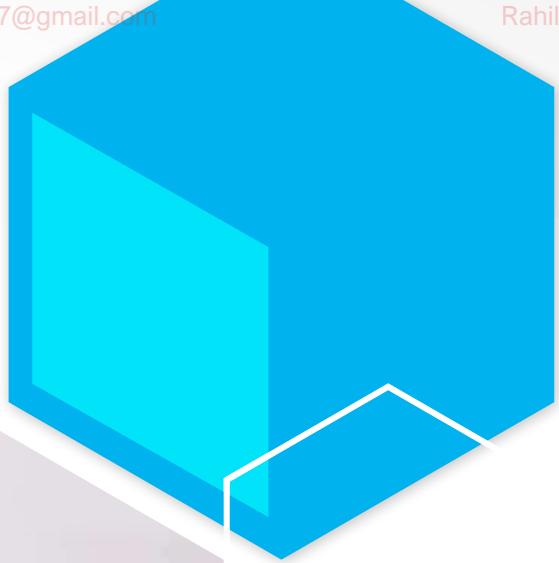




Electric Vehicle Performance Analysis with Simulink & Simscape (EV-PASS-2024)



Simulink® Fundamentals for Automotive Applications

Table of Contents

SLBE-A1023V1

TOC - 1

Table of Contents

© COPYRIGHT 2010-2023 by The MathWorks, Inc.

This Training Course Notebook, along with other Training Course Examples and Exercises, shall at all times remain the intellectual property of The MathWorks, Inc. The MathWorks, Inc. reserves all rights in these materials. No part of these materials may be photocopied, reproduced in any form, or distributed without prior written consent from 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.

Table of Contents

Table of Contents

- 1. Introduction
- 2. Modeling Discrete Systems
- 3. Modeling Continuous Systems

Appendices

Introduction

MathWorks® at a Glance

MathWorks® is the leading developer of mathematical computing software in the world.

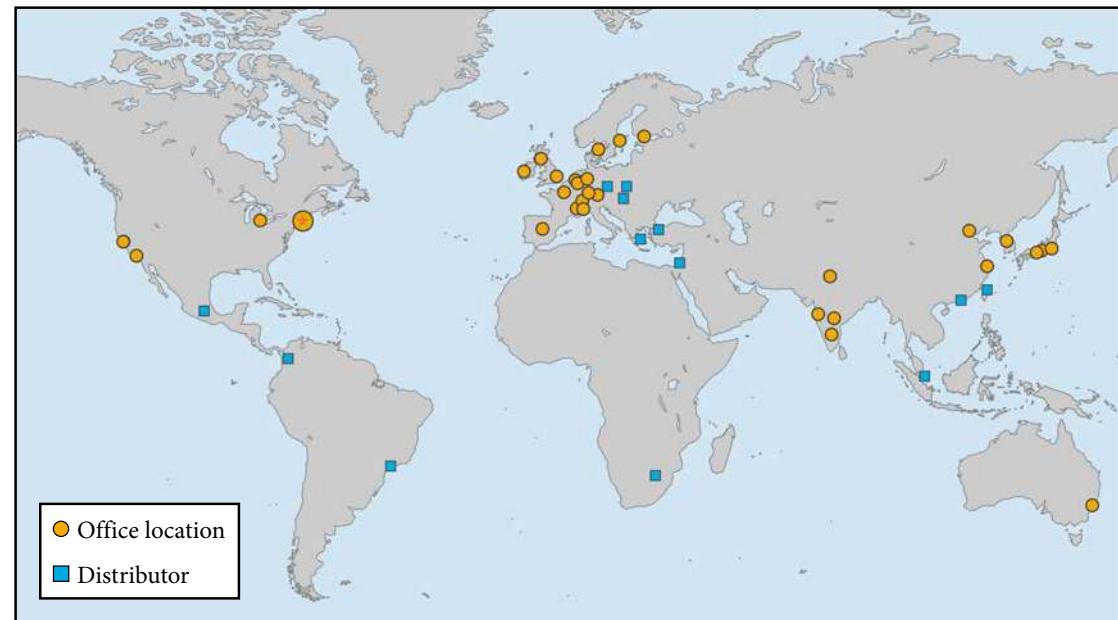
- Privately held
- Over 4000 employees worldwide
- More than 2 million users in 180+ countries

MathWorks products are relied upon by a variety of key industries to accelerate the pace of discovery, innovation, development, and learning, including

- Aerospace and defense
- Automotive
- Biological sciences
- Biotech and pharmaceutical
- Communications
- Electronics
- Energy production
- Financial services
- Industrial automation and machinery
- Medical devices
- Metals, materials, and mining
- Neuroscience
- Railway systems
- Semiconductors
- Software and internet

MathWorks supports programs that inspire learning and advance education in engineering, science, and math.

- 5000+ universities around the world
- 1800+ MATLAB® and Simulink® based books



- Academic support for research, fellowships, student competitions, and curriculum development

MathWorks supports its customers through a worldwide network of offices, distributors, and resellers.

Headquarters (Natick, MA USA)

www.mathworks.com
support@mathworks.com
+1-508-647-7000

Worldwide Offices

For information on any of our worldwide offices, please open the following location on the MathWorks site and choose a country:

www.mathworks.com/company/worldwide/

Introduction

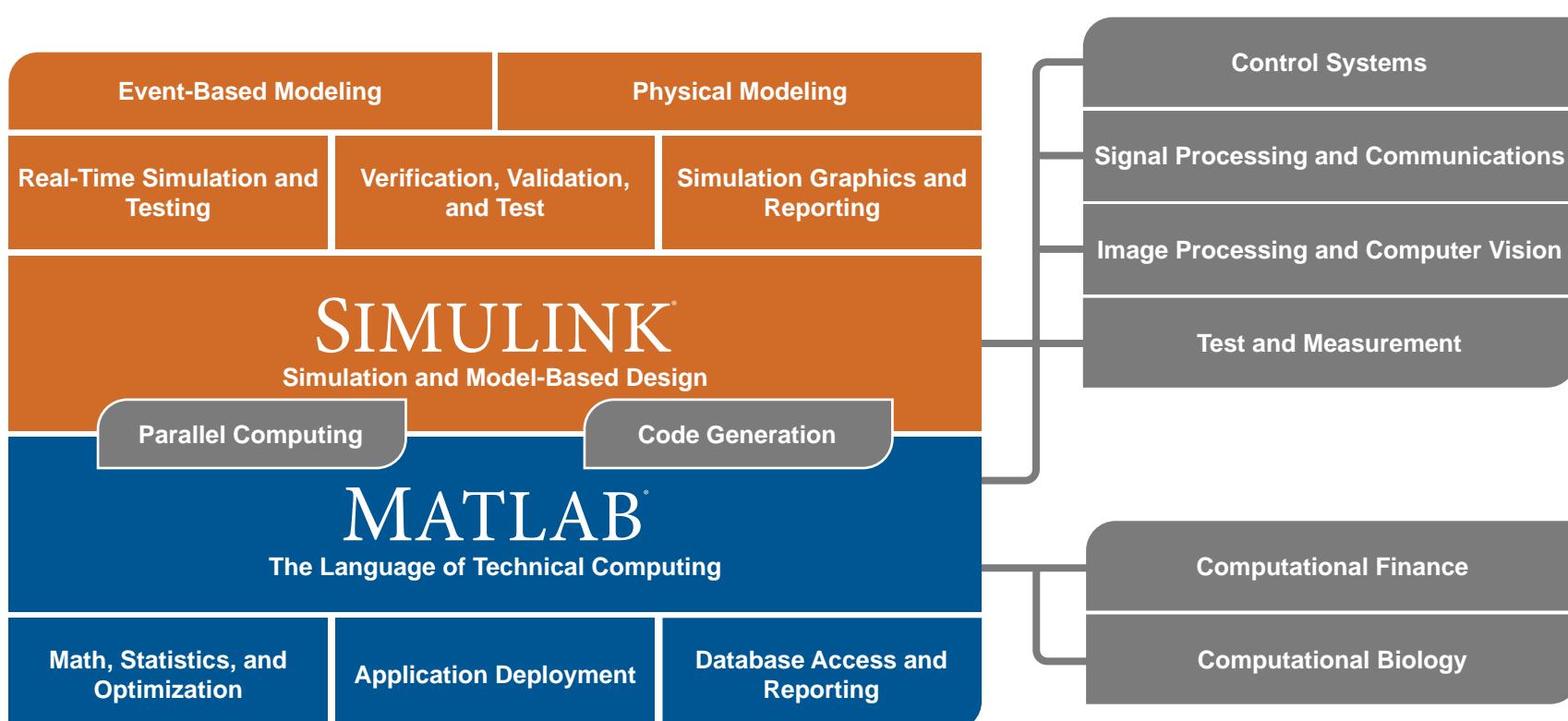
MathWorks® Product Overview

Key Characteristics of MATLAB

- The industry-standard, high-level programming language for algorithm development
- Numeric computation
- Parallel computing, with multicore and multiprocessor support
- Data analysis and visualization
- Toolboxes for signal and image processing, statistics, optimization, symbolic math, and other areas
- Tools for application development and deployment

Key Characteristics of Simulink

- Linear, nonlinear, discrete-time, continuous-time, hybrid, and multirate systems
- Foundation for Model-Based Design, including multidomain system modeling, real-time testing, automatic code generation, and verification and validation
- Open architecture for integrating models from other tools
- Applications in controls, signal processing, communications, physical modeling, and other system engineering areas

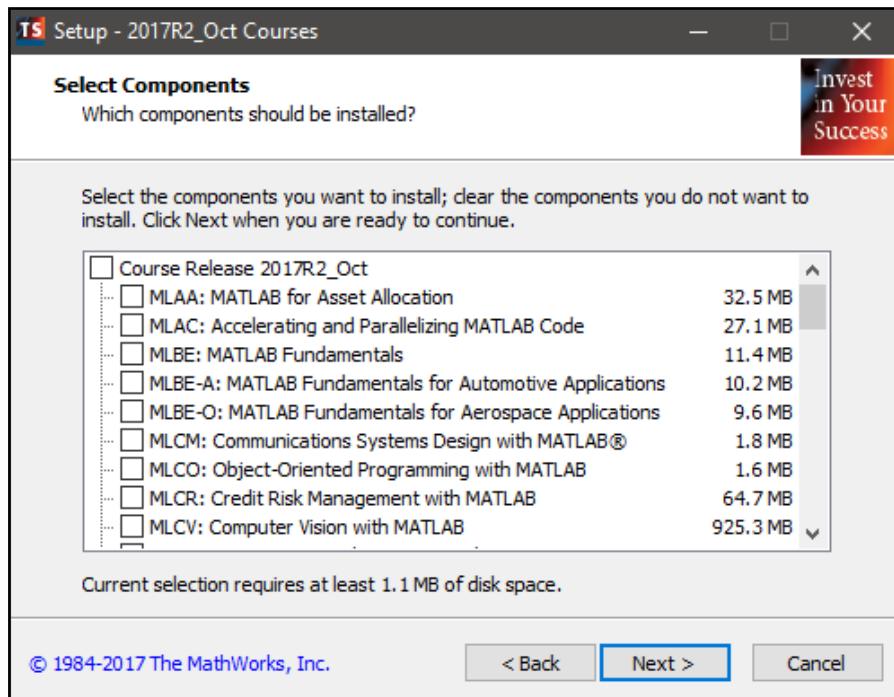


Introduction

Computer Setup

To get ready for class, you need to install the examples and exercises from your course DVD or from a USB drive provided by the instructor. Follow these steps:

1. Put the DVD in the DVD drive or plug in the USB drive.
2. The installer application will open automatically. If the installer application does not open automatically, open the DVD drive or USB drive in Windows® Explorer. Run the file **English_20XXRX_MMM.exe**.
3. Follow the prompts in the installer through the installation process. A shortcut will be created on your desktop to start MATLAB for this class.

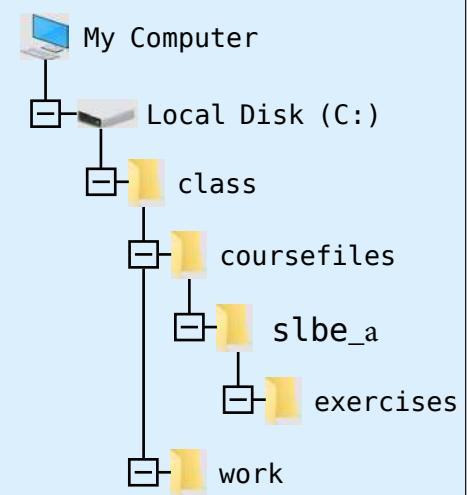


Typical setup

The installation creates a subdirectory of your chosen root directory (the default is C:\class) called **coursefiles**. This directory has subdirectories for each of the courses you install.

Each course directory has a subdirectory called **exercises** that contains all exercises and their solutions.

There is a subdirectory called **work**, which is empty. During class, put all your work in this subdirectory, so that it is on the path and easy to find.



In the installer, you have the following options:

- Choose a class root directory for your course files.
- Choose the courses for which you need to install the files. (Examples and exercises for all of our courses are on the DVD or USB drive.)
- Create a shortcut on the desktop to start MATLAB for this class. (This shortcut runs a **startup.m** file when starting MATLAB, that is customized for the installed course files)

Note A minimalistic install can be performed by navigating to the DVD drive or USB drive in MATLAB, executing the **installer.m** script inside the **zipfiles** folder, and selecting the appropriate courses to install.

Introduction

What Can You Do with Simulink®?

Simulink is a visual design tool in the overall MATLAB computational environment that facilitates Model-Based Design.

Simulink provides a graphical modeling environment for continuous-time, discrete-time, and hybrid system development and design. Simulink contains a set of block libraries that enable you to accurately design, simulate, implement, and test a variety of systems.

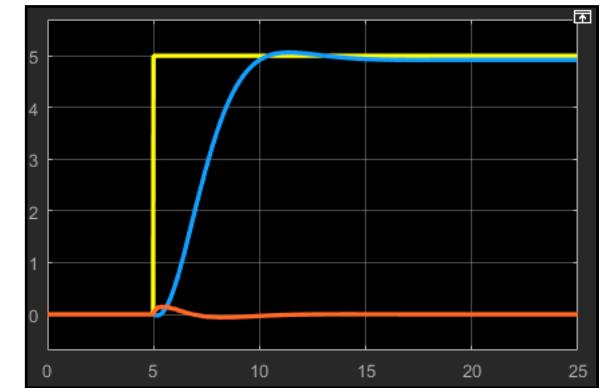
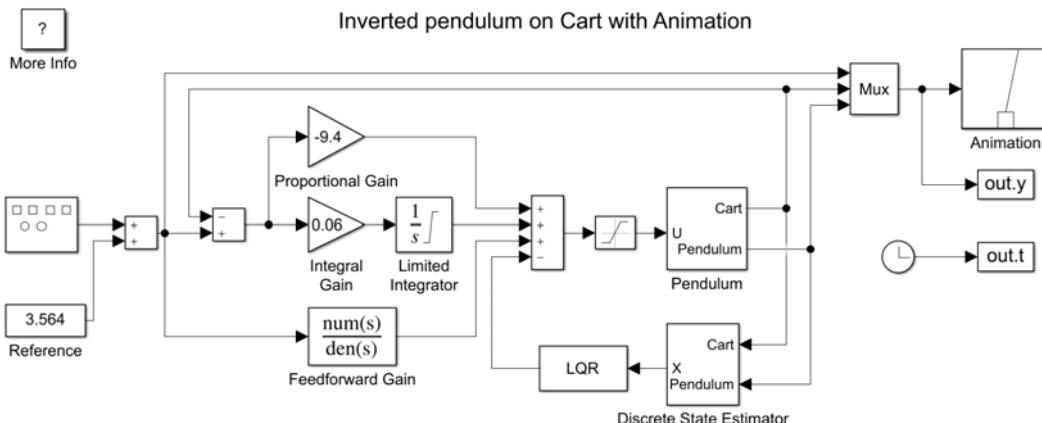
- Physical systems (continuous-time differential equations)
- Control algorithms (discrete-time difference equations)
- Filters and sensors

Simulink is extensible for custom feature development and distribution. You can create custom blocks from personal designs and distribute them by creating custom libraries. Also, Simulink supports the integration of custom and legacy code, including MATLAB code.

Try

Explore example Simulink models provided in the documentation. Select **Help > Examples** in the **Home** tab in the MATLAB toolbar. Then in the **Category** list on the left side of the documentation page, click **Simulink**.

Finally, Simulink is integrated with MATLAB, providing immediate access to an extensive range of tools for algorithm development, data visualization, data analysis and access, and numeric computation.



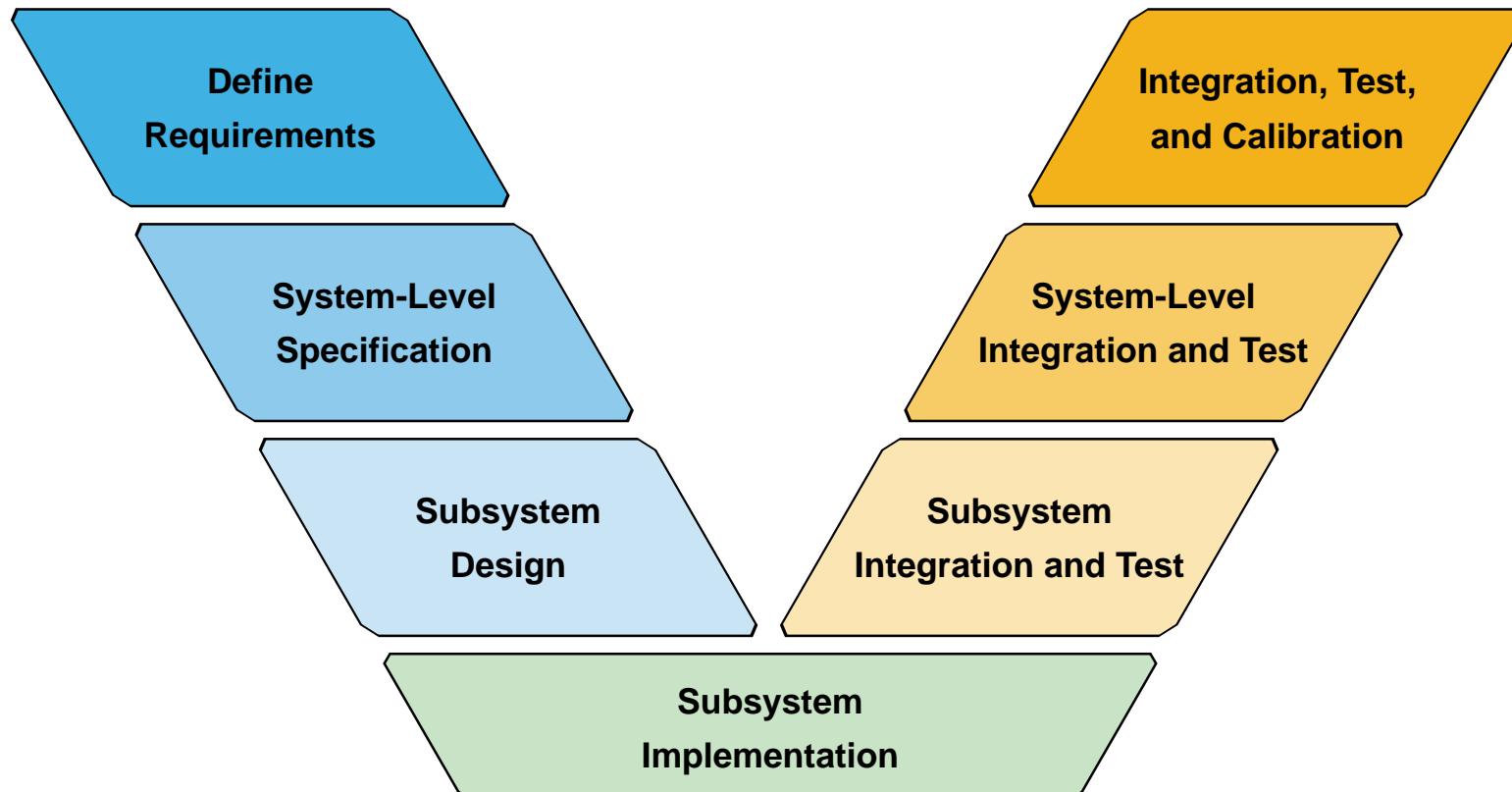
Introduction

System Design Process

The systematic design and realization process in many industries is represented by a V diagram as shown below. The two branches of the V correspond to two distinctly different activities:

- The left branch captures the decomposition of the initial system requirements into subsystems and components that are specified and implemented at an increasingly detailed level.
- The right branch represents the realization of these subsystems and components and their integration.

In the traditional approach, strict boundaries between design activities are enforced and documents are passed back and forth between the respective teams of engineers. This approach has many drawbacks. For instance, using documents as deliverables leads to a duplication of effort when information is needed in an electronic format. It is difficult to trace the source of errors along a paper trail.



Introduction

Model-Based Design with Simulink®

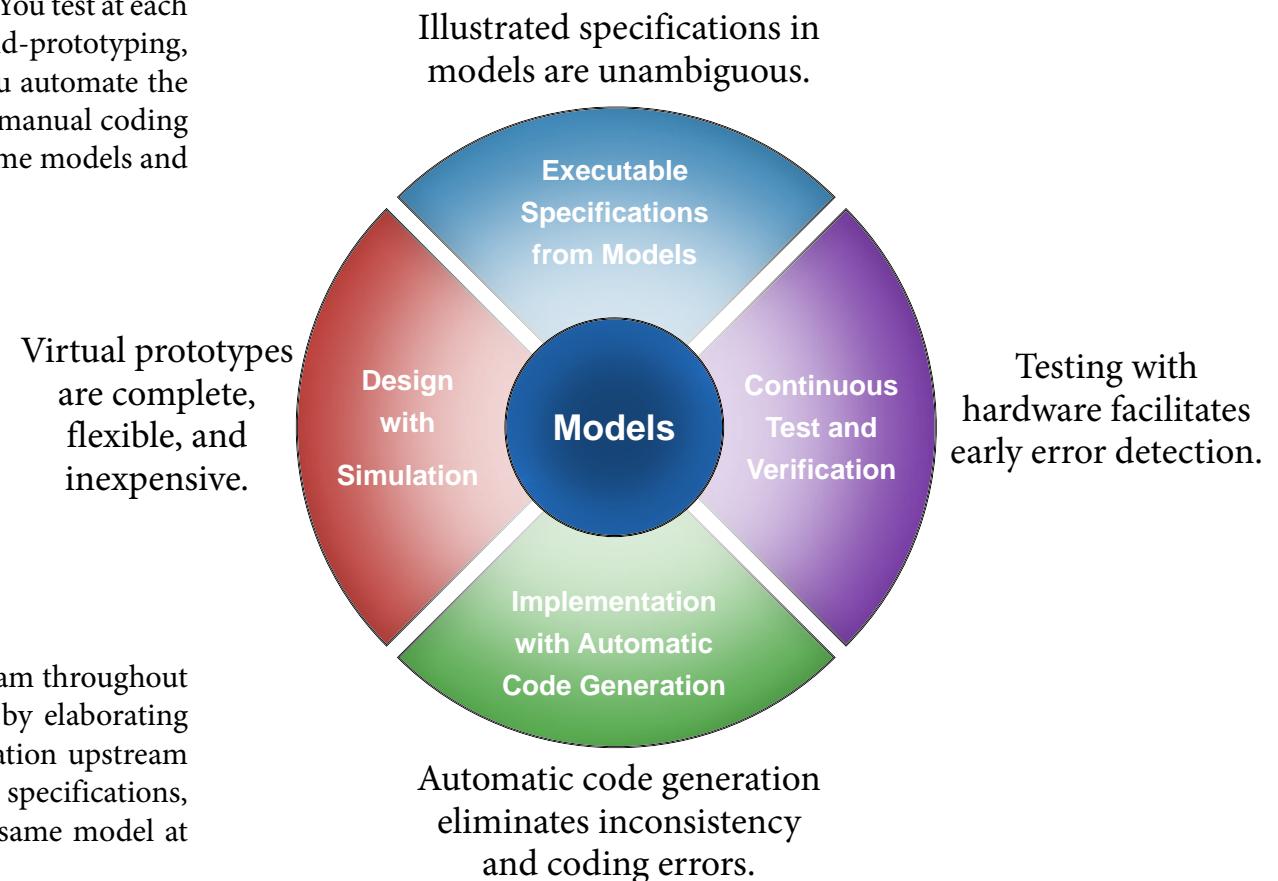
Model-Based Design refers to the process of using models throughout a design process. It starts with the conceptual phase, in which you might try out an idea using a graphical model and simulation. You elaborate the model, adding more detail at each design iteration – for example, adding data types as you near the implementation phase for an embedded system. You test at each design iteration using any combination of simulation, rapid-prototyping, hardware-in-the-loop, and model-coverage information. You automate the implementation through code generation, which eliminates manual coding errors. By tying together all these steps, all groups use the same models and talk the same language.

In Model-Based Design, the model is at the center of the design process, and the tasks you do surround the model. The model is the specification, the design, and the implementation, all in one place.

Executable models allow you to maintain one representation of the system you are designing and implementing. By using one model to simulate the system behavior, you can reduce the dependence on physical prototypes and rely on software prototypes.

Model-Based Design pushes the executable model downstream throughout the design flow all the way down to test and verification, by elaborating that same model through the phases. It also pushes verification upstream throughout the design flow all the way to requirements and specifications, allowing you to test and verify at every stage, and use the same model at every stage of your design.

Through this process, Model-Based Design enables a circular closed loop of productivity, efficiency, and innovation.



Introduction

Course Outline

Day 1

- Introduction
- Creating and Simulating a Model
- Modeling Programming Constructs
- Modeling Discrete Systems
- Modeling Continuous Systems

Day 2

- Developing Model Hierarchy
- Modeling Conditionally Executed Algorithms
- Referencing Model Components
- Creating Libraries
- Conclusion

Appendices

- Debugging Simulink® Models
- User Interface Reference
- Rate Transitions in Multirate Models
- Automating Modeling Tasks
- MATLAB® Review
- Solver Selection
- Exercises

Introduction

Overview of Dynamic Systems

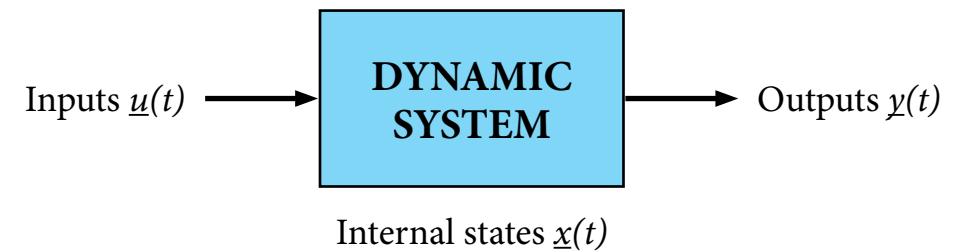
When modeling a dynamic system, it is possible to develop a representation in the generalized state-space form. This representation consists of these main equations:

- **Continuous state dynamics** – The first derivative of the state vector is a function of the state, input, and time.
- **Discrete state dynamics** – The next value of the state depends on the current state, input, and time.
- **Outputs** – The output value is a function of states, inputs, and time. This equation must not involve any dynamics.

The graphic on the right is a general representation of a dynamic system. Simulink assumes this construct for all blocks.

Using equations to model dynamic systems is referred to as *First Principles* modeling. You can use add-on tools to Simulink to model systems without having to derive equations. For example:

- Stateflow® models finite state machines.
- Simscape™ Multibody™ models rigid body dynamics.
- Simscape™ Electrical™ models electrical systems.



$$\dot{\underline{x}}(t) = f(\underline{x}(t), \underline{u}(t), t) \quad \leftarrow \text{Continuous state dynamics}$$

$$\underline{x}(t_{k+1}) = f(\underline{x}(t_k), \underline{u}(t_k), t_k) \quad \leftarrow \text{Discrete state dynamics}$$

$$\underline{y}(t) = g(\underline{x}(t), \underline{u}(t), t) \quad \leftarrow \text{Output equation}$$



Power Electronics Control Design with Simulink® and
Simscape™

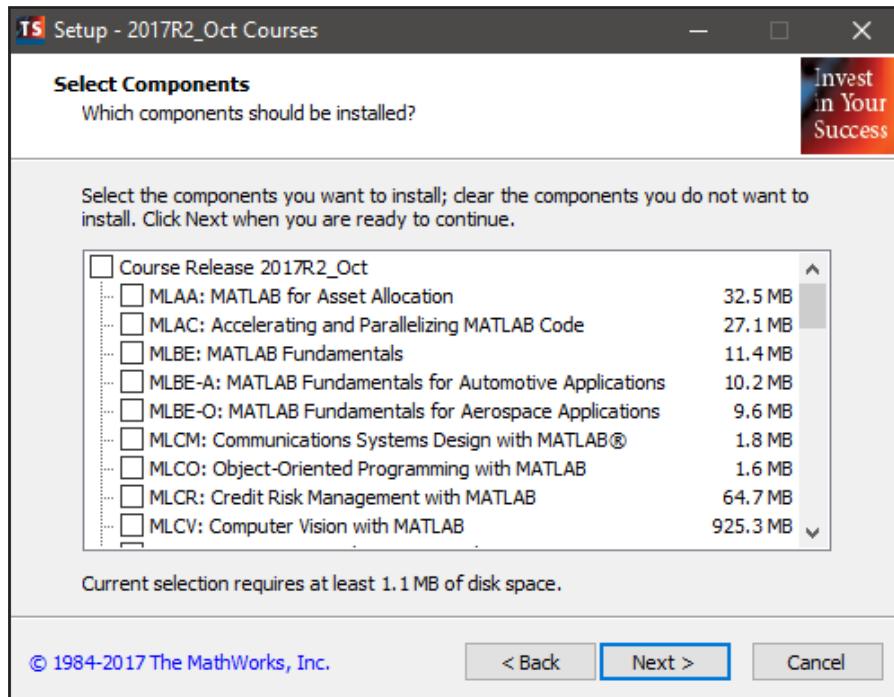
Introduction

Introduction

Computer Setup

To get ready for class, you need to install the examples and exercises from your course DVD or from a USB drive provided by the instructor. Follow these steps:

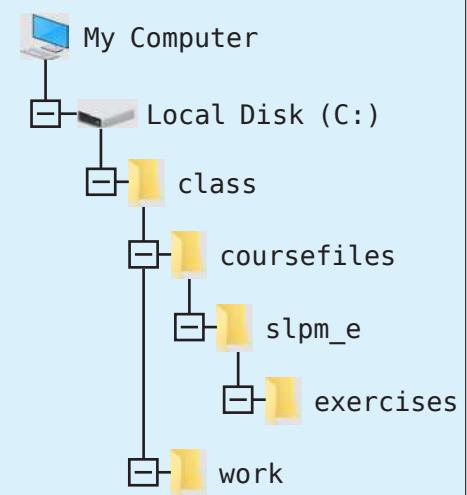
1. Put the DVD in the DVD drive or plug in the USB drive.
2. The installer application will open automatically. If the installer application does not open automatically, open the DVD drive or USB drive in Windows® Explorer. Run the file **English_20XXRX_MMM.exe**.
3. Follow the prompts in the installer through the installation process. A shortcut will be created on your desktop to start MATLAB for this class.



Typical setup

The installation creates a subdirectory of your chosen root directory (the default is **C:\class**) called **coursefiles**. This directory has subdirectories for each of the courses you install.

Each course directory has a subdirectory called **exercises** that contains all exercises and their solutions.



There is a subdirectory called **work**, which is empty. During class, put all your work in this subdirectory, so that it is on the path and easy to find.

In the installer, you have the following options:

- Choose a class root directory for your course files.
- Choose the courses for which you need to install the files. (Examples and exercises for all of our courses are on the DVD or USB drive.)
- Create a shortcut on the desktop to start MATLAB for this class. (This shortcut runs a **startup.m** file when starting MATLAB, that is customized for the installed course files)

Note A minimalistic install can be performed by navigating to the DVD drive or USB drive in MATLAB, executing the **installer.m** script inside the **zipfiles** folder, and selecting the appropriate courses to install.

Introduction

The Project Environment

The course files provided for this course are organized in a Project to provide a friendly environment to manage files within the file system. You can open the project interface by navigating to the slpm_e folder and double-clicking the `slpm_e.prj` file.

The SLPM-E project is structured as follows:

- **artifacts** – Power semiconductor device datasheets and parameterization scripts needed by the models in Chapter 3
- **coursefiles** – Simulink model files organized in numbered chapter subfolders
- **libraries** – Simulink library files
- **scripts** – MATLAB scripts to prepare and clean up the project environment
- **tests** – Data used by components for the system-level verification

The Project window provides two different file view options:

- **All** – Shows all files in the project folder
- **Project** – Shows only files that have been added to the project

Try

Open the SLPM-E project. Familiarize yourself with the environment.

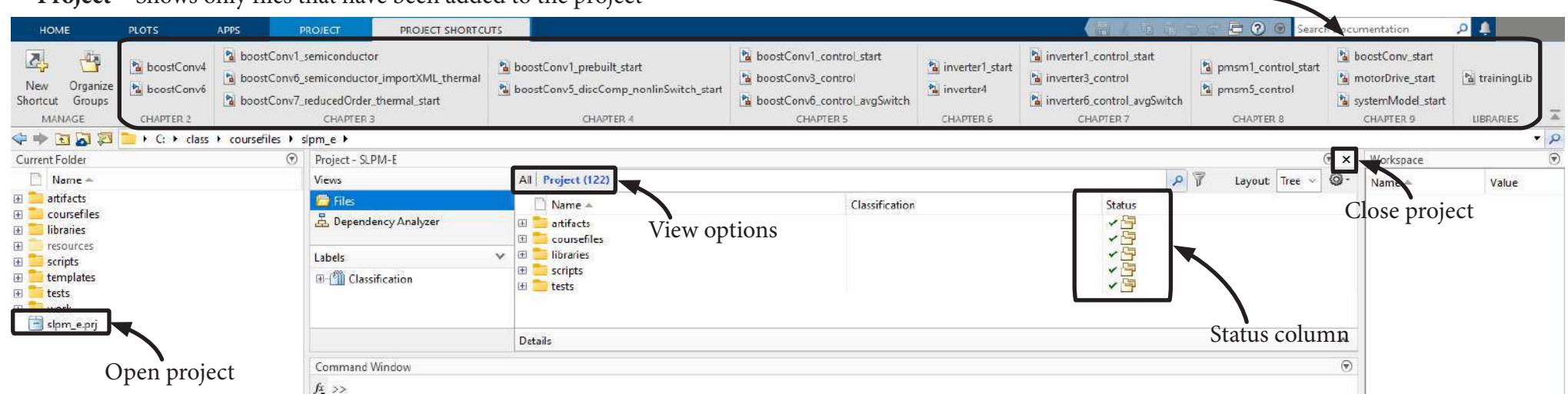
The icons in the **Status** column indicate whether each file or folder is

- **In the project** – Indicated with a green check ()
- **On the project path** – Indicated with a folder ()

Throughout this course, you will be creating new Simulink model files and you will need to add them to the project. Select the **All** view. Then, right-click the files you want to add to the project and select **Add to Project**. You can also select **Add Files** accessible from the drop-down menu of the **Tools** section from the **Project** tab to quickly add multiple files to the project.

Given the large number of files that are included in this project, it can become more difficult for you to find specific files. Shortcuts are provided to ease the access to certain files. Shortcuts appear in the **Project Shortcuts** tab and are organized into shortcut groups.

Refer to the training course *Simulink Model Management and Architecture* to learn how to create a project from scratch and how to use other useful features and tools for project management (such as source control integration).

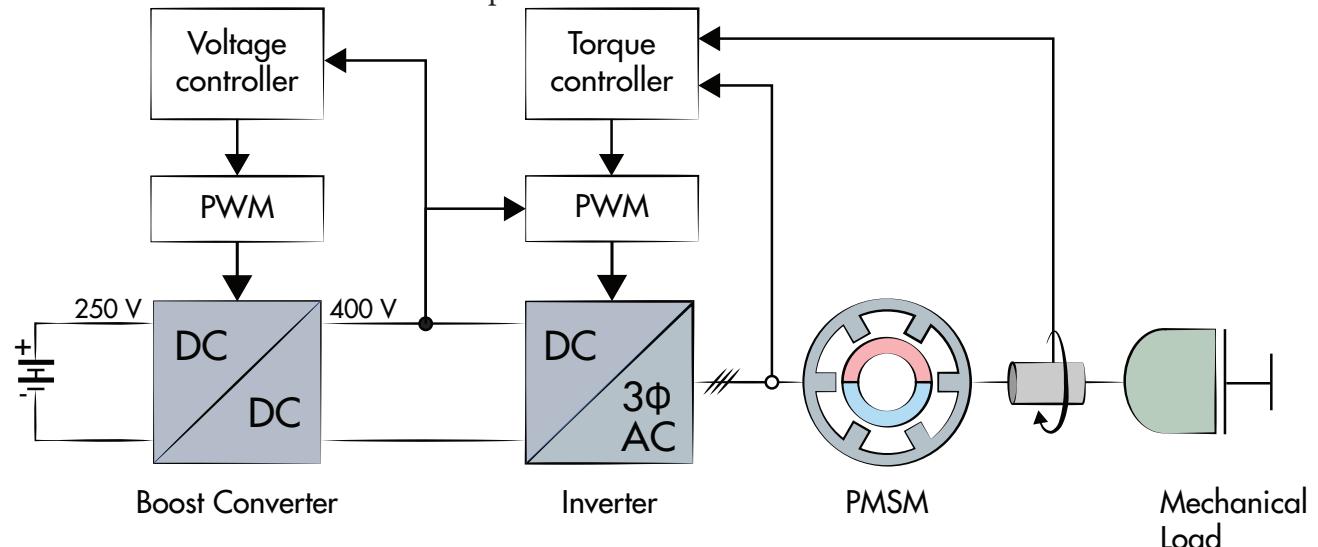


Introduction

Course Example: Hybrid-Electric Vehicle Motor Drive

Throughout the chapters of this course, you will learn the necessary skills to build a hybrid electric vehicle (HEV) motor drive model, specifically one capable of delivering 60 kW of mechanical power at 1250 rpm. This electromechanical system consists of the following elements:

- **Boost converter** — This is a direct current (DC) power electronic converter that steps up voltage. The output voltage is controlled using a voltage controller and a pulse width modulation (PWM) generator. This is covered in Chapters 2, 3, 4, 5.
- **Inverter** — This is a power electronic converter that converts DC into three-phase alternating current (AC). The three-phase output is controlled using a current controller and a PWM generator. This is covered in Chapters 6 and 7.
- **Permanent magnet synchronous motor (PMSM)** — This is an electric motor that converts three-phase electrical energy into rotational mechanical energy, characterized by torque and rotational speed. A torque controller controls the motor. This is covered in Chapter 8.



Try

Open and inspect the final system-level model that you will develop throughout this course. Identify each of the individual components.

`>> systemModel_final`

The power electronic converters are comprised of semiconductor switches that rapidly switch on and off. Simulating such systems can be computationally demanding due to the differing timescales of the fast switching behavior and the slower overall response. You will learn how to model these semiconductor switches, how to parameterize them, and how to select appropriate solver settings to successfully simulate them.

Introduction

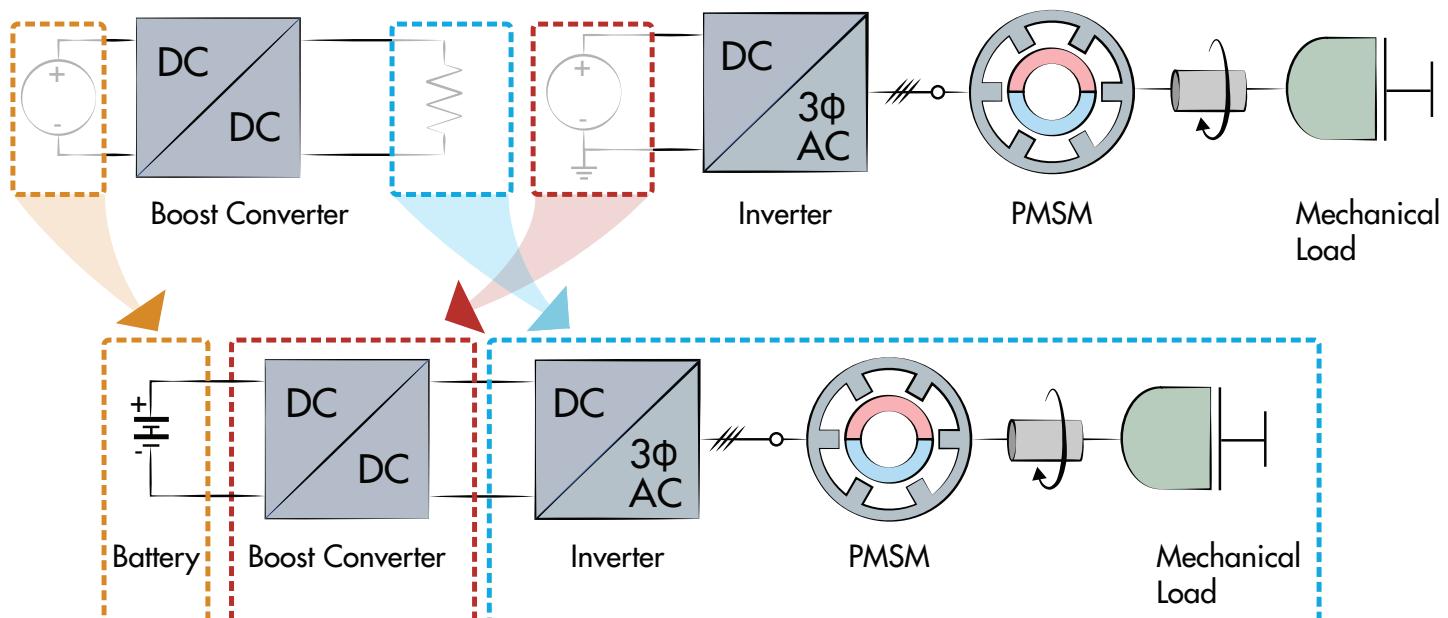
Course Fundamental Concepts

You will notice a common theme in this course to incrementally add complexity to a model while continuously testing. This is a best practice to model and test the system components separately before attempting to build a system-level model. Verifying the individual components will build confidence in the components and help you identify and isolate issues early. There are three main modeling concepts that are covered throughout the course in support of this best practice.

Incremental Modeling Approach — Rather than modeling the whole HEV motor drive system comprising of all functional components, prefer an incremental modeling approach. Identify the source and load subsystems of your power electronic system and use a lumped load resistor and a lumped voltage source for the source and load subsystems, respectively. Then, model the subsystems, design the controller, and test and verify the closed-loop dynamic performance treating the individual subsystems as standalone units.

Converter Modeling Fidelity — Modeling the switching effects may play a key role in modeling the standalone subsystem because real power electronic converters are switching by nature. However, simulating the switching is challenging for desktop simulation because of the discontinuous behavior of this operation. Identification and implementation of the proper converter model fidelity for each of the development phases may significantly speed up the simulation time and yet produce the desired accurate results in a controllable fashion.

System-Level Integration — Besides the due considerations on the topology for connecting the source and load subsystems in cascade, consider the converter model fidelity to be used when simulating the whole system and try to answer the following questions. Is switching important to simulate at the system level? What are the dynamics I would like to simulate? How can I architect my system-level model for the next phases of the Model-Based Design development? These questions will be answered in Chapter 9.





Power Electronics Control Design with Simulink® and
Simscape™

Modeling DC/DC Power Electronic Converters

Outline

- Boost converter operation principles
- Modeling an open-loop boost converter
- Measuring physical quantities
- Selecting a solver
- Inspecting simulation results
- Using prebuilt PWM blocks

Chapter Learning Outcomes

The attendee will be able to

- Build and simulate DC/DC power electronic converters using discrete component modeling approach.
- Sense and visualize physical quantities.
- Select solver settings suitable for power electronics models.
- Understand and use appropriate PWM techniques.

Course Example: Boost Converter

In this chapter, you will learn how to model a boost converter that increases the voltage supplied by the hybrid electric vehicle's (HEV) batteries from 250 to 400 Volts to drive the motor.

The boost converter, also known as a step-up converter, is a power electronic device that increases the voltage and decreases the current of a DC source. A typical boost converter is shown in the circuit on the right, consisting of an inductor, a semiconductor switch, a diode, and a capacitor.

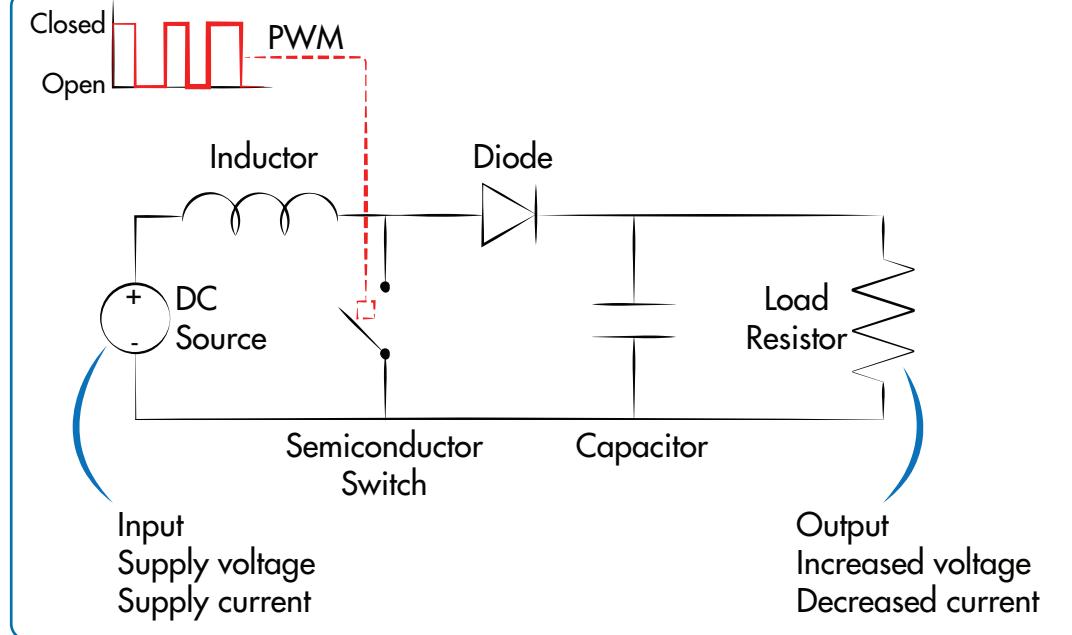
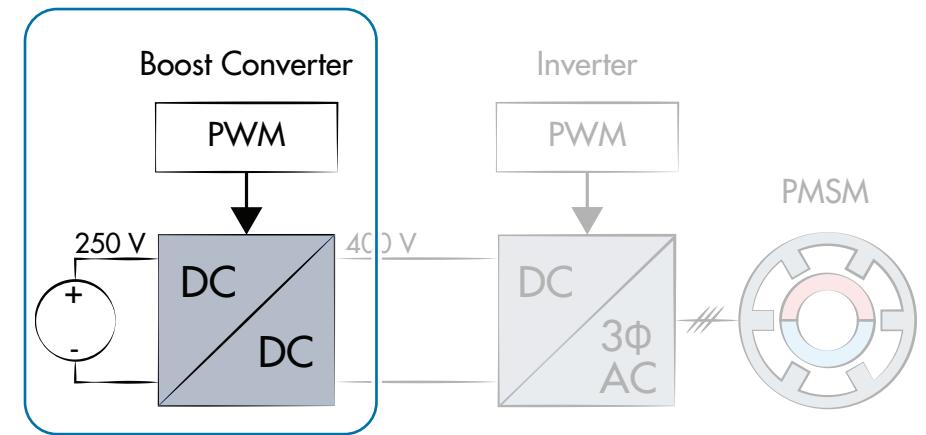
To understand the boost converter operation, consider both states of the semiconductor switch while the circuit is in steady state:

- **Switch closed** – On the input side, the DC source charges the inductor up to the source voltage. On the output side, current flows from the capacitor to the load (i.e., resistor). The diode prevents current from flowing from the capacitor to ground through the closed switch.
- **Switch open** – When the switch opens, current flows from the inductor to charge the capacitor and to the load.

For every power electronic converter topology, in steady-state, the net flux linkage of the inductor is zero over each switching period. This is the principle of *inductor volt-second balance* and brings the designer to find steady-state conditions for any topology. For the boost converter, applying this principle leads to the output voltage greater than the source voltage with the relationship shown in the next page.

The switching behavior in the circuit is specified using a pulse width modulation (PWM) signal that controls when the switch is open and closed. The percentage of time that the switch is closed over a single switching period is called the *duty cycle*. Changing the duty cycle controls the output voltage of the boost converter.

In this chapter, you will use the Simscape Electrical library to model an open-loop boost converter using discrete electrical components. You will use Simulink blocks to model a PWM generator.



Course Example: Boost Converter (Continued)

The boost converter in this chapter uses the design specifications provided in the table on the right. These specifications are selected to produce the following *steady-state operating conditions*:

1. The converter operates in *continuous current mode* (CCM), which means that the inductor current is always greater than zero.
2. The input current I_{in} is described with the following equation:

$$I_{in} = \frac{V_{in}}{(1 - D)^2 R}$$

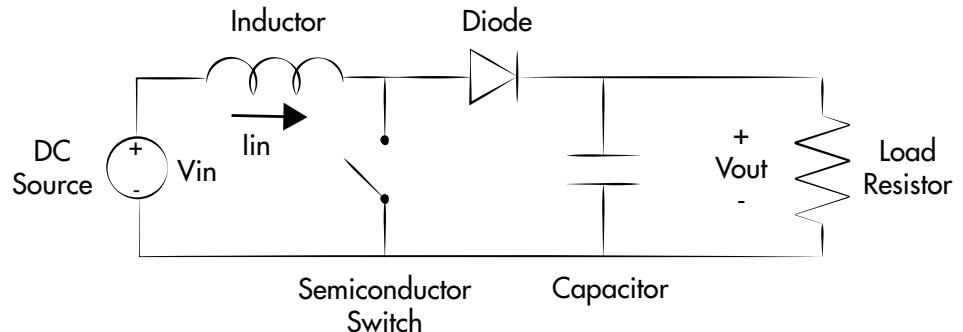
Here, V_{in} is the input voltage, D is the duty cycle, and R is the load resistance.

3. The output voltage V_{out} is described with the following equation:

$$V_{out} = \frac{V_{in}}{1 - D}$$

Note The above relationships refer to an ideal boost converter (no losses). The boost converter theoretically produces a voltage of 400 V and an input current of about 240 A when the PWM duty cycle is 37.5% in steady state. However, the boost converter modeled in this course is lossy. To compensate for the losses, the duty cycle value is rounded up to 40% (i.e., 0.4).

After you model the nonideal boost converter with the given design specifications, you add voltage and current sensors to test and verify whether the three operating conditions above are satisfied. Moreover, you can identify the lossy elements responsible for any discrepancy with respect to the results of an ideal boost converter by using the relationships above.



Element	Parameter	Value	Units
Voltage source	DC Voltage	250	V
Load resistor	Resistance	2.67	Ω
Inductor	Inductance	195	μH
	Series resistance	34.7	$\text{m}\Omega$
	Parallel conductance	0	$1/\Omega$
Capacitor	Capacitance	7	mF
	Series resistance	6.67	$\text{m}\Omega$
Diode	Forward voltage	0.6	V
	On resistance	2	$\text{m}\Omega$
	Off conductance	1e-8	$1/\Omega$
Semiconductor switch	On-state resistance	10	$\text{m}\Omega$
	Off-state conductance	1e-8	$1/\Omega$
PWM	Switching frequency	10	kHz
	Duty cycle	0.4	

Modeling DC/DC Power Electronic Converters

Creating a New Simscape Model

You can use the Simulink Start Page to create new models, open templates, find examples, and more. In this chapter, you create a power electronics model from a blank model. In later chapters, you will learn how to use electrical templates.

To create a new blank model,

1. Click **Simulink** in the MATLAB toolbar to open the Simulink Start Page.
2. Click **Blank Model** in the Simulink section of the New tab.

To model the boost converter, you can open the Library Browser and add components to the model from the **Simscape > Electrical** library.

Electrical networks modeled using Simscape components must contain the following two blocks:

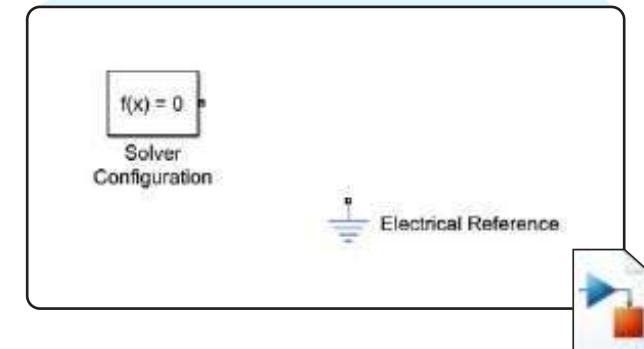
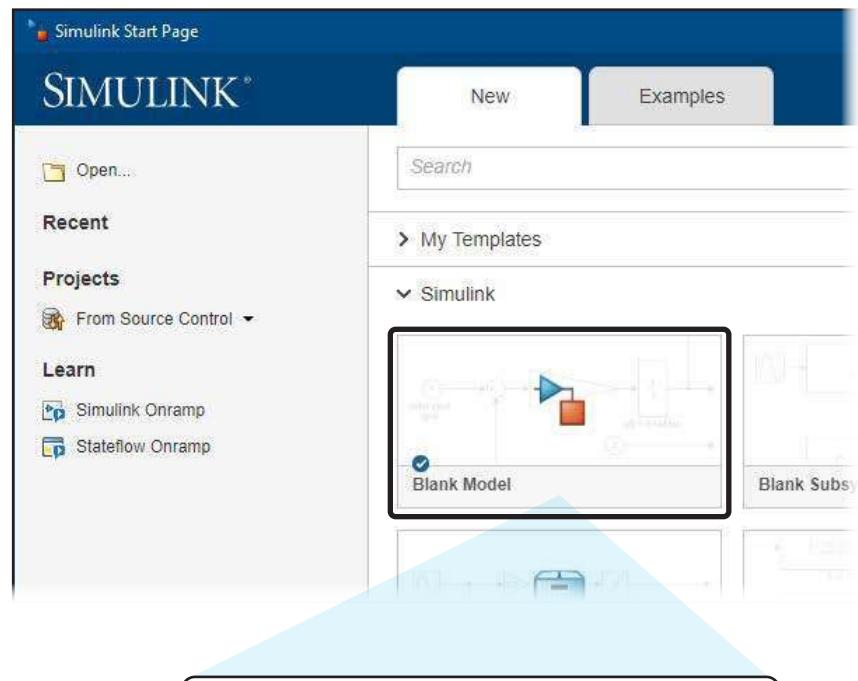
- Solver Configuration (**Simscape > Utilities** library) – This block defines the solver settings for a physical network. Each Simscape physical network must contain *only* one Solver Configuration block.
- Electrical Reference (**Simscape > Electrical > Connectors & References** library) – This block establishes a voltage reference level (i.e., ground). Each Simscape electrical network must contain *at least* one reference.

Note If you are unsure where to begin, try exploring the example models provided in the Simscape Electrical documentation. Many times, you may find a relevant model to use as a starting point. For more information, see **Simscape Electrical** in the documentation and click **Examples**.

Try

Use the Simulink Start Page to create a new blank model.

Then, add an Electrical Reference block and a Solver Configuration block to the model.



Exploring the Simscape Electrical Library

The Simscape Electrical library (**Simscape > Electrical**) contains Simscape components that use Simscape Foundation domains, including the Electrical and Three-Phase Electrical domains, for detailed modeling of electrical systems.

Simscape Electrical blocks are organized into the sublibraries listed below, with examples of some components inside:

- **Connectors & References** – electrical references, phase splitters
- **Control** – controllers, measurements, mathematical transforms
- **Electromechanical** – actuators, generators, motors, piezos, servos
- **Integrated Circuits** – operational amplifiers, timers, logic gates
- **Passive** – capacitors, inductors, resistors, transformers
- **Semiconductors & Converters** – diodes, transistors, power converters
- **Sensors & Transducers** – encoders, voltage sensors, current sensors
- **Sources** – batteries, voltage sources, current sources
- **Switches & Breakers** – breakers, fuses, relays, switches
- **Utilities** – environmental parameters and faults
- **Additional Components** – SPICE components
- **Specialized Power Systems** – power systems components using the specialized electrical domain

Some blocks in the Simscape Electrical library may appear similar to blocks in the Simscape Foundation Electrical library (**Simscape > Foundation Library > Electrical**). The Simscape Electrical blocks generally provide additional parameters not available in the foundation library. For example,

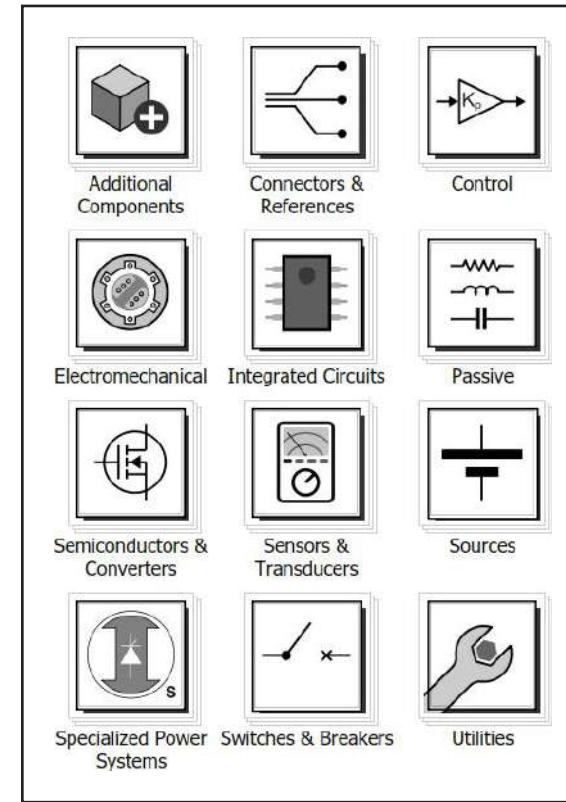
- The Resistor block in the **Foundation Library > Electrical > Electrical Elements** library provides only one parameter for resistance.
- The Resistor block in the **Electrical > Passive > Resistor** library provides multiple parameters for resistance, tolerance, operating limits, and faults.

Try

Open the Simscape Electrical library and explore the various components inside.

Compare the block parameters for similar blocks in the Foundation library and the Simscape Electrical library.

Note If you double-click the model canvas to add a block, pay close attention to which library the block is from. This will help avoid confusion between the Foundation library and Simscape Electrical blocks.



Modeling DC/DC Power Electronic Converters

Modeling a Boost Converter

To model the boost converter in the HEV course example, do the following:

1. Add the blocks listed below to your model.
 - Voltage Source (**Electrical > Sources** library)
 - Capacitor (**Electrical > Passive** library)
 - Inductor (**Electrical > Passive** library)
 - Resistor (**Electrical > Passive** library)
 - Diode (**Electrical > Semiconductors & Converters** library)
 - Ideal Semiconductor Switch (**Electrical > Semiconductors & Converters** library)
2. Arrange and connect the blocks according to the circuit diagram on the right. The lines connecting the circuit elements represent physical connections that exchange energy flows, characterized by voltage and current.
3. Change the name of the Resistor to Load.
4. Connect the Electrical Reference block to the negative port of the Voltage Source block to serve as the circuit ground.
5. Connect the Solver Configuration block to any point in the Simscape network to specify the solver parameters.

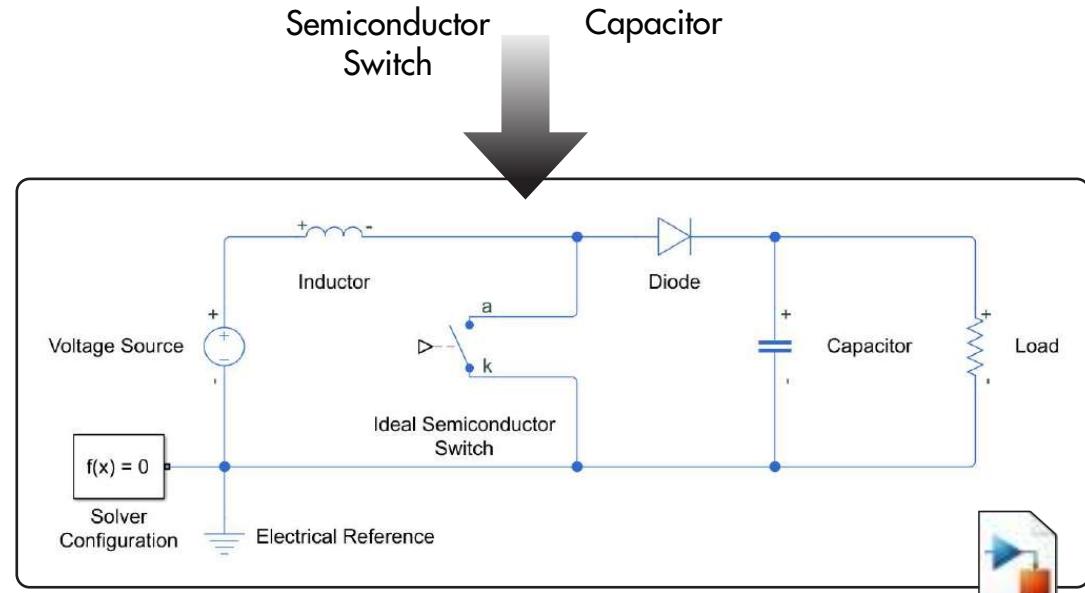
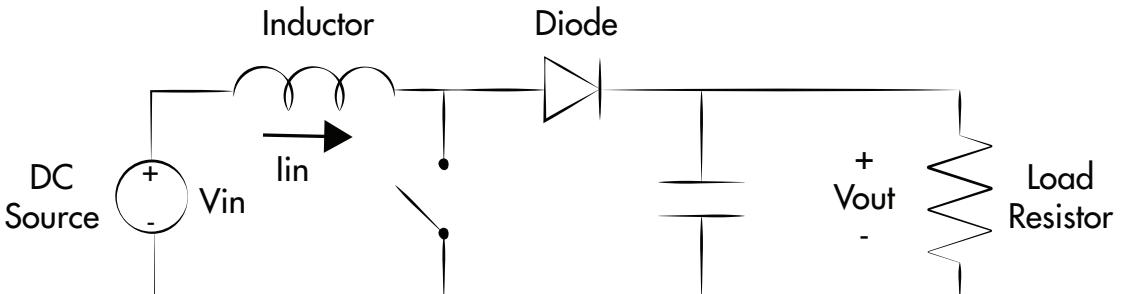
The Ideal Semiconductor Switch has an unconnected physical signal input that controls the switch. You will connect this to the PWM signal later.

Recall that physical signals in Simscape are directional and serve to carry signals between Simscape blocks. For more information, see **Simscape > Get Started with Simscape > Basic Principles of Physical Networks** in the documentation.

Note To branch an existing physical connection line, you can right-click the line and drag the mouse.

Try

Follow the steps listed on this page to build the boost converter model using blocks from the Simscape Electrical library.

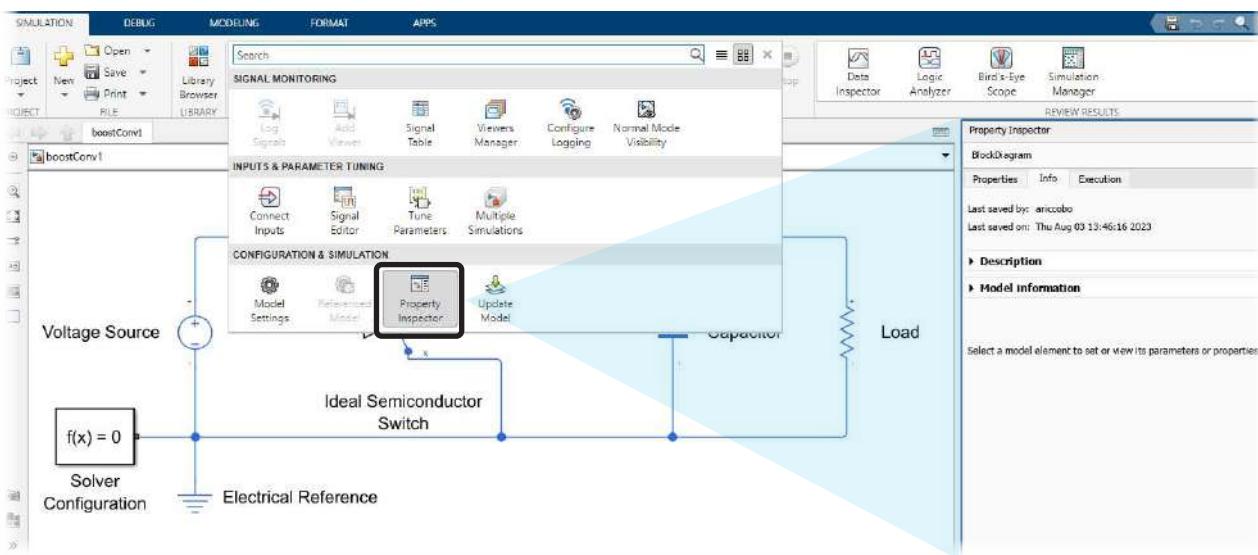


Modeling DC/DC Power Electronic Converters

Modeling a Boost Converter (Continued)

You must configure the parameters for each block in the boost converter circuit to match the design specifications in the table below.

Element	Parameter	Value	Units
Voltage source	DC voltage	250	V
Load resistor	Resistance	2.67	Ω
Inductor	Inductance	195	μH
	Series resistance	34.7	$\text{m}\Omega$
	Parallel conductance	0	$1/\Omega$
Capacitor	Capacitance	7	mF
	Series resistance	6.67	$\text{m}\Omega$
Diode	Forward voltage	0.6	V
	On resistance	2	$\text{m}\Omega$
	Off conductance	1e-8	$1/\Omega$
Semiconductor switch	On-state resistance	10	$\text{m}\Omega$
	Off-state conductance	1e-8	$1/\Omega$



Try

Use the Property Inspector to set the parameters for each block, based on the design specifications provided in the table on this page.

>> boostConv1

To set parameter values for a block,

1. Open the Property Inspector by clicking **Property Inspector** in the **Prepare** section of the **Simulation** tab.
2. Select a block in the model.
3. Use the **Property Inspector** pane in the model window to set relevant block parameters.

Note#1 The **Property Inspector** is also available in the **Design** section of the Modeling tab.

Note#2 You can add block annotations to improve readability by displaying block parameter values in the model canvas. To learn more about this, see **Simulink > Modeling > Configure Signals, States, and Parameters > Blocks > Specify Block Properties > Set Block Annotation Properties** in the documentation.

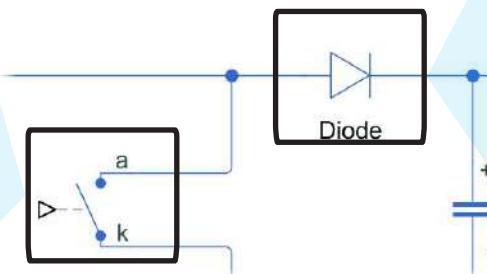
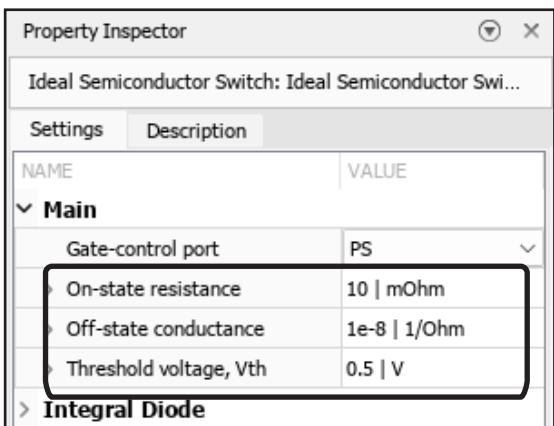
Modeling Ideal Semiconductor Switches

In this chapter, you are modeling the boost converter's main semiconductor switch with an Ideal Semiconductor Switch block and the diode with a Diode block. These blocks can be used to mimic ideal semiconductor switches that possess the following operating properties:

- No limit on the amount of current the device can carry in its on-state.
- No limit on the amount of voltage the device can block in its off-state.
- Zero or very little voltage drop when the device is in its on-state.
- Zero or very little off-state conductance, that is zero or very little current when the device is in the off-state.
- Zero switching time when the device changes from its on-state to its off-state and vice versa.
- It dissipates zero or nearly zero power.

Ideal Semiconductor Switch

If the gate-cathode voltage, that is the voltage at the PS input with respect to the k terminal, exceeds the specified threshold voltage, the Ideal Semiconductor Switch is in the on state and behaves like a linear resistor with the resistance specified in the **On resistance** parameter. Otherwise, the device is in the off state and behaves like a linear resistor with the small conductance specified in the **Off conductance** parameter.



Try

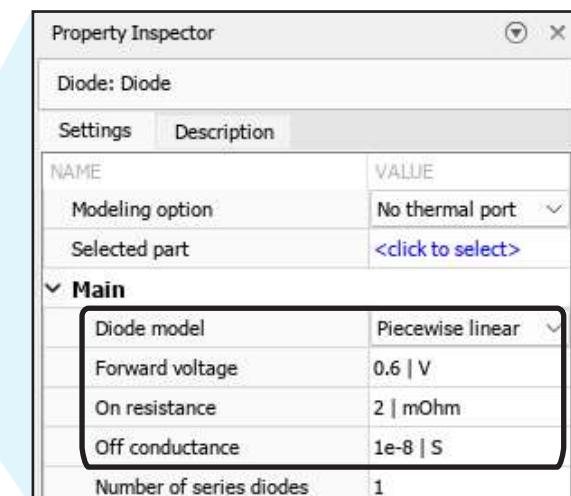
Analyze the main parameters of the Ideal Semiconductor Switch and Diode blocks by the usage of the Property Inspector.

>> boostConv1

Diode

The Diode block can represent either a piecewise linear diode, an exponential diode, or a diode with a tabulated I-V curve. When the **piecewise linear** diode model is selected, if the diode forward voltage exceeds the value specified in the **Forward voltage** parameter, the diode behaves as a linear resistor with the resistance specified in the **On resistance** parameter. Otherwise, the diode behaves as a linear resistor with the small conductance specified in the **Off conductance** parameter.

Note Some Simscape Electrical components provide tools to view block characteristics. For example, right-click the Diode block and select **Electrical > Basic Characteristics**.



Modeling a Pulse Width Modulation (PWM) Generator

The switching behavior of the semiconductor switch is controlled with a PWM signal that is characterized by

- A periodic carrier waveform with switching frequency of 10 kHz.
- A modulation signal valued from 0 to 1 representing the fraction of time that the signal is 1 over the switching period.

The PWM signal is generated by comparing the modulation signal (duty cycle) to the carrier waveform (sawtooth), as shown in the figure. This generates a value of 1 (switch on) when the duty cycle is greater than the sawtooth and 0 (switch off) otherwise.

Follow the steps below to model the PWM signal using Simulink blocks:

1. Add a Constant block (**Simulink > Sources** library). Name the block **Duty Cycle**, and set the **Value** to **0.4**.
2. Add a Repeating Sequence Stair block (**Simulink > Sources** library) to generate the sawtooth waveform. Name the block **Sawtooth**, set the **Vector of output values** to **linspace(0,1,20)**, and set the **Sample Time** to **5e-6**. This will generate a periodic sawtooth waveform with a frequency of 10 kHz that is sampled 20 times faster (200 kHz):

$$F_{\text{sawtooth}} = \frac{1}{(5 \mu\text{s}/\text{point})(20 \text{ points})} = 10 \text{ kHz}$$

3. Add a Relational Operator block (**Simulink > Logic and Bit Operations** library) performing the operation: **Duty Cycle >= Sawtooth**.

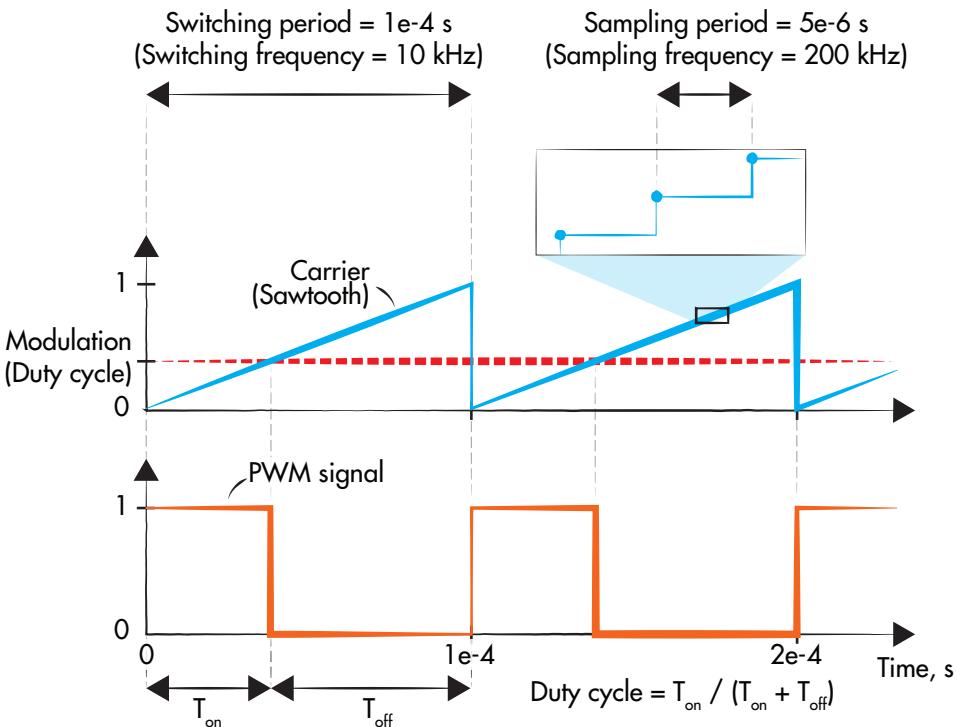
Note The sampling frequency of the PWM sets the resolution of the duty cycle. For simulation performance, set the sampling frequency to 20 to 50 times greater than the switching frequency. For high accuracy, set the sampling frequency to 100 to 200 times greater than the switching frequency.

Try

Follow the steps listed on this page to model the PWM signal generator.

This PWM is called *sawtooth trailing edge*. To implement a *sawtooth leading edge* PWM set the **Vector of output values** to **linspace(1,0,20)** in the Repeating Sequence Stair block. For a *triangular double edge* PWM, set the **Vector of output values** to **[linspace(0,1,10) linspace(1,0,10)]** – notice the 10 points during the ramp-up and the 10 points during the ramp-down.

Element	Parameter	Value	Units
PWM	Switching frequency	10	kHz
	Duty cycle	0.4	



Modeling a Pulse Width Modulation (PWM) Generator (Continued)

To connect the PWM signal to the Ideal Semiconductor Switch block in the Simscape network, do the following:

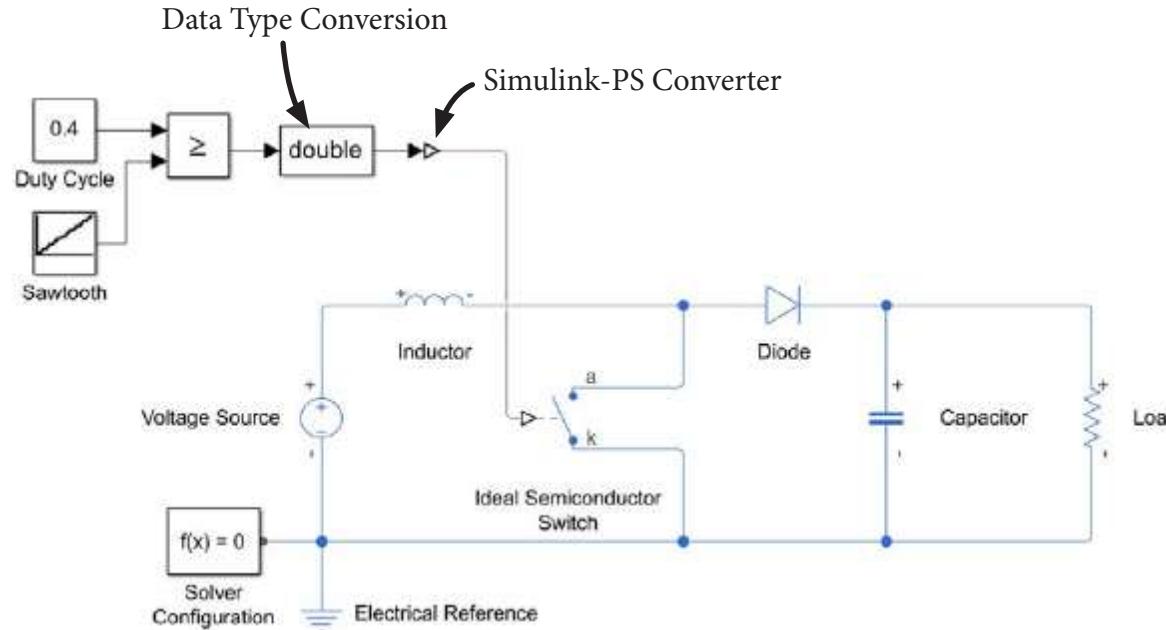
1. Add a Data Type Conversion block (**Simulink > Signal Attributes** library) and connect it to the output of the Relational Operator block. Set the **Output data type** to **double**. This converts the boolean output of the Relational Operator to a double signal so that it can be connected to the Simscape network with the consistent data type.
2. Add a Simulink-PS Converter block (**Simscape > Utilities**) to convert the double Simulink signal into a physical signal. Change the **Input signal unit** to **V**.
3. Connect the output of the Simulink-PS Converter block to the physical signal input of the Ideal Semiconductor Switch block.

Try

Use the steps on this page to connect the PWM generator to the Simscape network.

>> boostConv2

Note In other models, you may need to adjust the **Threshold voltage**, **Vth** parameter of the Ideal Semiconductor Switch block to agree with the signal range of the input physical signal. In this chapter's model, no changes are necessary because the input PWM signal ranges from 0 to 1 volt and the threshold voltage is set to 0.5 volts by default.



Measuring Physical Quantities

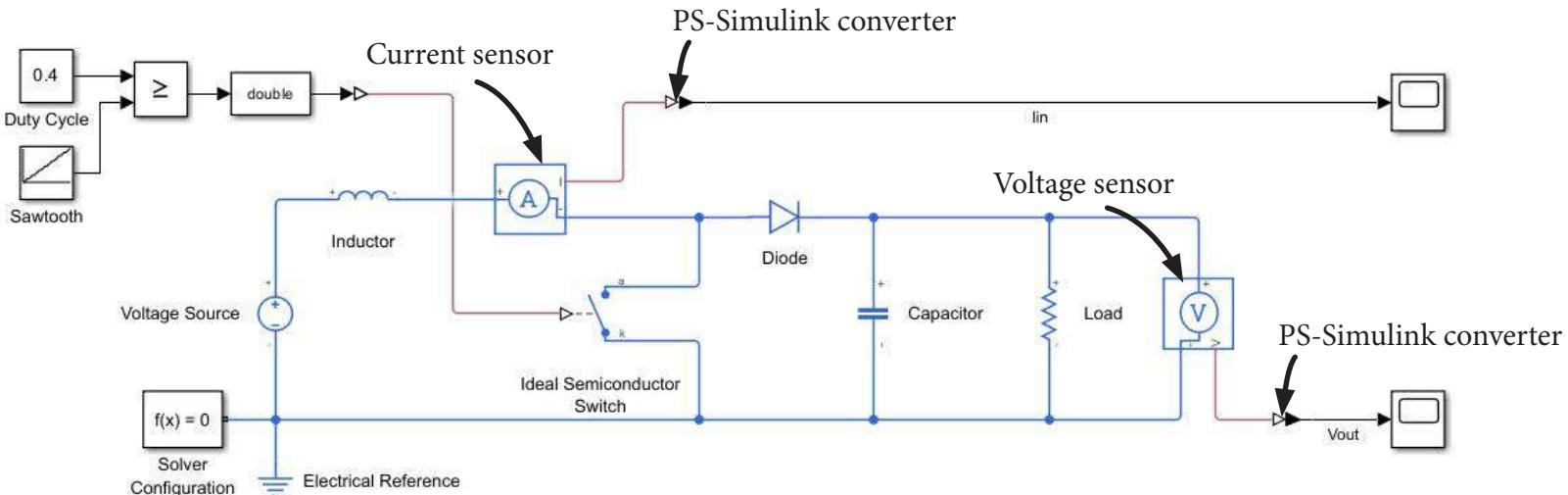
You can add sensors to measure the current and voltage at different branches and points in the model, respectively. This is useful to verify that the converter behaves as expected.

In the electrical domain, recall that

- Current is a *through* variable and should be measured through a branch in the model.
- Voltage is an *across* variable and should be measured across two points in the model.

Use the blocks below to make measurements in the model:

- Current Sensor (**Electrical > Sensors** library) – Measures the current through a branch using an ideal sensor (i.e., no series resistance).
- Voltage Sensor (**Electrical > Sensors** library) – Measures the voltage across two points using an ideal sensor (i.e., infinite parallel resistance).
- PS-Simulink Converter (**Simscape > Utilities** library) – Converts the physical signal output of a sensor to a Simulink signal.



Try

Add current and voltage sensors to measure the input current and the output voltage, respectively.

The output voltage measurement should be placed across the resistive load. The input current measurement should be placed in series with the Voltage Source.

Set the **Output signal unit** in the PS-Simulink Converter blocks to A for current and V for voltage, respectively. Then, label each resulting Simulink signal appropriately (e.g., I_{in} , V_{out}).

>> boostConv3

Note For more information about through and across variables, see **Simscape > Getting Started with Simscape > Basic Principles of Modeling Physical Networks > Variable Types** in the documentation.

Choosing a Solver

Switch-mode power electronic converters are computationally intensive to simulate due to their stiff and highly discontinuous characteristics, with fast switching events and slower reactive components' dynamics.

When simulating Simscape power electronic networks, you can choose between using a variable-step global Simulink solver and a fixed-step local Simscape solver. The table below gives details, advantages, disadvantages, and recommended uses for each solver option.

When simulating switch-mode power electronic converters you can use the **ode23t** variable-step solver. The parameter values below generally work well for most power electronic models:

- Set the **Relative tolerance** to **1e-4**.
- Set the **Absolute tolerance** to **1e-4**.
- Disable the **Auto scale absolute tolerance** parameter.

Try

Set the simulation stop time to **0.05** seconds and simulate the model. Press **Stop** if the simulation is taking too long. What solver is being used?

Open the Configuration Parameters (select **Model Settings** in the **Modeling** tab) and select the **Solver** pane. Set the **Solver** to **ode23t** and change the solver tolerances to the values given on this page.

Try to simulate a model again. Does the model simulate successfully?

```
>> boostConv4
```

You can always change solver settings to balance simulation accuracy with speed. Next, you will learn how to switch this model to use a local fixed-step solver.

Note For faster simulation but less accuracy, you can use the variable-step **odeN** solver with proper choice of the integration method. This lightweight solver is able to accurately resolve zero crossings but does not control the simulation error. The recommended integration method is **ode1be** (**Backward Euler**).

	Variable-step global Simulink solver	Fixed-step local Simscape solver
Recommended solvers	ode23t , ode15s	Backward Euler to start, can switch to others later
Simulink behavior	Simulink treats the network as a continuous system.	Simulink treats the network as a discrete system.
Advantages	<ul style="list-style-type: none"> • Ensures simulation error within specified tolerances • Resolves zero crossings accurately • Features specialized solvers for stiff systems 	<ul style="list-style-type: none"> • Provides more robust solver • Supports code generation workflows • Features specialized solvers for Simscape networks
Disadvantages	<ul style="list-style-type: none"> • Simulation may not converge for large or very stiff models • Can drastically slow down simulation to resolve rapid changes or zero crossings 	<ul style="list-style-type: none"> • No control of simulation error • No zero crossing detection • User must carefully choose a time step size
Recommended uses	<ul style="list-style-type: none"> • Smaller models and individual converters • General power electronic modeling and development 	<ul style="list-style-type: none"> • Large models and/or long simulation times • Simulations with rapid changes or many zero crossings • Code generation and hardware-in-the-loop testing

Verifying the Converter

After the model is simulated and you have scopes to visualize the data, you can test and verify the three steady-state operating conditions presented at the beginning of this chapter.

1. Boost Converter operates in Continuous Current Mode (CCM)

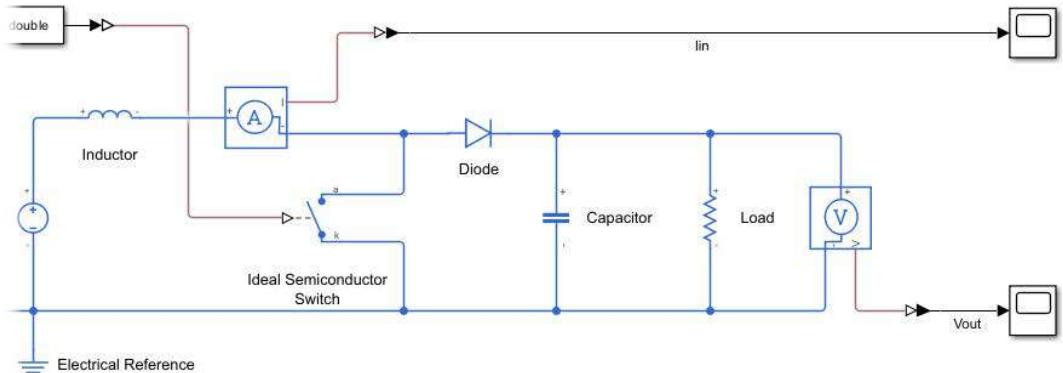
If the boost converter is operating in CCM, the input current (i.e., inductor current) should be greater than zero in steady state. You can see in the scope that once the converter reaches steady state, the input current is always positive. Thus, the boost converter is operating in continuous current mode.

2. Input Current is related to the Duty Cycle

At a duty cycle value of 0.4, the boost converter has an average input current of about 250 A at steady state. If you increase the duty cycle, the input current also increases.

3. Boosted Output Voltage is related to the Duty Cycle

At a duty cycle value of 0.4, the boost converter has an average output voltage of about 397 V. If you increase the duty cycle, the output voltage also increases.



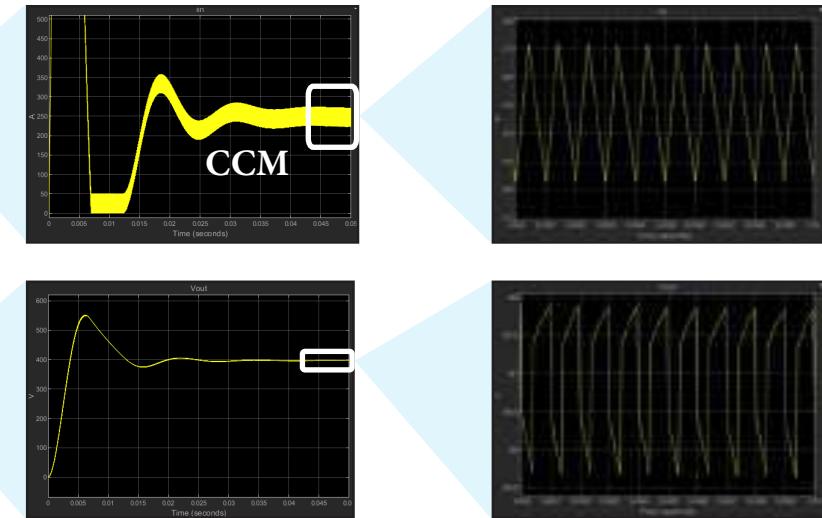
Try

Verify that the three steady-state operating conditions are satisfied by the boost converter.

From the Solver Configuration block, try enabling the **Use local solver** with **Solver type** set to **Backward Euler** and **Sample time** equal to **5e - 6**. Experiment with other sample time values and inspect the results. When finished, disable the local solver and resave the model.

Note You can enable the **Information Overlays > Units** option in the **Debug** tab to display units in the model and on the scope axes. To display time units in a scope, in the Time tab from the scope Configuration Properties you can set the **Time units** to **Seconds** or **Metric** (based on Time Span) and check the box **Show time-axis label**.

The simulation results show higher input current and lower output voltage compared to theoretical expectations. The discrepancy can be attributed to parasitic losses that exist in the model. In the next chapter, you will learn how to characterize these losses. The ultimate goal for this boost converter is to build a controller to maintain a steady output voltage of 400 V under all operating conditions.



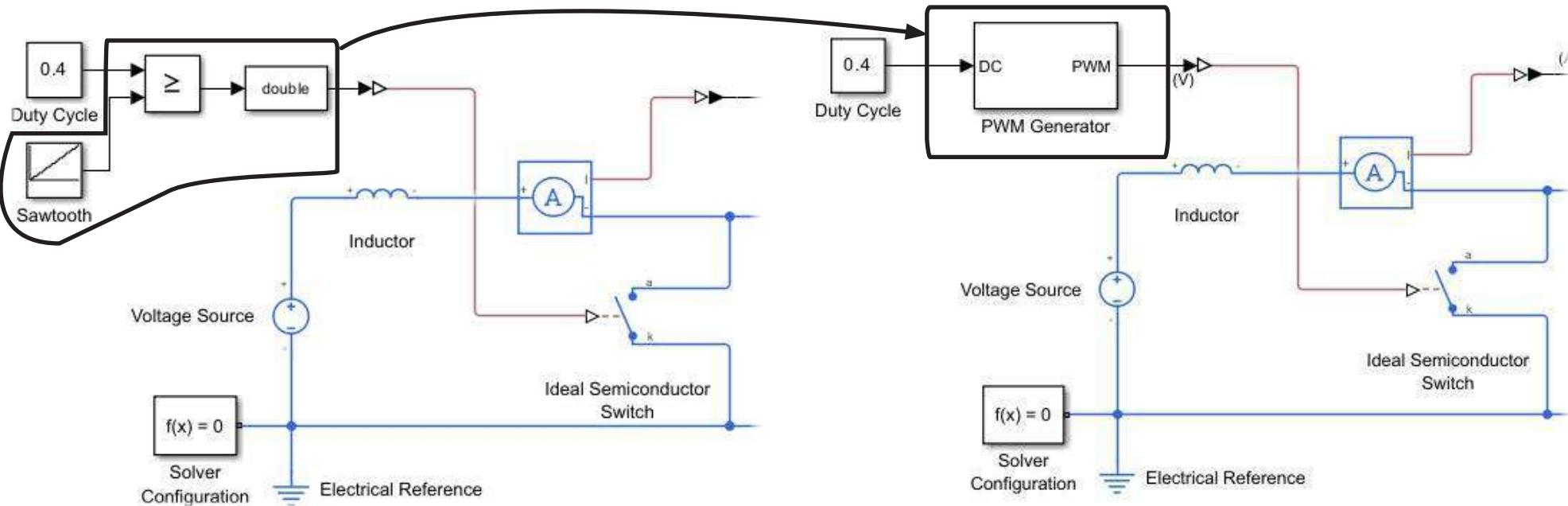
Using Prebuilt PWM Blocks

You have so far modeled a *custom-built* PWM using Simulink blocks. This approach comes with advantages in terms of flexibility for the development of ad-hoc modulation techniques. You are now going to explore *prebuilt* PWMs that are available as library blocks. The library **Electrical > Control > Pulse Width Modulation** is equipped with several PWMs serving specific power electronic topologies.

Starting from **boostConv4** follow the steps:

1. Replace the custom PWM implementation with the PWM Generator block. Set the **Timer period (s)** to **1e-4**, and specify a **Sample time** of **5e-6**.
2. Run and inspect the results.
3. Save the model as **boostConv5**.

Note The PWM Generator block implements a carrier-based algorithm. See the documentation for more details.



Try

Follow the steps listed on this page to replace the custom-built PWM implementation with the PWM Generator block.

Compare the simulation results of the two following models. Do they produce the same results?

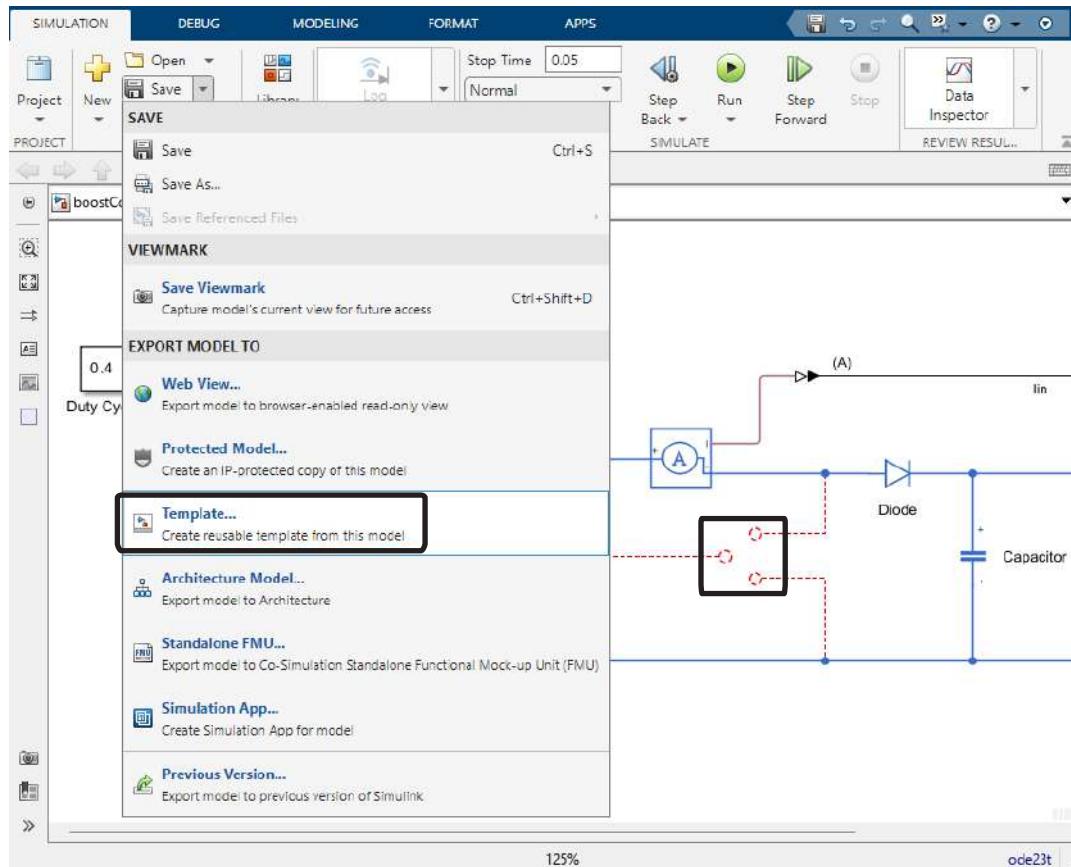
```
>> boostConv4  
>> boostConv5
```

For the PWM Generator block, try experimenting with other carrier counters. Refer to the documentation for more information on the types of carrier counters.

Exporting as Template

In the next chapter, you will replace the Ideal Semiconductor Switch with a semiconductor component that can better capture the switching behavior of real semiconductor devices. To ease this task, you can prepare your model including the recommended solver settings along with other simulation settings and export it as a template. Follow the steps starting from `boostConv5`:

1. Delete the Ideal Semiconductor Switch.
2. Click on **Templates...** under the **Save** drop-down menu in the **Simulation** tab.

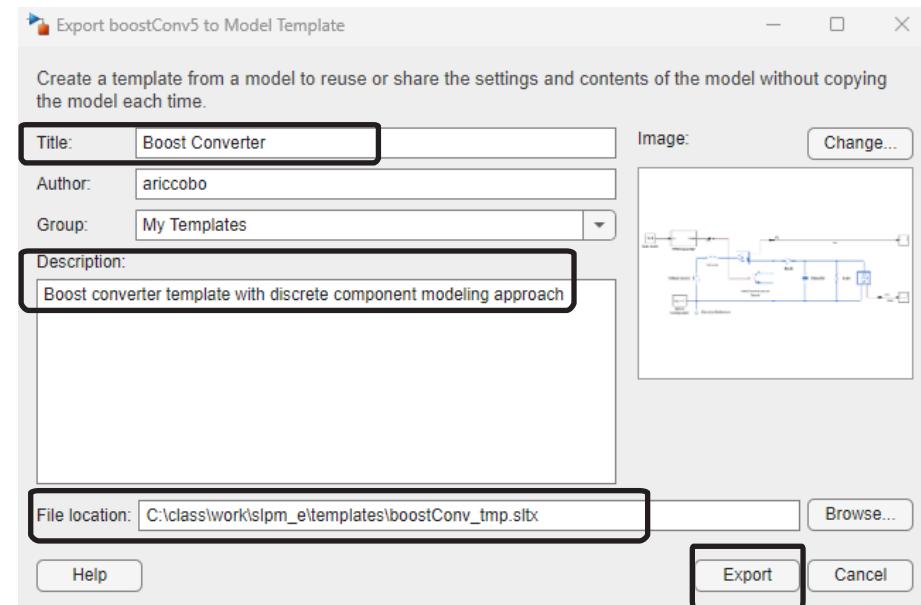


Try

Follow the steps listed on this page to export the model `boostConv5` as a template.

3. In the Export to Model Template dialog box, provide the title, description, file name, and location as depicted below.
4. Click on Export and ignore the following warning message.

Note The project folder . . . \templates is not in the project. That's why this folder is not visible from the Project View. However, since it is on the project path, your template will be available in the Simulink Startup page whenever this course project is open.



Summary

- Modeling a boost converter
- Measuring physical quantities
- Visualizing results
- Selecting a solver

Test Your Knowledge

1. (T/F): The Resistor block from the **Simscape > Electrical > Passive** library provides additional parameters that are not available in the Resistor block from the **Simscape > Foundation Library > Electrical > Electrical Elements** library.
2. The _____ block should be oriented to measure through a branch in a Simscape model. The _____ block should be oriented to measure across two points in a Simscape model.
 - A. Ideal Voltage Sensor, Ideal Current Sensor
 - B. Ideal Current Sensor, Ideal Voltage Sensor
 - C. Neither
3. (T/F): You can connect the measured voltage output from an Ideal Voltage Sensor block directly to a Scope block.
4. Which solver will slow down the simulation to resolve rapid changes or zero crossings?
 - A. Variable-step global solver
 - B. Fixed-step local solver
 - C. Both
 - D. Neither

Answers

1. T
2. B
3. F
4. A



A photograph of a man and a woman looking at a computer screen. The screen displays a 3D model of a power electronic converter, specifically a three-phase bridge rectifier. The man is pointing at the screen, and they both appear to be engaged in a discussion or review of the design.

Power Electronics Control Design with Simulink® and
Simscape™

Converter Model Fidelity

Converter Model Fidelity

Outline

- Selecting appropriate converter model fidelity
- Using prebuilt components
- Using discrete component modeling approach

Converter Model Fidelity

Chapter Learning Outcomes

The attendee will be able to

- Build models at an appropriate level of converter model fidelity.
- Understand what can be simulated with prebuilt converters and with converters made out of discrete components.

