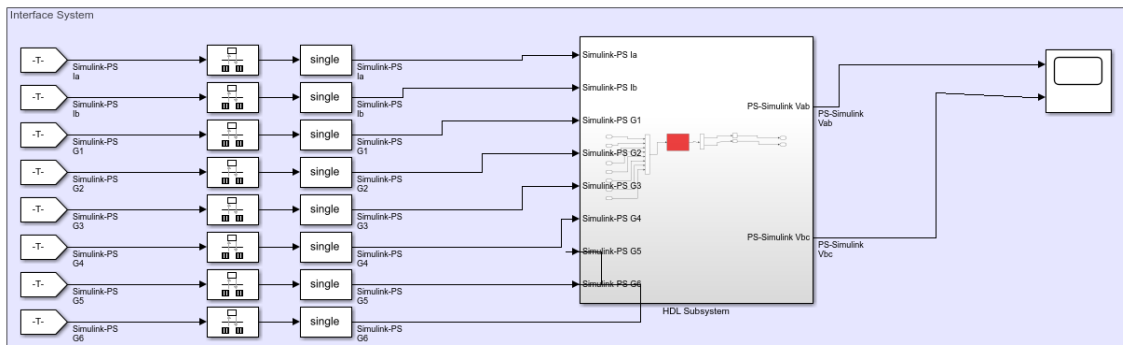
To open the Simscape HDL Workflow Advisor, run the `sschdladvisor` for your model.

```
sschdladvisor('ee_pmsm_drive_singleSL')
```



To generate the implementation model, in the Simscape HDL Workflow Advisor, leave the default settings and then run the tasks. You see a link to the model in the **Generate implementation model** task. To open the implementation model, enter:

```
load_system('gmStateSpaceHDL_ee_pmsm_drive_singleSL')
open_system('gmStateSpaceHDL_ee_pmsm_drive_singleSL')
```



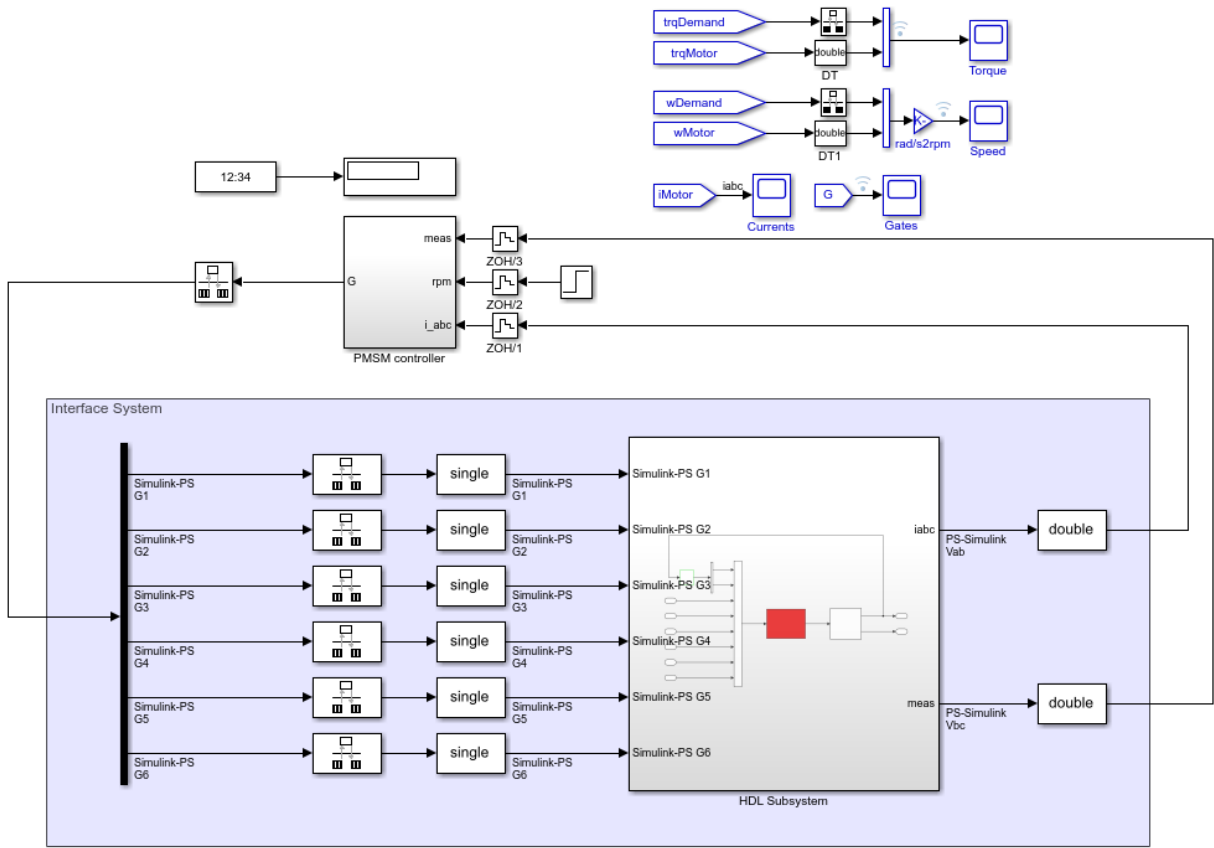
To execute the implementation model, modify the **Fixed-step size** to $T_s/5$ as **Number of Iterations** in `sschdladvisor` are 5.