Converter Model Fidelity

Understanding Fidelity

In this course, the term fidelity refers to how closely a model, converter, or component represents its real-life physical counterpart. The terms “low” and “high” are used to communicate the degree of fidelity, as described below:

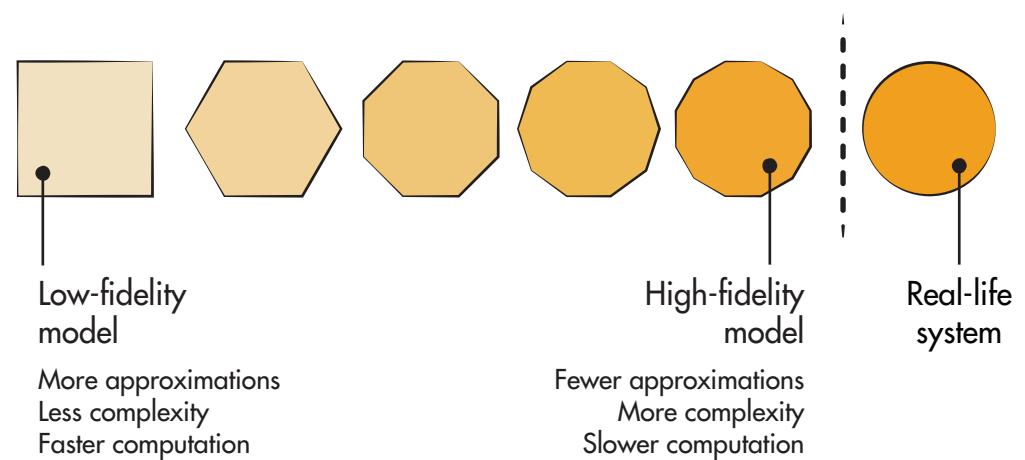
- **Low fidelity** – The system loosely represents the real-life physical system. The model may neglect system dynamics entirely or approximate them with simplified models.
- **High fidelity** – The system closely represents the real-life physical system. The model accounts for many factors that affect system dynamics.

Typically, low-fidelity models offer limited parameters but simulate quickly due to their simplicity. High-fidelity models offer many parameters that allow for precise tuning and customization but simulate slower due to the added complexity.

When modeling power electronic converters, many physical factors can influence the converter model fidelity. Examples include the following:

- Switching dynamics
- Parasitic losses
- Thermal effects
- Nonideal power sources
- Component tolerances
- Electrical noise

In this chapter you will learn how to control the converter model fidelity of a boost converter model, the purpose of the different fidelities, and how they affect the simulation results and performance.



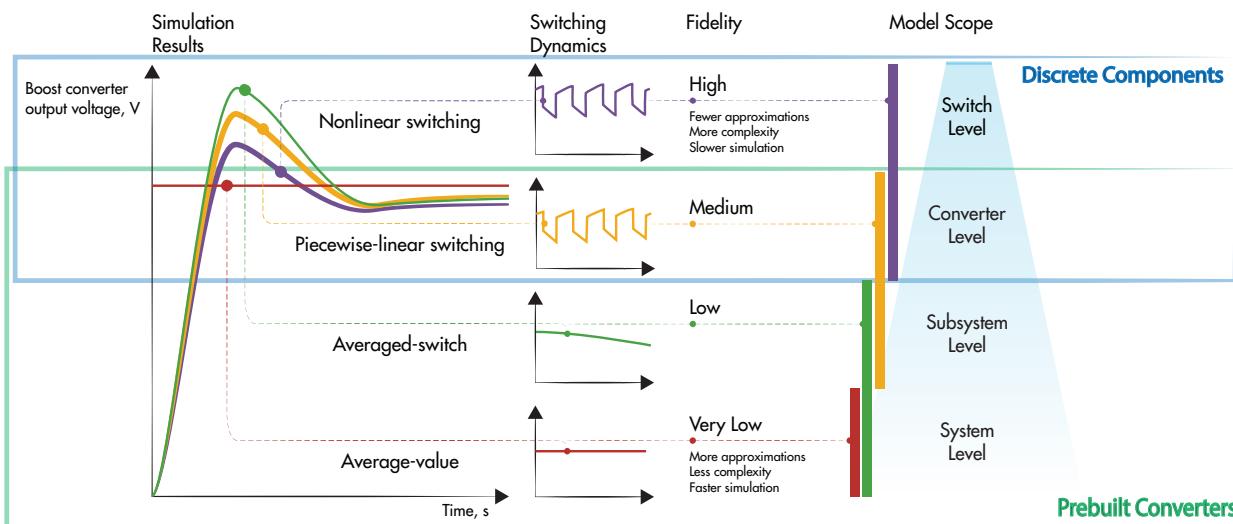
Converter Model Fidelity

Understanding Fidelity (Continued)

In this chapter, you will see how converter model fidelity influences simulation behavior. In Simscape Electrical, there are four levels of fidelity that you can use to model power electronic converters:

- Average-value** – Uses algebraic equations to calculate the converter output from the duty cycle without modeling any converter dynamics
- Averaged-switch** – Uses time-averaged PWM and a simplified circuit to model converter dynamics without modeling switching events
- Piecewise-linear switching** – Uses piecewise-linear equations to model converter dynamics with switching events
- Nonlinear switching** – Uses nonlinear equations to model converter dynamics with switching events

You can use the graphic below to help select a fidelity that matches the scope of your model. Typically lower fidelity models are used in large-size systems, while higher fidelity models are used for switch and converter level analyses and control design purposes.



Try

Explore the blocks in the **Semiconductors & Converters** and **Semiconductors & Converters > Converters** libraries.

As shown in previous chapters, power electronic converters are made up of active components (the semiconductor switching devices) and passive reactive components (inductors and capacitors) connected in a way to form a specific converter topology. There are two modeling approaches to build power electronic converters:

Discrete components

You have so far used this approach. You can rely on the **Electrical>Semiconductors&Converters** library for discrete semiconductor devices, such as transistors and diodes, and on the **Electrical > Passive** library for the passive reactive components to build your converter topologies. You can implement piecewise-linear and nonlinear switching models with this modeling approach.

Prebuilt converters

The **Electrical>Semiconductors&Converters>Converters** library provides prebuilt power electronic converters, inverters, choppers, and gate multiplexers. With this modeling approach, you can choose blocks already equipped with all the components comprising a specific converter topology. You can implement average value, averaged switch, and piecewise linear switching models with this modeling approach.

Note For both modeling approaches, you may still need to use blocks from **Electrical > Sources** and **Electrical > Passive** libraries to model the input voltage and output load, respectively.

Converter Model Fidelity

Course Example: Boost Converter with Prebuilt Converters

This chapter first uses prebuilt converter blocks in Simscape Electrical to demonstrate average-value, averaged-switch, and then piecewise-linear switching models for the boost converter. Later in the chapter, discrete components will be used to implement the fully nonlinear switching model for the same topology. You will learn how to structure a comparative analysis to assess the performance of each level of fidelity.

To get started with prebuilt converter modeling approach, you can rely on a “start model”:

1. Open the model **boostConv1_preBuilt_start** from the **Project Shortcuts** tab.
2. Inspect the model and read the annotations.

Alternatively, you can open a new model from a Simscape Electrical template.

1. Click the **Simulink** button in the MATLAB toolbar.
2. Expand the **Simscape** section of the **New** tab.
3. Click **Electrical**.

Make the following changes to the new model:

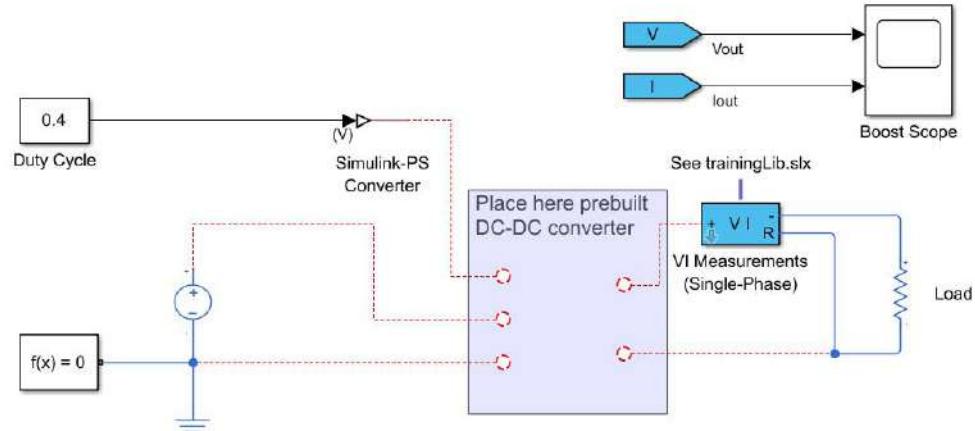
4. Delete the PS-Simulink Converter blocks, the Spectrum Analyzer block, and the annotation.
5. Add a Voltage Source block (**Electrical > Sources** library) and set the **DC Voltage** to **250 V**.
6. Add a Constant block (**Simulink > Sources** library) and set the **Constant value** to **0.4**. Name the block **Duty Cycle**.
7. Add a Resistor (**Electrical > Passive** library), name it **Load**, and set the **Resistance** to **2.67 Ohms**.
8. Open the **trainingLib.slx** library from the **Project Shortcuts** tab.

Try

Use either the “start model” **boostConv1_preBuilt_start** or the Simulink Start Page to create a new model from the **Electrical** template. Use the steps described on this page to modify the new model.

>> trainingLib

- Drag the VI Measurements (Single-Phase) block from the library into your model. This block contains a Voltage Sensor block and a Current Sensor block to help organize electrical measurements. The output measurements are transmitted using Goto blocks and can be retrieved using a From block.
- Drag the Scope block that is already set with time units. Rename the Scope block to **Boost Scope**.
- 9. Add two From blocks (**Simulink > Signal Routing** library) to the model. Change the **Goto tag** parameters to **V** and **I**, respectively.
- 10. Connect all blocks to match the model shown below.
- 11. Label the signals connected to the scope block **Vout** and **Iout**, respectively.



Converter Model Fidelity

Using Average-Value Models

Average-value models simulate very quickly but produce results that capture only the output voltage dependency from the duty cycle without any converter dynamics. This is good for simulating converters in system-level models, enabling you to observe system-level characteristics without slowing the simulation with calculations of converter dynamics.

Average-value models feature the lowest converter model fidelity. These models use algebraic relationships to calculate the converter output without modeling any converter dynamics.

To build an average-value boost converter,

1. Add an Average-Value DC-DC converter block (**Electrical > Semiconductors & Converters > Converters** library) and set the **Converter Type** parameter to **Boost converter**.

Furthermore, you need the following settings before simulating the model.

2. Enable the **Information Overlays > Units** option in the **Debug** tab to display units in the model and on the scope axes.
3. Open the Configuration Parameters and set the following parameters in the Solver pane:
 - **Solver** is **ode23t**
 - **Relative error** and **Absolute error** are **1e - 4**
 - Disable the **Auto scale absolute tolerance** parameter
 - Set the simulation **Stop time** to **0 . 05** seconds.

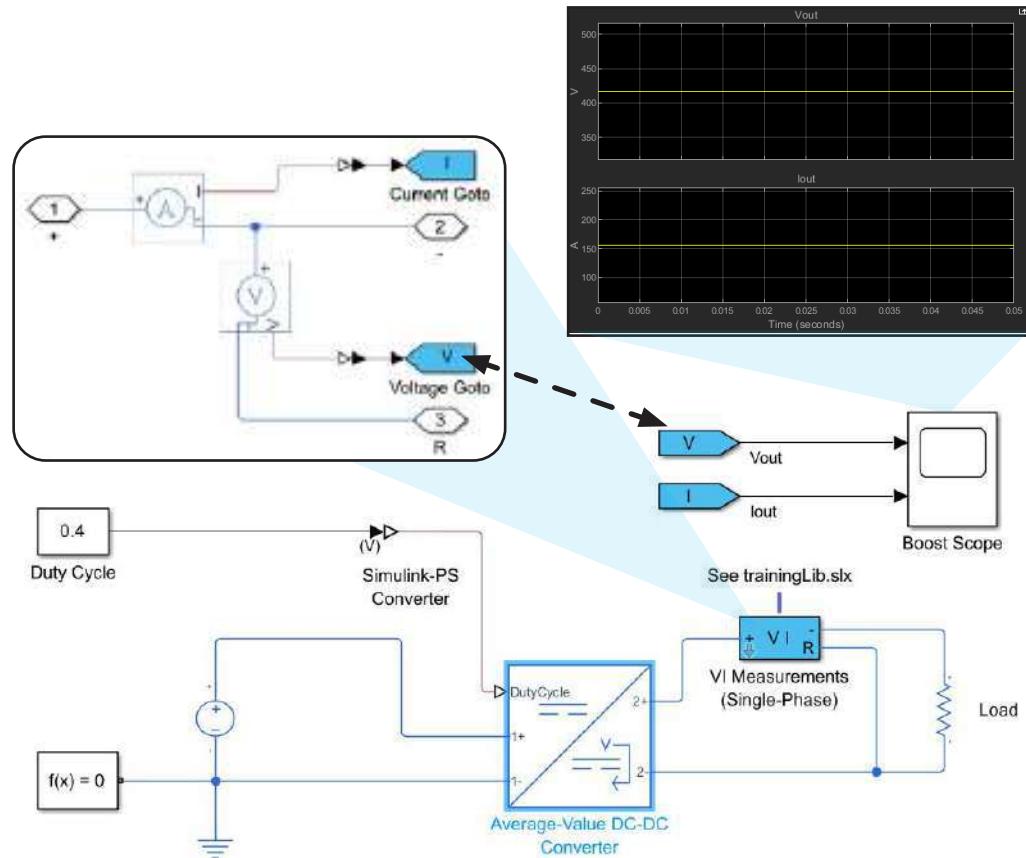
Note As a model grows, you should keep it organized to maintain readability. You can change the background color of From and Goto blocks to improve the model's readability. To do this, select a block and set the **Background** parameter in the **Format** tab.

Try

Follow the steps on this page to build an average-value boost converter model. Simulate the model and inspect the results.

When finished, save the model as **boostConv2_prebuilt_avgValue**.

```
>> trainingLib
>> boostConv2_prebuilt_avgValue
```



Converter Model Fidelity

Using Averaged-Switch Models

Averaged-switch models simulate quickly and produce more detailed results compared to average-value models. This is good for simulating system-level and subsystem-level models with more detailed dynamic responses, but still without the need to simulate the individual switching events.

Averaged-switch converter models use an averaged switch with an antiparallel diode able to be *partially opened* when the duty cycle is between 0 and 1. The converter dynamics include the inductor-capacitor circuit as well. The converter output is calculated by considering the circuit response and the time-averaged value of the PWM signal driving the switching devices.

To build an averaged-switch boost converter,

1. Replace the Average-Value DC-DC Converter with a Boost Converter block (**Electrical > Semiconductors & Converters > Converters** library). Set the following block parameters.

In the **Switching Device** section:

- **Switching device is Averaged Switch**
- **On-state resistance is 10 mOhm**

In the **Diode** section:

- **Forward voltage is 0.6 V**
- **On resistance is 2 mOhm**
- **Off conductance is 1e-8 1/0hm**

In the **LC parameters** section:

- **Inductance is 195 uH**
- **Inductor series resistance is 34.7e-3 Ohm**
- **Capacitance is 7 mF**
- **Capacitor effective series resistance is 6.67e-3 Ohm**

2. Connect all blocks to match the model shown on the right.

Note#1 To calculate the time-averaged value of the PWM signal, you can

- **Use the duty cycle** – This is the simplest approach. You can undersample the model and use modulation waveforms instead of PWM pulses.

Try

Follow the steps on this page to convert the average-value model to an averaged-switch model. Simulate the model and inspect the results.

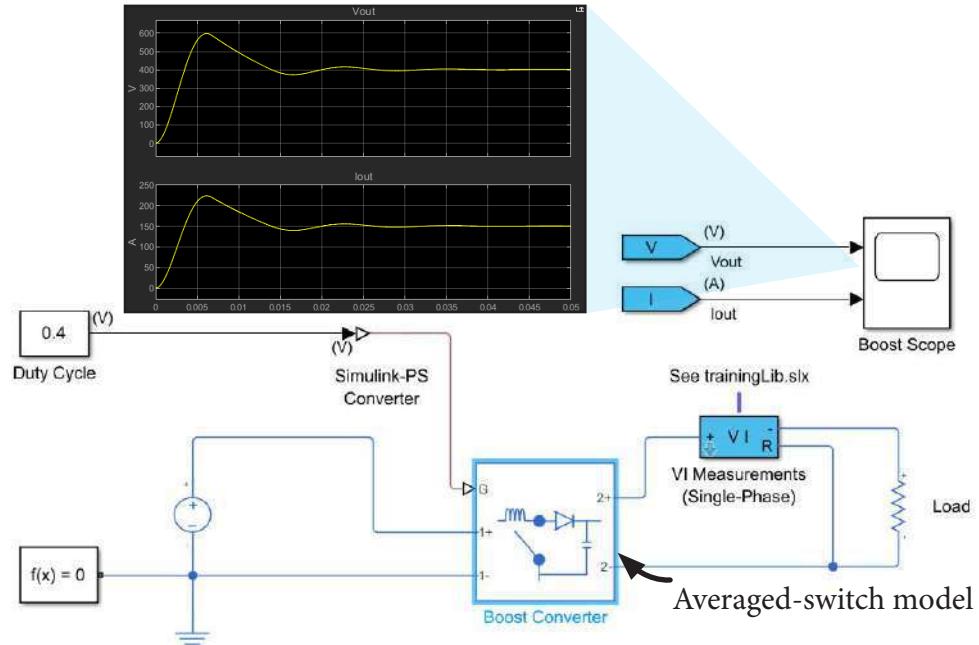
Then, save the model as **boostConv3_prebuilt_avgSwitch**.

```
>> boostConv3_prebuilt_avgSwitch
```

- **Average PWM pulses** – This approach requires additional computation but gives more control over the averaging operation.

Both approaches are useful to speed up hardware-in-the-loop simulations.

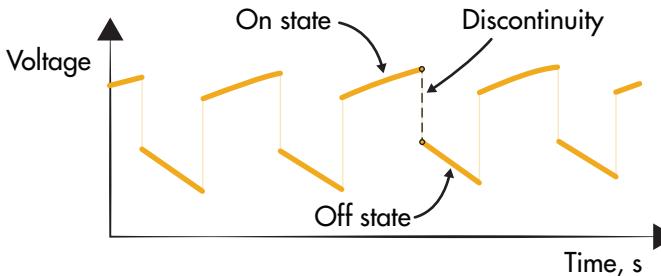
Note#2 With the default **Modeling option** to **Nonsynchronous converter**, power flow is unidirectional from the source to the load due to the blocking forward diode. If you set the **Modeling option** to **Synchronous converter**, the topology comprises two switching devices. Special care is required with the driving duty cycle. This is covered in the final chapter.



Converter Model Fidelity

Using Piecewise-Linear Switching Models

Piecewise-linear switching models generally take more time to simulate than averaged-switch models because the solver needs to take time steps small enough to resolve each switching event. Controllers are often tuned with this level of fidelity. Piecewise-linear switching models use piecewise-linear equations to mathematically describe the “on” and “off” states of the switch. A change in the switch state (i.e., switching event) creates a discontinuity in the waveforms.



To convert your averaged-switch boost converter to use a piecewise-linear switching model,

1. In the Boost Converter block, set the **Switching device** parameter to **IGBT**, set the **Off-state conductance** to **1e-8 1/0hm**, and set the **Threshold voltage** to **0.5 V**.
2. Add a PWM Generator block (**Electrical > Control > Pulse Width Modulation** library) to the model. Set the **Timer period** to **1e-4** and the **Sample time** to **5e-6**.
3. Connect the blocks as shown on the right.

The type of solver used will affect how the switching events are resolved:

- **Variable-step solver** – Zero-crossing detection will automatically ensure that the switching events are resolved in the simulation. Remember that large numbers of zero crossings can slow simulations.
- **Fixed-step solver** – Choose a fixed-step size equal to the PWM sample time. Remember that smaller time steps lead to slower simulations.

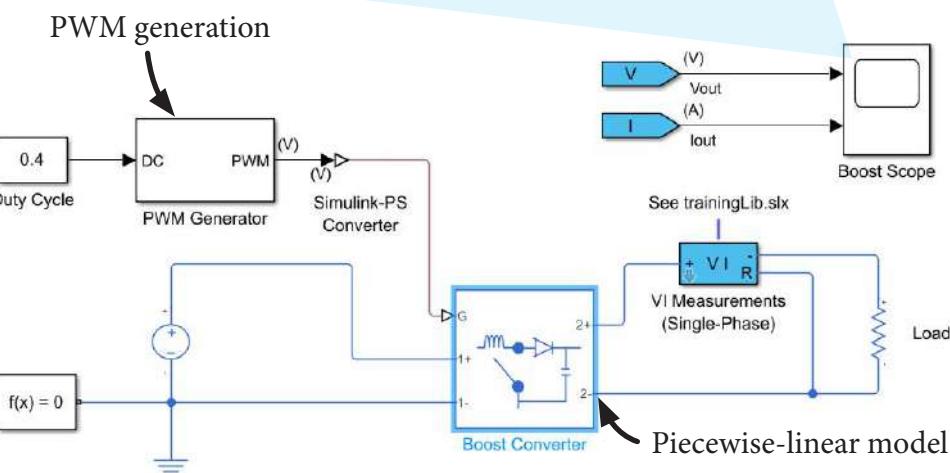
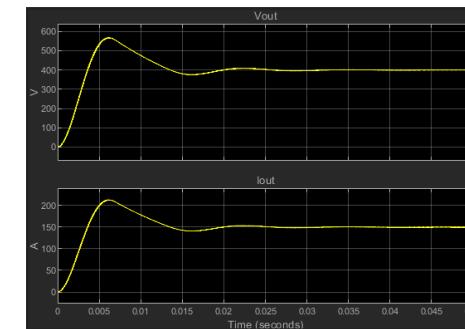
Try

Follow the steps on this page to convert the averaged-switch model into a piecewise-linear switching model. Simulate the model and inspect the results.

Then, save the model as **boostConv4_prebuilt_linSwitch**.

```
>> boostConv4_prebuilt_linSwitch
```

Note Many converter and chopper blocks in the **Electrical > Semiconductors & Converters > Converters** library provide easy options to swap from piecewise-linear switching to averaged-switch and vice versa. PWM generation is required with piecewise-linear switching models, while you can drive averaged-switch models directly with the duty cycle.



Converter Model Fidelity

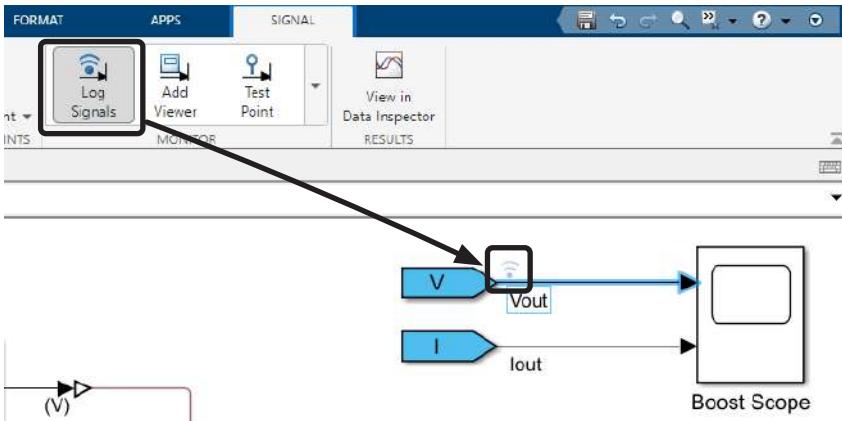
Using Piecewise-Linear Switching Models (Continued)

The blocks labeled “Ideal”, “Ideal, Switching”, and “Piecewise Linear” in the Electrical > Semiconductors & Converters library all use piecewise-linear switching models. In the previous chapter, you in practice implemented a piecewise-linear switching model because you used the IGBT (Ideal, Switching) block and Diode block (with **Diode model** set to **Piecewise linear**).

You can perform a consistency check for the models with discrete components and with prebuilt converter that implement the same level of fidelity.

1. Open the following models: **boostConv4_prebuilt_linSwitch** and **boostConv1_semiconductor**.
2. Click to select the signal labeled **Vout** and enable the **Log Signals** in the **Signal** (or **Simulation**) tab. Do this for both models.
3. Run the models.
4. Open the Simulation Data Inspector and compare the two simulation runs.

With no implementation or parametrization errors, the two simulation runs will produce identical results.



Try

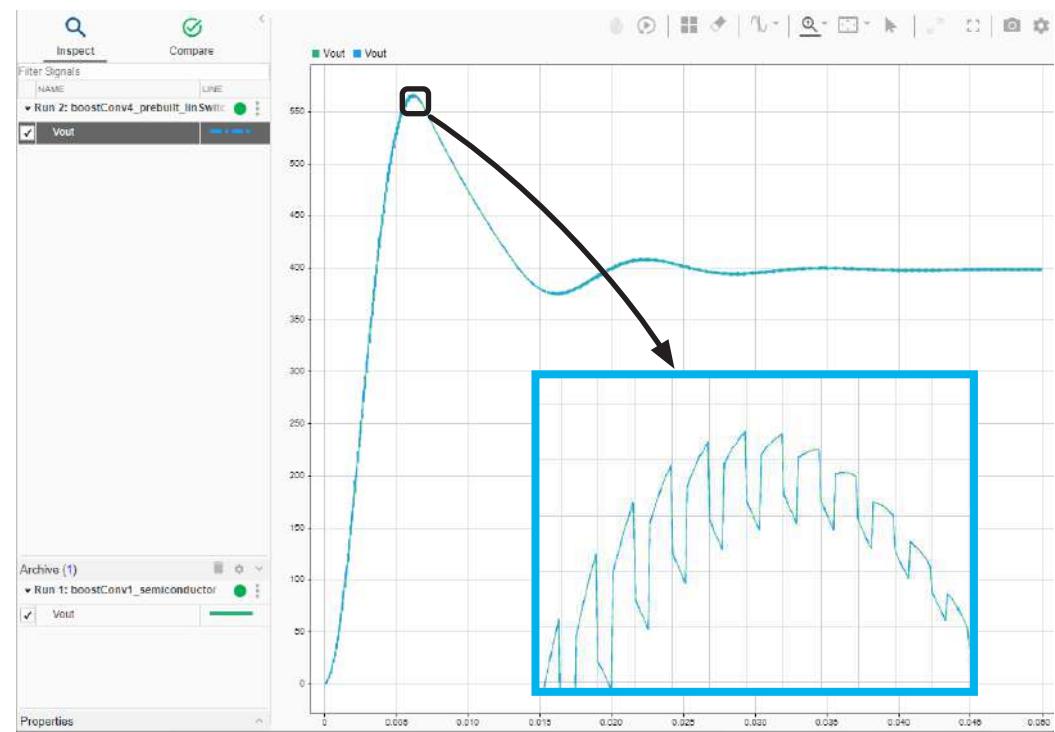
Follow the steps on this page to perform a consistency check for the two following models implementing the same level of fidelity.

```
>> boostConv4_prebuilt_linSwitch  
>> boostConv1_semiconductor
```

Note With prebuilt converters, you **cannot**

- Model switching losses
- Show the thermal port
- Use pre-parametrization workflows.

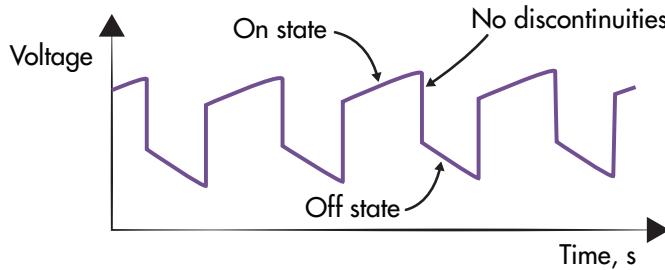
Piecewise-linear switching models are widely used for linearization and control applications as well as for simulating converter-level and subsystem-level models with discontinuous dynamic responses due to switching.



Converter Model Fidelity

Using Nonlinear Switching Models

Nonlinear switching models use nonlinear equations to mathematically describe the “on” and “off” behavior of the switch. Although there may be sharp transients, switching waveforms do not have any discontinuities.



In the **Electrical > Semiconductors & Converters** library, the blocks labeled “N-Channel”, “P-Channel”, “NPN”, and “PNP” all use nonlinear switching models. To build a converter with a nonlinear switching model, you will need to use these semiconductor devices. Notice that the Diode block can be parametrized for nonlinear switching behavior as well by setting the **Diode model** to **Exponential** or **Tabulated I-V curve**.

Simscape Electrical blocks that use nonlinear switching models provide many parameters for customization. This enables you to fine-tune the blocks’ static and dynamic characteristics. You can refer to a device’s manufacturer datasheet to obtain all relevant parameter values. If any parameter values are unknown, you can do one of the following:

- Manually tune unknown parameter values.
- Write an optimization routine to find optimal parameter values that generate the expected response characteristics given in the datasheet.
- Use the Response Optimization tool (included with Simulink Design Optimization™) to find optimal parameter values.

Starting from the **boostConv5_discComp_nonlinSwitch_start** model, you will use an N-Channel IGBT block to incorporate a nonlinear

Try

Open the **boostConv5_discComp_nonlinSwitch_start** model. Follow the steps on this page to add an N-Channel IGBT block to the model. After parameterizing the block, view its basic characteristics.

switching model in your boost converter model. To do this,

1. Add an N-Channel IGBT block (**Electrical > Semiconductors & Converters** library) to your model. Connect the block as shown below.
2. Set the block parameters to the values shown in the table below. Or, you can use the library block My N-Channel IGBT from **trainingLib**.

To check a nonlinear switching device’s parameterization, right-click the block and select **Electrical > Basic characteristics**. This will plot the block’s basic characteristics using the specified parameter values.

N-Channel IGBT Parameter	Value	Units
Zero gate voltage collector current, I_{ces}	1	mA
Voltage at which I_{ces} is defined	6500	V
Gate-emitter threshold voltage, $V_{ge(th)}$	6.5	V
Collector-emitter saturation voltage, $V_{ce(sat)}$	3.1	V
Collector current at which $V_{ce(sat)}$ is defined	1000	A
Gate-emitter voltage at which $V_{ce(sat)}$ is defined	15	V
Input capacitance, C_{ies}	101	nF
Reverse transfer capacitance, C_{res}	7.0	nF
Output capacitance, C_{oes}	17	nF
Emission coefficient, N	2.8	
Forward Early voltage, V_{AF}	170	V
Collector resistance, R_C	1e-4	Ohm
Emitter resistance, R_E	1e-4	Ohm
Internal gate resistance, R_G	1e-3	Ohm
Forward current transfer ratio, B_F	270	

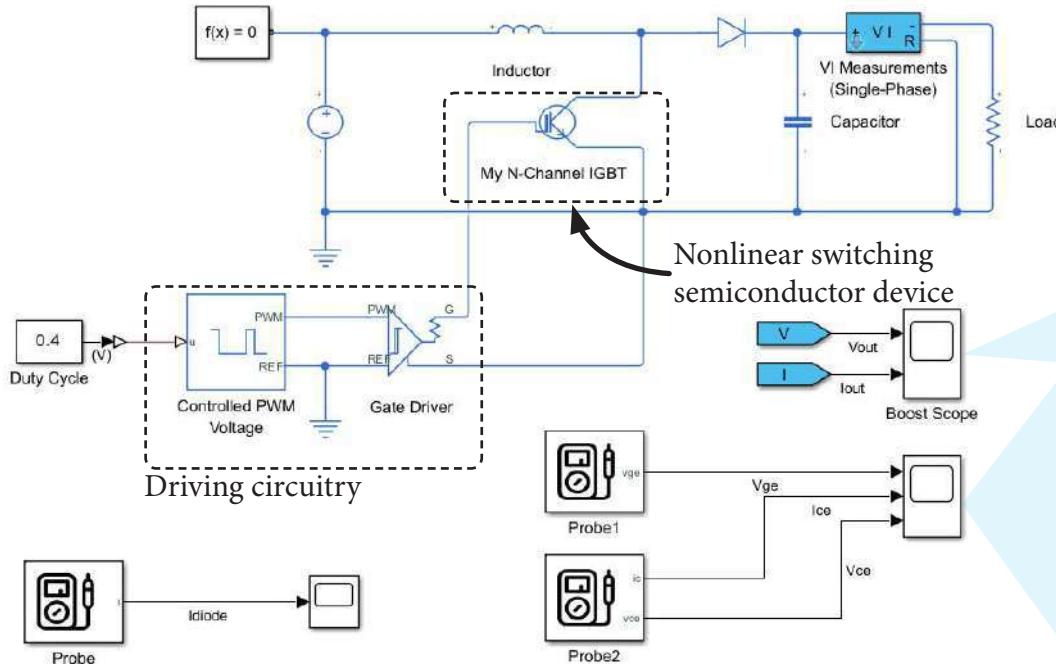
Converter Model Fidelity

Using Nonlinear Switching Models (Continued)

All the three-terminal blocks labeled “N-Channel”, “P-Channel”, “NPN”, and “PNP” have the device’s base or gate terminal exposed as physical connection port. To ensure fast switching of such blocks in presence of nonzero gate (or base) capacitances, you will need to model a gate driver with a specific output impedance able to supply the device gate (or base) with the appropriate voltage levels.

After adding the N-Channel IGBT block to the `boostConv5_discComp_nonlinSwitch_start` model, you need to model the gate driving circuitry. To do that,

1. Add a Controlled PWM Voltage block (**Electrical > Integrated Circuits** library). Double-click the block and set the **Modeling option** to **PS input**. Then, set the **PWM frequency** parameter to 10 kHz.



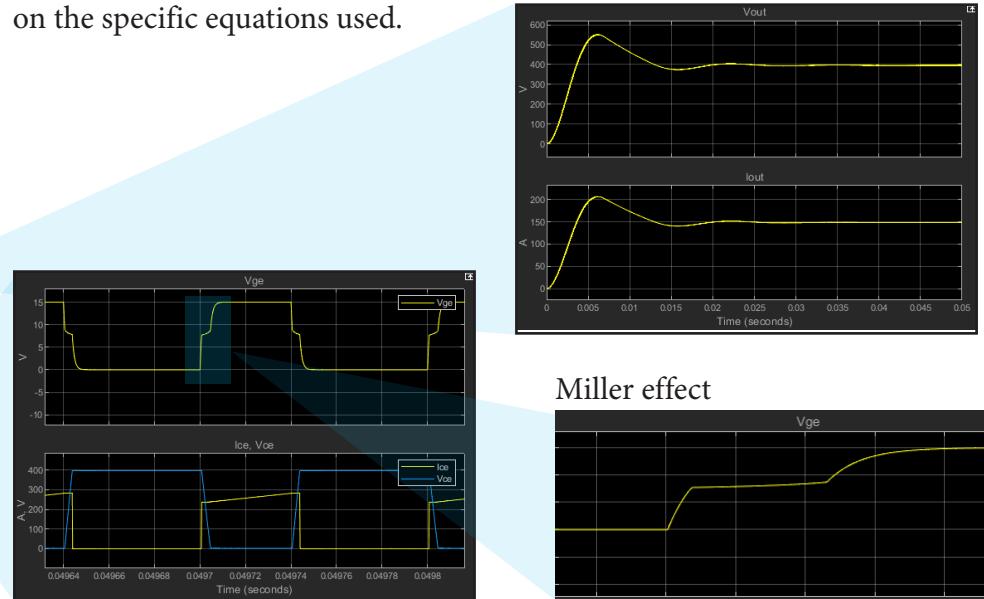
Try

Follow the steps on this page to model a gate driving circuitry, run the simulation, and inspect the results. How do the switching transients look?

>> boostConv6_discComp_nonlinSwitch

2. Add a Gate Driver block (**Electrical > Semiconductors & Converters** library). Double-click the block and set the **Input port** to **Electrical**. In the **Dynamics** section of the block’s dialog parameters, set both the **On-state gate drive resistance** and the **Off-state gate drive resistance** parameters to 9.4 Ohm.
3. Connect the blocks as shown below.
4. Run the simulation and inspect V_{out} , V_{ge} , I_{ce} , and V_{ce} waveforms. Notice the *Miller effect* due to the IGBT nonlinear capacitances.

Nonlinear switching models generally require the most time to simulate because the solver needs to take time steps small enough to resolve each switching transient and compute the complicated nonlinear equations at each step. You can refer to a block’s documentation for detailed information on the specific equations used.



Converter Model Fidelity

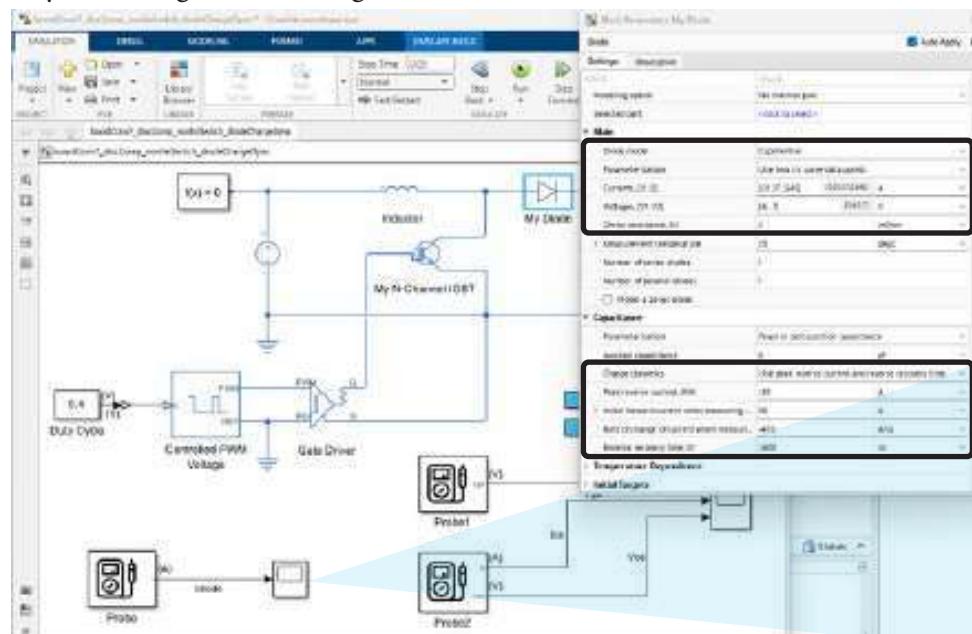
Adding Diode Nonlinear Switching Dynamics

In the previous chapter, you learned that the switching losses of a diode are dominated by the *reverse recovery losses*. You can model these losses for a Diode block by implementing nonlinear switching model. To do this,

1. Double-click the Diode block and parameterize it as shown in the figure below. Or, you can use the library block My Diode from **trainingLib**.
2. Run the simulation.
3. Inspect a switching period of **Idiode**, **Vout**, **Vge**, **Ice**, and **Vce** waveforms.

Reverse recovery transients occur when the diode turns off with negative current spikes that will exponentially extinguish to zero.

As the **boostConv7_discComp_nonlinSwitch_diodeChargeDyns** model is already set for Simscape logging, you can calculate the efficiency and losses with **ee_getEfficiency()** function. Compare with the results in the previous chapter:



Try

Follow the steps on this page to add nonlinear switching dynamics to the Diode block. After parameterizing it, view its basic characteristics.

Simulate the model to appreciate the diode reverse recovery transients.

Calculate the efficiency and losses with **ee_getEfficiency()** function. Compare with the results in the previous chapter.

```
>> boostConv7_discComp_nonlinSwitch_diodeChargeDyns
```

```
>> [eff,losstab] = ee_getEfficiency(...  
    'Load',out.simlog,0.0495,0.05)
```

Important Switching and conduction losses are not calculated separately with nonlinear switching components as they are with ideal switching blocks.

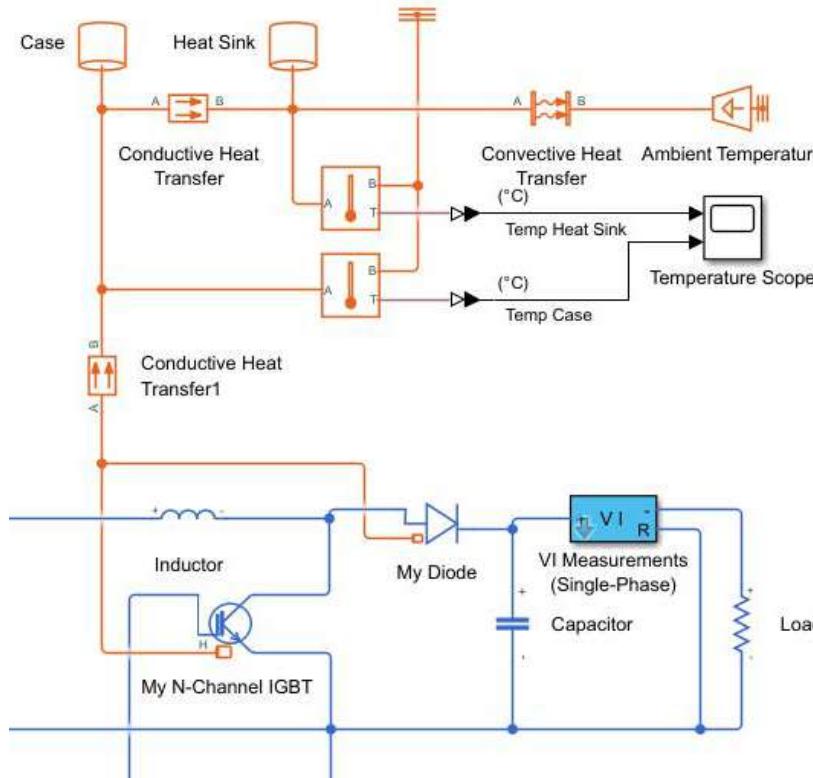


Converter Model Fidelity

Thermal Effects with Nonlinear Switching Models

You can model thermal effects for nonlinear switching semiconductor components by showing the thermal port and choosing an appropriate thermal network. This is conceptually identical to what was covered at the end of the previous chapter.

The `boostConv8_discComp_nonlinSwitch_thermal` model presents an example of a naturally cooled heatsink for the semiconductor switching devices of the boost converter. Both the N-Channel IGBT and Diode blocks have the thermal port exposed and the **Thermal network** parameter set to **External**. The thermal network below is entirely modeled by using the `boostConv8_discComp_nonlinSwitch_thermal`



Try

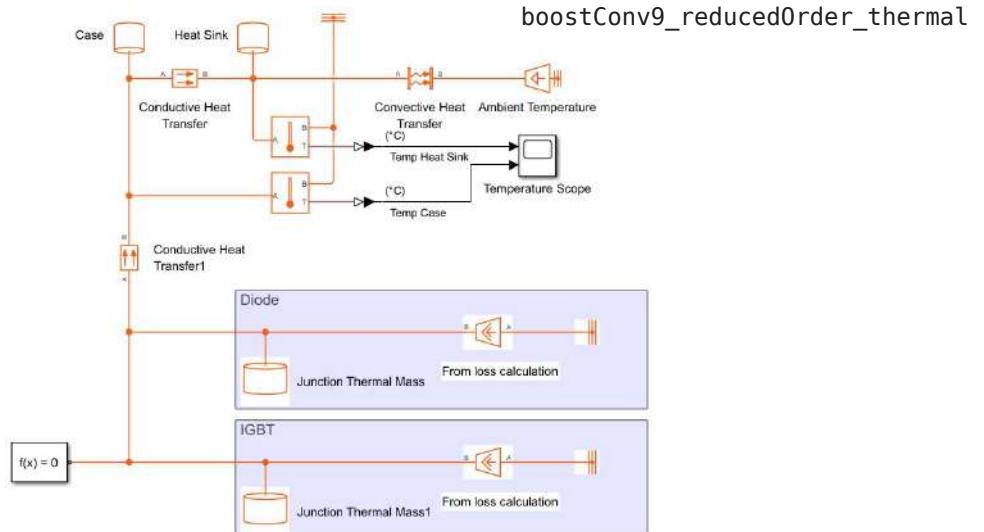
Open the `boostConv8_discComp_nonlinSwitch_thermal` model. View the N-Channel IGBT and Diode blocks' settings for modeling thermal effects. Simulate the model and inspect the results.

Then, open the `boostConv9_reducedOrder_thermal` model. Simulate it to find the steady-state temperatures. Are the measured temperatures within the safe operating range for semiconductor devices?

Simscape > Foundation Library > Thermal library. Within 50 ms of simulation time, the electrical network reaches its steady-state operating condition, but the thermal network would require much longer.

At this point, you can run the `boostConv9_reducedOrder_thermal` that counts for the measured losses for the IGBT and diode and uses them in the Heat Flow Rate Source blocks. After simulating for an hour, you should measure the temperatures of the heatsink and the case. If they are not within the safe range, you should iterate your thermal design.

For more information about thermal effects, search for **Simulating Thermal Effects in Semiconductors** in the documentation.



Converter Model Fidelity

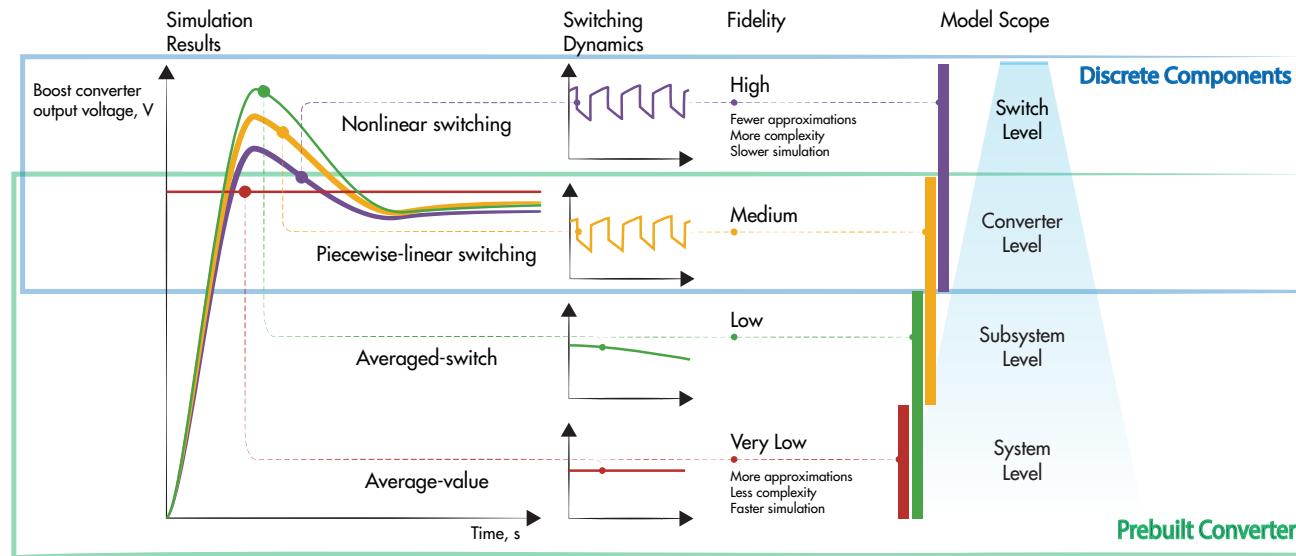
Summary

- Selecting appropriate converter model fidelity
- Using prebuilt components
- Using discrete component modeling approach

Try

To reproduce the plot below, open the following script from the project shortcuts and run it.

```
>> edit compare_boostConv
```



	Average-value	Averaged-switch	Piecewise-linear Switching	Nonlinear Switching
Discrete components?	✗	✗	✓	✓
Prebuilt converters?	✓	✓	✓	✗
Switching signal required?	✗	✗	✓	✓
Converter model fidelity	Very low	Low	Medium	High
Training model	boostConv2_prebuilt_avgValue	boostConv3_prebuilt_avgSwitch	boostConv4_prebuilt_linSwitch	boostConv6_discComp_nonlinSwitch

Converter Model Fidelity

Converter Model Fidelity

Test Your Knowledge

1. (T/F): Piecewise-linear switching models do not require a switching signal to be generated.
2. (Select all that apply) Which of the following converter levels of fidelity are available in the Boost Converter block from the **Electrical > Semiconductors & Converters > Converters** library?
 - A. Average-value
 - B. Averaged-switch
 - C. Piecewise-Linear Switching
 - D. Nonlinear Switching
3. (T/F): Piecewise-linear switching models can be implemented with both prebuilt converter blocks and discrete components.
4. (T/F): When you calculate the losses with `ee_getEfficiency()` for a nonlinear switching model, conduction and switching losses are calculated separately.

Answers

1. F
2. B, C
3. T
4. F



Power Electronics Control Design with Simulink® and
Simscape™

Digital Control Design

Outline

- Implementing a closed-loop voltage discrete PID control
- Setting model and control sample times
- Understanding linearization of power electronic converters
- Frequency Response Estimation
- Tuning the controller
- Test and verification

Chapter Learning Outcomes

The attendee will be able to

- Implement a discrete PID controller.
- Linearize a power electronics design.
- Tune a linearized power electronics design.
- Test and verify the closed-loop performance.

Background and Motivation

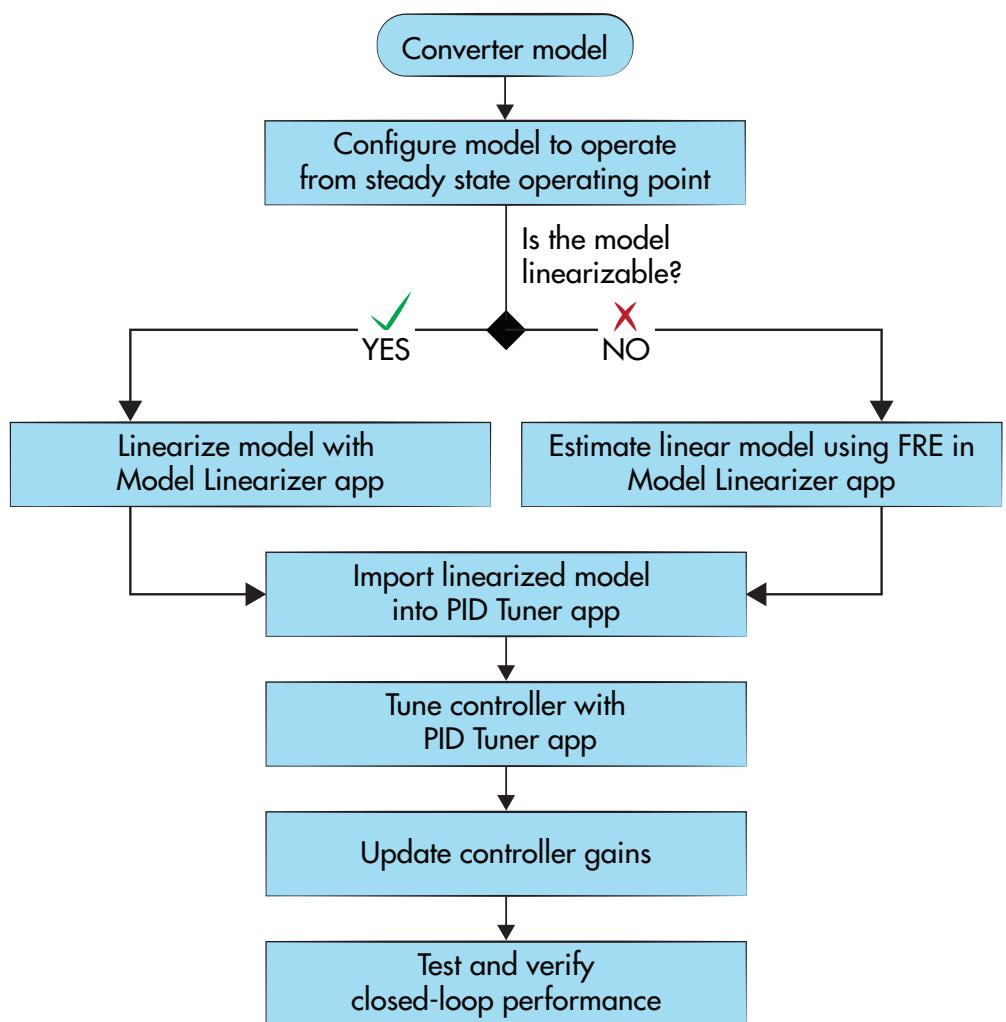
The boost converter models in previous chapters operated in an *open-loop* configuration, where the duty cycle input was independent of the boost converter output. In this chapter, you will implement *closed-loop* control using a discrete proportional-integral-derivative (PID) controller to maintain the boost converter at a desired output voltage. This is known as *voltage-mode control*. The full workflow covered in this chapter is shown on the right.

You will use the PID Tuner app to graphically tune the gains of the PID controller and produce an acceptable response. This tuning process is based on the linear control theory, which requires a linear time-invariant (LTI) representation of the plant model. If the plant model is nonlinear, it must first be linearized in order to use this tuning process.

Power electronic switching converter models (that is, piecewise-linear switching or nonlinear switching models) are generally not linearizable because of the discontinuous or highly nonlinear behavior associated with each switching event/transient. To overcome this difficulty, Frequency Response Estimation (FRE) can be used to find an estimated LTI representation of the open-loop switching converter for control design.

At the end of this chapter, the control system will be tested to verify that it regulates the boost converter output voltage despite changes to the load. In the final chapter, this model will be used to power the HEV motor drive system.

Note This chapter presents an *offline* approach to FRE and controller tuning, where the system is estimated and tuned outside of simulation. However, these steps can also be applied *online*, or during simulation. For more information, see **Simulink Control Design > Control System Design and Tuning > PID Controller Tuning > Tune PID Controller in Real Time Using Closed-Loop PID Autotuner Block** in the documentation.



Digital Control Design

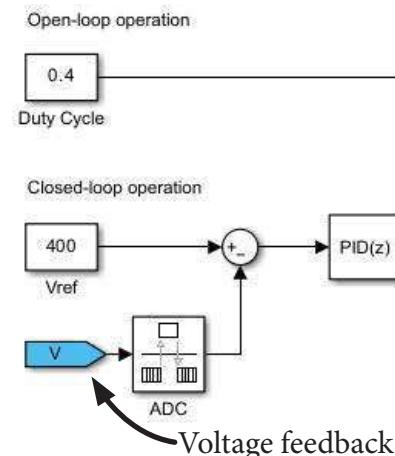
Modeling a Voltage-Mode Controller

Open the `boostConv1_control_start` model. In this model,

- The boost converter operates in open-loop mode with a 40% duty cycle.
- The boost converter block is configured to use a piecewise-linear switching IGBT model.
- The PWM Generator block timer period is set to $1e-4$ s (10 kHz switching frequency) and its sample time is set to $1e-6$ s (sampling frequency 100 times larger than the switching frequency to resolve the duty cycle with 1% increments).

Before going through the control design workflow, you need to implement the closed-loop voltage-mode controller. To do that,

- Add a Sum block (**Simulink > Math Operation** library) and set the **List of signs** to $|+-|$.
- Add a Constant block (**Simulink > Sources** library), name it `Vref`. Set the **Constant value** to **400**, which is the desired output voltage of the boost converter in volts. Set the **Sample time** to **$1e-4$** .
- Add a Rate Transition block (**Simulink > Signal Attributes**), name it `ADC`. Set the **Sample time** to **$1e-4$** . This block mimics an analog-to-digital converter (ADC) sampling the sensed output voltage.
- Add a **From** block (**Simulink > Signal Routing** library), set the **Goto tag** to `V` (this is the voltage measured by the VI Measurements block).
- Add a Discrete PID Controller block (**Simulink > Discrete** library).
- Add a Manual Switch block (**Simulink > Signal Routing** library) to quickly switch between open-loop and closed-loop configurations.



Try

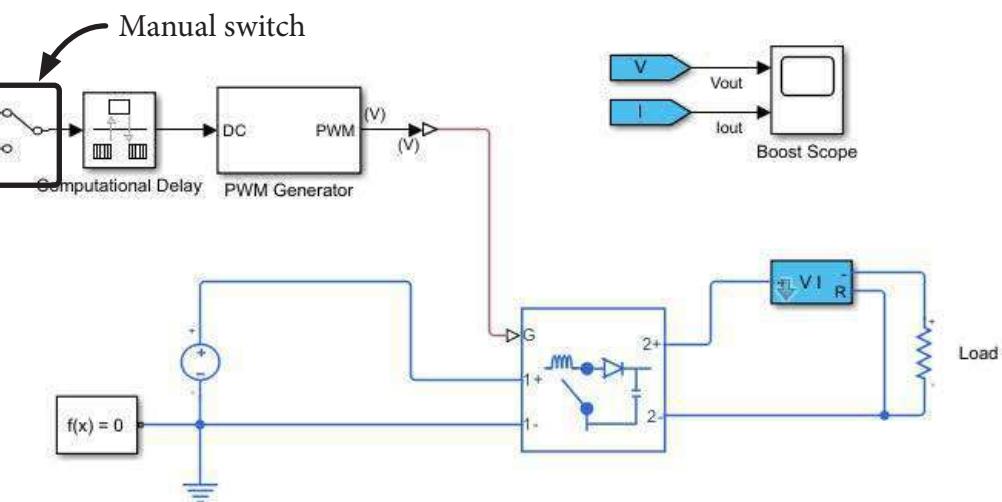
Open the `boostConv1_control_start` model. Follow the steps on this page and save this model as `boostConv2_control`.

Then, simulate the model and verify open-loop performance.

>> boostConv2_control

- Add another Rate Transition block, name it **Computational Delay**. This is necessary to accommodate the different sampling rates between the two blocks and to mimic the digital delay of real-world controllers.
- Connect all the blocks as shown in the figure below.

Change the position of the Manual Switch block to the closed-loop configuration and simulate the model. You'll notice that the controller does not maintain the desired output voltage around 400 volts. This is because the controller gains still must be tuned. Moreover, notice that the first Rate Transition block acts as a Zero-Order Hold (ZOH), while the second one as a Unit Delay ($1/z$). Then, change the position of the Manual Switch block back to the open-loop configuration.



Digital Control Design

Setting Model and Control Sample Times

Your digitally controlled boost converter model comprises a few different sample times for the model and control. You can select **Information Overlays > Colors** in the **Debug** tab to view the sample times used in the model.

For digitally controlled power electronic converters, the following recommended settings for model and control sample times apply.

Switching period

A power converter has a switching period $T_{sw} = 1/f_{sw}$, where f_{sw} is the switching frequency.

PWM sample time

Choose an adequate PWM sample time T_s^{PWM} to ensure the desired duty cycle resolution according to the following relationship:

$$T_s^{PWM} = T_{sw} / N$$

where N is an integer number that should be chosen between 20 and 50 for fast simulation with still acceptable accuracy, and between 100 and 200 for applications where you need small duty cycle resolution.

Control sample time

The control sample time T_s^{CTRL} is the calculation rate of the software algorithm implemented in the controller. It should be chosen to be something between the PWM sample time and the switching period, that is $T_s^{PWM} \leq T_s^{CTRL} \leq T_{sw}$. Typical choices are $T_s^{CTRL} = T_{sw}$ for single-update uniformly sampled PWMs, and $T_s^{CTRL} = T_{sw} / 2$ for double-update

Try

Analyze the model and control sample times based on the content of this page.

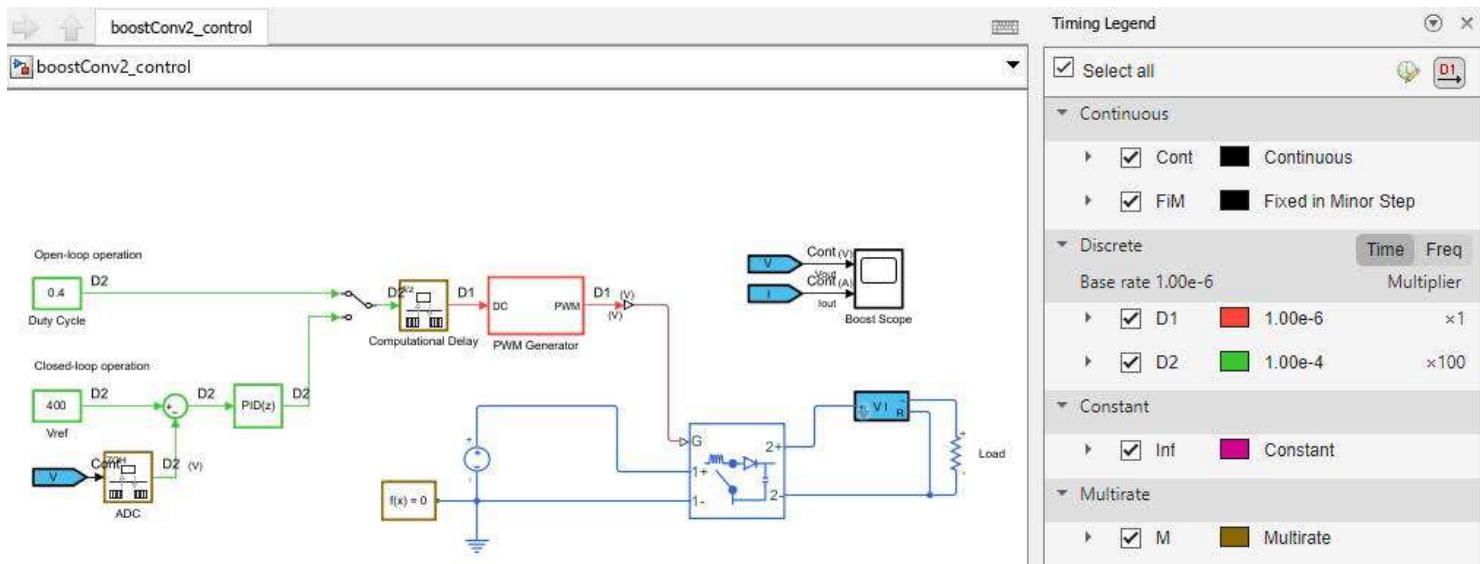
```
>> boostConv2_control
```

uniformly sampled PWMs.

Local Solver - the physical network sample time

If you enable the Local Solver, all the physical network states, which are continuous by default, become discrete. You can set the Local Solver sample time T_s as follows:

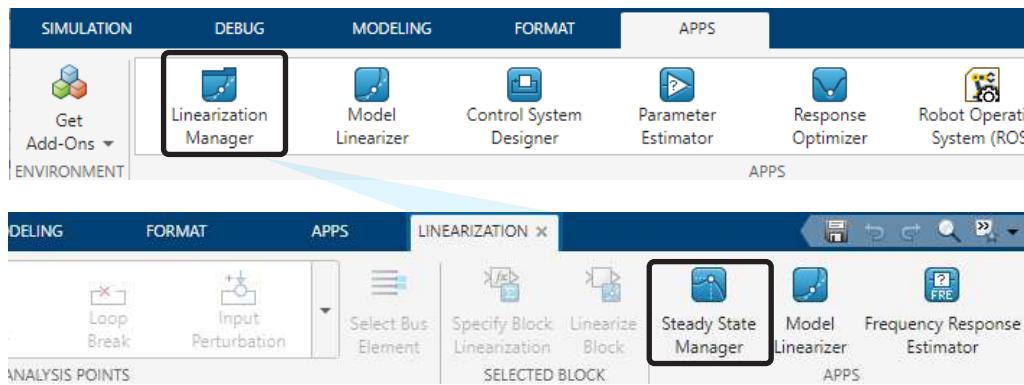
- $T_s = T_s^{PWM}$ for switching models (piecewise-linear and nonlinear)
- $T_s = T_{sw} / M$, where M is a positive integer, for averaged-switching models.



Digital Control Design

Finding a Steady-State Operating Point

After you have implemented the closed-loop controller, you can go ahead and apply the control design workflow. To get started, click on the **Linearization Manager** from the **Apps** tab. This action opens up the **Linearization** tab fully equipped with the tools (part of Simulink Control Design™) in support of the control design workflow.



The first step in the control design workflow is obtaining a steady-state *operating point* for the open-loop system. This operating point will contain information about the system in steady-state operation. It will be used later to generate an estimated LTI object for the open-loop system. To capture a steady-state operating point,

1. Open the Steady State Manager app by clicking **Steady State Manager** in the **Linearization** tab.
2. Click **Snapshots**.
3. In the **Create Snapshot Operating Point** dialog, enter a single time value or an array of time values. Then, click . The tool will simulate the model and capture the input and state information from the model at the specified times. When finished, the details of the operating point will automatically be displayed in a new tab.
4. Click **Set Initial Conditions** in the **Operating Point** tab to
 - Write the operating point to the base workspace (named **op1** by default).

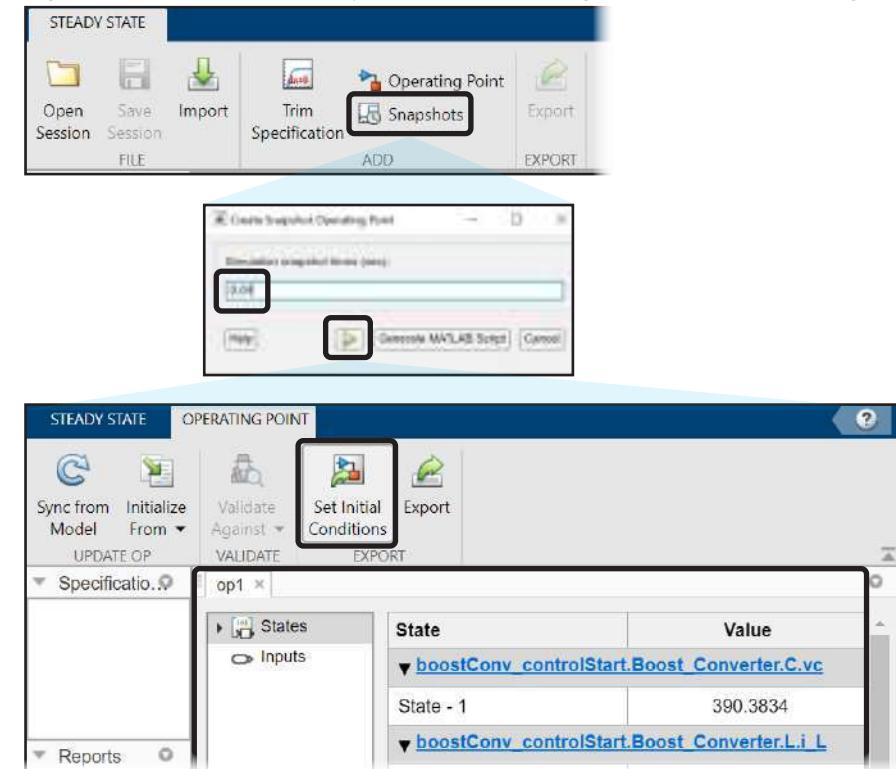
Try

Simulate the model in the open-loop configuration and open the Boost Scope. The voltage output should be in steady state at 0.04 seconds. Follow the steps on this page to obtain a steady-state operating point by snapshotting the model at 0.04 seconds.

Then, initialize the model at the operating point. Verify that the **op1** variable was created in the base workspace. Simulate the model and verify that the model now starts in steady-state operation.

- Update the model's **Input** and **Initial state** options in the **Data Import/Export** pane of the Configuration Parameters dialog.

Note You should save the operating point variable to a MAT-file. If you change the model's filename, you will need to regenerate the operating point.



Digital Control Design

Attempting to Tune the PID Gains

At this point, you will need to tune the controller gains (proportional, integral, and derivative) to obtain the desirable closed-loop system response.

Common methods available for tuning the controller gains are listed below:

- **System response tuning** – You can iteratively modify controller gains and test them via simulation to verify whether they produce the desired response characteristics. This can be time-consuming because many simulations may be required.

Note You can also manually write optimization algorithms, or use the Response Optimizer app in Simulink Design Optimization™ to tune the system response.

- **Linear control theory** – You can apply traditional control design techniques (using Bode plots, for example) to tune the controller gains. This approach requires an LTI representation of the plant model being controlled, and so the plant must be either linear or linearizable. Once the controller performs satisfactorily with the linearized system, you can test the controller gains with the nonlinear system.

Note You can also use automated tuning algorithms based on linear control theory, such as the PID Tuner app in Simulink Control Design.

Control engineers often prefer the latter approach because it enables them to design control systems using familiar traditional techniques. Additionally, these techniques do not require iterating through simulations.

To start tuning the PID controller parameters, double-click the Discrete PID Controller block to open the block parameters dialog and then click **Tune**. This will open the PID Tuner app to tune this controller block.

When the PID Tuner app opens, it attempts to linearize the plant model about the model's initial condition operating point. If the plant cannot be linearized, the app will display a warning.

Power electronic switching converter models (that is, piecewise-linear

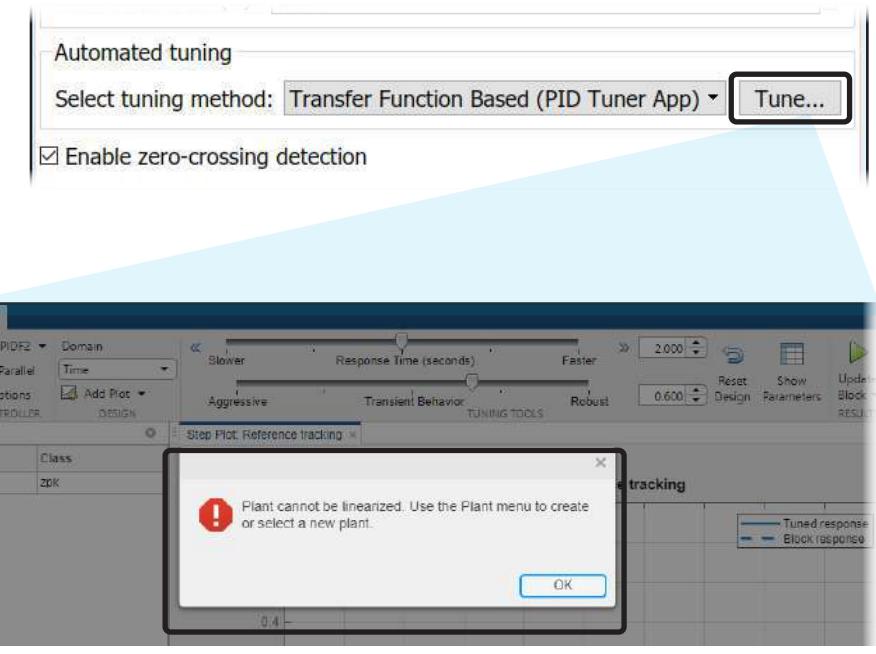
Try

Change the position of the Manual Switch block to enable the closed-loop configuration. Open the PID Tuner app and try to tune the voltage-mode controller. Is the boost converter model linearizable?

When done, change the position of the Manual Switch block back to the open-loop configuration.

switching or nonlinear switching models) generally fail this linearization process because of the discontinuous or highly nonlinear nature of the switching events/transients. To work around this issue, you can create an estimated LTI representation of the boost converter. In this course, you will estimate the plant model in the frequency domain, which is a common approach for power electronics control applications.

With an estimated LTI representation of the boost converter, you will then be able to use the PID Tuner app to tune the controller gains.



Digital Control Design

Understanding Frequency Response Estimation

On the previous page, you saw that the plant could not be linearized. Instead, you will use Frequency Response Estimation (FRE) to obtain an estimated LTI representation of the boost converter plant model.

FRE is the process of superimposing small perturbation signals of controllable amplitude and frequency content onto the input of a system at its steady-state operating point. The input and output response are measured and used to compute the system's frequency response and estimate a transfer function. This transfer function is an estimated LTI system that represents the system dynamics near the operating point.

The perturbation signal can be either

- **Narrowband** – Sine wave containing a single frequency
- **Wideband** – Random, chirp, or pseudo-random binary sequence signals that contain a broad range of frequencies

This chapter focuses on FRE techniques using narrowband perturbation signals. The narrowband approach generally gives high estimation accuracy, but can require more simulation and analysis time than the wideband approaches.

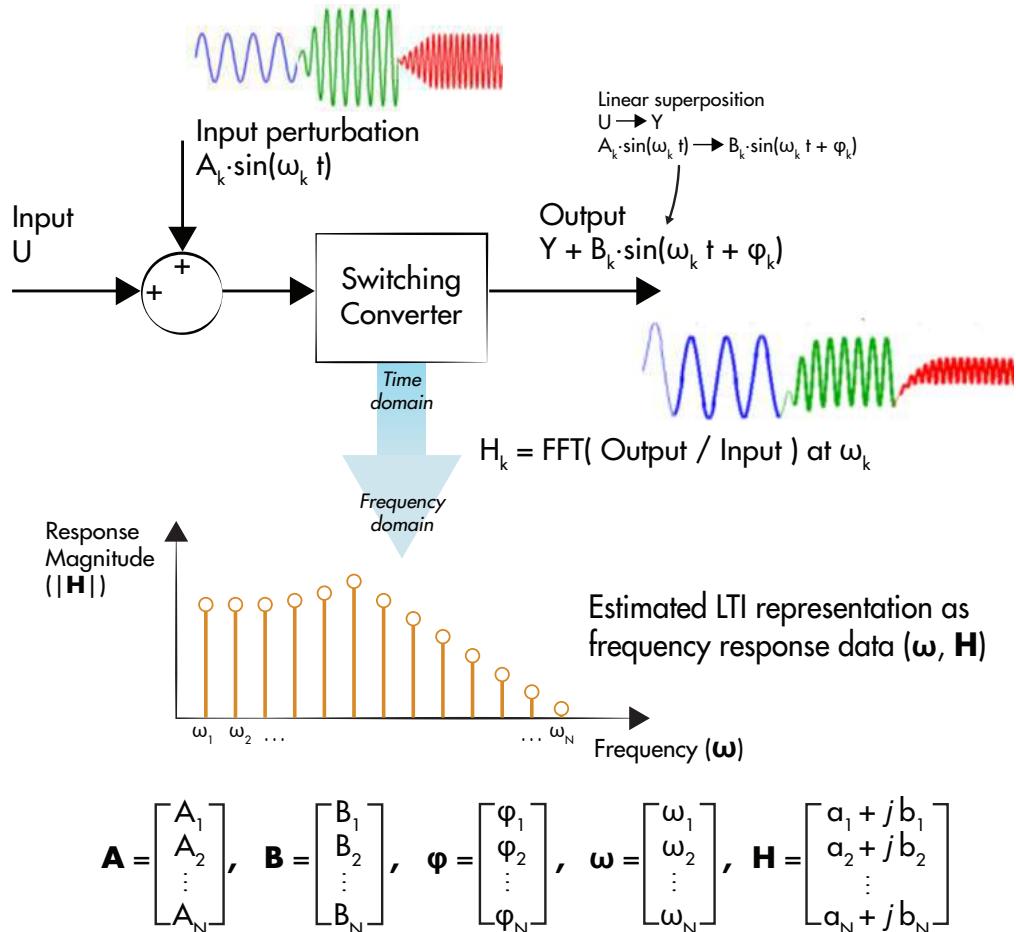
A narrowband perturbation signal is also known as an AC sweep or a sinestream, because the sine wave is being swept across a frequency range to cover the intended frequency spectrum. Specialized equipment widely used in power electronics applications, such as network analyzers, also uses these narrowband perturbations.

Note You can learn more about frequency response estimation in the [Simulink Control Design > Frequency Response Estimation > Frequency Response Estimation Basics](#) documentation.

In Simulink, narrowband frequency response estimation involves

1. Injecting the perturbation signal one frequency at a time.
2. Measuring input and output responses for each injected frequency.
3. Computing the Fast Fourier Transform (FFT) of the input and output responses.

The result is a frequency response data (FRD) object, which contains the frequency response data as a function of the discrete frequency points.



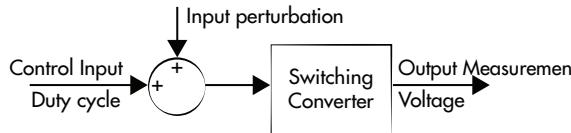
Digital Control Design

Setting Analysis Points

To perform frequency response estimation on the boost converter, you first need to indicate the following analysis points in the model:

- Input perturbation** – Point where the perturbation will be injected.
- Output measurement** – Point where the output will be measured.

In the boost converter model, the input perturbation analysis point is on the duty cycle signal and the output measurement analysis point is on the sampled voltage signal. This enables you to estimate the power converter control-to-output transfer function, which will be used to tune the controller.



To configure these analysis points,

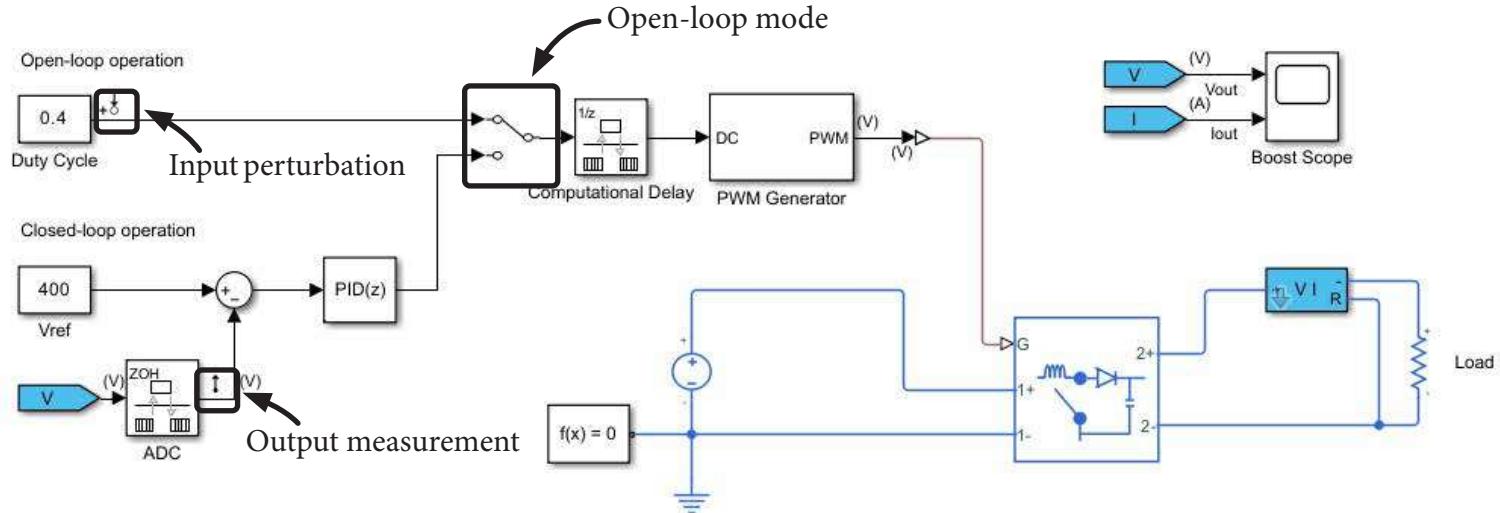
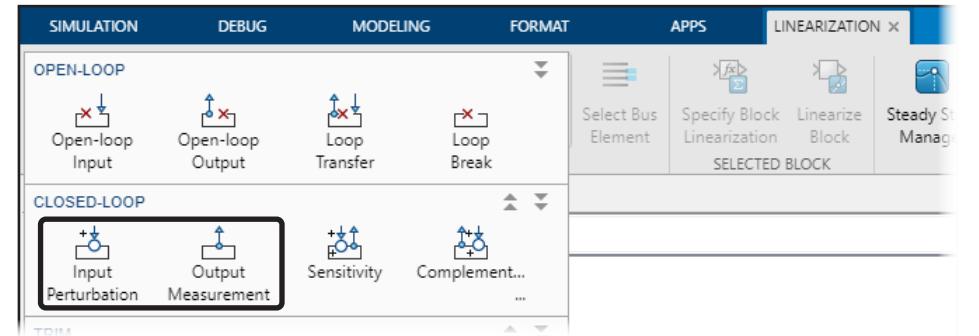
1. Make sure the position of the Manual Switch block is in the open-loop configuration.
2. From the **Linearization** tab, select the signal connected to the output of the Constant block named **Duty Cycle** and click on **Input Perturbation** in the **Insert Analysis Points** section. The signal will show the $\pm\Delta$ icon. This perturbation will be defined on the next page.
3. Select the signal connected to the output of the Rate Transition block named **ADC**. Select **Output Measurement** in the **Insert Analysis Points** section. The signal will show the \ddagger icon.

Try

Follow the steps on this page to set the input perturbation and output measurement analysis points in the model.

>> boostConv3_control

Note With the open-loop configuration you could alternatively use the open-loop input and output analysis points. However, you cannot use them with an unstable untuned closed-loop configuration. You will use the open-loop input and output analysis points for a stable untuned closed-loop configuration in Chapter 7. Refer to **Specify Portion of Model to Linearize** in the documentation for more information.



Digital Control Design

Setting Up Frequency Response Estimation

Once the analysis points are specified, you can use the Model Linearizer app to set up the frequency response estimation. To do this,

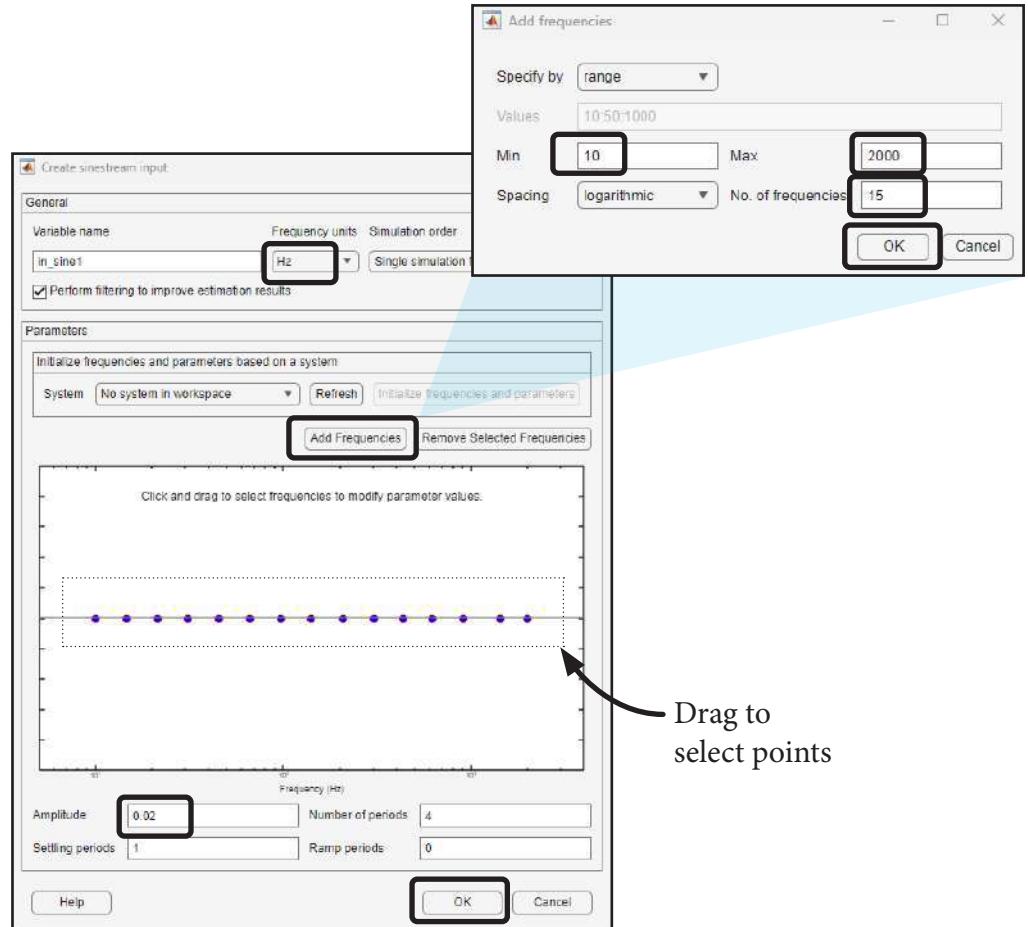
1. Ensure that the model is configured to use the steady-state operating point that you found earlier. Click **Model Settings** in the **Modeling** tab to open the Configuration Parameters dialog. Check that the **Input** and **Initial state** options in the **Data Import/Export** pane are enabled and refer to the correct operating point defined in the base workspace.
2. Then, click **Frequency Response Estimator** in the **Linearization** tab. This will open the **Model Linearizer** app and automatically switch to the **Estimation** tab.
3. Next to **Input Signal** in the **Setup** section, click **Create New > Fixed Sample Time Sinestream**.
4. In the **Specify fixed sample time** dialog, set the **Sample time (sec)** to **1e-4**. Then, click **OK**.
Note This sample time must exactly match the sample times of the input perturbation and output measurement signals.
5. In the **Create sinestream input** dialog, set the **Frequency units** to **Hz**.
6. Click on **Add Frequencies** to define a new sinestream. In the **Add frequencies** dialog, set the parameters shown on the right. Then click **OK**.
7. A dialog may open to notify you that some frequency values have been adjusted. If so, click **Close**. This step ensures that the sampling frequency is an integer multiple of the sinestream frequency values.
8. Click and drag to select all frequencies shown in the plot. Set the **Amplitude** to **0.02**. In general, you should set the amplitude to a value that is from 1% to 5% of the input value to ensure that it is a *small* perturbation. Then, click **OK**.

This will create a Sinestream object in the Linear Analysis Workspace.

Try

Follow the steps on this page to create a new sinestream input to be used in the frequency response estimation.

Note With the Parallel Computing Toolbox™, you can run the simulations for frequency response estimation in parallel. For more information on this workflow, see **Simulink Control Design > Control System Design and Tuning > PID Controller Tuning > Model-Based PID Controller Tuning > Design Controller for Power Electronics Model Using Frequency Response Data** in the documentation.



Digital Control Design

Running Frequency Response Estimation

At this point, the preparations are complete for frequency response estimation. To summarize, these included the following:

- Capturing a steady-state operating point and configuring the model to start simulation from this operating point
- Defining input perturbation and output measurement analysis points in the open-loop model
- Defining a fixed sample time sinestream containing a set of frequencies and amplitudes to perturb the model input

To start the frequency response estimation, click **Bode** in the **Estimate** section of the **Estimation** tab. During the estimation, the model will simulate with the sinestream perturbation signal injected onto the duty cycle signal. The response of the sampled output voltage is measured. The estimation creates a *frequency response data* object, which is the non-parametric estimation.

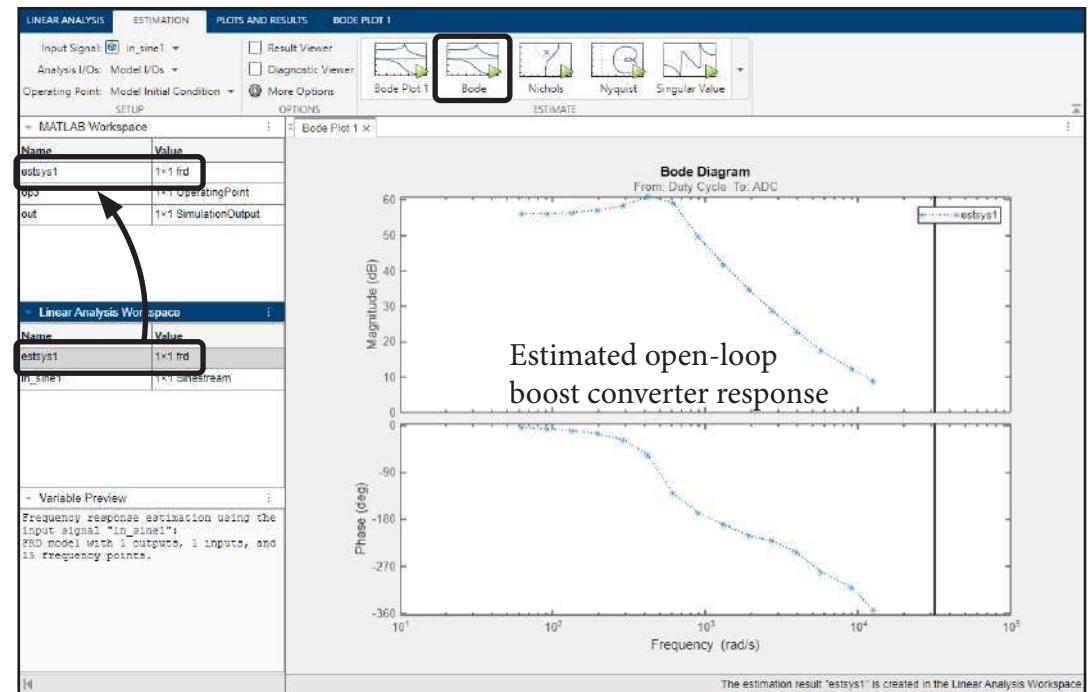
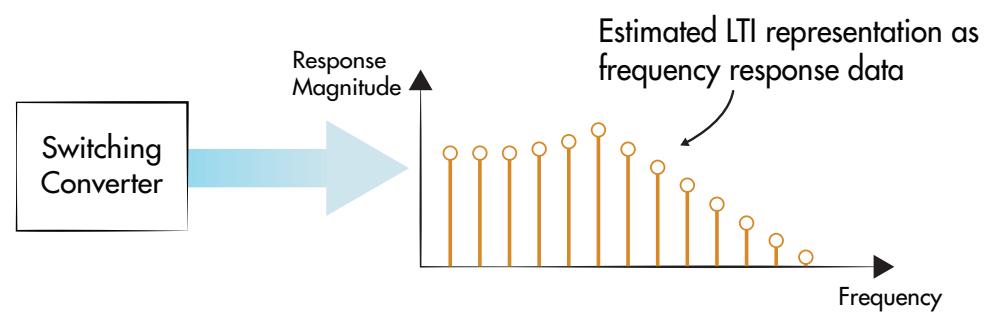
Once the estimation is completed, the Bode plot of the estimated system is displayed. You can right-click in the magnitude or phase sections of the Bode plot and use the **Characteristics** menu to overlay response characteristics. Use these tools to see if the generated results match your expectations for the system.

When finished, you will need to right-click on the `estsys1` object from the **Linear Analysis Workspace** and click on **Export to MATLAB Workspace** so that it will be accessible for controller tuning with the PID Tuner app.

Note Recall that the boost converter model is already configured to start from a steady-state operating point, so you do not need to change the **Operating Points** option. However, you can also use this option to select or generate a different point.

Try

Run the frequency response estimation analysis. Then, export the resulting `estsys1` `frd` object to the base workspace.



Digital Control Design

Tuning the Controller with PID Tuner

Now that you have an estimated LTI representation of the boost converter in the MATLAB workspace, you can use it to tune the voltage-mode controller using the PID Tuner app. To do this,

1. Set the Manual Switch block to operate in closed-loop mode.
2. Double-click the Discrete PID Controller block to open the block dialog parameters. Note the **Proportional (P)**, **Integral (I)**, and **Derivative (D)** default parameter values.
3. Click **Tune** to open the PID Tuner app.
4. As you saw earlier in this chapter, the app will try to linearize the plant. Ignore the linearization failure. Select **Plant > Import**. Then, select the frequency response data object **estsys1** and click **OK**.
5. Then, select **Add Plot > Bode > Open loop**. This will open a Bode plot showing the open-loop transfer function, which includes the controller and plant as shown on the right.
6. Close the **Step Plot: Reference tracking** plot because this plot will not be useful to view the estimated plant's response.

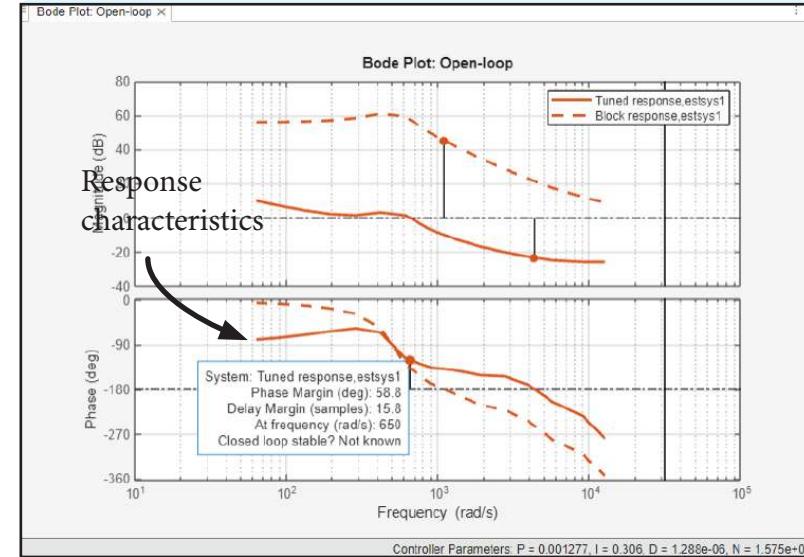
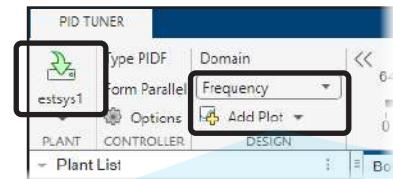
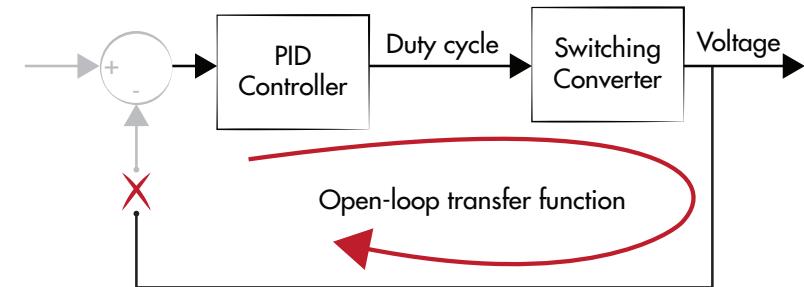
The Bode plot of the open-loop transfer function shows two responses:

- **Block response** – Response using the current gain values in the Discrete PID Controller block
- **Tuned response** – Response using new gain values set by the PID Tuner app

Before tuning, it can be helpful to also display additional response characteristics. In the Bode plot, right-click and select **Characteristics > All Stability Margins**. You can click the dot inside of the phase plot to show the phase margin and frequency values.

Try

Follow the steps on the page to launch the PID Tuner app. Configure the tuner to use the estimated LTI representation of the boost converter. Then, generate a Bode plot of the open-loop transfer function.



Digital Control Design

Tuning the Controller with PID Tuner (Continued)

Now, you can use the PID Tuner app to tune the voltage-mode controller gains. The **Domain** option in the **PID Tuner** tab lets you tune in the domain that you are most comfortable with:

- **Time** – Tune the response time and transient behavior of the response
- **Frequency** – Tune the bandwidth and phase margin of the response

In this chapter, you will tune the voltage-mode controller in the frequency domain, so set the **Domain** parameter to **Frequency**. Notice that the sliders in the toolbar now enable you to set the desired *bandwidth* and *phase margin* of the system. You can enter specific values for these parameters using the edit boxes next to the sliders.

Use the general guidelines below to begin tuning your controller:

- **Bandwidth** – The bandwidth must be less than the Nyquist frequency, which is half of the switching frequency. Try starting with a bandwidth ω_c that is about 10 times smaller than the Nyquist frequency:

$$\omega_{bw} \leq \frac{\pi}{10 \cdot T_{sw}}$$

where T_{sw} is switching period.

- **Phase margin** – Choose a phase margin between 45 and 60 degrees for generally acceptable performance.

Then, you can adjust the **Bandwidth** and **Phase Margin** sliders until the Bode plot displays an acceptable response that meets your design requirements. For additional information while tuning, click **Show Parameters**.

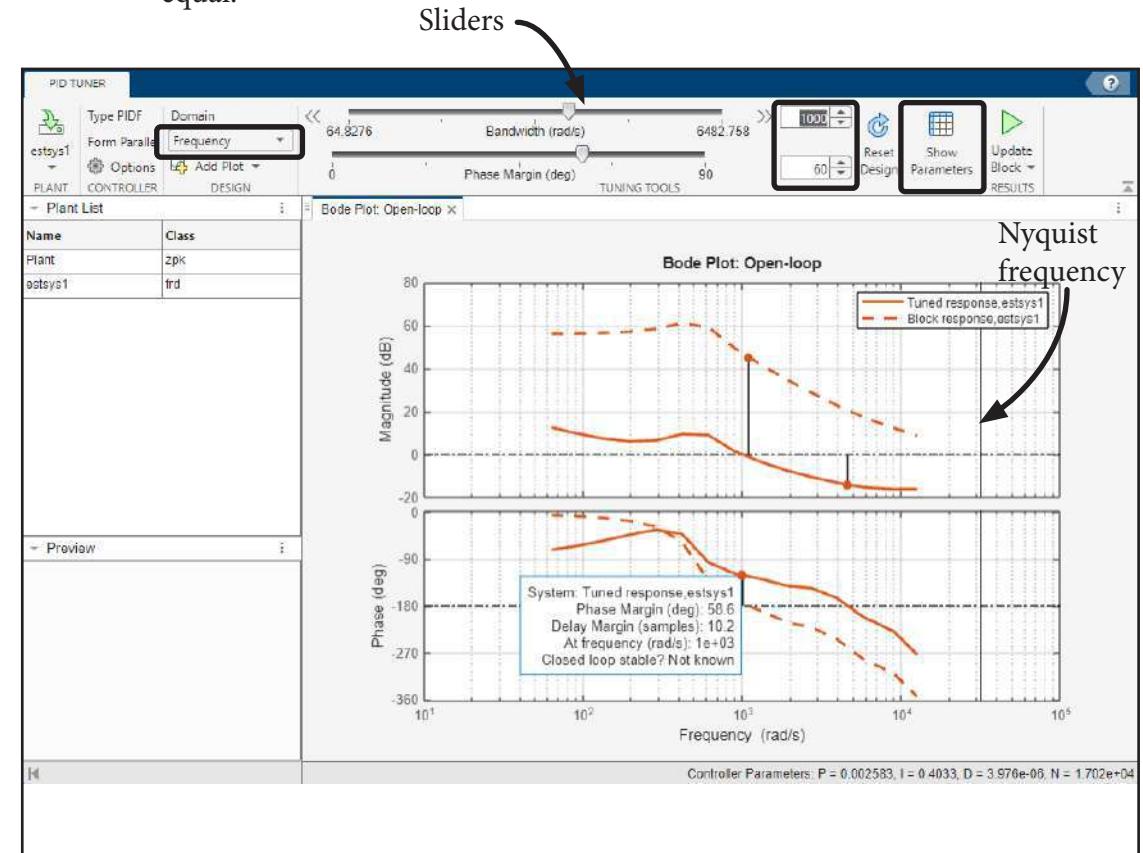
Try

Set the PID Tuner app to tune in the frequency domain. Try moving the **Bandwidth** and **Phase Margin** sliders to tune the controller. View the response of the open-loop transfer function in the Bode plot.

Then, manually enter the following parameters:

- **Bandwidth** is 1000 rad/s.
- **Phase margin** is 60 deg.

Note In the course example, the controller sampling frequency is equal to the switching frequency. In some systems, these frequencies may not be equal.



Digital Control Design

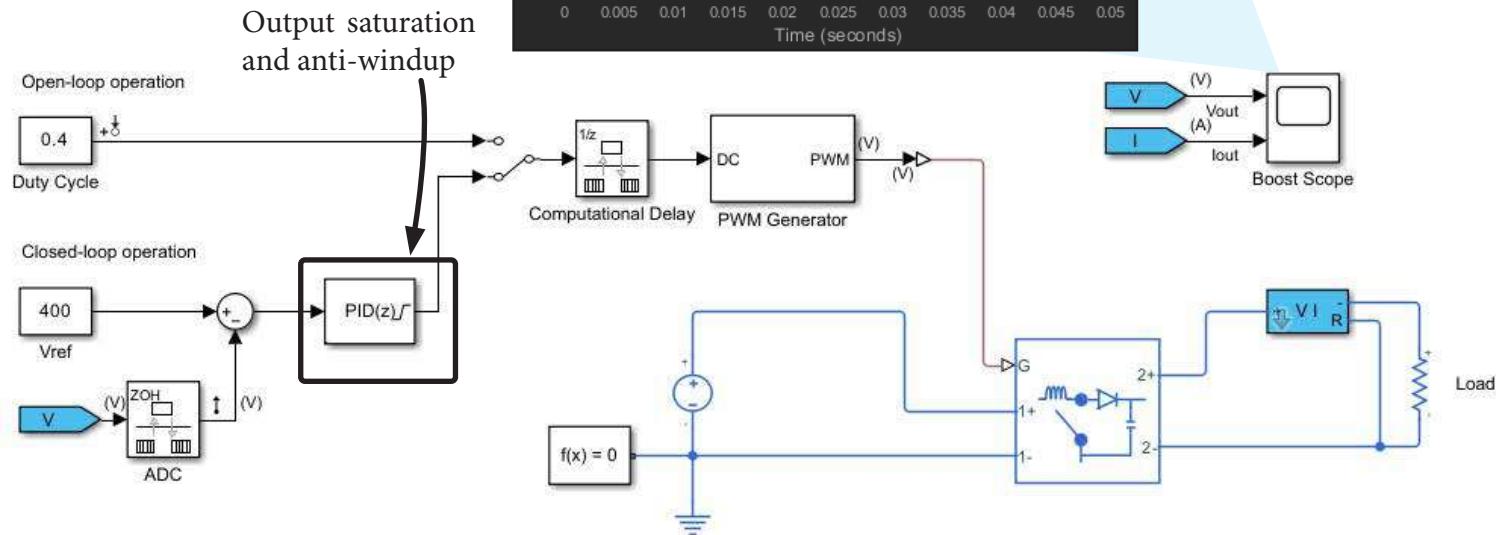
Tuning the Controller with PID Tuner (Continued)

When you are finished tuning, click **Update Block** to apply the calculated gains to the Discrete PID Controller block.