Reconfigure Simulink Model for HDL Code Generation

The single precision implementation model is reconfigured for HDL code generation. To reconfigure, place the PMSM model within the HDL Subsystem. Save the model as `gmStateSpaceHDL_ee_pmsm_drive_GenerateHDL`.

To open the reconfigured implementation model, enter:

```
load_system('gmStateSpaceHDL_ee_pmsm_drive_GenerateHDL')  
open_system('gmStateSpaceHDL_ee_pmsm_drive_GenerateHDL')
```

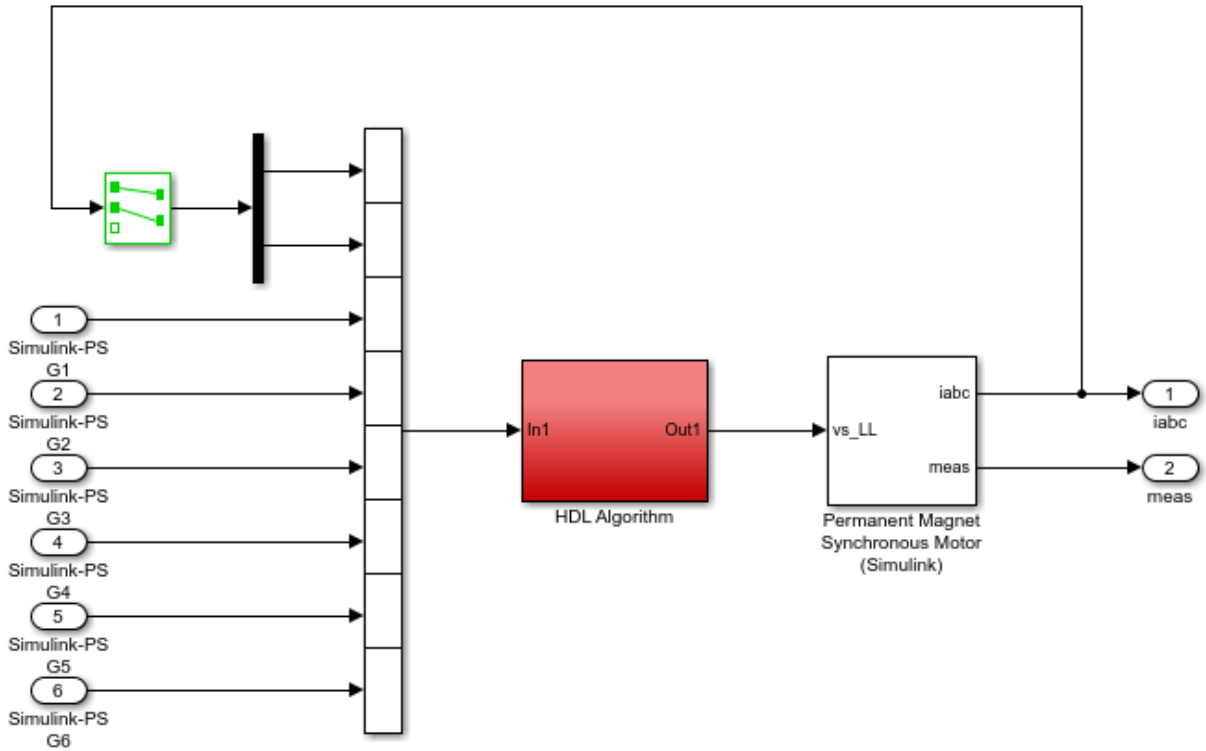



Three-Phase PMSM Drive

This example shows a Permanent Magnet Synchronous Machine (PMSM) and inverter sized for use in a typical hybrid vehicle. Here the inverter is connected directly to the vehicle battery, but often there is also a DC-DC converter stage in between. The model can be used to design the PMSM controller, selecting architecture and gains to achieve desired performance.