After tuning the voltage-mode controller, you should enable output saturation to limit the range of values of the duty cycle. To do this,

1. Double-click the Discrete PID Controller block to open its dialog parameters.
2. In the **Saturation** tab, make the following changes:
 - Enable the **Limit output** option.
 - Set the **Upper limit** to **0 .85**.
 - Set the **Lower limit** to **0 .15**.
 - Set the **Anti-windup Method** to **clamping**.
3. In the **Initialization** tab, set the **Integrator** and **Filter** initial conditions to **0 .15**.
4. Click **OK**. The icon on the Discrete PID Controller block will change to indicate that output saturation is enabled.

Before simulating the model, make sure that the model does not start from the steady-state operating point. Open the Configuration Parameters dialog (click **Model Settings** in the **Modeling** tab) and ensure that the **Input** and **Initial state** options in the **Data Import/Export** pane are disabled.



Try

Apply the calculated gains to the Discrete PID Controller block and enable output saturation, anti-windup, and initialization as described on this page.

Ensure that the model is not configured to begin simulation from the steady-state operating point found earlier in this chapter.

Simulate the model and inspect the output voltage in the scope. How does the controller perform?

>> boostConv4_control

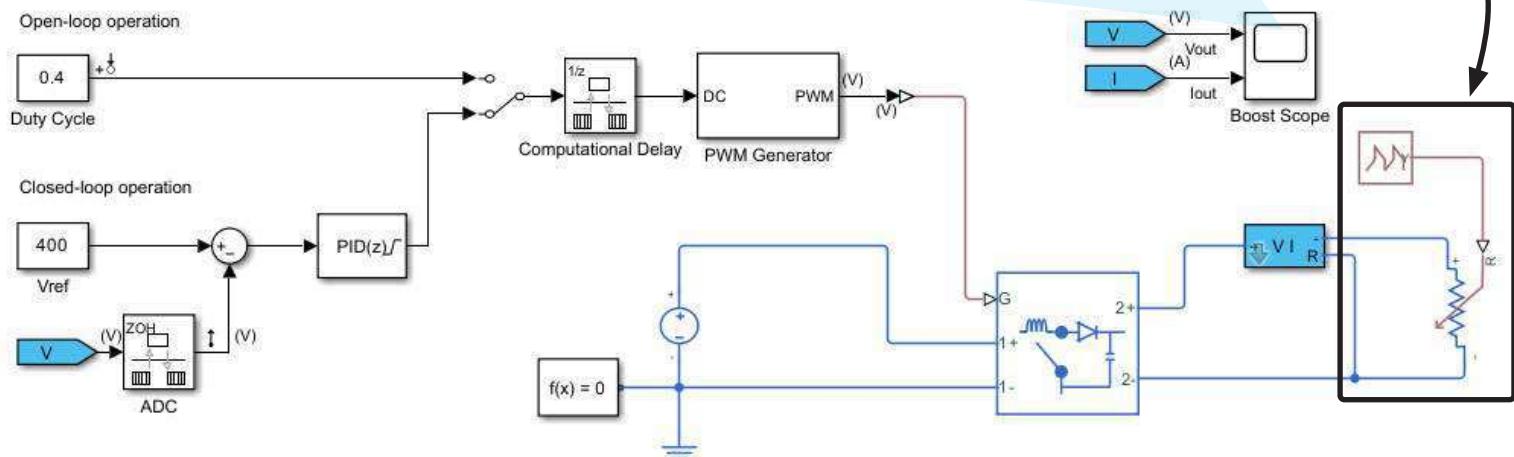
Digital Control Design

Verifying the Controller

After tuning the controller gains, it is a good practice to test the dynamic behavior to ensure that it meets your design specifications. It is especially important to verify the converter behavior before integrating it into a larger system model, such as the HEV motor drive system model that you will build in the final chapter.

First, you can verify that the controller appropriately adapts to disturbances in the load. This will ensure that the controller is able to perform under changing load conditions in the full motor drive model. To do this, implement a variable resistor in the model:

1. Replace the Load resistor with a Variable Resistor block (**Simscape > Foundation Library > Electrical > Electrical Elements library**).
2. Add a PS Repeating Sequence block (**Simscape > Foundation Library > Physical Signals > Sources library**). Connect it to the physical signal input of the Variable Resistor block. Then, set the following parameters
 - Initial output is 2.67.
 - Time offset is 0 s.
 - Signal type is Discrete.
 - Durations is [0.1, 0.1].
 - Start output values is [2.67, 5.34].
3. In the model, set the simulation **Stop time** to 0.3.



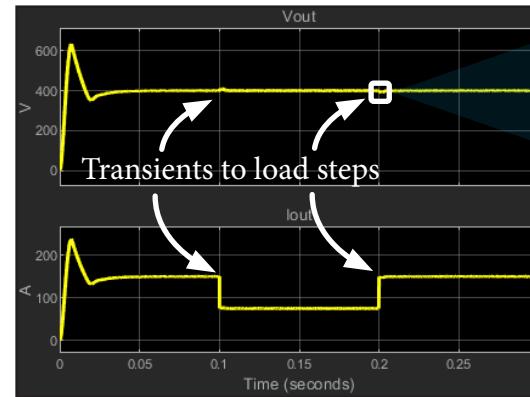
Try

Follow the steps on this page to implement a variable load.

Simulate the model and view the response on the scope. What happens to the voltage and current when the load resistance changes? Does the controller react well to disturbances in the load?

When finished, save the model.

>> boostConv5_control



Variable load

Digital Control Design

Verifying the Controller (Continued)

In the final chapter, you will integrate this boost converter model into the HEV motor drive system model. The system model, however, will use an averaged-switch boost converter model because it simulates faster and the individual switching events do not need to be resolved.

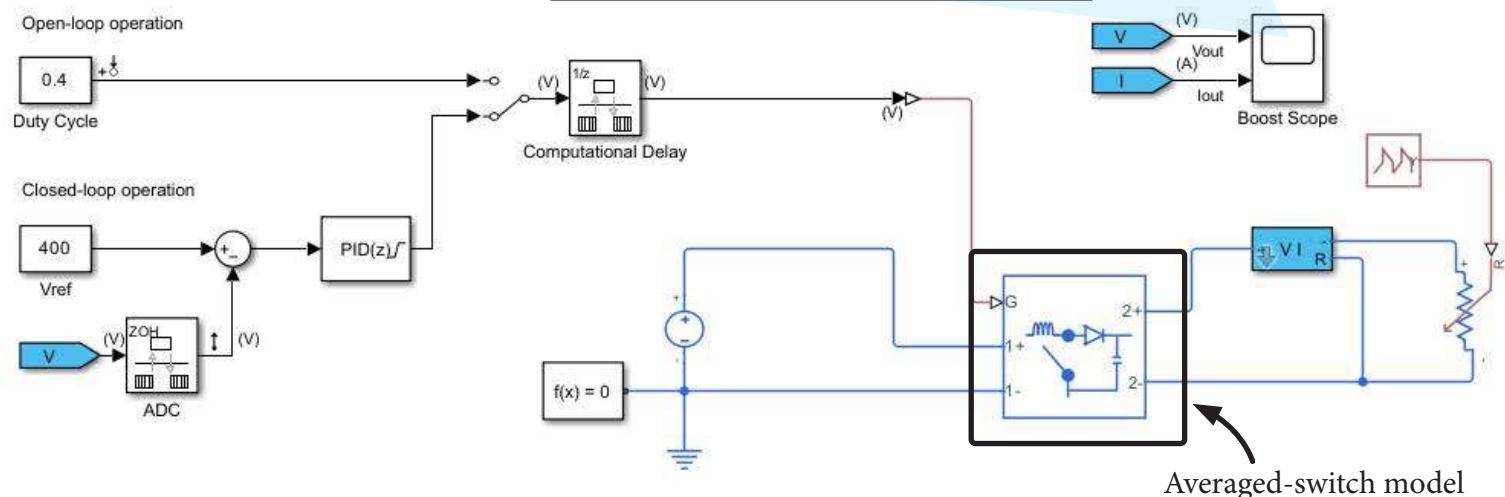
In preparation for the integration later, you should verify that the controller performs well with the averaged-switch model. To do this,

1. Delete the PWM Generator block. Connect the output of the Rate Transition block to the Simulink-PS Converter block.
2. In the Boost Converter block's dialog parameters, set the **Switching device** to **Averaged Switch**.
3. Run the simulation again.

For a more rigorous comparison of the piecewise linear and average switch models, you can use Simscape logging and compare simulation runs using Simulation Data Inspector.

Note If the simulation produces warnings about difficulty simulating the model, try increasing the **Number of consecutive min steps** parameter in the **Solver** pane of the model's **Configuration Parameters** dialog.

Note The Rate Transition block acting as unit delay ($1/z$) prevents the solver from finding an algebraic loop. For more information about refer to the training course **Modeling Physical Systems with Simscape**.



Try

Change the model to use an averaged-switch boost converter. Then, simulate the model and inspect the results. Does the controller perform well with the averaged switch model?

When finished, save the model as **boostConv6_control_avgSwitch**
>> boostConv6_control_avgSwitch

Summary

- Implementing closed-loop voltage control
- Linearizing power electronic converters
- Tuning the controller
- Testing the closed-loop performance

Test Your Knowledge

1. (Select all that apply) Which of the following are valid approaches to tune a controller with a nonlinear plant model?
 - A. Use PID Tuner directly with the nonlinear plant.
 - B. Use PID Tuner with a linearized version of the plant.
 - C. Use the Response Optimizer app to automatically tune the controller.
 - D. It is not possible to tune a controller with a nonlinear plant.
2. (T/F) When the PID Tuner is launched from the Discrete PID Controller block, it first attempts to analytically linearize the plant model.
3. (Select all that apply) When performing frequency response estimation, it is a good practice to
 - A. Inject and measure while the system-under-test is in steady state.
 - B. Use a perturbation signal magnitude of 1-5% of the input value.
 - C. Choose a maximum frequency equal to the switching frequency.
 - D. Use a sinestream perturbation signal with at least 1000 frequencies.
4. (Select all that apply) After tuning a PID Controller, you can verify the dynamic performance of a feedback-controlled power electronic converter by implementing the following tests:
 - A. Check the dynamic response to a time-varying load.
 - B. Check the dynamic response to a time-varying input voltage.
 - C. Check the dynamic response to aging components.

Answers

1. B, C
2. T
3. A, B
4. A, B, C



Power Electronics Control Design with Simulink® and
Simscape™

Modeling DC/AC Three-Phase Inverters

Outline

- Three-phase inverter operation principles
- Modeling an open-loop three-phase inverter in the abc reference frame
- Measuring three-phase physical quantities
- Characterizing harmonics and distortion
- Controlling inverter model fidelity

Chapter Learning Outcomes

The attendee will be able to

- Build and simulate three-phase alternating current (AC) power electronics systems.
- Sense three-phase physical quantities.
- Perform harmonic analysis to characterize distortion.

Modeling DC/AC Three-Phase Inverters

Course Example: Inverter

In the earlier chapters, you learned about modeling, fidelity, and control of DC power electronic systems. Many applications may also have three-phase AC electrical power systems that need to be modeled. In this chapter, you will learn how to use Simscape Electrical™ to build three-phase AC electrical models, measure three-phase voltage and current, and analyze the harmonic content of the three-phase voltage.

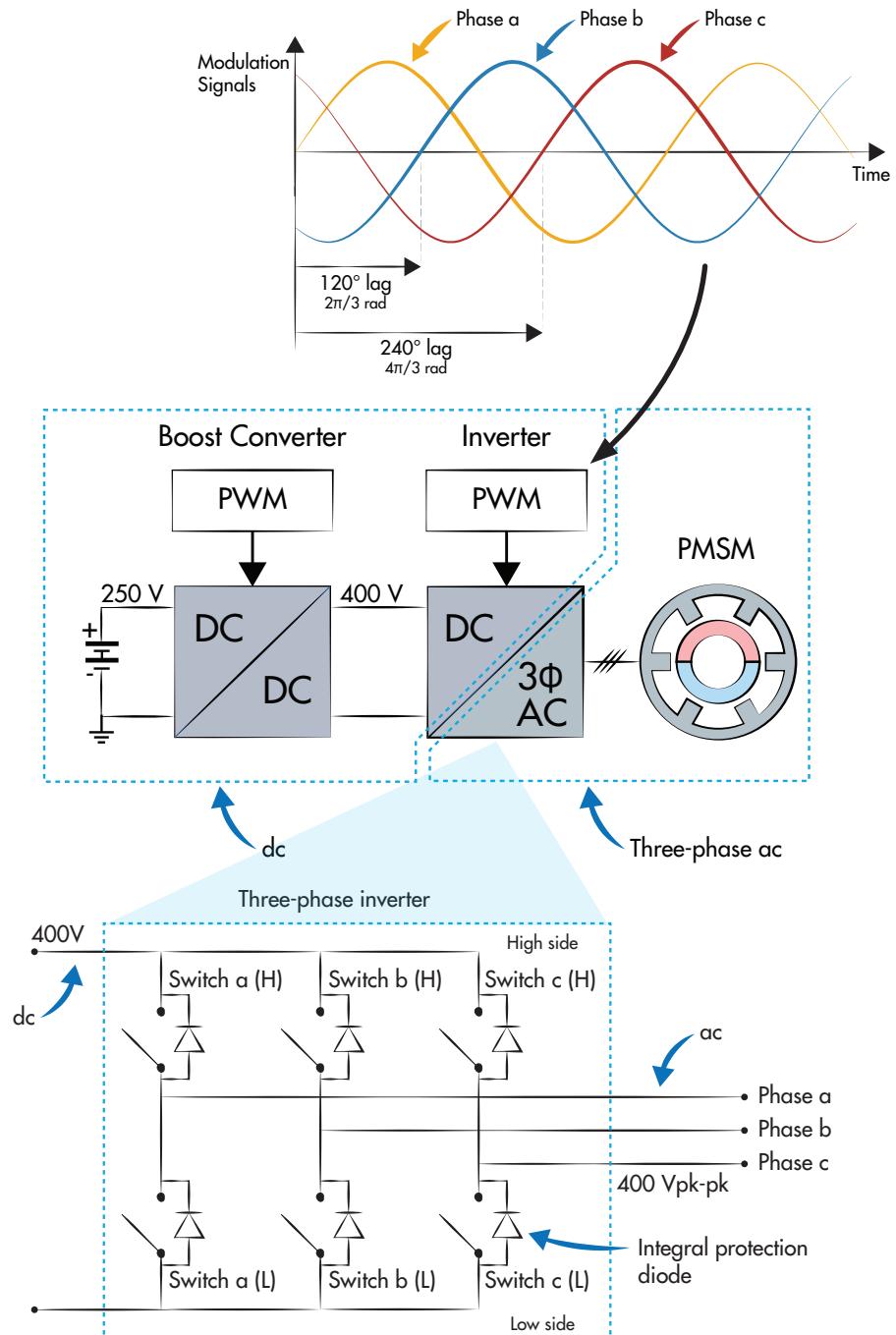
In the hybrid electric vehicle (HEV) motor drive course example model, an inverter is used to convert the 400 V_{dc} output of the boost converter into three-phase AC to drive the motor. The circuit diagram of the three-phase inverter is shown on the right and its main function is to change the DC input to the three-phase AC output.

This power electronic topology features six semiconductor switching devices with anti-parallel diodes (named integral protection diodes in the diagram). Each leg of the inverter has two switching devices, a High- and Low-side (referred as H and L in the diagram), and is connected to one of the phases at the output (referred as *Phase a*, *Phase b*, and *Phase c*).

The switching behavior of the six switches is controlled using three-phase pulse width modulation (PWM). Differently from DC power electronic converters, the modulation consists of a set of three different AC signals at the necessary frequency with the following characteristics:

- **Phase a** – has a 0° phase offset.
- **Phase b** – lags *phase a* by 120° ($2\pi/3$ radians).
- **Phase c** – lags *phase a* by 240° ($4\pi/3$ radians).

In this chapter, you will simulate the inverter with a passive load in an open-loop configuration to verify it produces AC voltages and currents at the expected frequencies.



Modeling DC/AC Three-Phase Inverters

Modeling the Three-Phase Load

You will use Simscape Electrical blocks to model a three-phase inverter with a passive load and simulate it in an open-loop configuration. Recall that when modeling electrical systems, you should model components and verify their behavior independently to reduce complexity when troubleshooting.

To isolate the three-phase inverter, the boost converter's DC output is represented with a DC voltage source. The motor's armature windings are represented by an equivalent phase resistance and inductance. This three-phase load is assumed to be *balanced* — that is, the load on each phase is equal. In Chapter 8, you will replace this load with a motor model to incorporate electro-mechanical dynamics in your model.

Start by opening the `inverter1_start` model. This model already has the following changes made:

- Added Solver Configuration block.
- Added DC Voltage Source block at 400 volts, representing the boost converter's output voltage.
- Set solver and solver tolerances as discussed in Chapter 2.

To begin modeling the three-phase system,

1. Add a Wye-Connected Load block (**Electrical > Passive > RLC Assemblies** library) to represent the PMSM motor's armature windings. Set the following parameter values:
 - **Parameterization** is **Specify component values directly**.
 - **Component structure** is **Series RL**.
 - **Resistance** is **5 mOhm**.
 - **Inductance** is **0.75 mH**.

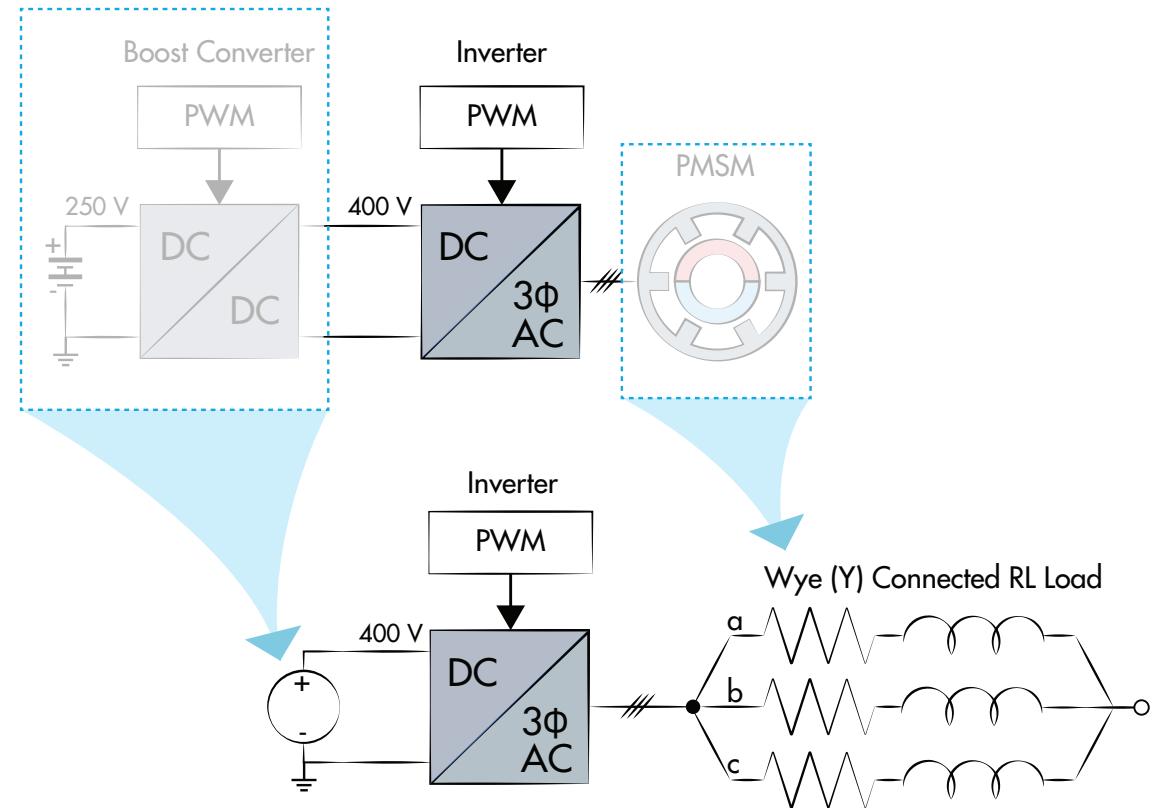
The \sim port of the Wye-Connected Load block is a three-phase electrical port that will be connected to the output of the three-phase inverter later.

Try

Open the `inverter1_start` model. Follow the steps on this page to add a three-phase resistive inductive load to represent the motor.

2. Add an Open Circuit block (**Electrical > Connectors & References** library). Connect it to the n port of the Wye-Connected Load block. This specifies the neutral point to be floating.

Note The **modeling option** parameter allows you to model three-phase connections using two types of ports: **Composite three-phase ports** and **Expanded three-phase ports**.



Modeling DC/AC Three-Phase Inverters

Modeling the Three-Phase Inverter

The three-phase inverter has six semiconductor switching devices that are turned on and off in a predetermined sequence. By manipulating the switching sequence and timing, you can control the frequency and magnitude of the three-phase output voltage.

The inverter modeled in this chapter uses insulated gate bipolar transistors (IGBT) for the semiconductor switching devices. This is a common choice for the rated voltage (400 V) and switching frequency (10 kHz).

In this chapter, you will model the three-phase inverter with a prebuilt block in Simscape Electrical that allows you to select piecewise-linear switching or averaged switch models. However, recall that you could also build the inverter model from discrete components for additional customization and higher fidelity (see Exercise A-20).

To model the three-phase inverter,

1. Add a Converter (Three-Phase) block (**Simscape > Electrical > Semiconductors & Converters > Converters** library). Then, set the following block parameters:

- **Switching device** is IGBT.
- **Threshold Voltage, V_{th}** is 0.5 V.
- **Integral protection diode** is Diode with no dynamics.

The integral protection diode protects the IGBT against reverse voltage by allowing a conducting path from the load back to the DC source when the AC phase voltage goes negative.

2. Select the Converter (Three-Phase) block and click the button in the **Format** tab to flip the block (left to right). This will orient the +/- DC ports to be on the left and the ~ three-phase port to be on the right.
3. Connect the + and - ports of the Converter (Three-Phase) block to the DC voltage supply.
4. Connect the ~ port of the Converter (Three-Phase) block to the Wye-Connected Load block.

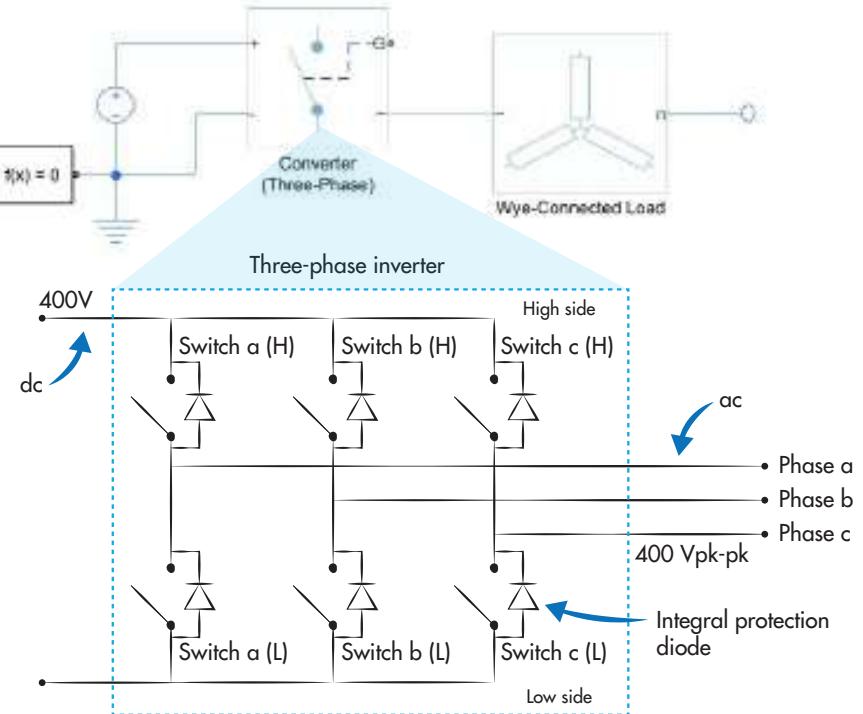
Try

Follow the steps listed on this page to add a three-phase inverter to your model.

>> inverter2

On the next page, you will use a PWM generator to generate signals to connect to the G port of the Converter (Three-Phase) block and control the output of the inverter.

Note As mentioned in the previous page, the **modeling option** parameter allows you to model three-phase ports as either composite or expanded. Notice the different block colors: light blue for composite three-phase ports and dark blue for expanded three-phase ports. You can use the Phase Splitter block (**Electrical > Connectors & References** library) to connect composite and expanded three-phase ports.



Modeling DC/AC Three-Phase Inverters

Generating PWM Signals

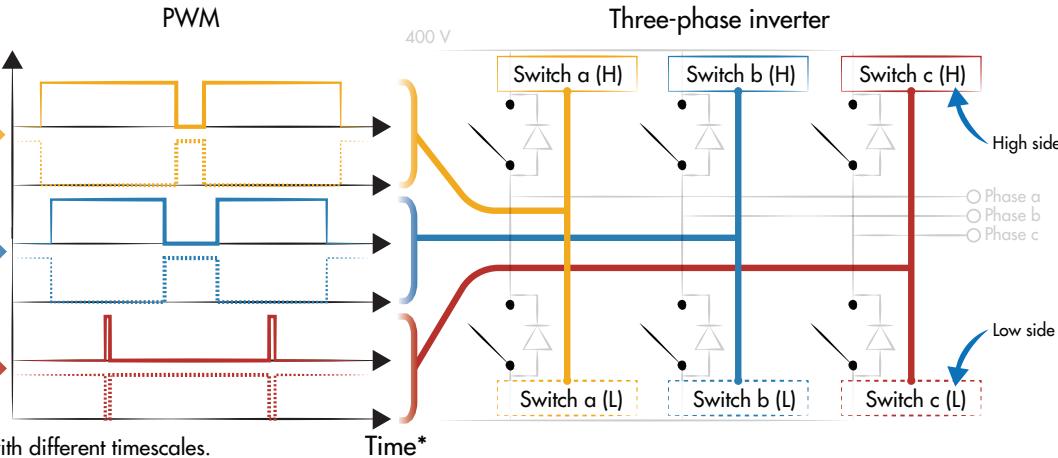
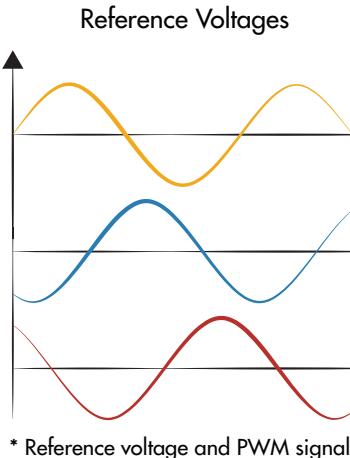
For the three-phase inverter to produce AC output, it needs *gate voltage signals* to command each of the six switches to open or close. Two switches are used to control the AC output for each phase, as shown in the schematic below. The two switches are arranged such that they cannot both be on at the same time, or else they will short-circuit the terminals of the DC supply.

A common approach for generating the gate voltage signals is to use pulse width modulation (PWM). In this chapter, you will use the PWM Generator (Three-Phase, Two-Level) block to generate the gate signals. This block takes three reference voltages as input and converts them into three pairs of complementary PWM waveforms. Each pair of waveforms drive the two switches for a particular phase.

For now, the reference voltage signals are represented with sine waves. This operates the inverter in open-loop configuration. In the next chapter, you will replace the sine waves with a closed-loop feedback configuration.

To generate the PWM signals in your model,

1. Add a PWM Generator (Three-phase, Two-level) block (**Simscape > Electrical > Control > Pulse Width Modulation** library) and set the following parameter values:



Try

Follow the steps listed on this page to add a PWM generator to your model.

- **Switching frequency (Hz)** is **1e4**.
 - **Sample time (s)** is **5e - 6**. Note that this sampling frequency is 20 times the switching frequency.
2. Add a Sine Wave block (**Simulink > Sources** library) to generate time-varying three-phase reference signals. Name it **Vabc**. Connect it to the **Vabc** input of the PWM generator. Set the following parameter values:
 - **Amplitude** to **200**.
 - **Frequency (rad/sec)** is **2*pi*60**. This is the fundamental frequency.
 - **Phase (rad)** is **[0, -2*pi/3, -4*pi/3]**.
 - **Sample time** is **1e - 4**.
 3. Add a Constant block (**Simulink > Sources** library) representing the DC voltage at the inverter input. Name it **Supply Voltage**. Connect it to the **vdc** input of the PWM generator. Set the **Constant value** to **400** and the **Sample time** to **1e - 4**.

Modeling DC/AC Three-Phase Inverters

Generating PWM Signals (Continued)

The PWM Generator (Three-Phase, Two-Level) block has two Simulink® signal outputs, as described below:

- **g** – A vector containing six gate voltage signals, one for each switch. The signals are valued 0 or 1 and are arranged in the following order: Switch a (H), Switch a (L), Switch b (H), Switch b (L), Switch c (H), Switch c (L). This output can be used to command the switches of an inverter using piecewise linear or nonlinear switching models.
- **ModWave** – A vector containing three modulation signals ranging from -1 to 1 based on the averaged gate voltage signals for each phase:

$$MW_x = 2 \bar{g}_{x(H)} - 1 = -2\bar{g}_{x(L)} + 1$$

where the bar denotes the time average over the switching period and x denotes phase a, b, or c. The signals are arranged in the following order: phase a, phase b, phase c. As you will see later, the ModWave output can be used to command an inverter using an averaged switch model.

In this chapter, the inverter uses a piecewise-linear switching model so you can use the g output. To connect the PWM Generator (Three-phase, Two-level) block to the Simscape™ network, you will need to

1. Add a Demux block (**Simulink > Signal Routing** library) to split the vector signal output from g into six individual Simulink signals.
2. Convert each Simulink signal into a Simscape physical signal using **Simulink-PS Converter** blocks.
3. Connect the physical signals to a Six-Pulse Gate Multiplexer block (**Electrical > Semiconductors & Converters > Converters** library).
4. Connect the multiplexer's G port to the converter block's G port.

Alternatively, you can use the Gate Driver block provided in the **trainingLib** library, which conveniently incorporates the Simulink-PS Converter blocks and the multiplexer block, as shown on this page.

Try

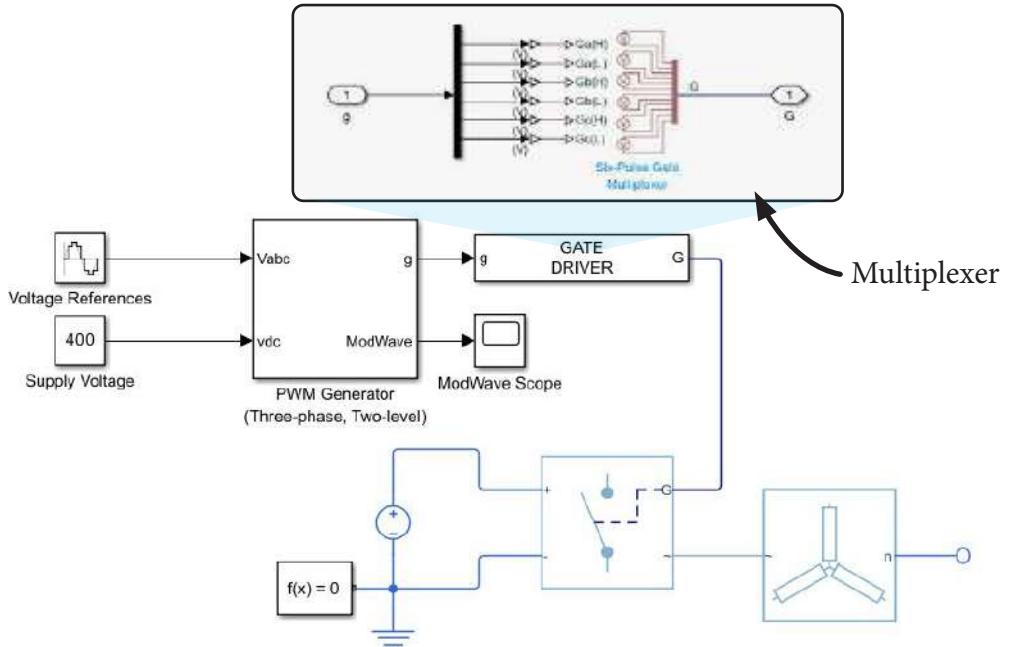
Add a Gate Driver block from the **trainingLib** library to connect the gate voltage signals from the PWM generator to the G port of the converter.

Add a Scope block to visualize the ModWave output from the PWM generator.

>> inverter3

Connect the ModWave output of the PWM generator to a Scope block and name it ModWave Scope. This helps you see when the modulation scheme is saturating (that is when ModWave has values of -1 or 1).

Note For the inverter switches to close, the gate voltage signals must be greater than the **Voltage threshold**, **Vth** parameter in the Converter (Three-Phase) block. In this case, the gate voltage signals have a maximum value of 1 volt and the threshold is 0.5 volts. In other cases, you may need to add a gain to appropriately scale the gate voltage signals.



Modeling DC/AC Three-Phase Inverters

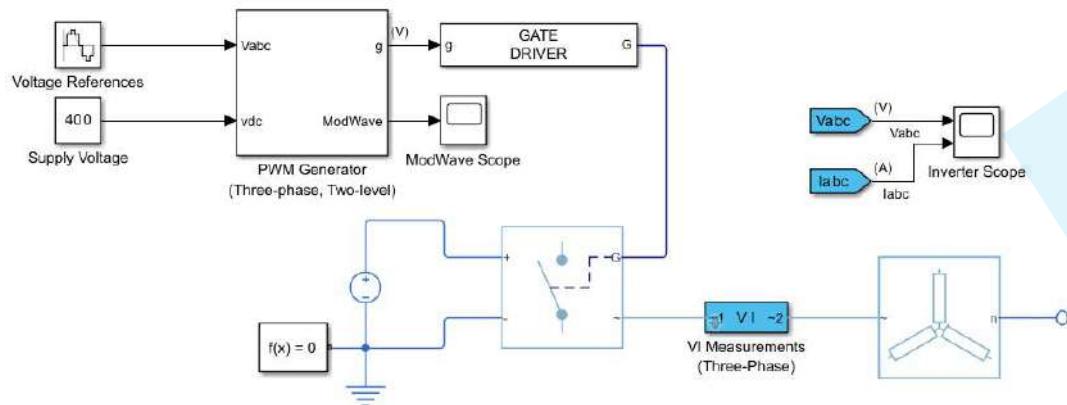
Measuring Three-Phase Quantities

The **Electrical > Sensors & Transducers** library includes a variety of three-phase electrical sensors. You can explore this library and look for sensor blocks that have “Three-Phase” in their names.

To visualize the three-phase voltage and current on the AC side of the inverter, you can use the VI Measurements (Three-Phase) block from the **trainingLib** library. This block contains a Current and Voltage Sensor (Three-Phase) block. The output measurements are transmitted using Goto blocks and can be retrieved using a From block with the tags **Vabc** and **Iabc**, as shown below.

Alternatively, you can add the sensor manually:

1. Delete the physical connection between the converter and the load.
2. Add a Current and Voltage Sensor (Three-Phase) block (**Simscape > Electrical > Sensors & Transducers** library). Set the **Voltage measurement type** to **phase-to-ground voltage**. Then, connect its ~ 1 port to the \sim port of the converter. Connect its ~ 2 port to the \sim port of the load.



Try

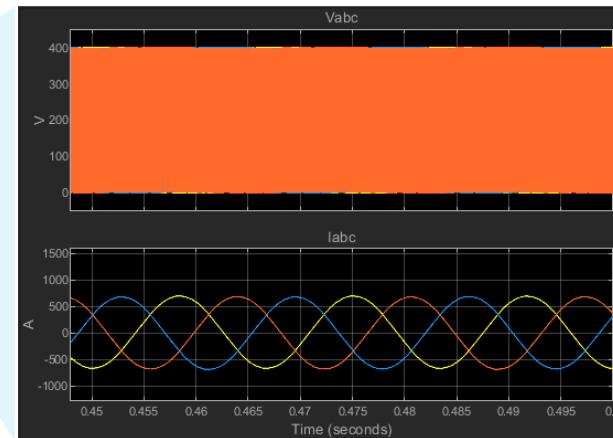
Open the **Electrical > Sensors & Transducers** library and view the different three-phase sensors available.

Add three-phase voltage and current measurements to your model. Simulate the model and inspect the three-phase inverter output.

```
>> inverter4
>> inverter4_noLib
```

3. Add two PS-Simulink Converter blocks (**Simscape > Utilities** library). Connect their inputs to the **V** and **I** physical signal outputs of the sensor. Set their **Output signal unit** parameters to **V** and **I**, respectively
4. Connect the outputs of the PS-Simulink Converter blocks to a scope for visualization. Label the signals **Vabc** and **Iabc**, respectively. Configure the scope to have two displays to show voltage and current separately. Show the time units. Name the scope Inverter Scope.

Note You are measuring the phase-to-ground voltage. Notice that the ground terminal is connected to the negative DC input port of the Converter (Three-Phase) block. You can also measure the voltage between phases by selecting the **Voltage measurement type** to **phase-to-phase voltage** in the Current and Voltage Sensor (Three-Phase) block.

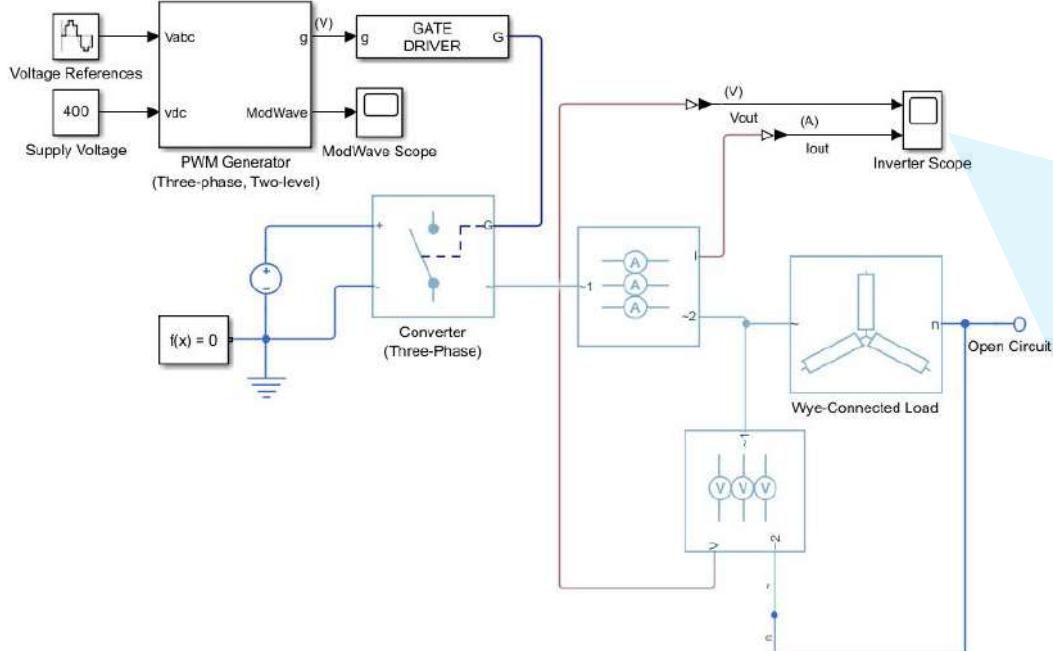


Modeling DC/AC Three-Phase Inverters

Measuring Three-Phase Quantities (Continued)

Measuring the phase-to-neutral voltage when the neutral port of the load is floating requires a voltage sensor across the load with special care of the composite three-phase port \sim and the regular electrical port n. Follow the steps:

1. Open the `inverter4_noLib` model.
2. Delete the Current and Voltage Sensor (Three-Phase) block.
3. Add a Current Sensor (Three-Phase) block (**Simscape > Electrical > Sensors & Transducers** library) in series between the Converter (Three-Phase) block and the Wye-Connected Load block.
4. Connect the physical signal output I to the input of the PS-Simulink Converter block for the current measurement.



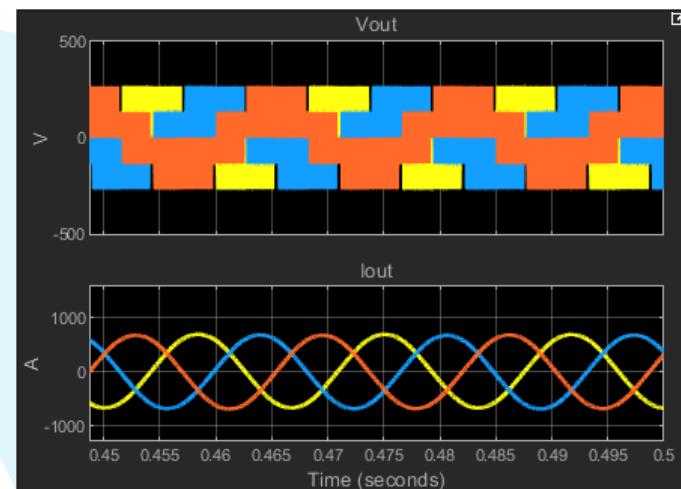
Try

Follow the steps on this page to measure the phase-to-neutral voltage when the neutral port of the load is floating. Simulate the model and inspect the three-phase inverter output.

```
>> inverter4_noLib
>> inverter4_Phase_to_Neutral
```

5. Add a Phase Voltage Sensor (Three-Phase) block (**Simscape > Electrical > Sensors & Transducers** library). Then, connect its ~ 1 port to the \sim port of the Wye-Connected Load block.
6. Add a Neutral Port (Three-Phase) block and connect its \sim -port to the ~ 2 port of the Voltage Sensor (Three-Phase) block. Connect the n port to the n port of the Wye-Connected Load block.
7. Connect the physical signal output V to the input of the PS-Simulink Converter block for the voltage measurement.

Note In this model, the inductor current is very sensitive to its voltage. To reduce the risk of seeing voltage spikes in the results, make sure that you set both the Why-Connected Load's **Parasitic parallel conductance** and the Neutral Port (Three-Phase)'s **Parasitic ground conductance** parameters to zero.



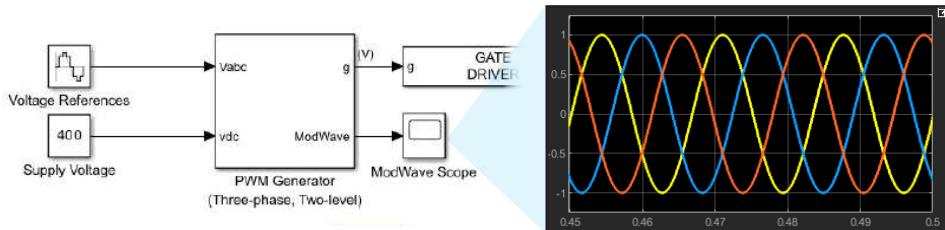
Modeling DC/AC Three-Phase Inverters

Verifying the Inverter in the Time Domain

To verify that the PWM generator and inverter are performing as expected, you can check that for any particular phase (a, b, or c):

- The modulation waveform's value is within the range of -1 to 1, but is not saturated for extended periods of time.
- The peak-to-peak phase-to-ground output voltage is equal to the supplied DC voltage (400 V) and it switches at the expected switching frequency (10 kHz).
- The current waveforms are sinusoidal and have a fundamental frequency matching the voltage references (60 Hz).

To check the modulation waveform, inspect the ModWave Scope after simulating the model. Values of -1 or 1 indicate that the full DC supply is being utilized. The waveform should remain within the range of -1 to 1 without saturating for the inverter to perform within its full power rating.



To check the inverter output voltage and current, open the Inverter Scope and follow the steps below:

- Open the scope window.
- Use the zoom tool () to zoom in to a region of interest.
- Click the Cursor Measurements () button.
- Click to drag the two cursors to encompass one period in a voltage or current signal.
- Examine the $1/\Delta T$ value specified in the Measurements panel.

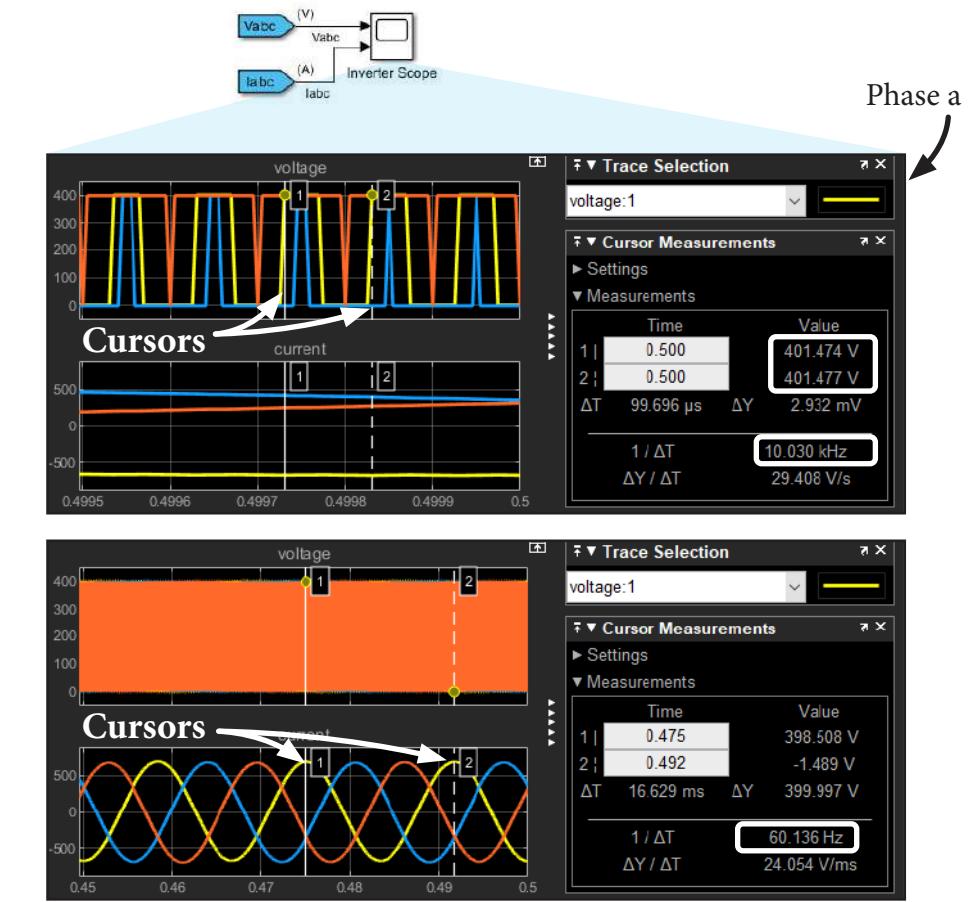
Try

>> inverter4

Open the ModWave Scope to verify that the modulation waveforms cover the full range from -1 to 1 but are not saturated for extended time.

Then, open the Inverter Scope and use cursor measurements to verify that

- Voltage switches between 0 and 400 volts with a frequency of 10 kHz
- Current is sinusoidal with a frequency of 60 Hz



Modeling DC/AC Three-Phase Inverters

Verifying the Inverter Using Harmonic Analysis

You can perform *harmonic analysis* on the AC output of the inverter to characterize its performance. The term *harmonic* refers to integer multiples of the fundamental (60 Hz) and switching (10 kHz) frequencies. It is important to be aware of the harmonics in your system because, beyond certain limits, they can jeopardize the operation of sensitive components.

You can visualize harmonic content using the Spectrum Analyzer block:

1. Add a Selector block (**Simulink > Signal Routing library**). Connect the input to the Vabc From block. In the block's dialog parameters, set the **Index** to 1 to select the *phase a* voltage.
2. Add a Zero-Order Hold block (**Simulink > Discrete library**) and connect it to the Selector block. Set the **Sample time** to $5e-6$.
3. Add a Spectrum Analyzer block (**Simscape > Utilities library**) and connect it to the Zero-Order Hold block.
4. Double-click the Spectrum Analyzer block.

In the **Analyzer** tab,

- In the Views section, select **RMS** under the Spectrum menu. This shows the root mean squared spectrum of the block's input signal.
- In the Bandwidth section, set **RBW (Hz)** to 5. This is the resolution bandwidth, representing the smallest frequency bandwidth that can be resolved in the spectrum.

In the **Estimation** tab,

- In the Frequency Options section, select **Span** and **Center Frequency** for the **Frequency Span**.
- Set **Span (Hz)** to $40e3$. This is the maximum frequency shown in the spectrum.
- Set **Center Frequency (Hz)** to $20e3$. This is the center frequency of the spectrum.

Try

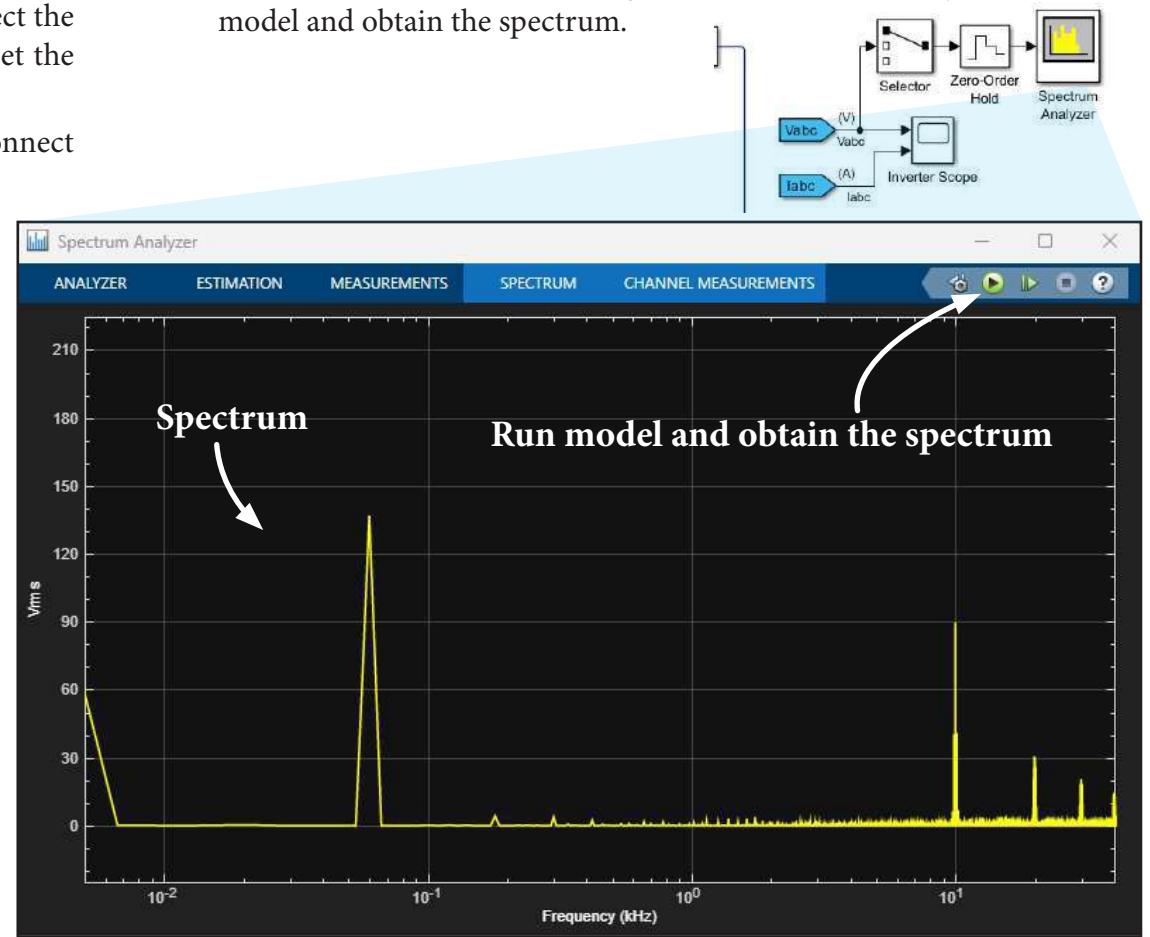
Follow the instructions on this page to add a Spectrum Analyzer block to the model to analyze the harmonics of *phase a* of the inverter voltage.

Then, simulate the model and inspect the results in the Spectrum Analyzer.

>> inverter5_spectrum

In the **Spectrum** tab,

- In the Scale section, set **Log** for the **Frequency Scale**.
- 5. Click the  button on the top-right of the Spectrum Analyzer to run the model and obtain the spectrum.



Modeling DC/AC Three-Phase Inverters

Verifying the Inverter Using Harmonic Analysis (Continued)

Additional analysis tools are available in the Measurements tab inside the Spectrum Analyzer to investigate the measured spectrum.

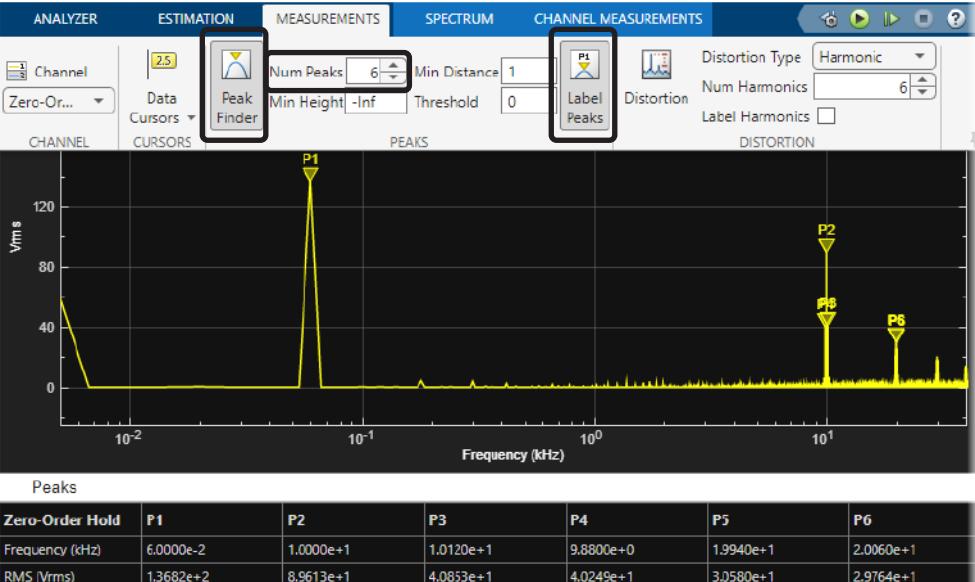
Peak Finder in the Peaks section

The peak finder tool automatically identifies peaks inside the spectrum. This is useful to identify the dominant frequencies in the spectrum. Click on the Peak Finder button to open the Peaks table under the plotted spectrum. Set **Num Peaks** to 6, and enable Label Peaks.

In the **Peaks table**, you can see a list showing the rms value and frequency of all peaks identified in the spectrum.

Distortion Measurements in the Distortion section

You can enable distortion measurements to characterize harmonics and intermodulation products in the spectrum. Click on the Distortion button to



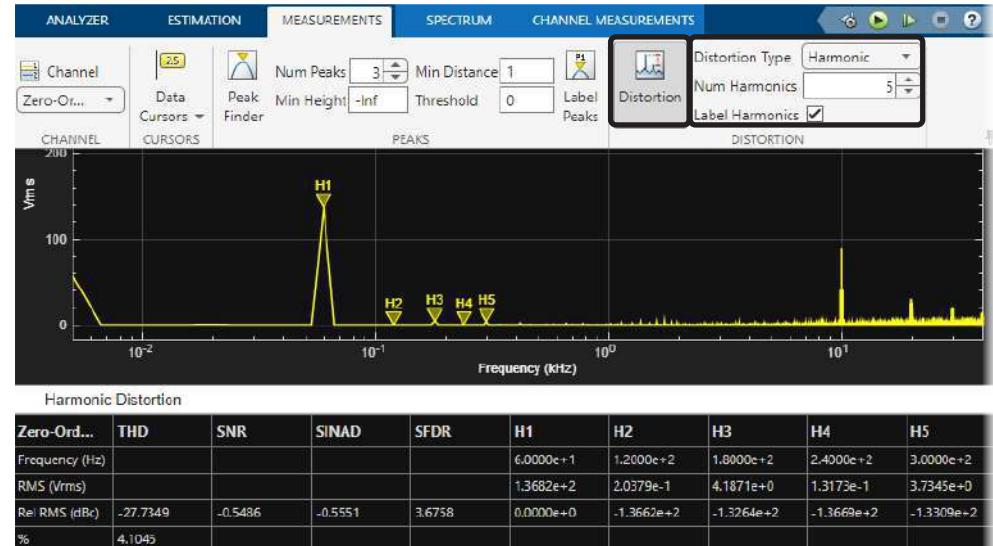
Try

Use the spectrum analyzer to do the following:

- Use the peak finder to identify the fundamental frequency (60 Hz) and the inverter switching frequency (10 kHz).
- Use the harmonic distortion measurements to identify the first five harmonics of the fundamental frequency.
- Use the intermodulation distortion measurements to identify the $2F_2 - F_1$ ($2 \times 10 \text{ kHz} - 60 \text{ Hz} = 19940 \text{ Hz}$) intermodulation product.

open the Harmonic Distortion table or Intermodulation table depending on your choice of the **Distortion Type** (Harmonic or Intermodulation, respectively).

To identify the first five harmonics, set **Distortion Type** to **Harmonic** and set **Num Harmonics** to 5. The frequency and amplitude of the first five harmonics are displayed in the Harmonic **Distortion** table. If the **Label Harmonics** option is enabled, markers will be also displayed to identify the harmonics in the spectrum.



Modeling DC/AC Three-Phase Inverters

Controlling Inverter Model Fidelity

In Chapter 4, you learned how to control model fidelity for DC/DC converters. The same model fidelity considerations can be made for DC/AC (and AC/DC) three-phase converters.

As you have already learned, prebuilt converters are available in the **Electrical > Semiconductors & Converters > Converters** library. Remember that with prebuilt converters you **cannot** model switching losses, show the thermal port, and use pre-parametrization workflows. Prebuilt converters allow you to implement the following fidelity levels.

Average-Value

The Average-Value Voltage Source Converter (Three-Phase) block converts electrical energy from AC to DC voltage or from DC to AC voltage according to an input three-phase modulation wave.

Note The Average-Value Inverter (Three-Phase) and Average-Value Rectifier (Three-Phase) blocks behave as a DC-voltage-controlled AC voltage source and an AC-voltage-controlled DC voltage source, respectively. They do not have a control input as the ratio you specify in the block dialog determines the ratio between the DC (AC) voltage and the AC (DC) voltage.

Averaged-Switch

The Converter (Three-Phase) block with the **Switching device** set to **Averaged switch** requires to be driven by six modulation waveforms, two for each converter leg, with maximum amplitude in the range between 0 and 1. This poses an extra modulation challenge compared to the DC/DC counterpart and it is shown on the following page.

Piecewise-Linear Switching using prebuilt converter blocks

The Converter (Three-Phase) block with the **Switching device** set to anything except for **Averaged switch** (for example IGBT) requires to be driven by six pulse waveforms, two for each converter leg, generated by a three-phase PWM, with amplitude that needs to be chosen depending on the **Voltage threshold, V_{th}** parameter you select in the block dialog.

Try

Read this summary page on inverter model fidelity and go back to Chapters 4 and 3 for more details.

You can model converters using discrete semiconductor devices available in the **Electrical > Semiconductors & Converters** library. Each switching device requires to be driven by a pulse waveform generated by a three-phase PWM. Discrete semiconductor devices can show the thermal port and allow you to model thermal effects. You can implement the following fidelity levels.

Piecewise-Linear Switching using discrete semiconductor device blocks

You can use the blocks labeled as “Ideal, Switching”, “Ideal”, and “Piecewise Linear”. Remember from Chapter 3 that the devices labeled as “Ideal, Switching” allow you to model the switching losses separately from the conduction losses and are supported with pre-parametrization workflows.

Nonlinear Switching

You can use the blocks labeled as “N-Channel”, “P-Channel”, “NPN”, and “PNP”. With these blocks, you can model the dynamic effects that are responsible for non-ideal switching transients of real-world semiconductor devices. To properly drive these devices, you must use a Gate Driver block. The Half-Bridge Driver block can be used for driving both the high- and low-side devices.

Bear in mind that inverter models using nonlinear switching model fidelity

- generally take quite long to simulate, and
- can report on losses, but switching losses are **not** calculated separately.

Modeling DC/AC Three-Phase Inverters

Modeling an Averaged Switch Inverter

In the next chapter, you will design the inverter control using a piecewise-linear switching model. Afterward, you will set the inverter model to use an averaged-switch model to reduce computation time.

The ModWave output of the PWM generator will be used to drive the averaged-switching inverter. The g output of the PWM generator is more computationally expensive and will not be needed. Therefore, you can replace that PWM block with a simpler version that can simulate faster.

To modify your piecewise-linear switching inverter model to use an averaged-switch model,

1. Delete the PWM Generator (Three-Phase, Two-Level) block.
2. Add a PWM Timing and Waveform Generator (Three-phase, Two-level) block (**Electrical > Control > Pulse Width Modulation** library). Connect the Vabc and vdc ports to the Voltage Reference and Supply Voltage blocks, respectively. Then, set the following parameters:
 - **Continuous PWM** is SPWM: sinusoidal PWM.
 - **Switching frequency (Hz)** is 1e4.
3. Replace the Gate Driver block with a ModWave Gate Driver block from the **trainingLib** library. This block converts the three modulation signals into six equivalent average gate voltage signals to drive the inverter. Connect it to the ModWave output of the PWM Timing and Waveform Generator (Three-phase, Two-level) block.
4. Double-click the Converter (Three-Phase) block and set the **Switching device** to **Averaged switch**.
5. Add two Terminator blocks and connect them to TgabcON and TgabcOFF outputs of the PWM Timing and Waveform Generator block.
6. Comment out the blocks for spectrum analysis as all

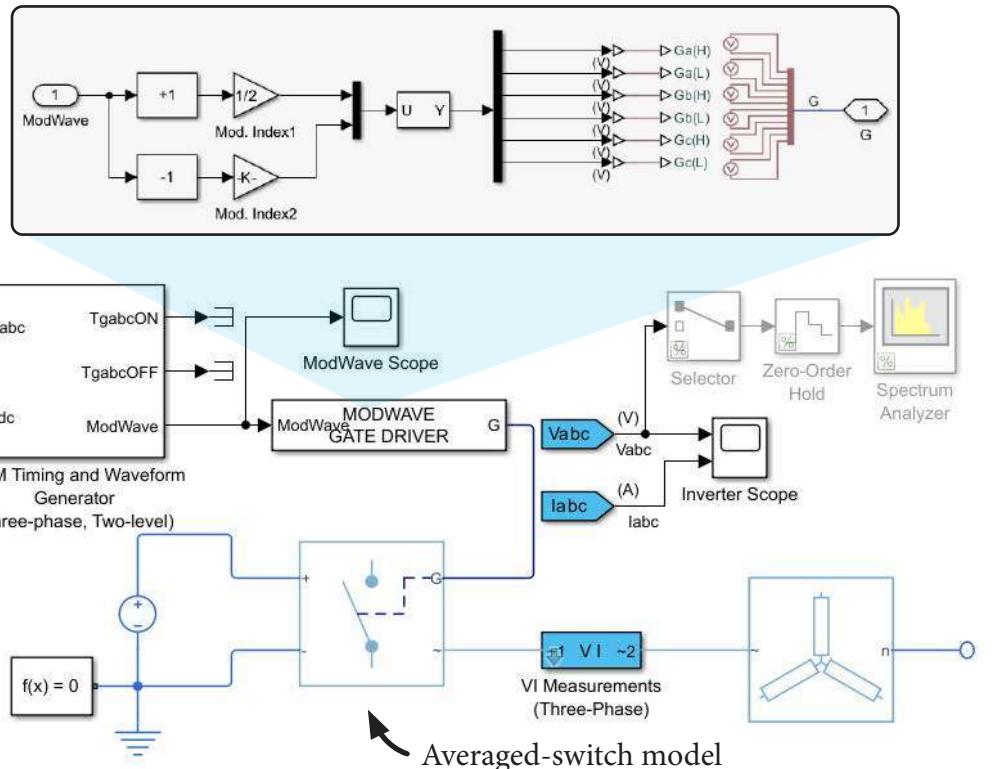
Try

Modify the inverter to use an averaged-switch model. Simulate the model and inspect the results. Note the output voltage now looks sinusoidal because the individual switching events are no longer captured.

>> inverter6_avgSwitch

the switching harmonics are eliminated by the selection of the averaged switch model.

Note If you did not replace the PWM Generator (Three-Phase, Two-Level) block and used its ModWave output to drive the ModWave Gate Driver block, you would “waste” computational time because the g output would be still calculated at the PWM sample time. You can select **Information Overlays > Colors** in the **Debug** tab to view the review sample times used in the model.



Summary

- Modeling a three-phase inverter
- Measuring three-phase physical quantities
- Characterizing harmonics and distortion
- Controlling inverter model fidelity

Test Your Knowledge

1. (T/F) It is possible to view and manipulate the individual phases of a three-phase signal.
2. (T/F) When measuring the currents of a three-phase signal, you must use three separate Current Sensor blocks.
3. (T/F) You can use the Spectrum Analyzer block to visualize the magnitude of the harmonics generated in a switching model.

Answers

1. True - You can view the expanded three-phase ports to gain access to individual phases. You can also use the Phase Splitter block (**Electrical > Connectors & References** library) to split a three-phase connection into its constituent phases.
2. F
3. T



Power Electronics Control Design with Simulink® and
Simscape™

Motor Control

Motor Control

Outline

- Modeling a PMSM motor
- Principles of Field-Oriented Control
- Implementing a motor torque control
- Verifying the motor control design

Motor Control

Chapter Learning Outcomes

The attendee will be able to

- Use prebuilt motor models in Simscape Electrical.
- Understand the principle of Field-Oriented Control.
- Implement a torque controller.

Motor Control

Course Example: Motor Control with a Permanent Magnet Synchronous Motor

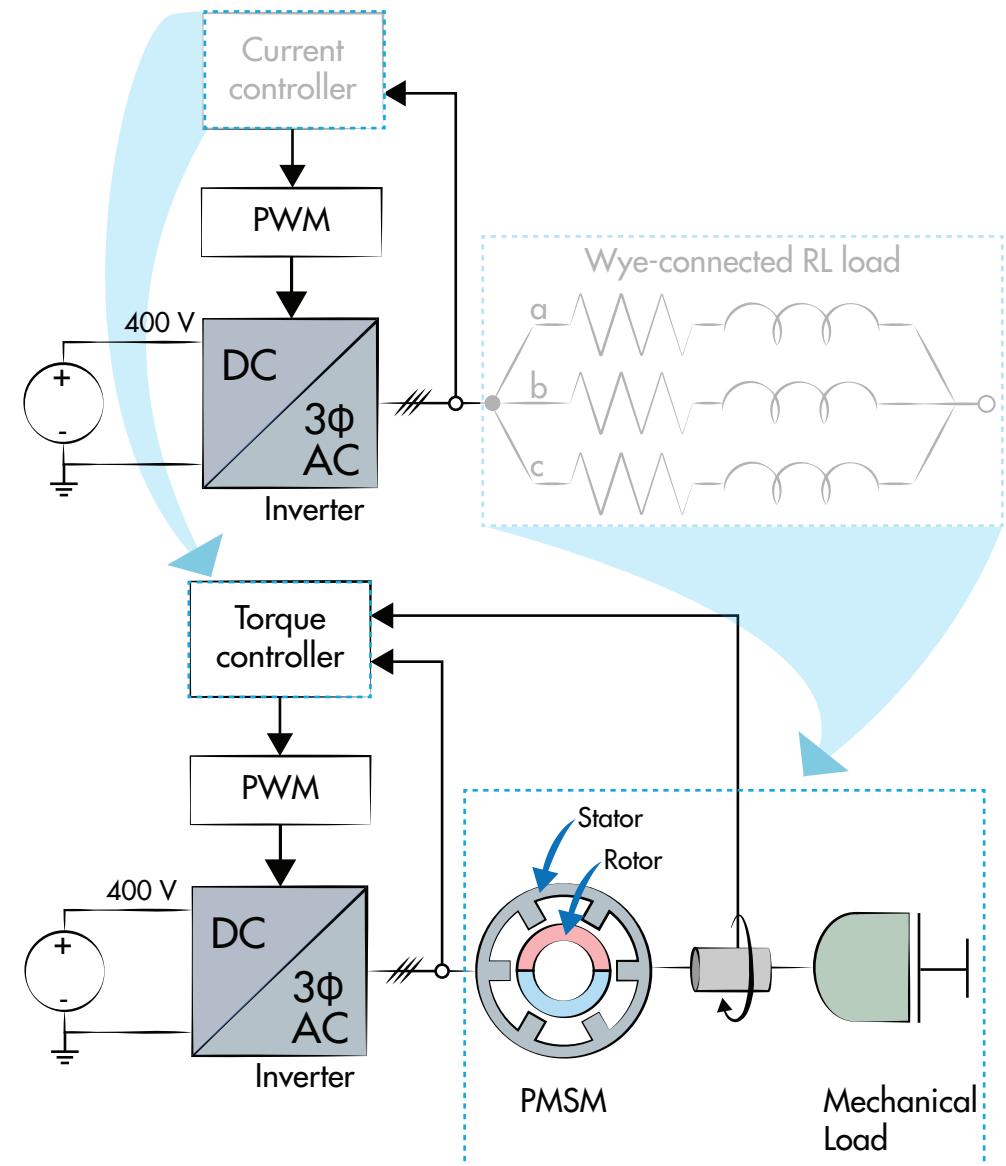
In the hybrid electric vehicle (HEV) motor drive course example, a permanent magnet synchronous motor (PMSM) provides torque to accelerate and decelerate the vehicle in response to the driver pressing the accelerator or brake pedal.

Within the PMSM, three-phase AC currents flow through the stator windings to produce a rotating magnetic field. Torque is generated as this rotating magnetic field interacts with the magnetic field from permanent magnets embedded within the rotor.

The current-controlled three-phase inverter that you modeled in the previous chapter will be used to drive the motor. You will replace the resistive-inductive Wye-connected load with a PMSM model and convert the current controller to a torque controller implementing a *Field-Oriented Control* that aims at regulating the motor current to produce the desired torque.

Then, you will add a dynamometer to set the motor speed. You will verify that the controlled motor design meets its desired operating specifications and can track a time-varying torque reference.

Note This chapter uses motor, inverter, and controller using Simscape Electrical and Simulink blocks. You can also use Motor Control Blockset™, which provides additional blocks and reference examples to interface your design with hardware. The blocks in this blockset are already optimized for code generation and hardware-in-the-loop workflows. For more information, see **Motor Control Blockset** in the documentation.



Motor Control

Modeling the PMSM

In the previous chapter, you modeled the current-controlled three-phase inverter with a resistive-inductive load. Now, you will replace this load with a PMSM model to more accurately simulate the motor's dynamic response.

The motor is designed to produce 460 N*m of torque at the base speed of 1250 rpm, corresponding to about 60 kW of peak power.

To model the PMSM, first set up the electrical side of the motor:

1. Replace the Wye-Connected Load and Open Circuit blocks with a PMSM block (**Electrical > Electromechanical > Permanent Magnet library**).
2. Set the following parameters in the PMSM block dialog parameters:
 - Number of pole pairs is 4.
 - Permanent magnet flux linkage is 0.3 Wb.
 - Stator d-axis inductance, L_d is 0.75 mH.
 - Stator q-axis inductance, L_q is 0.75 mH.
 - Stator resistance per phase, R_s is 5e-3 Ohm.
 - Zero sequence is Exclude. This excludes modeling the zero-sequence stator inductance, which increases simulation speed. This also disables port n (the electrical conserving port associated with the neutral phase).

Next, set up the mechanical side of the motor:

1. Add a **Mechanical Rotational Reference** block (**Foundation Library > Mechanical > Rotational Elements** library) and connect it to the C port of the PMSM block.
2. Add a Mechanical Measurements block from the **trainingLib** library and connect it to the R port of the PMSM block. This block contains an Ideal Torque Sensor block and an Ideal Rotational Motion Sensor block to measure the torque, angular velocity, and angular position at the motor output. The measurements are transmitted using Goto blocks and can be retrieved using From blocks using **theta**, **rpm**, and **T** tags.

Try

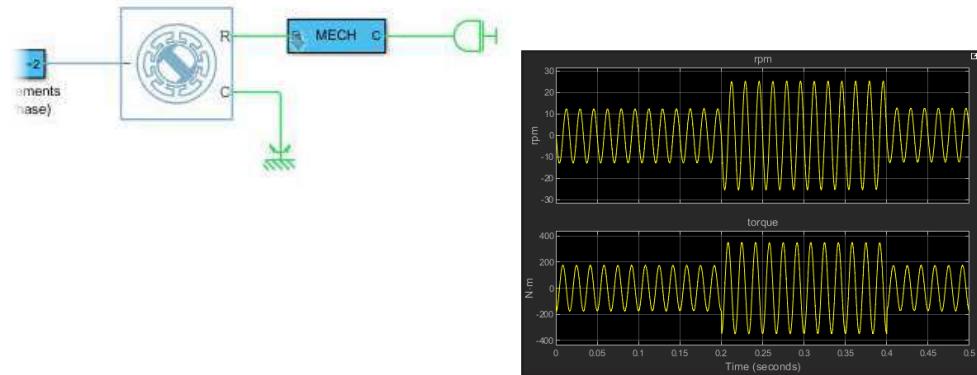
Open the **pmsm1_control_start** model. Follow the steps on this page replace to the Wye-Connected Load with a PMSM block. Simulate the model and inspect the output. What does the motor do? Why?

>> pmsm2_control

3. Add two From blocks. Set the **Goto tag** parameters to **T** and **rpm**, respectively. Then, add a new Scope block, name it **Motor Scope**, and connect it to the two From blocks. Label the two signals **torque** and **rpm**.
4. Add an Inertia block (**Foundation Library > Mechanical > Rotational Elements**). Connect it to the R port of the PMSM. Set the **Inertia** to **0.35 kg*m^2**.

Note#1 The parameters of the PMSM block can be pre-parameterized to match specific suppliers datasheets. A few other blocks from **Electrical > Electromechanical** library can be pre-parameterized. Search for **List of Pre-Parameterized Components** in the documentation.

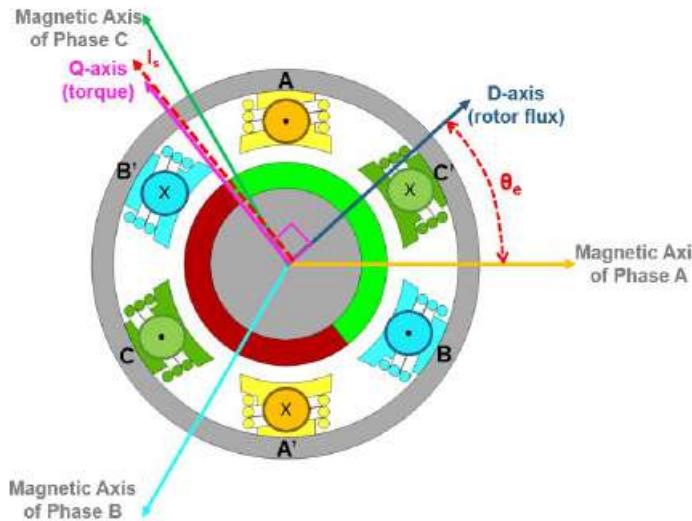
Note#2 For a higher fidelity motor model, you can use the FEM-Parameterized PMSM block (**Electrical > Electromechanical > Permanent Magnet library**). This enables you to model phenomena such as magnetic saturation and spatial harmonic effects.



Motor Control

Torque Control Principles

The electromagnetic torque of a PMSM is generated by the interaction of two rotating magnetic fields, the one produced by the stator and the one by the rotor. The rotating stator magnetic field is produced by energizing the stator coils with AC currents. In a balanced three-phase system, the currents energize three 120-degree spaced coils, thus resulting in a magnetic field rotating at the same frequency of the three-phase currents. The rotor magnetic field is produced by permanent magnets embedded in the rotor structure.



The essence of the *Field-Oriented Control* (FOC) you are going to implement is to align the stator current vector I_s that produces the rotating magnetic field (the red dashed arrow in the figure above) to the rotor torque axis. To have the motor performing at its peak efficiency, called *maximum torque per amp*, the stator magnetic flux must be at 90 electrical degrees compared to the rotor magnetic flux.

Try

Familiarize with the principles of the Field Oriented Control explained in this page.

To align the stator flux vector with the rotor torque axis, the key aspect of FOC is to know the position of the rotor (the angle between the rotor flux axis and the stator phase A magnetic axis) at any moment. From the results of your model, you can see that unsynchronized current excitation from the inverter would make the rotor of the PMSM oscillate.

You will implement a torque controller that helps the motor to start up and produce torque according to a specified reference value. In the HEV application, the reference value will come from the accelerator pedal – when the driver presses the pedal, it will increase the reference value and the PMSM will produce more torque.

For feedback to the controller, direct measurement of PMSM torque is impractical in many applications. Instead, you can apply indirect torque control, which uses current feedback to control the motor torque based on the equation below:

$$T = \frac{3}{2} N_{pp} \lambda_m I_q$$

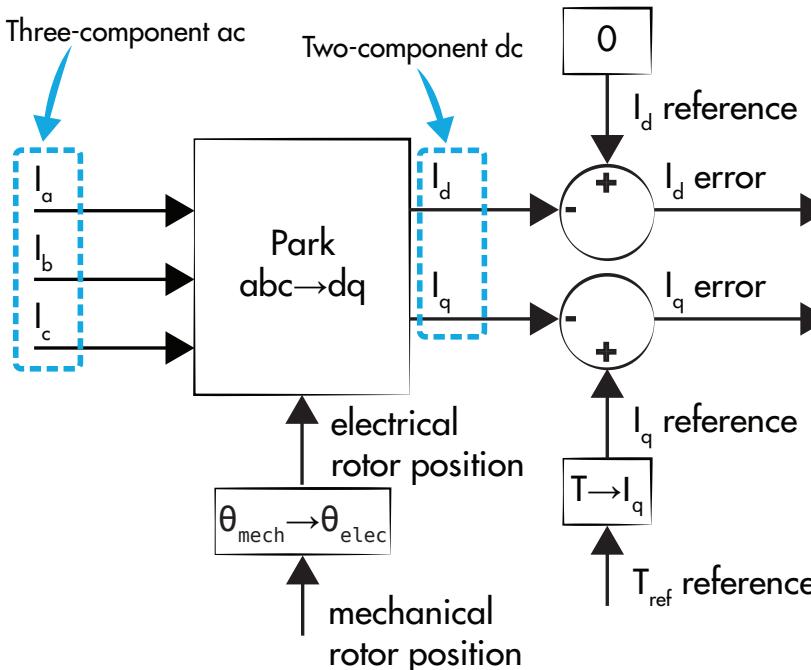
where N_{pp} is the number of pole pairs, λ_m is the flux linkage of the permanent magnet, and I_q is the q-axis current. This equation assumes that the PMSM inductances are symmetric ($L_d=L_q$). Likewise the inverter current control, the motor torque control is implemented into the orthogonal rotating d-q reference frame.

Motor Control

Implementing the Torque Controller

The two proportional-integral (PI) controllers you tuned in the previous chapter used to individually control the I_d and I_q currents have now the goal of regulating the torque of the PMSM. In the torque equation on the previous page, you can see that the torque depends on the I_q current only. Therefore, you will set the I_d controller reference to zero and calculate the I_q controller reference from the desired motor torque.

To get DC quantities in the control algorithm, Park and Inverse Park transforms are deployed. To achieve the desired synchronization, you will need to feed the rotor position to these mathematical transforms. However, you must use the electrical rotor position. This quantity can be easily calculated by multiplying the sensed mechanical rotor position by the number of magnetic pole pairs N_{pp} . Moreover, you need to pay attention to the position of the rotor, i.e. the angle between the rotor flux axis and the stator phase A magnetic axis. This is an important setting of the mathematical transforms.



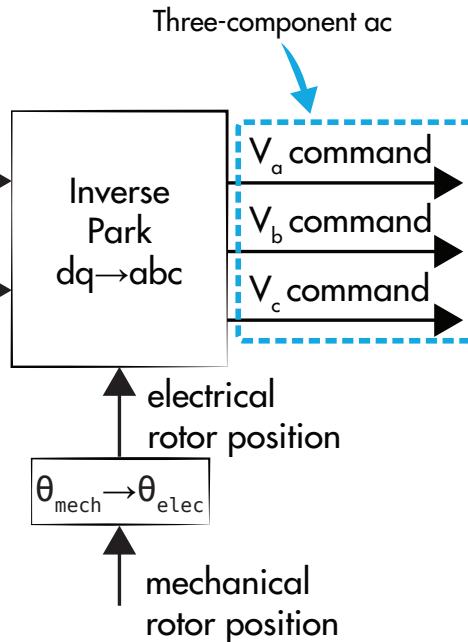
Try

Identify in the model the needed edits to convert the current control to a torque control.

>> pmsm2_control

Note#1 The Simscape Electrical block libraries provide additional prebuilt controller designs that support more advanced and robust motor control strategies. Explore the **Electrical > Control > PMSM Control** library to learn more.

Note#2 Motor Control Blockset can also help with many aspects of designing/implementing a motor drive system, such as parameter estimation and gain tuning.

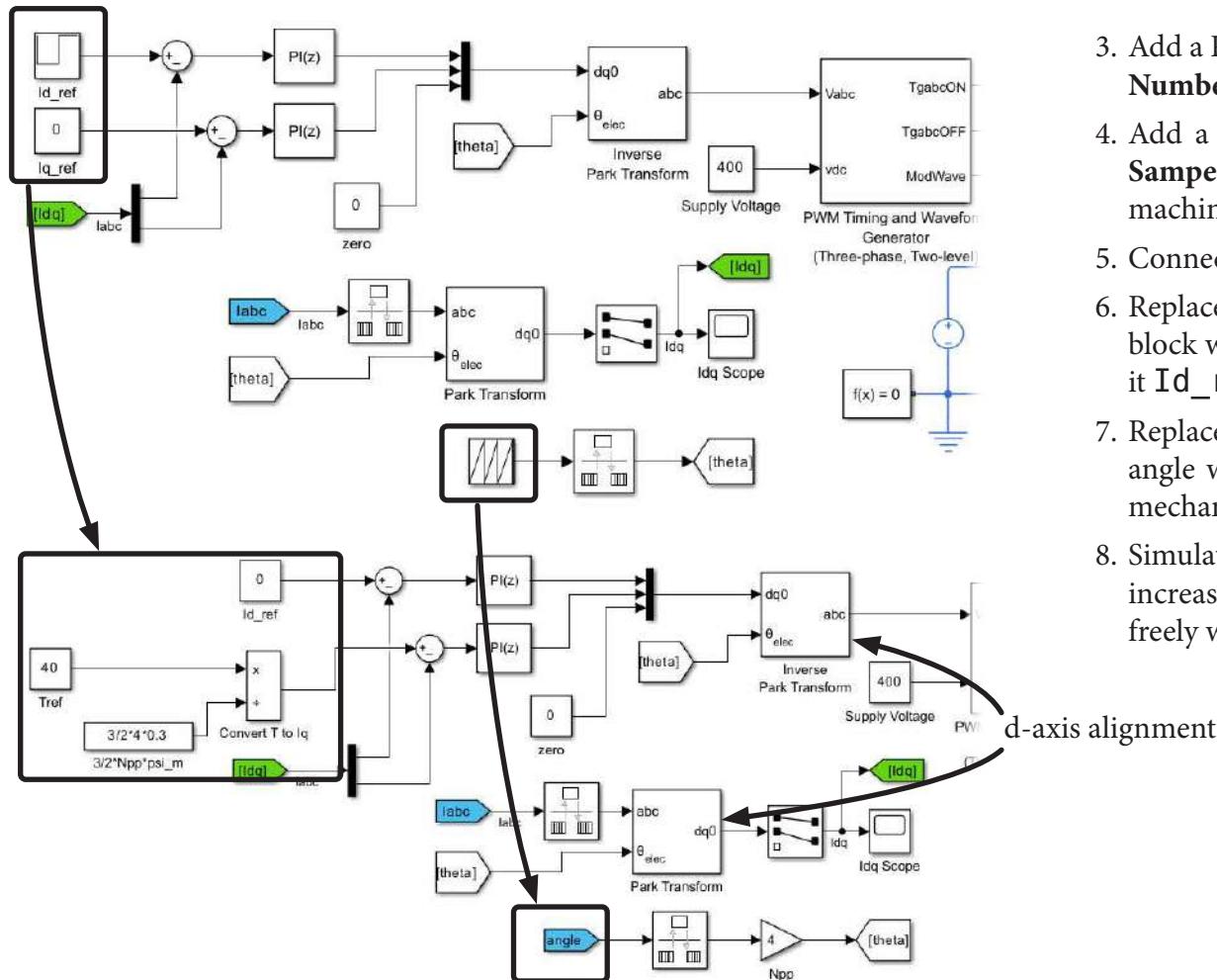


Motor Control

Modeling a Torque Controller (Continued)

To convert the current control to a torque control scheme follow the steps below:

1. For both Park transforms, select **D-axis** for the **Phase-a axis alignment**.
2. Disconnect the Constant block **Iq_ref**, rename it **T_ref**, and set its **Constant value** to **40**.



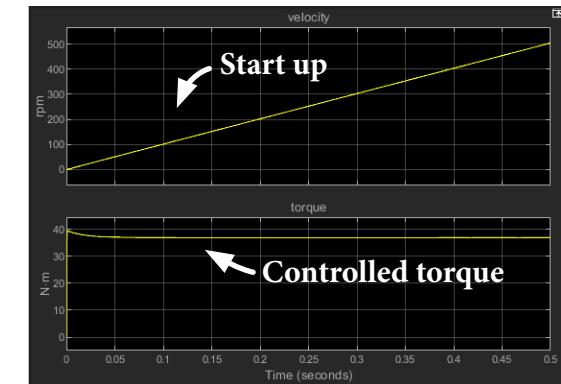
Try

Follow the steps on this page to convert the current control to a torque control implementing a Field Oriented Control scheme.

Simulate the model and inspect the Motor Scope. Does the PMSM now start up and spin?

```
>> pmsm3_control
```

3. Add a Product block (**Simulink > Math Operations library**) and set the **Number of inputs** to *****.
4. Add a Constant block, rename it as **$3/2 * Npp * \psi_i_m$** , and set the **Sampe time** to **$1e-4$** and the **Constant value** to **$3/2 * 4 * 0.3$** (the machine has 4 pole pairs and permanent magnet flux linkage of 0.3 Wb).
5. Connect these blocks to implement the relationship on page 8-6.
6. Replace the Repeating Sequence Stair block **Id_ref** with a Constant block with **Sampe time** to **$1e-4$** and the **Constant value** to **0**. Rename it **Id_ref**.
7. Replace the Repeating Sequence block used to create a linearly increasing angle with a From block with **Goto tag** set to **angle** to retrieve the mechanical rotor position measurement.
8. Simulate the model to see that the PMSM now starts from rest and increases speed while producing a constant torque. The rotor accelerates freely with no load.



Motor Control

Measuring Torque With a Dynamometer

The power, torque, and rotor velocity of the PMSM are related with the following equation:

$$P = T \dot{\theta}$$

To characterize the power of the motor under test, you can model a dynamometer that fixes the rotor speed of the PMSM. This provides a load to the motor so that you can verify the requested torque at a given speed. To do this,

1. Add an Ideal Angular Velocity Source block (**Simscape > Foundation Library > Mechanical > Mechanical Sources library**) to act as a dynamometer. Connect the R port to the Inertia block. Connect the C port to the Mechanical Rotational Reference block.
2. Add a Ramp block (**Simulink > Sources library**) and a Saturation block (**Simulink > Discontinuities library**) to define a velocity profile for the dynamometer. Connect these blocks together, as shown on the right. Set the Ramp and Saturation block parameters below to define a velocity profile that ramps up to the rated speed of 1250 rpm in 0.04 seconds:
 - In the Ramp block, set the **Slope** parameter to 31250.
 - In the Saturation block, set the **Upper limit** parameter to 1250.
3. Add a Simulink-PS Converter block (**Simscape > Utilities library**) to convert the velocity profile Simulink signal into a physical signal. In the **Units** tab, set the **Input signal unit** parameter to **rpm**. In the **Input Handling** tab, set the **Filtering and derivatives** parameter to **Filter input, derivatives calculated**. This is necessary to provide derivative values to the implicit **ode23t** solver.

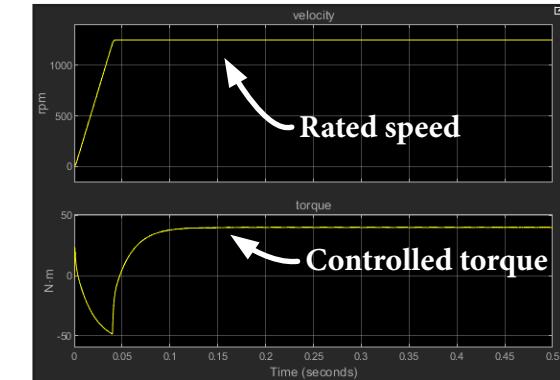
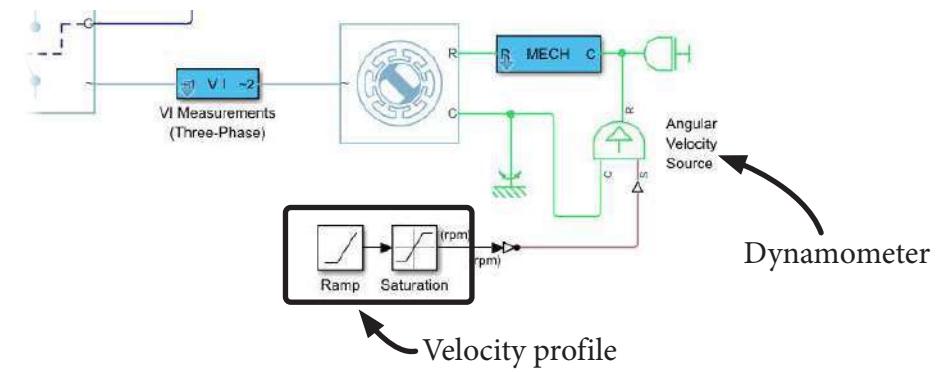
Try

Follow the steps on this page to add a dynamometer to your model that ramps the motor up to its rated speed of 1250 rpm.

Simulate the model and check whether the motor can generate 40 N*m of torque at 1250 rpm.

>> pmsm4_control

The dynamometer will start the motor from 0 to 1250 rpm over the first 0.04 seconds of simulation. The torque controller works to ensure that the requested torque is produced by the PMSM.



Motor Control

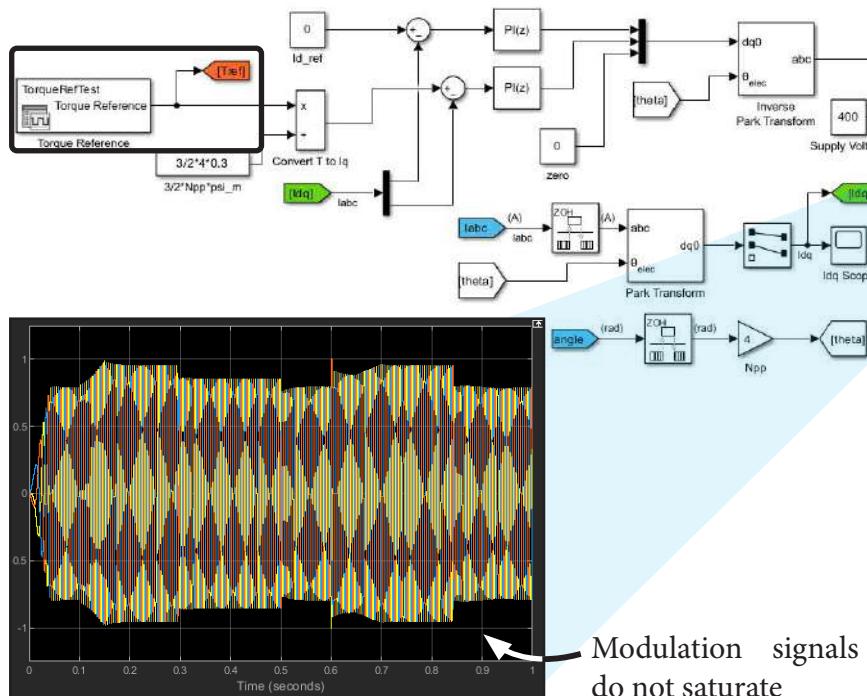
Verifying the PMSM and Controller

Recall that the HEV application requires a controlled PMSM design that:

- Is capable of sustaining 60 kW of mechanical power.
- Has a base speed of 1250 rpm.
- Can follow a torque command.

The dynamometer is already configured to operate the PMSM at the base speed of 1250 rpm. Make the following edits to the model.

1. Replace the **T_ref** block with a Signal Editor block (**Simulink > Sources** library). Pick the **torqueSignal.mat** file as **file name**. Set the Sample time to **1e-4** and check the **Interpolate data** box. The mat-file mimics a HEV test drive cycle.
2. Make the edits at the output of the Signal Editor block and at the second input of the Motor Scope as shown in the figure below.



Try

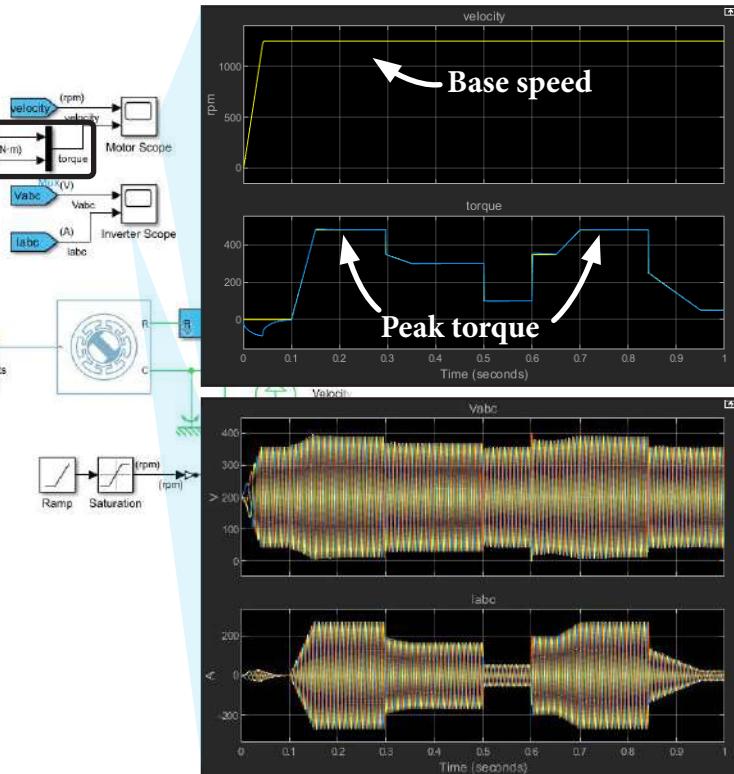
Verify that the controlled PMSM design satisfies the two conditions listed on this page.

>> pmsm5_control

When the torque reference signal is equal to 480 N*m at the base speed, the corresponding mechanical power is 60 kW. This may be found using the power-torque-speed equation on the previous page.

At the rated speed and peak torque, you should verify that

- The PMSM can sustain a torque of 460 N*m at 1250 rpm (60 kW).
- At 60 kW, the modulation waveforms are not saturating.
- The reference torque profile is well-tracked.



Summary

- Modeling a PMSM motor
- Understanding the Field-Oriented Control
- Implementing motor control
- Verifying the torque control design

Motor Control

Motor Control

Test Your Knowledge

1. (T/F) You cannot model nonlinear motor effects, such as magnetic saturation, in Simscape Electrical.

2. (Select all that apply) While implementing a torque control for a PMSM, you should

- A. Feed the mechanical rotor position directly to the direct and inverse Park transforms of the torque controller.
- B. For the direct and inverse Park transforms, select **Q-axis** for the **Phase-a axis alignment**.
- C. Employ a simple torque-current algebraic conversion and feed the torque reference command to the controller q-axis component.
- D. Use the same PI gains you may have tuned earlier for an inverter with a three-phase RL load equivalent to the stator coils' resistance and inductance.
- E. Simplify the control design for three-phase motors with an implementation in the abc reference frame.

3. (T/F) When verifying your torque control design for a PMSM, you should verify that the torque controller

- maintains the full loading condition, that is when the motor spins with the base speed at the peak torque, and
- well tracks a given torque reference profile.

Answers

1. F
2. C, E
3. T



Battery Modeling and Algorithm Development with
Simulink®

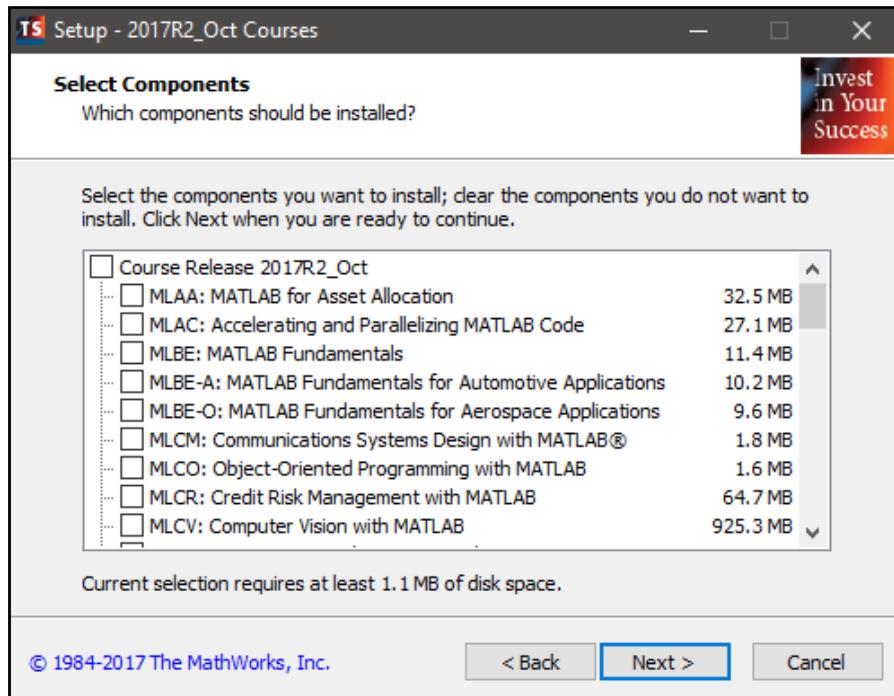
Introduction

Introduction

Computer Setup

To get ready for class, you need to install the examples and exercises from your course DVD or from a USB drive provided by the instructor. Follow these steps:

1. Put the DVD in the DVD drive or plug in the USB drive.
2. The installer application will open automatically. If the installer application does not open automatically, open the DVD drive or USB drive in Windows® Explorer. Run the file **English_20XXRX_MMM.exe**.
3. Follow the prompts in the installer through the installation process. A shortcut will be created on your desktop to start MATLAB for this class.

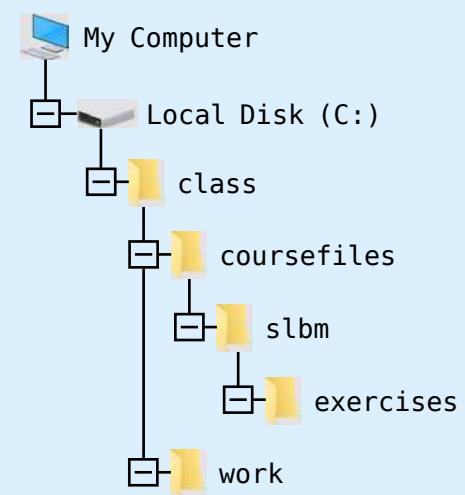


Typical setup

The installation creates a subdirectory of your chosen root directory (the default is C:\class) called **coursefiles**. This directory has subdirectories for each of the courses you install.

Each course directory has a subdirectory called **exercises** that contains all exercises and their solutions.

There is a subdirectory called **work**, which is empty. During class, put all your work in this subdirectory, so that it is on the path and easy to find.



In the installer, you have the following options:

- Choose a class root directory for your course files.
- Choose the courses for which you need to install the files. (Examples and exercises for all of our courses are on the DVD or USB drive.)
- Create a shortcut on the desktop to start MATLAB for this class. (This shortcut runs a **startup.m** file when starting MATLAB, that is customized for the installed course files)

Note A minimalistic install can be performed by navigating to the DVD drive or USB drive in MATLAB, executing the **installer.m** script inside the **zipfiles** folder, and selecting the appropriate courses to install.

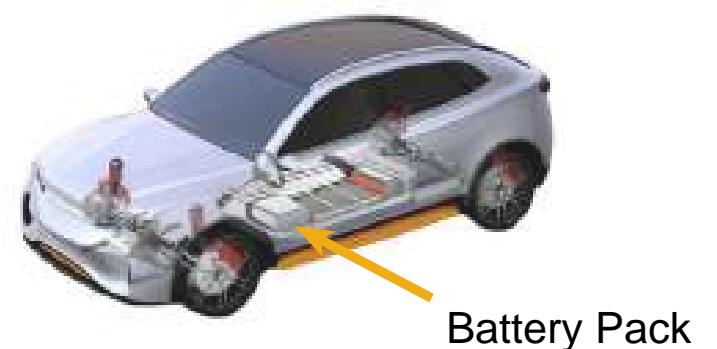
Introduction

Battery Modeling and Algorithm Development

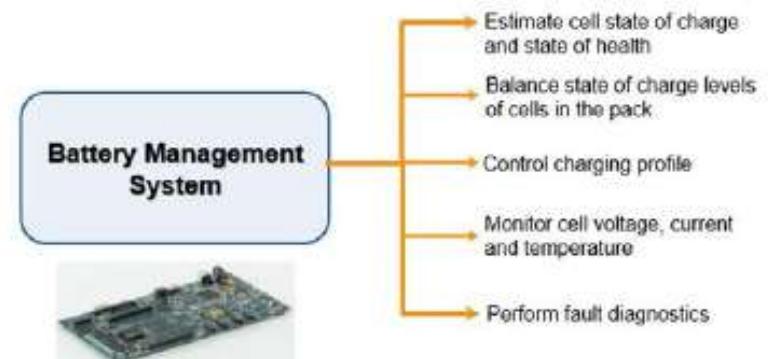
Recent advances in battery technology have facilitated the increase of battery-powered systems for a range of applications, from individual cellular phones to electrical grids and automobiles. Battery engineers are tasked with the development and deployment of efficient control algorithms for the battery management systems that regulate the battery pack operation to derive maximum efficiency out of the pack.

In this course, you will explore how MathWorks tools can aid you in:

- Model a cell unit including thermal and fading characteristics
- Perform cell characterization
- Construct battery packs and attach cooling plates
- Estimate State-Of-Charge and State-Of-Health
- Design the key functionalities of a battery management system including supervisory control, cell balancing, and fault diagnosis



Battery Pack



Battery Management System

Introduction

Course Learning Outcomes

The attendee will be able to

- Perform cell characterization
- Model battery packs in Simulink®
- Add thermal fidelity to battery models
- Design a control scheme for charging the battery
- Perform state estimation and cell balancing
- Compute current limits and design fault diagnostic system
- Perform closed-loop simulation of battery module and battery management system

Introduction

Course Outline

Day 1

- Introduction
- Getting started with a battery cell
- Cell characterization
- Battery pack modeling

Day 2

- State Estimation
- Battery management system
- Fault monitoring and current limit computation
- Conclusion

Appendices

- Kalman Filter
- Create battery objects by scripting
- Exercises



Battery Modeling and Algorithm Development with
Simulink®

Getting Started with a Battery Cell

Getting Started with a Battery Cell

Outline

- Battery terminologies
- Equivalent Circuit Model
- Charge and discharge a cell
- Modeling cell thermal effects
- Modeling cell degradation

The following names are trademarks of The MathWorks, Inc.:

MATLAB®; Simulink®; Simscape™; Simscape™ Battery™

Getting Started with a Battery Cell

Chapter Learning Outcomes

The attendee will be able to:

- Define terms used in a typical battery component
- Create a battery model using the Battery (Table-Based) block
- Discharge a battery cell at a given C-rate
- Charge a cell using the Constant Current-Constant Voltage method
- Add thermal and fading characteristics to the cell

Getting Started with a Battery Cell

Terms and Definitions

Throughout the course, you will encounter several terms that are used commonly while working on battery models or designing algorithms for the battery management system (BMS)

- Cell nominal capacity
- Cell nominal voltage
- C-rate
- Open-circuit voltage
- Terminal voltage
- State of charge

In this chapter, these terms are introduced and defined.

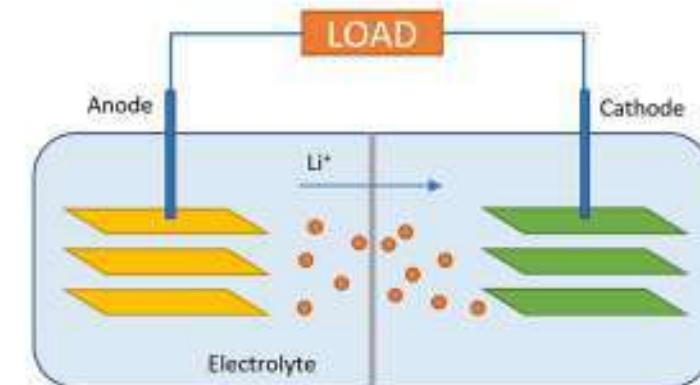
Getting Started with a Battery Cell

Cell

A cell is a single-unit energy source that produces electrical energy from chemical reactions. Typically, a cell contains a cathode(+) and an anode(-) being separated by the separator. They are placed in the electrolyte that assists the movement of the ions. When a cell is connected to an outer circuit (i.e. a resistive load), electrons go through the circuit while ions are transmitted through the electrolyte.

Given the different chemical constituents of a cell, the chemical reactions define the electrode potential. For example, a nickel-based cell gives a voltage of around 1.2 V, while a lithium-based cell provides an average voltage of around 3.6-3.7 V. In this course, the cell used is assumed to have lithium-based chemistry unless specified otherwise.

A battery is made up of cells connected in series or parallel configurations. The terms are often used interchangeably, but this course will use them distinctly: a cell is a singular source of energy, whereas a battery is a component made up of several such individual cells.



Getting Started with a Battery Cell

Nominal Cell Capacity and Voltage

- Nominal Cell Capacity:**

Cell capacity specifies the amount of charge a cell can hold. It is normally expressed with a unit of Ampere-hour (Ah). For example, a cell capacity of 250 mAh implies that the cell can supply a current of 250 mA for one hour. Typically, you can find the nominal cell capacity provided by the manufacturer on the cell packaging. It indicates the average capacity you can get from a fully charged cell under specified load conditions and temperature.

- Nominal Cell Voltage:**

The nominal cell voltage is the average voltage that can be obtained from the cell. The voltage provided by the cell is higher than the nominal voltage when fully charged, and lower than the nominal voltage when fully discharged.

Try

What is the nominal capacity and voltage in the lithium-ion cells shown below?

Combining the nominal cell capacity and voltage, you can generally gauge the average energy a cell can provide. But do note that these values are measured under nominal conditions. For example, a cell might not provide the same amount of energy when the temperature is too low. It is important to consider the environment and the load conditions when sizing the system.



Getting Started with a Battery Cell

C-rate

- C-rate**

The C-rate is computed as the ratio of charge or discharge current to the rated capacity of the cell.

$$\text{C-rate} = \frac{I}{q_{nom}}$$

Where

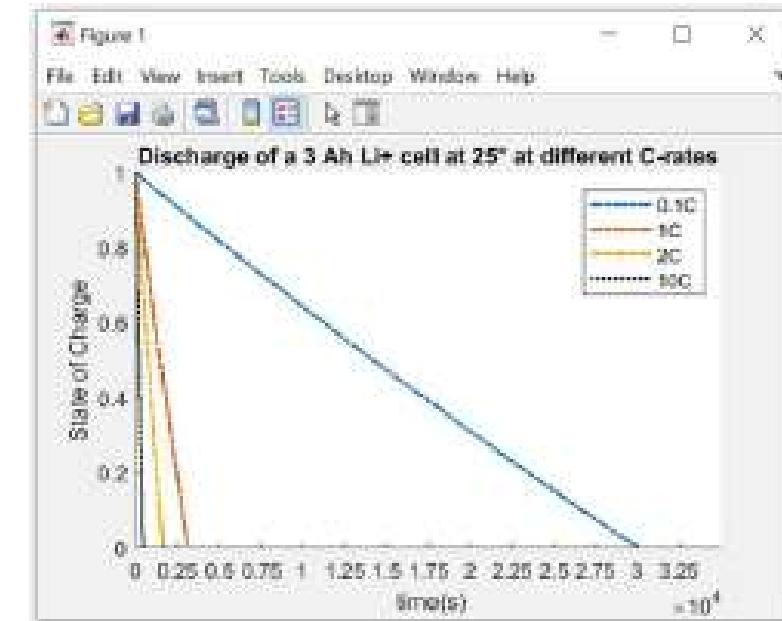
I – current going through the cell

q_{nom} – nominal cell capacity in Ah

If a cell is rated to have 2 Ah capacity and it is charged using a current of 2 A, then the cell is said to have charged at 1 C rate. If the same cell is discharged at 0.5 C rate, it implies that the cell is providing a current of 1 A for a duration of two hours to the load.

Try

What's the discharge current when discharging a 10 Ah cell with 0.1C?



Getting Started with a Battery Cell

State of Charge

- State of Charge

The State of Charge (SOC) of a cell is a virtual gauge indicating the amount of capacity remaining in the cell. SOC is defined as the ratio of current charge to nominal cell capacity.

$$SoC = \frac{q(t)}{q_{nom}}$$

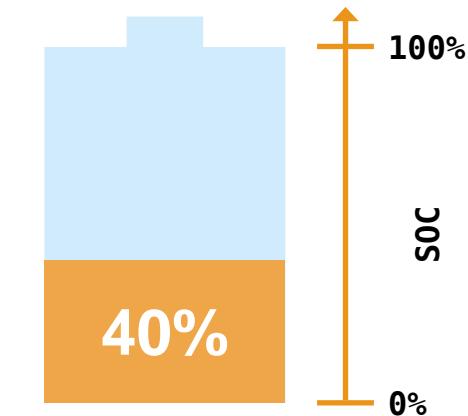
where

$q(t)$ – present cell charge

q_{nom} – nominal cell capacity in Ampere-seconds

SOC is a unitless value. It varies from 0 to 1 or 0% to 100%, which represent fully discharged and fully charged states respectively.

Ideally, the SOC of a cell can be calculated by measuring the concentration of the ions at one electrode. But it is hard to implement outside the lab. In reality, SOC is estimated based on measurable values such as the terminal voltage and current. We are going to discuss SOC estimation in the State Estimation chapter.



Getting Started with a Battery Cell

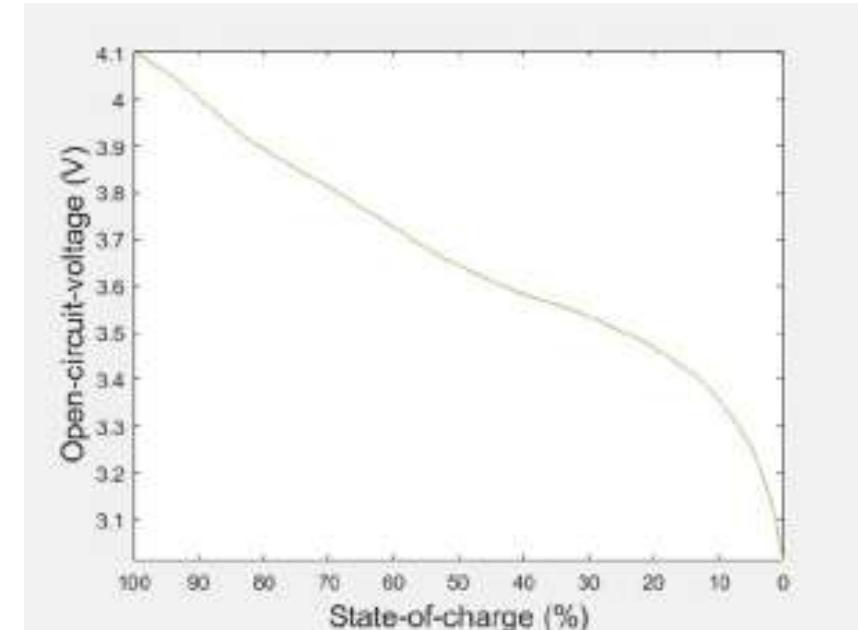
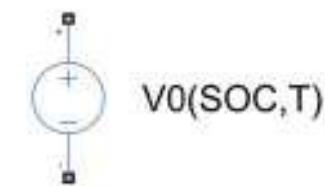
Equivalent Circuit Model, Open-Circuit Voltage, and Terminal Voltage

To accurately estimate the states of a cell, you need to have a good plant model that closely approximates the real cell behaviors. There are multiple ways to model the dynamics of a cell, such as modeling the electrochemistry of the cell. In this training, you are going to use the Equivalent Circuit Model (ECM) to model the cell. The ECM represents the cell's input/output relationship by using electric circuits. Compared with other methods, ECM has a good balance between simulation speed and accuracy which is more suitable for industrial applications.

- **Open Circuit Voltage**

A cell can be modeled as simply as a voltage source, where the voltage is a function of SOC and temperature. Typically, when a cell is not connected to any load, the voltage of a battery is a fixed value at a specific SOC and temperature. This voltage is recognized as the Open Circuit Voltage (OCV) and its value is often provided by the manufacturer. On the right is the SOC-OCV curve of a LiNiMnCo (NMC) cell.

Open
Circuit
Voltage



Getting Started with a Battery Cell

Equivalent Circuit Model, Open-Circuit Voltage, and Terminal Voltage (Continued)

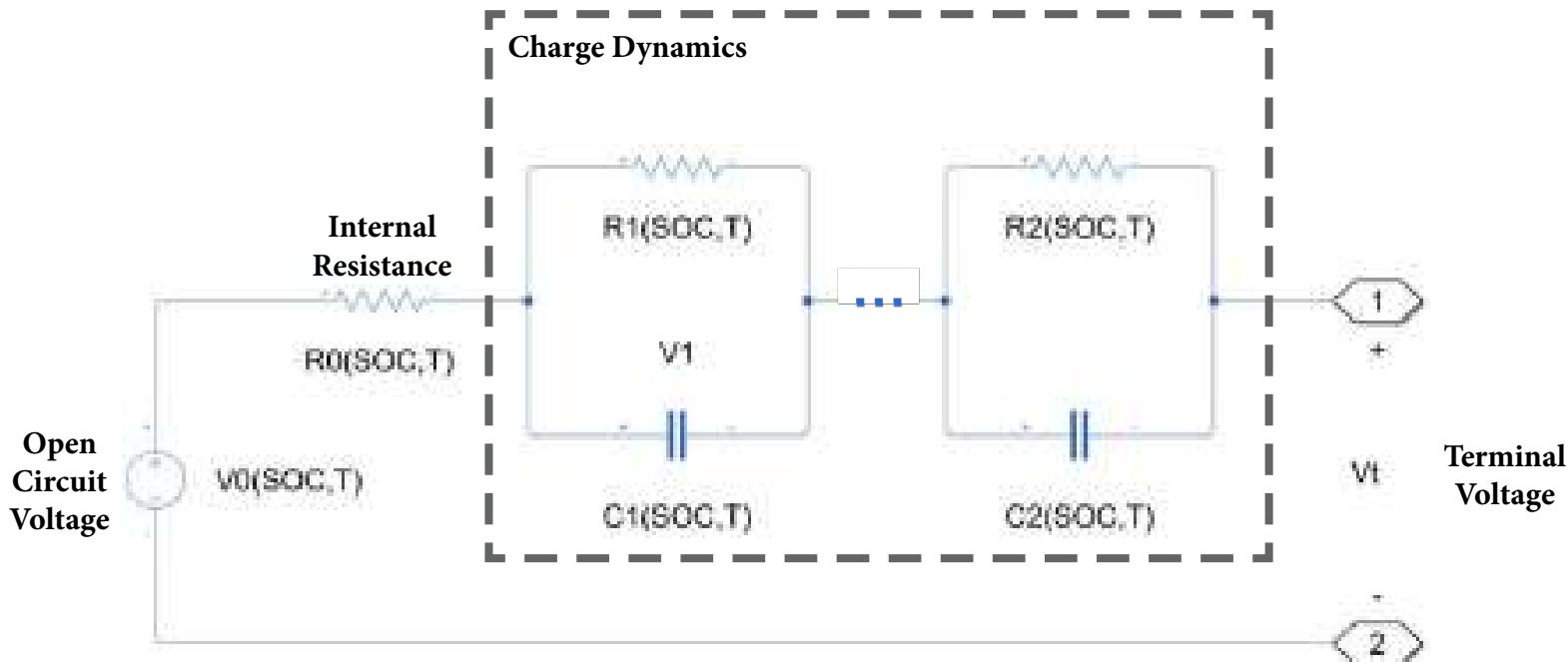
- Terminal Voltage

The terminal voltage of a battery is the measurable voltage difference between the two electrodes. When a cell is not connected to a load, the terminal voltage of the battery equals OCV. However, when the cell is in use, the measurable terminal voltage is subject to the cell's internal resistance and the diffusion dynamics of the ions.

- Equivalent Circuit Model

To model the cell behavior with more detail, below is the Thevenin ECM of a cell. The series resistor R_0 represents the Ohmic loss when current flows through the cell. The series of RC pairs mimic the diffusion of the ions within the cell. The modeler must decide the number of RC branches by considering the tradeoff between model accuracy and model complexity.

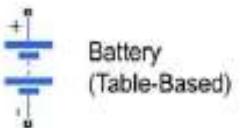
Note that the open-circuit voltage, V_0 , the ohmic resistance, R_0 , and the RC branches parameters are dependent on cell SOC and temperature.



Getting Started with a Battery Cell

Battery (Table-Based) Block

In Simulink, the Battery (Table-based) block from the Simscape > Battery > Cells library represents the high-fidelity equivalent circuit model described previously. The block calculates parameters (V_o , R_o , etc.) as a function of SOC and optional temperature using lookup tables.



Under the **Main** tab, you can use the **Tabulate parameters over temperature** setting to decide whether to tabulate parameters over temperatures or not. When the selection is **No**, V_o , R_o are 1D vectors with the same number of elements as the SOC vector. When the selection is **Yes**, V_o , R_o are 2D vectors with the same dimensions of (SOC, T).

> Vector of state-of-charge values, SOC	[0, .1, 25, 5, .75, 9, 1]
Tabulate parameters over temperature	<input checked="" type="checkbox"/> Yes
Current directionality	Disabled
> Vector of temperatures, T	[278, 293, 313] K
> Open-circuit voltage, V0(SOC,T)	[3.49, 3.5, 3.51; 3.55, 3.57, 3.56...] V

V0(V)	5°C	20°C	50°C
0%	3.49	3.50	3.51
50%	3.71	3.71	3.72
100%	4.19	4.19	4.19

Try

>> cellDischarge01_start

Add a Battery (Table-Block) from the Simscape > Battery > Cells library into the model.

You can select the charge dynamics topology under the **Dynamics** tab. To use one RC branch for charge dynamics, select **One time-constant dynamics** as **Charge dynamics**.

Charge dynamics	<input checked="" type="checkbox"/> One time-constant dynamics
> First polarization resistance, R1(SOC,T)	[.0109, .0029, .0013; .0069, .00...] Ohm
> First time constant, tau1(SOC,T)	[20, 36, 39; 31, 45, 39; 109, 10...] s

The initial conditions of the battery can be set under the **Initial Targets** tab. By default, the initial cell SOC is set to 1. You can override the **Value** to start at a different SOC.

Current (positive in)	<input type="checkbox"/>
Terminal voltage	<input type="checkbox"/>
State of charge	<input checked="" type="checkbox"/>
Priority	High
Value	1

Getting Started with a Battery Cell

Block Parameterization Manager

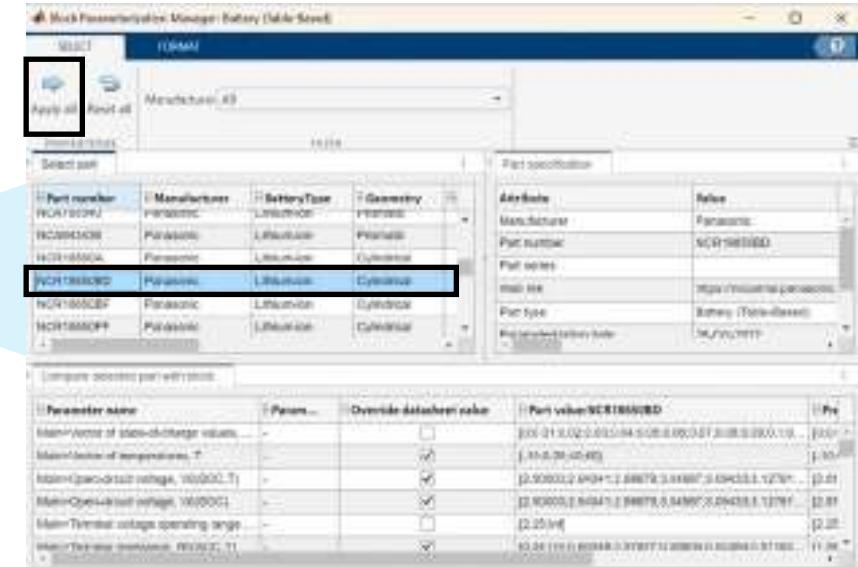
To ensure the Battery (Table-Based) block can accurately represent the real battery, it is crucial to provide accurate parameters. The upcoming chapter will introduce you to the parameter estimation workflow, which involves collecting experimental data and performing parameter estimation. This process can be time-consuming, as battery experiments often span days to complete. Alternatively, when you don't have battery parameters ready, you have the option to utilize pre-parameterized battery data available through the **Block Parameterization Manager**.

1. Click on the hyperlink under the **Selected part** to open the **Block Parameterization Manager**.
2. Select NCR18650BD from the **Select part** table.
3. Click **Apply all** to update the Battery (Table-Based) block parameter.

Try

Follow the steps to update the cell parameters using **Block Parameterization Manager**.

These parameters are collected from manufacturer data sheets. After selecting a part, you can find details about the parameters under Compare selected part with block. If the Override datasheet value is checked for a field, the corresponding Part value will override the block value after Apply all. For Battery (Table-Based) block, the provided parameters are Open circuit voltage V_0 , Terminal resistance R_0 , Cell capacity AH, and temperature independent fading table for open-circuit voltage and cell capacity.



Getting Started with a Battery Cell

Create a Discharge Circuit with Simscape

Next, you are going to connect the battery with a resistive load to perform a simple discharge.

1. Add a Resistor block from the **Simscape > Foundation Library > Electrical > Electrical Elements** library. Set the **Resistance** to 1 Ohm.
2. Turn the Resistor to vertical by pressing **Ctrl+R**. Connect the Battery (Table-Block) and the Resistor together to form a circuit.
3. To monitor the voltage across the load, add a Voltage Sensor block from **Simscape > Foundation Library > Electrical > Electrical Sensors** library. Connect the + and – terminals of the Voltage Sensor across the resistor.

In Simscape, the line connected between blocks represents the physical connections which is different from Simulink. You cannot directly connect a Simscape signal to a Simulink block. Instead, you need to convert the Simscape signal to a Simulink signal.

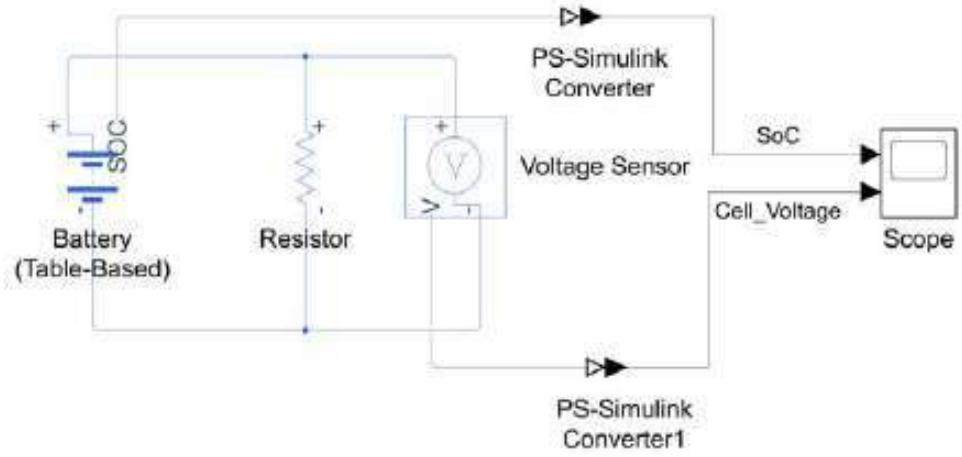
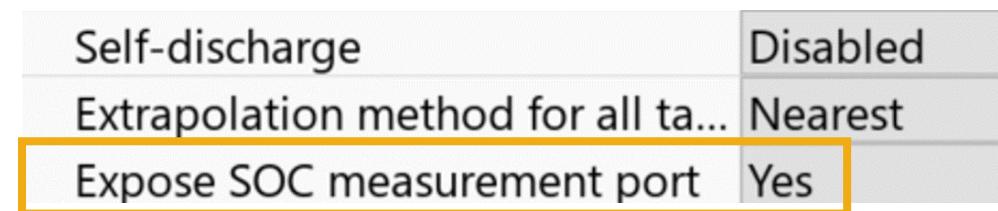
4. Add a PS-Simulink Converter block from the **Simscape > Utilities** library.
5. It is recommended that you specify the unit of a signal when converting. Set the **Output signal unit** of the PS-Simulink Converter to V.
6. Connect the V port from the Voltage Sensor block to the PS-Simulink Converter block, and the output to the scope.

The Battery (Table-based) block also provides an instrument to monitor the cell SOC. To enable this functionality:

7. Change **Expose SOC measurement port** to Yes in the **Block Parameters** dialog.
8. Connect the SOC port to a PS-Simulink Converter, and the output to the scope.

Try

Follow the steps to build the discharge circuit.



PS-Simulink Converter	
NAME	VALUE
Vector format	inherit
Output signal unit	V

Getting Started with a Battery Cell

Battery (Table-Based) – Discharging

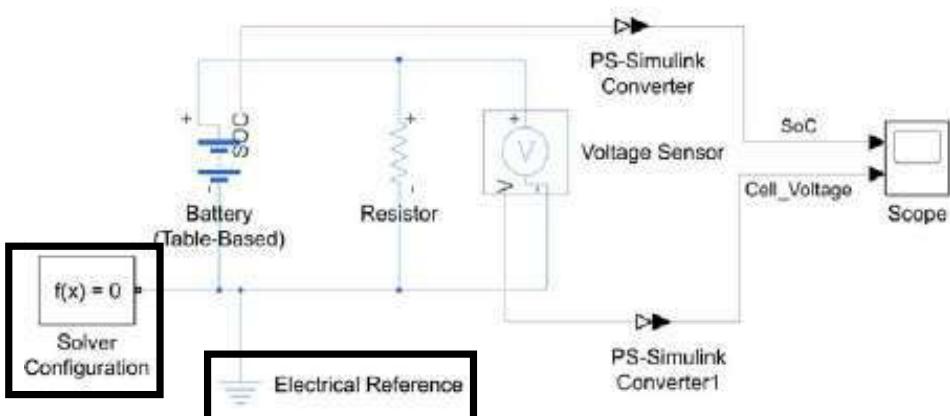
Other than the blocks building up the circuit, Simscape requires reference and utility blocks to facilitate the simulation.

For each Simscape domain, at least one reference block is required to establish a reference level for physical quantities. For example, if the discharge circuit is in the electrical domain, an Electrical Reference block should be added.

9. Connect an Electric Reference block from the **Simscape > Foundation Library > Electrical > Electrical Elements** library to the network.

Simscape requires a Solver Configuration block for each Simscape network. The block provides additional solver options and settings that are required before starting a simulation. The Solver Configuration block can be connected anywhere in the Simscape network.

10. Add a Solver Configuration block from the **Simscape > Utilities** library. Connect the block to a point in the circuit.



Try

```
>> cellDischarge02_soln
```

Normally there's no need to change the settings in the Solver Configuration block. By default, the Simscape diagram is solved by the global solver selected in the model's Configuration Parameters. Optionally you can select to use a local solver for the Simscape network separately from the rest of the Simulink model. In cases such as hardware-in-the-loop (HIL) where the simulation speed is a concern, using a local solver could potentially free other parts of the Simulink model to run faster (i.e., use a discrete global solver when there's no continuous state in the rest of the model).

11. Simulate the model. The cell slowly discharges through the resistor. Cell SOC gradually reduces from 1 to 0 and the cell voltage drops accordingly.

Note that if you extend the simulation time to 3600s, the cell will become over-discharged and you will get a warning message stating that “State of charge must be greater than or equal to zero”. When SOC is out of range, the voltage value will either hold the nearest or extrapolate linearly depending on your settings in the **Extrapolation method for all tables** in the **Block Parameters**. It is the responsibility of the Battery Management System (BMS) to limit the battery discharge within a minimum voltage cut-off range to avoid damaging the cell. You will learn to build a BMS later in the course.

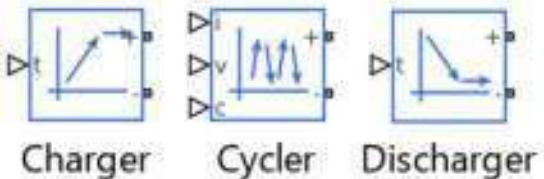
Getting Started with a Battery Cell

Constant Current – Constant Voltage (CC-CV) Charging

Constant Current – Constant Voltage (CC-CV) charging is one of the commonly used methods to charge a lithium-based cell. It is used to prevent the battery from being overcharged while charging it to its maximum capacity.

In the CC-CV charging scheme, the cell is charged using a constant current first until its terminal voltage reaches a predetermined maximum voltage. However, the OCV hasn't reached the maximum voltage yet since there are current-dependent voltage drops across the resistors. To further charge the cell while avoiding overvoltage, the charger switches to constant voltage mode. As OCV continues to increase, less current is required to keep the terminal voltage constant. When the charging current drops around zero, the battery is fully charged.

Simscape Battery simplifies the process of building charging, discharging, and cycling circuits by offering readily available blocks. These blocks are located under **Simscape > Battery > Cyclers** library.



You will use the Charger block to implement the CC-CV logic.

1. Open model `cellCharge01_start.slx`
2. Add a Charger block and connect it with the circuit accordingly.
3. Add a PS-Constant block with a value of 1 and connect it to the **t** port of the Charger.
4. Set up the **Block Parameters** of the Charger as shown in the figure.

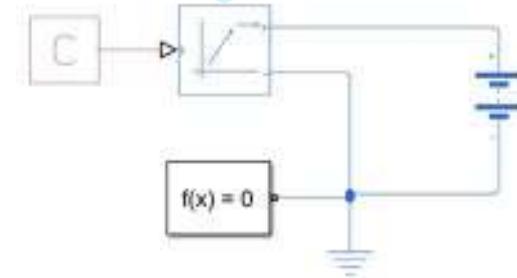
Try

Follow the steps to charge the battery using CC-CV logic.

`>> cellCharge01_start`

Parameters

Constant charging current	27	A
Voltage threshold	4.08	V
Constant charging voltage	4.08	V
Current threshold	CC_current/30	0.9 A

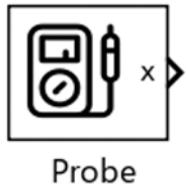


Field **Constant charging current** denotes the current used in the CC phase. The **Voltage threshold** indicates the voltage to switch from CC to CV. Then the charger keeps the voltage constant at **Constant charging voltage**. In the CV phase, the required current to keep the same constant voltage will gradually decrease. When it drops below the **Current threshold**, charging is considered completed and all current is cut off from the circuit. Typically a **Current threshold** of C/30 is considered small enough to end the CV charging.

Getting Started with a Battery Cell

Probing Signals

There are multiple ways to collect measurements from a Simscape diagram. In the previous exercise, you explicitly added voltage measurement to the circuit. As an alternative way, the Probe block from the **Simscape > Utilities** library lets you select variables from another block in the model and output the measurements as Simulink signals.



Probe

1. Add a Probe block from the **Simscape > Utilities** library.
2. Double-click on the Probe block to enable selection. Then single-click on the Battery (Table-Based) block.
3. In the drop-down menu, select **cell temperature**, **i**, **SOC**, and **v** as shown in the figure.
4. Connect the Probe to the Scope.
5. Simulate the model.



After selecting a block, the Probe context menu contains all the externally accessible variables of the selected block. Note that before 23a, only the variables exposed in the Initial Targets section of the Block Parameters are available for probing.

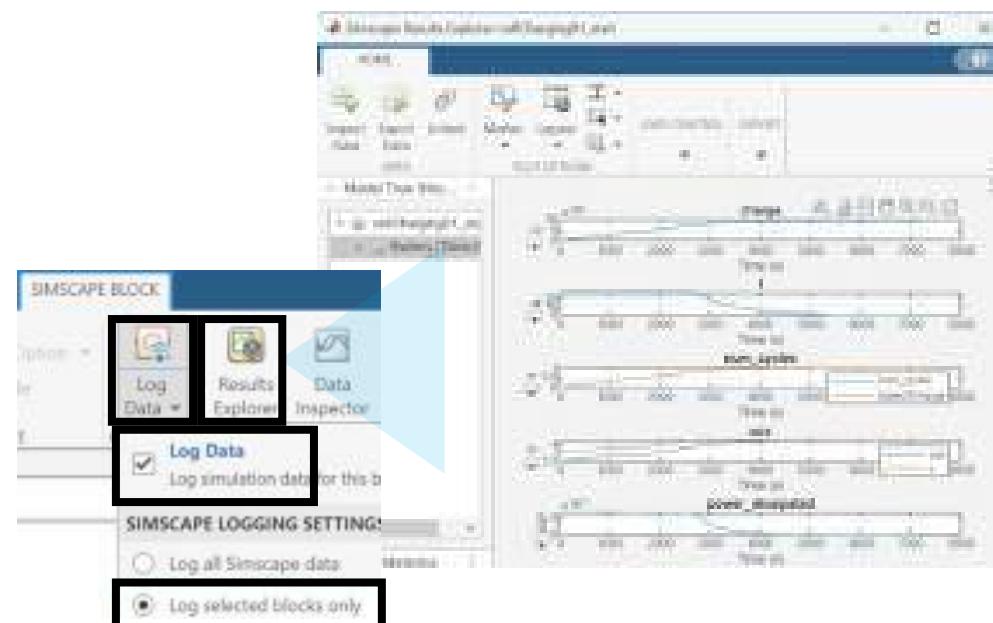
Try

Follow the steps to add a Probe block to the model.

```
>> cellCharge02_probe
```

Lastly, **Simscape Results Explorer** offers a third way of probing Simscape signals without adding blocks to the model.

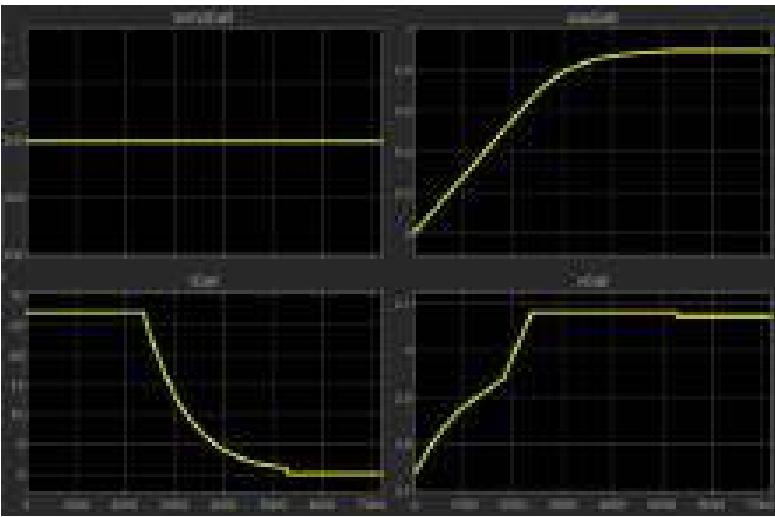
1. Select the Battery (Table-Based) block and go to the **SIMSCAPE BLOCK** tab of the block.
2. Under **Log Data**, check the checkbox of **Log Data** and select **Log selected blocks only**.
3. Simulate the model. Click on the **Results Explorer** to view the results.



Getting Started with a Battery Cell

Modeling Thermal Effect

In the CC phase, the charging current is held and voltage starts to increase. During this time, the SOC increases linearly. Once the terminal voltage hits the threshold, the charging switches over to CV mode. The voltage is kept as a constant and the current gradually decreases to the current threshold.



Note that the following convention is applied for the current:

- A positive indicates charging.
- A negative indicates discharging.

The simulation temperature of the cell is held constant currently. However, due to the internal resistance of the battery, heat is generated when current flows through the cell. The battery temperature is determined by the summation of all the ohmic losses included in the cell model.

$$M_{th}\dot{T} = \sum_i \frac{V_{T,i}^2}{R_{T,i}}$$

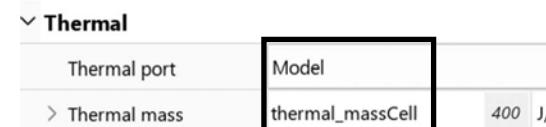
Try

Enable the **Thermal port** of the Battery (Table-Based) block and set the **Thermal mass** parameter.

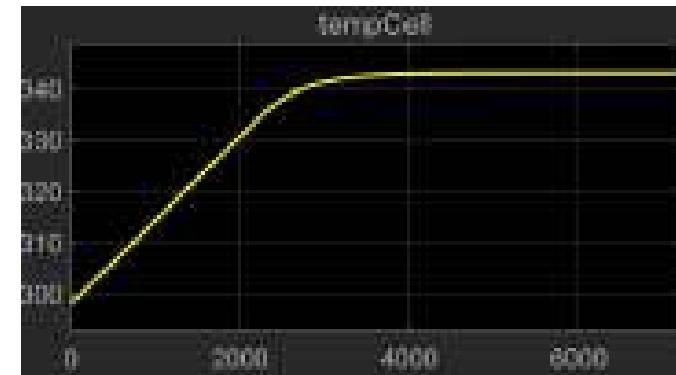
M_{th} is the thermal mass of the battery. It equals the mass of the cell times the specific heat capacity of the cell. Specific heat capacity has the unit of J/kgK. It describes how much heat is needed for one unit of mass to increase the temperature for one unit. The specific heat capacity of the cell should be determined by thermal experiment.

To model and measure the cell temperature changes, you need to enable the thermal port of the Battery(Table-based) block.

1. Go under the **Thermal** tab of the **Block Parameters**. Set the **Thermal port** to **Model**.
2. Change the Thermal Mass of the battery to **thermal_massCell**.
3. Simulate the model.



By enabling the thermal model, cell temperature continuously rises during charging. The temperature rise is proportional to the charging current.



Getting Started with a Battery Cell

Modeling Thermal Effect (Continue)

You can further connect the **Thermal** port of the Battery (Table-Based) block with additional thermal circuits to model the cell heat exchange with the ambient environment. Assume a simple case where the ambient temperature is the average air temperature surrounding the cell. The battery exchanges heat with the ambient environment through convection represented in the formula.

$$Q = kA(T_A - T_B)$$

T_A and T_B represent the temperature of the battery and the ambient environment. A is the surface area of the battery, Q is the heat flow, k is the heat transfer coefficient, and the unit of k is $\text{W}/\text{m}^2\text{K}$. Typically, the convection heat transfer coefficient for air is in the range of 2.5-25.

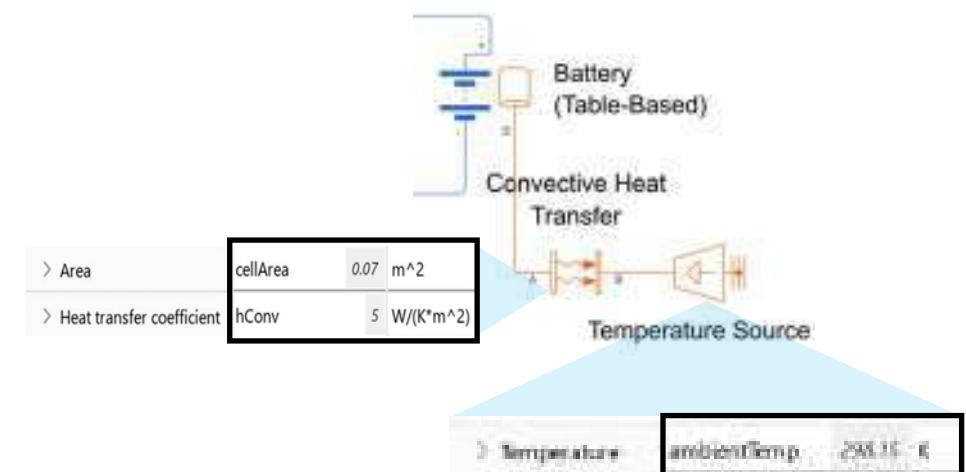
In Simulink, you can use a Temperature Source block to represent the ambient temperature and a Convective Heat Transfer block to model the heat exchange.

1. Add a Temperature Source block from the **Simscape > Foundation Library > Thermal > Thermal Sources** library into the model.
2. Add a Convective Heat Transfer block from the **Simscape > Foundation Library > Thermal > Thermal Elements** library to the model.
3. Connect the **Convective Heat Transfer** block between the **Temperature Source** block and the thermal port of the battery. Note the direction of the **Convective Heat Transfer** block does not impact the result.
4. Set the parameters of the two blocks as shown in the figure.
5. Simulate the model.

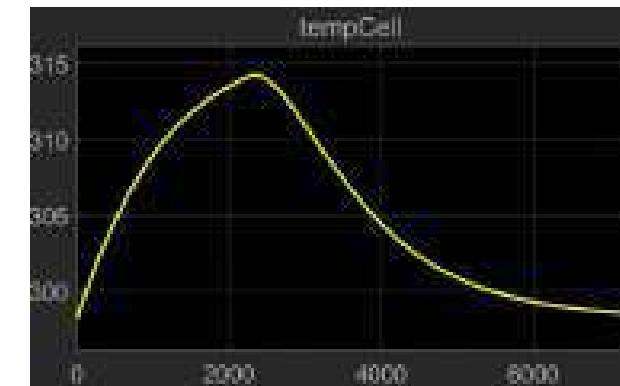
Try

Follow the steps to add a Probe block to the model.

>> cellCharge03_thermal



By modeling the ambient cooling circuit, the highest cell temperature decreases. Cell temperature still rises during the CC phase when more heat is generated compared to what can be dissipated. Less heat is generated during the CV phase compared to ambient cooling, cell temperature gradually winds down.



Getting Started with a Battery Cell

Modeling Battery Degradation

All batteries have a limited lifespan. A battery degrades during use and storage due to irreversible side reactions, such as the growth of the new solid-electrolyte interface layer, li-ion plating, etc. As a consequence, the performance of the battery continues to decline and reaches a point where it needs to be replaced. Typically, if a battery's capacity fades to 80% of its original value, it reaches the End of Life (EOL).

In the Battery (Table-Based) block, you can specify the battery fade parameters using the **Fade** tab either by using equations or look-up table. Follow the steps below to model a rapid cell fading by using the equation-based method. The cell capacity is set to fade to 80% of its nominal capacity after 10 cycles.

1. Open model `cellFade01_start.slx`.
2. In the **Block Parameters** of the Battery (Table-Based) block, under the **Fade** tab, change the **Fade characteristic** to **Equations**.
3. Change the **Number of discharge cycles, N** to 10.
4. Change the **Change in cell capacity after N discharge cycles (%)** to -20.
5. Simulate the model.

The equation used to represent battery fading is derived from empirical data, indicating that the change in capacity follows a square root relationship with respect to the discharge cycles.

$$q_{fade} = q_{nom} \left(1 + \frac{\delta}{100} \sqrt{\frac{n}{N}} \right)$$

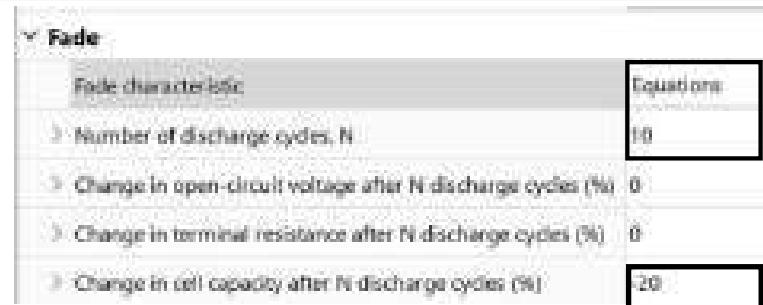
Where

- q_{fade} is the faded capacity
- q_{nom} is the nominal capacity at the beginning of life
- δ is the percent change in the capacity over N cycles
- n is the current count of the discharge cycle

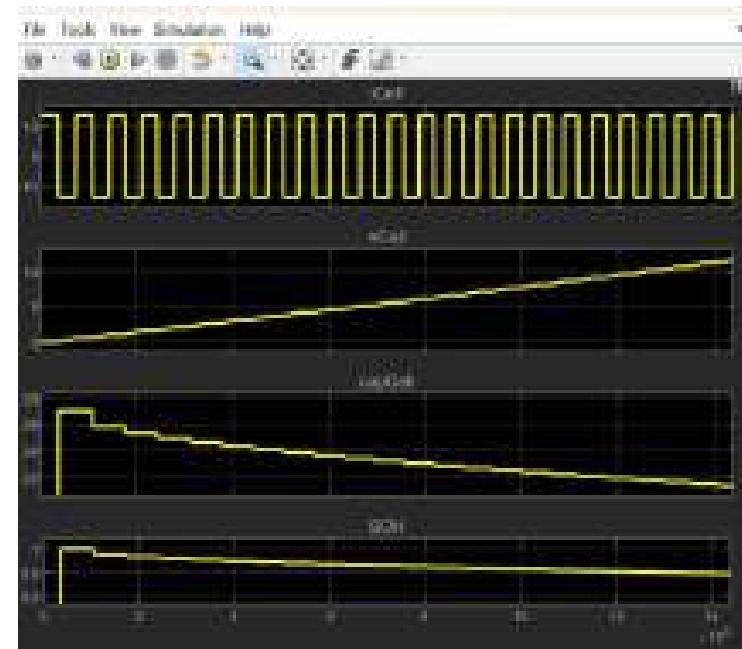
Try

Follow the steps to add battery fading to the model.

`>> cellFade01_start`



Note that the battery block does not provide capacity measurement as an output. An external capacity estimation is required to capture the change. In the simulation result, the cell capacity is calculated within the `ChargeDischargeControl` subsystem. You will learn more about capacity estimation in the state estimation chapter.



Getting Started with a Battery Cell

Modeling Battery Degradation (Continued)

To more accurately model the battery fade, you can model the battery fading as lookup tables and provide your data. Lookup tables can either be temperature independent or temperature dependent.

1. In the **Block Parameters** of the Battery (Table-Based) block, under the **Fade** tab, change the **Fade characteristic** to **Lookup tables (temperature dependent)**.
2. Set the fields under the Fade tab as shown below.
3. Simulate the model.

Fade	
Fade characteristic	Lookup tables (temperature dependent)
> Vector of discharge cycle values, N	N0
> Vector of temperatures for fade data, Tfade	Tfade <1x6
> Percentage change in open-circuit voltage, ...	dV0
> Percentage change in terminal resistance, ...	dR0
> Percentage change in cell capacity, dAH(N,...)	dAH
> Percentage change in first polarization resi...	dR1

The battery fading table represents a temperature-dependent cell fading, as depicted in the figure. The percentage of battery fade increases as the cycle count and temperature rise. In practice, it is important to regulate the battery temperature to mitigate the rate of battery fading.

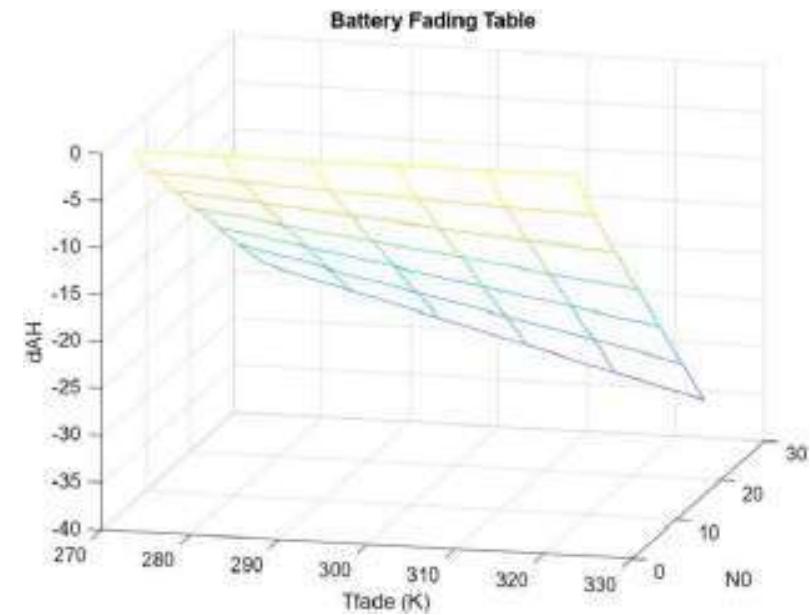
Try

Follow the steps to enable lookup table based fade modeling.

Or,

>> edit cellFadeScript

Run the script and examine the battery capacity fade under different temperatures.



Getting Started with a Battery Cell

Summary

- Battery terminologies
- Equivalent Circuit Model
- Charge and discharge a cell
- Model cell thermal effects
- Model cell degradation

Getting Started with a Battery Cell

Test Your Knowledge

1. Which of the following C rate represents discharging a 20Ah battery with 40A?
 - A. 0.2C
 - B. 0.5C
 - C. 2C
2. (T/F) The unit of SOC is Coulomb or Ampere hour.
3. (T/F) Charging a cell with CC-CV scheme till the end of the CC phase, the measured cell terminal voltage equals to the cell open-circuit voltage.

Getting Started with a Battery Cell

Answers

1. C
2. False
3. False



Battery Modeling and Algorithm Development with
Simulink®

Battery Pack Modeling

Battery Pack Modeling

Outline

- Create battery module and pack objects with the Battery Builder app
- Control the model resolution of battery objects
- Add thermal fidelity to the battery objects
- Model cooling plates attached to the battery objects

The following names are trademarks of The MathWorks, Inc.:

MATLAB®; Simulink®; Simscape™; Simscape™ Battery™

Chapter Learning Outcomes

The attendee will be able to:

- Create battery objects with Simscape Battery
- Model the thermal environment for battery simulation

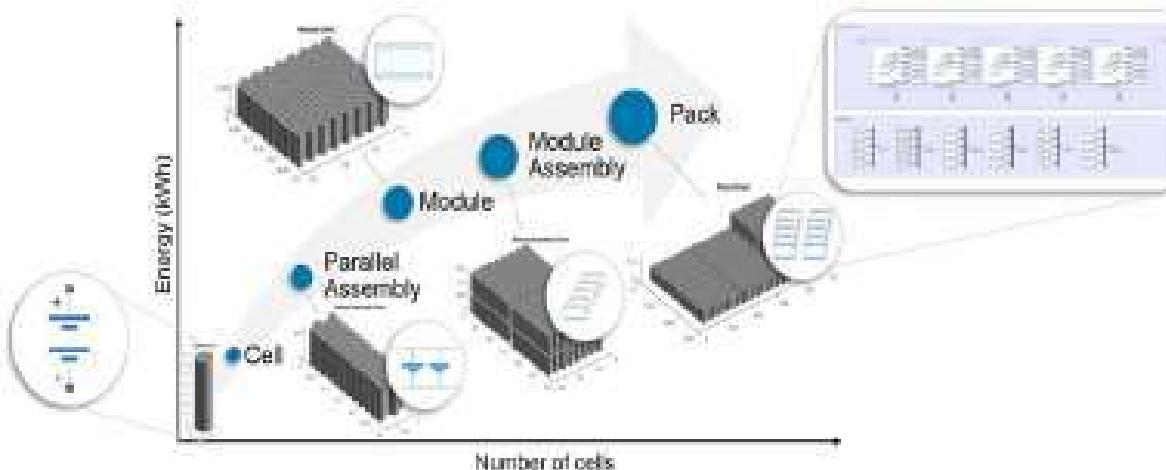
Battery Pack Modeling

Battery Builder

A lithium-ion cell has a nominal voltage of 3.7 V. To address the typical voltage and energy requirements of an electric vehicle (200-800V, 12-100kWh), multiple battery cells need to be connected to increase the voltage and energy output.

A cell module is created by wiring identical cells either in series or parallel combinations to boost the power output. It is then housed within a protective frame to protect the cells from external disturbance. Cell modules can be further combined into a battery pack to meet the energy requirements of a given application.

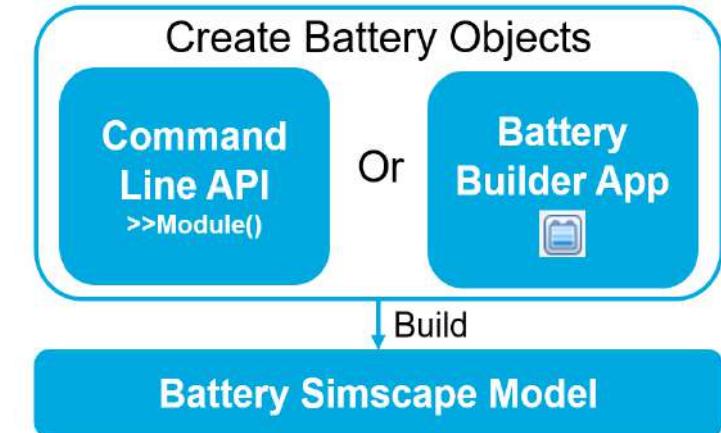
The pack modeling process often requires tedious manual connection of cells and accessories in Simulink. Simscape Battery provides an object-oriented programming way with MATLAB to automate the creation of battery packs. MATLAB objects allow you to define battery specifications, visualize batteries in 3-D spaces, assign simulation strategies, and generate customized Simulink libraries.



You can create battery objects at five different levels of hierarchies:

- **Cell**: creates a single electrochemical battery cell
- **ParallelAssembly**: creates a parallel assembly object with cells connected in parallel
- **Module**: creates a module object with parallel assemblies connected in series
- **ModuleAssembly**: creates a module assembly object with modules connected in series or parallel
- **Pack**: creates a pack object with module assemblies connected in series or parallel

These objects can either be created by scripting or using the **Battery Builder** app. In this chapter, you will follow the workflow of using the **Battery Builder** app to create battery objects in an interactive way. The scripting version can be found in Appendix *Creating Battery Objects by Scripting*.



Battery Pack Modeling

Battery Builder (Continued)

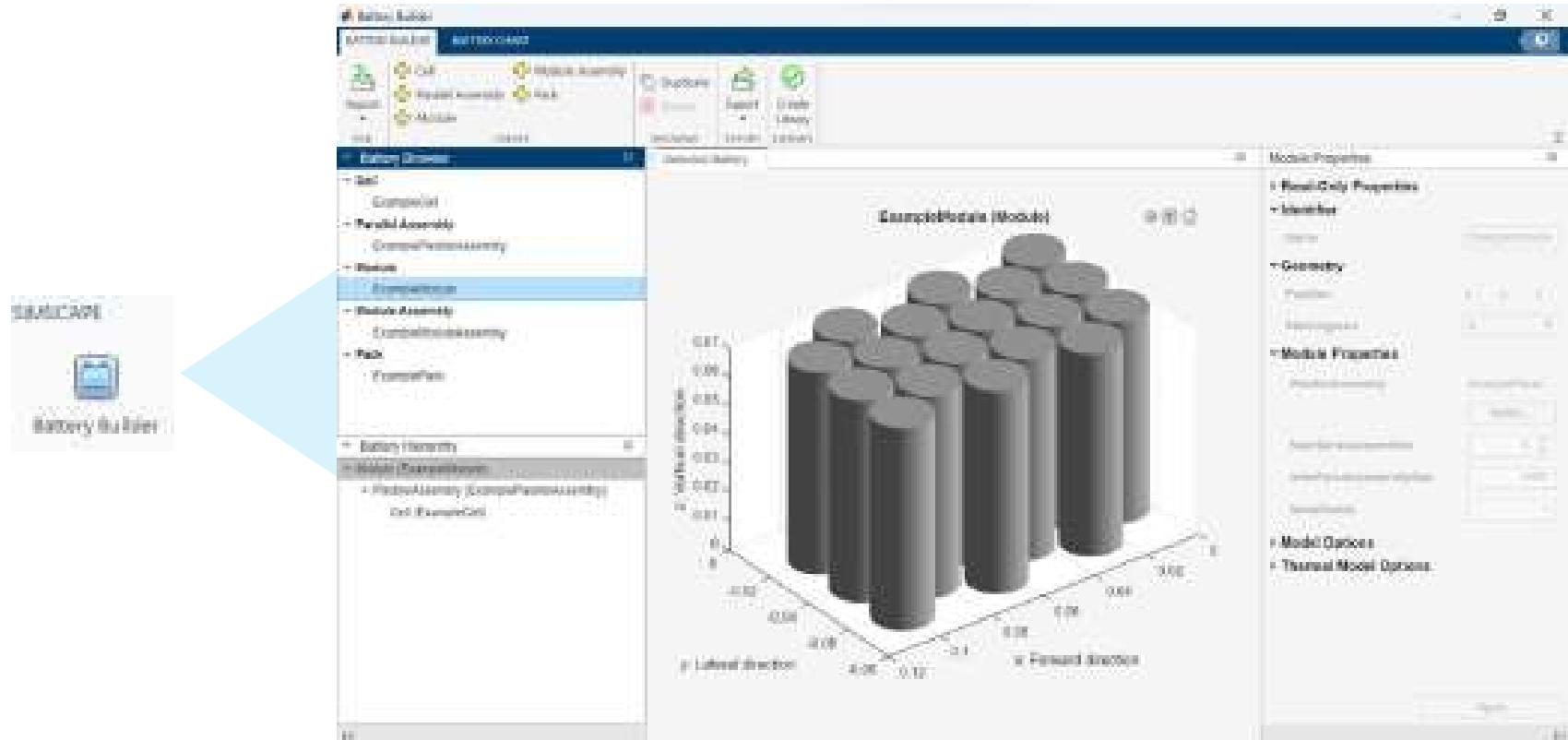
Battery Builder app provides an interactive way of creating, modifying, and visualizing battery objects. To open the **Battery Builder** app, navigate under the **APPS** tab in MATLAB, and find **Battery Builder**. You can also open the app by using the command **batteryBuilder**.

By default, **Battery Builder** app includes a set of battery objects for your reference. You can duplicate the default objects for modification or create battery objects from scratch.

Try

>> batteryBuilder

Open the **Battery Builder** app.



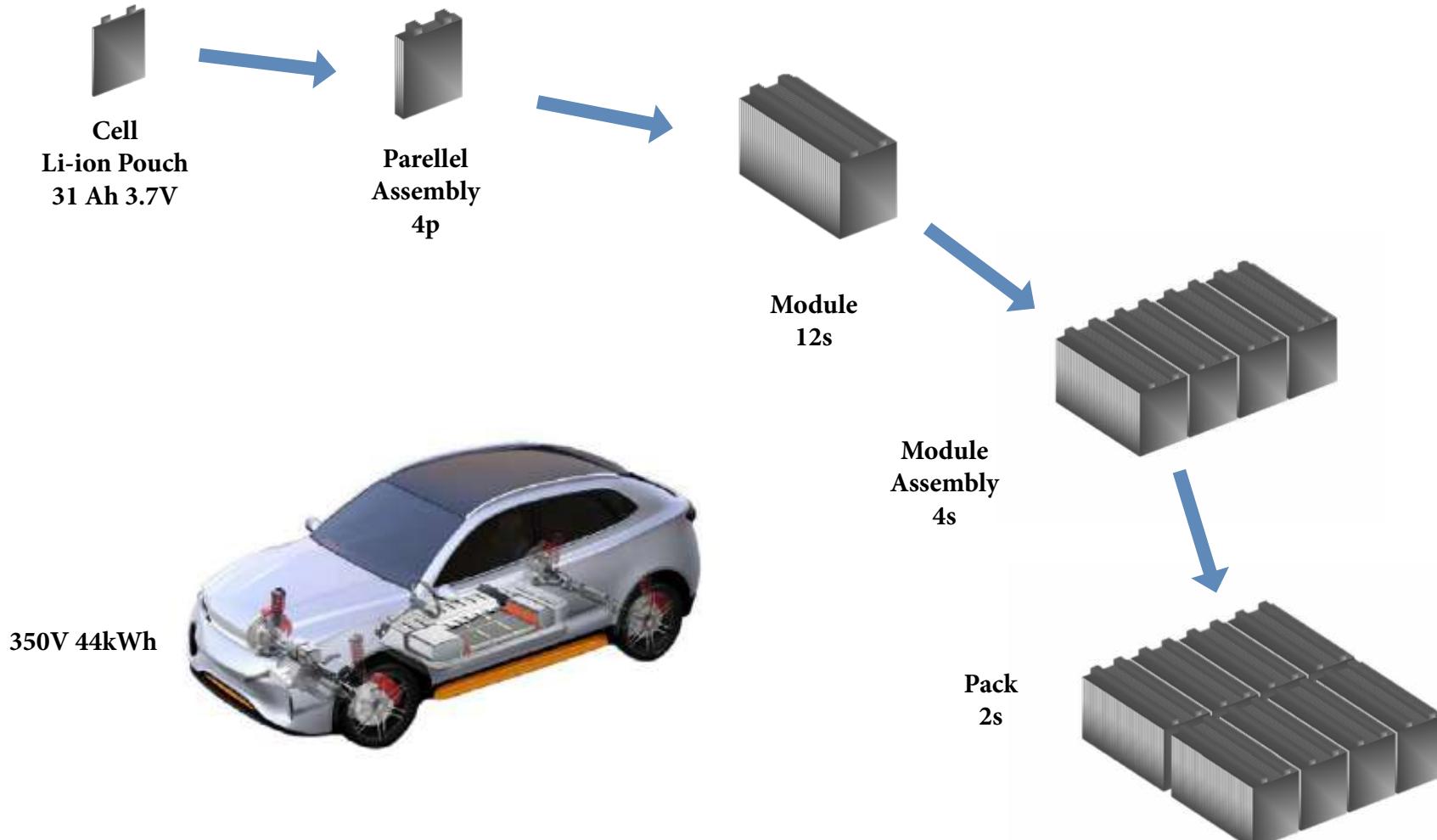
Battery Pack Modeling

Course Example – PHEV Battery Pack

As the course example, you create a battery pack of a plug-in hybrid electric vehicle with a targeting voltage of 350V and an energy level of 44kWh.

The selected cell has a nominal capacity of 31Ah and a nominal voltage of 3.7V. To meet the voltage and energy criteria, the topology of the pack is 96s4p.

To ensure the battery operates in the desired temperature range, a liquid cooling plate with parallel channels is placed underneath the pack.



Battery Pack Modeling

Creating a Battery Cell Object

1. Instantiate a battery cell object by clicking the **Cell** button under **BATTERY BUILDER > CREATE**.
2. In the **Cell Properties** panel, change the **Name** of the cell to **myCell**.



Cell Geometry

The geometry of a cell impacts the sizing and thermal design of a battery system. Simscape Battery supports three types of cell geometries: cylindrical, prismatic, and pouch. Each geometry requires distinct dimensions for its construction.



Cylindrical Cell

Radius
Height



Prismatic Cell

Length
Thickness
Height



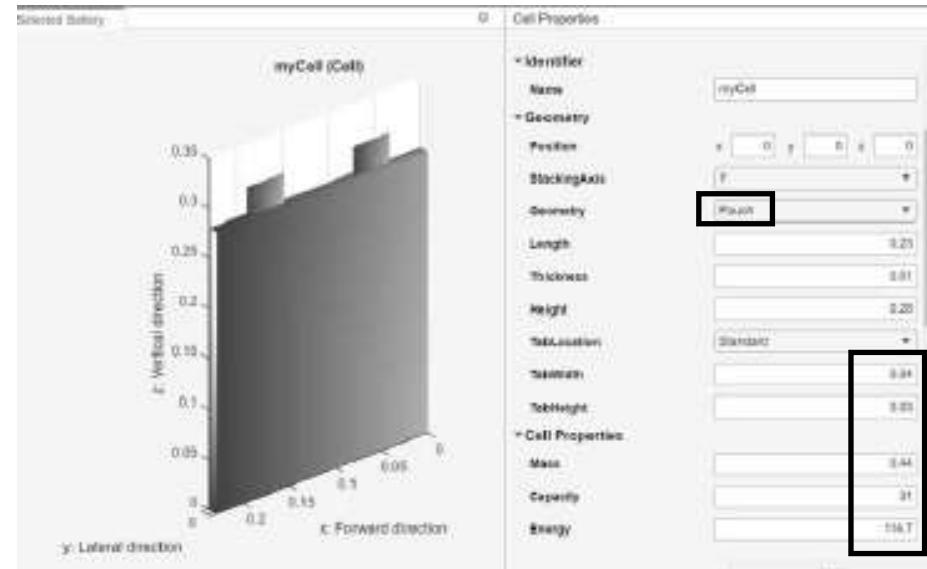
Pouch Cell

Length
Thickness
Height
TabLocation
TabWidth
TabHeight

Try

Follow the steps to create a batter cell object with Pouch geometry.

3. Select **Geometry** as **Pouch** under **Cell Properties**. Click **Apply** to examine the visualization of the pouch cell.



Cell Properties

Along with the battery cell object, you can further specify the mass, capacity, and energy of the cell as **Cell Properties**. These values will be carried over to higher level of hierarchies to calculate the cumulative values for the entire pack.

4. Specify Cell Properties as follow:

- **Mass** = **0 . 44** (kg)
- **Capacity** = **31** (Ah)
- **Energy** = **114 . 7** (Wh)

5. Click **Apply** to apply the changes.

Battery Pack Modeling

Cell Model Options

The **Cell Model Options** controls how the cell appears in Simulink. By default, the Battery (Table-Based) block is selected as the fundamental building block given by **CellModelBlockPath**.

You can specify **CellModelBlockPath** to point to any battery block in the Simulink library. If the existing battery blocks do not meet your requirements, you can write your own block with the Simscape language and generate a Simulink block with **ssc_build**. Additional information about coding with the Simscape language is available in the course *Modeling Physical Systems with Simscape*.

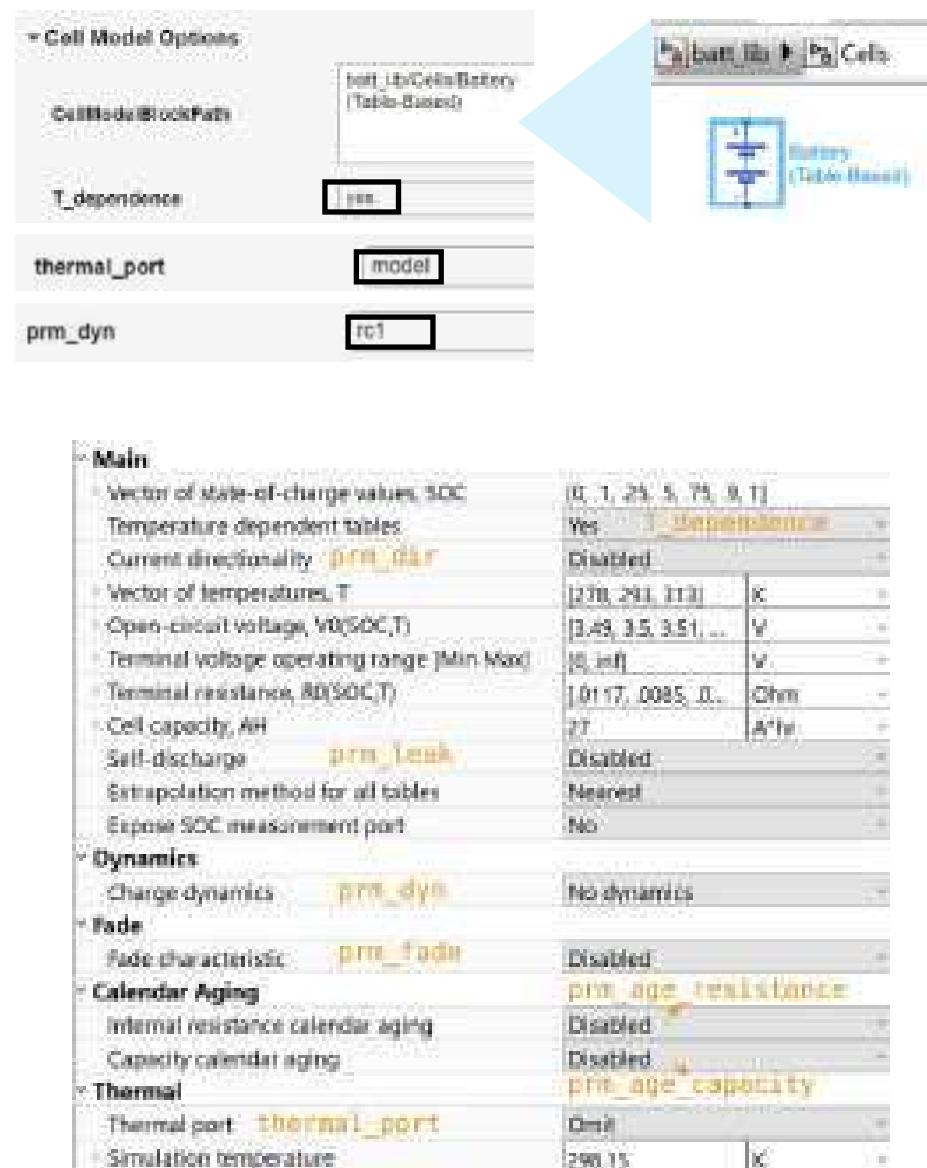
The remaining fields within the **Cell Model Options** align with the **Block Parameters** of the selected battery block. Please refer to the figure for a mapping between the options and the **Block Parameters** dialog.

Note **Cell Model Options** cannot be modified after block creation. The selected options determine which fields appear in the **Block Parameters** dialog, while the unselected options are excluded from the generated block.

1. To enable the thermal modeling options, under **Cell Model Options**, change **T_dependence** to **yes** and **thermal_port** to **model**.
2. To select a 1RC model for the cell, change **prm_dyn** to **rc1** to model one RC dynamics.
3. Click **Apply** to apply the changes.

Try

Follow the steps to enable RC dynamics and thermal modeling options for the cell.

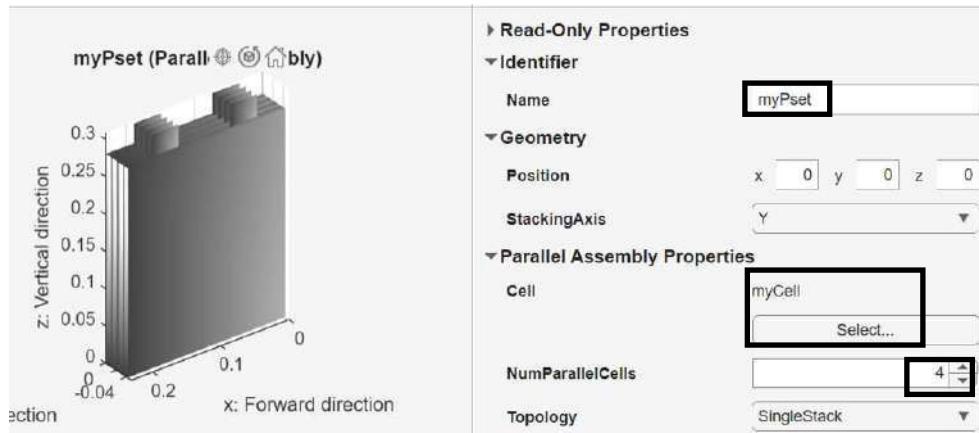


Battery Pack Modeling

Creating a Parallel Assembly Object

After creating the cell object, you connect four cells in parallel to create a parallel assembly.

1. Click on the **Parallel Assembly** button to create a new parallel assembly.
2. Change the **Name** of the parallel assembly to **myPset**.
3. Select **myCell** as the **Cell** for the parallel assembly (Note that a reselect is required if **myCell** is modified.)
4. Change the **NumParallelCells** to **4**.
5. Click **Apply** to apply the changes.



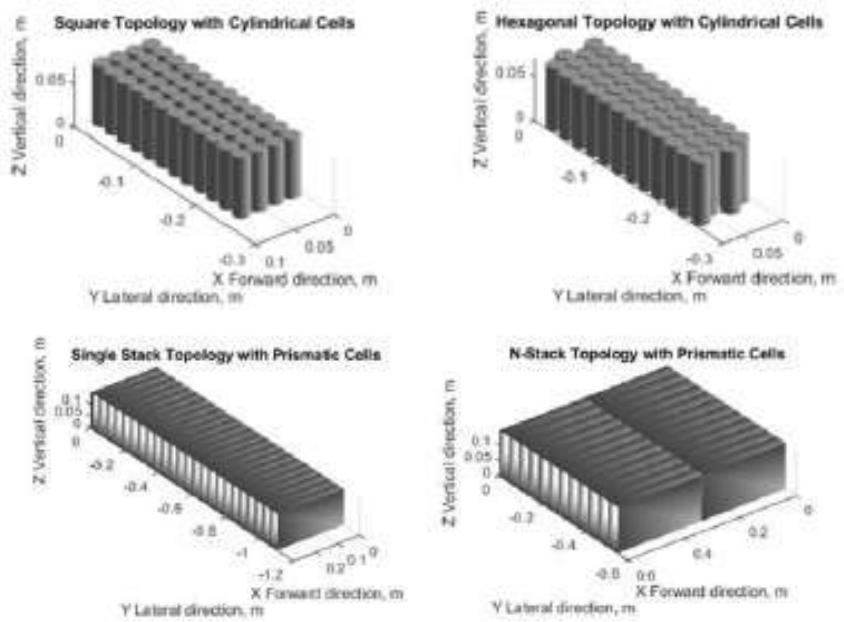
Geometry-wise, you can incorporate **InterCellGap** and **Topology** into your battery pack design. The **Topology** specifies the geometrical arrangement of cells relative to the cell format:

- **Pouch cell, prismatic cell:** Single Stack, N-Stack
- **Cylindrical cell:** Square, Hexagonal

Please refer to documentation **Simscape Battery > Battery Pack Modeling > ParallelAssembly** for more information.

Try

Follow the steps to create a parallel assembly with four cells connected in parallel.



With more than one cell, the **Battery Builder** automatically calculates the cumulative mass, volume, and capacity of the parallel assembly based on the given cell level information. You can find these values under **Read-Only Properties** of the parallel assembly.

+ Read-Only Properties					
Type	ParallelAssembly				
NumModels	1				
CumulativeMass	1.76 kg				
PackagingVolume	0.0030659 m³				
CumulativeCellCapacity	124 A·h				
CumulativeCellEnergy	458.8 Wh				
<table border="1" style="width: 100%; text-align: center;"> <tr> <td>1</td><td>2</td><td>3</td><td>4</td></tr> </table>		1	2	3	4
1	2	3	4		
Layout					

Battery Pack Modeling

Creating a Battery Module Object

Next, you create a module object that comprises multiple parallel assemblies in series.

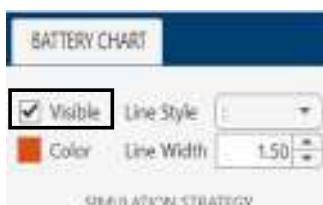
1. Click on the **Module** button to create a new module.
2. Change the **Name** of the parallel assembly to **myModLmp**.
3. Select **myPset** as the **ParallelAssembly** for the module.
4. Change the **NumSeriesAssemblies** to 12.
5. Click **Apply** to apply the changes.

Model Resolution

For parallel assemblies and modules containing multiple cells, you can set the **ModelResolution** to be either **Lumped**, **Grouped**, or **Detailed**. With **Detailed** resolution, each individual cell will be represented by a separate cell model (one set of equivalent circuit model.) While for a **Lumped** module, only one cell model will be implemented to represent the entire module. The parameters are scaled accordingly based on the number of cells in series(S) and in parallel(P).

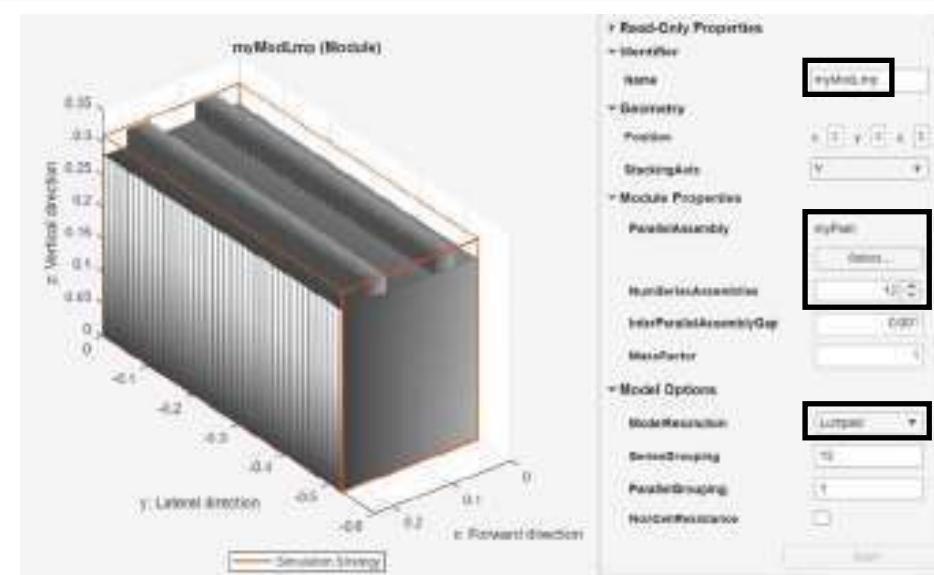
- Open Circuit Voltage: $V_{O_{Lmp}} = V_{O_{cell}} * S$
- Resistances: $R_{O_{Lmp}} = R_{O_{cell}} * S/P; R_{I_{Lmp}} = R_{I_{cell}} * S/P$
- Capacitance: $C_{I_{Lmp}} = C_{I_{cell}} * P/S$
- Capacity: $AH_{Lmp} = AH_{cell} * P$
- Thermal mass: $TM_{Lmp} = TM_{cell} * S * P$

You can visualize the model resolution by checking the **Visible** checkbox under **BATTERY CHART > SIMULATION STRATEGY**. Each orange box indicates one cell model block. When the Model Resolution is **Lumped**, only one orange box is wrapped around the entire model.



Try

Follow the steps to create a lumped module object.



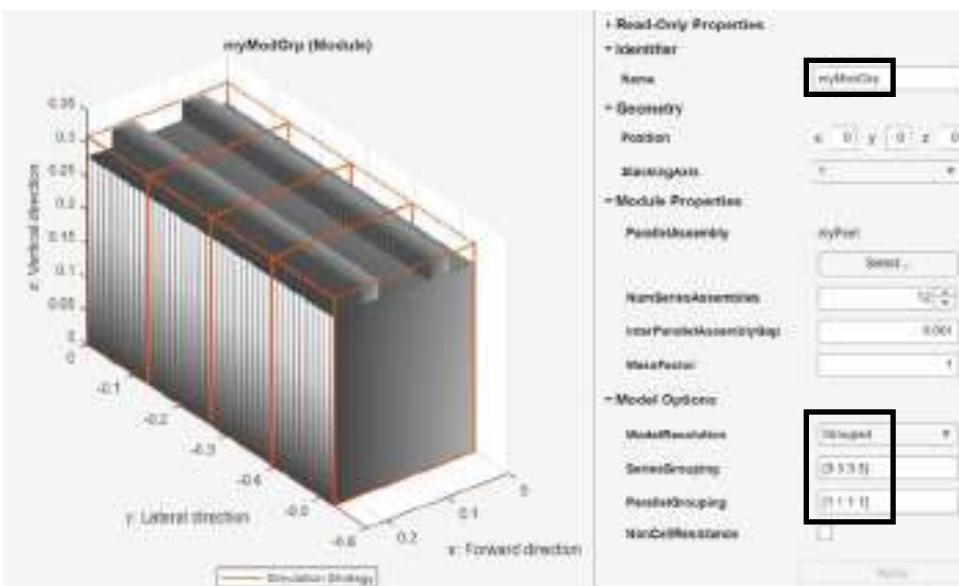
To balance simulation accuracy and speed, it is important to choose an appropriate model resolution. Desktop simulation can generally handle 60-80 cell models, whereas for hardware-in-the-loop (HIL), this number shrinks to around 30-40 cell models. To optimize your simulation, it is best to avoid creating models with over 100 cell models unless necessary.

Battery Pack Modeling

Creating a Battery Module Object (Continued)

To provide more flexibility when selecting model resolution, Simscape Battery allows users to define **Grouped** model resolution to combine custom number of cells in to one cell model. Follow the steps to divide the lumped module into four cell models, each with three parallel assemblies connected in series.

1. Duplicate `myModLmp` and rename the new module as `myModGrp`.
2. Change the **ModelResolution** of `myModGrp` to **Grouped**.
3. Change the **SeriesGrouping** to [3 3 3 3].
4. Click **Apply** to apply the changes.



Try

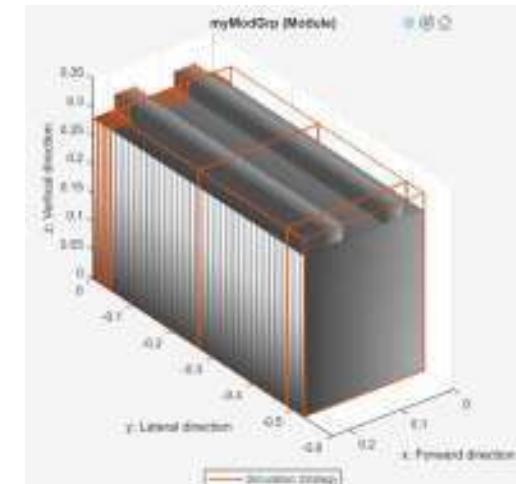
Follow the steps to create a module with grouped model resolution.

SeriesGrouping controls how you divide the series connections into multiple cell models. It is a vector whose sum of the elements must equal **NumSeriesAssemblies**.

ParallelGrouping controls the modeling strategy of parallel assemblies within the cell models defined by the **SeriesGrouping** property. **ParallelGrouping** should be a vector with the same length as **SeriesGrouping**. Each entry of the **ParallelGrouping** vector can be either 1 (lumped) or **NumParallelCells** (detailed.) Note that if the entry in **SeriesGrouping** is bigger than one, the **ParallelGrouping** can only be 1 (lumped). Here is an example of using **SeriesGrouping** and **ParallelGrouping** combined.

SeriesGrouping: [1 5 5 1]

ParallelGrouping: [4 1 1 1]



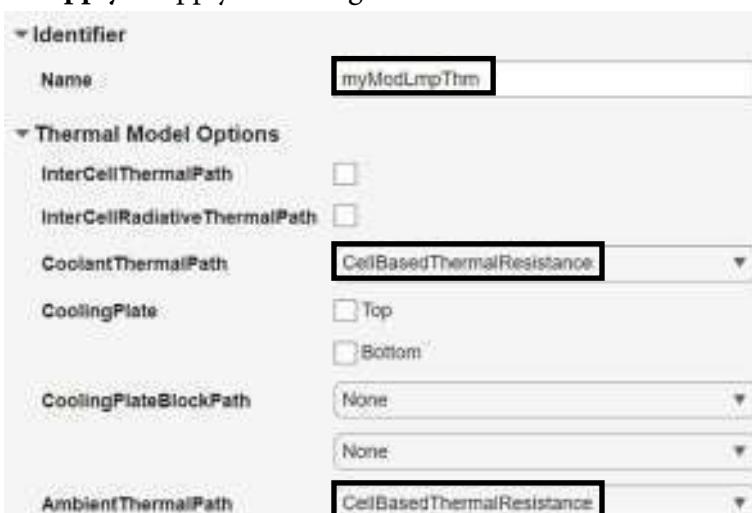
Battery Pack Modeling

Adding Thermal Model Options

Other than connecting cells in series and parallel electrically, Simscape Battery also supports you in constructing the thermal circuit along the way. Based on the selected model resolution, each cell model is represented by a thermal mass in 1D. Then to calculate the heat flow through the thermal mass, you can use the settings under **Thermal Model Options** to specify the thermal boundary conditions. These boundary conditions define the specific heat transfer mechanisms between the cell model and its surroundings, such as cell-to-ambient, cell-to-coolant, and cell-to-cell.

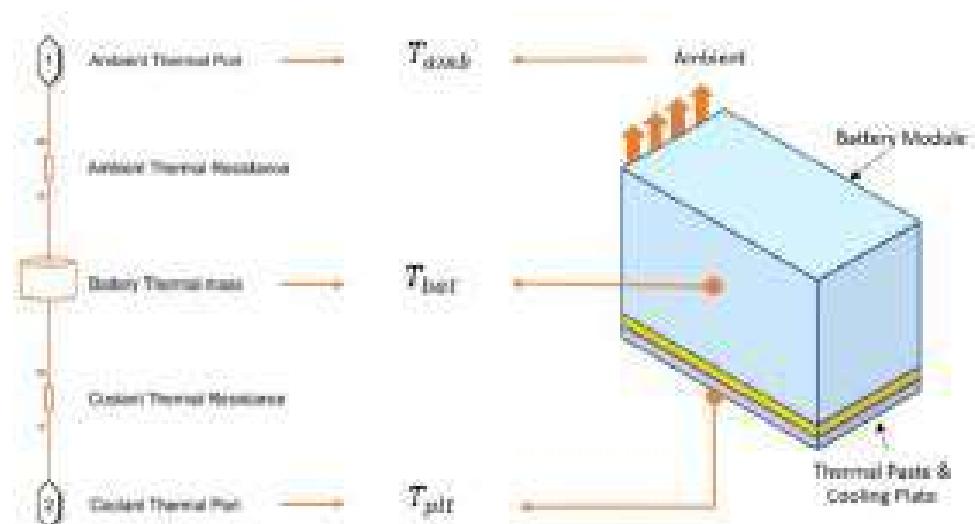
Consider the simplest case where the entire battery module is lumped as one cell model. Assume the module is exchanging heat with the ambient environment through convection while placed on a cooling plate. Simscape Battery allows you to incorporate thermal resistances to represent these heat exchanges. This is enabled by using **AmbientThermalPath** and **CoolantThermalPath**.

1. Duplicate **myModLmp** and rename it as **myModLmpThm**.
2. Change the **AmbientThermalPath** and **CoolantThermalPath** to **CellBasedThermalResistance**.
3. Click **Apply** to apply the changes.



Try

Follow the steps to add thermal model options to a lumped battery module.



Battery Pack Modeling

Adding Thermal Model Options (Continued)

Lumped battery module utilize one thermal node to characterize the temperature of the entire module, thereby disregarding variations in temperature caused by locations. If **AmbientThermalPath** and **CoolantThermalPath** are enabled for a grouped module, thermal resistances are inserted between each cell model thermal mass and the thermal ports. You can vary the values of the thermal resistances to create a spread of temperature.

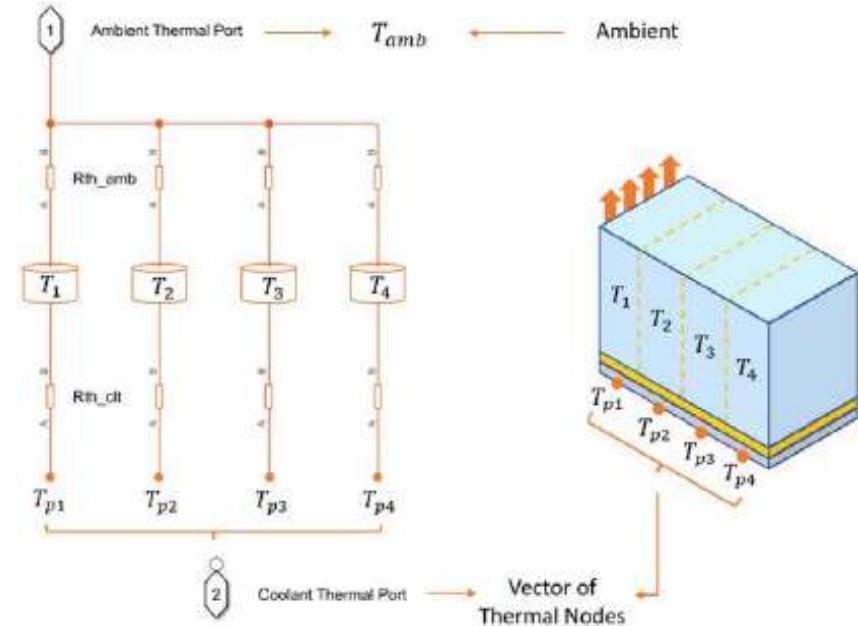
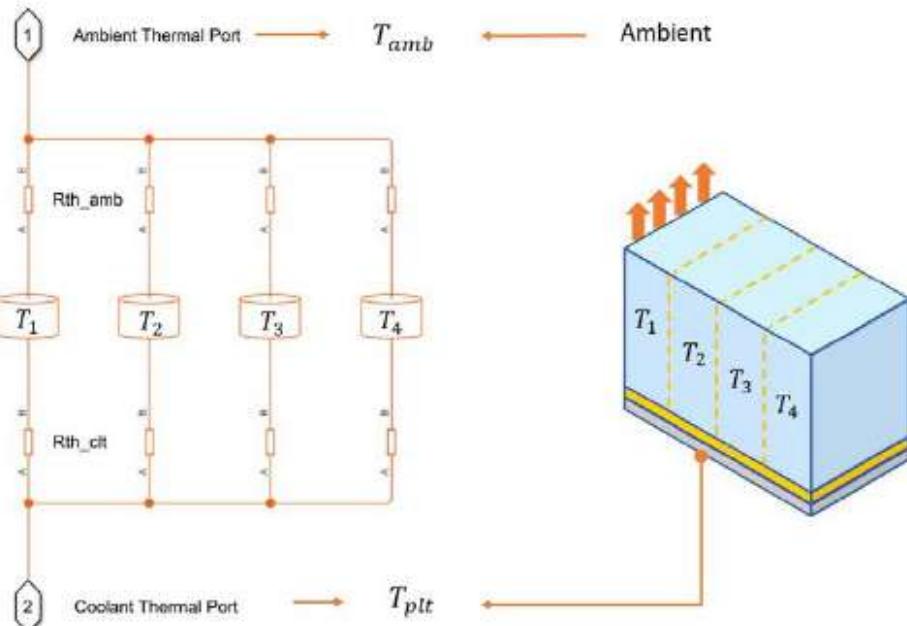
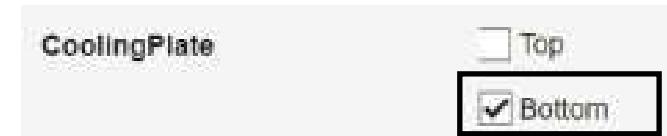
1. Duplicate **myModGrp** and rename it as **myModGrpThm**.
2. Change the **AmbientThermalPath** and **CoolantThermalPath** to **CellBasedThermalResistance**.

Try

Follow the steps to add thermal model options to a grouped battery module.

While a grouped battery module uses a separate thermal mass per cell model, by default it comes with a scalar thermal port, which suggests the exterior temperature is lumped as a single thermal node. Instead of using a single thermal node for the thermal boundary condition, you can enable the **CoolingPlate** option to output an array of thermal nodes.

3. Check the **Bottom** checkbox for option **CoolingPlate**.
4. Click **Apply** to apply the changes.



Battery Pack Modeling

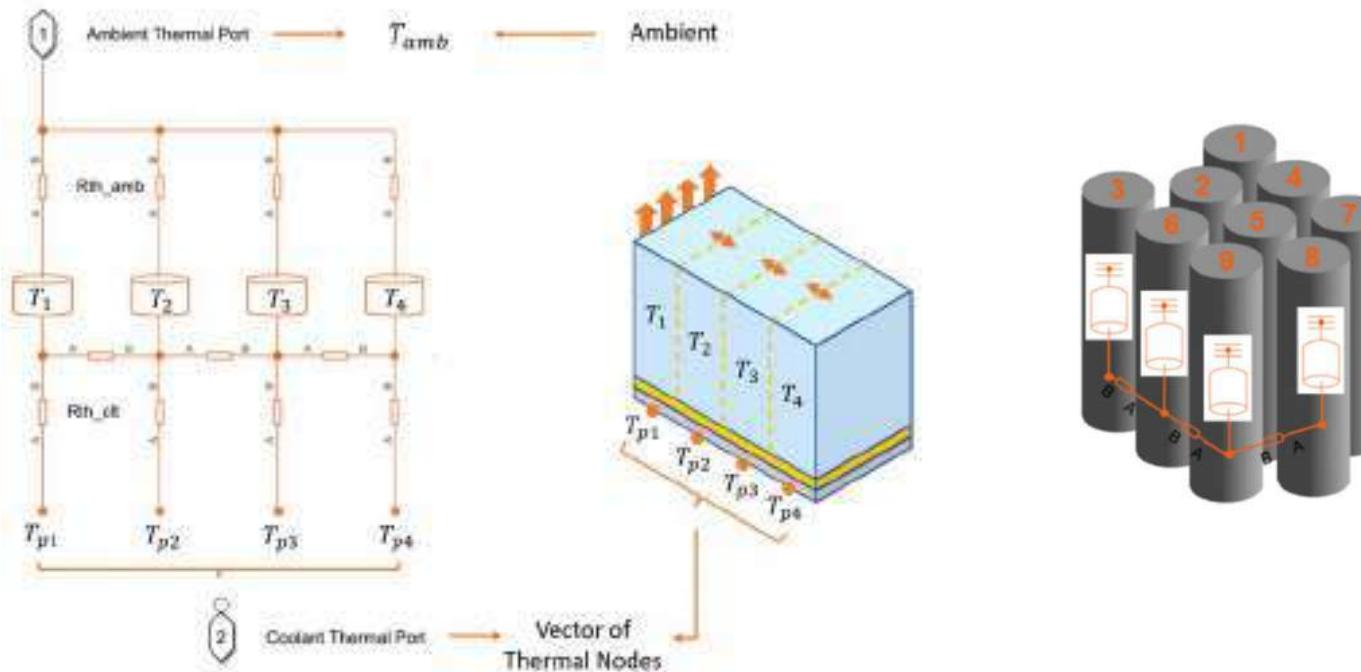
Adding Thermal Model Options (Continued)

Lastly, as the cells are sitting next to each other in the module, you should also consider including the heat transfer between cells.

1. Check the **InterCellThermalPath** checkbox.
2. Click **Apply** to apply the changes.



When **InterCellThermalPath** is enabled, Simscape Battery inserts thermal resistance between adjacent cell models. If you have more than one row of cells, the paths between rows are also included.

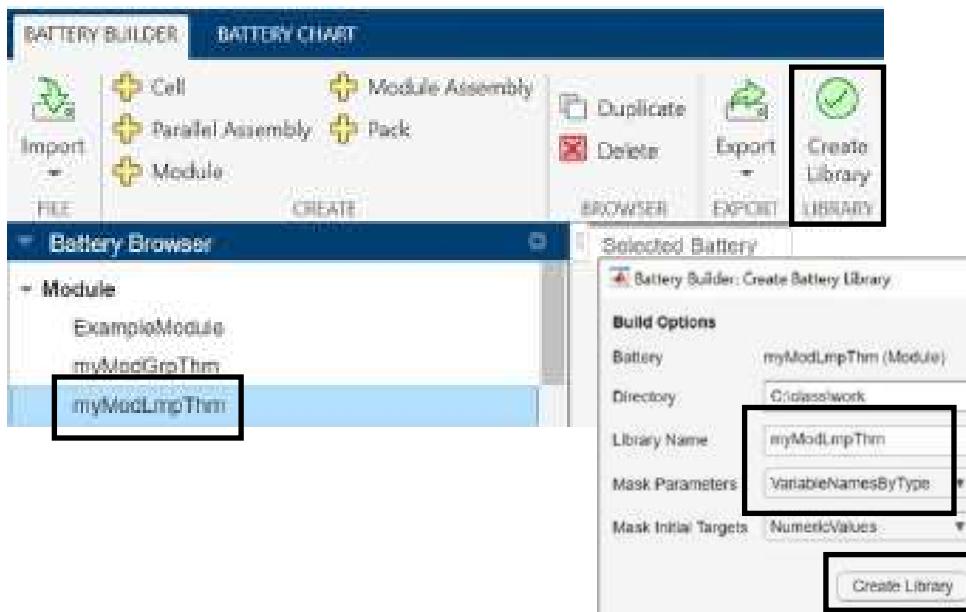


Battery Pack Modeling

Generating Simscape Battery Blocks

After creating the battery objects, you use the **Create Library** option to generate custom Simscape components corresponding to these objects.

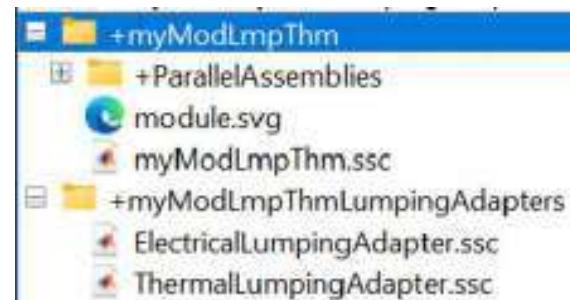
1. Select **myModLmpThm** from the **Battery Browser** and click on **Create Library**.
2. Change the **Library Name** to **myModLmpThm**.
3. Change **Mask Parameters** to **VariableNamesByType**.
4. Click on **Create Library** to generate the files.



During the build process, two folders are created within the build directory. These folders contain various **.SSC** files that define the components with the Simscape Language. The folder named **+LibraryName** includes the **.SSC** files for the module and its subcomponent parallel assembly. The selected cell model serves as the fundamental building block, and is then connected to electrical and thermal lumping adapters for scaling purposes. The lumping adapters are defined in the folder **+LibraryNameLumpingAdapters**.