To open the **HDL Subsystem** block, enter:

```
open_system('gmStateSpaceHDL_ee_pmsm_drive_GenerateHDL/HDL Subsystem')
```



Generate HDL Code and Validation Model

Before you can generate HDL code, you must:

- 1 Select the **Treat each discrete rate as a separate task** check box.
- 2 Set **Single task rate transition** to error.
- 3 Set **Multitask rate transition** to error.

```
set_param('gmStateSpaceHDL_ee_pmsm_drive_GenerateHDL','EnableMultiTasking', 'on', ...
          '_SingleTaskRateTransMsg', 'error')
set_param('gmStateSpaceHDL_ee_pmsm_drive_GenerateHDL','EnableMultiTasking', 'on', ...
          '_MultitaskRateTransMsg', 'error')
```

It is recommended to enable generation of the validation model. The validation model compares the output of the generated model after code generation and the modified Simscape plant model. To learn more, see [Generated Model and Validation Model](#).

To save validation model generation settings on your Simulink model, run these commands:

```
modelName = 'gmStateSpaceHDL_ee_pmsm_drive_GenerateHDL';  
hdlset_param(modelName, 'TargetDirectory', 'C:/Temp/hdlsrc')  
hdlset_param(modelName, 'GenerateValidationModel', 'on');
```

To generate HDL code, run this command:

```
makehdl('gmStateSpaceHDL_ee_pmsm_drive_GenerateHDL/HDL Subsystem')
```

By default, HDL Coder generates VHDL code. To generate Verilog code, run this command:

```
makehdl('gmStateSpaceHDL_ee_pmsm_drive_GenerateHDL/HDL Subsystem', 'TargetLanguage', 'Verilog');
```

The generated HDL code and the validation model is saved in C:\Temp\hdlsrc folder. The generated code is saved as HDL_Subsystem_tc.vhd.

Optionally, you can:

- 1 Verify the model generated after HDL code generation by using the validation model. To open the validation model, click the link to [gmStateSpaceHDL_ee_pmsm_drive_GenerateHDL.slx](#) when you generate code.
- 2 Deploy the generated HDL code on Speedgoat FPGA I/O boards or other target platforms. For more information, see [Deploy Simscape plant models to Speedgoat FPGA IO modules](#).

See Also

Functions

[checkhdl](#) | [makehdl](#)

More About

- [“Generate HDL Code from Simscape Models” \(HDL Coder\)](#)

- “Getting Started with Simscape Electrical”
- “Simscape HDL Workflow Advisor Tasks” (HDL Coder)
- “Validate HDL Implementation Model to Simscape Algorithm” (HDL Coder)

Validate HDL Implementation Model to Simscape Algorithm

If you design your algorithm by using Simscape switched linear blocks, you can run the Simscape HDL Workflow Advisor to generate an HDL implementation model. The HDL implementation model represents the Simscape algorithm by using Simulink blocks that are compatible for HDL code generation. Before you prototype the implementation model on an FPGA or target Speedgoat FPGA I/O modules, it is recommended that you verify the functionality of your design in the Simulink modeling environment. To verify the functionality, you can specify insertion of validation logic in the HDL implementation model when you run the Simscape HDL Workflow Advisor. This logic verifies whether the HDL implementation numerically matches the original Simscape algorithm. In some cases, there can be a mismatch in simulation results between the Simscape algorithm and the corresponding HDL implementation. Such mismatches generate warnings or assertions when you simulate the implementation model. To resolve the warnings, you can use a combination of various settings in the **Generate implementation model** task as illustrated below.

Bridge Rectifier Model

This example uses the bridge rectifier model to illustrate how to generate an implementation model with validation logic inserted in the model, and how you can resolve any assertions that may be generated when you simulate the implementation model.