Try

Follow the steps to create the Simscape Battery block for **myModLmpThm**. Examine the generated **.ssc** file.



myModLmpThm.ssc

```
components(ExternalAccess=observe)
myCell = batterycm.table_battery(SOC_vec = SOC_vecCell, ...
    T_vec = T_vecCell,V0_mat = V0_matCell,V_range = V_rangeCell,
    AH = AHCell,extrapolation_option = extrapolation_optionCell,
    thermal_mass = thermal_massCell,...
```

ElectricalLumpingAdapter.ssc

```
equations
CellNode.v == LumpedNode.v /CellsInSeries;
CurrentLumped == CurrentCell * (CellsInParallel-1);
end
```

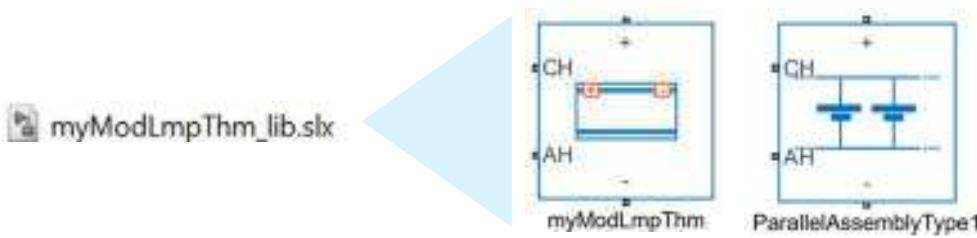
ThermalLumpingAdapter.ssc

```
equations
CellNode.T == LumpedNode.T /CellsInSeries;
HeatFlowLumped == HeatFlowBattery * (CellsInParallel-1);
end
```

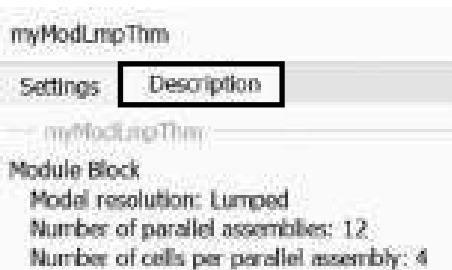
Battery Pack Modeling

Generating Simscape Battery Blocks (Continued)

The .ssc files are then transformed into Simscape blocks stored in `LibraryName_lib.slx`. This library comprises generated module and parallel assembly blocks.



The generated blocks have a similar **Block Parameters** dialog as the selected cell model. Information regarding the topology and model resolution are listed under the **Description** tab.



The content in the **Settings** tab varies based on the selected **Cell Model Options**. If **Mask Parameters** is set as **VariableNamesByType** during library creation, the parameter fields are automatically pre-filled with a data structure that bears the name of the battery object. This data structure is defined in `LibraryName_param.m`. You can use this .m file as a template and incorporate your own parameters for initialization.

Try

`>> myModLmpThm_lib`

Examine the generated blocks.

`>> edit myModLmpThm_param`

Examine the generated parameter initialization script.

myModLmpThm	
	Settings Description
NAME	VALUE
Main	
> Vector of state-of-charge values, SOC	myModLmpThm.SOC_vecCell
> Vector of temperatures, T	myModLmpThm.T_vecCell
> Open-circuit voltage, V0(SOC,T)	myModLmpThm.V0_matCell
> Terminal voltage operating range [Min Max]	myModLmpThm.V_rangeCell
> Terminal resistance, R0(SOC,T)	myModLmpThm.R0_matCell
> Cell capacity, AH	myModLmpThm.AHCell



%> myModLmpThm

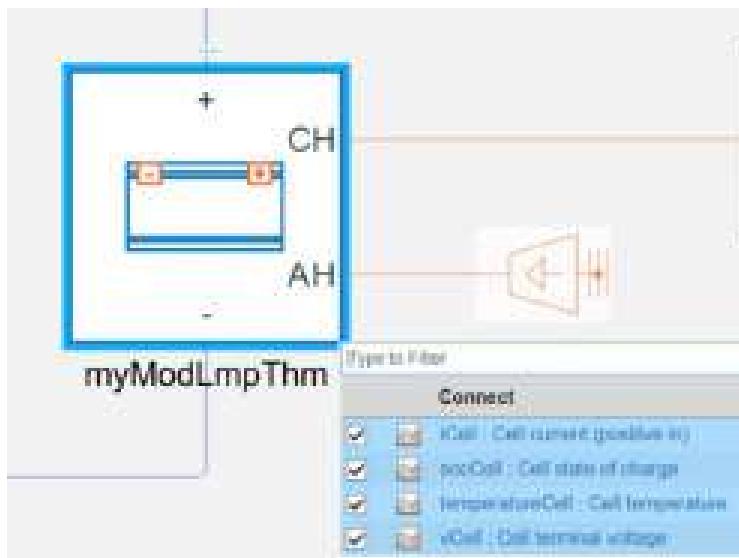
```
myModLmpThm.SOC_vecCell = [0, .1, .25, .5, .75, .9, 1]; % Vector
myModLmpThm.T_vecCell = [278, 293, 313]; % Vector of temperature
myModLmpThm.V0_matCell = [3.49, 3.5, 3.51; 3.55, 3.57, 3.56; 3.6];
myModLmpThm.V_rangeCell = [0, inf]; % Terminal voltage operating
myModLmpThm.R0_matCell = [.0117, .0085, .009; .011, .0085, .009];
```

Battery Pack Modeling

Logging Measurements From An Simscape Battery Block

Now you are ready to use the battery module in simulation.

1. Open the harness model `moduleLmp01_start.slx`.
2. Copy a `myModLmpThm` block from the library to the model.
3. Connect the + and the – terminals to the **Controlled Current Source**.
4. Double-click on the **Probe** block and select `myModLmpThm`. Check the checkboxes of `iCell`, `socCell`, `temperatureCell`, and `vCell` to expose the measurements.
5. Connect the **Probe** to the **Scope**.



Try

Follow the steps to add a battery module to the simulation harness.

```
>> moduleLmp01_start
```

Essentially, the custom battery module is a composition of multiple Simscape blocks, such as the **Battery (Table-Based) block** and the **Thermal Resistor**. Using a **Probe** block with a battery module allows you to access all Simscape nodes, variables, and intermediates within the composition.

You can find the list of measurements produced at a module level by checking the generated `.ssc` file. To prevent signal logging congestion caused by repeated values, the module block introduces variables with a **Cell** suffix (for example `vCell`) whose dimension correspond to the number of cell models. Alternatively, if your objective is to design a production BMS algorithm, where the signal dimension should mirror the actual number of measurements, the module block also includes variables with a **ParallelAssembly** suffix, whose dimensions match the number of parallel assemblies.

```
nodes
p = foundation.electrical.electrical; % +
n = foundation.electrical.electrical; % -
Cinth = foundation.thermal.thermal; % CH
AmpH = foundation.thermal.thermal; % AH
end

variables
iCell = {0,'A'}; % Cell current (positive in)
vCell = {0,'V'}; % Cell terminal voltage
socCell = {value={1,'1'},priority=priority.high}; % Cell state of charge
numCyclesCell = {value={0,'1'},priority=priority.high}; % Cell discharge cycles
temperatureCell = {value={290.15,'K'},priority=priority.high}; % Cell temperature
vParallelAssembly = {value={repmat(0,12,1),'V'}},priority=priority.none; % Parallel Assembly Voltage
socParallelAssembly = {value={repmat(1,12,1),'1'},priority=priority.none}; % Parallel Assembly state of charge
end

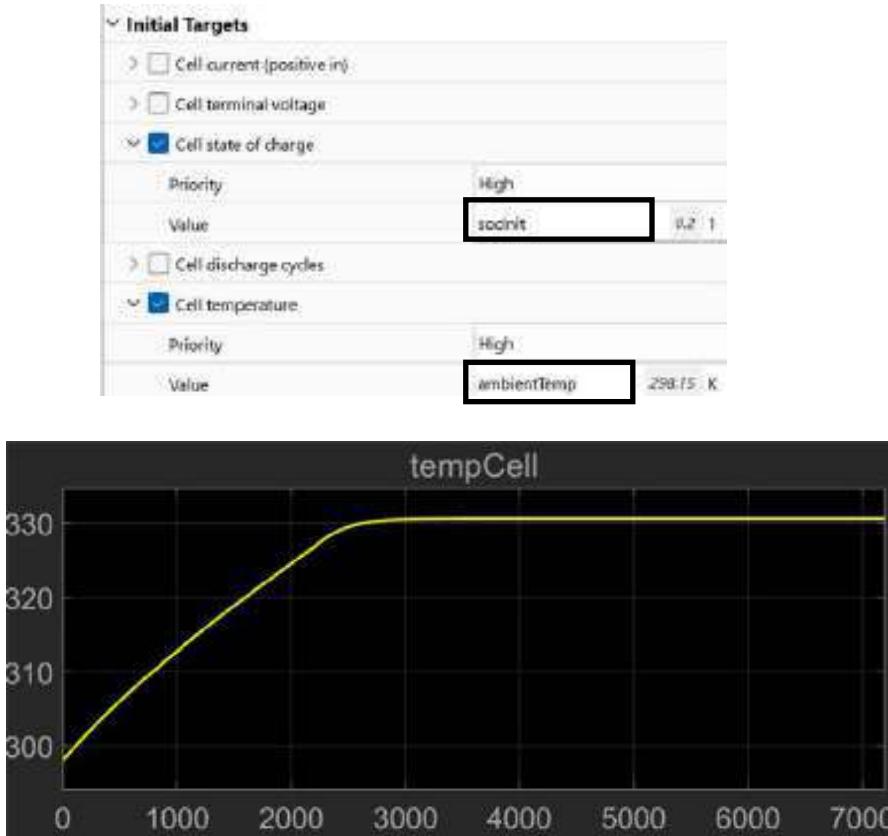
intermediates
power_dissipated = myCell.power_dissipated*((S*P)-1); % Power dissipated by (S*P)-1 cells
end
```

Battery Pack Modeling

Heat Generated By The Module

In this chapter, you analyze the battery and cooling performance under CCCV charging. The battery first goes under 1C fast charge until the terminal voltage hits 4.15V. Then, constant voltage charging is applied. The battery is assumed to be placed under an ambient environment with a constant temperature of 298.15K (25°C). After adding the module block, you will need to provide appropriate initial conditions for the simulation.

1. Go to the Block Parameters of myModLmpThm. Under **Initial Targets**, set **Cell state of charge** to **socInit** and set **Cell temperature** to **ambientTemp**.
2. Simulate the model to examine the heat generated by the cell.



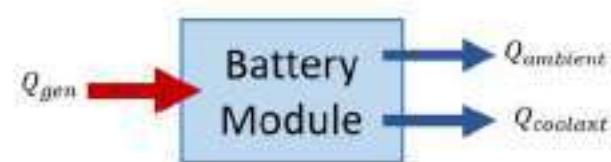
Try

Follow the steps to set the initial condition of the module. Simulate the model and examine the heat generated during fast charging.

As introduced in the cell modeling chapter, due to the internal resistance of the battery, heat is generated when current flows through the cell. The generated heat flow rate (q_{gen}) is determined by the summation of all the ohmic losses included in the cell model.

$$c_p m \dot{T} = q_{gen} = \sum_i \frac{V_{T,i}^2}{R_{T,i}}$$

When there is no heat exchange, all generated heat will be converted to a temperature rise of the battery, with a ratio that depend on the mass of the cell (m) and the specific heat capacity (c_p) of the material. Given the specific heat capacity of 964 J/kgK, the temperature rise during charging is around 31K, brining the final temperature to 330K (57°C). Next, you model the heat exchange through the ambient thermal path and the coolant thermal path to bring the temperature down to a normal operating range. The overall energy flow is represented as shown below.



In this course, the following symbols are used to denote heat transfer related terms.

- Q : Energy transfer, J
- q : Heat transfer rate, W
- q' : Heat transfer rate per unit length W/m
- q'' : Heat flux (heat transfer rate per unit area), W/m²

Battery Pack Modeling

Adding Ambient Thermal Path

You first model the heat exchange between the module and the ambient environment.

1. Add a **Temperature Source** block from the **Simscape > Foundation Library > Thermal > Thermal Sources** library into the model. Set the **Temperature** to **ambientTemp**.
2. Connect the **Temperature source** to the **AH** port and simulate the model.



Thermal resistance

Depending on how the battery module is placed on the equipment, modeling the heat exchange between the battery and the ambient environment can be complicated. In Simscape Battery, you will use a **Thermal Resistance** to represent the overall heat transfer resistance between the ambient environment and the battery module.



Thermal resistance is a heat property that measures how effectively a component resists the flow of heat. Thermal resistance has a unit of K/W. It is defined as the ratio of temperature change per Watt of heat flow going through a specific object.

$$R_{th} = \frac{\Delta T}{q}$$

- R_{th} : Thermal resistance (K/W)
- ΔT : Temperature difference across the object (K)
- q : Rate of heat flow (W)

Try

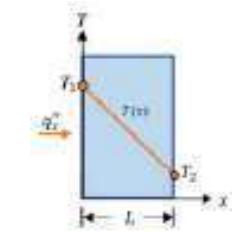
Follow the steps to add an constant temperature ambient environment to the model. Simulate the model.

With the given equation, a known rate of heat flow is required to calculate the value of the thermal resistance. You can apply Fourier's law and Newton's law to calculate the rate of heat transfer for conduction and convection respectively. Both state that the rate of heat transfer is proportional to the temperature difference.

- Fourier's law of conduction in 1D

- Heat flux:
$$q''_x = k \frac{\Delta T}{\Delta x} = k \frac{T_1 - T_2}{L}$$
- Thermal resistance:
$$R_{cond} = \frac{\Delta T}{q_x} = \frac{L}{kA}$$

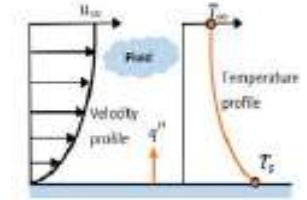
k : Thermal conductivity (W/m·K)
 A : Area of the plane (m^2)



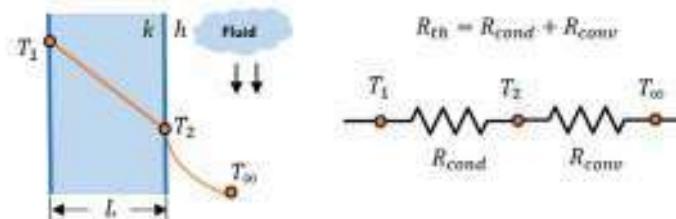
- Newton's law of cooling in 1D

- Heat flux:
$$q'' = h(T_s - T_\infty) = h\Delta T$$
- Thermal resistance:
$$R_{conv} = \frac{\Delta T}{q} = \frac{1}{hA}$$

h : Convection heat transfer coefficient (W/m²·K)
 A : Area of the plane (m^2)



Thermal resistance is analogous to electrical resistance. You can combine thermal resistances in series and in parallel to calculate the overall heat transfer if multiple layers of materials are involved.



Battery Pack Modeling

Adding Ambient Thermal Path (Continued)

Considering the overall thermal resistance of the ambient thermal path, the resistance value of each layer is heavily dependent on material properties, geometry, and interface conditions, making it challenging to be calculated accurately. Typically, this value is determined directly through experimental measurements. Below is a reference experiment set up to determine the thermal resistance.

1. Place the battery in a temperature chamber with an ambient temperature of 298.15K (25°C). Soak the battery for long enough time to make sure the battery temperature is the same as the ambient.
2. Discharge the battery with 1C until the battery temperature rises to 318.15K (45°C), stop the discharge.
3. Record how the temperature changes with time. Stop the experiment when the battery temperature drops back to ambient temperature again.
4. Calculate the thermal resistance from the cooling time constant.

In the experiment, all the heat generated by the battery is dissipated through the ambient thermal resistance. Solving the energy balance you can obtain the equation of battery temperature change with time.

- Assume all heat generated is dissipated through R:

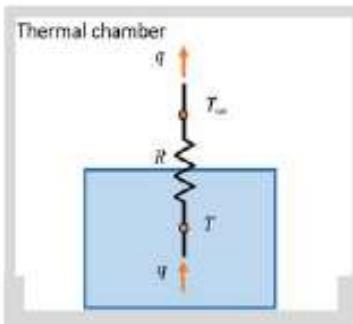
$$mc_p \frac{dT}{dt} = q = \frac{T - T_\infty}{R} \Rightarrow \frac{dT}{dt} = \frac{1}{mc_p R} (T - T_\infty)$$

- Integrate and get the analytical solution:

$$T(t) = (T_0 - T_\infty)e^{-t/\tau} + T_\infty$$

Where

- $\tau = mc_p R$ Thermal time constant
- $T_0 = T(t=0)$ Initial condition

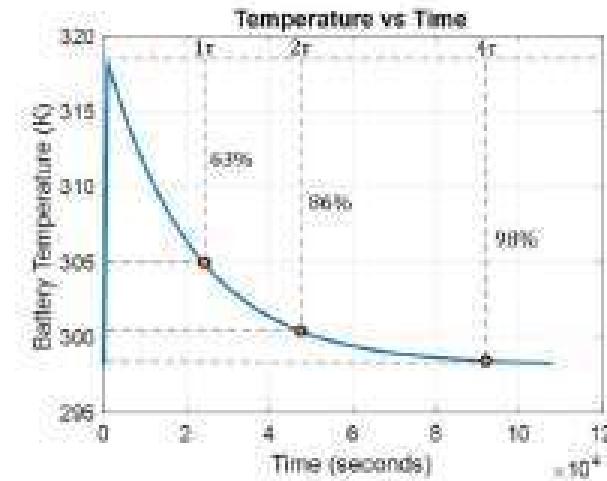


Try

```
>> edit thermalResCal
```

Run the script to calculate the thermal resistance.

By recording the temperature data, you can obtain the time constant from the cooling curve as shown below. Then, the thermal resistance can be calculated as $R = \tau / mc_p$.



Alternatively, you can use the **Parameter Estimator** app as introduced in chapter *cell characterization* for similar purpose. Here are some reference ambient thermal path resistances for cells with different geometries.

Cell Type	Cylindrical	Pouch	Prismatic
Capacity (Ah)	5	50	150
Thermal Resistance (K/W)	2000	250	20

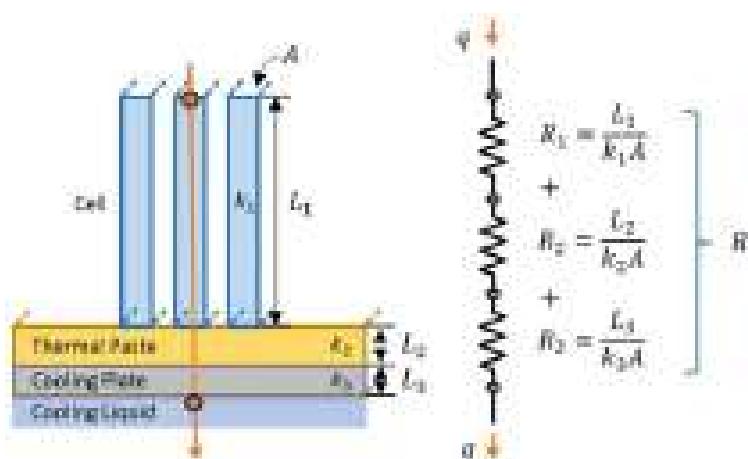
Battery Pack Modeling

Adding Coolant Thermal Path

Similar to the ambient thermal path, in Simscape Battery, the heat transfer between the battery and the coolant is also represented by a thermal resistance. Here are some reference coolant thermal path resistances accounting for the resistance between the cell and the coolant.

Cell Type	Cylindrical	Pouch	Prismatic
Capacity (Ah)	5	50	150
Thermal Resistance (K/W)	15-25	1-5	0.1-0.5

The heat transfer path to the cooling plate is relatively straightforward. Instead of experiment, you can also directly calculate the combined thermal resistance. Consider a simplified scenario where a battery is positioned on a layer of thermal paste and then attached to a cooling plate. The temperature sensor is placed on top of the cell and the coolant temperature is measured. The diagram represents the entire thermal circuit. By merging the thermal resistances of each layer, the coolant thermal path resistance can be calculated.



Try

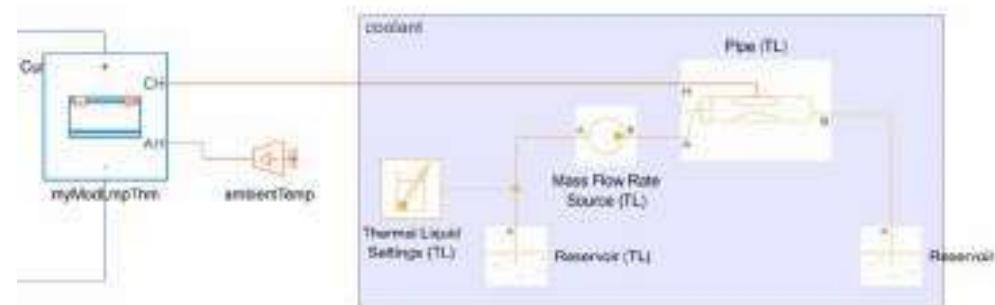
Uncomment the coolant subsystem and connect it to the CH port of the battery module. Simulate the model.

Or,

```
>> moduleLmp02_coolant
```

Modeling the coolant

While the coolant temperature can be represented as simple as a Temperature Source, in this example, you will be investigating what mass flow rate is required to lower the temperature to an acceptable level. The Simscape **Foundation Library** provides blocks for modeling the flow of the coolant and its heat exchange, which fall under the **Thermal Liquid** domain. Follow the Try box to connect the module to the **coolant** subsystem modeled with thermal liquid blocks.

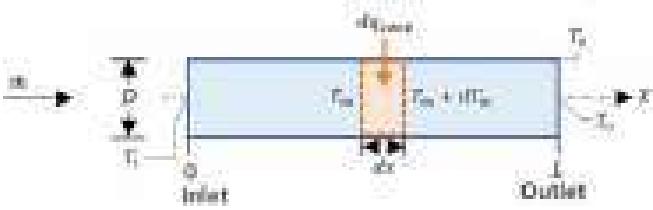


The Pipe (TL) block represents a pipeline segment with a fixed volume of liquid. The liquid experiences heat transfer due to convection between the fluid and the pipe wall. Here we are considering a pipe segment with the same length as the battery module. The liquid in the pipe is driven by a constant mass flow rate determined by the Mass Flow Rate Source (TL) block. To set the boundary conditions in the thermal liquid network, both sides of the pipe are connected to Reservoir (TL) blocks, that set the pressure and temperature of liquid assumed with infinite volume. The cross-sectional areas of all the blocks are set as the same.

Battery Pack Modeling

Pipe Internal Flow Heat Transfer Fundamentals

Consider a circular pipe with length L and diameter D. Water is flowing through the pipe with a constant mass flow rate m. The flow is assumed fully developed which indicates the temperature distribution across the flow remains constant. T_m to denote the mean temperature over the cross-section of the tube. The convective heat transfer between the pipe wall and the fluid causes T_m to change continuously with the pipe axis.



Assume the surface temperature of the pipe is held at constant T_s, and the temperature difference between T_s and T_m decays exponentially along the pipe axis. This relationship can be derived by applying energy balance to a control volume along the tube as shown below.

- Apply energy balance to the control volume:

$$dq_{conv} = \dot{m}c_p dT_m = q_s P dx \quad P = \pi D$$

$$\frac{dT_m}{dx} = \frac{q_s P}{\dot{m}c_p} = \frac{P}{\dot{m}c_p} h(T_s - T_m)$$

- Define $\Delta T = T_s - T_m$, integrate over the pipe length:

$$\ln \frac{\Delta T_o}{\Delta T_i} = \frac{PL}{\dot{m}c_p} h \quad \frac{\Delta T_o}{\Delta T_i} = \frac{T_s - T_o}{T_s - T_i} = \exp \left(-\frac{PL}{\dot{m}c_p} h \right)$$

Then, the overall heat transfer over the pipe can be written as:

$$q_{conv} = \dot{m}c_p (\Delta T_i - \Delta T_o)$$

$$q_{conv} = \dot{m}c_p \Delta T_i \left(1 - \exp \left(-\frac{PL}{\dot{m}c_p} h \right) \right)$$

To calculate the heat transfer, you must calculate the convective heat transfer coefficient h. Under fully developed conditions, h can be considered as a constant. The Pipe (TL) block calculates h from the Nusselt number Nu (ratio of convective and conductive heat transfer).

$$Nu_D = \frac{\text{convective heat transfer}}{\text{conductive heat transfer}} = \frac{hD}{k}$$

k : Liquid thermal conductivity (W/m·K)

The value of the Nusselt number can be obtained theoretically or empirically depending on the flow regime. For laminar flow in a circular pipe, Nu is a constant since the flow is relatively uniform. In turbulent flow, Nu can be calculated from the empirical Gnielinski correlation as shown below.

$$Nu_D = 3.66 \quad \text{laminar flow}$$

$$Nu_D = \frac{(f/8)(Re - 1000)Pr}{1 + 12.7(f/8)^{1/2}(Pr^{2/3} - 1)} \quad \text{turbulent flow}$$

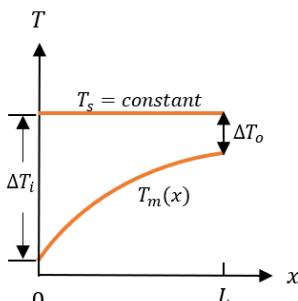
Pr: Liquid Prandtl number

f : Darcy friction factor

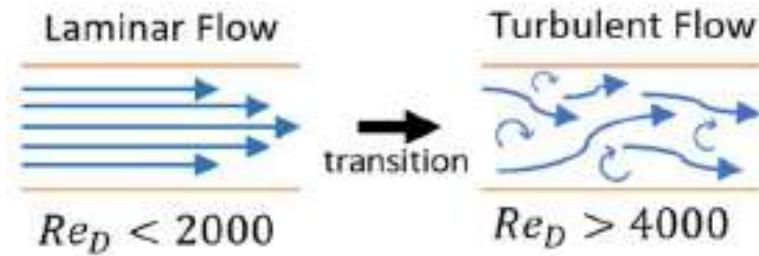
The flow regime of a fluid is governed by the dimensionless Reynolds number Re. It is a ratio between inertial force and viscous force.

$$Re_D = \frac{\text{inertia forces}}{\text{viscous forces}} = \frac{uL}{v} = \frac{\rho u L}{\mu} = \frac{\dot{m}D}{A\mu}$$

ρ : fluid density A: cross-section area
 u : fluid velocity v: kinematic viscosity
 μ : dynamic viscosity



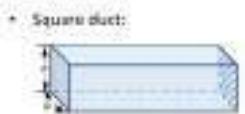
A small Reynolds number indicates a laminar flow while a big Reynolds number indicates a turbulent flow. The chart below indicates the correlation between the flow regime to the Reynolds number.



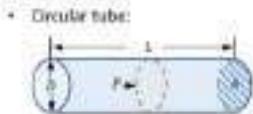
Battery Pack Modeling

Adding Coolant Thermal Path (Continued)

When using the Pipe (TL) block, you can specify the pipe **Geometry** in terms of the **Pipe length**, **Cross-sectional area**, and **Hydraulic diameter**. Shape plays an important factor when it comes to fluid calculation. As most of the formulas are derived based on circular geometry, if you are working with a different geometry, you must calculate the equivalent hydraulic diameter that transforms the non-circular shape into its circular representative. Hydraulic diameter is defined as four times the cross-sectional area divided by the wetted perimeter.



$$D_h = \frac{4a^2}{4a} = a$$



$$D_h = \frac{4\pi r^2 / 4}{\pi r l} = D$$



$$D_h = \frac{4ab}{2(a+b)} = \frac{2ab}{a+b}$$

$$\text{Hydraulic Diameter: } D_h = \frac{4A}{P}$$

Getting the correct hydraulic diameter is important as it impacts the calculation of the Reynolds number and the Nusselt number. Meanwhile, the shape factor also impacts your **Friction and Heat Transfer** settings. The default block parameters are based on a circular pipe. For laminar flow in a non-circular pipe, you must find the corresponding Darcy friction factor and Nusselt number from the table below. For turbulent flow, the equation mentioned on the previous page still applies.

Cross Section	b / a	Nu_D (Uniform T_s)	f
	-	3.66	64
	1.0	2.98	57
	2.0	3.39	62
	4.0	4.44	73

Try

Examine the parameters of the Pipe (TL) block.

Pipe (TL)	
Settings	Description
NAME:	
Geometry	
» Pipe length	moduleLength 0.22 m
» Cross-sectional area	crossArea 2e-05 m^2
» Hydraulic diameter	D 0.005 m
Friction and Heat Transfer	
» Aggregate equivalent length of local resistances	1 m
» Internal surface absolute roughness	15e-6 m
» Laminar flow upper Reynolds number limit	2000
» Turbulent flow lower Reynolds number limit	4000
» Laminar friction constant for Darcy friction factor	64
» Nusselt number for laminar flow heat transfer	3.66
Effects and Initial Conditions	
» Fluid dynamic compressibility	Off
» Initial liquid temperature	AmbientTemp 289.15 K

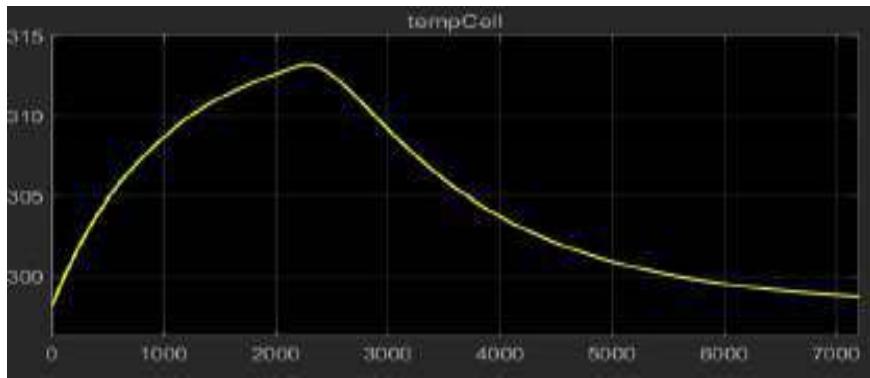
Additionally, you can specify the type of liquid using a Thermal Liquid Settings (TL) block. This block includes parameters regarding liquid density, thermal conductivity, and kinematic viscosity, which are used in the pipe calculation. The default block parameters are based on water. You will need to update the values in the table if you are using other cooling liquids. Or alternatively, use the Thermal Liquid Properties (TL) block (with pre-defined tables of common liquids) from the Simscape Fluids library.

Thermal Liquid Settings (TL)		
Internal Energy		
Internal energy parameterization	Specific heat vector - (p(T))	
» Specific heat at constant pressure vector, c_p [4.2199; 4.7955; 4.184; 4.180...]	kJ/(K*kg)	
Viscosity and Conductivity		
» Kinematic viscosity vector, nu(T)	[1.7817; 1.3061; 1.0032; 8006...]	mm^2/s
» Thermal conductivity vector, k(T)	[561.04; 583.03; 598.81; 615.4...]	W/(K^2*m)

Battery Pack Modeling

Simulating Battery Module with Grouped Fidelity

After simulating the lumped model, you observe that cooling brings the highest temperature down to 313K (40°C) and the end-of-charge temperature is around 298K (25°C). However, the temperature difference with respect to cell model locations is not captured.



Generate a module with grouped fidelity

To improve the simulation fidelity you use a grouped cell model to examine the temperature variance based on cell locations.

1. In the **Battery Builder** app, select **myModGrpThm** and **Create Library**. Name the library as **myModGrpThm** and change **Mask Parameters** to **VariableNamesByType**.
2. Open the harness model **moduleGrp01_start.slx**, and copy a **myModGrpThm** block from the library to the model and connect it with the existing circuit.
3. Go to the **Block Parameters** of **myGrpThm**. Under **Initial Targets**, set **Cell state of charge** to **repmat(socInit, 4, 1)** and set **Cell temperature** to **repmat(ambientTemp, 4, 1)**.

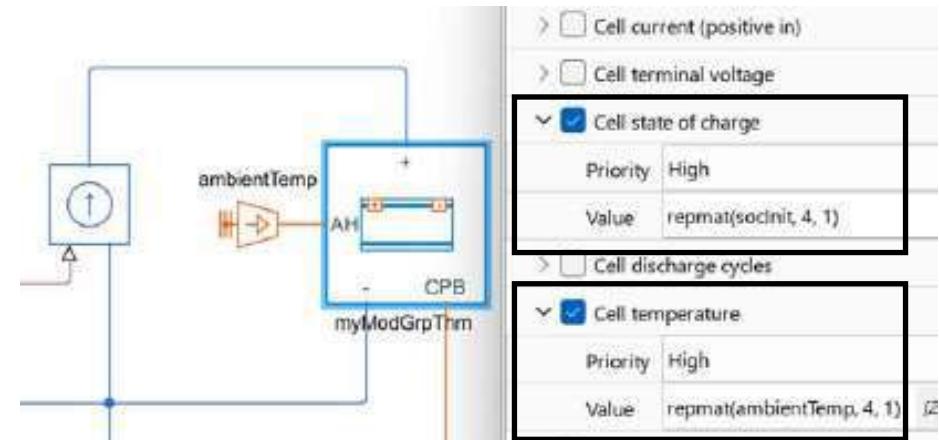
Note that, with grouped model resolution, you can provide different initial target values to each individual cell model. Function **repmat(A, m, n)** returns a matrix of copies of **A** with dimension **mxn**.

Try

```
>> moduleGrp01_start
```

Follow the steps to add a grouped module into the model.

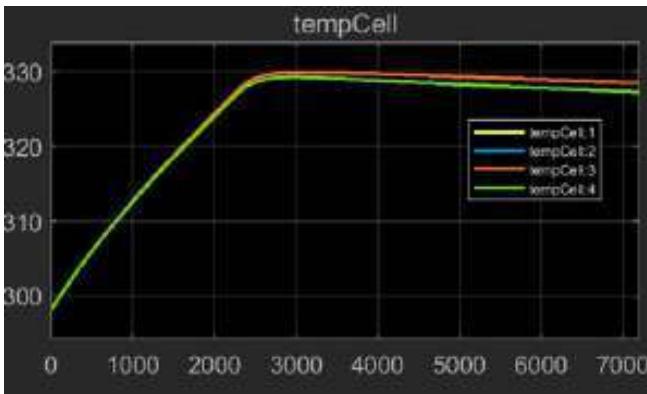
4. Setup the Probe block by exposing **iCell**, **socCell**, **temperatureCell**, and **vCell** measurements. Connect the Probe to the scope.
5. Simulate the model.



Battery Pack Modeling

Simulating Battery Module with Grouped Fidelity (Continued)

To address the temperature variance caused by location differences, you can provide different ambient thermal path resistances and coolant thermal path resistances. For example, the ambient thermal resistances of cells in the middle of a module should be bigger than the ones on the edge. Below is the simulation result with ambient thermal resistance of [100 400 400 100]. The maximum temperature difference is 1.1K.



You can use a **BatterySimulationChart** to visualize battery states from simulation as an overlay to the battery chart. Simscape simulation log (simlog) must be enabled to record the simulation data. You can turn on the setting from the command line by running

```
>> set_param(modelname,"SimscapeLogType",'all');
```

Then, create a **BatterySimulationLog** object to link the simlog with the battery object.

```
>> batterySimLog = ...
    BatterySimulationLog(batteryObj,simLog);
>> batterySimLog.SelectedVariable = "temperatureCell";
```

Try

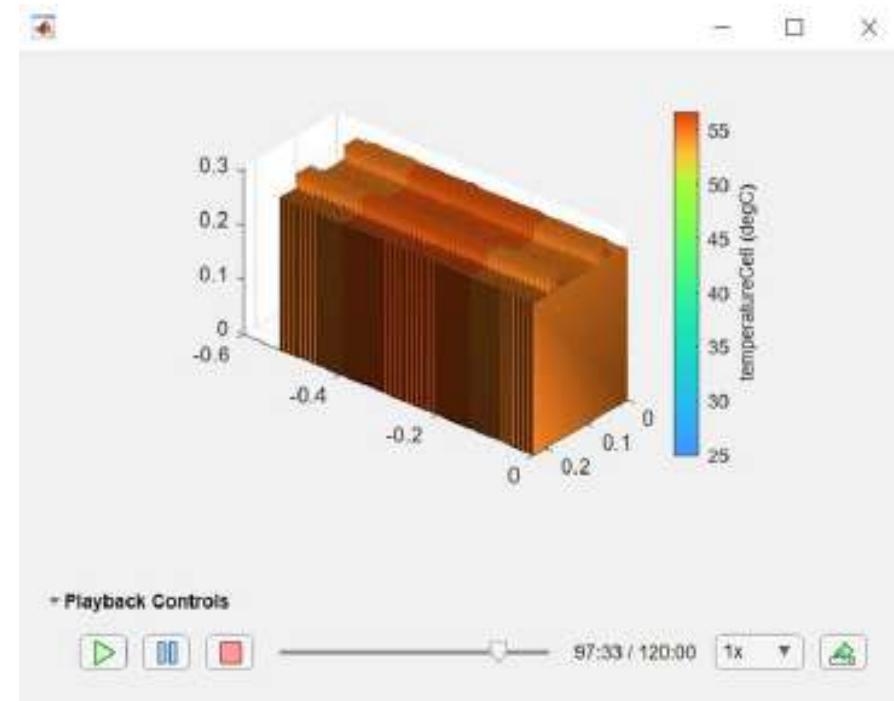
```
>> temperaturePlot_module
```

Run the script to generate a temperature plot.

Lastly, you will create a **BatterySimulationChart** to visualize the data.

```
>> moduleChart = BatterySimulationChart(..., ...
    Parent =figure, ...
    BatterySimulationLog = batterySimLog);
>> moduleChartColorBar = colorbar(moduleChart);
```

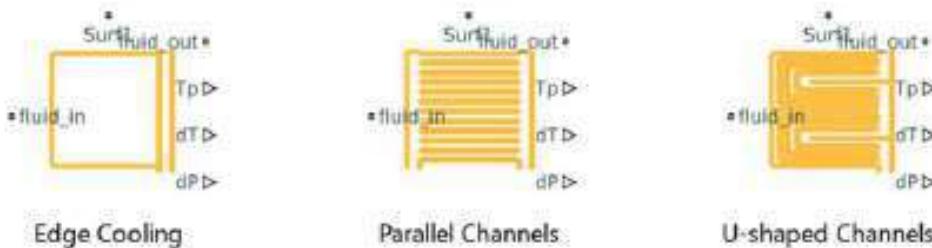
You can use the **Playback Controls** to replay the temperature change in the battery module and export a video if needed. The visualization indicates the cells in the middle have higher temperatures than the cell on the edge.



Battery Pack Modeling

Edge Cooling

In addition to enhancing the simulation fidelity of the module, increasing the cooling plate resolution is also possible. Simscape Battery comes with a set of cooling system blocks that can be used to interface with the battery module. You will explore the simplest case edge cooling first and explore other cooling methods in later sections.



1. Add an Edge Cooling block from the **Simscape > Battery > Thermal** library. Connected the block with the existing circuit.

To accommodate the varying sizes of battery modules, the dimensions of the cooling plate should be determined based on specific module objects. When creating a battery object, Simscape Battery automatically calculates the **ThermalNodes** property that stores the dimensions and locations of the cell models. This information can then be passed to a cooling plate block for configuration.

2. In the **Battery Builder** app, export **myModGrpThm** into the base workspace, and rename the object as **myModGrpThmObj**.
3. Double-click on **myModGrpThmObj**, and examine the thermal nodes stored under **myModGrpThmObj .ThermalNodes .Bottom**.



myModGrpThmObj 1x1 Module	
myModGrpThmObj.ThermalNodes.Bottom	
Field	Value
Locations	[0.1150,0.0655;0.1150,0.1975;0.1]
Dimensions	[0.2300,0.1318;0.2300,0.1318;0.2]
NumNodes	4

Try

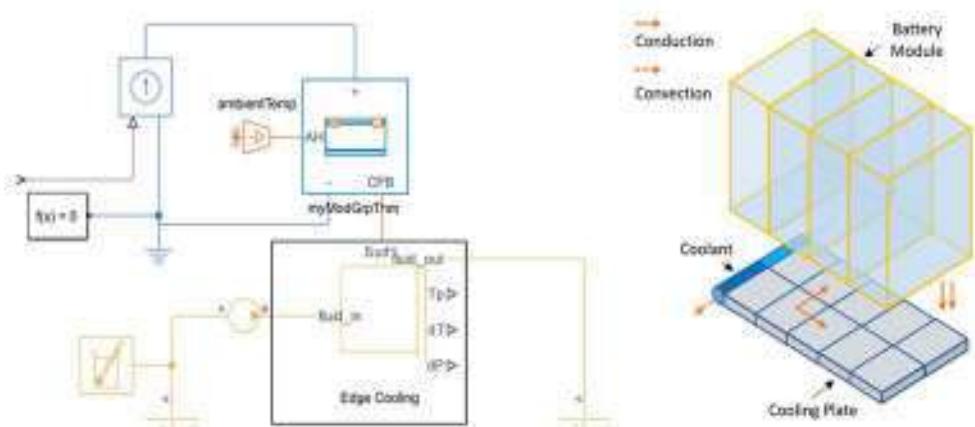
```
>> moduleGrp02_edgeCool
```

Follow the steps to add an Edge Cooling plate to the model.

4. Open the **Block Parameters** of the Edge Cooling block. Set up the **Interface** settings by using the information stored under **myModGrpThmObj**.

Interface

Battery Connectivity	Single sided
Number of battery thermal nodes (sur...	myModGrpThmObj.ThermalNodes.Bottom.NumNodes
Dimension of battery thermal nodes (s...	myModGrpThmObj.ThermalNodes.Bottom.Dimensions
Coordinates of battery thermal nodes ...	myModGrpThmObj.ThermalNodes.Bottom.Locations



Battery Pack Modeling

Edge Cooling (Continued)

Just like how Simscape Battery generates custom battery modules, the cooling plate blocks also employ built-in Simscape blocks to construct the thermal circuit. After determining the size of the cooling plate, it is subsequently divided into segments based on the number of partitions in X and Y dimensions. Each individual plate segment will be represented by a Thermal Mass.

Number of partitions in X dimension for the cooling plate	2
Number of partitions in Y dimension for the cooling plate	5

The heat transfer of a plate segment can be summarized into three components.

- **Conduction heat transfer between the plate segments**

In between the plate segments, the Conductive Heat Transfer blocks are in place to model the heat transfer between the plate segments. You can configure the cooling plate details under the **Plate Material** section. Follow the figure to set the parameter of the plate.

Plate Material

> Thickness of cooling plate material	2e-3	m
> Thermal conductivity of cooling plate mat...	154	W/(K*m)
> Density of cooling plate material	2730	kg/m^3
> Specific heat of cooling plate material	893	J/(K*kg)
> Initial temperature of the cooling plate an...	ambientTemp	298.15 K

- **Convective heat transfer to the coolant**

In the edge cooling configuration, the coolant flows at one end of the flat plate. Similar to the previous example, the modeling of the coolant is handled by the Pipe (TL) block. You can attach the pipes to the edge determined by **Select edge to be cooled**.

Try

```
>> moduleGrp02_edgeCool
```

Update the Edge Cooling block parameters as shown in the figures.

Design

Select edge to be cooled

Edge cooling along X axis at Y = Ymin

> Cooling channel hydraulic diameter

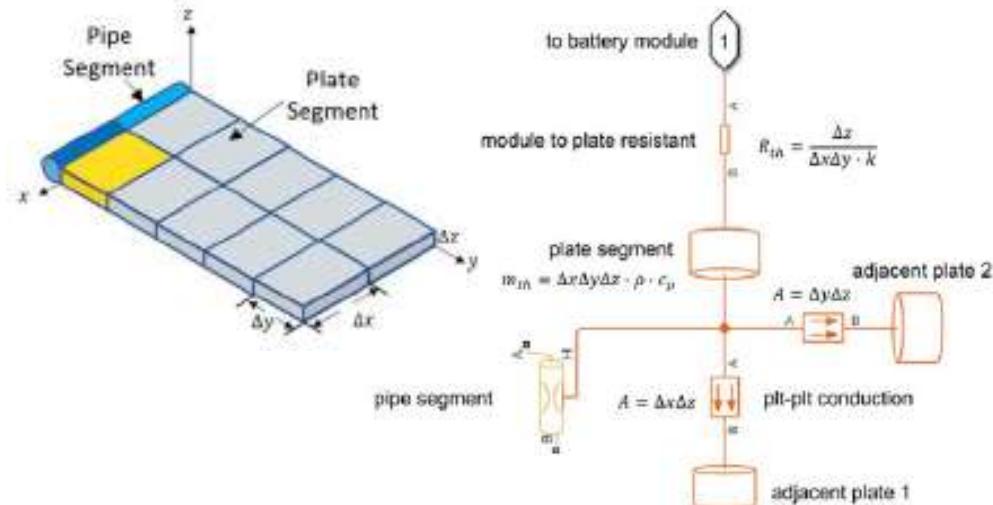
5e-3 m

> Cooling channel cross-sectional area

2e-5 m^2

> Coolant channel roughness

1.5e-05 m



- **Conduction heat transfer between the battery and the plate**

The conduction heat transfer between the battery and the plate segment is modeled by Thermal Resistance. Detailed implementation is listed on the next page.

Battery Pack Modeling

Edge Cooling (Continued)

In the implementation of battery to plate heat transfer, all cell models are assumed to be connected to all cooling plate segments. Then, the calculated heat flow is multiplied by a contact ratio between zero and one. The contact ratio is calculated based on the actual contact area of the cell model and the cooling plate segment. To illustrate, the figure below shows the heat flow between cell model k and plate segment i,j. The red number on the plate indicates the contact ratio. Eventually the total heat flow of a specific cell model or a cooling plate can be calculated by summing up related heat flow components.

- Heat flow between cell model k and plate segment i,j :

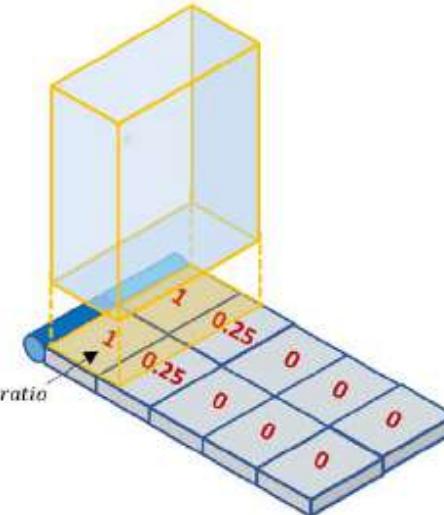
$$q_{bat(k),plt(i,j)} = \frac{T_{plt(i,j)} - T_{bat(k)}}{R_{th}} \cdot ratio$$

- Overall heat flow of plant segment i,j :

$$q_{plt(i,j)} = \sum_{k=1}^w \frac{T_{plt(i,j)} - T_{bat(k)}}{R_{th}} \cdot ratio$$

- Overall heat flow of cell model k :

$$q_{bat(k)} = \sum_{i=1}^m \sum_{j=1}^n \frac{T_{plt(i,j)} - T_{bat(k)}}{R_{th}} \cdot ratio$$



where

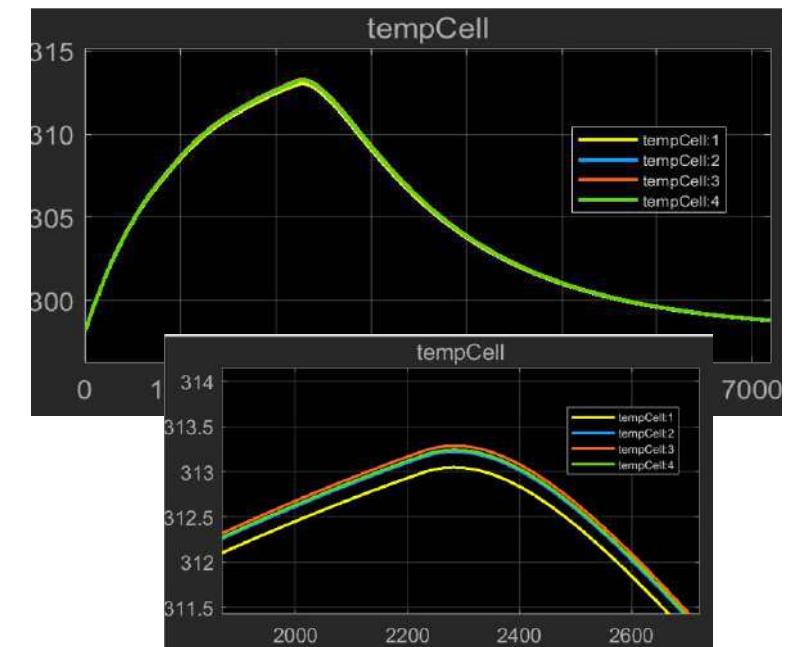
- w : number of cell models
- m, n : number of plat segemnts in x and y directions

Try

```
>> moduleGrp03_edgeCool
```

Simulate the model to explore the temperature difference between the cell models.

Simulate the model with edge cooling and zoom in to view the temperature differences of the four cell models. The simulation result shows that cell model one has the lowest temperature as it is closest to the coolant. The maximum temperature delta between the cell models is 0.3K.



Battery Pack Modeling

Creating A Module Assembly

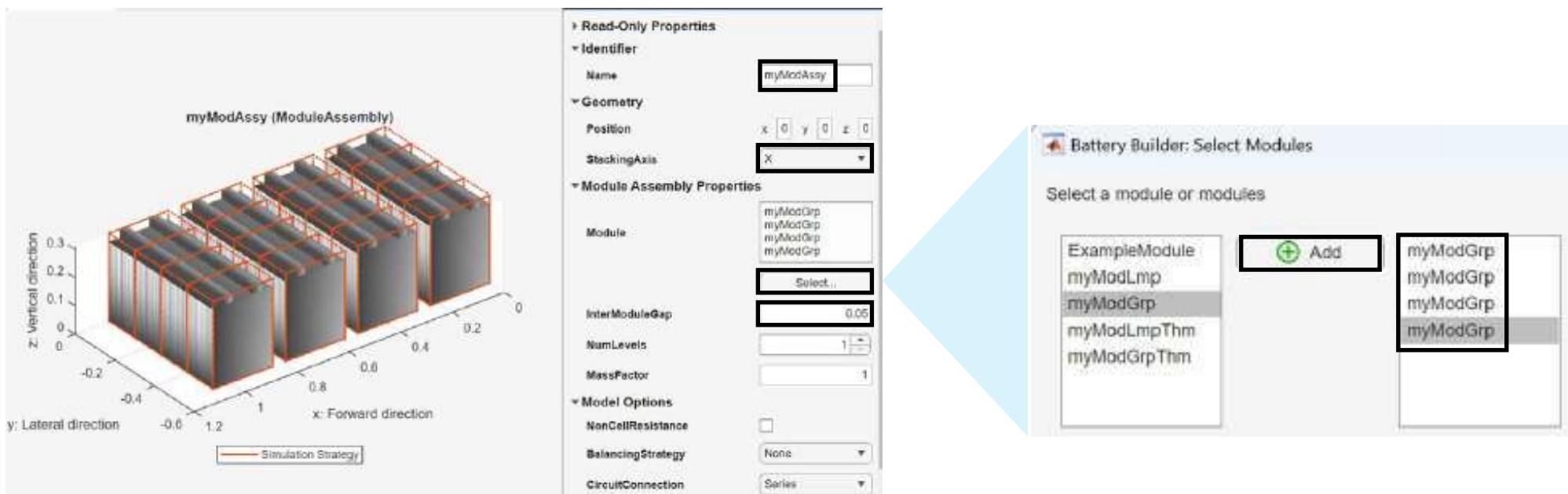
Next, you will continue your job in the **Battery Builder** app to create a module assembly and battery pack.

1. Click on the **Module Assembly** button to create a new module assembly.
2. Change the **Name** of the module assembly to **myModAssy**.
3. Click on the **Select** button and select the module for the assembly. Add four **myModGrp** to the module assembly.
4. Change the **StackingAxis** to **X**.
5. Change the **InterModuleGap** to **0 . 05**.
6. Click **Apply** to apply the changes.

Try

Follow the steps to create a module assembly.

At the module assembly level, Simscape Battery no longer asks you the number of modules to connect in series or parallel. Instead, you add modules individually and use the **CircuitConnection** property to choose between **Series** and **Parallel** connections. This setup enables you to combine different modules (that is modules with different fidelities) into a single module assembly.



Battery Pack Modeling

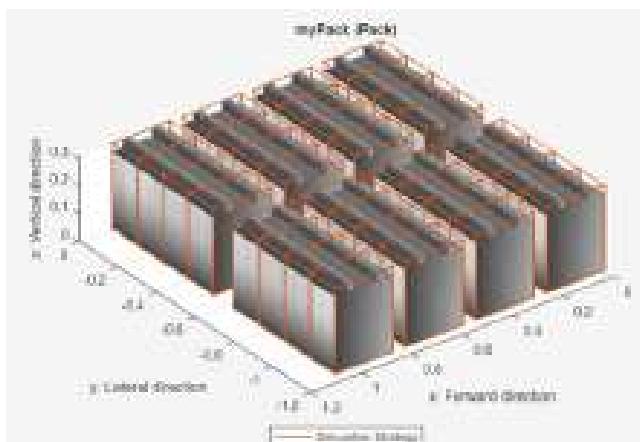
Creating Battery Pack

In the end, you create the full battery pack by connecting two module assemblies in series.

1. Click on the **Pack** button to create a new pack.
2. Change the **Name** of the pack to **myPack**.
3. Click on the **Select** button and add two **myModAssy** to the pack.
4. Change the **StackingAxis** to **Y**.
5. Change the **InterModuleGap** to **0.1**.

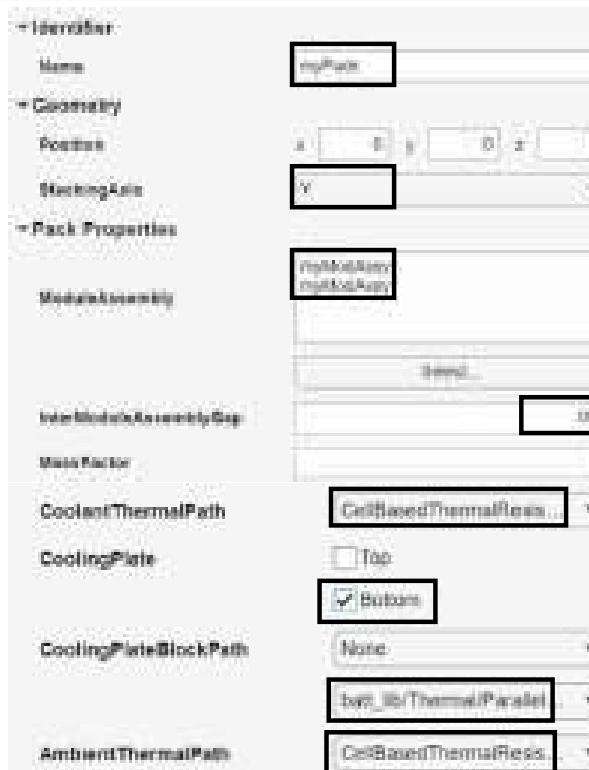
When creating battery objects using Simscape Battery, **Thermal Model Options** should always be set at the top level of hierarchy. In this example, you insert a **Parallel Channels** cooling plate at the bottom of the battery pack.

6. Change the **AmbientThermalPath** and **CoolantThermalPath** to **CellBasedThermalResistance**.
7. Check the **Bottom** checkbox for option **CoolingPlate**.
8. Under **CoolingPlateBlockPath**, change the second drop-down to **batt_lib/Thermal/Parallel Channels**.
9. Click **Apply** to apply the changes.



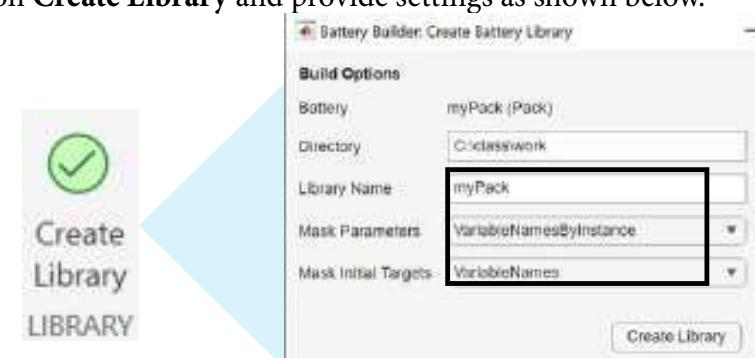
Try

Follow the steps to create a battery pack.



Lastly, generate the Simulink library for the battery pack.

10. Click on **Create Library** and provide settings as shown below.



Battery Pack Modeling

Simulating a Battery Pack

Module assembly and pack objects are generated as subsystems that comprise module blocks. You can find them under the library file `LibraryName.slx`. Inside the pack or module assemblies, the subcomponents are electrically and thermally wired together based on the topology settings. Output signals are combined into vectors to be passed between layers. If the cooling plate property is specified, the corresponding cooling plate block will be inserted automatically.

Try

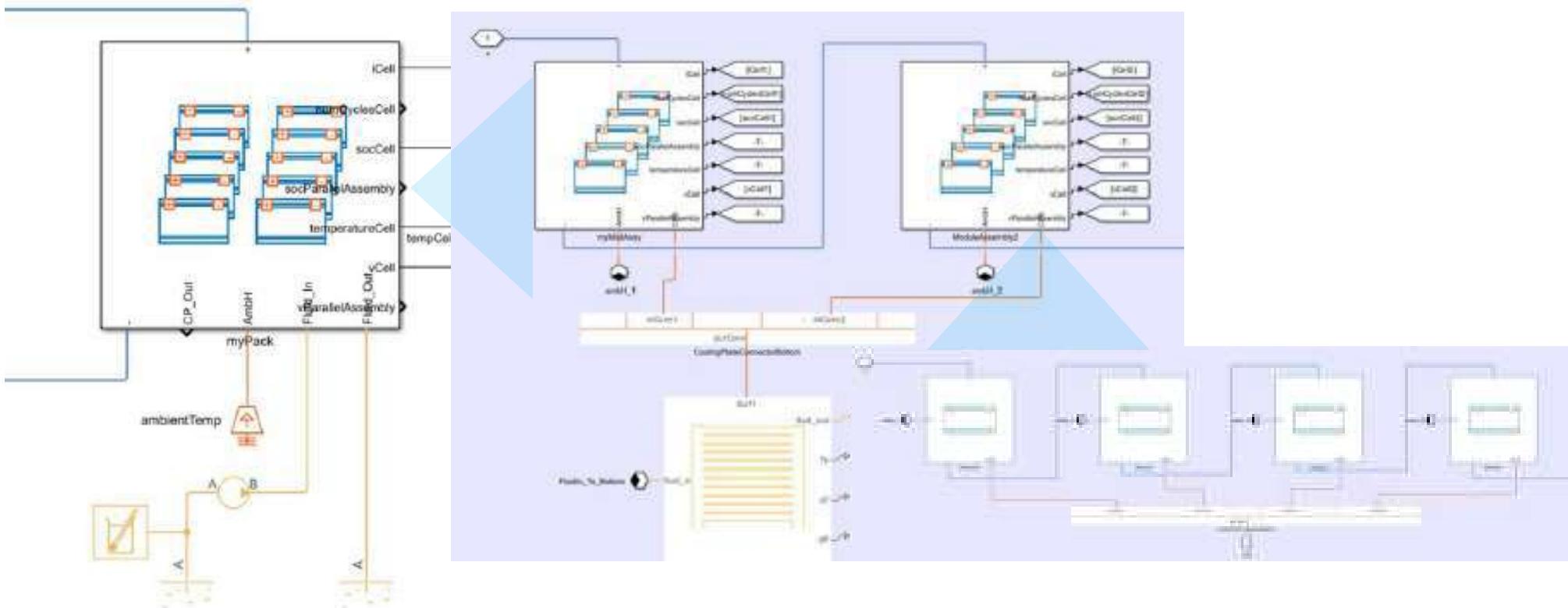
```
>> myPack_lib
```

Examine the generated blocks.

```
>> pack01_start
```

Add the battery pack into the model.

When creating the battery pack, the **Mask Parameters** (such as `V0`, `R0`) of each module inside the pack or module assembly can either be the same (`VariableNamesByType`) or per-instance specific (`VariableNamesByInstance`).



Battery Pack Modeling

Parallel Cooling

The Parallel Channels block models a cooling plate with multiple parallel channels and a pair of distributor pipes. Similar to the Edge Cooling block, the Parallel Channels block uses Thermal mass to represent the plate segment and Pipe (TL) to represent the coolant in the pipe. The number of plate segments is determined by number of partitions in X and Y dimension for the cooling plate. The number of Pipe(TL) block required for the Parallel Channels is determined by the combination of **Number of coolant channels**, **Select channel orientation direction**, and the plate segment partitions. Follow the steps to set up a parallel cooling plate with four cooling channels.

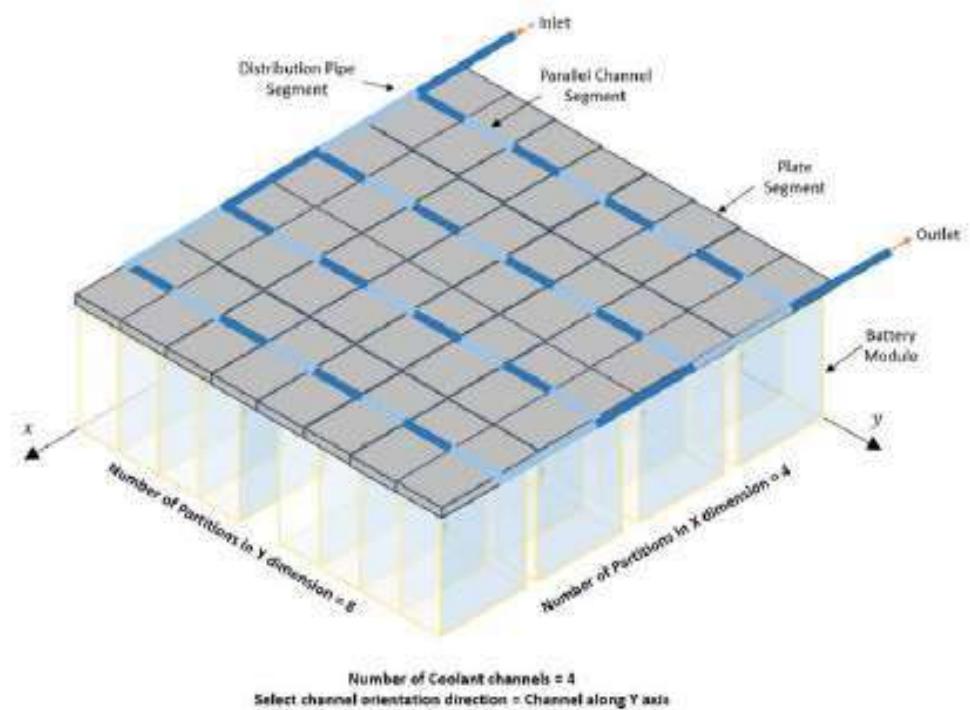
1. Go into myPack subsystem, and open the **Block Parameters** of **CoolingPlateBottom**.
2. Set the **Select channel orientation direction** as: **Channel along Y axis**.

Interface	
Number of partitions in X dimension for t...	CoolingPlateBottom.plateDimX
Number of partitions in Y dimension for t...	CoolingPlateBottom.plateDimY
Design	
Number of coolant channels	CoolingPlateBottom.nChannels
Select channel orientation direction	Channels along Y axis

In the given setup, four parallel channels go along the Y axis. Each parallel channel is divided into eight pipe segments given by Y dimension partitions. The distribution pipes go along the X axis. Each distribution pipe is divided into four pipe segments given by X dimension partitions.

Try

Follow the steps to configure the parallel cooling plate. Simulate the model.



When the number of partitions is greater than the number of coolant channels, each plate segment is connected to a separate pipe segment. When the number of partitions is smaller than the number of coolant channel, multiple pipes are mapped to the same plate segment. To get a desired temperature spread, it is important to select appropriate plate partitions to match with the number of coolant channels and the battery grouping strategies.

Battery Pack Modeling

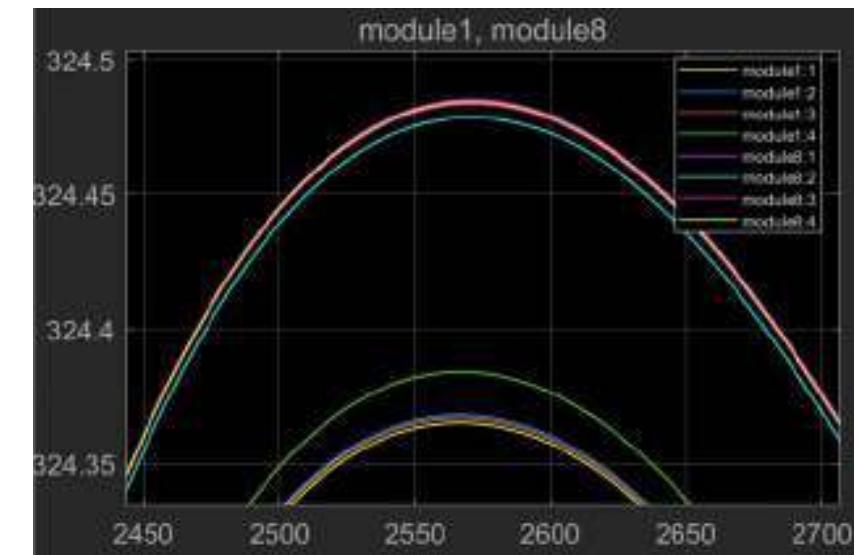
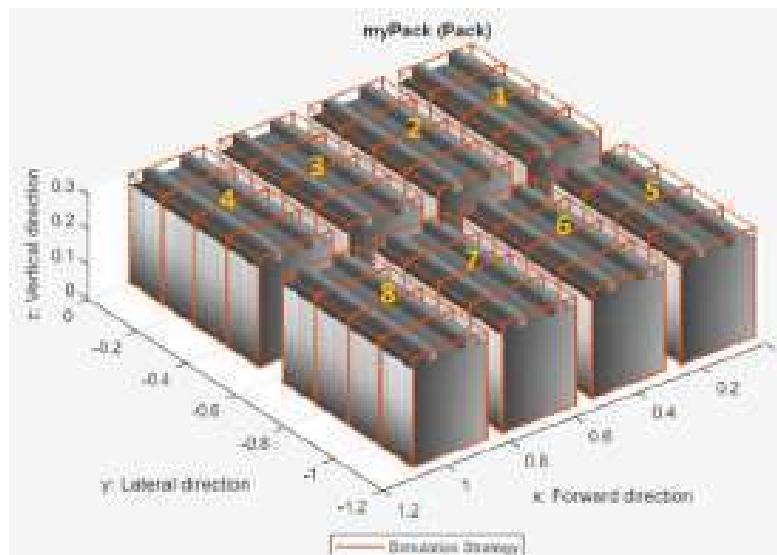
Parallel Cooling (Continued)

Simulate the model and examine the temperature differences between module one and module eight. Module one is closer to the inlet and module eight is closer to the outlet.

Try

```
>> pack02_parCool
```

Simulate the model and examine the temperature differences between modules.



Battery Pack Modeling

Summary

- Create battery module and pack objects with the Battery Builder app
- Control the model resolution of battery objects
- Add thermal fidelity to the battery objects
- Model cooling plates attached to the battery objects

Battery Pack Modeling

Test Your Knowledge

1. (T/F) When generating a battery pack with the Battery Builder app, a .ssc file is created to represent the pack.
2. (T/F) You can combine modules with different fidelities into a single module assembly.
3. (T/F) Setting the **CoolantThermalPath** option will provides you an array of cooling thermal nodes.

Answers

1. False
2. True
3. False



Battery Modeling and Algorithm Development with
Simulink®

State Estimation

State Estimation

Outline

- State of Charge estimation with Coulomb Counting
- State of Charge estimation with Extended Kalman Filter
- State of Health estimation
- State estimation of battery pack

The following names are trademarks of The MathWorks, Inc.:

MATLAB®; Simulink®; Simscape™ Battery™

State Estimation

Chapter Learning Outcomes

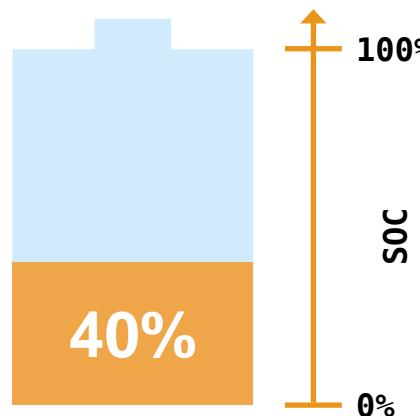
The attendee will be able to:

- Estimate the State of Charge with Coulomb Counting and Extended Kalman Filter.
- Estimate the State of Health of a battery based on capacity fade

State Estimation

State Estimation

State Of Charge (SOC) is defined as the ratio of the amount of extractable charge capacity available in the battery to its nominal rated capacity. It is similar to the fuel gauge of a conventional automobile since the SOC indicates the amount of electrical energy available to do work.

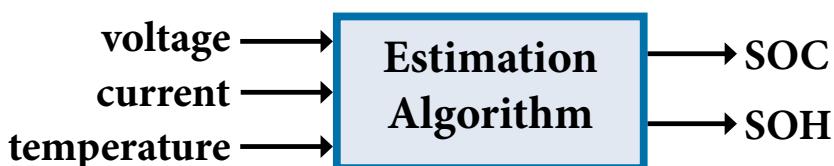
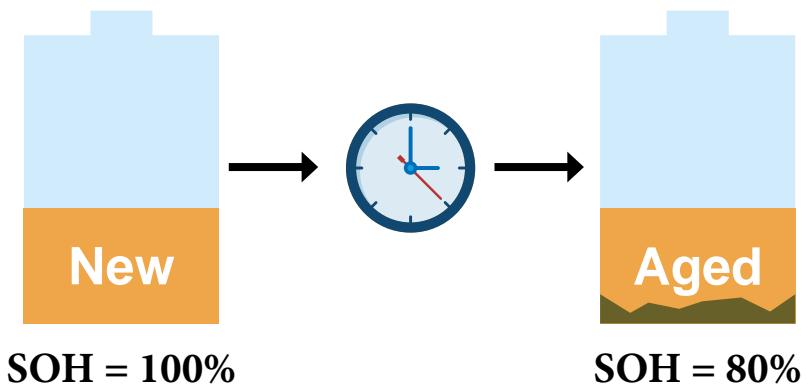


Accurate estimation of SOC increases user confidence to rely on the battery system, improves the performance of the battery pack, and helps optimize the operation of the battery pack to improve its longevity.

Since the amount of extractable capacity or charge of a cell cannot be measured directly by any means, the SOC of a battery cell must be estimated by making use of available current and voltage measurements. In this chapter, you will use Coulomb Counting and Extended Kalman Filter to measure the state of charge.

State of Health (SOH) provides indications of the level of degradation of a battery compared to its ideal condition. It is a crucial metric that determines the remaining lifespan and capacity. Typically, a new battery will be at 100% SOH at the time of manufacture. SOH gradually decreases as the battery degrades due to various factors, such as temperature, usage patterns, and charging habits. Monitoring the SOH of a battery is essential for optimizing its performance and making informed decisions regarding battery maintenance.

The SOH of a battery, much like SOC, is not directly measurable. Unlike SOC, however, SOH does not have a precise definition. Instead, it can be inferred through indicators such as capacity fade or growth of internal resistances. In this chapter, you will explore the use of state estimators that can capture the change in battery capacity and internal resistances. Then, SOH will be calculated based on these changes.



State Estimation

Estimator Blocks

Simscape Battery offers a variety of estimator blocks that facilitate the workflow of estimating SOC, terminal resistance, and the SOH by using coulomb counting and Kalman filter algorithms. You can find these blocks under the **Simscape > Battery > BMS > Estimators** library. Find below a summary of blocks listed in the library.

Coulomb counting based SOC estimation blocks:

- SOC Estimator (Coulomb Counting)
- SOC Estimator (Coulomb Counting, Variable Capacity)

Kalman filter based SOC estimation blocks:

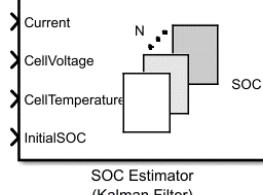
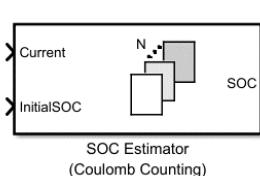
- SOC Estimator (Kalman Filter)
- SOC Estimator (Kalman Filter, Variable Capacity)

Internal to these blocks, you can select the extended Kalman filter or unscented Kalman filter as the filter type.

Adaptive Kalman filter based SOC estimation blocks:

- SOC Estimator (Adaptive Kalman Filter)
- SOC Estimator (Adaptive Kalman Filter, Variable Capacity)

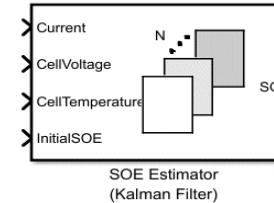
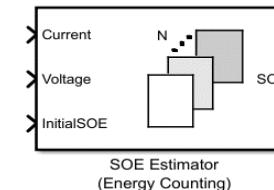
The adaptive Kalman filter considers the terminal resistance as a state of the estimator. It co-estimates the SOC and the change in terminal resistance.



Try

Examine the blocks under **Simscape > Battery > BMS > Estimators** library.

Similar to the SOC estimators, Simscape Battery also provides a set of State-Of-Energy (SOE) estimators. These blocks replace SOC with SOE as the state in the estimators.

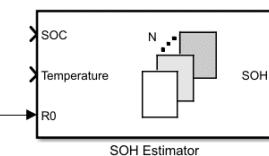
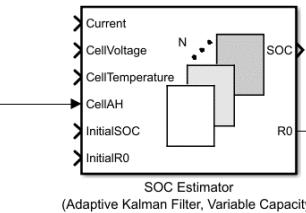
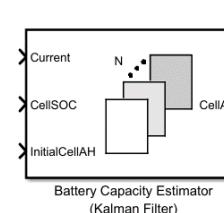


The SOH of a battery can be influenced by factors such as terminal resistance and capacity. Other than estimating terminal resistance with an adaptive Kalman filter, you can estimate the change in battery capacity with the Battery Capacity Estimator (Kalman Filter) block. The updated capacity can be used as an input to SOC estimator blocks that support variable capacity.

SOH estimation blocks:

- Battery Capacity Estimator (Kalman Filter)
- SOH Estimator
- SOH Estimator (Capacity-Based)

SOH Estimator (Capacity-Based) and SOH Estimator (based on terminal resistance) are blocks that convert the changes in battery characteristics into an SOH value based on lookup tables.



State Estimation

Coulomb Counting

The SOC of a battery cell is defined as the ratio of the present cell capacity $q(t)$ to its rated cell capacity q_{nom} .

$$SoC = \frac{q(t)}{q_{nom}}$$

Since the cell capacity cannot be measured directly, the Coulomb Counting algorithm makes use of battery current measurements to estimate the present charge level of the battery cell using the equation

$$q(t) = \int_0^t i(\tau) d\tau$$

where

$q(t)$ – accumulated charge

$i(t)$ – battery current as measured by the sensors

The BMS is a discrete system, executing tasks at a predefined sample time. The discretized equation for computing the state of charge is

$$SOC(k) = SOC(k-1) + \frac{T_s i(k)}{q_{nom}} \\ 0 \leq SOC(k) \leq 1$$

where

T_s – sample time

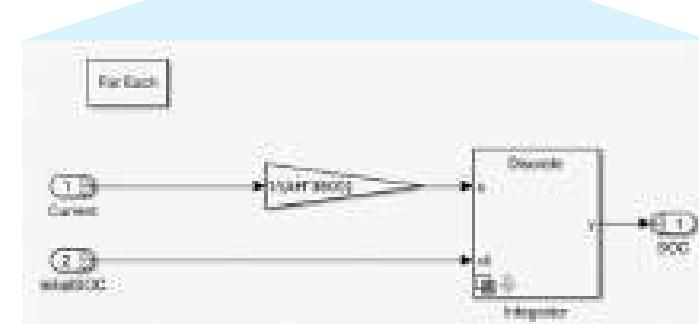
q_{nom} -- nominal cell capacity in Ampere-seconds

Try

>> cellSOCEst01_start

Insert a SOC Estimator (Coulomb Counting) block from the Simscape > Battery > BMS > Estimators library. Select the block and press **Ctrl+U** to look under the mask.

Simscape Battery offers a variety of state estimator blocks. Follow the Try box to insert a SOC Estimator (Coulomb Counting) block and examine the logic modeled.



State Estimation

Coulomb Counting (Continued)

Coulomb Counting algorithm requires knowledge of the battery capacity, sample time, and initial SOC condition.

1. Set the **Cell capacity** and **Sample time** of the SOC Estimator (Coulomb Counting) as **Capacity** and **Ts**.
2. Simulate the model.

Cell capacity, AH (A*Hr)
Capacity
Sample time (-1 for inherited)
Ts

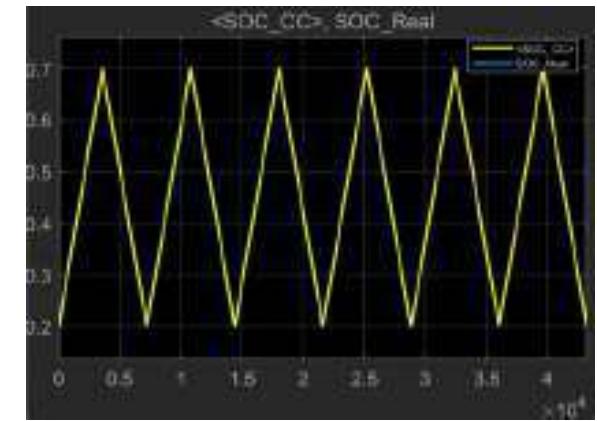
Coulomb Counting is a computationally inexpensive algorithm used to estimate the SOC of a battery cell; however, it has its limitations. The accuracy of the Coulomb Counting depends primarily on a precise estimate of the initial SOC. Starting with an arbitrary initial value results in the accumulation of integral errors over time and subsequently provide a poor estimate of the SOC. Coulomb Counting is also negatively impacted by measurement errors introduced by the sensors and the change in cell capacity due to component aging or cell self-discharging current.

1. Change **SOCGuess** to **0.3** from the MATLAB command line.
2. Simulate the model again.

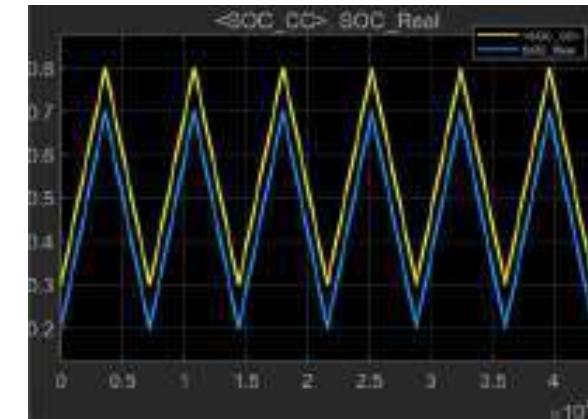
For more precise SOC estimation, the Coulomb Counting algorithm can be modified to include a periodic re-calibration by using voltage measurement. Alternatively, algorithms like a Kalman Filter (Extended Kalman Filter, Unscented Kalman Filter) can be used to provide an accurate estimate of the cell SOC.

Try

```
>> cellSOCEst02_CC
```



SOCGuess = 0.2



SOCGuess = 0.3

State Estimation

Kalman Filter

The Kalman Filter is an estimation algorithm that is used to obtain system state variables that cannot be directly measured. It is particularly effective in dealing with systems involving uncertainties and noisy measurements. The Kalman Filter combines information from model prediction and system measurement to make optimal state estimation. It is based on several assumptions:

- The Kalman Filter requires a linear model to describe state variation and measurement equations.
- The noises are assumed to be independent and have normal distributions.

Given a linear plant model described by the state update and measurement functions:

$$\begin{aligned}\dot{x} &= Ax + Bu + w \\ y &= Cx + Du + v\end{aligned}$$

Where

- x is the state of the system
- y is the system output
- u is the system input
- w is the process noise that represents the uncertainties not captured in the system dynamic model.
- v is the measurement noise of the sensor.

The process noise w and the measurement noise v follow normal distributions.

$$\begin{aligned}p(w) &\sim N(0, Q) \\ p(v) &\sim N(0, R)\end{aligned}$$

Where

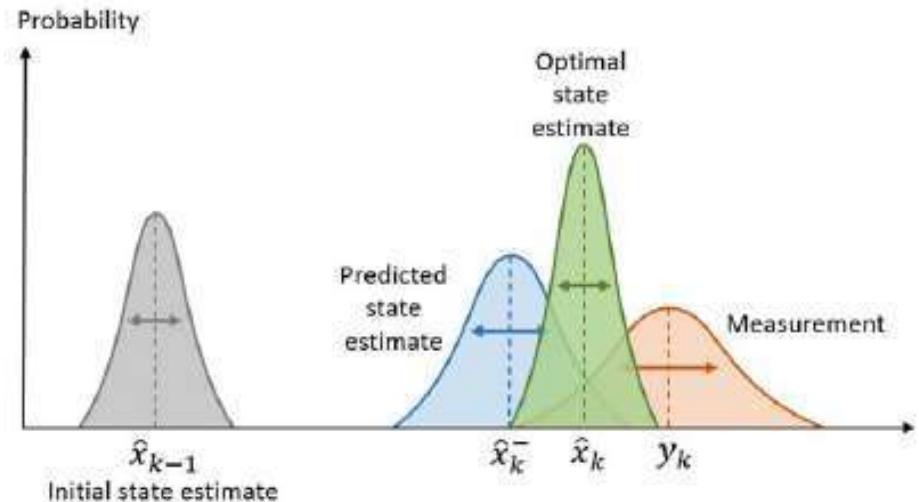
- Q is the process noise covariance
- R is the measurement noise covariance

Since the true state values are not directly observable, they are also treated as random variables following normal distribution. In the initial step, you will need to provide the initial state values $x(0)$ and the error covariance P_0 .

The Kalman Filter process is usually separated into two steps:

1. **Predict:** the Kalman Filter uses the system dynamic model and the current state value to predict the state of the next time step.
2. **Update:** the Kalman Filter compares the prediction with the actual sensor measurement, and corrects the prediction based on the measurement.

Throughout the predict-update process, the covariances of the estimated states are also updated and propagated along with the state values. The goal of the Kalman Filter is to minimize these covariances, which represent the uncertainty in the estimated states, aiming to converge towards more deterministic and accurate state values.



To learn more about the Kalman Filter, please refer to Appendix *Kalman Filter Design in Simulink*.

State Estimation

SOC Estimation Using Kalman Filter

Though the Kalman Filter is a powerful tool, it cannot be directly applied to battery SOC estimation as the equivalent circuit model and the parameters are not linear. If a simplified linear equivalent circuit model is produced, the Kalman Filter can be applied.

Consider an oversimplified case where we only consider a constant internal resistance R_0 and ignore all RC dynamics. Meanwhile, the SOC-OCV relationship is also linearized. The linear system equation can be written as:

$$SOC(k) = SOC(k-1) + \frac{T_i}{q_{nom}} i(k)$$

$$V(k) = mSOC(k) + R_0 i(k)$$

Where

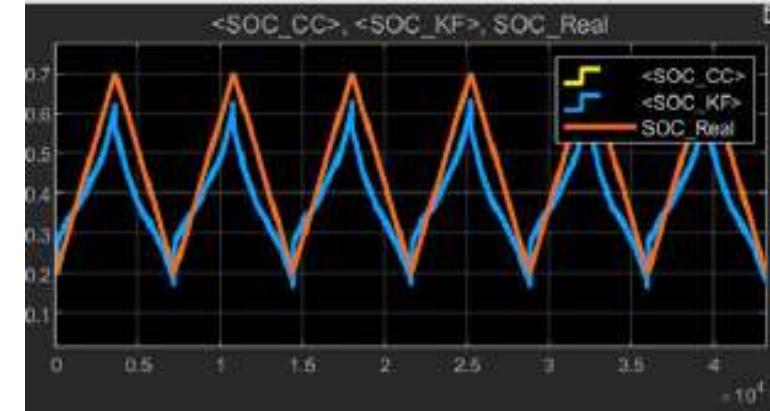
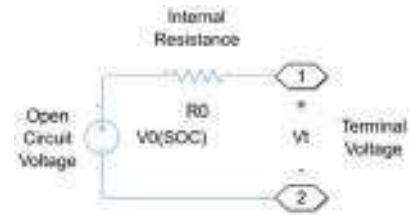
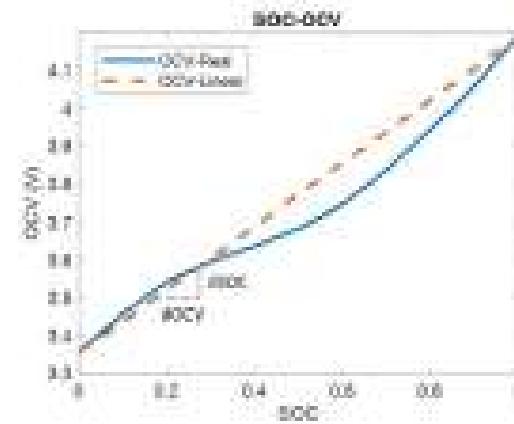
- m is the slope of the SOC-OCV curve
- $V(k)$ is the adjusted voltage measurement, $V(k) = V_c(k) - 3.35$

Follow the Try box to simulate a model using Kalman Filter to estimate SOC. The result is poor due to the inherent nonlinearity of the model. In the next step, a variation or extension of the Kalman Filter called the Extended Kalman Filter will be used for estimating the states of a nonlinear dynamic system.

Try

Open and simulate the model

>> simpleKF



State Estimation

SOC Estimation Using Extended Kalman Filter

The Extended Kalman Filter is modified from the classical Kalman Filter to adapt to nonlinear models. Instead of linearizing the entire plant model, it works by linearizing the nonlinear system at its operating point. Given a nonlinear system characterized by the equations:

$$\begin{aligned}\dot{x} &= f(x, u) \\ y &= h(x)\end{aligned}$$

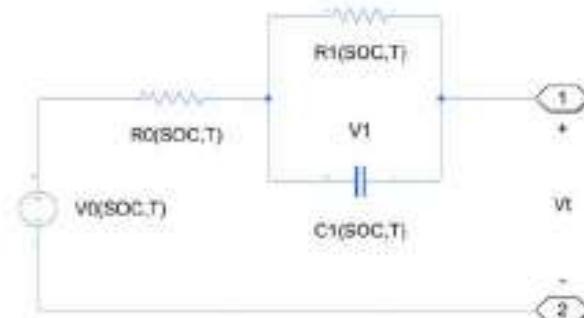
where the evolution of the states is governed by a nonlinear function f and h is a nonlinear function mapping states to output. A discrete linear model is extracted from this system at different points of the state-space by computing Jacobians. The linearized state-space representation can be described using the equations:

$$x(k+1) = Fx(k) + Gu(k)$$

$$y(k) = Hx(k)$$

$$F = \frac{\partial f(x, u)}{\partial x} \Big|_{x=x_k}, G = \frac{\partial f(x, u)}{\partial x} \Big|_{u=u_k}, H = \frac{\partial h(x, u)}{\partial x} \Big|_{x=x_k}$$

With this linearized discrete model, the predict-update steps of the classical Kalman Filter are applied. This forms the basis of the Extended Kalman Filter algorithm.



Try

>> cellSOCEst01_start

Insert a SOC Estimator (Kalman Filter) block from the Simscape > Battery > BMS > Estimators library. Select the block and press **Ctrl+U** to look under the mask. Examine the EKF structure and the Jacobian implementation.

In Simscape Battery, the SOC Estimator (Kalman Filter) block implements an EKF using a 1RC equivalent circuit model. The state transition and output equations can be written as:

$$\begin{aligned}\frac{dSOC}{dt} &= -\frac{i}{3600AH} \\ \frac{dV_1}{dt} &= \frac{i}{C_1(SOC, T)} - \frac{V_1}{R_1(SOC, T)C_1(SOC, T)} \\ V_t &= V_0(SOC, T) - iR_0 - V_1\end{aligned}$$

Translating these into state-space form:

$$\begin{aligned}x &= [SOC \quad V_1]^T \\ f(x, i) &= \begin{bmatrix} 0 & -\frac{i}{3600AH} \\ \frac{i}{C_1(SOC, T)} - \frac{V_1}{R_1(SOC, T)C_1(SOC, T)} & 0 \end{bmatrix} \\ h(x, i) &= V_0(SOC, T) - iR_0 - V_1\end{aligned}$$

Then at every time step, the EKF algorithm computes these Jacobians to linearize the plant model:

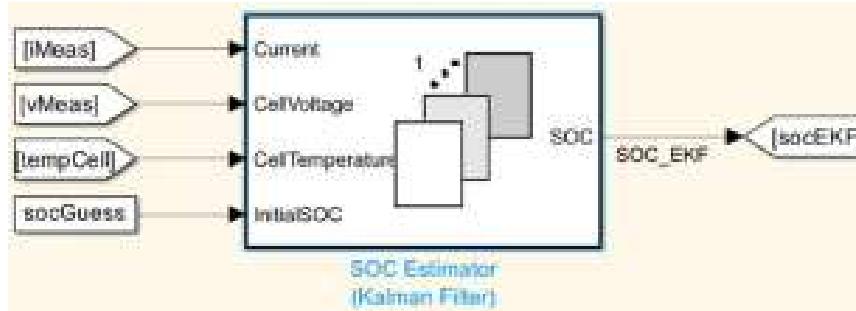
$$\begin{aligned}F &= \frac{\partial f}{\partial x} \\ H &= \frac{\partial h}{\partial x} \\ F_d &= \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{-T_S}{R_1C_1}} \end{bmatrix} \\ H_d &= \begin{bmatrix} \frac{\partial V_0}{\partial SOC} & -1 \end{bmatrix}\end{aligned}$$

State Estimation

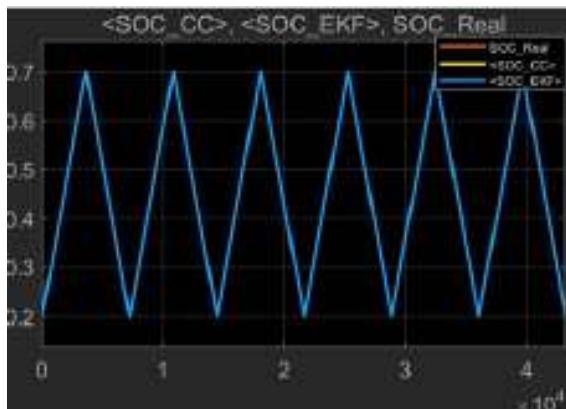
SOC Estimation Using Extended Kalman Filter (Continued)

To accurately use the nonlinear system model in the EKF, you will provide the table-based battery parameters under the **Block Parameters > System Model** tab. Then, under the **Main** tab, you will provide the covariance matrices and sample time information. With two states [SOC, V1] and one measurement V_t , the dimensions of the covariance matrices are:

- Process noise covariance Q : 2x2
- Measurement noise covariance R : 1x1
- Initial state error covariance P_0 : 2x2



Simulate the model, the SOC estimated by EKF nicely converges with the real SOC.



Try

Follow the screenshot to setup the **Block Parameters** of the SOC Estimator(Kalman Filter) block.

Simulate the model.

Main System Model

Specify Current input as cell current(s)

Filter type Extended Kalman filter

Covariance of the process noise, Q

Q

Covariance of the measurement noise, R

R

Initial state error covariance, P0

P0

Sample time (-1 for inherited)

Ts

Main System Model

Charge dynamics One time-constant dynamics

Vector of state-of-charge values, SOC (-)

SOC

Vector of temperatures, T

T_mat

Terminal resistance, R0(SOC,T), (ohm)

R0_mat

First polarization resistance, R1(SOC,T), (ohm)

R1_mat

First time constant, tau1(SOC,T), (s)

tau1_mat

No-load voltage, V0(SOC,T), (V)

V0_mat

Cell capacity, AH (A*Hr)

Capacity

State Estimation

Tuning Extended Kalman Filter

Tuning an EKF requires careful selection of the covariance matrices based on understanding of the underlying system dynamics and noise characteristics. This process often involves rounds of experimentation and trial and error. A well-tuned EKF should provide rapid convergence to an accurate state estimation. Here is a summary of how different covariance matrices impact the performance of EKF.

- Process Noise Covariance Q**

The process noise of a system represents the disturbances that are not addressed by the system dynamic model. For example, using a battery equivalent circuit model with no RC pairs results in larger process noise compared to the 1RC model. The variance in the state values due to the addition of process noise is represented by the process noise covariance matrix Q . A smaller Q implies a higher level of confidence in the accuracy of the system dynamic model. In such cases, the EKF will place more emphasis on the model prediction.

- Measurement Noise Covariance R**

When measurements are taken, inaccurate sensor readings are inevitably included. The measurement noise covariance R indicates the spread of the measurement noise. Typically these values can be found on the sensor's datasheet. A small R value indicates high confidence in the sensor's readings.

- Initial State Error Covariance P0**

The initial state error covariance represents the confidence in the initial guess of the system states. Compared to Coulomb Counting, EKF has the ability to correct inaccurate initial SOC guesses. A big value in P_0 indicates low confidence in initial state error.

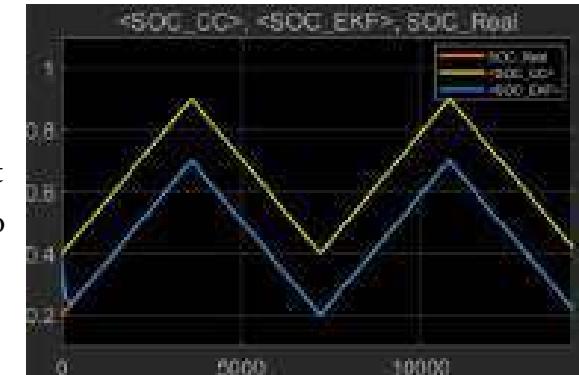
Try

```
>> edit cellSOCEst01_script
```

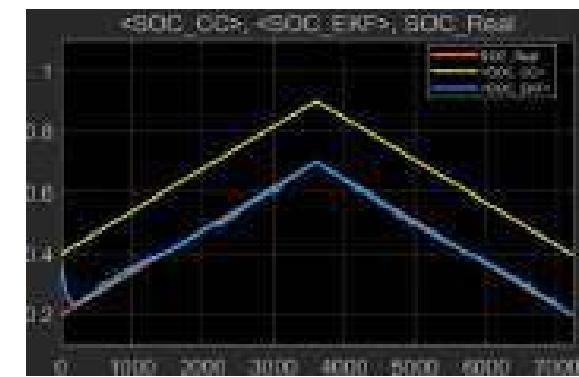
Open the script and change the values of the covariance matrices, examine the change in the EKF estimation output.

In practice, determining the appropriate values for covariance matrices often involves iterative experimentation. For SOC estimation, it is critical that you record accurate reference data, such as SOC obtained from high-precision Coulomb Counting, to serve as a benchmark during the tuning process. Additionally, it is important to validate the tuned EKF using various test cases that encompass all potential scenarios. This validation ensures the reliability of the EKF under a range of conditions.

- SOCGuess = 0.4**
- SOC_CC keeps the offset**
- SOC_EKF converges to the correct value**



- Diag of Q = 1e-3**
- Diag of R = 1e-5**
- SOC_EKF tends to prioritize the measurement over the model prediction**

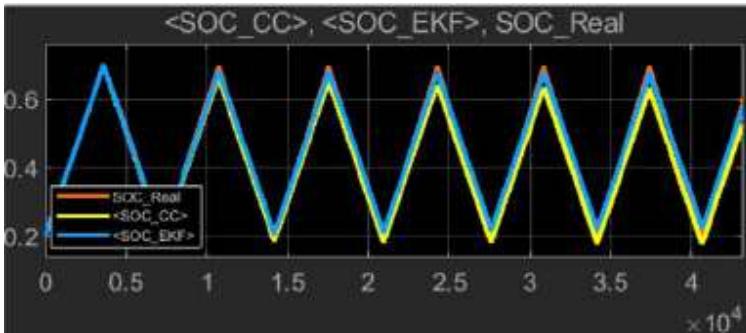


State Estimation

SOH Estimation

By far you have successfully implemented SOC estimation with EKF. However, the SOC Estimator (Kalman Filter) block assumes constant battery capacity. While in reality, the capacity fades as the battery ages, thus lowering the credibility of the system model.

1. In model `cellSOCEst01_start.slx`, change the Fade characteristic of the Battery (Table-Based) block to Equations.
2. Run the simulation and examine the SOC estimation performance of Coulomb Counting and EKF. The SOC estimation results become poor as the battery ages.



To address this issue, Simscape Battery provides a SOC Estimator (Kalman Filter, Variable Capacity) block where the value of capacity can be fed as an input to the block.

1. Add a SOC Estimator (Kalman Filter, Variable Capacity) block into the model.
2. In **Block Parameters**, set the **System Model** parameters as **SOC**, **T_mat**, **R0_mat**, **R1_mat**, **taul1_mat**, **V0_mat** accordingly.
3. Change the process noise, measurement noise, and initial state error covariances to **Q_SOC**, **R_SOC**, and **P0_SOC**.
4. Change the **Sample time** to **Ts**.

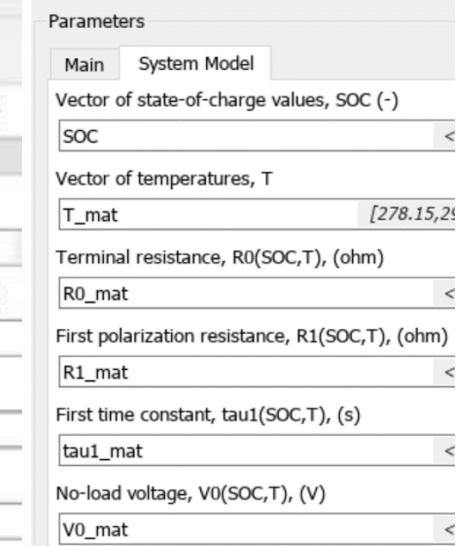
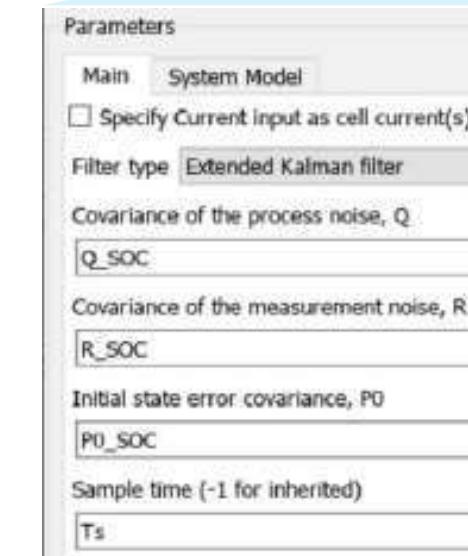
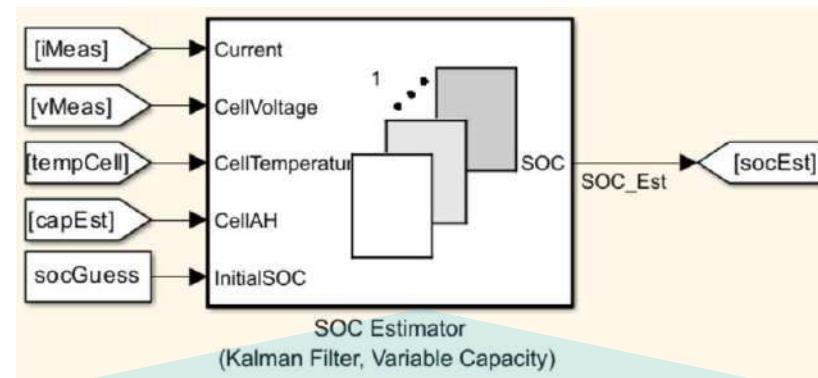
Try

`>> cellSOCEst01_start`

Follow the step to examine the SOC estimation performance with an aged battery.

`>> cellSOHEstCap01_start`

Add a SOC Estimator (Kalman Filter, Variable Capacity) block and set up the parameters.



State Estimation

SOH Estimation (Continued)

To work in pairs with the SOC estimator, a capacitor estimator is needed. As a state estimator, the Kalman Filter can also be applied for capacity estimation. Considering the battery capacity as the unknown state, the system equation can be written as:

$$\frac{dQ}{dt} = 0$$

$$0 = \frac{dSOC}{dt} + \frac{i}{3600Q}$$

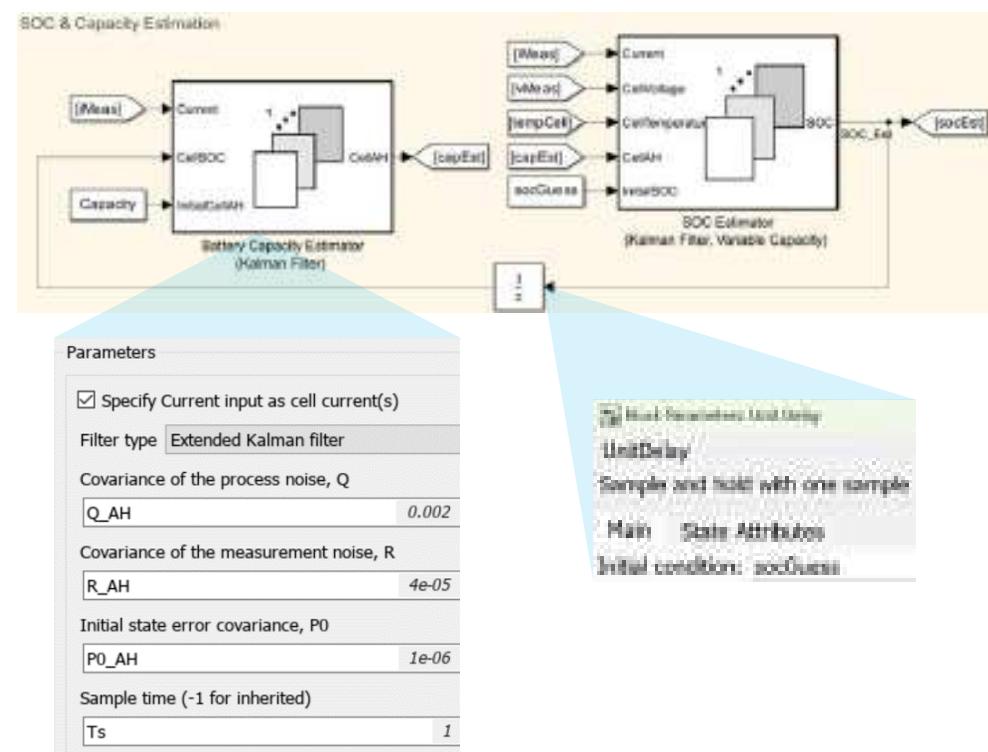
Where Q is the capacity in Ampere-hour

In Simscape Battery, the Battery Capacity Estimator (Kalman Filter) block implements the equations above.

1. Add a Battery Capacity Estimator (Kalman Filter) block.
2. Change the process noise, measurement noise, and initial state error covariances to Q_AH , R_AH , and $P0_AH$.
3. Change the **Sample time** to Ts .
4. Connect the two estimator blocks as shown in the figure.
5. Insert a Unit Delay block to break the algebraic loop. Set the **Initial condition** of the block as **SOCGuess**.
6. Simulate the model.

Try

Follow the steps to add the capacity estimation logic.



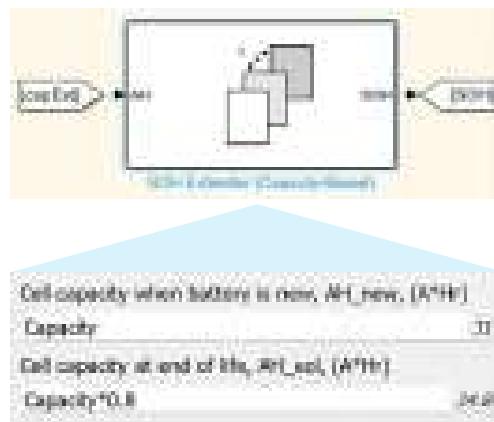
When tuning the covariance matrices of the Battery Capacity Estimator (Kalman Filter) block, it is important to note that the system equation denotes the battery capacity remains relatively constant. Therefore, the capacity update is primarily driven by the process noise covariance. To allow for more flexibility in accommodating changes, a larger covariance value should be given.

State Estimation

SOH Estimation (Continued)

The degradation in battery capacity is commonly used as an indicator of the change in the battery's State of Health. After estimating the capacity, you can further feed the estimated capacity into a SOH Estimator (Capacity-Based) block to derive the SOH.

1. Add a SOH Estimator (Capacity-Based) block into the model.
2. Connect the block and set the Block Parameters as shown in the figure.
3. Simulate the model.



This block calculates the SOH by comparing the capacity at the current time with the capacity recorded when the battery is new.

$$SOH_Q = \frac{Q - Q_{eol}}{Q_{new} - Q_{eol}}$$

Where

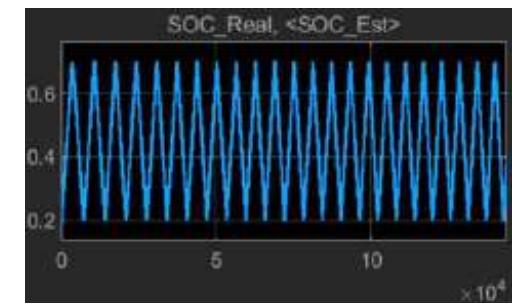
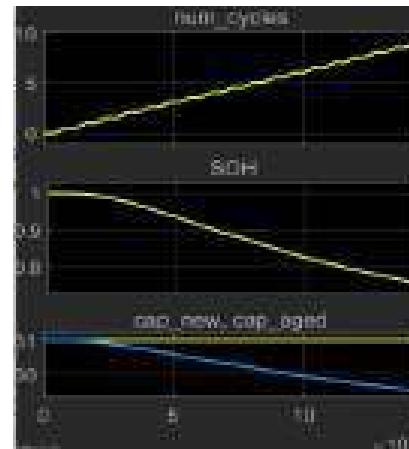
- Q_{new} is the capacity when the battery is new.
- Q_{eol} is the capacity at the end-of-life (EOL)

Usually, if a battery capacity decreases to 80% of its nominal value, it reaches the EOL. The value of SOH is 1 at the beginning of life and 0 at the EOL. In this simulated setup, the battery will reach the EOL after 10 cycles.

Try

Follow the steps to calculate the SOH based on capacity fade.

>> cellSOHEstCap02_soln



With the correction of capacity estimation, the SOC estimation coming out from EKF has improved. Other than using capacity as an SOH indicator, you can also use changes in internal resistance for a similar purpose. Simscape Battery also provides a SOC Estimator (Adaptive Kalman Filter) block that includes R_0 as one extra state. You can refer to the documentation example **Battery State-of-Health Estimation** for more information.

State Estimation

States Estimation of Battery Pack

When a battery pack contains multiple cells, it is important to monitor the individual states of each cell to gain valuable insights into energy distribution and potential imbalance. Cells in a battery pack could age at different rates due to temperature differences, overcharging, and over-discharging. By closely monitoring the states of individual cells, potential issues can be detected early, allowing for preventive measures or immediate actions to mitigate risks.

The estimators provided by Simscape Battery support vector inputs. To estimate the states of cells in a battery pack, you should feed the block with vectors of cell measurements and initial states. Internally, a For Each subsystem is implemented to calculate the individual states of each cell.

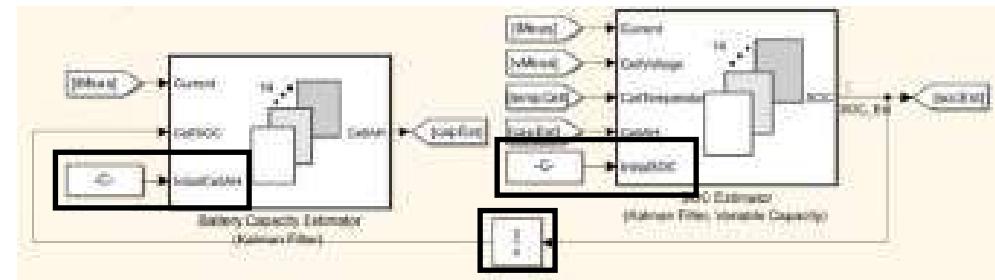
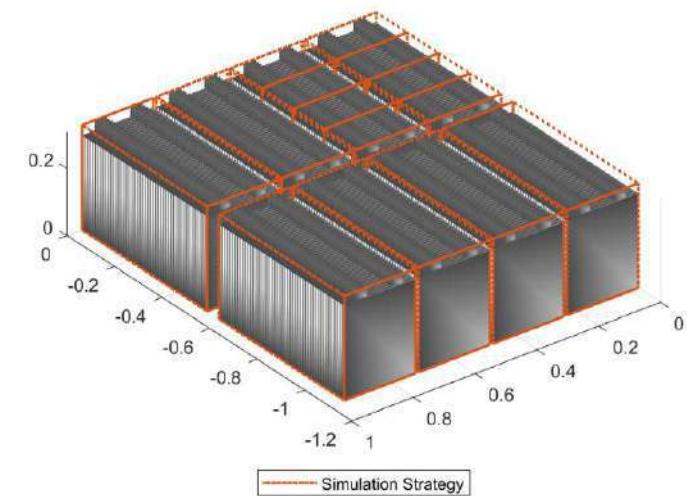
1. Open model `packStatesEst01_start.slx`.
2. Check the checkbox of **Specify Current input as cell current(s)** in both of the estimator blocks.
3. Change the value of the Constant block connected to **InitialCellAH** to `repmat(Capacity, 14, 1)`.
4. Change the value of the Constant block connected to **InitialSOC** to `repmat(SOCGuess, 14, 1)`.
5. Change the **Initial condition** of the Unit Delay to `repmat(SOCGuess, 14, 1)`.
6. Simulate the model.

The aforementioned model simulates a battery pack with temperature fading enabled. The battery pack is placed in an ambient environment without active cooling. Cells at different locations may experience varying rates of aging as a result of temperature differences. To optimize simulation time, a total of 14 cell models are utilized. The first two modules employ grouped fidelity, while the remaining modules are lumped.

Try

```
>> packStatesEst01_start
```

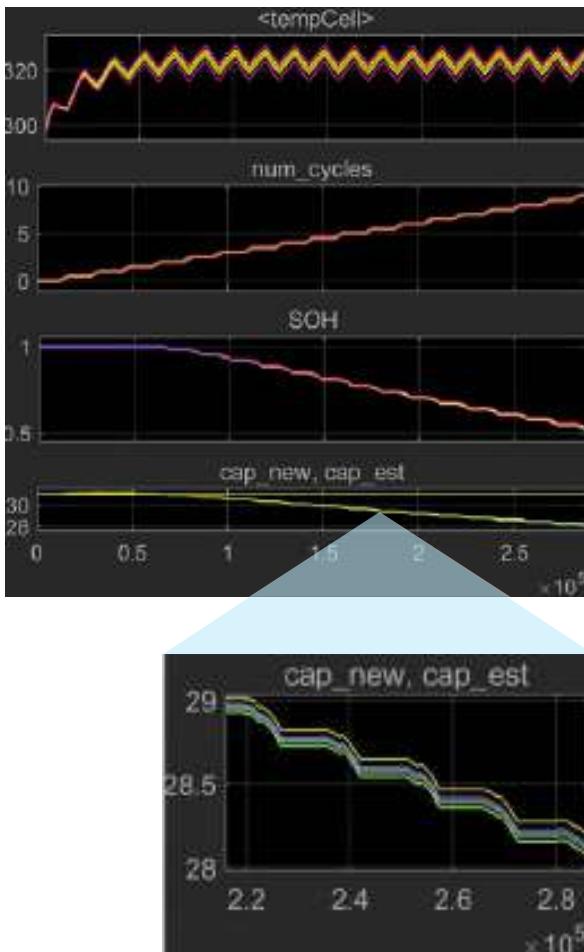
Follow the steps to provide vector inputs to the estimator blocks.



State Estimation

States Estimation of Battery Pack (Continued)

A variance in ambient thermal path resistances is given to the first two modules to produce temperature spread. Consequently, cells subjected to higher temperatures tend to age faster than the rest of the cells.



Try

Analyze the simulation result.

```
>> packStatesEst02_soln
```

Zooming into the SOC estimation curve, despite receiving the same amount of current input, each cell experiences different changes in SOC due to its varying capacity. This discrepancy results in an imbalanced battery pack, where the performance is constrained by the cell with the smallest capacity. To address this issue, it is crucial to consider active cooling and balancing to mitigate the problem.



State Estimation

Summary

- State of Charge estimation with Coulomb Counting
- State of Charge estimation with Extended Kalman Filter
- State of Health estimation
- State estimation of battery pack

State Estimation

Test Your Knowledge

1. (T/F) Using a simple Kalman Filter with a linear battery model is sufficient for getting good estimates of the SOC of a battery cell.
2. (T/F) In EKF, a big process noise covariance indicates high confidence in the system model.
3. (Select all that apply) What are some of the factors that Coulomb Counting is prone to but can be addressed by Kalman Filter?
 - A. Measurement noise
 - B. Battery fading
 - C. Unknown initial condition

State Estimation

Answers

1. False
2. False
3. ABC



Battery Modeling and Algorithm Development with
Simulink®

Battery Management System

Outline

- Overview of the battery management system
- Design Stateflow logic to charge a cell using CC-CV control scheme
- Design supervisory control logic of battery management system using Stateflow
- Implement a passive cell balancing network
- Create test scenarios for battery management system using Simulink Test

The following names are trademarks of The MathWorks, Inc.:
Simulink®; Stateflow®; Simscape™ Battery™; Simulink Test™

Chapter Learning Outcomes

The attendee will be able to:

- Describe the key functionalities of a battery management system
- Model supervisory control logic with Stateflow
- Model passive cell balancing
- Define test input with the Test Sequence block

Overview of the Battery Management System

Lithium-ion battery packs are the predominant energy storage systems in aircraft, electric vehicles, portable devices, and other equipment requiring a reliable, high-energy-density, low-weight power source. The battery management system (BMS) is responsible for safe operation, performance, and battery life under diverse charge-discharge and environmental conditions.

When designing a BMS, engineers develop feedback and supervisory control that

- Monitors cell voltage, current, and temperature.
- Estimates state-of-charge and state-of-health.
- Controls the charging profile.
- Balances the state-of-charge of individual cells.
- Limits power input and output for thermal and overcharge protection.
- Isolates the battery pack from the load when necessary.

Battery Management System

Designing the BMS Algorithms

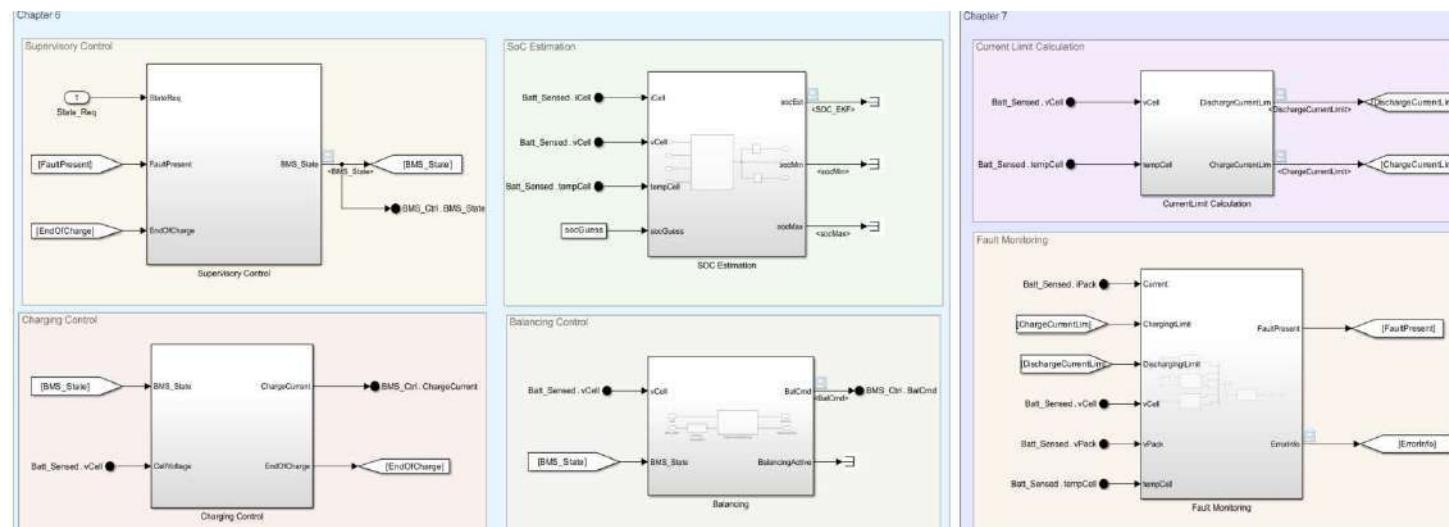
Starting from this chapter the focus will be on designing different control algorithms that are required in a BMS. The contents include:

Chapter 6 – Battery Management System

Covers the modeling of the supervisory control logic to decide the BMS operation modes and discusses how to build the charging and discharging logic with Stateflow. It also covers the design of a passive cell balancing algorithm.

Chapter 7 – Fault Diagnostics and Current Limit Calculation

Focuses on computing battery pack's charging and discharging current limits. It also covers fault detection (sensor fault and voltage fault, etc.) during the battery pack operation.



Try

Open the BMS template model and examine each component.

>> BMS01_start

Required Toolboxes

Simscape Battery - Simscape Battery provides prebuild BMS blocks for cell balancing, current management, and protection etc. You will learn how these blocks facilitate the BMS design.

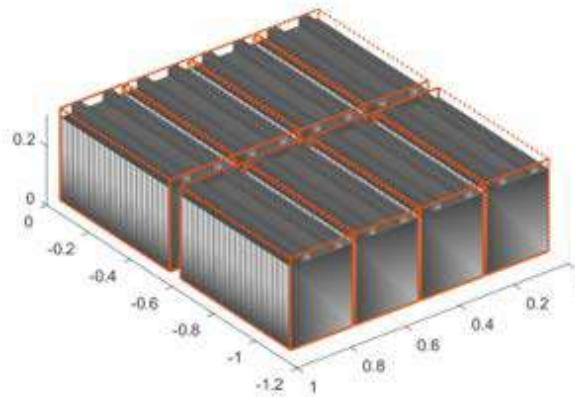
Stateflow - Instead of using prebuild blocks, Stateflow enables you to describe custom complex logic-driven systems in a graphical way. You will use Stateflow to implement the supervisory control logic.

Simulink Test - Simulink Test provides you blocks to easily create time-based test cases. You will use blocks from the Simulink Test library to provide input to the system.

Battery Management System

Battery Plant

In chapter four, you have learned how to create battery pack with different fidelities. For simplicity and avoid signal congestion, the battery pack implemented in this example is lumped with eight cell models. Each module is represented by one cell model.

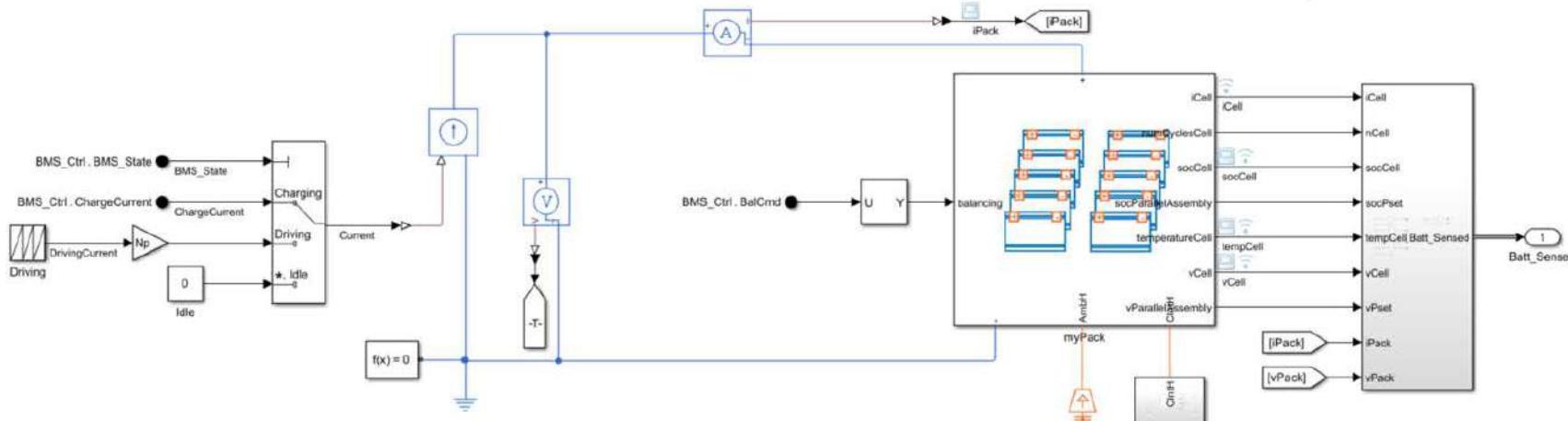


Try

Navigate to the **Plant** subsystem to examine the battery plant.

Additional to the signals produced by each cell model, pack level current and voltage are monitored. These signals are combined into the **Batt_Sensed** bus to be provided to the BMS.

Depending on the user request, the selected current profile should be given to the battery. A Multiport Switch is implemented to select the current profile based on the **BMS_State**. The driving current profile is given by the Repeating Sequence block running UDDS profile repeatedly. The charging current will be calculated by the BMS. You will learn to model the charging logic and setup **BMS_State** in the next step.



Battery Management System

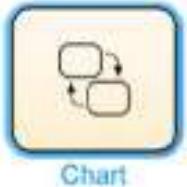
Constructing the Charging Logic with Stateflow

Unlike discharging a battery by running the load profile, the required charging current is usually decided by the BMS. Based on the SOC, voltage, etc., the BMS should control the current to prevent the battery from being overcharged, while trying to charge the battery to its full capacity.

In chapter two, you have implemented the CC-CV charging logic by using the Charger block provided by Simscape Battery. However, that is a Simscape block which is not optimal for BMS code generation. Instead, you can use Stateflow to easily set up a state machine with a finite number of modes. Go into the **Charging Control** subsystem and build the charging logic with Stateflow.

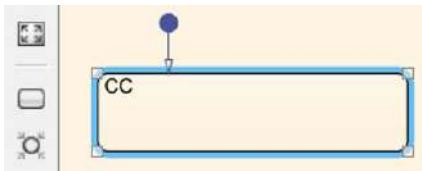
Creating a new chart

1. Add a Chart block from the Stateflow library to the model.



Adding states

2. Use the State toolbar button on the left to add two states into the model.



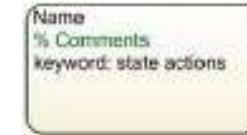
Try

Follow the steps to create the CC-CV charging logic.

Labeling the states

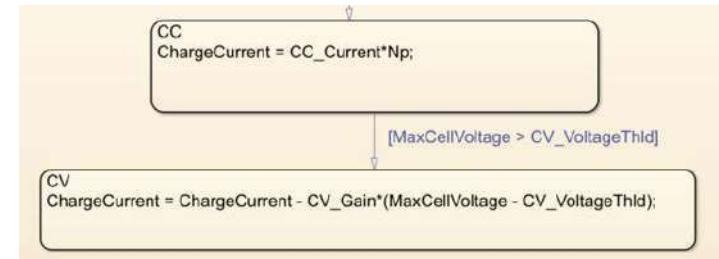
A generic state label consists of state name and state actions. The syntax is:

Note: The state action keyword controls when the actions are carried out.



Commonly used keywords are

- **entry** (or **en**) – Executes upon entering the state
 - **exit** (or **ex**) – Executes upon exiting the state
 - **during** (or **du**) – Executes when the state starts as active and remains active (no transition occurs)
3. Label the states as CC and CV.
 4. Give the corresponding state actions to the CC and CV states.
 - In the CC charging state, the charging current equals a constant of **CC_Current**.
 - In the CV charging state, the charging current is adjusted by the voltage difference between the maximum cell voltage and the voltage threshold.



Battery Management System

Constructing the Charging Logic with Stateflow (Continued)

Adding transitions

When the maximum cell voltage reaches the voltage threshold, the BMS should switch from CC charging to CV charging to prevent overvoltage. In Stateflow, you can create transitions to go from one state to another and provide transition labels to guard when the transition is valid.

5. Move your mouse to the edge of the CC state and drag it towards the edge of the CV state.
6. Double-click the transition to enable editing. Add `MaxCellVoltage > CV_VoltageThld` as the transition condition. Transition condition should be enclosed in [].

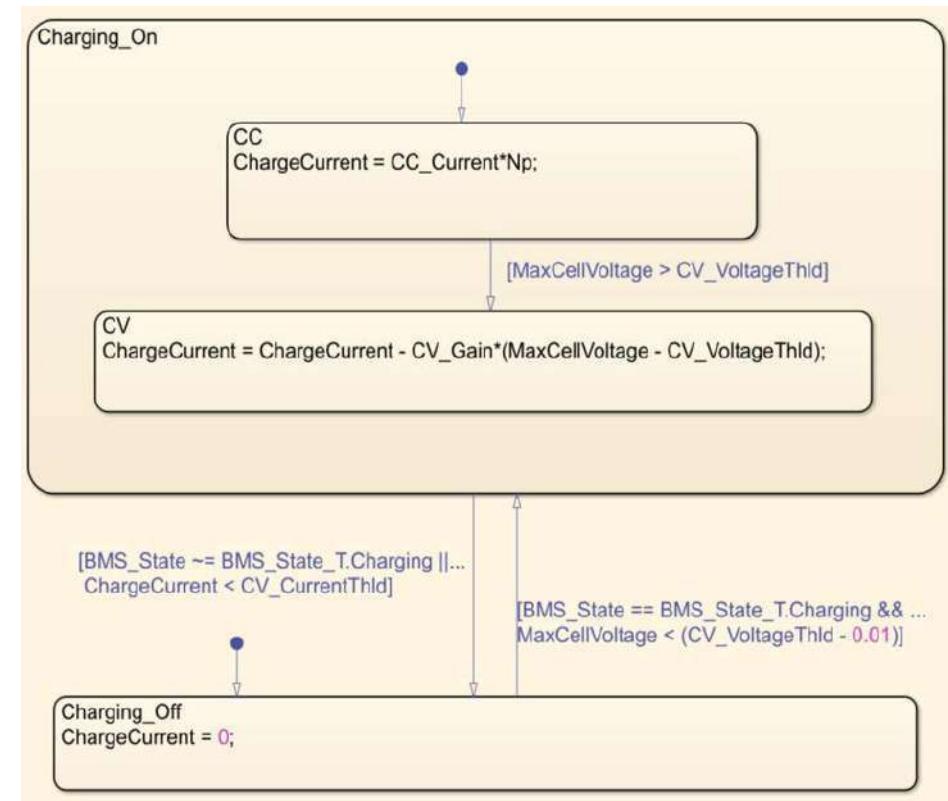
Creating super states

The CC-CV charging should only be enabled when the BMS_State is under Charging. You can add the extra layer of charging On/Off logic by adding hierarchies into the existing chart.

7. Select the CC and CV states and **Create Superstate** from the selection. Name the superstate **Charging_On**.
8. Create another state called **Charging_Off**. Give `ChargeCurrent = 0;` as the state action.
9. Add transitions between the **Charging_Off** and **Charging_On** states.
 - **Off to On:** when `BMS_State` is `Charging` and when `MaxCellVoltage` is smaller than `CV_VoltageThld - 0.01`
 - **On to Off:** when `BMS_State` is not `Charging` or when the end of CV charging is reached (`ChargeCurrent` is smaller than the cutoff current threshold).

Try

Follow the steps to create the CC-CV charging logic.



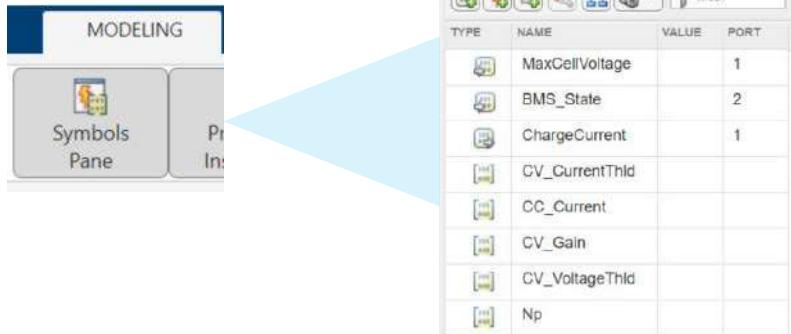
Battery Management System

Constructing the Charging Logic with Stateflow (Continued)

Defining chart data

In Stateflow, you must define the variable names used in a chart as chart data. You can use the **Symbols Pane** to create them.

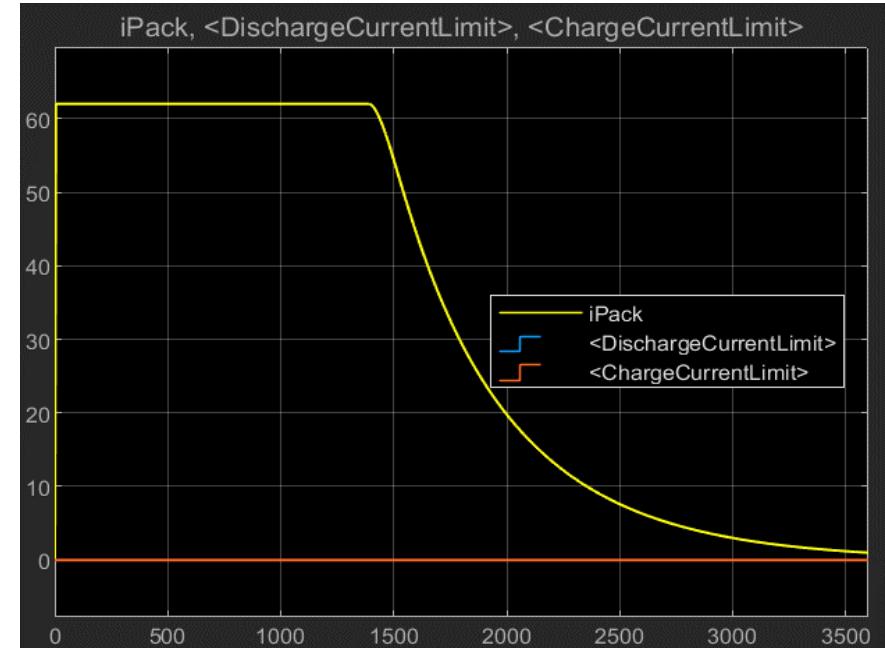
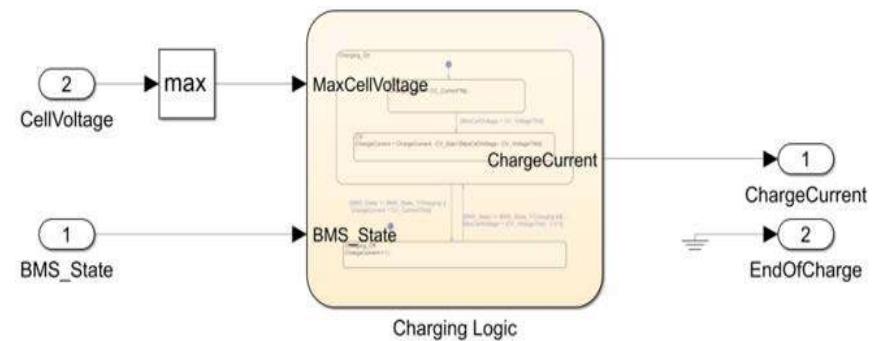
1. Open the **Symbols Pane** under the **Modeling** tab, **Design Data** section.
2. Define the data with the following scope.
 - Input Data: **MaxCellVoltage BMS_State**
 - Output Data: **ChargeCurrent**
 - Parameter Data: **CC_Current, CV_Gain, CV_VoltageThld, CV_CurrentThld, Np**
3. Connect the input/output signal lines outside the chart.
4. Simulate the model.



Try

Follow the steps to create the CC-CV charging logic.

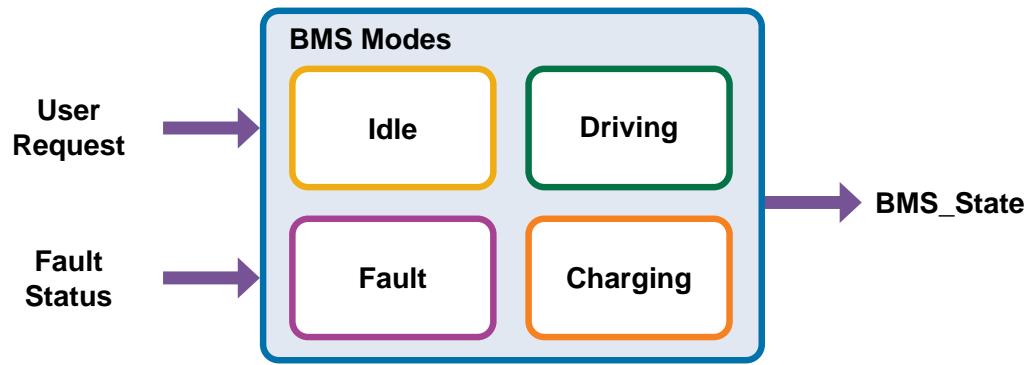
>> BMS02_charging



Battery Management System

Building the Supervisory Control Logic

Now you have discharged and charged the battery using separate control logic. It is time to combine all the functionalities together. The BMS decides which mode to operate on based on user request and current fault status. For the rest of this chapter, you will develop the supervisory control logic to decide the BMS state and trigger different BMS functionalities based on the selection.



System requirements

Depending on the **BMS_state**, a different current profile should be given to the battery. For example, when charging the battery, the charging current should be provided by the charging control logic in the BMS. When the user requests driving mode, the driving current profile should be given to the battery. If the BMS is in an idle state or at fault, all current should be cut off.

In Simulink, you can use a Multiport Switch to select the current profile based on the **BMS_State**. Instead of using numbers to represent **BMS_State**, you can use an enumeration data type to improve the readability of your model. **BMS_State_T** is a predefined enumeration data type that includes all the enumerated values for **BMS_State**.

Try

Open **BMS_State_T.m** and examine the enumeration data type.

>> edit BMS_State_T.m

```

classdef BMS_State_T < Simulink.IntEnumType
  enumeration
    Charging(1)
    Driving(2)
    Balancing(3)
    Idle(4)
    Fault(5)
  end
end
  
```

Battery Management System

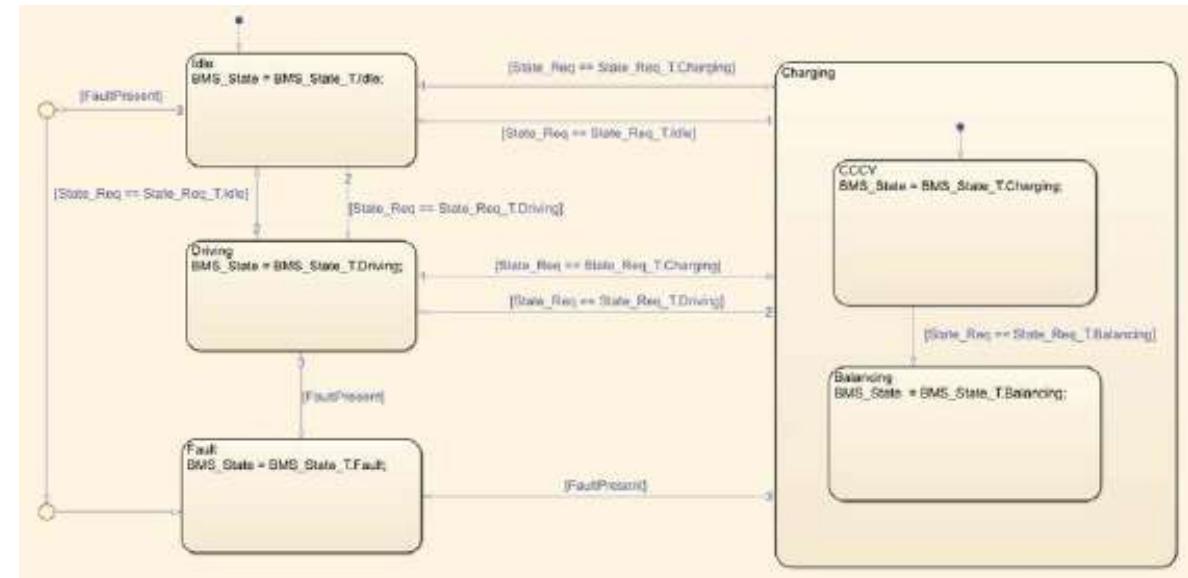
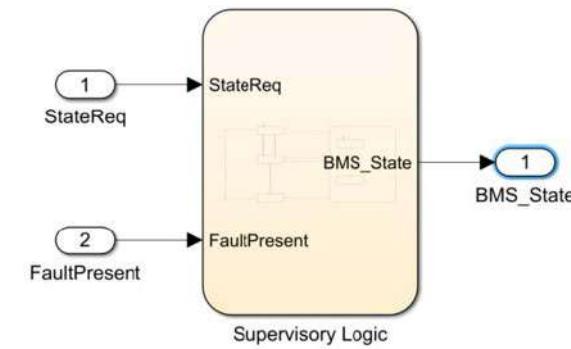
Supervisory Control State Machine

To decide the actual **BMS_State**, you will design the supervisory control state machine. The state machine takes user request **State_Req** and fault status **FaultPresent** as input, and produces **BMS_State** as output.

1. Add a Chart block into the **BMS** subsystem.
2. Add four states: **Idle**, **Driving**, **Charging**, **Fault**.
3. Inside the **Charging** state, add two substates **CCCV** and **Balancing**.
4. Set **BMS_State** values in the state actions.
5. Add the transitions based on **State_Req**, to transition between **Idle**, **Driving**, and **Charging**.
6. Add **FaultPresent** as the transition from each state to the **Fault** state.
7. Define **State_Req** and **FaultPresent** as inputs, and **BMS_State** as output in the **Symbols Pane**.
8. Connect the input/output signal lines outside the chart.

Try

Follow the steps to model the supervisory control state machine. Or, copy the prebuild chart from **supervisoryCtrl_chart.slx**



Battery Management System

Creating Test Sequence Input

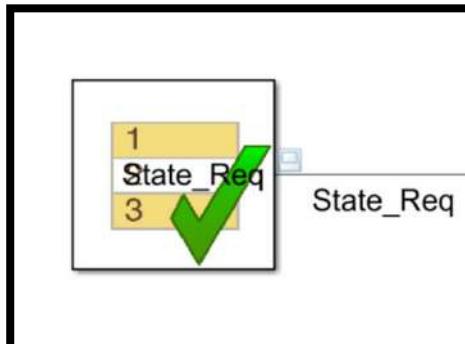
To test the supervisory control logic, you will design test cases that provide valid **State_Req** information. In Simulink, you can use the Test Sequence block from the **Simulink Test** library for the design. Instead of specifying the input at every time step, the Test Sequence block enables you to describe the test based on steps and transitions, similar to test descriptions in natural language.

You will now modify the Test Sequence block to implement the following logic to change **State_Req** between **Idle**, **Driving**, and **Charging**.

1. Standby for 60 seconds, give **State_Req** as **Idle**.
2. Transition to the **Driving** state, stay in the **Driving** state for 17500 seconds.
3. In the end, charge the battery by going into the **Charging** state.

To create a **Step**, specify the step name in the first line of the cell. Any additional lines are MATLAB® expressions known as step actions.

To add a step after the existing step, you can right-click, or place your cursor over the existing step cell to **Add step after**.



Step	Transition	Next Step	Description
: Standby State_Req = State_Req_T.Idle;	1. after(60,sec)	Driving	
Add step after • Add sub-step •			
Driving State_Req = State_Req_T.Driving;	1. after(17500,sec)	Charging	
Charging State_Req = State_Req_T.Charging			

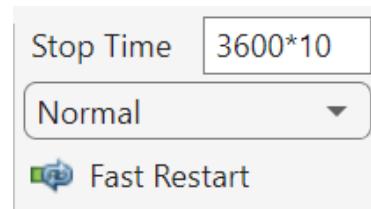
Battery Management System

Creating Test Sequence Input (Continued)

Similar to the requirement of Stateflow, you must create the symbols for input, output, parameters, etc., before using them in the Test Sequence block. You can add them from the **Symbols Pane**.



Change simulation **Stop Time** to $3600*10$. Simulate the model to test the supervisory control logic. The system goes under driving mode first, then switches to constant current charging. In the next section, the supervisory control logic is combined with the CC-CV charging logic.



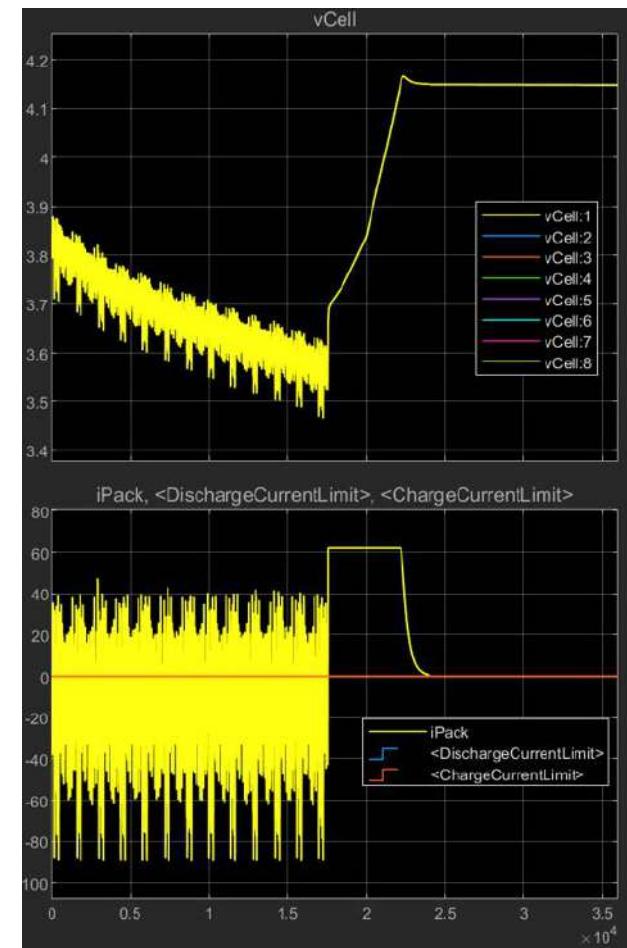
Try

Add **State_Req** as an output in the **Symbol Pane**.

Simulate the model.

Or,

```
>> BMS03_supervisory
```

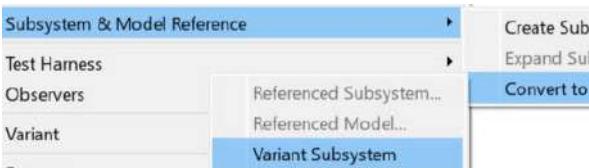


Battery Management System

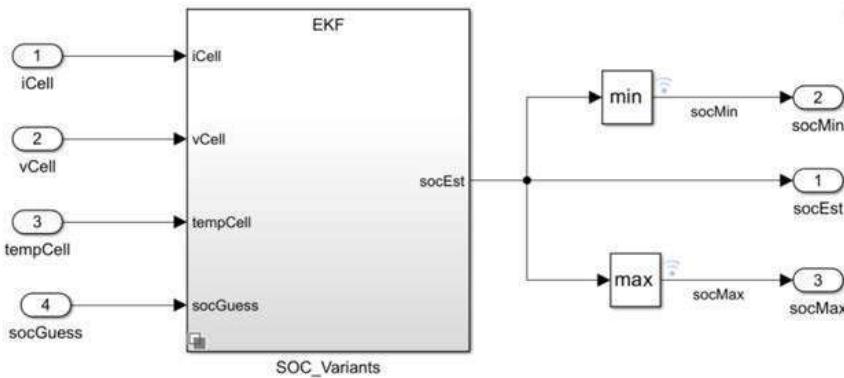
Integrating SOC Estimation

In chapter five, you have implemented coulomb counting and extended Kalman filter to estimate the SOC of a cell. When you have multiple options to complete a task, you can employ component variant to facilitate the implementation.

1. Go into the **SOC Estimation** subsystem, uncomment the **CoulombCounting** and the **EKF** subsystems.
2. Right-click on the **EKF** subsystem and go to **Subsystem & Model Reference** > **Convert to** > **Variant Subsystem**.
3. Rename the subsystem as **SOC_Variants**.



4. Cut and copy the **CoulombCounting** subsystem as another option into the variant subsystem.
5. Connect **SOC_Variants** with inputs and outputs as shown in the figure.



Try

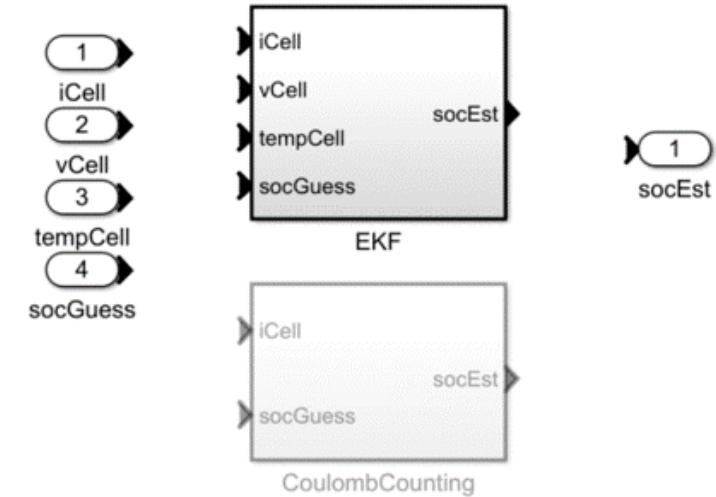
Follow the steps to create component variants with the two SOC estimation methods.

>> BMS04_socEst

In variant subsystems, signals are not explicitly connected between ports. Instead, they are mapped by the name of the ports. To select an active variant, follow the steps to control the variant selection with a predefined variable **socMethod**.

6. Right-click on the **SOC_Variants** and go to its **Block Parameters**. Change the **Variant control mode** to **expression**.
7. Provide the **Variant control expression** as shown in the figure.

Variant choices (table of variant systems)	
Name (read-only)	Variant control expression
CoulombCounting	<code>socMethod == 1</code>
EKF	<code>socMethod == 2</code>



Battery Management System

Cell Balancing

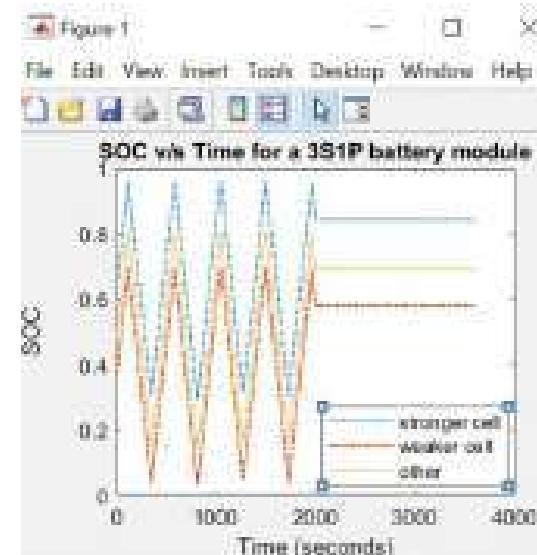
A battery pack is said to be unbalanced if the battery cells in the pack have a high difference in their SOC values.

During the battery pack operation over time, the cells have an inherent tendency to diverge to different SOC values and go out of balance. This imbalance may be caused by the difference in the coulombic efficiency of the cells, different leakage currents or self-discharge rates, the difference in the temperature gradient at different parts of the pack, or a combination of these factors.

The imbalance in the SOC levels makes the battery pack unusable or yields poor performance over time. In a series combination of cells, the cell with the lowest SOC value limits the discharging of the battery pack as it reaches the discharging limit first. Even if the other cells still have a significant charge remaining in them, they cannot be utilized. Similarly in charging, to avoid overcharging the cell with the highest SOC, other cells cannot reach a fully charged state.



The imbalance in the SOC necessitates the regulation in the charge level of individual cells. The BMS uses balancing schemes to ensure the cells are maintained at similar SOC values. You can use either passive (dissipative) or active balancing schemes for this task. In this course, a passive balancing algorithm is used to balance in SOC levels.



Battery Management System

Passive Cell Balancing

Passive balancing is a dissipative balancing technique where the excess charge of the cells with high SOC values is dissipated out to the environment in the form of heat. It makes use of basic electronic circuitry of switches and resistors as shown in the figure.

Since heat is being generated during balancing, the value of the balancing resistor should be carefully selected based on the balancing time and its power rating. A smaller balancing current is favorable to heat dissipation if a longer balancing time is allowed.

For example, assume you have 6 hours to balance the battery pack of your EV during the evening. The capacity of each cell is 30Ah and the maximum SOC difference is $dSOC = 5\%$. The desired balancing current can be calculated as:

$$I_{bal} = \frac{dSOC \times Q_{nom}}{dT} = \frac{5\% \times 30}{6} = 0.25(A)$$

If the balancing is conducted around the voltage of 4V, the required balancing resistor is:

$$R_{bleed} = \frac{V_t}{I_{bal}} = \frac{4}{0.25} = 16 (\Omega)$$

Try

>> balancingInitCondt

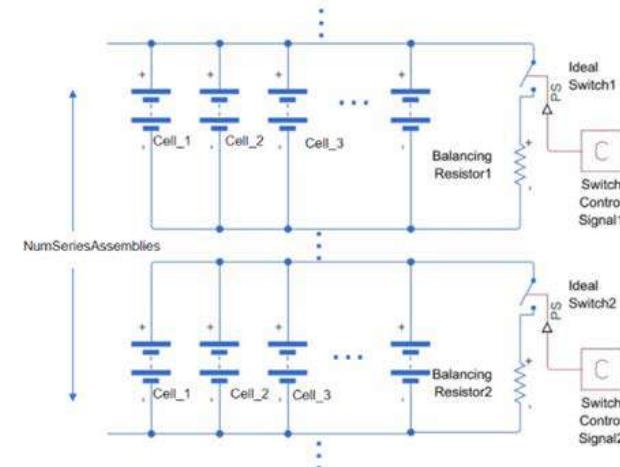
Run the script to initialize an imbalanced battery pack.

When using Simscape Battery to create custom battery pack, you can embed passive balancing resistors by setting **BalancingStrategy** to **Passive**.

BalancingStrategy **Passive**

The balancing resistances are added in parallel per parallel assembly. You can configure the balancing resistance through the **Block Parameters** of the module.

Cell Balancing	
> Cell balancing switch closed resistance	myModAssy.myModGr... Ohm
> Cell balancing switch open conductance	myModAssy.myModGr... 1/Ohm
> Cell balancing switch operation threshold	myModAssy.myModGrp.CellBalancing
> Cell balancing shunt resistance	myModAssy.myModGr... Ohm



Battery Management System

Building the Balancing Logic

There are multiple battery variables that can be used as balancing criteria, such as SOC and terminal voltage. If the cell SOC can be measured accurately, using SOC directly for balancing is a straightforward method. However, algorithms such as Extended Kalman Filter require large computation efforts which makes the implementation expensive. On the other hand, terminal voltage can be obtained easily through sensor measurement but it does not provide a good representation of the SOC when the battery is not settled. In real applications, voltage-based passive balancing is usually conducted by the end of CC-CV charging when the terminal voltage is close enough to the open-circuit voltage.

In this section, you will develop a voltage-based balancing logic using prebuild Simscape Battery block. Using the cell with the lowest SOC as the reference, the BMS activates the resistive network of the cells whose SOC is higher than the reference and allows the excess charge to dissipate out of the cells through the resistors in the form of heat energy.

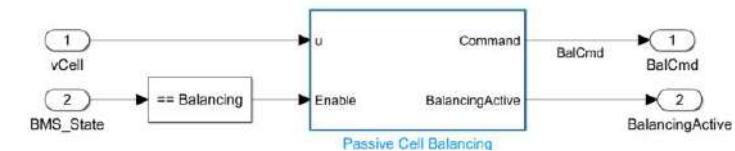
1. Go into the **Balancing** subsystem and add a Passive Cell Balancing block into the model.
2. Connect the input and output as shown in the figure.
3. Set the **Threshold** for balancing to **balThld**.

To unit test the balancing logic, you will temporary set the **BMS_State** to balancing.

4. In the **Supervisory Control** subsystem, add a Constant block and provide the value of **BMS_State_T.Balancing**.
5. Connect the Constant to port **BMS_State**.
6. Simulate the model.

Try

Add a Passive Cell Balancing block into the **Balancing** subsystem.

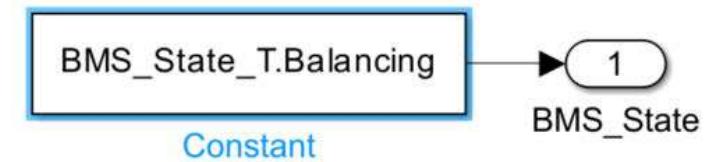


Parameters

Main Signal Attributes

Threshold for balancing

balThld

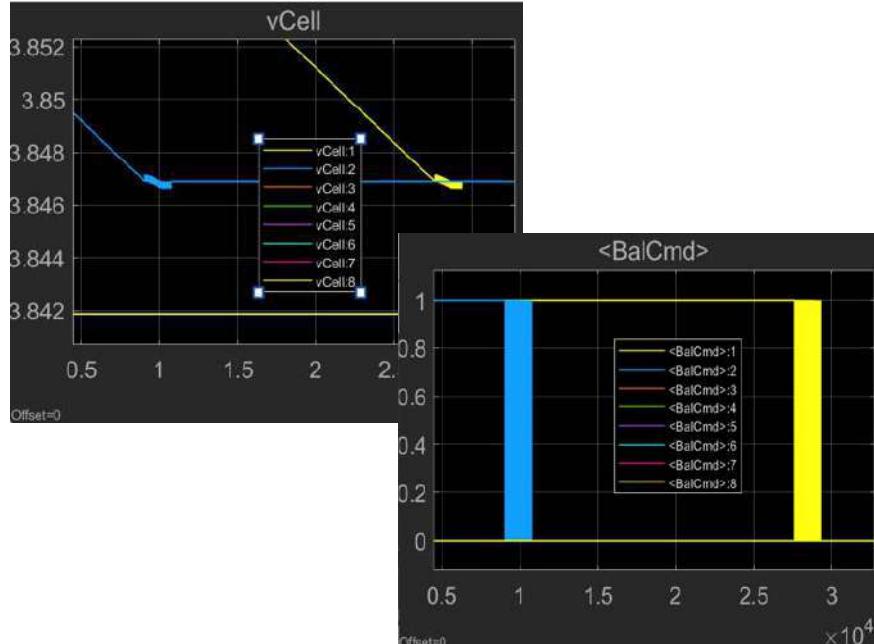


Battery Management System

Balancing Logic Solution

Starting with different SOC levels of 72%, 71%, and 70%, the cells converge to a maximum voltage difference of 0.005 V (**BalVoltageThld**). The SOC differences between the cells are minimized to around 0.5%.

However, there is a significant on-off switching event in the **BalCmd** as the SOC levels approach the threshold value. Since the balancing is done based on the terminal voltage value, cutting off the balancing current increases the terminal voltage above the threshold, which causes the switches to close again. The switching continues until the recovery voltage is smaller than the voltage threshold.



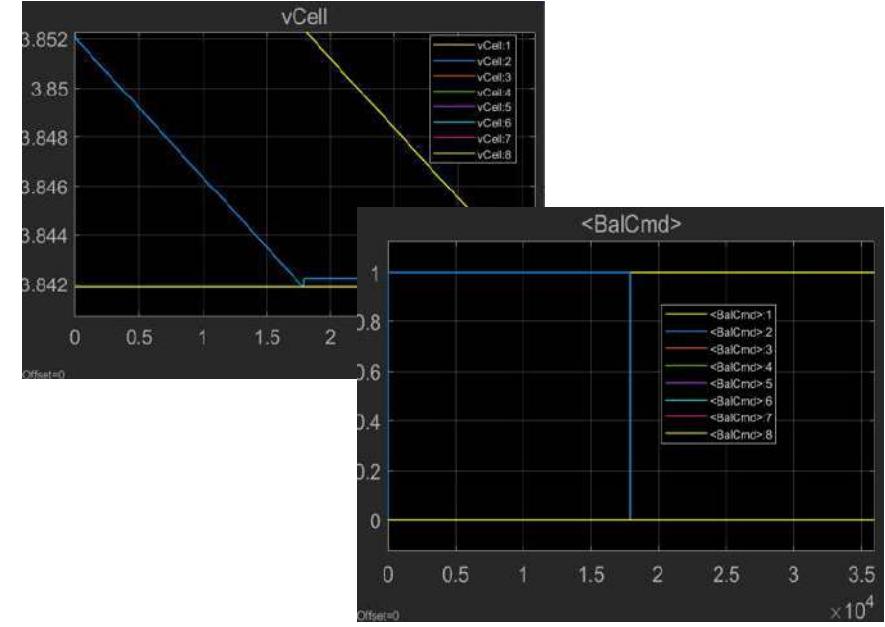
Try

Include hysteresis setting to debounce the balancing performance.

The balancing logic can be improved by adding hysteresis into the voltage threshold.

1. Set the **Hysteresis band for switching balancing off** to **balHyst**.
2. Simulate the model again.

Hysteresis band for switching balancing off
balHyst



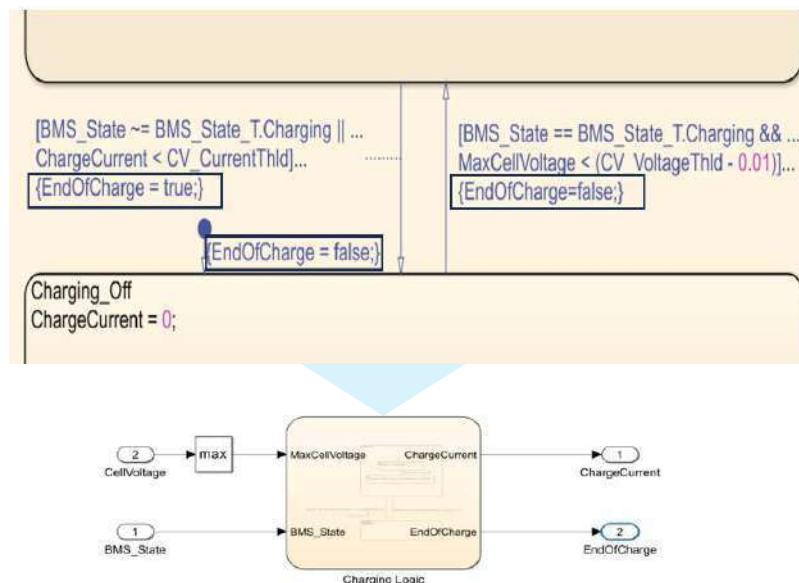
Battery Management System

Integration with Battery Management System

The balancing algorithm can now be integrated as one of the supervisory control tasks done by the BMS.

The BMS should initiate the balancing task at the end of the charging period of the battery module. To notify the supervisory control logic the battery reaches the end of charge:

1. Add an output variable **EndOfCharge** to the **Charging Logic** chart.
2. Set the default value of **EndOfCharge** as false on the **Default Transition** when entering the chart.
3. Add a condition action to change **EndOfCharge** to true while transitioning out of the **Charging_On** state, thereby indicating the end of the charging process.
4. To reset the **EndOfCharge** flag, change **EndOfCharge** to false on the transition from **Charging_Off** to **Charging_On**.

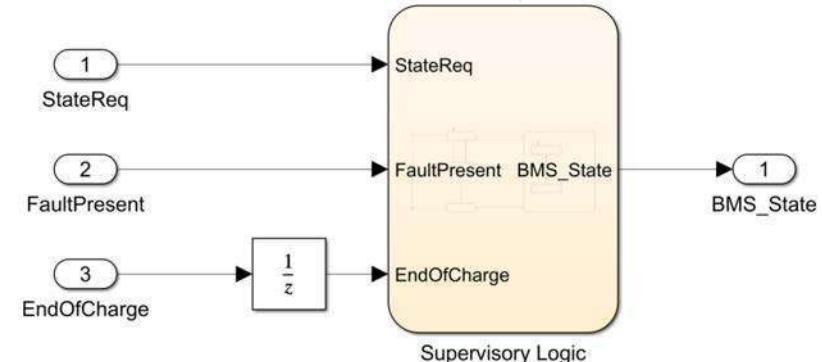
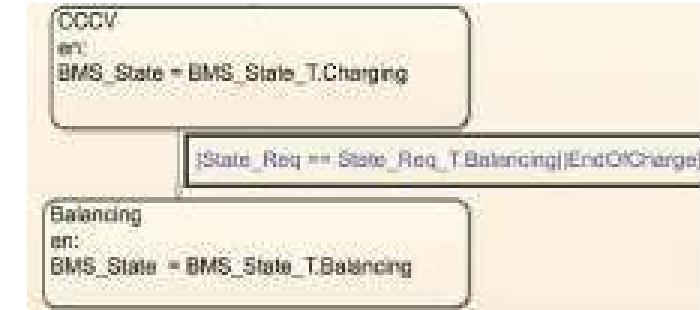


Try

>> BMS05_balancing

Follow the steps to trigger balancing by the end of CC-CV charging

5. Add **EndOfCharge** as an input to the **Supervisory_Control** chart. Use **EndOfCharge** as an Or condition to transit from the **CCCV** substate to the **Balancing** substate.
6. Restore the connection to the **BMS_State**.
7. Simulate the model.



Summary

- Overview of the battery management system
- Design Stateflow® logic to charge a cell using CC-CV control scheme
- Design supervisory control logic of battery management system using Stateflow®
- Implement a passive cell balancing network °
- Create test scenarios for battery management system using Simulink Test™

Test Your Knowledge

1. (T/F) To use the enumeration data type defined in file A.m, you need to run the .m file before using it in Simulink®.
2. (T/F) Test Sequence blocks must be configured to have at least one input and one output.
3. (T/F) In a Test Sequence block, the value you specified in the transition column must be a logical expression.

Answers

1. False
2. False
3. True



A photograph of a man and a woman looking at a computer screen together. The man is pointing at the screen, and they both appear to be engaged in a discussion or review of some data or software interface.

Power Electronics Control Design with Simulink® and
Simscape™

Modeling DC/DC Power Electronic Converters

Outline

- Boost converter operation principles
- Modeling an open-loop boost converter
- Measuring physical quantities
- Selecting a solver
- Inspecting simulation results
- Using prebuilt PWM blocks

Chapter Learning Outcomes

The attendee will be able to

- Build and simulate DC/DC power electronic converters using discrete component modeling approach.
- Sense and visualize physical quantities.
- Select solver settings suitable for power electronics models.
- Understand and use appropriate PWM techniques.

Course Example: Boost Converter

In this chapter, you will learn how to model a boost converter that increases the voltage supplied by the hybrid electric vehicle's (HEV) batteries from 250 to 400 Volts to drive the motor.

The boost converter, also known as a step-up converter, is a power electronic device that increases the voltage and decreases the current of a DC source. A typical boost converter is shown in the circuit on the right, consisting of an inductor, a semiconductor switch, a diode, and a capacitor.

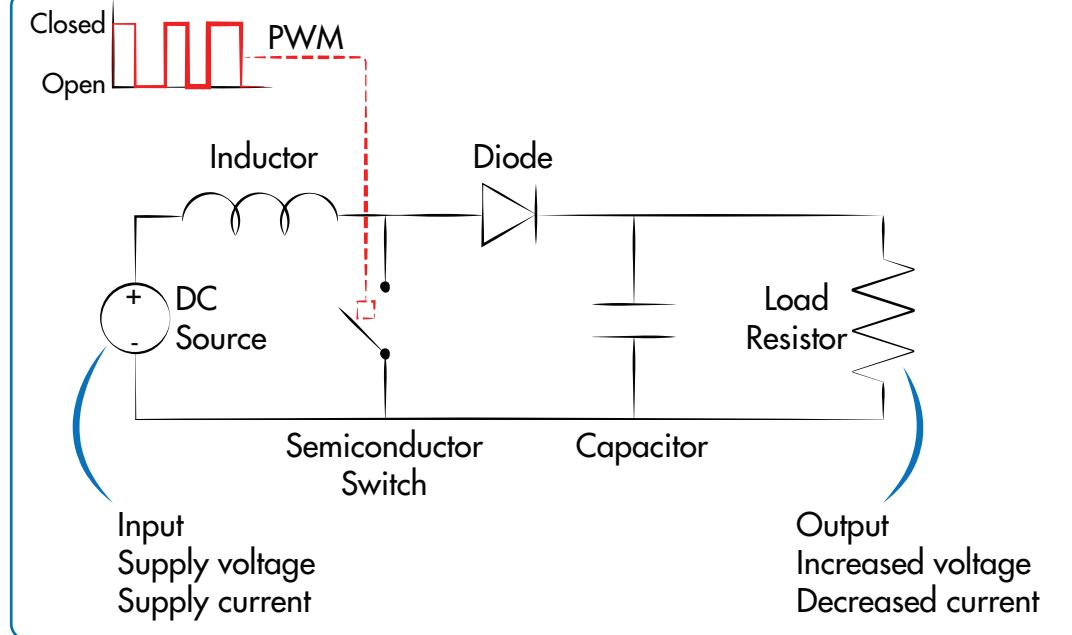
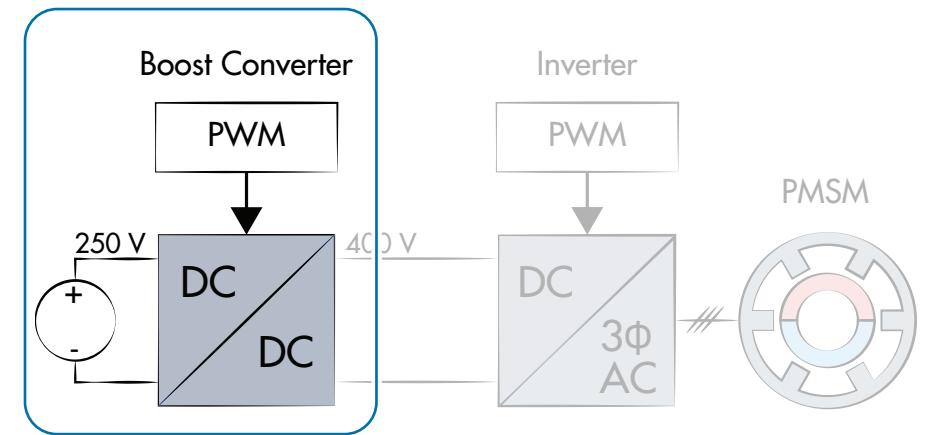
To understand the boost converter operation, consider both states of the semiconductor switch while the circuit is in steady state:

- **Switch closed** – On the input side, the DC source charges the inductor up to the source voltage. On the output side, current flows from the capacitor to the load (i.e., resistor). The diode prevents current from flowing from the capacitor to ground through the closed switch.
- **Switch open** – When the switch opens, current flows from the inductor to charge the capacitor and to the load.

For every power electronic converter topology, in steady-state, the net flux linkage of the inductor is zero over each switching period. This is the principle of *inductor volt-second balance* and brings the designer to find steady-state conditions for any topology. For the boost converter, applying this principle leads to the output voltage greater than the source voltage with the relationship shown in the next page.

The switching behavior in the circuit is specified using a pulse width modulation (PWM) signal that controls when the switch is open and closed. The percentage of time that the switch is closed over a single switching period is called the *duty cycle*. Changing the duty cycle controls the output voltage of the boost converter.

In this chapter, you will use the Simscape Electrical library to model an open-loop boost converter using discrete electrical components. You will use Simulink blocks to model a PWM generator.



Course Example: Boost Converter (Continued)

The boost converter in this chapter uses the design specifications provided in the table on the right. These specifications are selected to produce the following *steady-state operating conditions*:

1. The converter operates in *continuous current mode* (CCM), which means that the inductor current is always greater than zero.
2. The input current I_{in} is described with the following equation:

$$I_{in} = \frac{V_{in}}{(1 - D)^2 R}$$

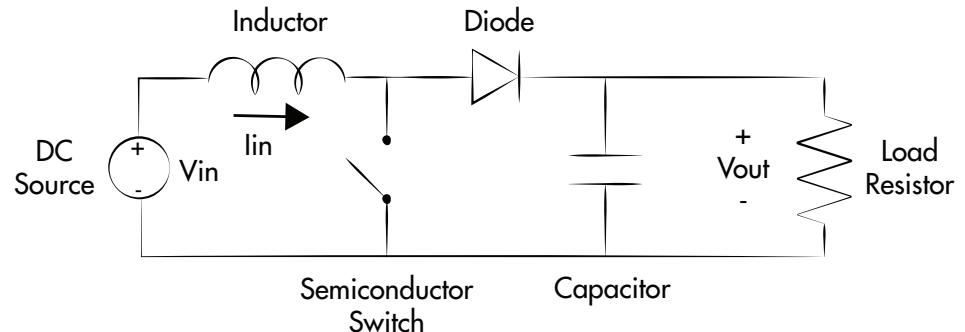
Here, V_{in} is the input voltage, D is the duty cycle, and R is the load resistance.

3. The output voltage V_{out} is described with the following equation:

$$V_{out} = \frac{V_{in}}{1 - D}$$

Note The above relationships refer to an ideal boost converter (no losses). The boost converter theoretically produces a voltage of 400 V and an input current of about 240 A when the PWM duty cycle is 37.5% in steady state. However, the boost converter modeled in this course is lossy. To compensate for the losses, the duty cycle value is rounded up to 40% (i.e., 0.4).

After you model the nonideal boost converter with the given design specifications, you add voltage and current sensors to test and verify whether the three operating conditions above are satisfied. Moreover, you can identify the lossy elements responsible for any discrepancy with respect to the results of an ideal boost converter by using the relationships above.



Element	Parameter	Value	Units
Voltage source	DC Voltage	250	V
Load resistor	Resistance	2.67	Ω
Inductor	Inductance	195	μH
	Series resistance	34.7	$\text{m}\Omega$
	Parallel conductance	0	$1/\Omega$
Capacitor	Capacitance	7	mF
	Series resistance	6.67	$\text{m}\Omega$
Diode	Forward voltage	0.6	V
	On resistance	2	$\text{m}\Omega$
	Off conductance	1e-8	$1/\Omega$
Semiconductor switch	On-state resistance	10	$\text{m}\Omega$
	Off-state conductance	1e-8	$1/\Omega$
PWM	Switching frequency	10	kHz
	Duty cycle	0.4	

Modeling DC/DC Power Electronic Converters

Creating a New Simscape Model

You can use the Simulink Start Page to create new models, open templates, find examples, and more. In this chapter, you create a power electronics model from a blank model. In later chapters, you will learn how to use electrical templates.

To create a new blank model,

1. Click **Simulink** in the MATLAB toolbar to open the Simulink Start Page.
2. Click **Blank Model** in the Simulink section of the New tab.

To model the boost converter, you can open the Library Browser and add components to the model from the **Simscape > Electrical** library.

Electrical networks modeled using Simscape components must contain the following two blocks:

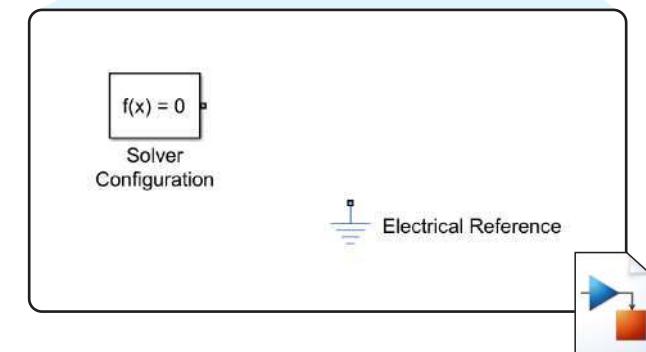
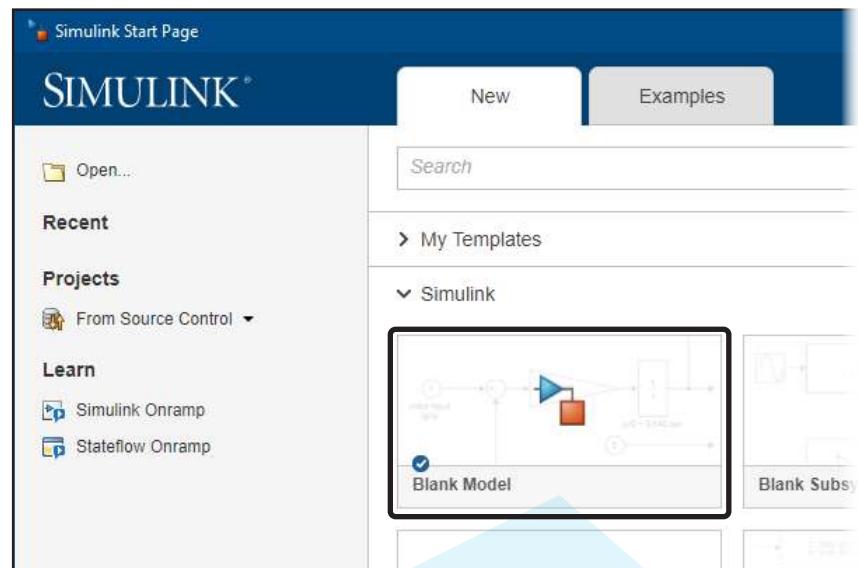
- Solver Configuration (**Simscape > Utilities** library) – This block defines the solver settings for a physical network. Each Simscape physical network must contain *only* one Solver Configuration block.
- Electrical Reference (**Simscape > Electrical > Connectors & References** library) – This block establishes a voltage reference level (i.e., ground). Each Simscape electrical network must contain *at least* one reference.

Note If you are unsure where to begin, try exploring the example models provided in the Simscape Electrical documentation. Many times, you may find a relevant model to use as a starting point. For more information, see **Simscape Electrical** in the documentation and click **Examples**.

Try

Use the Simulink Start Page to create a new blank model.

Then, add an Electrical Reference block and a Solver Configuration block to the model.



Exploring the Simscape Electrical Library

The Simscape Electrical library (**Simscape > Electrical**) contains Simscape components that use Simscape Foundation domains, including the Electrical and Three-Phase Electrical domains, for detailed modeling of electrical systems.

Simscape Electrical blocks are organized into the sublibraries listed below, with examples of some components inside:

- **Connectors & References** – electrical references, phase splitters
- **Control** – controllers, measurements, mathematical transforms
- **Electromechanical** – actuators, generators, motors, piezos, servos
- **Integrated Circuits** – operational amplifiers, timers, logic gates
- **Passive** – capacitors, inductors, resistors, transformers
- **Semiconductors & Converters** – diodes, transistors, power converters
- **Sensors & Transducers** – encoders, voltage sensors, current sensors
- **Sources** – batteries, voltage sources, current sources
- **Switches & Breakers** – breakers, fuses, relays, switches
- **Utilities** – environmental parameters and faults
- **Additional Components** – SPICE components
- **Specialized Power Systems** – power systems components using the specialized electrical domain

Some blocks in the Simscape Electrical library may appear similar to blocks in the Simscape Foundation Electrical library (**Simscape > Foundation Library > Electrical**). The Simscape Electrical blocks generally provide additional parameters not available in the foundation library. For example,

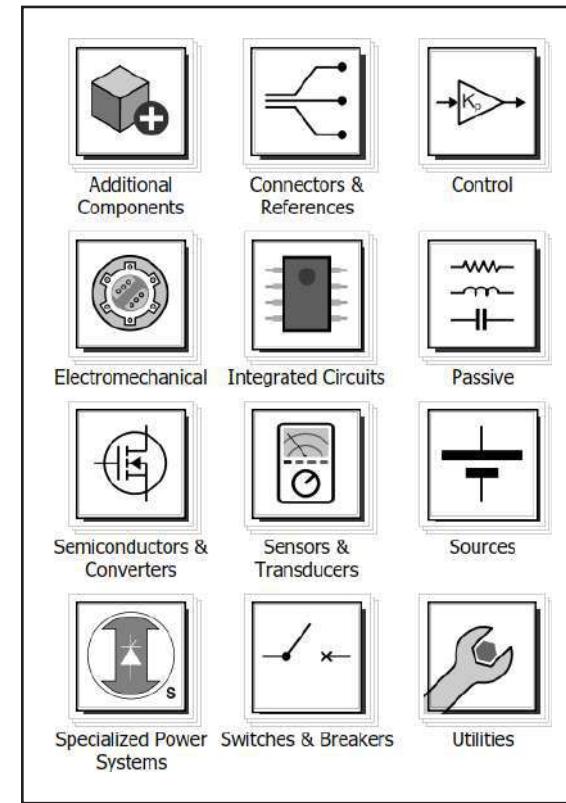
- The Resistor block in the **Foundation Library > Electrical > Electrical Elements** library provides only one parameter for resistance.
- The Resistor block in the **Electrical > Passive > Resistor** library provides multiple parameters for resistance, tolerance, operating limits, and faults.

Try

Open the Simscape Electrical library and explore the various components inside.

Compare the block parameters for similar blocks in the Foundation library and the Simscape Electrical library.

Note If you double-click the model canvas to add a block, pay close attention to which library the block is from. This will help avoid confusion between the Foundation library and Simscape Electrical blocks.



Modeling DC/DC Power Electronic Converters

Modeling a Boost Converter

To model the boost converter in the HEV course example, do the following:

1. Add the blocks listed below to your model.
 - Voltage Source (**Electrical > Sources** library)
 - Capacitor (**Electrical > Passive** library)
 - Inductor (**Electrical > Passive** library)
 - Resistor (**Electrical > Passive** library)
 - Diode (**Electrical > Semiconductors & Converters** library)
 - Ideal Semiconductor Switch (**Electrical > Semiconductors & Converters** library)
2. Arrange and connect the blocks according to the circuit diagram on the right. The lines connecting the circuit elements represent physical connections that exchange energy flows, characterized by voltage and current.
3. Change the name of the Resistor to Load.
4. Connect the Electrical Reference block to the negative port of the Voltage Source block to serve as the circuit ground.
5. Connect the Solver Configuration block to any point in the Simscape network to specify the solver parameters.

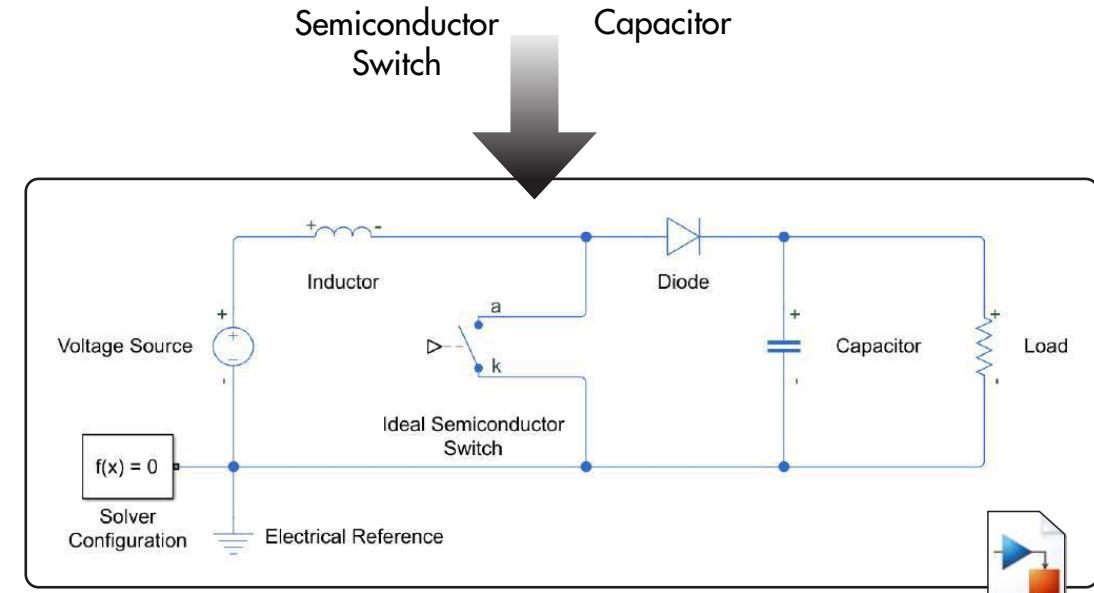
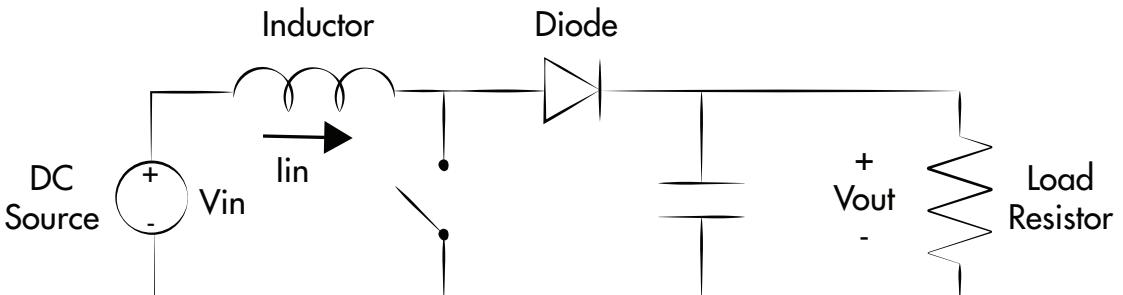
The Ideal Semiconductor Switch has an unconnected physical signal input that controls the switch. You will connect this to the PWM signal later.

Recall that physical signals in Simscape are directional and serve to carry signals between Simscape blocks. For more information, see **Simscape > Get Started with Simscape > Basic Principles of Physical Networks** in the documentation.

Note To branch an existing physical connection line, you can right-click the line and drag the mouse.

Try

Follow the steps listed on this page to build the boost converter model using blocks from the Simscape Electrical library.

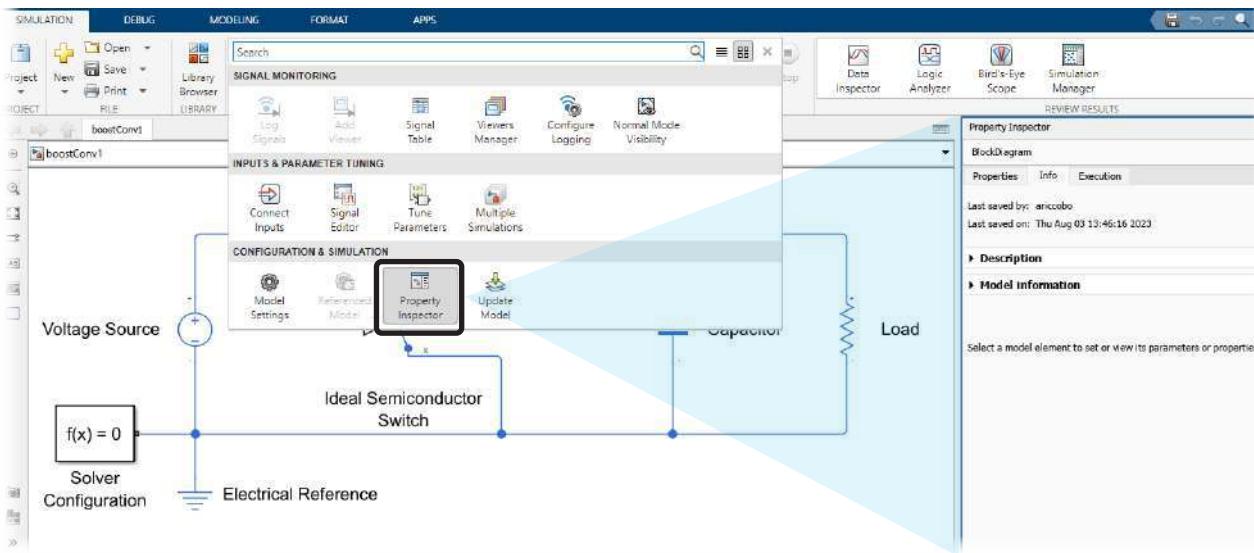


Modeling DC/DC Power Electronic Converters

Modeling a Boost Converter (Continued)

You must configure the parameters for each block in the boost converter circuit to match the design specifications in the table below.

Element	Parameter	Value	Units
Voltage source	DC voltage	250	V
Load resistor	Resistance	2.67	Ω
Inductor	Inductance	195	μH
	Series resistance	34.7	$\text{m}\Omega$
	Parallel conductance	0	$1/\Omega$
Capacitor	Capacitance	7	mF
	Series resistance	6.67	$\text{m}\Omega$
Diode	Forward voltage	0.6	V
	On resistance	2	$\text{m}\Omega$
	Off conductance	1e-8	$1/\Omega$
Semiconductor switch	On-state resistance	10	$\text{m}\Omega$
	Off-state conductance	1e-8	$1/\Omega$



Try

Use the Property Inspector to set the parameters for each block, based on the design specifications provided in the table on this page.

>> boostConv1

To set parameter values for a block,

1. Open the Property Inspector by clicking **Property Inspector** in the **Prepare** section of the **Simulation** tab.
2. Select a block in the model.
3. Use the **Property Inspector** pane in the model window to set relevant block parameters.

Note#1 The **Property Inspector** is also available in the **Design** section of the Modeling tab.

Note#2 You can add block annotations to improve readability by displaying block parameter values in the model canvas. To learn more about this, see **Simulink > Modeling > Configure Signals, States, and Parameters > Blocks > Specify Block Properties > Set Block Annotation Properties** in the documentation.

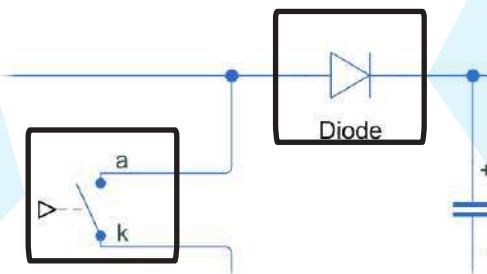
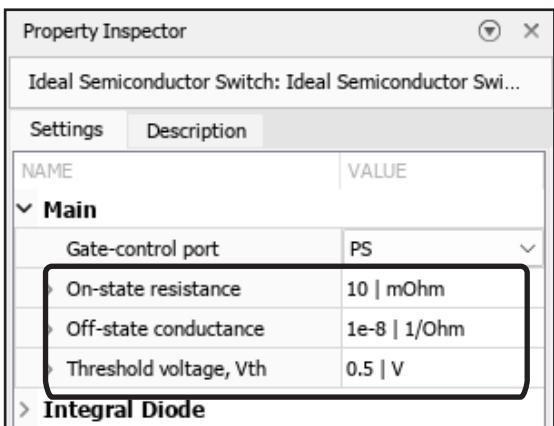
Modeling Ideal Semiconductor Switches

In this chapter, you are modeling the boost converter's main semiconductor switch with an Ideal Semiconductor Switch block and the diode with a Diode block. These blocks can be used to mimic ideal semiconductor switches that possess the following operating properties:

- No limit on the amount of current the device can carry in its on-state.
- No limit on the amount of voltage the device can block in its off-state.
- Zero or very little voltage drop when the device is in its on-state.
- Zero or very little off-state conductance, that is zero or very little current when the device is in the off-state.
- Zero switching time when the device changes from its on-state to its off-state and vice versa.
- It dissipates zero or nearly zero power.

Ideal Semiconductor Switch

If the gate-cathode voltage, that is the voltage at the PS input with respect to the k terminal, exceeds the specified threshold voltage, the Ideal Semiconductor Switch is in the on state and behaves like a linear resistor with the resistance specified in the **On resistance** parameter. Otherwise, the device is in the off state and behaves like a linear resistor with the small conductance specified in the **Off conductance** parameter.



Try

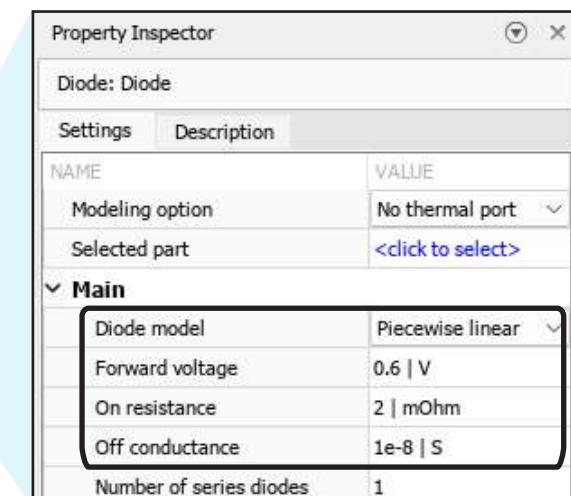
Analyze the main parameters of the Ideal Semiconductor Switch and Diode blocks by the usage of the Property Inspector.

>> boostConv1

Diode

The Diode block can represent either a piecewise linear diode, an exponential diode, or a diode with a tabulated I-V curve. When the **piecewise linear** diode model is selected, if the diode forward voltage exceeds the value specified in the **Forward voltage** parameter, the diode behaves as a linear resistor with the resistance specified in the **On resistance** parameter. Otherwise, the diode behaves as a linear resistor with the small conductance specified in the **Off conductance** parameter.

Note Some Simscape Electrical components provide tools to view block characteristics. For example, right-click the Diode block and select **Electrical > Basic Characteristics**.



Modeling a Pulse Width Modulation (PWM) Generator

The switching behavior of the semiconductor switch is controlled with a PWM signal that is characterized by

- A periodic carrier waveform with switching frequency of 10 kHz.
- A modulation signal valued from 0 to 1 representing the fraction of time that the signal is 1 over the switching period.

The PWM signal is generated by comparing the modulation signal (duty cycle) to the carrier waveform (sawtooth), as shown in the figure. This generates a value of 1 (switch on) when the duty cycle is greater than the sawtooth and 0 (switch off) otherwise.

Follow the steps below to model the PWM signal using Simulink blocks:

1. Add a Constant block (**Simulink > Sources** library). Name the block **Duty Cycle**, and set the **Value** to **0.4**.
2. Add a Repeating Sequence Stair block (**Simulink > Sources** library) to generate the sawtooth waveform. Name the block **Sawtooth**, set the **Vector of output values** to **linspace(0,1,20)**, and set the **Sample Time** to **5e-6**. This will generate a periodic sawtooth waveform with a frequency of 10 kHz that is sampled 20 times faster (200 kHz):

$$F_{\text{sawtooth}} = \frac{1}{(5 \mu\text{s}/\text{point})(20 \text{ points})} = 10 \text{ kHz}$$

3. Add a Relational Operator block (**Simulink > Logic and Bit Operations** library) performing the operation: **Duty Cycle >= Sawtooth**.

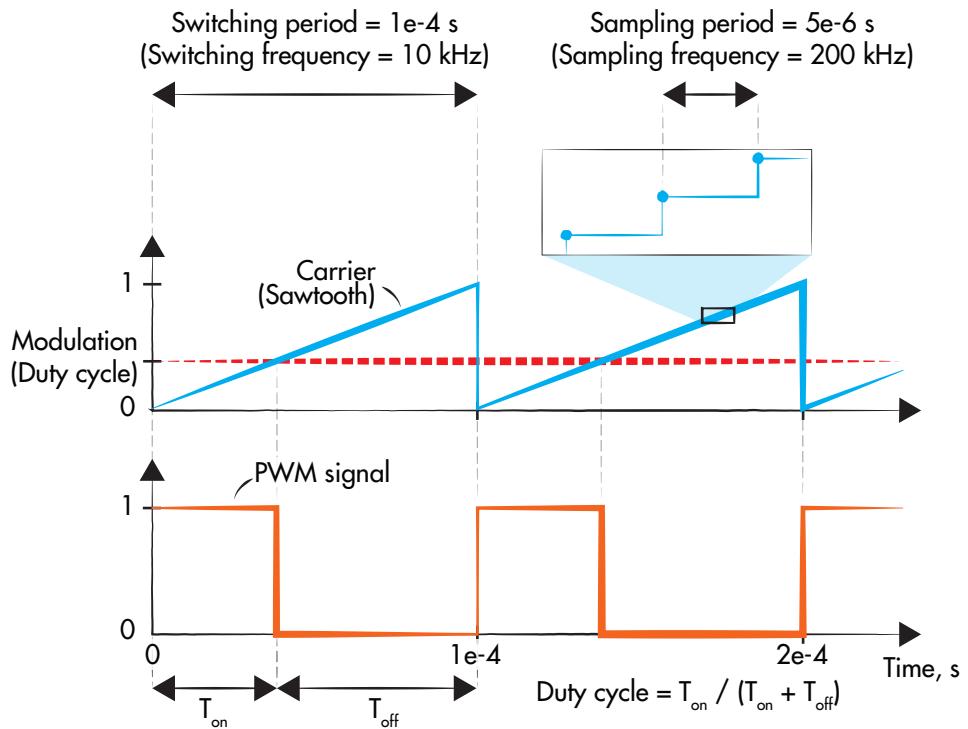
Note The sampling frequency of the PWM sets the resolution of the duty cycle. For simulation performance, set the sampling frequency to 20 to 50 times greater than the switching frequency. For high accuracy, set the sampling frequency to 100 to 200 times greater than the switching frequency.

Try

Follow the steps listed on this page to model the PWM signal generator.

This PWM is called *sawtooth trailing edge*. To implement a *sawtooth leading edge* PWM set the **Vector of output values** to **linspace(1,0,20)** in the Repeating Sequence Stair block. For a *triangular double edge* PWM, set the **Vector of output values** to **[linspace(0,1,10) linspace(1,0,10)]** – notice the 10 points during the ramp-up and the 10 points during the ramp-down.

Element	Parameter	Value	Units
PWM	Switching frequency	10	kHz
	Duty cycle	0.4	



Modeling a Pulse Width Modulation (PWM) Generator (Continued)

To connect the PWM signal to the Ideal Semiconductor Switch block in the Simscape network, do the following:

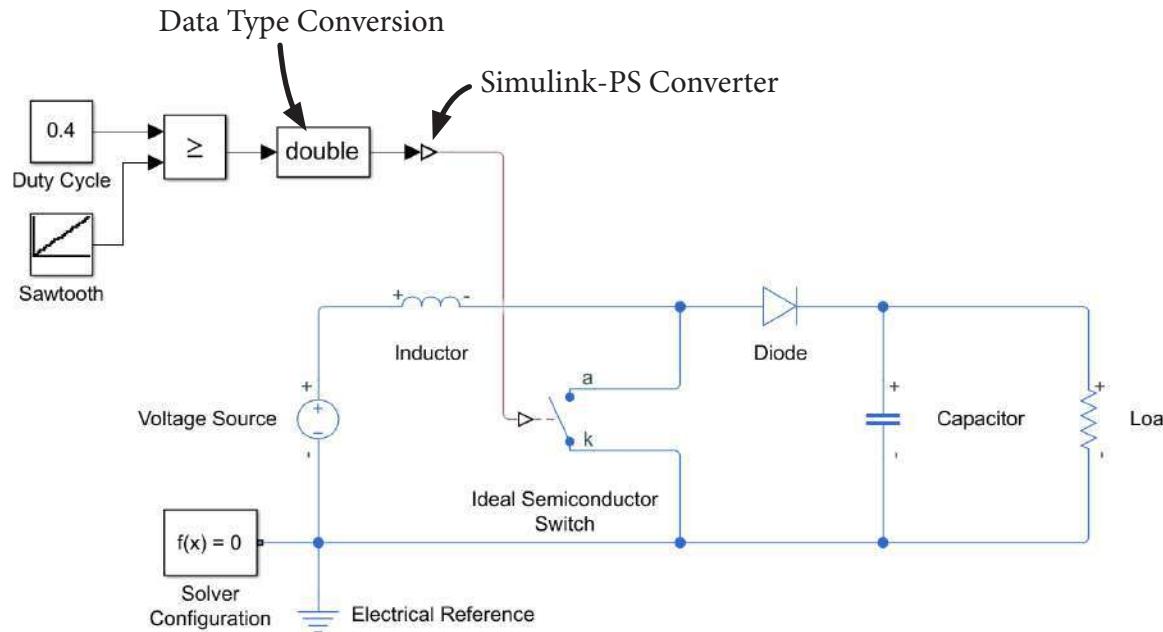
1. Add a Data Type Conversion block (**Simulink > Signal Attributes** library) and connect it to the output of the Relational Operator block. Set the **Output data type** to **double**. This converts the boolean output of the Relational Operator to a double signal so that it can be connected to the Simscape network with the consistent data type.
2. Add a Simulink-PS Converter block (**Simscape > Utilities**) to convert the double Simulink signal into a physical signal. Change the **Input signal unit** to **V**.
3. Connect the output of the Simulink-PS Converter block to the physical signal input of the Ideal Semiconductor Switch block.

Try

Use the steps on this page to connect the PWM generator to the Simscape network.

>> boostConv2

Note In other models, you may need to adjust the **Threshold voltage**, **Vth** parameter of the Ideal Semiconductor Switch block to agree with the signal range of the input physical signal. In this chapter's model, no changes are necessary because the input PWM signal ranges from 0 to 1 volt and the threshold voltage is set to 0.5 volts by default.



Measuring Physical Quantities

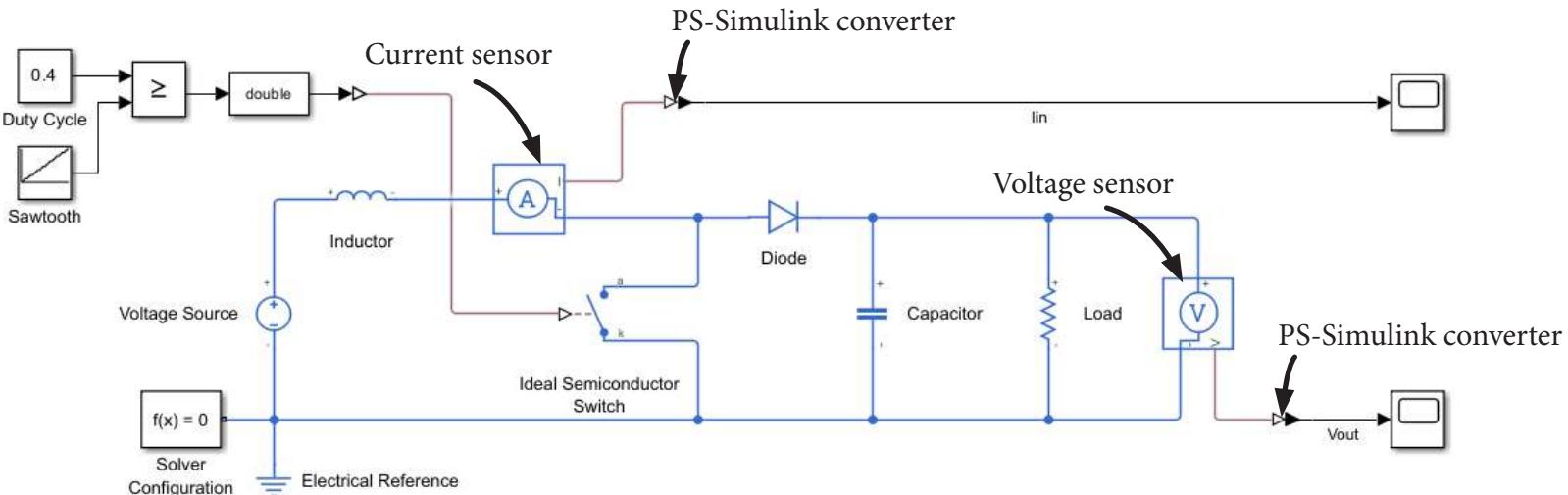
You can add sensors to measure the current and voltage at different branches and points in the model, respectively. This is useful to verify that the converter behaves as expected.

In the electrical domain, recall that

- Current is a *through* variable and should be measured through a branch in the model.
- Voltage is an *across* variable and should be measured across two points in the model.

Use the blocks below to make measurements in the model:

- Current Sensor (**Electrical > Sensors** library) – Measures the current through a branch using an ideal sensor (i.e., no series resistance).
- Voltage Sensor (**Electrical > Sensors** library) – Measures the voltage across two points using an ideal sensor (i.e., infinite parallel resistance).
- PS-Simulink Converter (**Simscape > Utilities** library) – Converts the physical signal output of a sensor to a Simulink signal.



Try

Add current and voltage sensors to measure the input current and the output voltage, respectively.

The output voltage measurement should be placed across the resistive load. The input current measurement should be placed in series with the Voltage Source.