- 1 Open the bridge rectifier model. To learn how the bridge rectifier is implemented, open the `Simscape_system` Subsystem. To open the bridge rectifier model, in the MATLAB Command Window, enter:

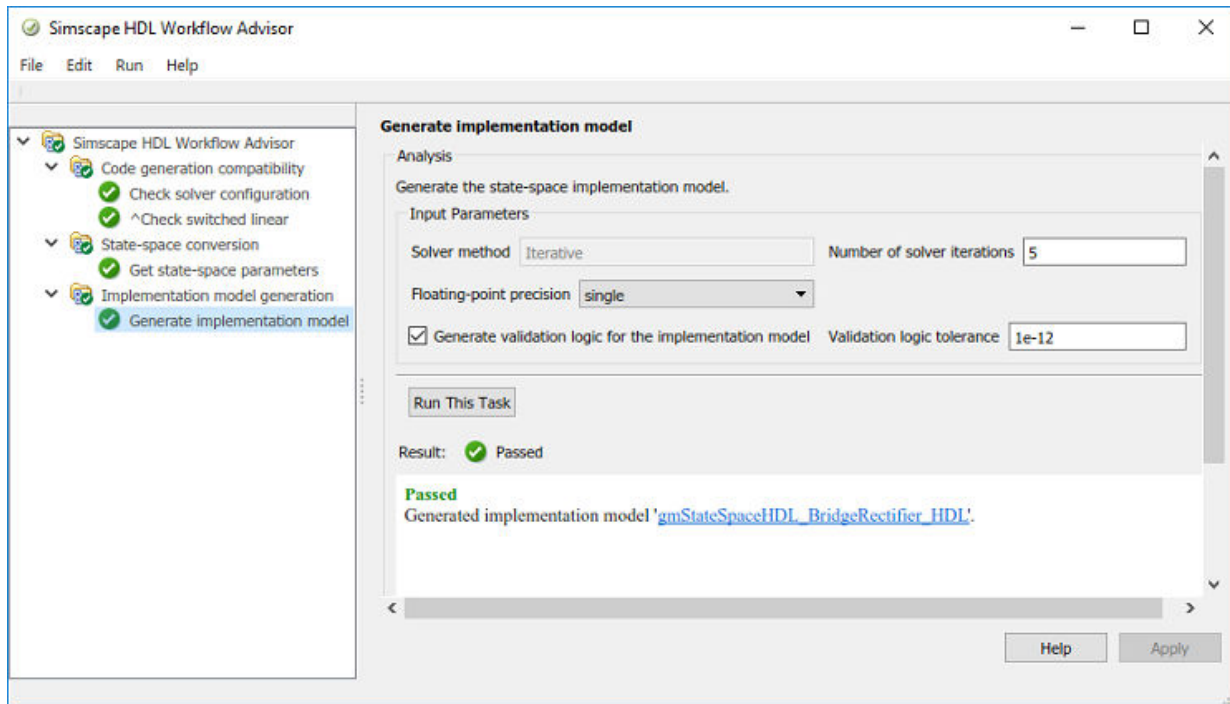
```
open_system('sschdlexBridgeRectifierExample')
open_system('sschdlexBridgeRectifierExample/Simscape_system')
```

- 2 Save a local copy of this model and rename it as `BridgeRectifier_HDL` to run the Simscape to HDL workflow. Open the Simscape HDL Workflow Advisor for your model:

```
sschdladvisor('BridgeRectifier_HDL')
```

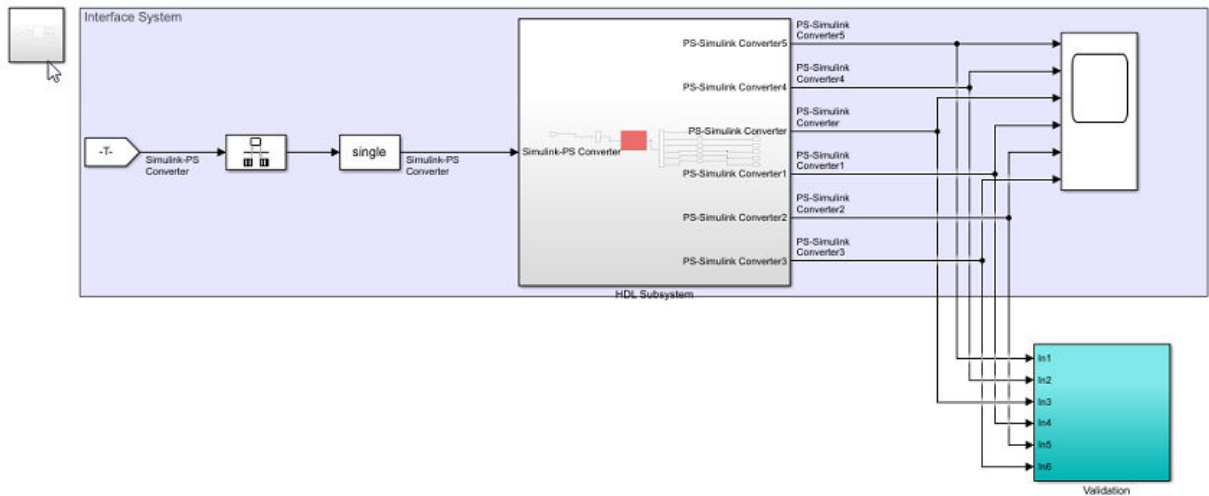
- 3 Right-click the **Get state-space parameters** task and select **Run to Selected Task** to run all tasks in the Advisor except for the **Generate implementation model** task.

- 4 In the **Generate implementation model** task, select the **Generate validation logic for the implementation model** check box. Leave the other options with their default values and select **Run This Task**.



After running this task, you see a link to the generated HDL implementation model. After you generate the implementation model, do not close the Simscape HDL Workflow Advisor UI. By keeping this window open, if you have to fix any warnings generated from the implementation model, you only have to modify the settings in the **Generate implementation model** task and then rerun this task. You do not have to modify or rerun other tasks.

- 5 Click the link to open the HDL implementation model. You see a **Validation Subsystem** that compares the simulation results of the Simscape model to the HDL implementation model. Simulate the implementation model.



You see that simulating the model generates multiple assertions indicating a mismatch in the simulation results. If you open the Diagnostic Viewer, you see this message:

```
Assertion detected in
'gmStateSpaceHDL_BridgeRectifier_HDL_SimMismatch/Validation/Check
Static Range1' at time 0.04186 [4982 similar]
```

The message indicates that the Simscape™ algorithm does not match the equivalent HDL implementation. To resolve the validation mismatch, you can modify various settings in the **Generate implementation model** task until the HDL implementation model matches the Simscape algorithm. The settings include **Validation logic tolerance**, **Number of solver iterations**, and **Floating-point precision**. In most cases, to resolve the numeric mismatch, you may want to use a combination of these settings.

Increase Validation Logic Tolerance

Conversion of a Simscape algorithm to an equivalent HDL implementation leads to rounding errors. By default, the **Validation logic tolerance** is set to $1e-12$. This tolerance value is relatively small and can be difficult to achieve especially with single-precision data types in the HDL implementation model. It is good practice to increase the **Validation logic tolerance** to an initial value such as $1e-4$. After you increase the tolerance value, make sure that you select **Generate validation logic for the implementation model**. Run the task to generate the HDL implementation model that

includes validation logic. Simulate the model and check whether the simulation displays any warnings in the Diagnostic Viewer. If the simulation results produce warnings, proceed to the next step to increase the number of solver iterations.

Increase Number of Solver Iterations

Before you generate the HDL implementation model, in your Simscape model, you use a Solver Configuration block that is configured to use a fixed-step, discrete solver that has **Solver type** set to **Backward Euler**. Based on these settings, the switched linear workflow arrives at a state-space representation for every mode in a physical system. The solver method use an iterative algorithm that executes the state-space model multiple times to determine and arrive at the correct mode. The default **Number of solver iterations** is 5. To improve the numeric accuracy, increase the **Number of solver iterations**.

You can start with an initial value of 20 for the **Number of solver iterations**. Run the **Generate implementation model** task while making sure that you selected **Generate validation logic for the implementation model**. Simulate the model and check whether the simulation displays any warnings in the Diagnostic Viewer. If the simulation results produce warnings, you can increase the **Number of solver iterations** by increments of 20, generate the implementation model, and then simulate the model to see if it meets the numeric accuracy requirements.

Note When you increase the number of solver iterations, the code generator changes the sample time of the generated HDL implementation model. A large number of iterations can increase the simulation time significantly.