Set the **Output signal unit** in the PS-Simulink Converter blocks to A for current and V for voltage, respectively. Then, label each resulting Simulink signal appropriately (e.g., **Iin**, **Vout**).

>> boostConv3

Note For more information about through and across variables, see **Simscape > Getting Started with Simscape > Basic Principles of Modeling Physical Networks > Variable Types** in the documentation.

Choosing a Solver

Switch-mode power electronic converters are computationally intensive to simulate due to their stiff and highly discontinuous characteristics, with fast switching events and slower reactive components' dynamics.

When simulating Simscape power electronic networks, you can choose between using a variable-step global Simulink solver and a fixed-step local Simscape solver. The table below gives details, advantages, disadvantages, and recommended uses for each solver option.

When simulating switch-mode power electronic converters you can use the `ode23t` variable-step solver. The parameter values below generally work well for most power electronic models:

- Set the **Relative tolerance** to `1e-4`.
- Set the **Absolute tolerance** to `1e-4`.
- Disable the **Auto scale absolute tolerance** parameter.

Try

Set the simulation stop time to `0.05` seconds and simulate the model. Press **Stop** if the simulation is taking too long. What solver is being used?

Open the Configuration Parameters (select **Model Settings** in the **Modeling** tab) and select the **Solver** pane. Set the **Solver** to `ode23t` and change the solver tolerances to the values given on this page.

Try to simulate a model again. Does the model simulate successfully?

```
>> boostConv4
```

You can always change solver settings to balance simulation accuracy with speed. Next, you will learn how to switch this model to use a local fixed-step solver.

Note For faster simulation but less accuracy, you can use the variable-step `odeN` solver with proper choice of the integration method. This lightweight solver is able to accurately resolve zero crossings but does not control the simulation error. The recommended integration method is `ode1be` (**Backward Euler**).

	Variable-step global Simulink solver	Fixed-step local Simscape solver
Recommended solvers	<code>ode23t</code> , <code>ode15s</code>	<code>Backward Euler</code> to start, can switch to others later
Simulink behavior	Simulink treats the network as a continuous system.	Simulink treats the network as a discrete system.
Advantages	<ul style="list-style-type: none"> • Ensures simulation error within specified tolerances • Resolves zero crossings accurately • Features specialized solvers for stiff systems 	<ul style="list-style-type: none"> • Provides more robust solver • Supports code generation workflows • Features specialized solvers for Simscape networks
Disadvantages	<ul style="list-style-type: none"> • Simulation may not converge for large or very stiff models • Can drastically slow down simulation to resolve rapid changes or zero crossings 	<ul style="list-style-type: none"> • No control of simulation error • No zero crossing detection • User must carefully choose a time step size
Recommended uses	<ul style="list-style-type: none"> • Smaller models and individual converters • General power electronic modeling and development 	<ul style="list-style-type: none"> • Large models and/or long simulation times • Simulations with rapid changes or many zero crossings • Code generation and hardware-in-the-loop testing

Verifying the Converter

After the model is simulated and you have scopes to visualize the data, you can test and verify the three steady-state operating conditions presented at the beginning of this chapter.

1. Boost Converter operates in Continuous Current Mode (CCM)

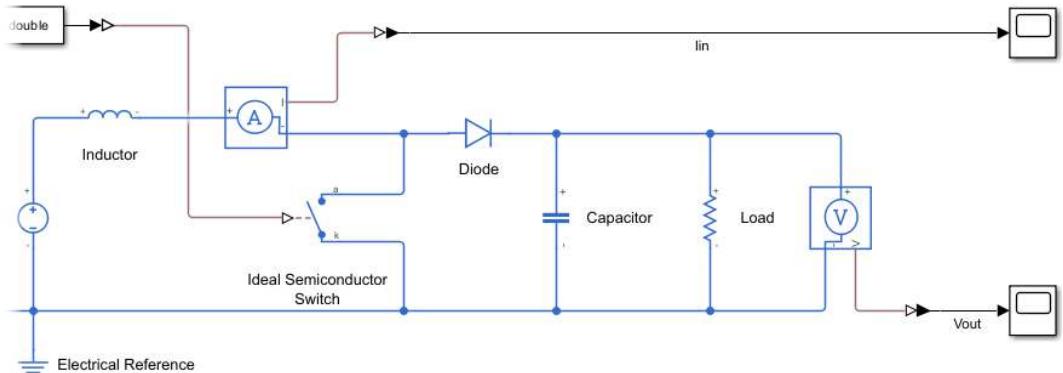
If the boost converter is operating in CCM, the input current (i.e., inductor current) should be greater than zero in steady state. You can see in the scope that once the converter reaches steady state, the input current is always positive. Thus, the boost converter is operating in continuous current mode.

2. Input Current is related to the Duty Cycle

At a duty cycle value of 0.4, the boost converter has an average input current of about 250 A at steady state. If you increase the duty cycle, the input current also increases.

3. Boosted Output Voltage is related to the Duty Cycle

At a duty cycle value of 0.4, the boost converter has an average output voltage of about 397 V. If you increase the duty cycle, the output voltage also increases.



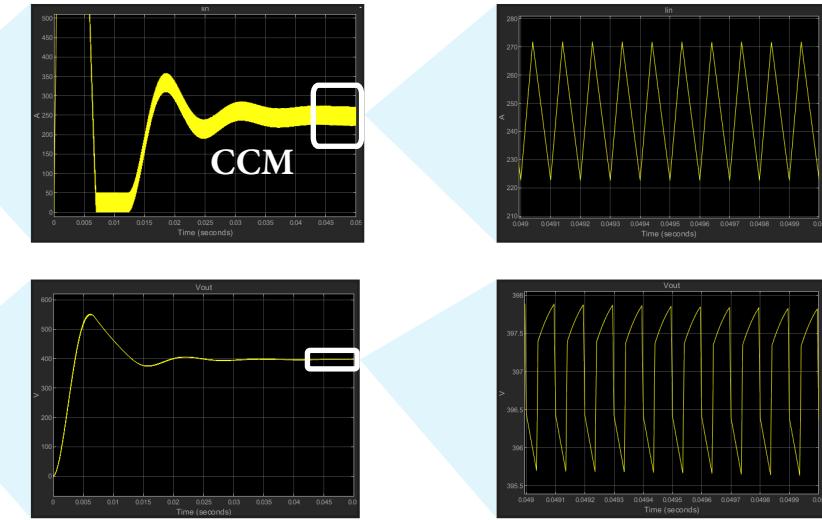
Try

Verify that the three steady-state operating conditions are satisfied by the boost converter.

From the Solver Configuration block, try enabling the **Use local solver** with **Solver type** set to **Backward Euler** and **Sample time** equal to **5e - 6**. Experiment with other sample time values and inspect the results. When finished, disable the local solver and resave the model.

Note You can enable the **Information Overlays > Units** option in the **Debug** tab to display units in the model and on the scope axes. To display time units in a scope, in the Time tab from the scope Configuration Properties you can set the **Time units** to **Seconds** or **Metric** (based on Time Span) and check the box **Show time-axis label**.

The simulation results show higher input current and lower output voltage compared to theoretical expectations. The discrepancy can be attributed to parasitic losses that exist in the model. In the next chapter, you will learn how to characterize these losses. The ultimate goal for this boost converter is to build a controller to maintain a steady output voltage of 400 V under all operating conditions.



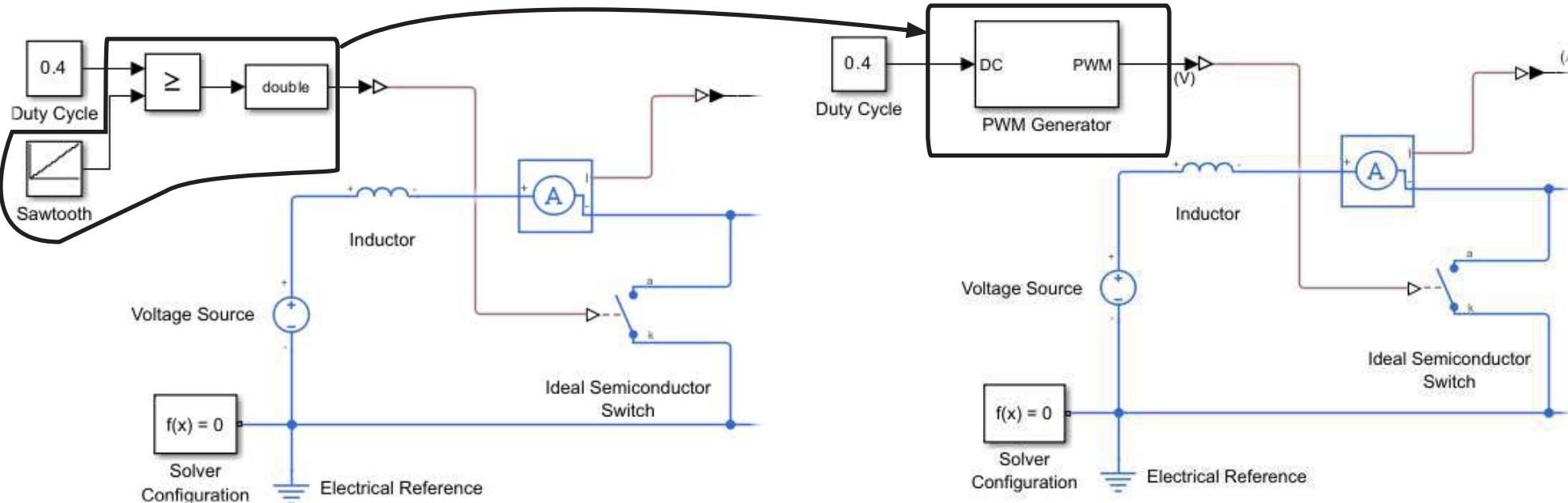
Using Prebuilt PWM Blocks

You have so far modeled a *custom-built* PWM using Simulink blocks. This approach comes with advantages in terms of flexibility for the development of ad-hoc modulation techniques. You are now going to explore *prebuilt* PWMs that are available as library blocks. The library **Electrical > Control > Pulse Width Modulation** is equipped with several PWMs serving specific power electronic topologies.

Starting from **boostConv4** follow the steps:

1. Replace the custom PWM implementation with the PWM Generator block. Set the **Timer period (s)** to **1e-4**, and specify a **Sample time** of **5e-6**.
2. Run and inspect the results.
3. Save the model as **boostConv5**.

Note The PWM Generator block implements a carrier-based algorithm. See the documentation for more details.



Try

Follow the steps listed on this page to replace the custom-built PWM implementation with the PWM Generator block.

Compare the simulation results of the two following models. Do they produce the same results?

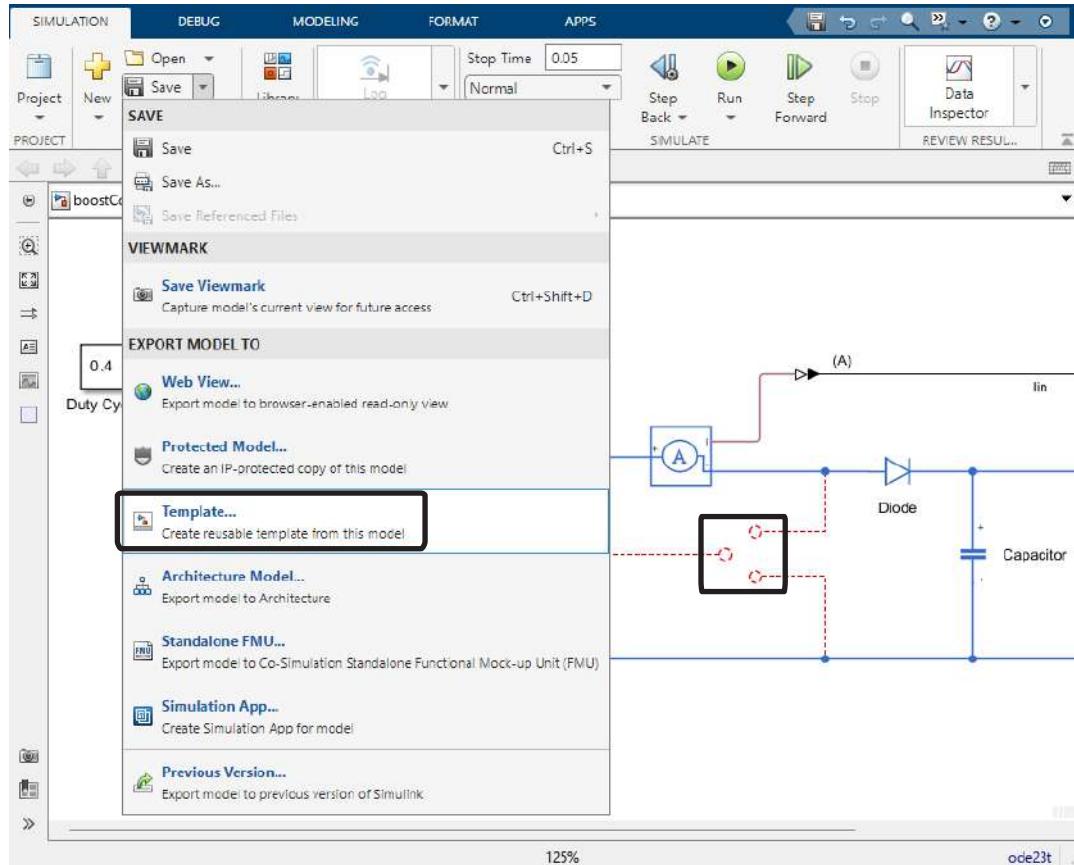
```
>> boostConv4
>> boostConv5
```

For the PWM Generator block, try experimenting with other carrier counters. Refer to the documentation for more information on the types of carrier counters.

Exporting as Template

In the next chapter, you will replace the Ideal Semiconductor Switch with a semiconductor component that can better capture the switching behavior of real semiconductor devices. To ease this task, you can prepare your model including the recommended solver settings along with other simulation settings and export it as a template. Follow the steps starting from `boostConv5`:

1. Delete the Ideal Semiconductor Switch.
2. Click on **Templates...** under the **Save** drop-down menu in the **Simulation** tab.

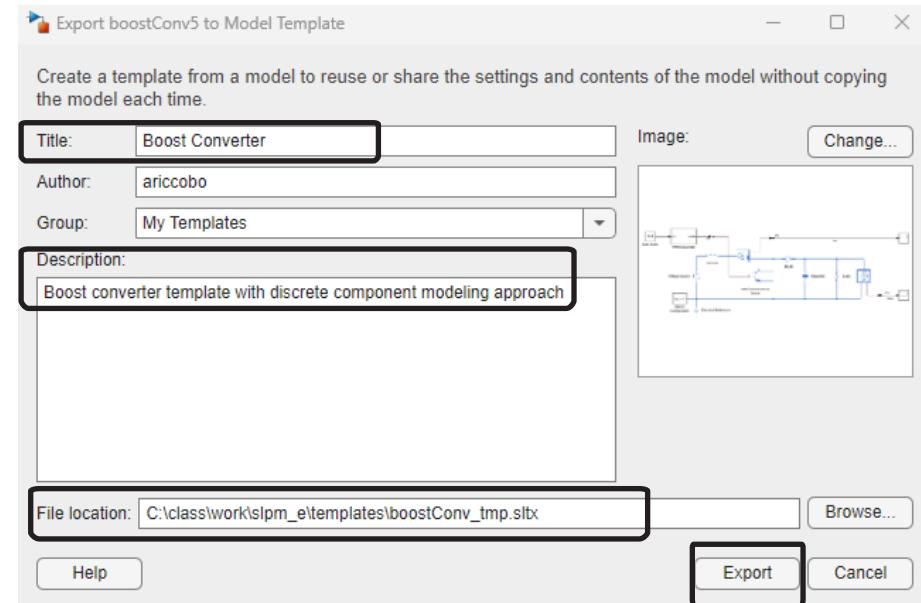


Try

Follow the steps listed on this page to export the model `boostConv5` as a template.

3. In the Export to Model Template dialog box, provide the title, description, file name, and location as depicted below.
4. Click on Export and ignore the following warning message.

Note The project folder . . . \templates is not in the project. That's why this folder is not visible from the Project View. However, since it is on the project path, your template will be available in the Simulink Startup page whenever this course project is open.



Summary

- Modeling a boost converter
- Measuring physical quantities
- Visualizing results
- Selecting a solver

Test Your Knowledge

1. (T/F): The Resistor block from the **Simscape > Electrical > Passive** library provides additional parameters that are not available in the Resistor block from the **Simscape > Foundation Library > Electrical > Electrical Elements** library.
2. The _____ block should be oriented to measure through a branch in a Simscape model. The _____ block should be oriented to measure across two points in a Simscape model.
 - A. Ideal Voltage Sensor, Ideal Current Sensor
 - B. Ideal Current Sensor, Ideal Voltage Sensor
 - C. Neither
3. (T/F): You can connect the measured voltage output from an Ideal Voltage Sensor block directly to a Scope block.
4. Which solver will slow down the simulation to resolve rapid changes or zero crossings?
 - A. Variable-step global solver
 - B. Fixed-step local solver
 - C. Both
 - D. Neither

Answers

1. T
2. B
3. F
4. A



Power Electronics Control Design with Simulink® and
Simscape™

Converter Model Fidelity

Converter Model Fidelity

Outline

- Selecting appropriate converter model fidelity
- Using prebuilt components
- Using discrete component modeling approach

Converter Model Fidelity

Chapter Learning Outcomes

The attendee will be able to

- Build models at an appropriate level of converter model fidelity.
- Understand what can be simulated with prebuilt converters and with converters made out of discrete components.

Converter Model Fidelity

Understanding Fidelity

In this course, the term fidelity refers to how closely a model, converter, or component represents its real-life physical counterpart. The terms “low” and “high” are used to communicate the degree of fidelity, as described below:

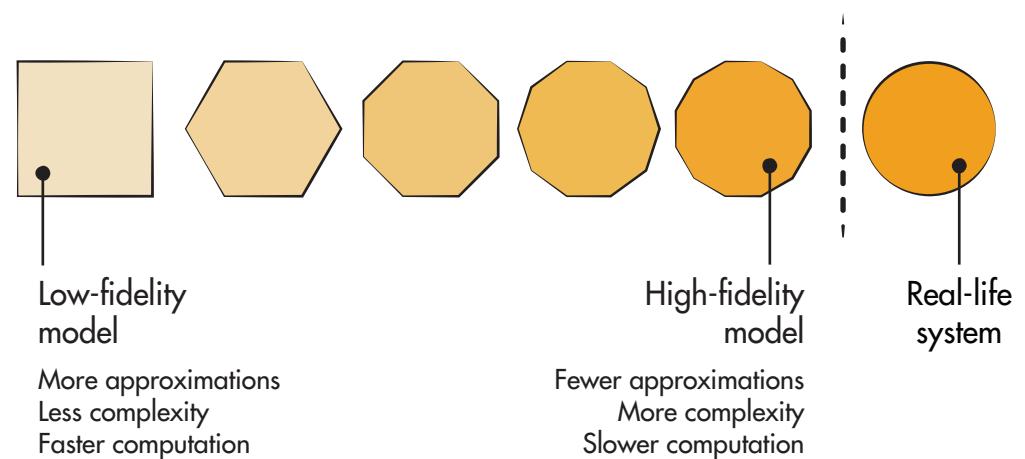
- **Low fidelity** – The system loosely represents the real-life physical system. The model may neglect system dynamics entirely or approximate them with simplified models.
- **High fidelity** – The system closely represents the real-life physical system. The model accounts for many factors that affect system dynamics.

Typically, low-fidelity models offer limited parameters but simulate quickly due to their simplicity. High-fidelity models offer many parameters that allow for precise tuning and customization but simulate slower due to the added complexity.

When modeling power electronic converters, many physical factors can influence the converter model fidelity. Examples include the following:

- Switching dynamics
- Parasitic losses
- Thermal effects
- Nonideal power sources
- Component tolerances
- Electrical noise

In this chapter you will learn how to control the converter model fidelity of a boost converter model, the purpose of the different fidelities, and how they affect the simulation results and performance.



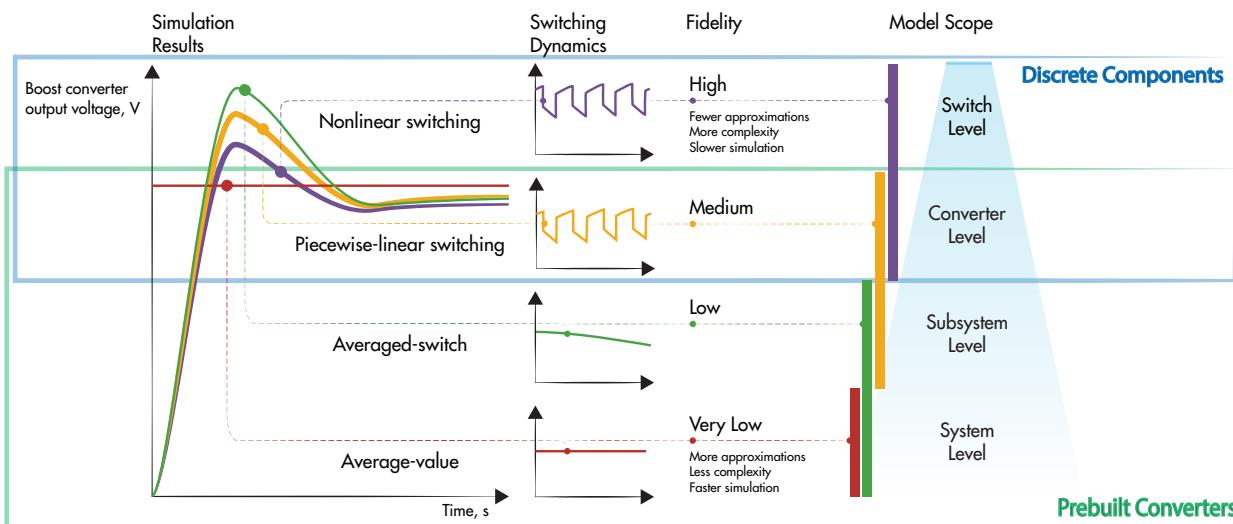
Converter Model Fidelity

Understanding Fidelity (Continued)

In this chapter, you will see how converter model fidelity influences simulation behavior. In Simscape Electrical, there are four levels of fidelity that you can use to model power electronic converters:

- Average-value** – Uses algebraic equations to calculate the converter output from the duty cycle without modeling any converter dynamics
- Averaged-switch** – Uses time-averaged PWM and a simplified circuit to model converter dynamics without modeling switching events
- Piecewise-linear switching** – Uses piecewise-linear equations to model converter dynamics with switching events
- Nonlinear switching** – Uses nonlinear equations to model converter dynamics with switching events

You can use the graphic below to help select a fidelity that matches the scope of your model. Typically lower fidelity models are used in large-size systems, while higher fidelity models are used for switch and converter level analyses and control design purposes.



Try

Explore the blocks in the **Semiconductors & Converters** and **Semiconductors & Converters > Converters** libraries.

As shown in previous chapters, power electronic converters are made up of active components (the semiconductor switching devices) and passive reactive components (inductors and capacitors) connected in a way to form a specific converter topology. There are two modeling approaches to build power electronic converters:

Discrete components

You have so far used this approach. You can rely on the **Electrical>Semiconductors&Converters** library for discrete semiconductor devices, such as transistors and diodes, and on the **Electrical > Passive** library for the passive reactive components to build your converter topologies. You can implement piecewise-linear and nonlinear switching models with this modeling approach.

Prebuilt converters

The **Electrical>Semiconductors&Converters>Converters** library provides prebuilt power electronic converters, inverters, choppers, and gate multiplexers. With this modeling approach, you can choose blocks already equipped with all the components comprising a specific converter topology. You can implement average value, averaged switch, and piecewise linear switching models with this modeling approach.

Note For both modeling approaches, you may still need to use blocks from **Electrical > Sources** and **Electrical > Passive** libraries to model the input voltage and output load, respectively.

Converter Model Fidelity

Course Example: Boost Converter with Prebuilt Converters

This chapter first uses prebuilt converter blocks in Simscape Electrical to demonstrate average-value, averaged-switch, and then piecewise-linear switching models for the boost converter. Later in the chapter, discrete components will be used to implement the fully nonlinear switching model for the same topology. You will learn how to structure a comparative analysis to assess the performance of each level of fidelity.

To get started with prebuilt converter modeling approach, you can rely on a “start model”:

1. Open the model **boostConv1_preBuilt_start** from the **Project Shortcuts** tab.
2. Inspect the model and read the annotations.

Alternatively, you can open a new model from a Simscape Electrical template.

1. Click the **Simulink** button in the MATLAB toolbar.
2. Expand the **Simscape** section of the **New** tab.
3. Click **Electrical**.

Make the following changes to the new model:

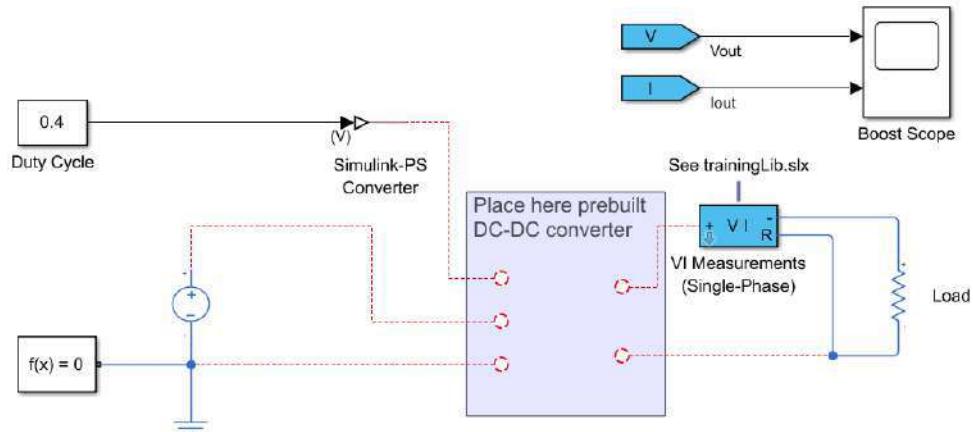
4. Delete the PS-Simulink Converter blocks, the Spectrum Analyzer block, and the annotation.
5. Add a Voltage Source block (**Electrical > Sources** library) and set the **DC Voltage** to **250 V**.
6. Add a Constant block (**Simulink > Sources** library) and set the **Constant value** to **0.4**. Name the block **Duty Cycle**.
7. Add a Resistor (**Electrical > Passive** library), name it **Load**, and set the **Resistance** to **2.67 Ohms**.
8. Open the **trainingLib.slx** library from the **Project Shortcuts** tab.

Try

Use either the “start model” **boostConv1_preBuilt_start** or the Simulink Start Page to create a new model from the **Electrical** template. Use the steps described on this page to modify the new model.

>> trainingLib

- Drag the VI Measurements (Single-Phase) block from the library into your model. This block contains a Voltage Sensor block and a Current Sensor block to help organize electrical measurements. The output measurements are transmitted using Goto blocks and can be retrieved using a From block.
- Drag the Scope block that is already set with time units. Rename the Scope block to **Boost Scope**.
- 9. Add two From blocks (**Simulink > Signal Routing** library) to the model. Change the **Goto tag** parameters to **V** and **I**, respectively.
- 10. Connect all blocks to match the model shown below.
- 11. Label the signals connected to the scope block **Vout** and **Iout**, respectively.



Converter Model Fidelity

Using Average-Value Models

Average-value models simulate very quickly but produce results that capture only the output voltage dependency from the duty cycle without any converter dynamics. This is good for simulating converters in system-level models, enabling you to observe system-level characteristics without slowing the simulation with calculations of converter dynamics.

Average-value models feature the lowest converter model fidelity. These models use algebraic relationships to calculate the converter output without modeling any converter dynamics.

To build an average-value boost converter,

1. Add an Average-Value DC-DC converter block (**Electrical > Semiconductors & Converters > Converters** library) and set the **Converter Type** parameter to **Boost converter**.

Furthermore, you need the following settings before simulating the model.

2. Enable the **Information Overlays > Units** option in the **Debug** tab to display units in the model and on the scope axes.
3. Open the Configuration Parameters and set the following parameters in the Solver pane:
 - **Solver** is **ode23t**
 - **Relative error** and **Absolute error** are **1e - 4**
 - Disable the **Auto scale absolute tolerance** parameter
 - Set the simulation **Stop time** to **0 . 05** seconds.

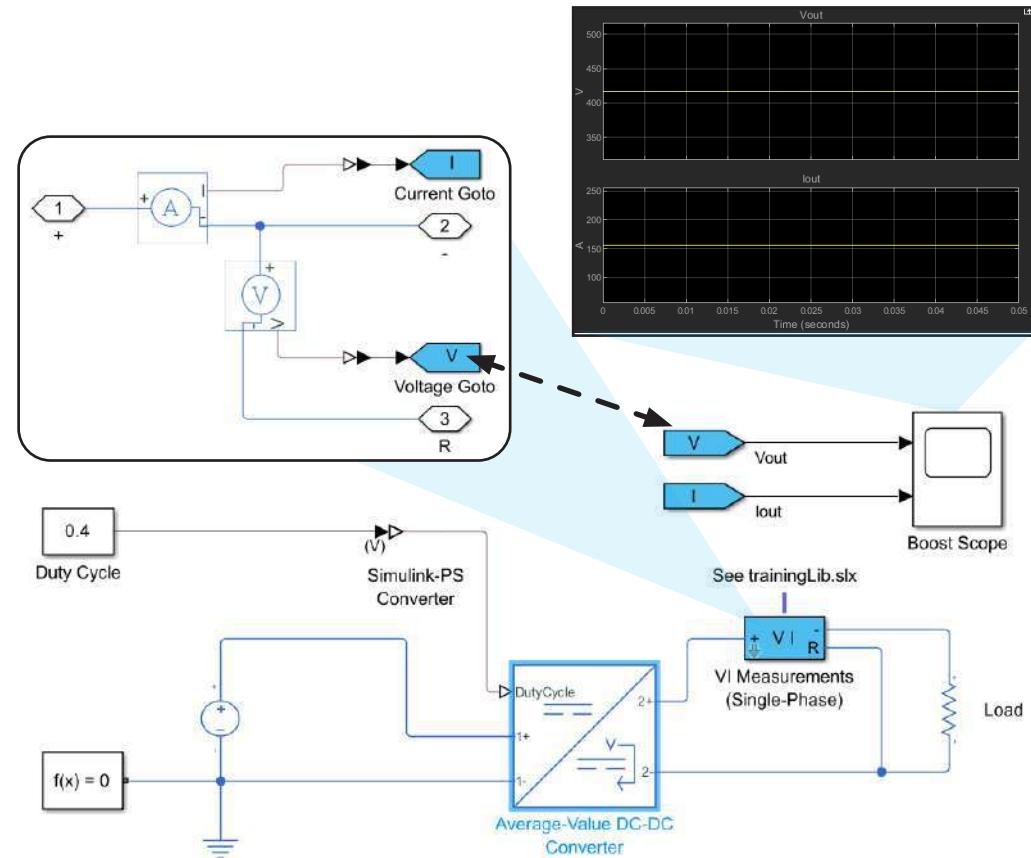
Note As a model grows, you should keep it organized to maintain readability. You can change the background color of From and Goto blocks to improve the model's readability. To do this, select a block and set the **Background** parameter in the **Format** tab.

Try

Follow the steps on this page to build an average-value boost converter model. Simulate the model and inspect the results.

When finished, save the model as **boostConv2_prebuilt_avgValue**.

```
>> trainingLib
>> boostConv2_prebuilt_avgValue
```



Converter Model Fidelity

Using Averaged-Switch Models

Averaged-switch models simulate quickly and produce more detailed results compared to average-value models. This is good for simulating system-level and subsystem-level models with more detailed dynamic responses, but still without the need to simulate the individual switching events.

Averaged-switch converter models use an averaged switch with an antiparallel diode able to be *partially opened* when the duty cycle is between 0 and 1. The converter dynamics include the inductor-capacitor circuit as well. The converter output is calculated by considering the circuit response and the time-averaged value of the PWM signal driving the switching devices.

To build an averaged-switch boost converter,

1. Replace the Average-Value DC-DC Converter with a Boost Converter block (**Electrical > Semiconductors & Converters > Converters** library). Set the following block parameters.

In the **Switching Device** section:

- **Switching device is Averaged Switch**
- **On-state resistance is 10 mOhm**

In the **Diode** section:

- **Forward voltage is 0.6 V**
- **On resistance is 2 mOhm**
- **Off conductance is 1e-8 1/0hm**

In the **LC parameters** section:

- **Inductance is 195 uH**
- **Inductor series resistance is 34.7e-3 Ohm**
- **Capacitance is 7 mF**
- **Capacitor effective series resistance is 6.67e-3 Ohm**

2. Connect all blocks to match the model shown on the right.

Note#1 To calculate the time-averaged value of the PWM signal, you can

- **Use the duty cycle** – This is the simplest approach. You can undersample the model and use modulation waveforms instead of PWM pulses.

Try

Follow the steps on this page to convert the average-value model to an averaged-switch model. Simulate the model and inspect the results.

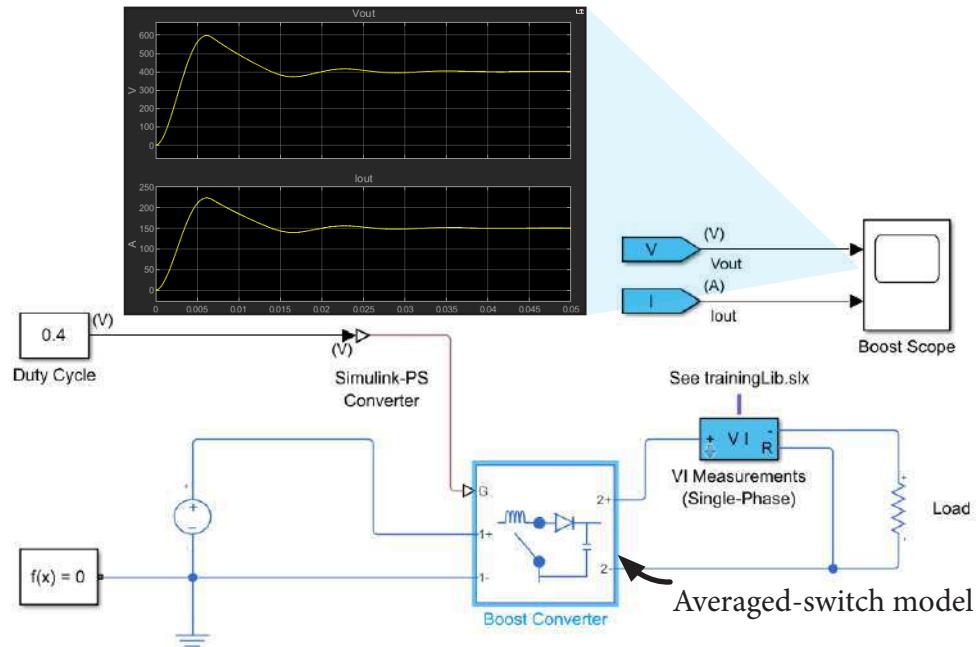
Then, save the model as **boostConv3_prebuilt_avgSwitch**.

```
>> boostConv3_prebuilt_avgSwitch
```

- **Average PWM pulses** – This approach requires additional computation but gives more control over the averaging operation.

Both approaches are useful to speed up hardware-in-the-loop simulations.

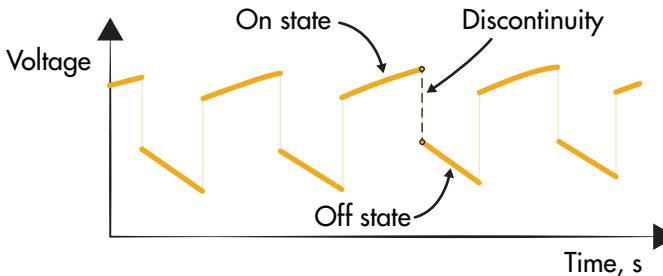
Note#2 With the default **Modeling option** to **Nonsynchronous converter**, power flow is unidirectional from the source to the load due to the blocking forward diode. If you set the **Modeling option** to **Synchronous converter**, the topology comprises two switching devices. Special care is required with the driving duty cycle. This is covered in the final chapter.



Converter Model Fidelity

Using Piecewise-Linear Switching Models

Piecewise-linear switching models generally take more time to simulate than averaged-switch models because the solver needs to take time steps small enough to resolve each switching event. Controllers are often tuned with this level of fidelity. Piecewise-linear switching models use piecewise-linear equations to mathematically describe the “on” and “off” states of the switch. A change in the switch state (i.e., switching event) creates a discontinuity in the waveforms.



To convert your averaged-switch boost converter to use a piecewise-linear switching model,

1. In the Boost Converter block, set the **Switching device** parameter to **IGBT**, set the **Off-state conductance** to **$1e-8$ 1/0hm**, and set the **Threshold voltage** to **0.5 V**.
2. Add a PWM Generator block (**Electrical > Control > Pulse Width Modulation** library) to the model. Set the **Timer period** to **$1e-4$** and the **Sample time** to **$5e-6$** .
3. Connect the blocks as shown on the right.

The type of solver used will affect how the switching events are resolved:

- **Variable-step solver** – Zero-crossing detection will automatically ensure that the switching events are resolved in the simulation. Remember that large numbers of zero crossings can slow simulations.
- **Fixed-step solver** – Choose a fixed-step size equal to the PWM sample time. Remember that smaller time steps lead to slower simulations.

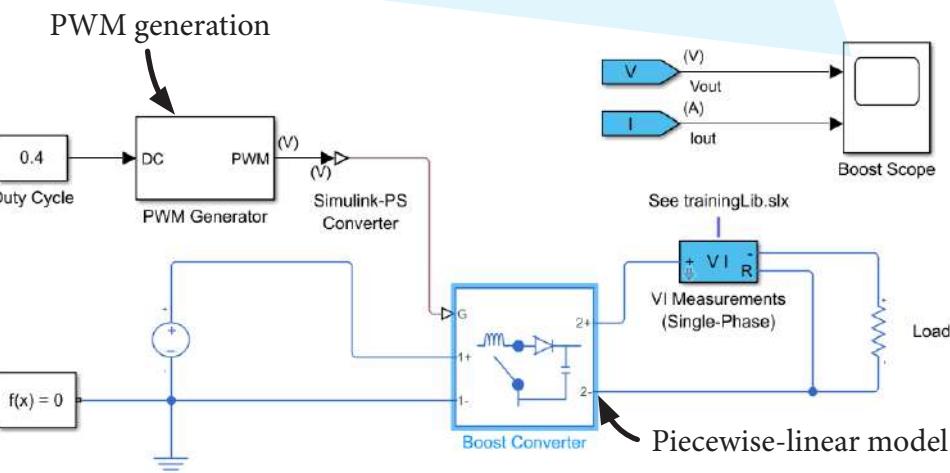
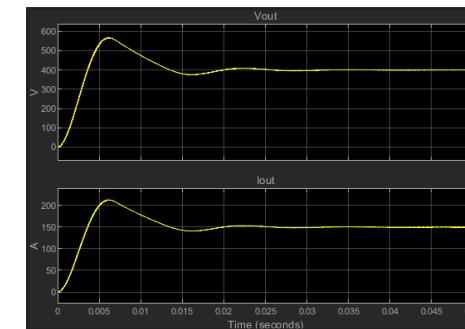
Try

Follow the steps on this page to convert the averaged-switch model into a piecewise-linear switching model. Simulate the model and inspect the results.

Then, save the model as **boostConv4_prebuilt_linSwitch**.

```
>> boostConv4_prebuilt_linSwitch
```

Note Many converter and chopper blocks in the **Electrical > Semiconductors & Converters > Converters** library provide easy options to swap from piecewise-linear switching to averaged-switch and vice versa. PWM generation is required with piecewise-linear switching models, while you can drive averaged-switch models directly with the duty cycle.



Converter Model Fidelity

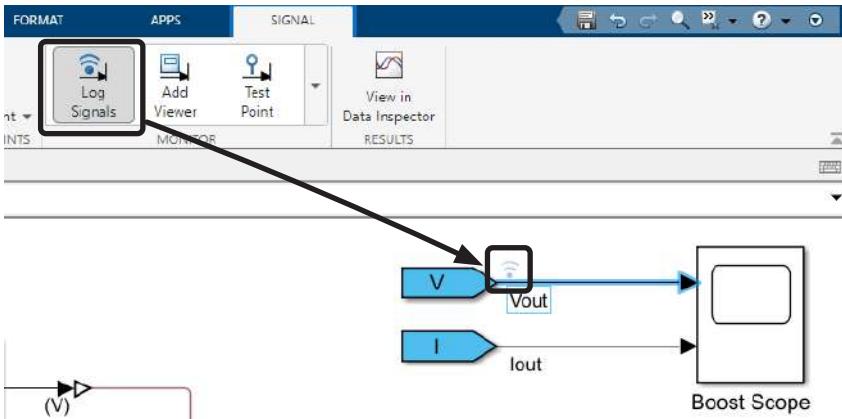
Using Piecewise-Linear Switching Models (Continued)

The blocks labeled “Ideal”, “Ideal, Switching”, and “Piecewise Linear” in the Electrical > Semiconductors & Converters library all use piecewise-linear switching models. In the previous chapter, you in practice implemented a piecewise-linear switching model because you used the IGBT (Ideal, Switching) block and Diode block (with **Diode model** set to **Piecewise linear**).

You can perform a consistency check for the models with discrete components and with prebuilt converter that implement the same level of fidelity.

1. Open the following models: **boostConv4_prebuilt_linSwitch** and **boostConv1_semiconductor**.
2. Click to select the signal labeled **Vout** and enable the **Log Signals** in the **Signal** (or **Simulation**) tab. Do this for both models.
3. Run the models.
4. Open the Simulation Data Inspector and compare the two simulation runs.

With no implementation or parametrization errors, the two simulation runs will produce identical results.



Try

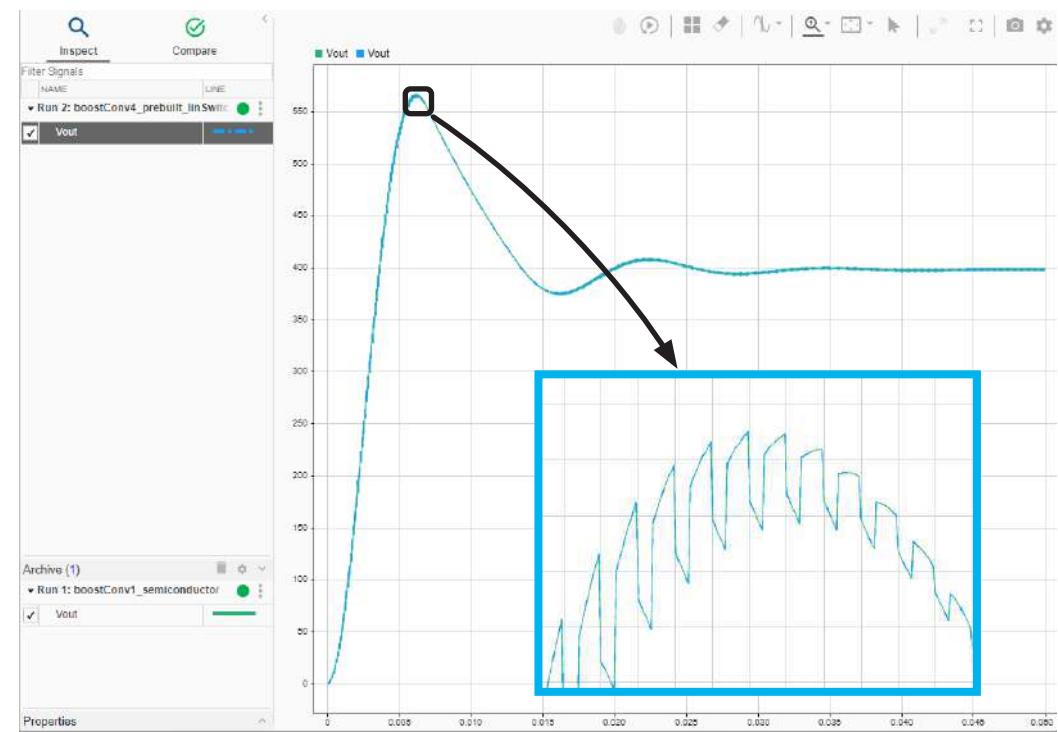
Follow the steps on this page to perform a consistency check for the two following models implementing the same level of fidelity.

```
>> boostConv4_prebuilt_linSwitch  
>> boostConv1_semiconductor
```

Note With prebuilt converters, you **cannot**

- Model switching losses
- Show the thermal port
- Use pre-parametrization workflows.

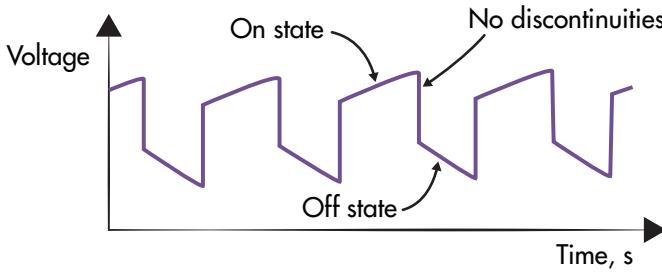
Piecewise-linear switching models are widely used for linearization and control applications as well as for simulating converter-level and subsystem-level models with discontinuous dynamic responses due to switching.



Converter Model Fidelity

Using Nonlinear Switching Models

Nonlinear switching models use nonlinear equations to mathematically describe the “on” and “off” behavior of the switch. Although there may be sharp transients, switching waveforms do not have any discontinuities.



In the **Electrical > Semiconductors & Converters** library, the blocks labeled “N-Channel”, “P-Channel”, “NPN”, and “PNP” all use nonlinear switching models. To build a converter with a nonlinear switching model, you will need to use these semiconductor devices. Notice that the Diode block can be parametrized for nonlinear switching behavior as well by setting the **Diode model** to **Exponential** or **Tabulated I-V curve**.

Simscape Electrical blocks that use nonlinear switching models provide many parameters for customization. This enables you to fine-tune the blocks’ static and dynamic characteristics. You can refer to a device’s manufacturer datasheet to obtain all relevant parameter values. If any parameter values are unknown, you can do one of the following:

- Manually tune unknown parameter values.
- Write an optimization routine to find optimal parameter values that generate the expected response characteristics given in the datasheet.
- Use the Response Optimization tool (included with Simulink Design Optimization™) to find optimal parameter values.

Starting from the **boostConv5_discComp_nonlinSwitch_start** model, you will use an N-Channel IGBT block to incorporate a nonlinear

Try

Open the **boostConv5_discComp_nonlinSwitch_start** model. Follow the steps on this page to add an N-Channel IGBT block to the model. After parameterizing the block, view its basic characteristics.

switching model in your boost converter model. To do this,

1. Add an N-Channel IGBT block (**Electrical > Semiconductors & Converters** library) to your model. Connect the block as shown below.
2. Set the block parameters to the values shown in the table below. Or, you can use the library block My N-Channel IGBT from **trainingLib**.

To check a nonlinear switching device’s parameterization, right-click the block and select **Electrical > Basic characteristics**. This will plot the block’s basic characteristics using the specified parameter values.

N-Channel IGBT Parameter	Value	Units
Zero gate voltage collector current, I_{ces}	1	mA
Voltage at which I_{ces} is defined	6500	V
Gate-emitter threshold voltage, $V_{ge(th)}$	6.5	V
Collector-emitter saturation voltage, $V_{ce(sat)}$	3.1	V
Collector current at which $V_{ce(sat)}$ is defined	1000	A
Gate-emitter voltage at which $V_{ce(sat)}$ is defined	15	V
Input capacitance, C_{ies}	101	nF
Reverse transfer capacitance, C_{res}	7.0	nF
Output capacitance, C_{oes}	17	nF
Emission coefficient, N	2.8	
Forward Early voltage, V_{AF}	170	V
Collector resistance, R_C	1e-4	Ohm
Emitter resistance, R_E	1e-4	Ohm
Internal gate resistance, R_G	1e-3	Ohm
Forward current transfer ratio, B_F	270	

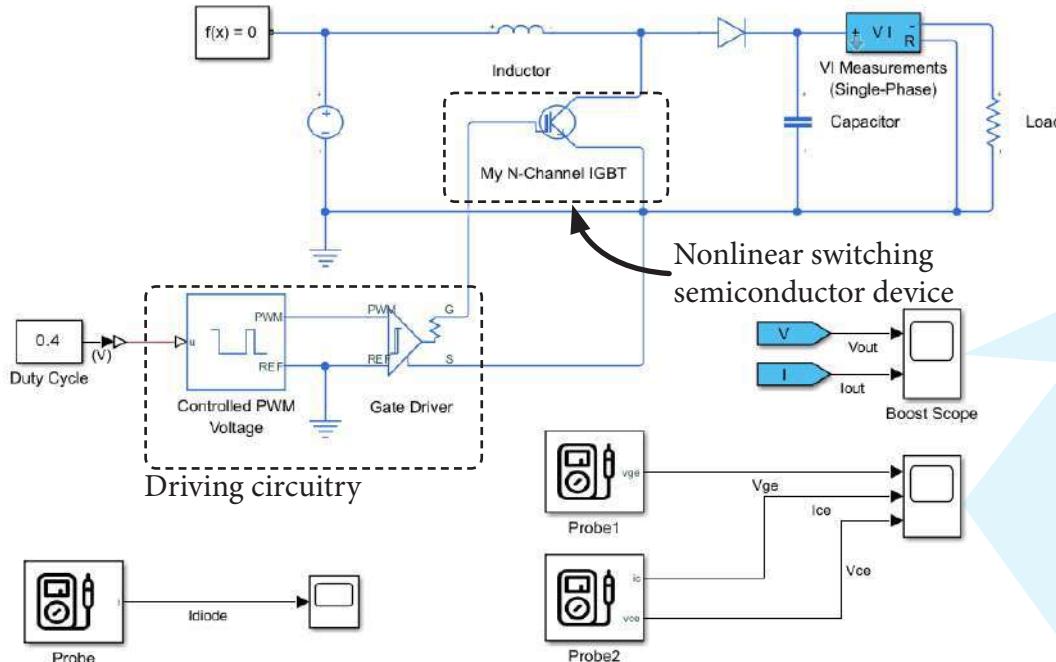
Converter Model Fidelity

Using Nonlinear Switching Models (Continued)

All the three-terminal blocks labeled “N-Channel”, “P-Channel”, “NPN”, and “PNP” have the device’s base or gate terminal exposed as physical connection port. To ensure fast switching of such blocks in presence of nonzero gate (or base) capacitances, you will need to model a gate driver with a specific output impedance able to supply the device gate (or base) with the appropriate voltage levels.

After adding the N-Channel IGBT block to the `boostConv5_discComp_nonlinSwitch_start` model, you need to model the gate driving circuitry. To do that,

1. Add a Controlled PWM Voltage block (**Electrical > Integrated Circuits** library). Double-click the block and set the **Modeling option** to **PS input**. Then, set the **PWM frequency** parameter to 10 kHz.



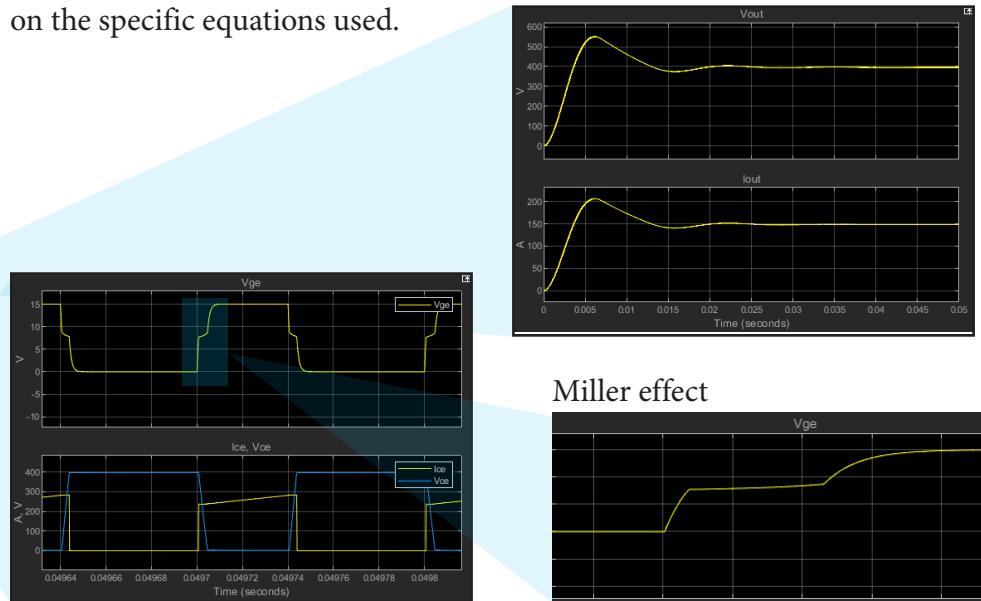
Try

Follow the steps on this page to model a gate driving circuitry, run the simulation, and inspect the results. How do the switching transients look?

>> boostConv6_discComp_nonlinSwitch

2. Add a Gate Driver block (**Electrical > Semiconductors & Converters** library). Double-click the block and set the **Input port** to **Electrical**. In the **Dynamics** section of the block’s dialog parameters, set both the **On-state gate drive resistance** and the **Off-state gate drive resistance** parameters to **9.4 Ohm**.
3. Connect the blocks as shown below.
4. Run the simulation and inspect **Vout**, **Vge**, **Ice**, and **Vce** waveforms. Notice the *Miller effect* due to the IGBT nonlinear capacitances.

Nonlinear switching models generally require the most time to simulate because the solver needs to take time steps small enough to resolve each switching transient and compute the complicated nonlinear equations at each step. You can refer to a block’s documentation for detailed information on the specific equations used.



Converter Model Fidelity

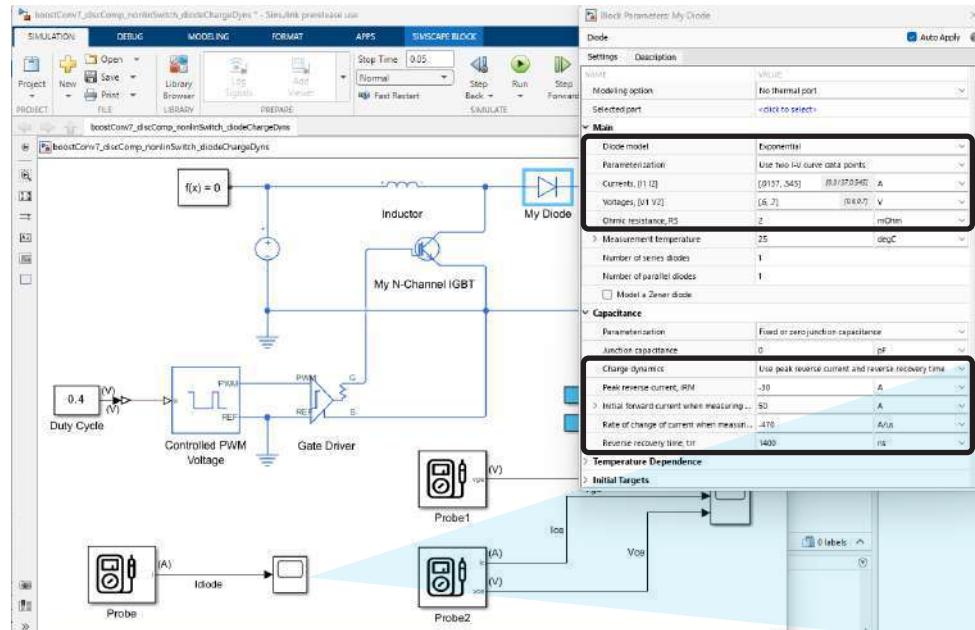
Adding Diode Nonlinear Switching Dynamics

In the previous chapter, you learned that the switching losses of a diode are dominated by the *reverse recovery losses*. You can model these losses for a Diode block by implementing nonlinear switching model. To do this,

1. Double-click the Diode block and parameterize it as shown in the figure below. Or, you can use the library block My Diode from **trainingLib**.
2. Run the simulation.
3. Inspect a switching period of **Idiode**, **Vout**, **Vge**, **Ice**, and **Vce** waveforms.

Reverse recovery transients occur when the diode turns off with negative current spikes that will exponentially extinguish to zero.

As the **boostConv7_discComp_nonlinSwitch_diodeChargeDy**ns model is already set for Simscape logging, you can calculate the efficiency and losses with **ee_getEfficiency()** function, by executing the following function in the Command Window:



Try

Follow the steps on this page to add nonlinear switching dynamics to the Diode block. After parameterizing it, view its basic characteristics.

Simulate the model to appreciate the diode reverse recovery transients.

Calculate the efficiency and losses with **ee_getEfficiency()** function. Compare with the results in the previous chapter.

```
>> boostConv7_discComp_nonlinSwitch_diodeChargeDy
```

```
>> [eff,losstab] = ee_getEfficiency(...  
'Load',out.simlog,0.0495,0.05)
```

Important Switching and conduction losses are not calculated separately with nonlinear switching components as they are with ideal switching blocks.

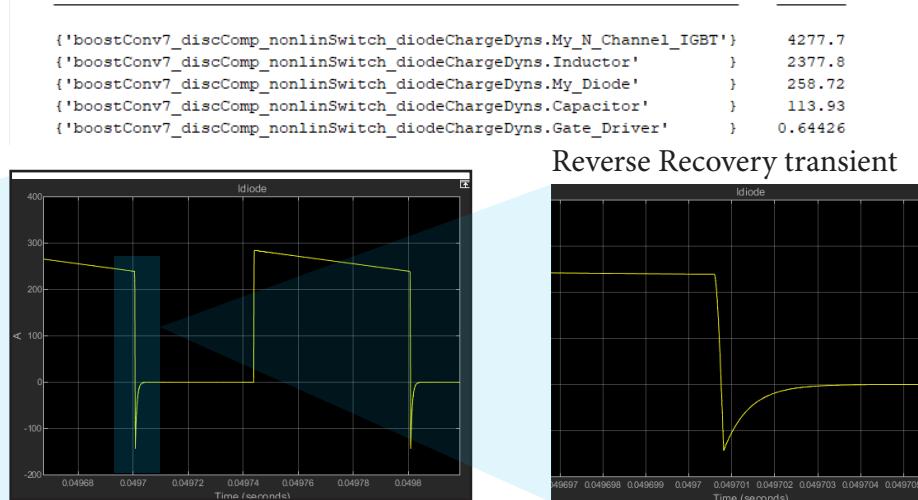
```
eff =
```

```
89.2670
```

```
losstab =
```

```
5x2 table
```

	Power
'boostConv7_discComp_nonlinSwitch_diodeChargeDy.My_N_Channel_IGBT'	4277.7
'boostConv7_discComp_nonlinSwitch_diodeChargeDy.Inductor'	2377.8
'boostConv7_discComp_nonlinSwitch_diodeChargeDy.My_Diode'	258.72
'boostConv7_discComp_nonlinSwitch_diodeChargeDy.Capacitor'	113.93
'boostConv7_discComp_nonlinSwitch_diodeChargeDy.Gate_Driver'	0.64426

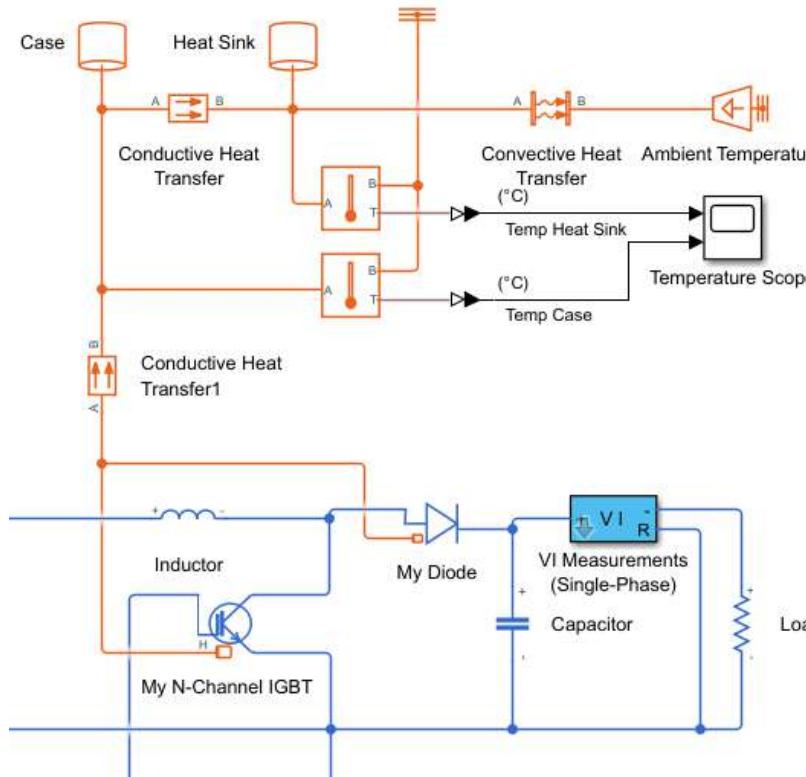


Converter Model Fidelity

Thermal Effects with Nonlinear Switching Models

You can model thermal effects for nonlinear switching semiconductor components by showing the thermal port and choosing an appropriate thermal network. This is conceptually identical to what was covered at the end of the previous chapter.

The `boostConv8_discComp_nonlinSwitch_thermal` model presents an example of a naturally cooled heatsink for the semiconductor switching devices of the boost converter. Both the N-Channel IGBT and Diode blocks have the thermal port exposed and the **Thermal network** parameter set to **External**. The thermal network below is entirely modeled by using the `boostConv8_discComp_nonlinSwitch_thermal`



Try

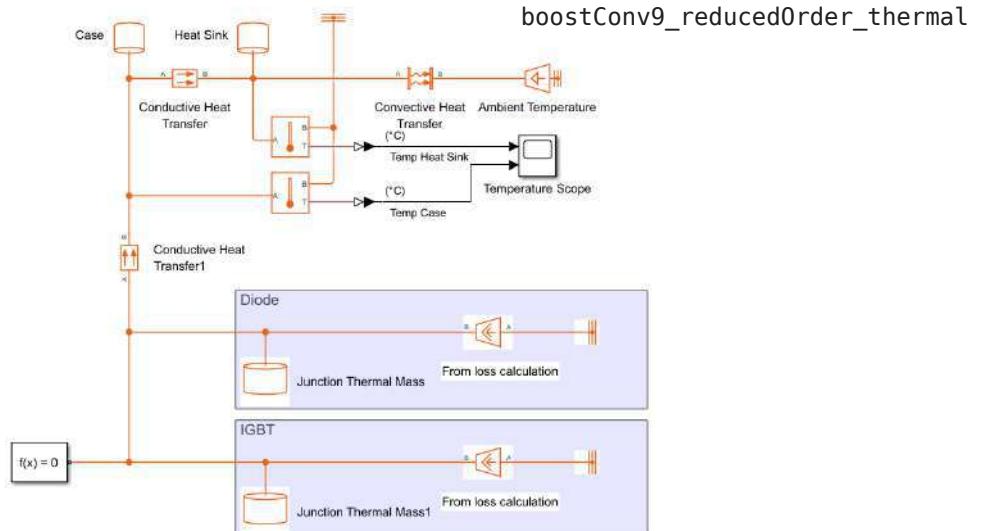
Open the `boostConv8_discComp_nonlinSwitch_thermal` model. View the N-Channel IGBT and Diode blocks' settings for modeling thermal effects. Simulate the model and inspect the results.

Then, open the `boostConv9_reducedOrder_thermal` model. Simulate it to find the steady-state temperatures. Are the measured temperatures within the safe operating range for semiconductor devices?

Simscape > Foundation Library > Thermal library. Within 50 ms of simulation time, the electrical network reaches its steady-state operating condition, but the thermal network would require much longer.

At this point, you can run the `boostConv9_reducedOrder_thermal` that counts for the measured losses for the IGBT and diode and uses them in the Heat Flow Rate Source blocks. After simulating for an hour, you should measure the temperatures of the heatsink and the case. If they are not within the safe range, you should iterate your thermal design.

For more information about thermal effects, search for **Simulating Thermal Effects in Semiconductors** in the documentation.



Converter Model Fidelity

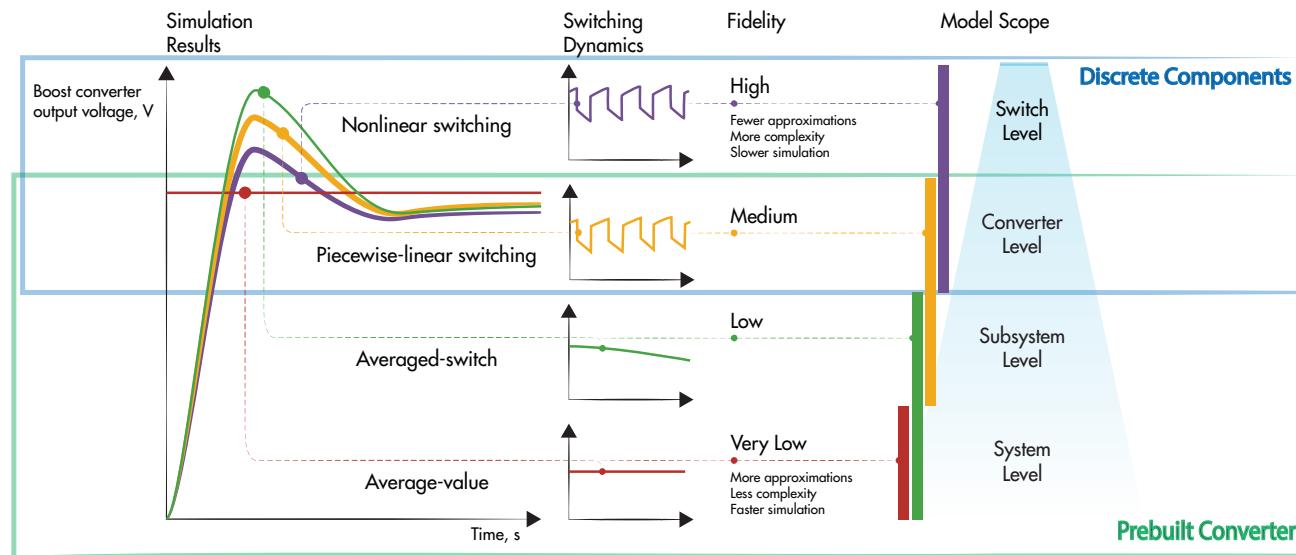
Summary

- Selecting appropriate converter model fidelity
- Using prebuilt components
- Using discrete component modeling approach

Try

To reproduce the plot below, open the following script from the project shortcuts and run it.

```
>> edit compare_boostConv
```



	Average-value	Averaged-switch	Piecewise-linear Switching	Nonlinear Switching
Discrete components?	✗	✗	✓	✓
Prebuilt converters?	✓	✓	✓	✗
Switching signal required?	✗	✗	✓	✓
Converter model fidelity	Very low	Low	Medium	High
Training model	boostConv2_prebuilt_avgValue	boostConv3_prebuilt_avgSwitch	boostConv4_prebuilt_linSwitch	boostConv6_discComp_nonlinSwitch

Converter Model Fidelity

Converter Model Fidelity

Test Your Knowledge

1. (T/F): Piecewise-linear switching models do not require a switching signal to be generated.
2. (Select all that apply) Which of the following converter levels of fidelity are available in the Boost Converter block from the **Electrical > Semiconductors & Converters > Converters** library?
 - A. Average-value
 - B. Averaged-switch
 - C. Piecewise-Linear Switching
 - D. Nonlinear Switching
3. (T/F): Piecewise-linear switching models can be implemented with both prebuilt converter blocks and discrete components.
4. (T/F): When you calculate the losses with `ee_getEfficiency()` for a nonlinear switching model, conduction and switching losses are calculated separately.

Answers

1. F
2. B, C
3. T
4. F



Power Electronics Control Design with Simulink® and
Simscape™

Digital Control Design

Outline

- Implementing a closed-loop voltage discrete PID control
- Setting model and control sample times
- Understanding linearization of power electronic converters
- Frequency Response Estimation
- Tuning the controller
- Test and verification

Chapter Learning Outcomes

The attendee will be able to

- Implement a discrete PID controller.
- Linearize a power electronics design.
- Tune a linearized power electronics design.
- Test and verify the closed-loop performance.

Background and Motivation

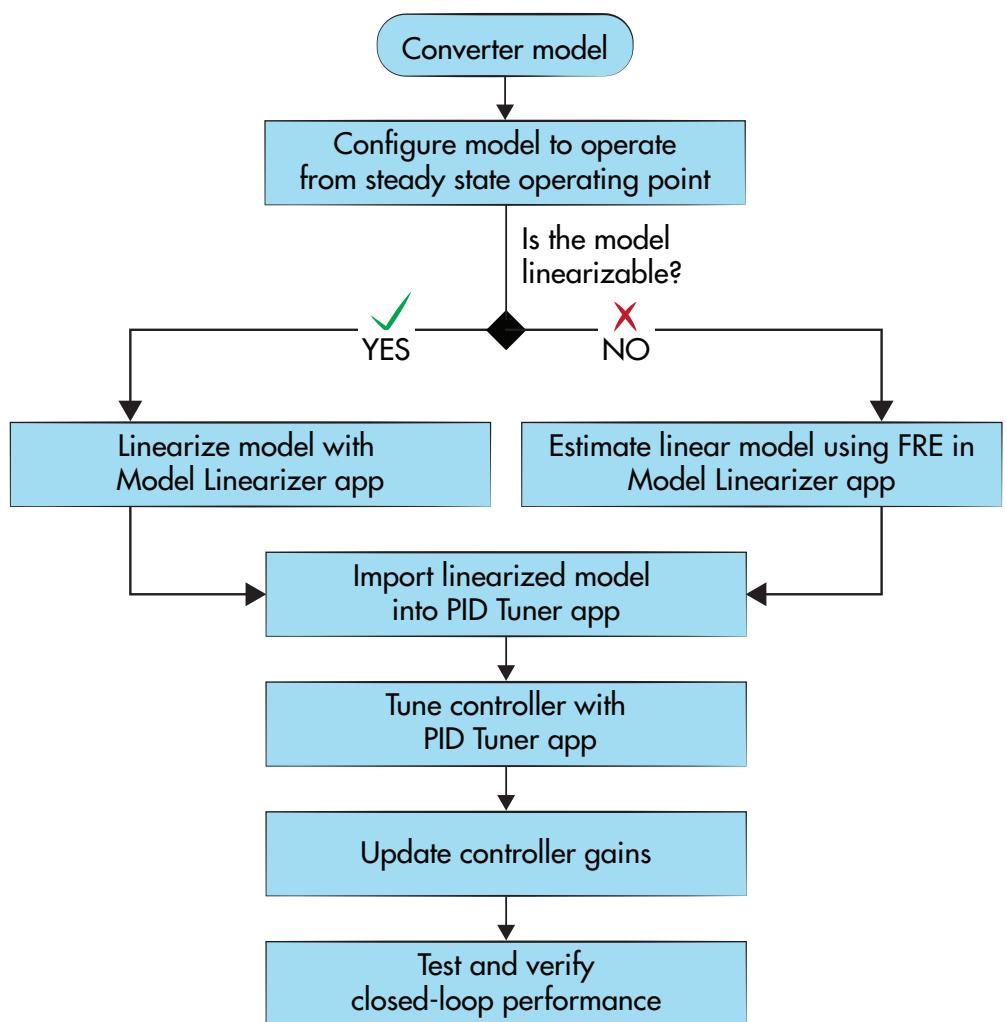
The boost converter models in previous chapters operated in an *open-loop* configuration, where the duty cycle input was independent of the boost converter output. In this chapter, you will implement *closed-loop* control using a discrete proportional-integral-derivative (PID) controller to maintain the boost converter at a desired output voltage. This is known as *voltage-mode control*. The full workflow covered in this chapter is shown on the right.

You will use the PID Tuner app to graphically tune the gains of the PID controller and produce an acceptable response. This tuning process is based on the linear control theory, which requires a linear time-invariant (LTI) representation of the plant model. If the plant model is nonlinear, it must first be linearized in order to use this tuning process.

Power electronic switching converter models (that is, piecewise-linear switching or nonlinear switching models) are generally not linearizable because of the discontinuous or highly nonlinear behavior associated with each switching event/transient. To overcome this difficulty, Frequency Response Estimation (FRE) can be used to find an estimated LTI representation of the open-loop switching converter for control design.

At the end of this chapter, the control system will be tested to verify that it regulates the boost converter output voltage despite changes to the load. In the final chapter, this model will be used to power the HEV motor drive system.

Note This chapter presents an *offline* approach to FRE and controller tuning, where the system is estimated and tuned outside of simulation. However, these steps can also be applied *online*, or during simulation. For more information, see **Simulink Control Design > Control System Design and Tuning > PID Controller Tuning > Tune PID Controller in Real Time Using Closed-Loop PID Autotuner Block** in the documentation.



Digital Control Design

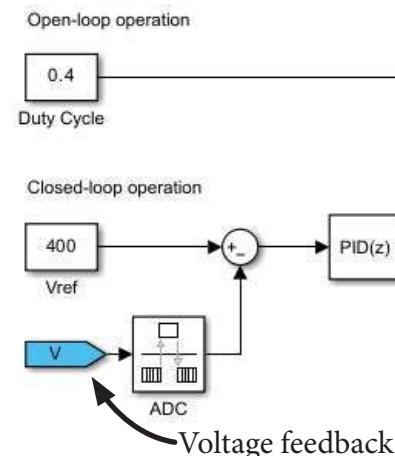
Modeling a Voltage-Mode Controller

Open the `boostConv1_control_start` model. In this model,

- The boost converter operates in open-loop mode with a 40% duty cycle.
- The boost converter block is configured to use a piecewise-linear switching IGBT model.
- The PWM Generator block timer period is set to $1e-4$ s (10 kHz switching frequency) and its sample time is set to $1e-6$ s (sampling frequency 100 times larger than the switching frequency to resolve the duty cycle with 1% increments).

Before going through the control design workflow, you need to implement the closed-loop voltage-mode controller. To do that,

- Add a Sum block (**Simulink > Math Operation** library) and set the **List of signs** to $|+-|$.
- Add a Constant block (**Simulink > Sources** library), name it `Vref`. Set the **Constant value** to **400**, which is the desired output voltage of the boost converter in volts. Set the **Sample time** to **$1e-4$** .
- Add a Rate Transition block (**Simulink > Signal Attributes**), name it `ADC`. Set the **Sample time** to **$1e-4$** . This block mimics an analog-to-digital converter (ADC) sampling the sensed output voltage.
- Add a **From** block (**Simulink > Signal Routing** library), set the **Goto tag** to `V` (this is the voltage measured by the VI Measurements block).
- Add a Discrete PID Controller block (**Simulink > Discrete** library).
- Add a Manual Switch block (**Simulink > Signal Routing** library) to quickly switch between open-loop and closed-loop configurations.



Try

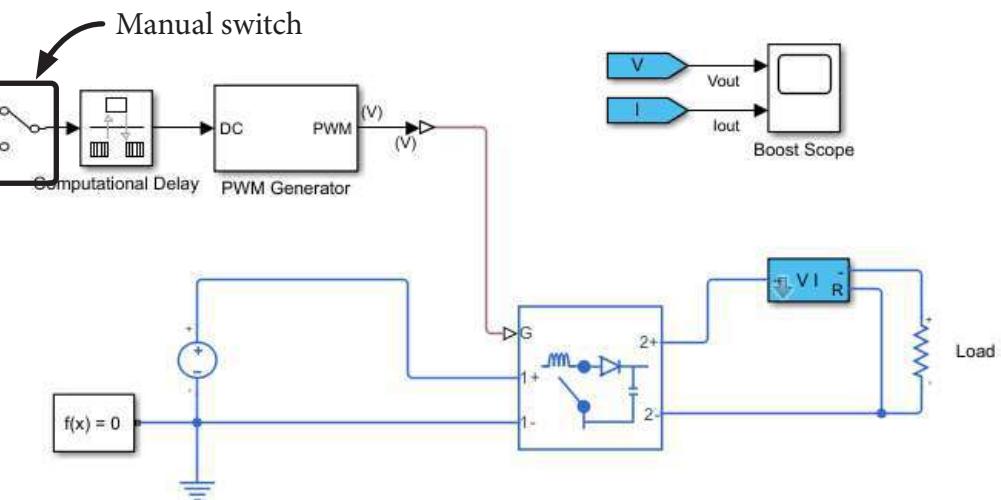
Open the `boostConv1_control_start` model. Follow the steps on this page and save this model as `boostConv2_control`.

Then, simulate the model and verify open-loop performance.

```
>> boostConv2_control
```

- Add another Rate Transition block, name it **Computational Delay**. This is necessary to accommodate the different sampling rates between the two blocks and to mimic the digital delay of real-world controllers.
- Connect all the blocks as shown in the figure below.

Change the position of the Manual Switch block to the closed-loop configuration and simulate the model. You'll notice that the controller does not maintain the desired output voltage around 400 volts. This is because the controller gains still must be tuned. Moreover, notice that the first Rate Transition block acts as a Zero-Order Hold (ZOH), while the second one as a Unit Delay ($1/z$). Then, change the position of the Manual Switch block back to the open-loop configuration.



Digital Control Design

Setting Model and Control Sample Times

Your digitally controlled boost converter model comprises a few different sample times for the model and control. You can select **Information Overlays > Colors** in the **Debug** tab to view the sample times used in the model.

For digitally controlled power electronic converters, the following recommended settings for model and control sample times apply.

Switching period

A power converter has a switching period $T_{sw} = 1/f_{sw}$, where f_{sw} is the switching frequency.

PWM sample time

Choose an adequate PWM sample time T_s^{PWM} to ensure the desired duty cycle resolution according to the following relationship:

$$T_s^{PWM} = T_{sw} / N$$

where N is an integer number that should be chosen between 20 and 50 for fast simulation with still acceptable accuracy, and between 100 and 200 for applications where you need small duty cycle resolution.

Control sample time

The control sample time T_s^{CTRL} is the calculation rate of the software algorithm implemented in the controller. It should be chosen to be something between the PWM sample time and the switching period, that is $T_s^{PWM} \leq T_s^{CTRL} \leq T_{sw}$. Typical choices are $T_s^{CTRL} = T_{sw}$ for single-update uniformly sampled PWMs, and $T_s^{CTRL} = T_{sw} / 2$ for double-update

Try

Analyze the model and control sample times based on the content of this page.

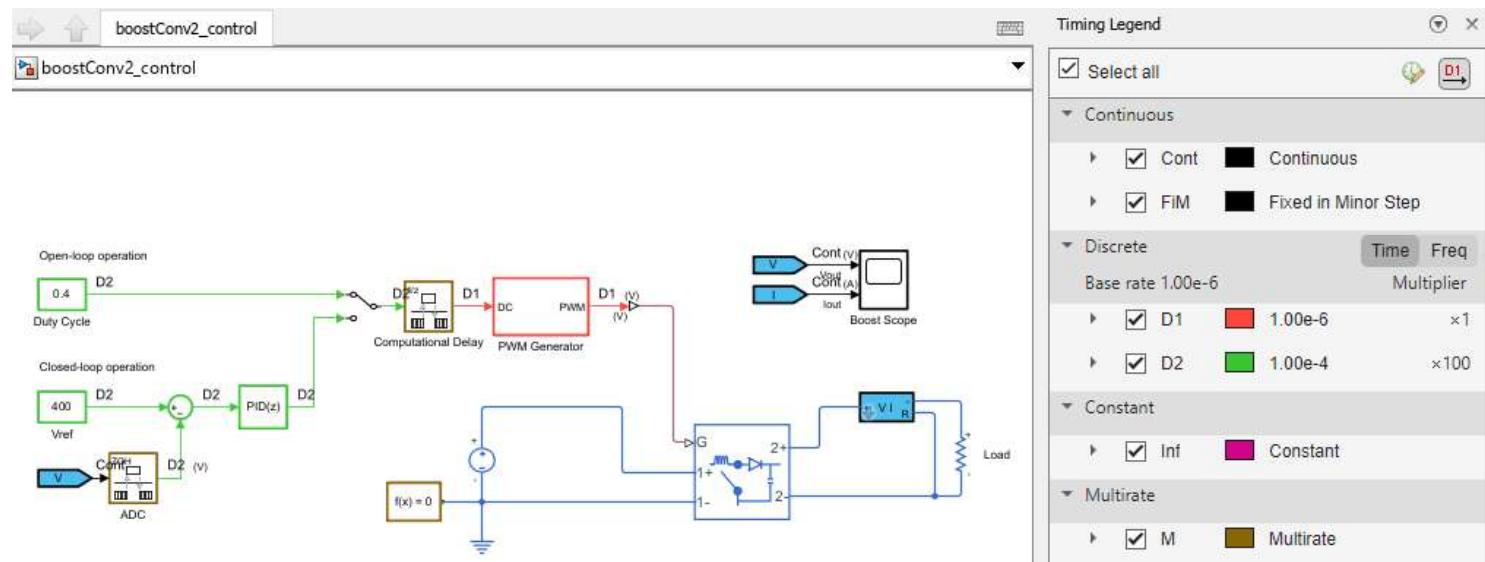
>> boostConv2_control

uniformly sampled PWMs.

Local Solver - the physical network sample time

If you enable the Local Solver, all the physical network states, which are continuous by default, become discrete. You can set the Local Solver sample time T_s as follows:

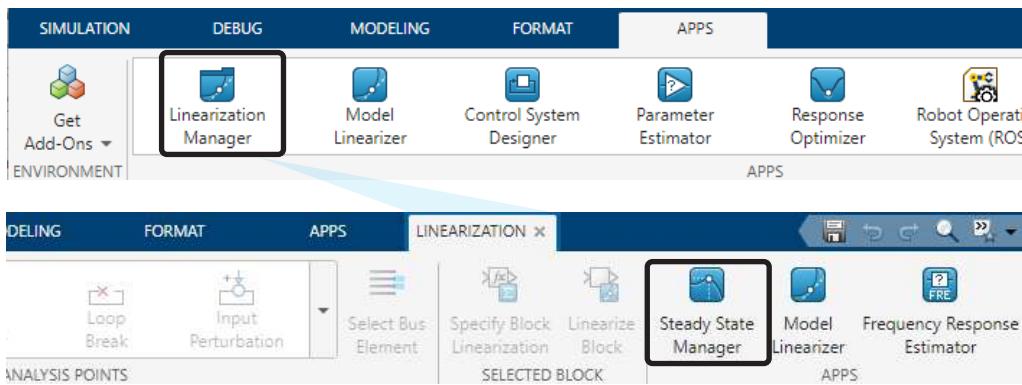
- $T_s = T_s^{PWM}$ for switching models (piecewise-linear and nonlinear)
- $T_s = T_{sw} / M$, where M is a positive integer, for averaged-switching models.



Digital Control Design

Finding a Steady-State Operating Point

After you have implemented the closed-loop controller, you can go ahead and apply the control design workflow. To get started, click on the **Linearization Manager** from the **Apps** tab. This action opens up the **Linearization** tab fully equipped with the tools (part of Simulink Control Design™) in support of the control design workflow.



The first step in the control design workflow is obtaining a steady-state *operating point* for the open-loop system. This operating point will contain information about the system in steady-state operation. It will be used later to generate an estimated LTI object for the open-loop system. To capture a steady-state operating point,

1. Open the Steady State Manager app by clicking **Steady State Manager** in the **Linearization** tab.
2. Click **Snapshots**.
3. In the **Create Snapshot Operating Point** dialog, enter a single time value or an array of time values. Then, click . The tool will simulate the model and capture the input and state information from the model at the specified times. When finished, the details of the operating point will automatically be displayed in a new tab.
4. Click **Set Initial Conditions** in the **Operating Point** tab to
 - Write the operating point to the base workspace (named **op1** by default).

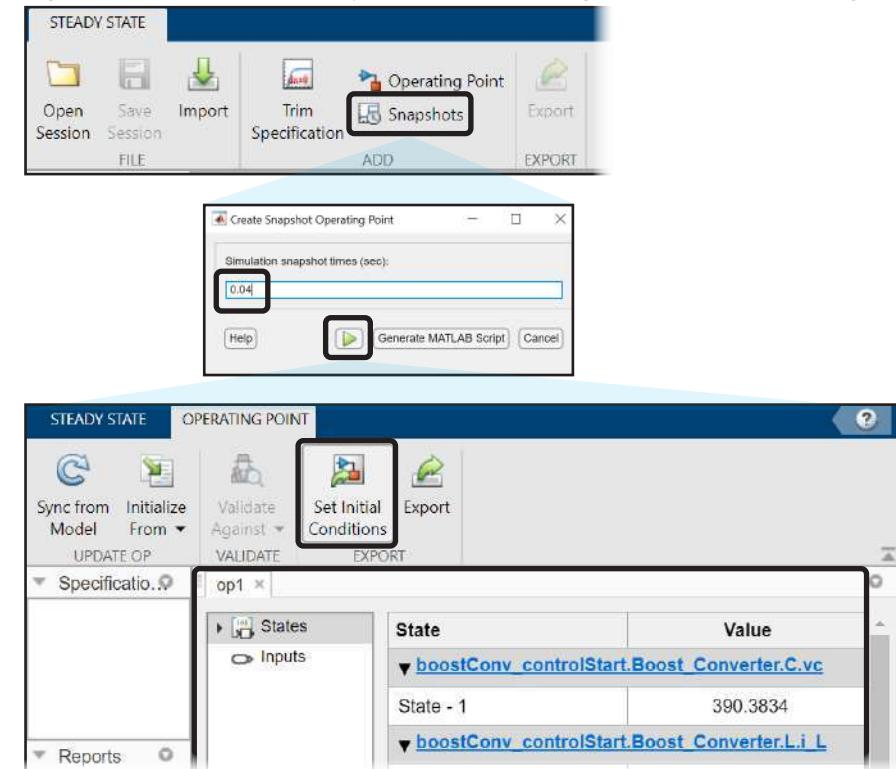
Try

Simulate the model in the open-loop configuration and open the Boost Scope. The voltage output should be in steady state at 0.04 seconds. Follow the steps on this page to obtain a steady-state operating point by snapshotting the model at 0.04 seconds.

Then, initialize the model at the operating point. Verify that the **op1** variable was created in the base workspace. Simulate the model and verify that the model now starts in steady-state operation.

- Update the model's **Input** and **Initial state** options in the **Data Import/Export** pane of the Configuration Parameters dialog.

Note You should save the operating point variable to a MAT-file. If you change the model's filename, you will need to regenerate the operating point.



Digital Control Design

Attempting to Tune the PID Gains

At this point, you will need to tune the controller gains (proportional, integral, and derivative) to obtain the desirable closed-loop system response.

Common methods available for tuning the controller gains are listed below:

- **System response tuning** – You can iteratively modify controller gains and test them via simulation to verify whether they produce the desired response characteristics. This can be time-consuming because many simulations may be required.

Note You can also manually write optimization algorithms, or use the Response Optimizer app in Simulink Design Optimization™ to tune the system response.

- **Linear control theory** – You can apply traditional control design techniques (using Bode plots, for example) to tune the controller gains. This approach requires an LTI representation of the plant model being controlled, and so the plant must be either linear or linearizable. Once the controller performs satisfactorily with the linearized system, you can test the controller gains with the nonlinear system.

Note You can also use automated tuning algorithms based on linear control theory, such as the PID Tuner app in Simulink Control Design.

Control engineers often prefer the latter approach because it enables them to design control systems using familiar traditional techniques. Additionally, these techniques do not require iterating through simulations.

To start tuning the PID controller parameters, double-click the Discrete PID Controller block to open the block parameters dialog and then click **Tune**. This will open the PID Tuner app to tune this controller block.

When the PID Tuner app opens, it attempts to linearize the plant model about the model's initial condition operating point. If the plant cannot be linearized, the app will display a warning.

Power electronic switching converter models (that is, piecewise-linear

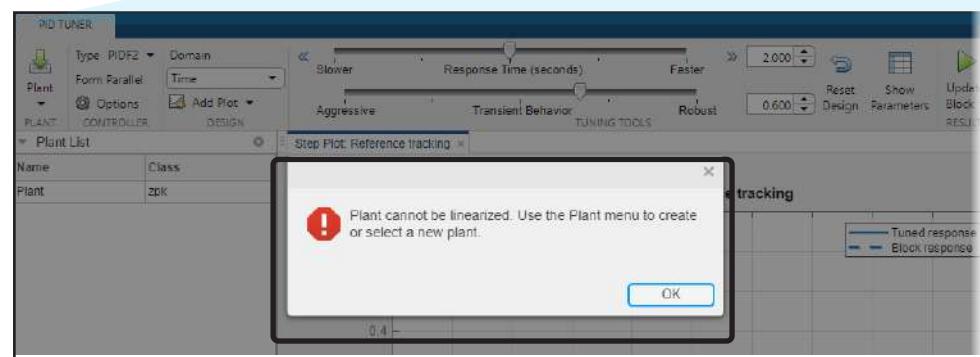
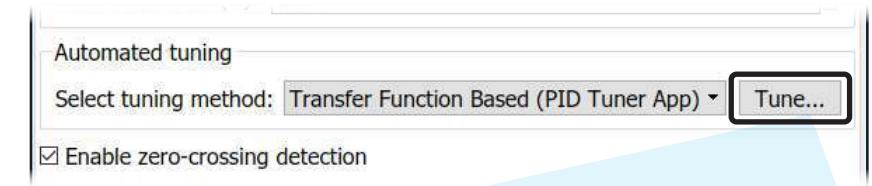
Try

Change the position of the Manual Switch block to enable the closed-loop configuration. Open the PID Tuner app and try to tune the voltage-mode controller. Is the boost converter model linearizable?

When done, change the position of the Manual Switch block back to the open-loop configuration.

switching or nonlinear switching models) generally fail this linearization process because of the discontinuous or highly nonlinear nature of the switching events/transients. To work around this issue, you can create an estimated LTI representation of the boost converter. In this course, you will estimate the plant model in the frequency domain, which is a common approach for power electronics control applications.

With an estimated LTI representation of the boost converter, you will then be able to use the PID Tuner app to tune the controller gains.



Digital Control Design

Understanding Frequency Response Estimation

On the previous page, you saw that the plant could not be linearized. Instead, you will use Frequency Response Estimation (FRE) to obtain an estimated LTI representation of the boost converter plant model.

FRE is the process of superimposing small perturbation signals of controllable amplitude and frequency content onto the input of a system at its steady-state operating point. The input and output response are measured and used to compute the system's frequency response and estimate a transfer function. This transfer function is an estimated LTI system that represents the system dynamics near the operating point.

The perturbation signal can be either

- **Narrowband** – Sine wave containing a single frequency
- **Wideband** – Random, chirp, or pseudo-random binary sequence signals that contain a broad range of frequencies

This chapter focuses on FRE techniques using narrowband perturbation signals. The narrowband approach generally gives high estimation accuracy, but can require more simulation and analysis time than the wideband approaches.

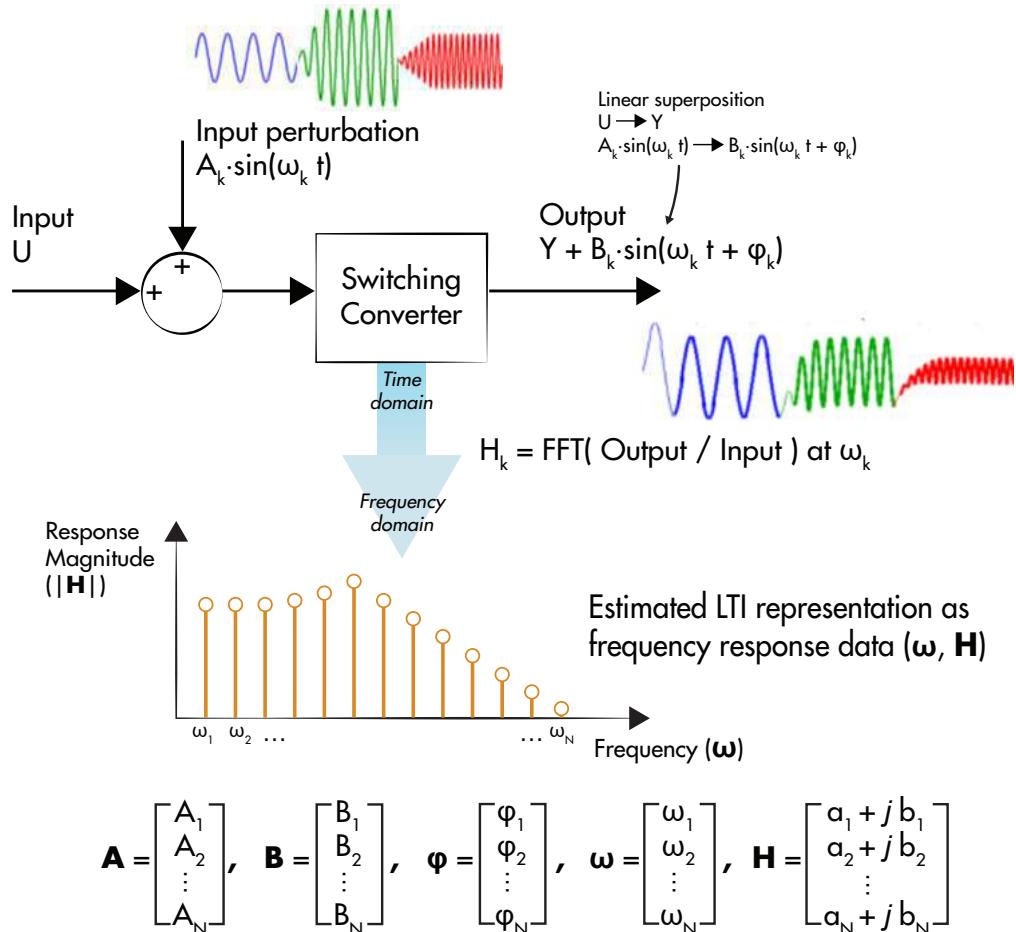
A narrowband perturbation signal is also known as an AC sweep or a sinestream, because the sine wave is being swept across a frequency range to cover the intended frequency spectrum. Specialized equipment widely used in power electronics applications, such as network analyzers, also uses these narrowband perturbations.

Note You can learn more about frequency response estimation in the [Simulink Control Design > Frequency Response Estimation > Frequency Response Estimation Basics](#) documentation.

In Simulink, narrowband frequency response estimation involves

1. Injecting the perturbation signal one frequency at a time.
2. Measuring input and output responses for each injected frequency.
3. Computing the Fast Fourier Transform (FFT) of the input and output responses.

The result is a frequency response data (FRD) object, which contains the frequency response data as a function of the discrete frequency points.



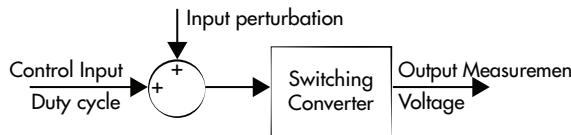
Digital Control Design

Setting Analysis Points

To perform frequency response estimation on the boost converter, you first need to indicate the following analysis points in the model:

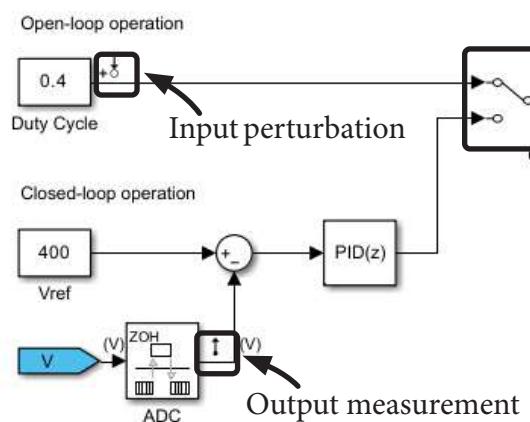
- Input perturbation** – Point where the perturbation will be injected.
- Output measurement** – Point where the output will be measured.

In the boost converter model, the input perturbation analysis point is on the duty cycle signal and the output measurement analysis point is on the sampled voltage signal. This enables you to estimate the power converter control-to-output transfer function, which will be used to tune the controller.



To configure these analysis points,

1. Make sure the position of the Manual Switch block is in the open-loop configuration.
2. From the **Linearization** tab, select the signal connected to the output of the Constant block named **Duty Cycle** and click on **Input Perturbation** in the **Insert Analysis Points** section. The signal will show the $\text{+}\text{\textcircled{d}}$ icon. This perturbation will be defined on the next page.
3. Select the signal connected to the output of the Rate Transition block named **ADC**. Select **Output Measurement** in the **Insert Analysis Points** section. The signal will show the $\text{\textcircled{d}}$ icon.

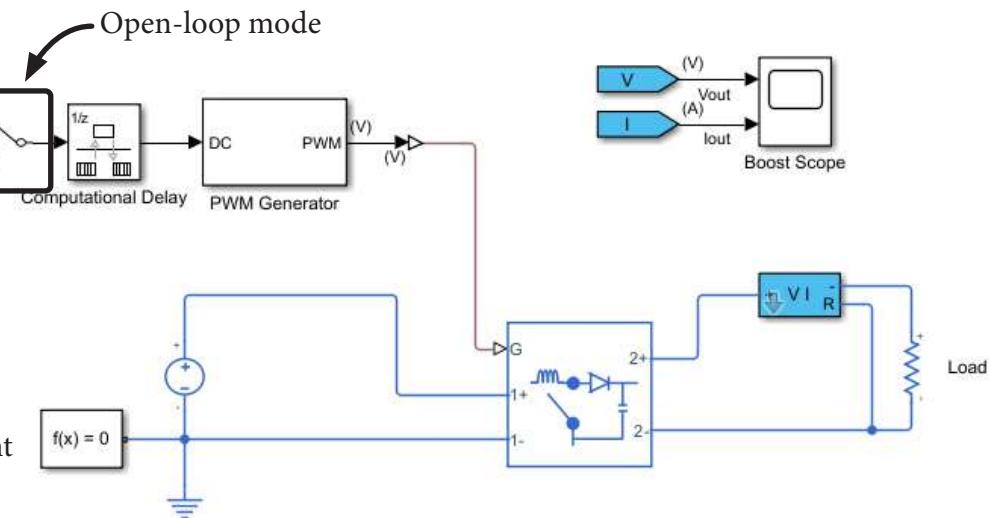
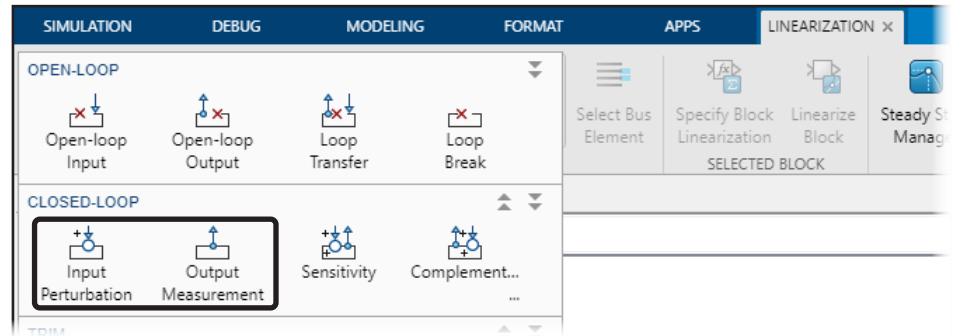


Try

Follow the steps on this page to set the input perturbation and output measurement analysis points in the model.

>> boostConv3_control

Note With the open-loop configuration you could alternatively use the open-loop input and output analysis points. However, you cannot use them with an unstable untuned closed-loop configuration. You will use the open-loop input and output analysis points for a stable untuned closed-loop configuration in Chapter 7. Refer to **Specify Portion of Model to Linearize** in the documentation for more information.