If your implementation model does not meet the accuracy requirements after you use a value of 100 for **Number of solver iterations**, proceed to the next step to use double-precision floating-point data types.

Use Double-Precision Floating-Point

You can use the **Floating-point precision** setting in the **Generate implementation model** task to specify whether you want to generate an HDL implementation model with **single** or **double** data types. If you select **double** data types, the numeric accuracy of the generated model increases. However, using **double** data types in your model can increase the area consumption and reduce the target frequency. It is recommended that

you use `single` data types, and then simulate the generated model to see if your design meets the numeric accuracy requirements. Run the **Generate implementation model** task while making sure that you selected **Generate validation logic for the implementation model**. Simulate the model and check whether the simulation displays any warnings in the Diagnostic Viewer.

If the simulation results still produce warnings, proceed to the first step to further increase the validation logic tolerance. Use a tolerance value of `1e-03` and then simulate the model to see if the numeric accuracy requirements are met. If you still see warnings in the Diagnostic Viewer, increase the number of solver iterations. You can continue iterating between these steps till the HDL implementation model numerically matches the Simscape algorithm.

For the bridge rectifier model, to resolve the warnings, set the **Validation logic tolerance** to `1e-4` and specify the **Floating-point precision** as `double`. After you generate the implementation model with the validation logic, you see that simulating the model does not display any warnings in the Diagnostic Viewer.

See Also

Functions

`simscape.findNonlinearBlocks` | `sschdladvisor`

More About

- “Generate HDL Code from Simscape Models” (HDL Coder)
- “Simscape HDL Workflow Advisor Tasks” (HDL Coder)
- “Deploy Simscape™ Plant Models to Speedgoat FPGA I/O Modules” (HDL Coder)

Improve Sampling Rate of HDL Implementation Model Generated from Simscape Algorithm

If you design your algorithm by using Simscape switched linear blocks, you can run the Simscape HDL Workflow Advisor to generate an HDL implementation model. When you open the HDL implementation model, you see the HDL algorithm that models the state-space representation by using Simulink blocks that are compatible for HDL code generation. To learn more about the Simscape HDL Workflow Advisor, see “Simscape HDL Workflow Advisor Tasks” (HDL Coder).

Sampling Frequency

When you generate HDL code and deploy the plant model onto an FPGA, you may want to improve the sampling frequency. The sampling frequency depends on these parameters:

- FPGA clock frequency
- Oversampling factor
- Number of solver iterations

$$\text{Sampling Frequency} = \text{FPGA clock frequency} / (\text{Oversampling factor} * \text{Number of solver iterations})$$

To improve the sampling rate, you want to maximize the FPGA clock frequency, and minimize the oversampling factor and number of solver iterations. As you improve the sampling rate, make sure that the updated sampling frequency is equivalent to the fixed sample time that you specify for your original Simscape model by using the Solver Configuration block. To learn more about how this block is used in your model before running the Simscape HDL Workflow Advisor, see “Generate HDL Code from Simscape Models” (HDL Coder).

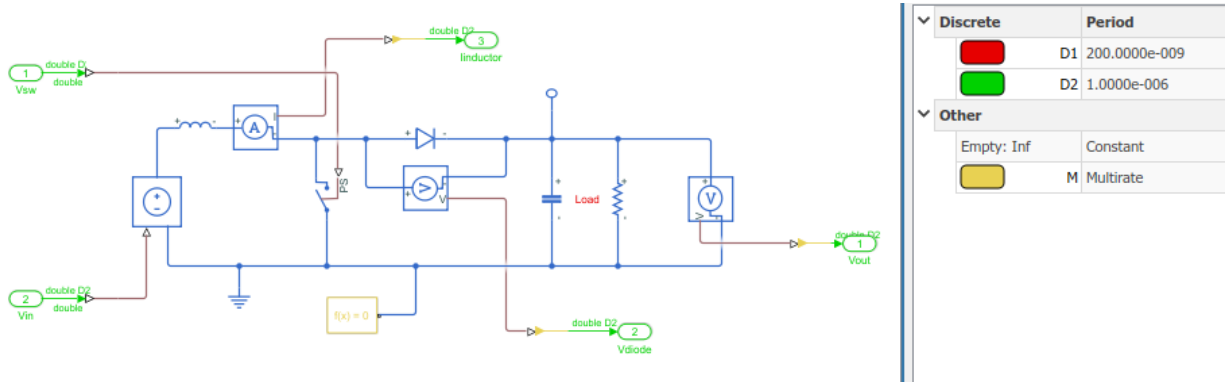
The preceding section uses the boost converter model as an example to illustrate how you can modify the oversampling factor and the number of solver iterations to improve the sampling rate.

Boost Converter Model

This example uses the boost converter model to illustrate the change in sample time in the generated HDL implementation model and the oversampling factor that is saved on the model.

- 1 Open the boost converter model. To learn how the boost converter is implemented, open the `Simscape_system` Subsystem. To open the boost converter model, in the MATLAB Command Window, enter:

```
open_system('sschdlexBoostConverterExample')
open_system('sschdlexBoostConverterExample/Simscape_system')
```



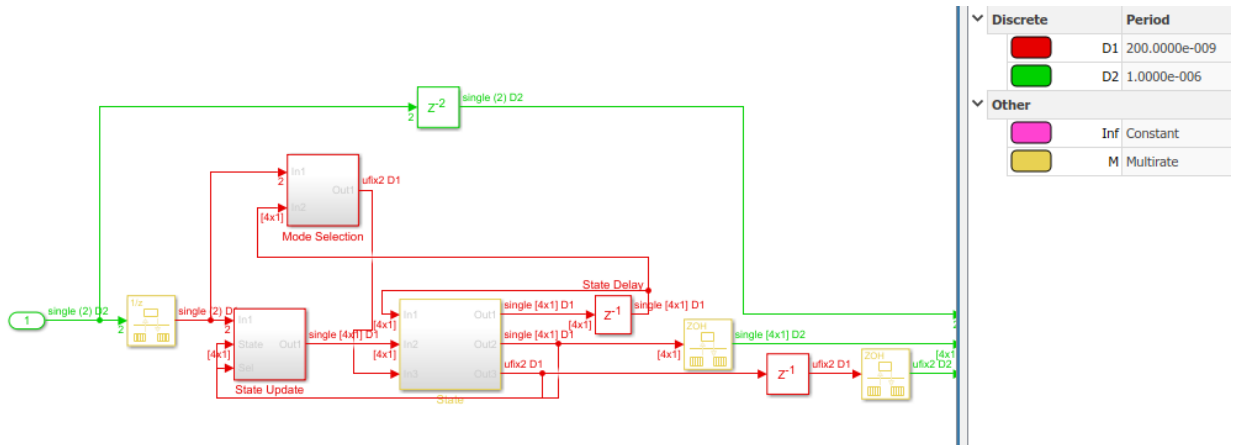
You see that the model runs at a sample time $1e-6$. The sample time of $200e-9$ corresponds to the sample time of the sources that drive the Simscape algorithm.

- 2 Save a local copy of this model and rename it as `BoostConverter_HDL` to run the Simscape to HDL workflow. Open the Simscape HDL Workflow Advisor for your model:

```
sschdladvisor('BoostConverter_HDL')
```

- 3 Run the workflow to the **Generate implementation model** task.

After running this task, you see a link to the generated HDL implementation model. Click the link to open the HDL implementation model. If you simulate the HDL implementation model and navigate the model to the HDL Algorithm Subsystem, you see that the model uses `single` data types, and contains Rate Transition and Zero Order Hold blocks that run the model at a sample time $200e-9$, which is 5 times faster than the original Simscape model.



- 4 Run this command to see the HDL parameter settings that are saved on the model:

```
hdlsaveparams('gmStateSpaceHDL_BoostConverter_HDL')
```

```
%% Set Model 'gmStateSpaceHDL_BoostConverter_HDL' HDL parameters
hdlset_param('gmStateSpaceHDL_BoostConverter_HDL', ...
'FloatingPointTargetConfiguration', hdlcoder.createFloatingPointTargetConfig('NativeFloatingPoint' ...
, 'LatencyStrategy', 'MIN') ...
);
hdlset_param('gmStateSpaceHDL_BoostConverter_HDL', 'HDLSubsystem', 'gmStateSpaceHDL_BoostConverter_HDL');
hdlset_param('gmStateSpaceHDL_BoostConverter_HDL', 'MaskParameterAsGeneric', 'on');
hdlset_param('gmStateSpaceHDL_BoostConverter_HDL', 'Oversampling', 60);

% Set SubSystem HDL parameters
hdlset_param('gmStateSpaceHDL_BoostConverter_HDL/HDL Subsystem', 'FlattenHierarchy', 'on');

% Set SubSystem HDL parameters
hdlset_param('gmStateSpaceHDL_BoostConverter_HDL/HDL Subsystem/HDL Algorithm/State Update/Multiply State', ...
'SharingFactor', 1);
```

The HDL parameters that are saved indicate that the model has the native floating-point mode enabled and uses an **Oversampling factor** of 60 and has **Latency Strategy** set to MIN. This default combination of HDL parameters that are saved on the model offers an optimal trade-off between oversampling factor and the target FPGA clock frequency and improves the sampling frequency. If you want to further improve the sampling frequency, you can reduce the number of iterations and the oversampling factor as illustrated below.

Reducing Number of Solver Iterations

The Simscape HDL Workflow Advisor uses an iterative method that determines the number of iterations required to calculate the mode in each time step. If **N** is the **Number**

of solver iterations, the HDL implementation model runs at a sample rate that is N times faster than the original Simscape algorithm. By default, for switched linear models, the **Number of solver iterations** is 5. The generated HDL implementation model runs 5 times faster than the original Simscape model.

To improve the sampling rate, you want to reduce the number of solver iterations. The number of solver iterations depends on various factors such as the complexity of your design, the number of modes in the design that the workflow must calculate, and so on. In the **Generate implementation model** task of the Simscape HDL Workflow Advisor, start by reducing the **Number of solver iterations** to a value such as 3, select **Generate validation logic for the implementation model**, and then generate the HDL implementation model. Simulate the HDL implementation model and open the Diagnostic Viewer to make sure that the model does not display any warnings or assertions.

If you see warnings or assertions, it indicates a simulation mismatch because the number of solver iterations that you specified is not adequate to compute the required number of modes in the state-space design. To resolve the warnings, you may want to increase the **Validation logic tolerance** to a value such as $1e-4$ and then simulate the HDL implementation model. If you still see warnings, increase the **Number of solver iterations**.

Note It is recommended that you do not change the **Floating-point precision** to double. Double-precision operations have large latencies and require a large **Oversampling factor** to allocate sufficient delays for the floating-point operations, which reduces the sampling frequency.

To learn more about how you can resolve the simulation mismatch, see “Validate HDL Implementation Model to Simscape Algorithm” (HDL Coder).

Using Oversampling Factor and Latency Strategy

The Oversampling factor specifies the factor by which the FPGA clock rate is a multiple of the HDL implementation model base sample rate. The HDL implementation model contains feedback loops and performs multiplication of large matrices that have floating-point data types inside the feedback loops. To accommodate the large latency introduced by these floating-point operations inside the feedback loops, the code generator uses a large value of oversampling factor in conjunction with the clock-rate pipelining optimization on the model. For more information, see “Strategy 1: Global Oversampling” (HDL Coder).

In the Configuration Parameters dialog box, on the **HDL Code Generation > Floating Point** pane, you can specify a **Latency Strategy** of Min, Max, or Zero on the model. The oversampling factor and latency strategy of the floating-point operator must be varied in conjunction. By default, the oversampling factor of 60 and minimum latency strategy gives an optimal sampling frequency. In most cases, when you use minimum latency strategy, you can start with an **Oversampling factor** of 60, generate HDL code, and then further reduce the oversampling factor until the code generator is unable to allocate delays for the floating-point operations.

If you specify Max latency, the floating-point operations introduce the maximum number of delays. To make sure that these delays can be allocated, increase the **Oversampling factor**. You may want to increase the **Oversampling factor** to a value such as 100 depending on the complexity of your HDL design. The maximum latency strategy provides the maximum FPGA clock frequency that you can achieve. If the increase in FPGA clock frequency outweighs the increase in oversampling factor, you can use the maximum latency strategy instead.

For the boost converter model, the default settings of **Number of solver iterations** set to 5, **Oversampling factor** set to 60, and **Latency Strategy** set to Min provides the optimal sampling frequency.

See Also

Functions

`simscape.findNonlinearBlocks` | `sschdladvisor`

More About

- “Solvers for Real-Time Simulation” (Simscape)
- “Latency Considerations with Native Floating Point” (HDL Coder)
- “Deploy Simscape™ Plant Models to Speedgoat FPGA I/O Modules” (HDL Coder)