Digital Control Design

Setting Up Frequency Response Estimation

Once the analysis points are specified, you can use the Model Linearizer app to set up the frequency response estimation. To do this,

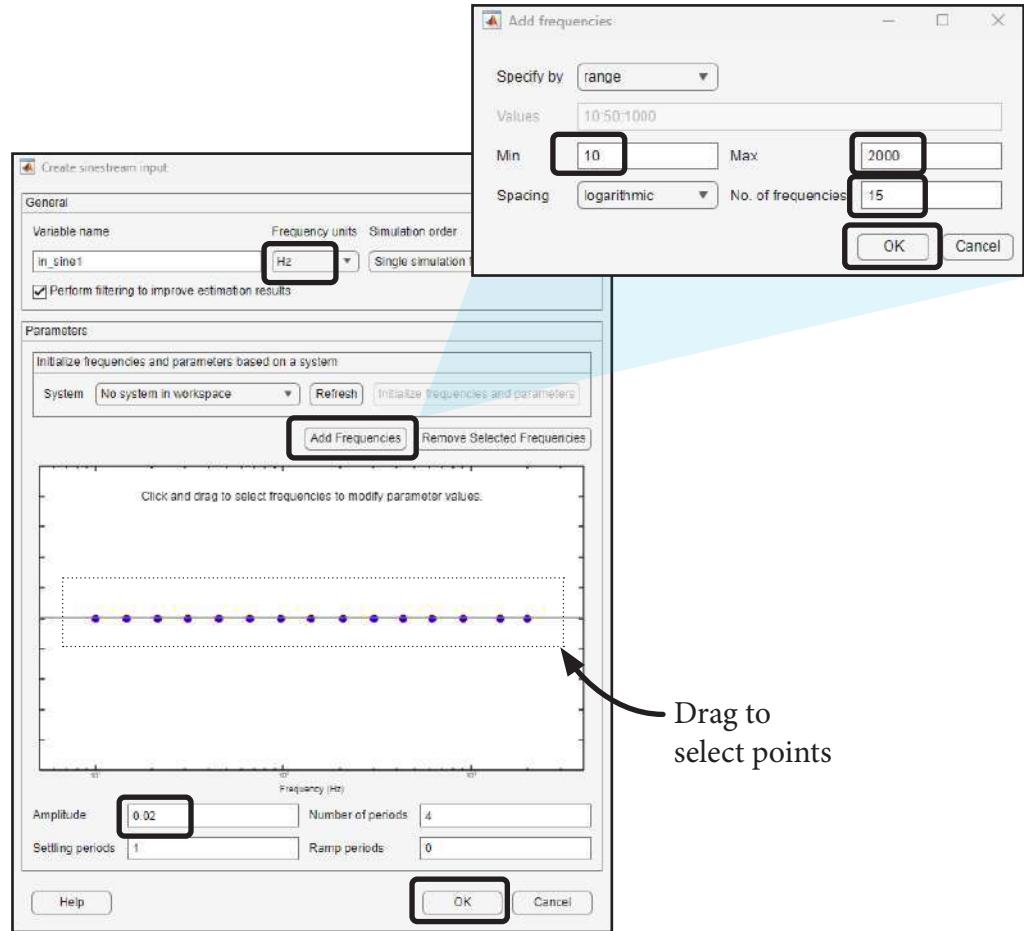
1. Ensure that the model is configured to use the steady-state operating point that you found earlier. Click **Model Settings** in the **Modeling** tab to open the Configuration Parameters dialog. Check that the **Input** and **Initial state** options in the **Data Import/Export** pane are enabled and refer to the correct operating point defined in the base workspace.
2. Then, click **Frequency Response Estimator** in the **Linearization** tab. This will open the **Model Linearizer** app and automatically switch to the **Estimation** tab.
3. Next to **Input Signal** in the **Setup** section, click **Create New > Fixed Sample Time Sinestream**.
4. In the **Specify fixed sample time** dialog, set the **Sample time (sec)** to **1e-4**. Then, click **OK**.
Note This sample time must exactly match the sample times of the input perturbation and output measurement signals.
5. In the **Create sinestream input** dialog, set the **Frequency units** to **Hz**.
6. Click on **Add Frequencies** to define a new sinestream. In the **Add frequencies** dialog, set the parameters shown on the right. Then click **OK**.
7. A dialog may open to notify you that some frequency values have been adjusted. If so, click **Close**. This step ensures that the sampling frequency is an integer multiple of the sinestream frequency values.
8. Click and drag to select all frequencies shown in the plot. Set the **Amplitude** to **0.02**. In general, you should set the amplitude to a value that is from 1% to 5% of the input value to ensure that it is a *small* perturbation. Then, click **OK**.

This will create a Sinestream object in the Linear Analysis Workspace.

Try

Follow the steps on this page to create a new sinestream input to be used in the frequency response estimation.

Note With the Parallel Computing Toolbox™, you can run the simulations for frequency response estimation in parallel. For more information on this workflow, see **Simulink Control Design > Control System Design and Tuning > PID Controller Tuning > Model-Based PID Controller Tuning > Design Controller for Power Electronics Model Using Frequency Response Data** in the documentation.



Digital Control Design

Running Frequency Response Estimation

At this point, the preparations are complete for frequency response estimation. To summarize, these included the following:

- Capturing a steady-state operating point and configuring the model to start simulation from this operating point
- Defining input perturbation and output measurement analysis points in the open-loop model
- Defining a fixed sample time sinestream containing a set of frequencies and amplitudes to perturb the model input

To start the frequency response estimation, click **Bode** in the **Estimate** section of the **Estimation** tab. During the estimation, the model will simulate with the sinestream perturbation signal injected onto the duty cycle signal. The response of the sampled output voltage is measured. The estimation creates a *frequency response data* object, which is the non-parametric estimation.

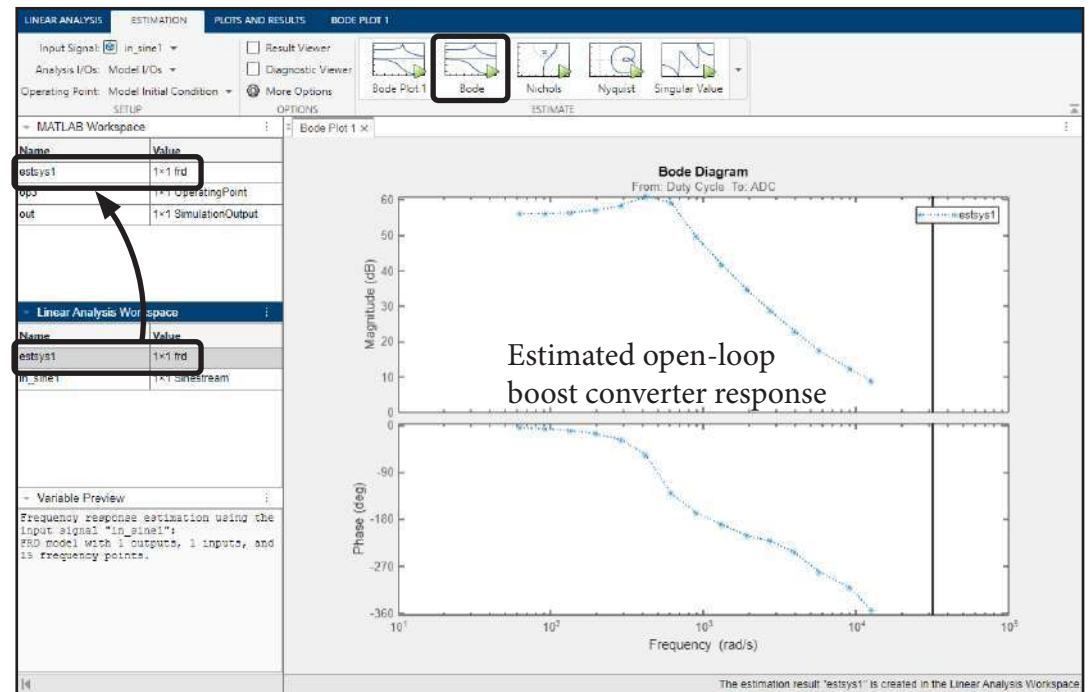
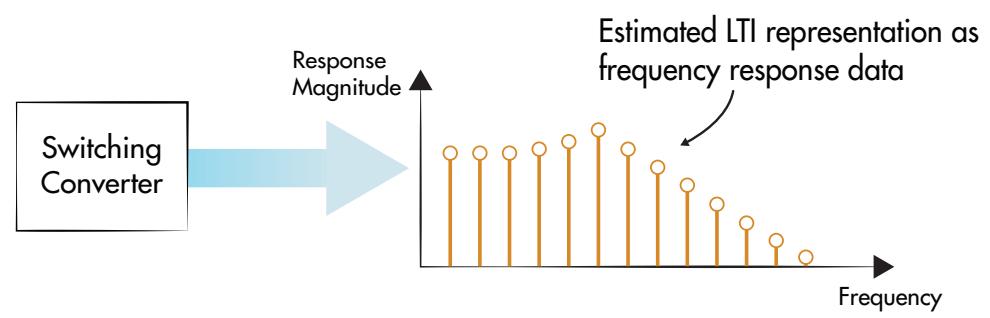
Once the estimation is completed, the Bode plot of the estimated system is displayed. You can right-click in the magnitude or phase sections of the Bode plot and use the **Characteristics** menu to overlay response characteristics. Use these tools to see if the generated results match your expectations for the system.

When finished, you will need to right-click on the `estsys1` object from the **Linear Analysis Workspace** and click on **Export to MATLAB Workspace** so that it will be accessible for controller tuning with the PID Tuner app.

Note Recall that the boost converter model is already configured to start from a steady-state operating point, so you do not need to change the **Operating Points** option. However, you can also use this option to select or generate a different point.

Try

Run the frequency response estimation analysis. Then, export the resulting `estsys1` `frd` object to the base workspace.



Digital Control Design

Tuning the Controller with PID Tuner

Now that you have an estimated LTI representation of the boost converter in the MATLAB workspace, you can use it to tune the voltage-mode controller using the PID Tuner app. To do this,

1. Set the Manual Switch block to operate in closed-loop mode.
2. Double-click the Discrete PID Controller block to open the block dialog parameters. Note the **Proportional (P)**, **Integral (I)**, and **Derivative (D)** default parameter values.
3. Click **Tune** to open the PID Tuner app.
4. As you saw earlier in this chapter, the app will try to linearize the plant. Ignore the linearization failure. Select **Plant > Import**. Then, select the frequency response data object **estsys1** and click **OK**.
5. Then, select **Add Plot > Bode > Open loop**. This will open a Bode plot showing the open-loop transfer function, which includes the controller and plant as shown on the right.
6. Close the **Step Plot: Reference tracking** plot because this plot will not be useful to view the estimated plant's response.

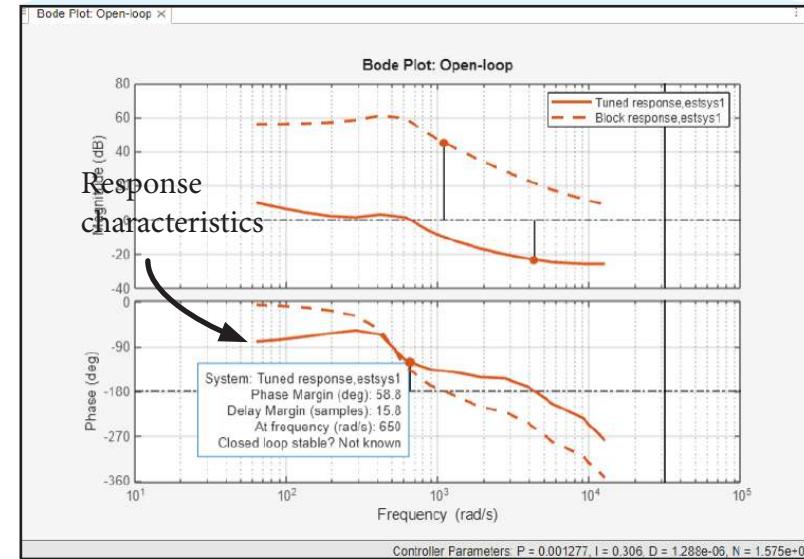
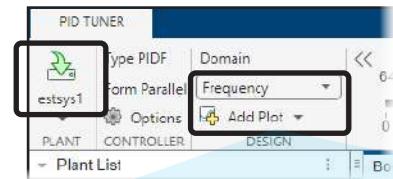
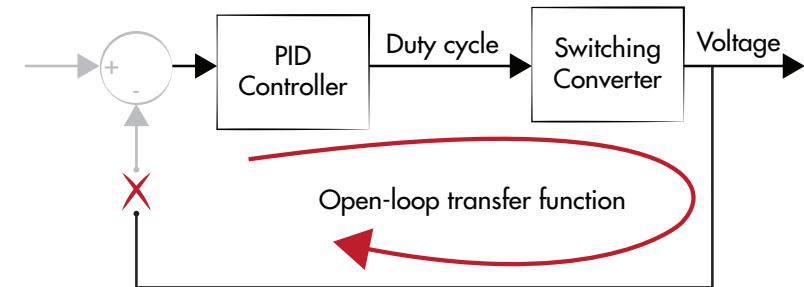
The Bode plot of the open-loop transfer function shows two responses:

- **Block response** – Response using the current gain values in the Discrete PID Controller block
- **Tuned response** – Response using new gain values set by the PID Tuner app

Before tuning, it can be helpful to also display additional response characteristics. In the Bode plot, right-click and select **Characteristics > All Stability Margins**. You can click the dot inside of the phase plot to show the phase margin and frequency values.

Try

Follow the steps on the page to launch the PID Tuner app. Configure the tuner to use the estimated LTI representation of the boost converter. Then, generate a Bode plot of the open-loop transfer function.



Digital Control Design

Tuning the Controller with PID Tuner (Continued)

Now, you can use the PID Tuner app to tune the voltage-mode controller gains. The **Domain** option in the **PID Tuner** tab lets you tune in the domain that you are most comfortable with:

- **Time** – Tune the response time and transient behavior of the response
- **Frequency** – Tune the bandwidth and phase margin of the response

In this chapter, you will tune the voltage-mode controller in the frequency domain, so set the **Domain** parameter to **Frequency**. Notice that the sliders in the toolbar now enable you to set the desired *bandwidth* and *phase margin* of the system. You can enter specific values for these parameters using the edit boxes next to the sliders.

Use the general guidelines below to begin tuning your controller:

- **Bandwidth** – The bandwidth must be less than the Nyquist frequency, which is half of the switching frequency. Try starting with a bandwidth ω_c that is about 10 times smaller than the Nyquist frequency:

$$\omega_{bw} \leq \frac{\pi}{10 \cdot T_{sw}}$$

where T_{sw} is switching period.

- **Phase margin** – Choose a phase margin between 45 and 60 degrees for generally acceptable performance.

Then, you can adjust the **Bandwidth** and **Phase Margin** sliders until the Bode plot displays an acceptable response that meets your design requirements. For additional information while tuning, click **Show Parameters**.

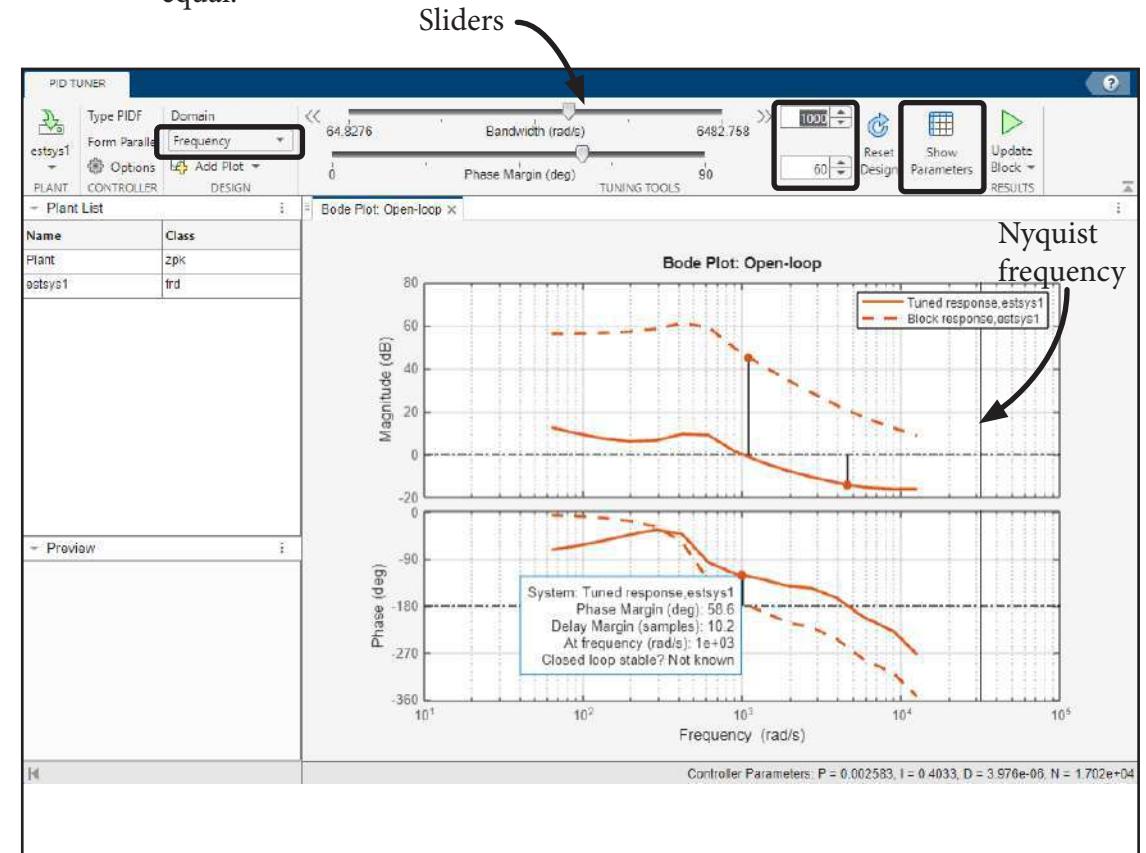
Try

Set the PID Tuner app to tune in the frequency domain. Try moving the **Bandwidth** and **Phase Margin** sliders to tune the controller. View the response of the open-loop transfer function in the Bode plot.

Then, manually enter the following parameters:

- **Bandwidth** is 1000 rad/s.
- **Phase margin** is 60 deg.

Note In the course example, the controller sampling frequency is equal to the switching frequency. In some systems, these frequencies may not be equal.



Digital Control Design

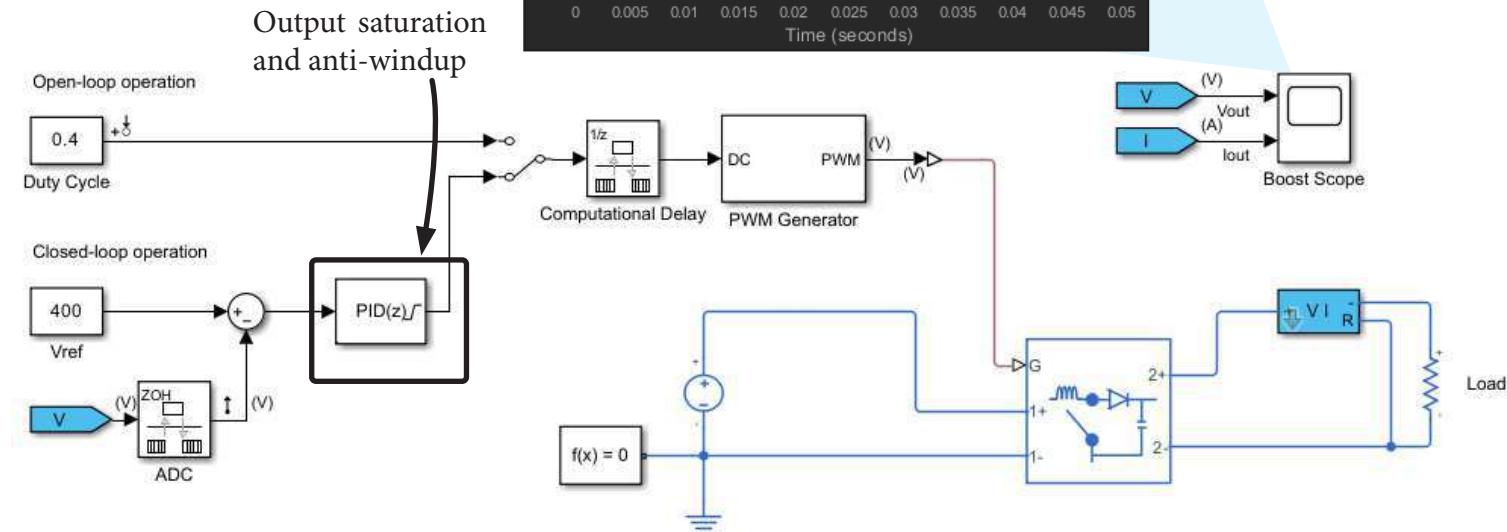
Tuning the Controller with PID Tuner (Continued)

When you are finished tuning, click **Update Block** to apply the calculated gains to the Discrete PID Controller block.

After tuning the voltage-mode controller, you should enable output saturation to limit the range of values of the duty cycle. To do this,

1. Double-click the Discrete PID Controller block to open its dialog parameters.
2. In the **Saturation** tab, make the following changes:
 - Enable the **Limit output** option.
 - Set the **Upper limit** to **0 .85**.
 - Set the **Lower limit** to **0 .15**.
 - Set the **Anti-windup Method** to **clamping**.
3. In the **Initialization** tab, set the **Integrator** and **Filter** initial conditions to **0 .15**.
4. Click **OK**. The icon on the Discrete PID Controller block will change to indicate that output saturation is enabled.

Before simulating the model, make sure that the model does not start from the steady-state operating point. Open the Configuration Parameters dialog (click **Model Settings** in the **Modeling** tab) and ensure that the **Input** and **Initial state** options in the **Data Import/Export** pane are disabled.



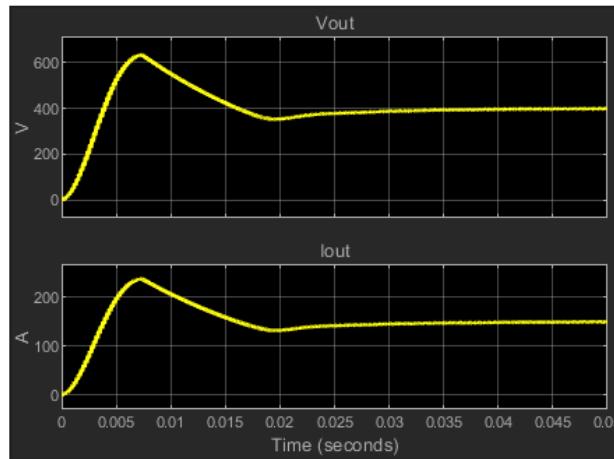
Try

Apply the calculated gains to the Discrete PID Controller block and enable output saturation, anti-windup, and initialization as described on this page.

Ensure that the model is not configured to begin simulation from the steady-state operating point found earlier in this chapter.

Simulate the model and inspect the output voltage in the scope. How does the controller perform?

```
>> boostConv4_control
```



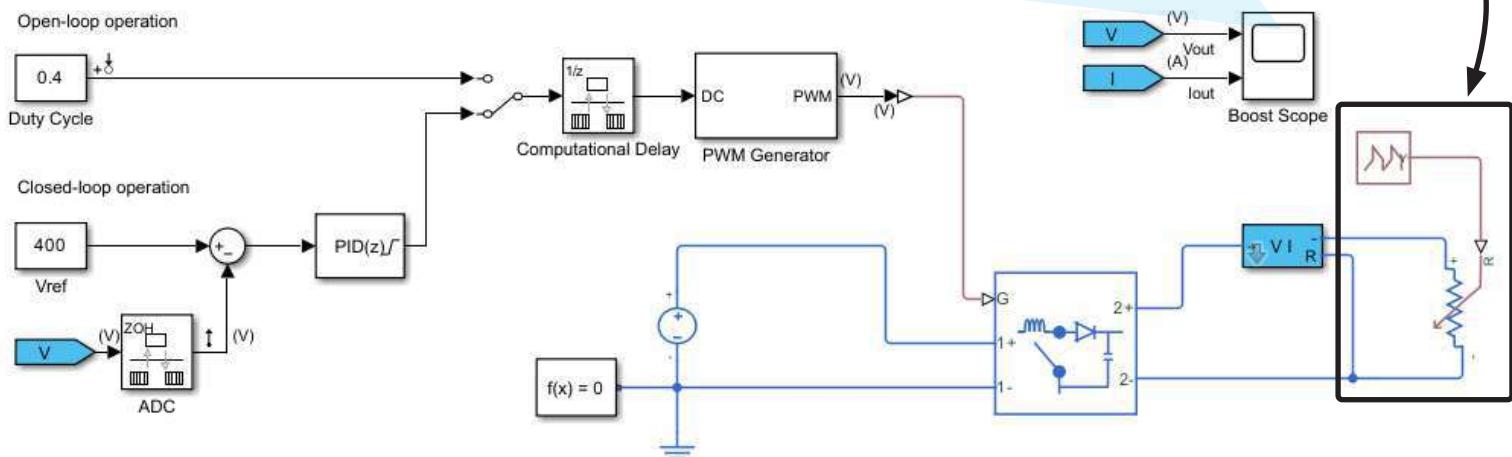
Digital Control Design

Verifying the Controller

After tuning the controller gains, it is a good practice to test the dynamic behavior to ensure that it meets your design specifications. It is especially important to verify the converter behavior before integrating it into a larger system model, such as the HEV motor drive system model that you will build in the final chapter.

First, you can verify that the controller appropriately adapts to disturbances in the load. This will ensure that the controller is able to perform under changing load conditions in the full motor drive model. To do this, implement a variable resistor in the model:

1. Replace the Load resistor with a Variable Resistor block (**Simscape > Foundation Library > Electrical > Electrical Elements library**).
2. Add a PS Repeating Sequence block (**Simscape > Foundation Library > Physical Signals > Sources library**). Connect it to the physical signal input of the Variable Resistor block. Then, set the following parameters
 - Initial output is 2.67.
 - Time offset is 0 s.
 - Signal type is Discrete.
 - Durations is [0.1, 0.1].
 - Start output values is [2.67, 5.34].
3. In the model, set the simulation **Stop time** to 0.3.



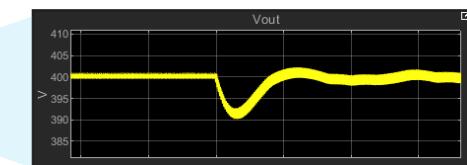
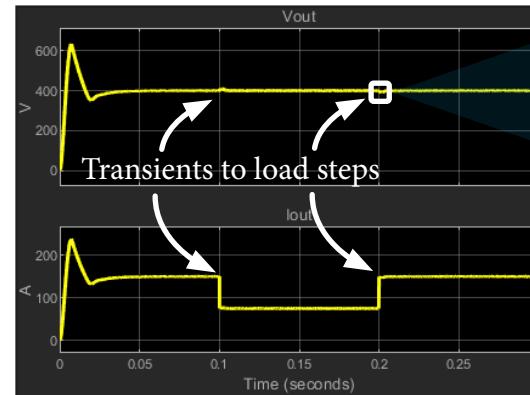
Try

Follow the steps on this page to implement a variable load.

Simulate the model and view the response on the scope. What happens to the voltage and current when the load resistance changes? Does the controller react well to disturbances in the load?

When finished, save the model.

>> boostConv5_control



Digital Control Design

Verifying the Controller (Continued)

In the final chapter, you will integrate this boost converter model into the HEV motor drive system model. The system model, however, will use an averaged-switch boost converter model because it simulates faster and the individual switching events do not need to be resolved.

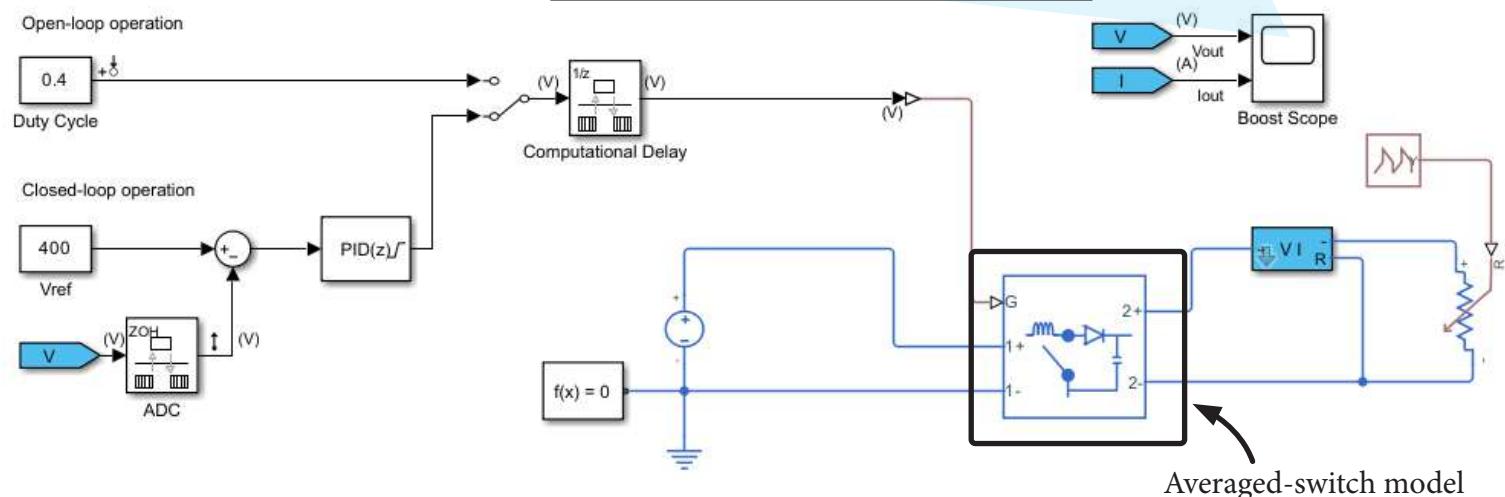
In preparation for the integration later, you should verify that the controller performs well with the averaged-switch model. To do this,

1. Delete the PWM Generator block. Connect the output of the Rate Transition block to the Simulink-PS Converter block.
2. In the Boost Converter block's dialog parameters, set the **Switching device** to **Averaged Switch**.
3. Run the simulation again.

For a more rigorous comparison of the piecewise linear and average switch models, you can use Simscape logging and compare simulation runs using Simulation Data Inspector.

Note If the simulation produces warnings about difficulty simulating the model, try increasing the **Number of consecutive min steps** parameter in the **Solver** pane of the model's **Configuration Parameters** dialog.

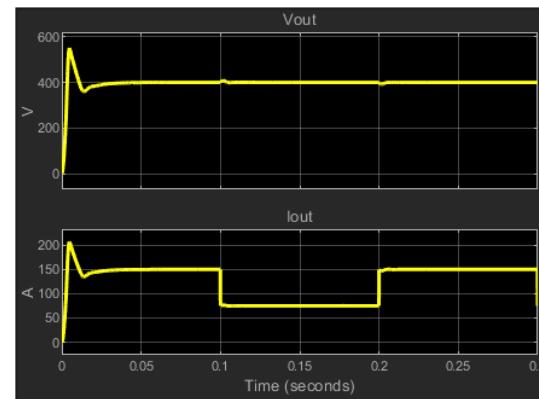
Note The Rate Transition block acting as unit delay ($1/z$) prevents the solver from finding an algebraic loop. For more information about refer to the training course **Modeling Physical Systems with Simscape**.



Try

Change the model to use an averaged-switch boost converter. Then, simulate the model and inspect the results. Does the controller perform well with the averaged switch model?

When finished, save the model as **boostConv6_control_avgSwitch**
>> boostConv6_control_avgSwitch



Summary

- Implementing closed-loop voltage control
- Linearizing power electronic converters
- Tuning the controller
- Testing the closed-loop performance

Test Your Knowledge

1. (Select all that apply) Which of the following are valid approaches to tune a controller with a nonlinear plant model?
 - A. Use PID Tuner directly with the nonlinear plant.
 - B. Use PID Tuner with a linearized version of the plant.
 - C. Use the Response Optimizer app to automatically tune the controller.
 - D. It is not possible to tune a controller with a nonlinear plant.
2. (T/F) When the PID Tuner is launched from the Discrete PID Controller block, it first attempts to analytically linearize the plant model.
3. (Select all that apply) When performing frequency response estimation, it is a good practice to
 - A. Inject and measure while the system-under-test is in steady state.
 - B. Use a perturbation signal magnitude of 1-5% of the input value.
 - C. Choose a maximum frequency equal to the switching frequency.
 - D. Use a sinestream perturbation signal with at least 1000 frequencies.
4. (Select all that apply) After tuning a PID Controller, you can verify the dynamic performance of a feedback-controlled power electronic converter by implementing the following tests:
 - A. Check the dynamic response to a time-varying load.
 - B. Check the dynamic response to a time-varying input voltage.
 - C. Check the dynamic response to aging components.

Answers

1. B, C
2. T
3. A, B
4. A, B, C



Power Electronics Control Design with Simulink® and
Simscape™

Modeling DC/AC Three-Phase Inverters

Outline

- Three-phase inverter operation principles
- Modeling an open-loop three-phase inverter in the abc reference frame
- Measuring three-phase physical quantities
- Characterizing harmonics and distortion
- Controlling inverter model fidelity

Chapter Learning Outcomes

The attendee will be able to

- Build and simulate three-phase alternating current (AC) power electronics systems.
- Sense three-phase physical quantities.
- Perform harmonic analysis to characterize distortion.

Modeling DC/AC Three-Phase Inverters

Course Example: Inverter

In the earlier chapters, you learned about modeling, fidelity, and control of DC power electronic systems. Many applications may also have three-phase AC electrical power systems that need to be modeled. In this chapter, you will learn how to use Simscape Electrical™ to build three-phase AC electrical models, measure three-phase voltage and current, and analyze the harmonic content of the three-phase voltage.

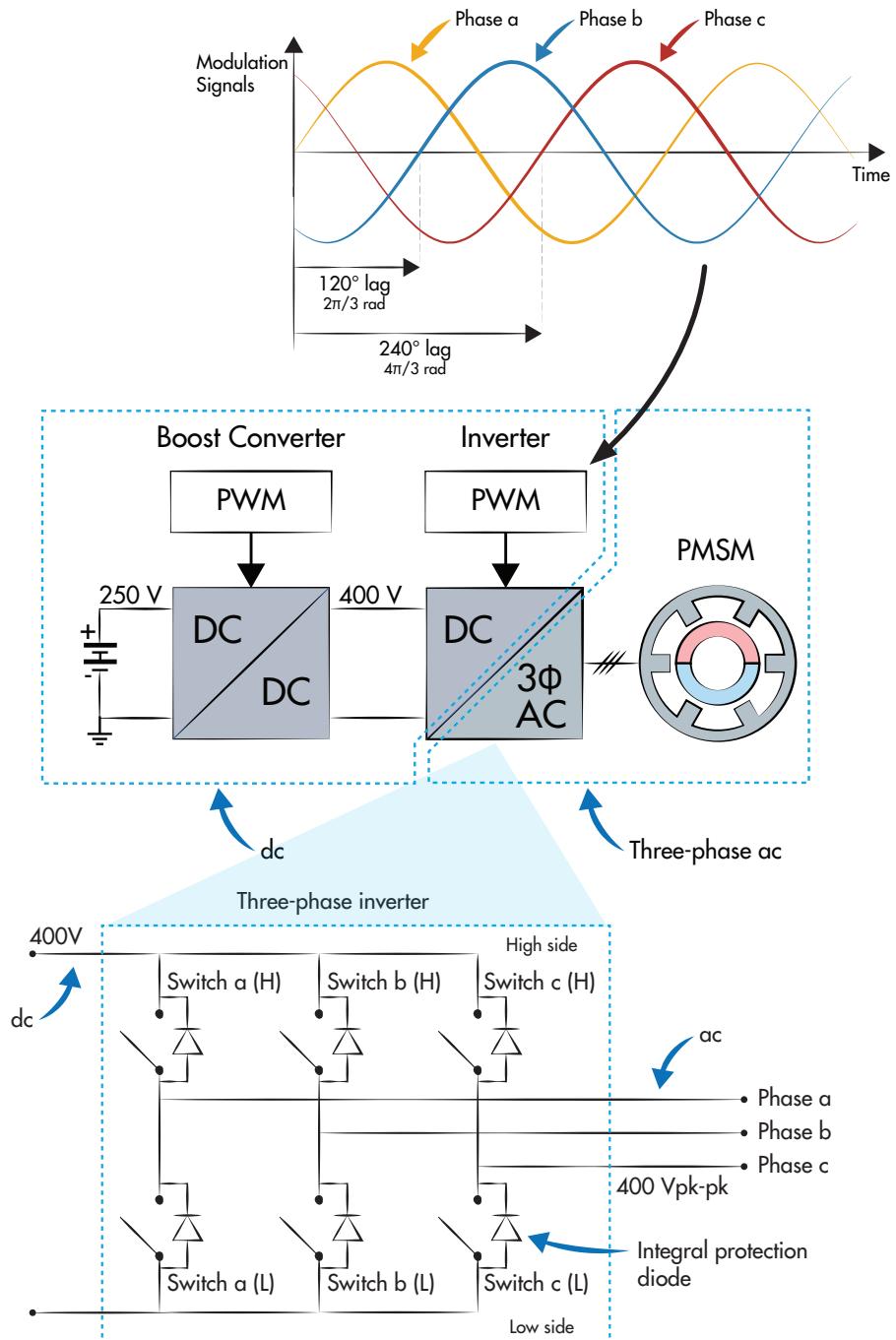
In the hybrid electric vehicle (HEV) motor drive course example model, an inverter is used to convert the 400 V_{dc} output of the boost converter into three-phase AC to drive the motor. The circuit diagram of the three-phase inverter is shown on the right and its main function is to change the DC input to the three-phase AC output.

This power electronic topology features six semiconductor switching devices with anti-parallel diodes (named integral protection diodes in the diagram). Each leg of the inverter has two switching devices, a High- and Low-side (referred as H and L in the diagram), and is connected to one of the phases at the output (referred as *Phase a*, *Phase b*, and *Phase c*).

The switching behavior of the six switches is controlled using three-phase pulse width modulation (PWM). Differently from DC power electronic converters, the modulation consists of a set of three different AC signals at the necessary frequency with the following characteristics:

- **Phase a** – has a 0° phase offset.
- **Phase b** – lags *phase a* by 120° ($2\pi/3$ radians).
- **Phase c** – lags *phase a* by 240° ($4\pi/3$ radians).

In this chapter, you will simulate the inverter with a passive load in an open-loop configuration to verify it produces AC voltages and currents at the expected frequencies.



Modeling DC/AC Three-Phase Inverters

Modeling the Three-Phase Load

You will use Simscape Electrical blocks to model a three-phase inverter with a passive load and simulate it in an open-loop configuration. Recall that when modeling electrical systems, you should model components and verify their behavior independently to reduce complexity when troubleshooting.

To isolate the three-phase inverter, the boost converter's DC output is represented with a DC voltage source. The motor's armature windings are represented by an equivalent phase resistance and inductance. This three-phase load is assumed to be *balanced* — that is, the load on each phase is equal. In Chapter 8, you will replace this load with a motor model to incorporate electro-mechanical dynamics in your model.

Start by opening the `inverter1_start` model. This model already has the following changes made:

- Added Solver Configuration block.
- Added DC Voltage Source block at 250 volts, representing the boost converter's output voltage.
- Set solver and solver tolerances as discussed in Chapter 2.

To begin modeling the three-phase system,

1. Add a Wye-Connected Load block (**Electrical > Passive > RLC Assemblies** library) to represent the PMSM motor's armature windings. Set the following parameter values:
 - **Parameterization** is **Specify component values directly**.
 - **Component structure** is **Series RL**.
 - **Resistance** is **5 mOhm**.
 - **Inductance** is **0.75 mH**.

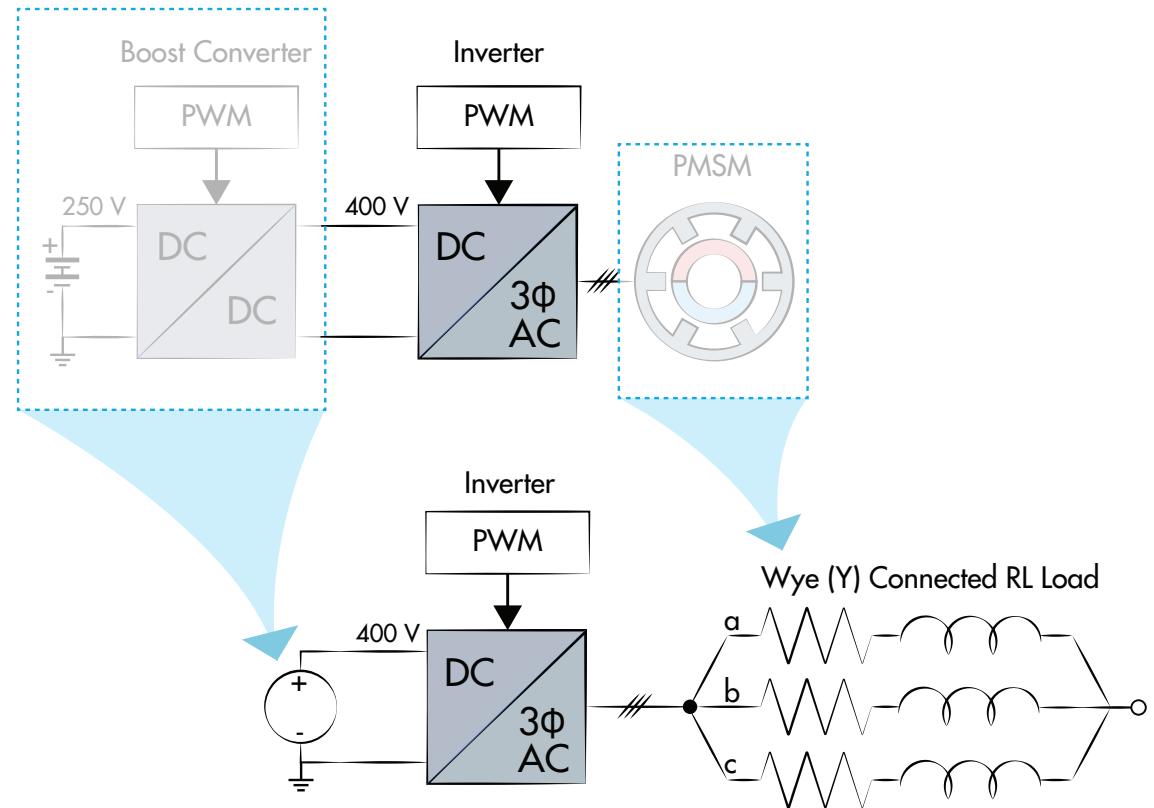
The \sim port of the Wye-Connected Load block is a three-phase electrical port that will be connected to the output of the three-phase inverter later.

Try

Open the `inverter1_start` model. Follow the steps on this page to add a three-phase resistive inductive load to represent the motor.

2. Add an Open Circuit block (**Electrical > Connectors & References** library). Connect it to the n port of the Wye-Connected Load block. This specifies the neutral point to be floating.

Note The **modeling option** parameter allows you to model three-phase connections using two types of ports: **Composite three-phase ports** and **Expanded three-phase ports**.



Modeling DC/AC Three-Phase Inverters

Modeling the Three-Phase Inverter

The three-phase inverter has six semiconductor switching devices that are turned on and off in a predetermined sequence. By manipulating the switching sequence and timing, you can control the frequency and magnitude of the three-phase output voltage.

The inverter modeled in this chapter uses insulated gate bipolar transistors (IGBT) for the semiconductor switching devices. This is a common choice for the rated voltage (400 V) and switching frequency (10 kHz).

In this chapter, you will model the three-phase inverter with a prebuilt block in Simscape Electrical that allows you to select piecewise-linear switching or averaged switch models. However, recall that you could also build the inverter model from discrete components for additional customization and higher fidelity (see Exercise A-20).

To model the three-phase inverter,

1. Add a Converter (Three-Phase) block (**Simscape > Electrical > Semiconductors & Converters > Converters** library). Then, set the following block parameters:

- **Switching device** is IGBT.
- **Threshold Voltage, V_{th}** is 0.5 V.
- **Integral protection diode** is Diode with no dynamics.

The integral protection diode protects the IGBT against reverse voltage by allowing a conducting path from the load back to the DC source when the AC phase voltage goes negative.

2. Select the Converter (Three-Phase) block and click the button in the **Format** tab to flip the block (left to right). This will orient the +/- DC ports to be on the left and the ~ three-phase port to be on the right.
3. Connect the + and - ports of the Converter (Three-Phase) block to the DC voltage supply.
4. Connect the ~ port of the Converter (Three-Phase) block to the Wye-Connected Load block.

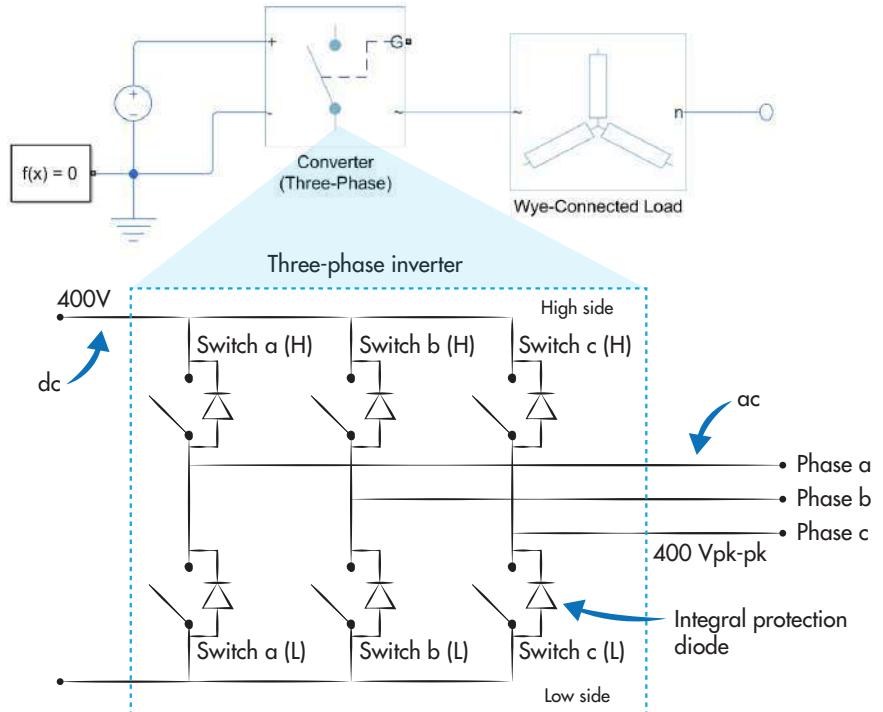
Try

Follow the steps listed on this page to add a three-phase inverter to your model.

>> inverter2

On the next page, you will use a PWM generator to generate signals to connect to the G port of the Converter (Three-Phase) block and control the output of the inverter.

Note As mentioned in the previous page, the **modeling option** parameter allows you to model three-phase ports as either composite or expanded. Notice the different block colors: light blue for composite three-phase ports and dark blue for expanded three-phase ports. You can use the Phase Splitter block (**Electrical > Connectors & References** library) to connect composite and expanded three-phase ports.



Modeling DC/AC Three-Phase Inverters

Generating PWM Signals

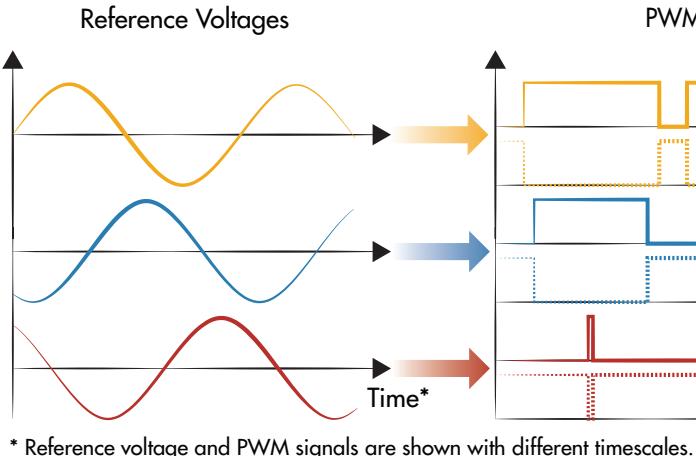
For the three-phase inverter to produce AC output, it needs *gate voltage signals* to command each of the six switches to open or close. Two switches are used to control the AC output for each phase, as shown in the schematic below. The two switches are arranged such that they cannot both be on at the same time, or else they will short-circuit the terminals of the DC supply.

A common approach for generating the gate voltage signals is to use pulse width modulation (PWM). In this chapter, you will use the PWM Generator (Three-Phase, Two-Level) block to generate the gate signals. This block takes three reference voltages as input and converts them into three pairs of complementary PWM waveforms. Each pair of waveforms drive the two switches for a particular phase.

For now, the reference voltage signals are represented with sine waves. This operates the inverter in open-loop configuration. In the next chapter, you will replace the sine waves with a closed-loop feedback configuration.

To generate the PWM signals in your model,

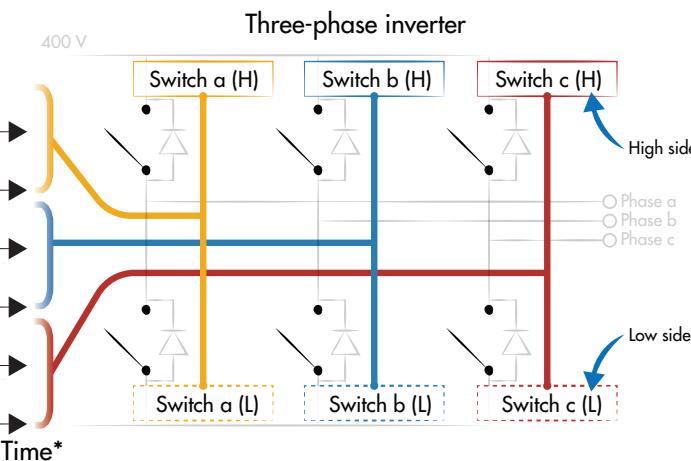
1. Add a PWM Generator (Three-phase, Two-level) block (**Simscape > Electrical > Control > Pulse Width Modulation** library) and set the following parameter values:



Try

Follow the steps listed on this page to add a PWM generator to your model.

- **Switching frequency (Hz)** is **1e4**.
 - **Sample time (s)** is **5e - 6**. Note that this sampling frequency is 20 times the switching frequency.
2. Add a Sine Wave block (**Simulink > Sources** library) to generate time-varying three-phase reference signals. Name it **Vabc**. Connect it to the **Vabc** input of the PWM generator. Set the following parameter values:
 - **Amplitude** to **200**.
 - **Frequency (rad/sec)** is **2*pi*60**. This is the fundamental frequency.
 - **Phase (rad)** is **[0, -2*pi/3, -4*pi/3]**.
 - **Sample time** is **1e - 4**.
 3. Add a Constant block (**Simulink > Sources** library) representing the DC voltage at the inverter input. Name it **Supply Voltage**. Connect it to the **vdc** input of the PWM generator. Set the **Constant value** to **400** and the **Sample time** to **1e - 4**.



Modeling DC/AC Three-Phase Inverters

Generating PWM Signals (Continued)

The PWM Generator (Three-Phase, Two-Level) block has two Simulink® signal outputs, as described below:

- **g** – A vector containing six gate voltage signals, one for each switch. The signals are valued 0 or 1 and are arranged in the following order: Switch a (H), Switch a (L), Switch b (H), Switch b (L), Switch c (H), Switch c (L). This output can be used to command the switches of an inverter using piecewise linear or nonlinear switching models.
- **ModWave** – A vector containing three modulation signals ranging from -1 to 1 based on the averaged gate voltage signals for each phase:

$$MW_x = 2 \bar{g}_{x(H)} - 1 = -2 \bar{g}_{x(L)} + 1$$

where the bar denotes the time average over the switching period and x denotes phase a, b, or c. The signals are arranged in the following order: phase a, phase b, phase c. As you will see later, the ModWave output can be used to command an inverter using an averaged switch model.

In this chapter, the inverter uses a piecewise-linear switching model so you can use the g output. To connect the PWM Generator (Three-phase, Two-level) block to the Simscape™ network, you will need to

1. Add a Demux block (**Simulink > Signal Routing** library) to split the vector signal output from g into six individual Simulink signals.
2. Convert each Simulink signal into a Simscape physical signal using **Simulink-PS Converter** blocks.
3. Connect the physical signals to a Six-Pulse Gate Multiplexer block (**Electrical > Semiconductors & Converters > Converters** library).
4. Connect the multiplexer's G port to the converter block's G port.

Alternatively, you can use the Gate Driver block provided in the **trainingLib** library, which conveniently incorporates the Simulink-PS Converter blocks and the multiplexer block, as shown on this page.

Try

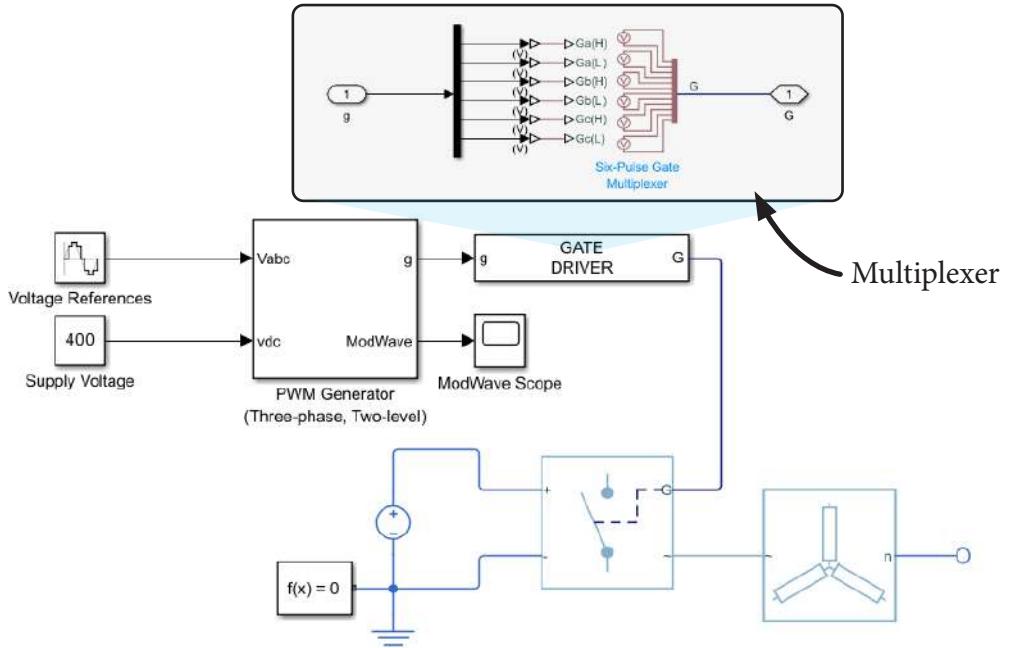
Add a Gate Driver block from the **trainingLib** library to connect the gate voltage signals from the PWM generator to the G port of the converter.

Add a Scope block to visualize the ModWave output from the PWM generator.

>> inverter3

Connect the ModWave output of the PWM generator to a Scope block and name it ModWave Scope. This helps you see when the modulation scheme is saturating (that is when ModWave has values of -1 or 1).

Note For the inverter switches to close, the gate voltage signals must be greater than the **Voltage threshold**, **Vth** parameter in the Converter (Three-Phase) block. In this case, the gate voltage signals have a maximum value of 1 volt and the threshold is 0.5 volts. In other cases, you may need to add a gain to appropriately scale the gate voltage signals.



Modeling DC/AC Three-Phase Inverters

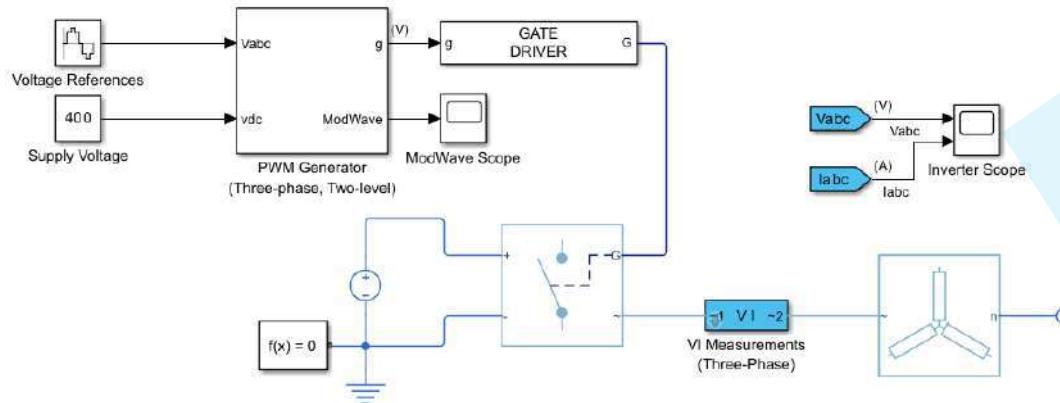
Measuring Three-Phase Quantities

The **Electrical > Sensors & Transducers** library includes a variety of three-phase electrical sensors. You can explore this library and look for sensor blocks that have “Three-Phase” in their names.

To visualize the three-phase voltage and current on the AC side of the inverter, you can use the VI Measurements (Three-Phase) block from the **trainingLib** library. This block contains a Current and Voltage Sensor (Three-Phase) block. The output measurements are transmitted using Goto blocks and can be retrieved using a From block with the tags **Vabc** and **Iabc**, as shown below.

Alternatively, you can add the sensor manually:

1. Delete the physical connection between the converter and the load.
2. Add a Current and Voltage Sensor (Three-Phase) block (**Simscape > Electrical > Sensors & Transducers** library). Set the **Voltage measurement type** to **phase-to-ground voltage**. Then, connect its ~ 1 port to the \sim port of the converter. Connect its ~ 2 port to the \sim port of the load.



Try

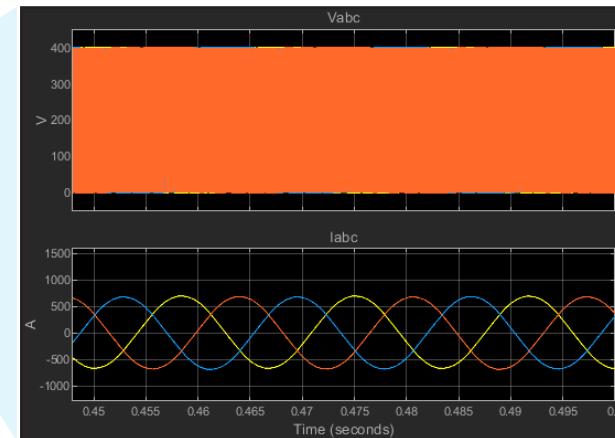
Open the **Electrical > Sensors & Transducers** library and view the different three-phase sensors available.

Add three-phase voltage and current measurements to your model. Simulate the model and inspect the three-phase inverter output.

```
>> inverter4
>> inverter4_noLib
```

3. Add two PS-Simulink Converter blocks (**Simscape > Utilities** library). Connect their inputs to the **V** and **I** physical signal outputs of the sensor. Set their **Output signal unit** parameters to **V** and **I**, respectively
4. Connect the outputs of the PS-Simulink Converter blocks to a scope for visualization. Label the signals **Vabc** and **Iabc**, respectively. Configure the scope to have two displays to show voltage and current separately. Show the time units. Name the scope Inverter Scope.

Note You are measuring the phase-to-ground voltage. Notice that the ground terminal is connected to the negative DC input port of the Converter (Three-Phase) block. You can also measure the voltage between phases by selecting the **Voltage measurement type** to **phase-to-phase voltage** in the Current and Voltage Sensor (Three-Phase) block.

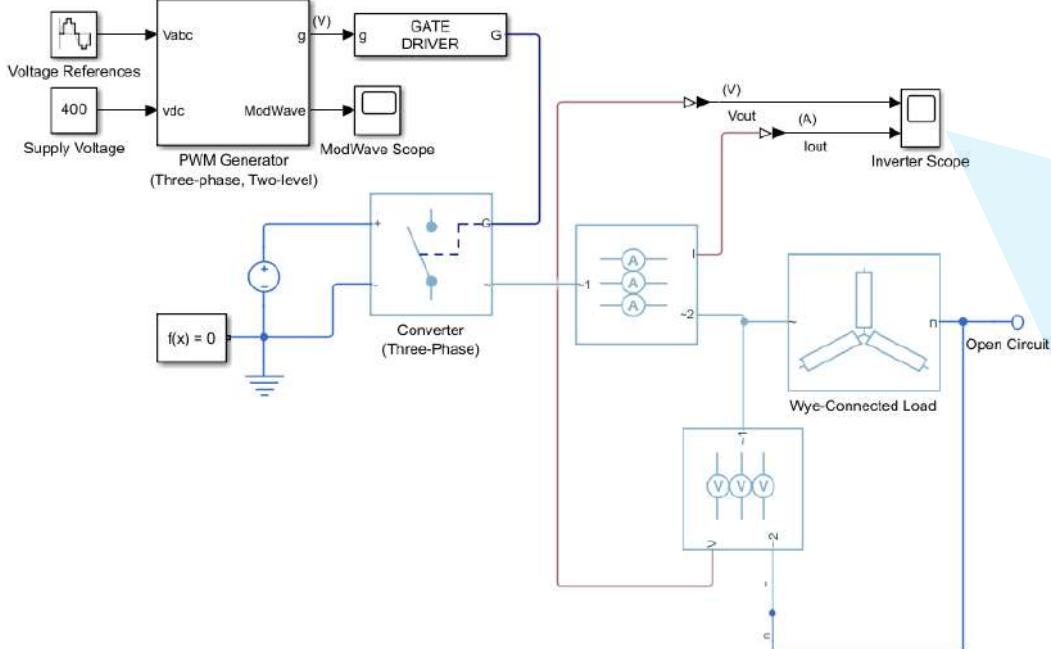


Modeling DC/AC Three-Phase Inverters

Measuring Three-Phase Quantities (Continued)

Measuring the phase-to-neutral voltage when the neutral port of the load is floating requires a voltage sensor across the load with special care of the composite three-phase port \sim and the regular electrical port n. Follow the steps:

1. Open the `inverter4_noLib` model.
2. Delete the Current and Voltage Sensor (Three-Phase) block.
3. Add a Current Sensor (Three-Phase) block (**Simscape > Electrical > Sensors & Transducers** library) in series between the Converter (Three-Phase) block and the Wye-Connected Load block.
4. Connect the physical signal output I to the input of the PS-Simulink Converter block for the current measurement.



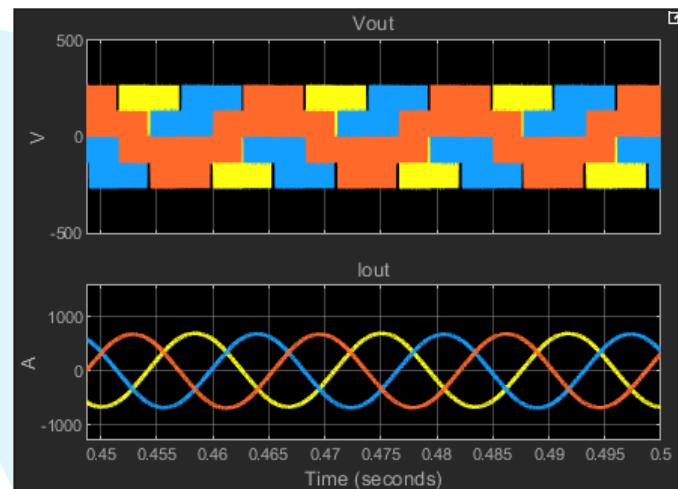
Try

Follow the steps on this page to measure the phase-to-neutral voltage when the neutral port of the load is floating. Simulate the model and inspect the three-phase inverter output.

```
>> inverter4_noLib
>> inverter4_Phase_to_Neutral
```

5. Add a Phase Voltage Sensor (Three-Phase) block (**Simscape > Electrical > Sensors & Transducers** library). Then, connect its ~ 1 port to the \sim port of the Wye-Connected Load block.
6. Add a Neutral Port (Three-Phase) block and connect its \sim -port to the ~ 2 port of the Voltage Sensor (Three-Phase) block. Connect the n port to the n port of the Wye-Connected Load block.
7. Connect the physical signal output V to the input of the PS-Simulink Converter block for the voltage measurement.

Note In this model, the inductor current is very sensitive to its voltage. To reduce the risk of seeing voltage spikes in the results, make sure that you set both the Why-Connected Load's **Parasitic parallel conductance** and the Neutral Port (Three-Phase)'s **Parasitic ground conductance** parameters to zero.



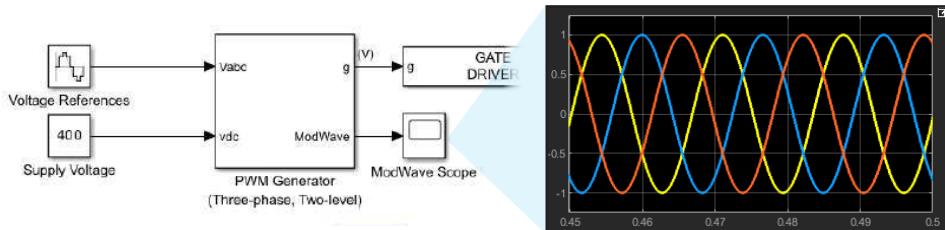
Modeling DC/AC Three-Phase Inverters

Verifying the Inverter in the Time Domain

To verify that the PWM generator and inverter are performing as expected, you can check that for any particular phase (a, b, or c):

- The modulation waveform's value is within the range of -1 to 1, but is not saturated for extended periods of time.
- The peak-to-peak phase-to-ground output voltage is equal to the supplied DC voltage (400 V) and it switches at the expected switching frequency (10 kHz).
- The current waveforms are sinusoidal and have a fundamental frequency matching the voltage references (60 Hz).

To check the modulation waveform, inspect the ModWave Scope after simulating the model. Values of -1 or 1 indicate that the full DC supply is being utilized. The waveform should remain within the range of -1 to 1 without saturating for the inverter to perform within its full power rating.



To check the inverter output voltage and current, open the Inverter Scope and follow the steps below:

- Open the scope window.
- Use the zoom tool () to zoom in to a region of interest.
- Click the Cursor Measurements () button.
- Click to drag the two cursors to encompass one period in a voltage or current signal.
- Examine the $1/\Delta T$ value specified in the Measurements panel.

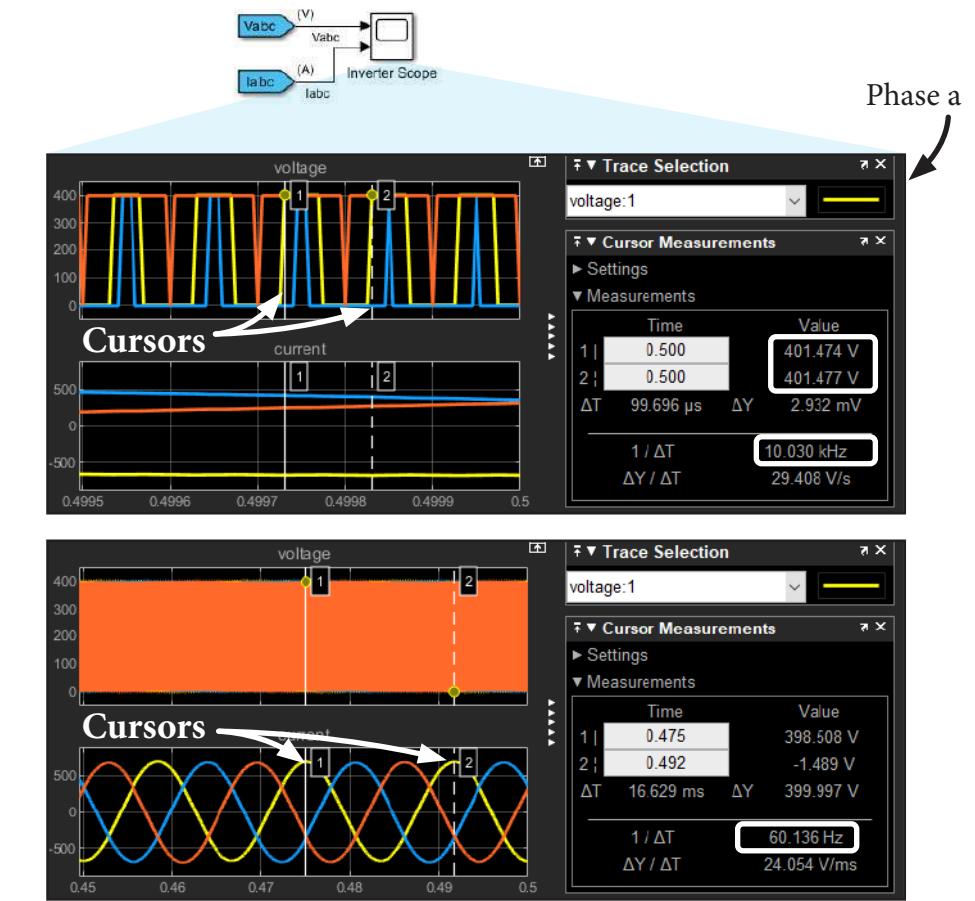
Try

>> inverter4

Open the ModWave Scope to verify that the modulation waveforms cover the full range from -1 to 1 but are not saturated for extended time.

Then, open the Inverter Scope and use cursor measurements to verify that

- Voltage switches between 0 and 400 volts with a frequency of 10 kHz
- Current is sinusoidal with a frequency of 60 Hz



Modeling DC/AC Three-Phase Inverters

Verifying the Inverter Using Harmonic Analysis

You can perform *harmonic analysis* on the AC output of the inverter to characterize its performance. The term *harmonic* refers to integer multiples of the fundamental (60 Hz) and switching (10 kHz) frequencies. It is important to be aware of the harmonics in your system because, beyond certain limits, they can jeopardize the operation of sensitive components.

You can visualize harmonic content using the Spectrum Analyzer block:

1. Add a Selector block (**Simulink > Signal Routing library**). Connect the input to the Vabc From block. In the block's dialog parameters, set the **Index** to 1 to select the *phase a* voltage.
2. Add a Zero-Order Hold block (**Simulink > Discrete library**) and connect it to the Selector block. Set the **Sample time** to $5e-6$.
3. Add a Spectrum Analyzer block (**Simscape > Utilities library**) and connect it to the Zero-Order Hold block.
4. Double-click the Spectrum Analyzer block.

In the **Analyzer** tab,

- In the Views section, select **RMS** under the Spectrum menu. This shows the root mean squared spectrum of the block's input signal.
- In the Bandwidth section, set **RBW (Hz)** to 5. This is the resolution bandwidth, representing the smallest frequency bandwidth that can be resolved in the spectrum.

In the **Estimation** tab,

- In the Frequency Options section, select **Span** and **Center Frequency** for the **Frequency Span**.
- Set **Span (Hz)** to $40e3$. This is the maximum frequency shown in the spectrum.
- Set **Center Frequency (Hz)** to $20e3$. This is the center frequency of the spectrum.

Try

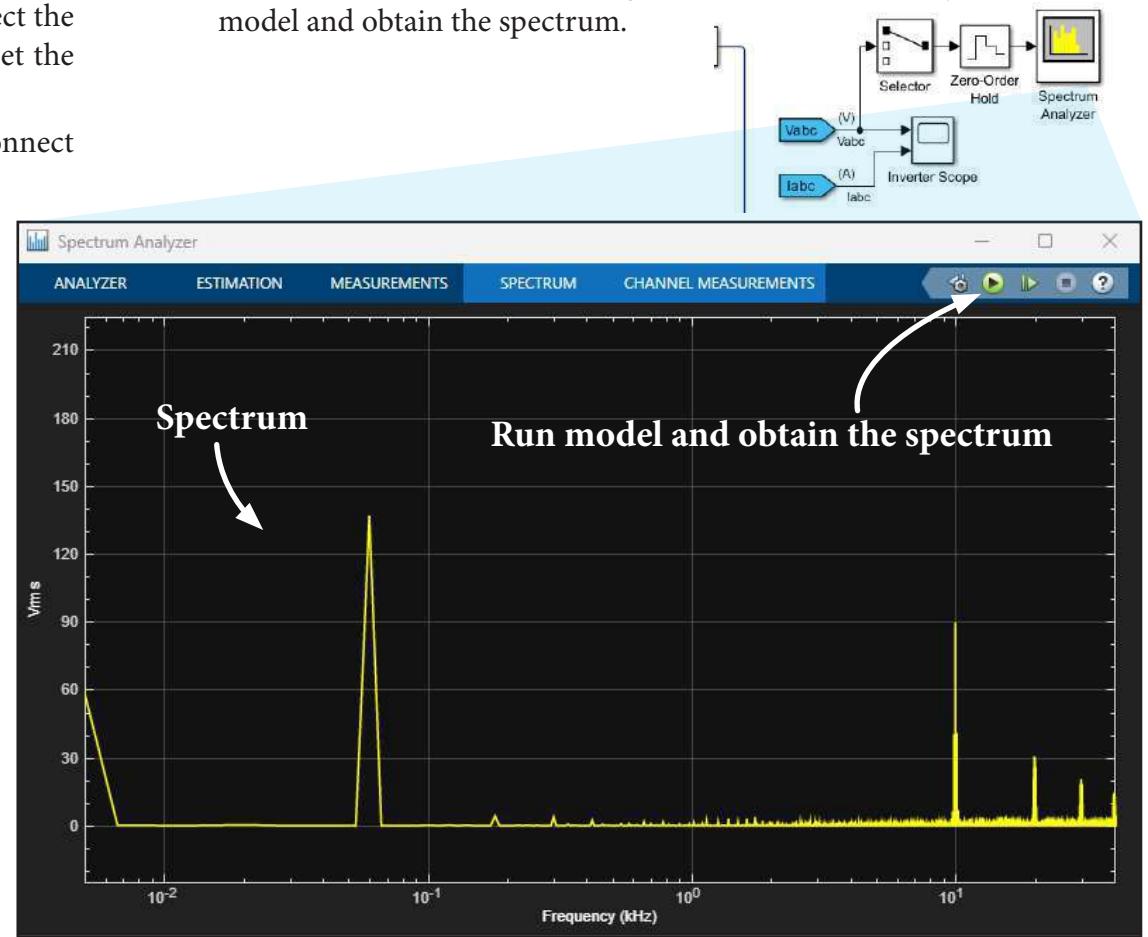
Follow the instructions on this page to add a Spectrum Analyzer block to the model to analyze the harmonics of *phase a* of the inverter voltage.

Then, simulate the model and inspect the results in the Spectrum Analyzer.

>> inverter5_spectrum

In the **Spectrum** tab,

- In the Scale section, set **Log** for the **Frequency Scale**.
- 5. Click the  button on the top-right of the Spectrum Analyzer to run the model and obtain the spectrum.



Modeling DC/AC Three-Phase Inverters

Verifying the Inverter Using Harmonic Analysis (Continued)

Additional analysis tools are available in the Measurements tab inside the Spectrum Analyzer to investigate the measured spectrum.

Peak Finder in the Peaks section

The peak finder tool automatically identifies peaks inside the spectrum. This is useful to identify the dominant frequencies in the spectrum. Click on the Peak Finder button to open the Peaks table under the plotted spectrum. Set **Num Peaks** to 6, and enable Label Peaks.

In the **Peaks table**, you can see a list showing the rms value and frequency of all peaks identified in the spectrum.

Distortion Measurements in the Distortion section

You can enable distortion measurements to characterize harmonics and intermodulation products in the spectrum. Click on the Distortion button to



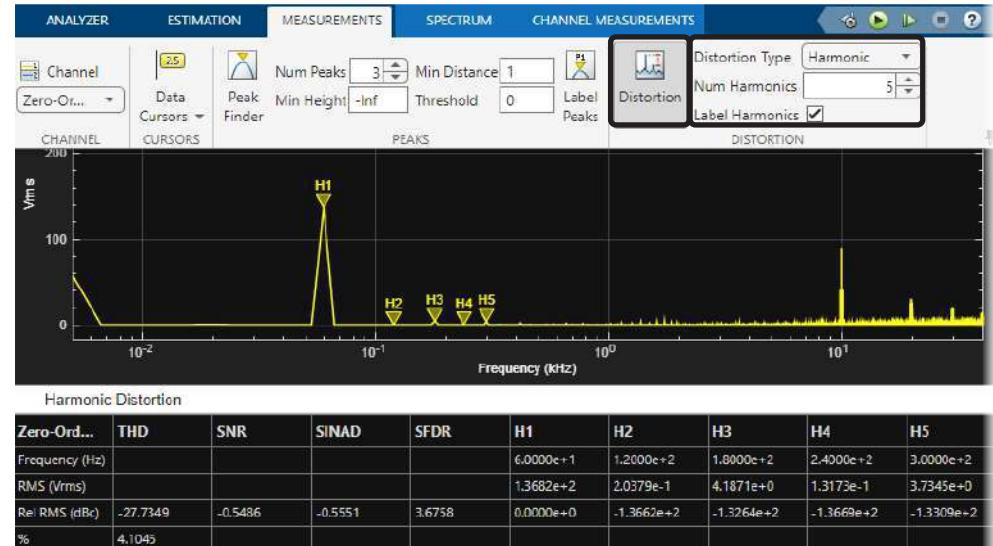
Try

Use the spectrum analyzer to do the following:

- Use the peak finder to identify the fundamental frequency (60 Hz) and the inverter switching frequency (10 kHz).
- Use the harmonic distortion measurements to identify the first five harmonics of the fundamental frequency.
- Use the intermodulation distortion measurements to identify the $2F_2 - F_1$ ($2 \times 10 \text{ kHz} - 60 \text{ Hz} = 19940 \text{ Hz}$) intermodulation product.

open the Harmonic Distortion table or Intermodulation table depending on your choice of the **Distortion Type** (Harmonic or Intermodulation, respectively).

To identify the first five harmonics, set **Distortion Type** to **Harmonic** and set **Num Harmonics** to 5. The frequency and amplitude of the first five harmonics are displayed in the Harmonic **Distortion** table. If the **Label Harmonics** option is enabled, markers will be also displayed to identify the harmonics in the spectrum.



Modeling DC/AC Three-Phase Inverters

Controlling Inverter Model Fidelity

In Chapter 4, you learned how to control model fidelity for DC/DC converters. The same model fidelity considerations can be made for DC/AC (and AC/DC) three-phase converters.

As you have already learned, prebuilt converters are available in the **Electrical > Semiconductors & Converters > Converters** library. Remember that with prebuilt converters you **cannot** model switching losses, show the thermal port, and use pre-parametrization workflows. Prebuilt converters allow you to implement the following fidelity levels.

Average-Value

The Average-Value Voltage Source Converter (Three-Phase) block converts electrical energy from AC to DC voltage or from DC to AC voltage according to an input three-phase modulation wave.

Note The Average-Value Inverter (Three-Phase) and Average-Value Rectifier (Three-Phase) blocks behave as a DC-voltage-controlled AC voltage source and an AC-voltage-controlled DC voltage source, respectively. They do not have a control input as the ratio you specify in the block dialog determines the ratio between the DC (AC) voltage and the AC (DC) voltage.

Averaged-Switch

The Converter (Three-Phase) block with the **Switching device** set to **Averaged switch** requires to be driven by six modulation waveforms, two for each converter leg, with maximum amplitude in the range between 0 and 1. This poses an extra modulation challenge compared to the DC/DC counterpart and it is shown on the following page.

Piecewise-Linear Switching using prebuilt converter blocks

The Converter (Three-Phase) block with the **Switching device** set to anything except for **Averaged switch** (for example IGBT) requires to be driven by six pulse waveforms, two for each converter leg, generated by a three-phase PWM, with amplitude that needs to be chosen depending on the **Voltage threshold, V_{th}** parameter you select in the block dialog.

Try

Read this summary page on inverter model fidelity and go back to Chapters 4 and 3 for more details.

You can model converters using discrete semiconductor devices available in the **Electrical > Semiconductors & Converters** library. Each switching device requires to be driven by a pulse waveform generated by a three-phase PWM. Discrete semiconductor devices can show the thermal port and allow you to model thermal effects. You can implement the following fidelity levels.

Piecewise-Linear Switching using discrete semiconductor device blocks

You can use the blocks labeled as “Ideal, Switching”, “Ideal”, and “Piecewise Linear”. Remember from Chapter 3 that the devices labeled as “Ideal, Switching” allow you to model the switching losses separately from the conduction losses and are supported with pre-parametrization workflows.

Nonlinear Switching

You can use the blocks labeled as “N-Channel”, “P-Channel”, “NPN”, and “PNP”. With these blocks, you can model the dynamic effects that are responsible for non-ideal switching transients of real-world semiconductor devices. To properly drive these devices, you must use a Gate Driver block. The Half-Bridge Driver block can be used for driving both the high- and low-side devices.

Bear in mind that inverter models using nonlinear switching model fidelity

- generally take quite long to simulate, and
- can report on losses, but switching losses are **not** calculated separately.

Modeling DC/AC Three-Phase Inverters

Modeling an Averaged Switch Inverter

In the next chapter, you will design the inverter control using a piecewise-linear switching model. Afterward, you will set the inverter model to use an averaged-switch model to reduce computation time.

The ModWave output of the PWM generator will be used to drive the averaged-switching inverter. The g output of the PWM generator is more computationally expensive and will not be needed. Therefore, you can replace that PWM block with a simpler version that can simulate faster.

To modify your piecewise-linear switching inverter model to use an averaged-switch model,

1. Delete the PWM Generator (Three-Phase, Two-Level) block.
2. Add a PWM Timing and Waveform Generator (Three-phase, Two-level) block (**Electrical > Control > Pulse Width Modulation** library). Connect the Vabc and vdc ports to the Voltage Reference and Supply Voltage blocks, respectively. Then, set the following parameters:
 - **Continuous PWM** is SPWM: sinusoidal PWM.
 - **Switching frequency (Hz)** is 1e4.
3. Replace the Gate Driver block with a ModWave Gate Driver block from the **trainingLib** library. This block converts the three modulation signals into six equivalent average gate voltage signals to drive the inverter. Connect it to the ModWave output of the PWM Timing and Waveform Generator (Three-phase, Two-level) block.
4. Double-click the Converter (Three-Phase) block and set the **Switching device** to **Averaged switch**.
5. Add two Terminator blocks and connect them to TgabcON and TgabcOFF outputs of the PWM Timing and Waveform Generator block.
6. Comment out the blocks for spectrum analysis as all

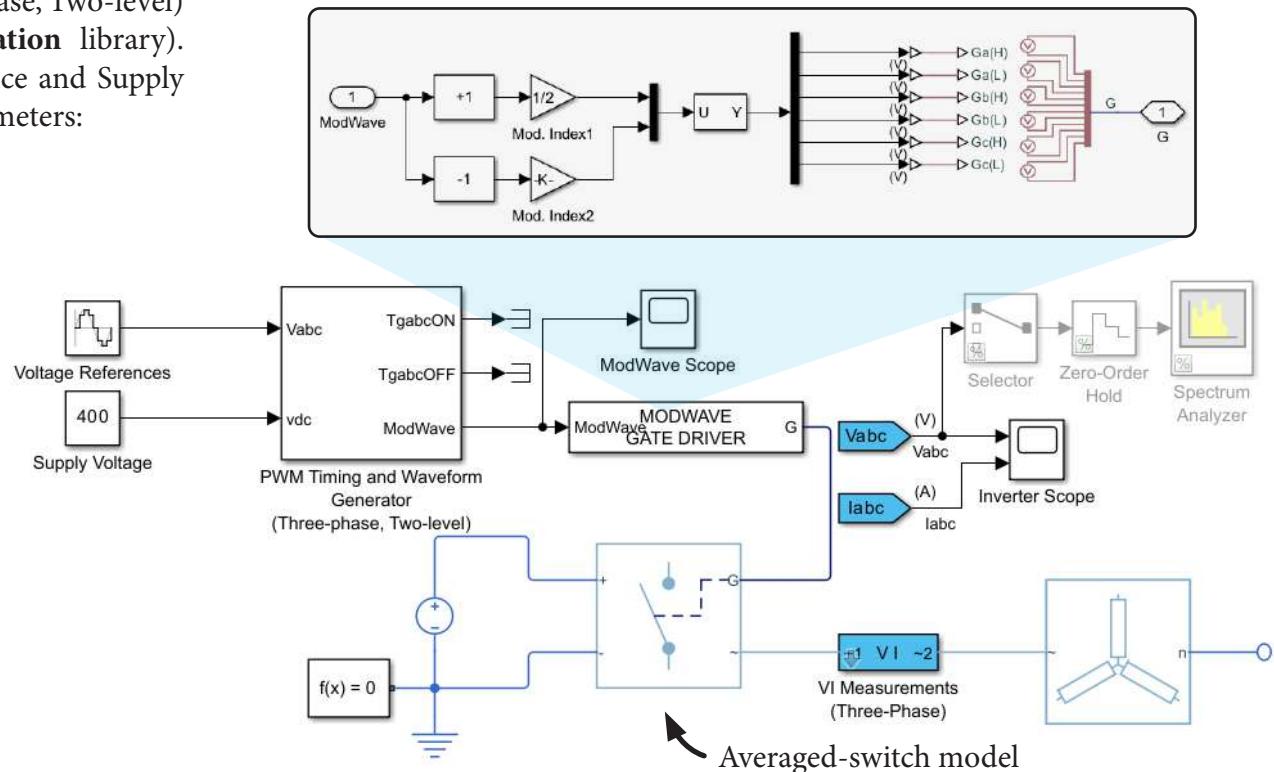
Try

Modify the inverter to use an averaged-switch model. Simulate the model and inspect the results. Note the output voltage now looks sinusoidal because the individual switching events are no longer captured.

>> inverter6_avgSwitch

the switching harmonics are eliminated by the selection of the averaged switch model.

Note If you did not replace the PWM Generator (Three-Phase, Two-Level) block and used its ModWave output to drive the ModWave Gate Driver block, you would “waste” computational time because the g output would be still calculated at the PWM sample time. You can select **Information Overlays > Colors** in the **Debug** tab to view the review sample times used in the model.



Summary

- Modeling a three-phase inverter
- Measuring three-phase physical quantities
- Characterizing harmonics and distortion
- Controlling inverter model fidelity

Test Your Knowledge

1. (T/F) It is possible to view and manipulate the individual phases of a three-phase signal.
2. (T/F) When measuring the currents of a three-phase signal, you must use three separate Current Sensor blocks.
3. (T/F) You can use the Spectrum Analyzer block to visualize the magnitude of the harmonics generated in a switching model.

Answers

1. True - You can view the expanded three-phase ports to gain access to individual phases. You can also use the Phase Splitter block (**Electrical > Connectors & References** library) to split a three-phase connection into its constituent phases.
2. F
3. T



Battery Modeling and Algorithm Development with
Simulink®

Introduction

Introduction

MathWorks® at a Glance

MathWorks® is the leading developer of mathematical computing software in the world.

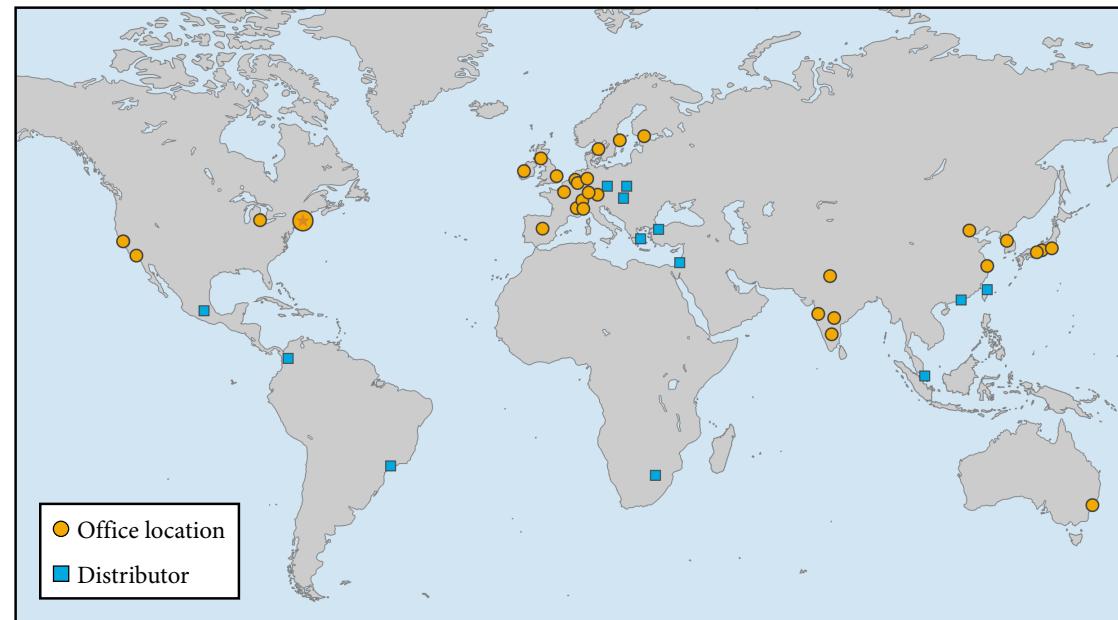
- Privately held
- Over 4000 employees worldwide
- More than 2 million users in 180+ countries

MathWorks products are relied upon by a variety of key industries to accelerate the pace of discovery, innovation, development, and learning, including

- Aerospace and defense
- Automotive
- Biological sciences
- Biotech and pharmaceutical
- Communications
- Electronics
- Energy production
- Financial services
- Industrial automation and machinery
- Medical devices
- Metals, materials, and mining
- Neuroscience
- Railway systems
- Semiconductors
- Software and internet

MathWorks supports programs that inspire learning and advance education in engineering, science, and math.

- 5000+ universities around the world
- 1800+ MATLAB® and Simulink® based books



- Academic support for research, fellowships, student competitions, and curriculum development

MathWorks supports its customers through a worldwide network of offices, distributors, and resellers.

Headquarters (Natick, MA USA)

www.mathworks.com
support@mathworks.com
+1-508-647-7000

Worldwide Offices

For information on any of our worldwide offices, please open the following location on the MathWorks site and choose a country:

www.mathworks.com/company/worldwide/

Introduction

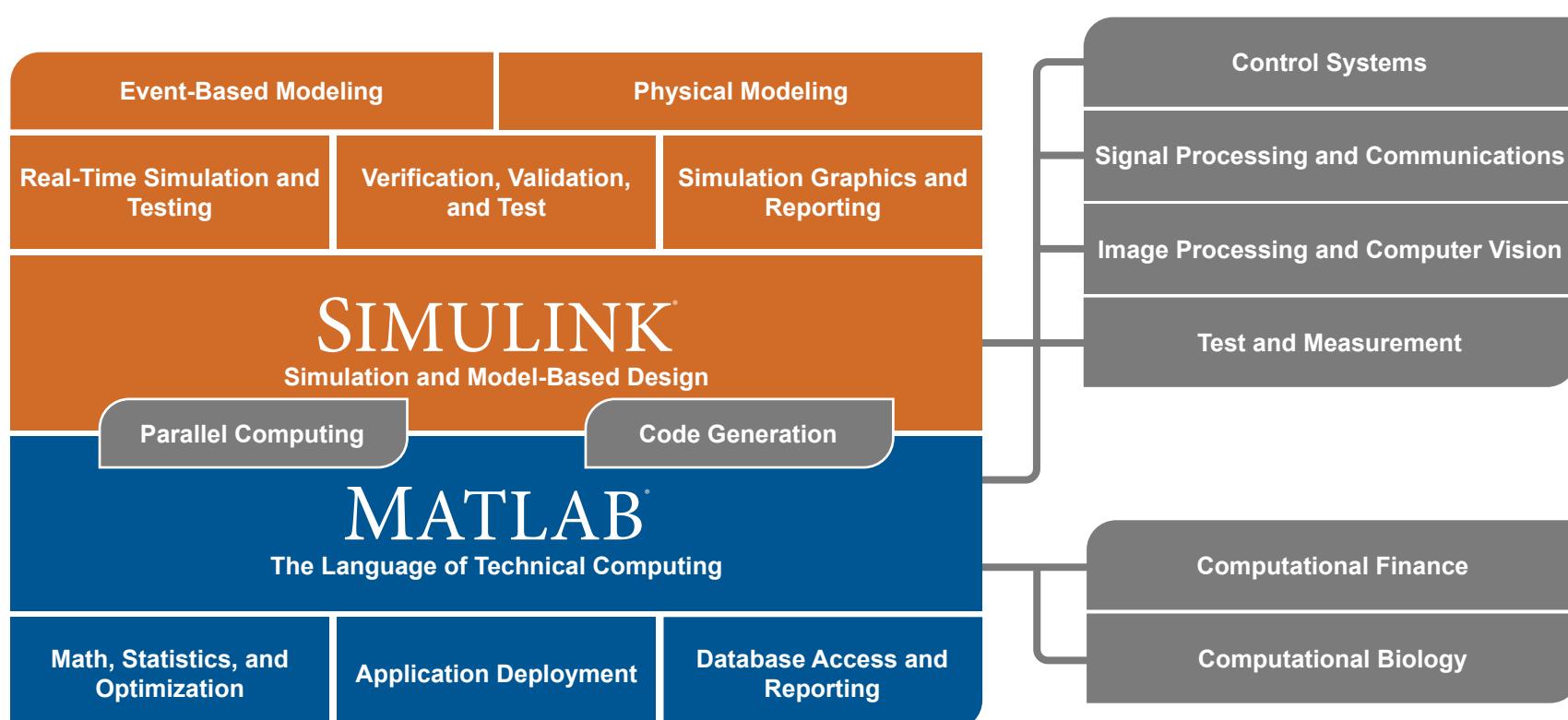
MathWorks® Product Overview

Key Characteristics of MATLAB

- The industry-standard, high-level programming language for algorithm development
- Numeric computation
- Parallel computing, with multicore and multiprocessor support
- Data analysis and visualization
- Toolboxes for signal and image processing, statistics, optimization, symbolic math, and other areas
- Tools for application development and deployment

Key Characteristics of Simulink

- Linear, nonlinear, discrete-time, continuous-time, hybrid, and multirate systems
- Foundation for Model-Based Design, including multidomain system modeling, real-time testing, automatic code generation, and verification and validation
- Open architecture for integrating models from other tools
- Applications in controls, signal processing, communications, physical modeling, and other system engineering areas

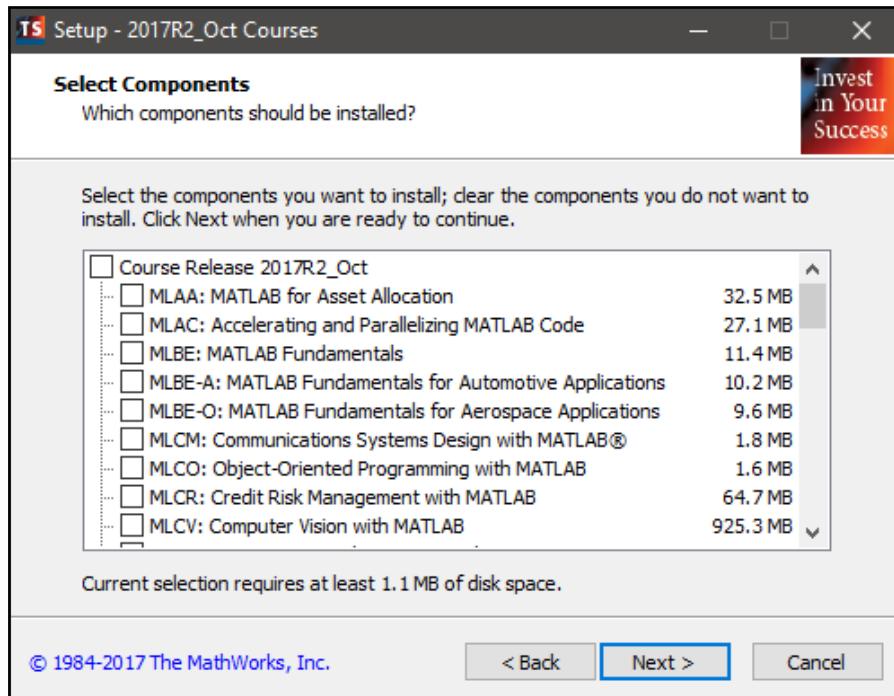


Introduction

Computer Setup

To get ready for class, you need to install the examples and exercises from your course DVD or from a USB drive provided by the instructor. Follow these steps:

1. Put the DVD in the DVD drive or plug in the USB drive.
2. The installer application will open automatically. If the installer application does not open automatically, open the DVD drive or USB drive in Windows® Explorer. Run the file **English_20XXRX_MMM.exe**.
3. Follow the prompts in the installer through the installation process. A shortcut will be created on your desktop to start MATLAB for this class.

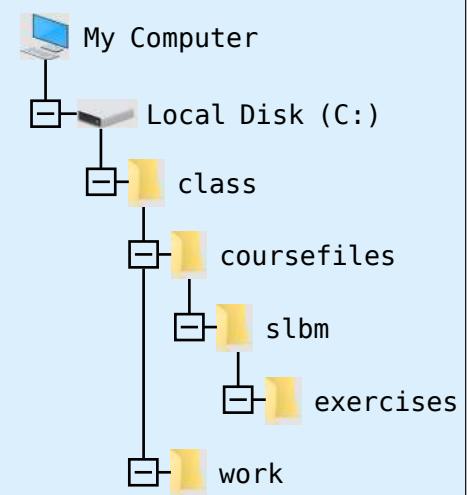


Typical setup

The installation creates a subdirectory of your chosen root directory (the default is C:\class) called **coursefiles**. This directory has subdirectories for each of the courses you install.

Each course directory has a subdirectory called **exercises** that contains all exercises and their solutions.

There is a subdirectory called **work**, which is empty. During class, put all your work in this subdirectory, so that it is on the path and easy to find.



In the installer, you have the following options:

- Choose a class root directory for your course files.
- Choose the courses for which you need to install the files. (Examples and exercises for all of our courses are on the DVD or USB drive.)
- Create a shortcut on the desktop to start MATLAB for this class. (This shortcut runs a **startup.m** file when starting MATLAB, that is customized for the installed course files)

Note A minimalistic install can be performed by navigating to the DVD drive or USB drive in MATLAB, executing the **installer.m** script inside the **zipfiles** folder, and selecting the appropriate courses to install.

Introduction

Battery Modeling and Algorithm Development

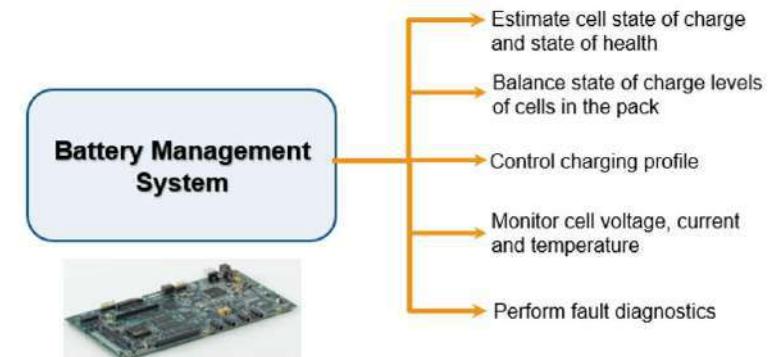
Recent advances in battery technology have facilitated the increase of battery-powered systems for a range of applications, from individual cellular phones to electrical grids and automobiles. Battery engineers are tasked with the development and deployment of efficient control algorithms for the battery management systems that regulate the battery pack operation to derive maximum efficiency out of the pack.

In this course, you will explore how MathWorks tools can aid you in:

- Model a cell unit including thermal and fading characteristics
- Perform cell characterization
- Construct battery packs and attach cooling plates
- Estimate State-Of-Charge and State-Of-Health
- Design the key functionalities of a battery management system including supervisory control, cell balancing, and fault diagnosis



Battery Pack



Battery Management System

Introduction

Course Learning Outcomes

The attendee will be able to

- Perform cell characterization
- Model battery packs in Simulink®
- Add thermal fidelity to battery models
- Design a control scheme for charging the battery
- Perform state estimation and cell balancing
- Compute current limits and design fault diagnostic system
- Perform closed-loop simulation of battery module and battery management system

Introduction

Course Outline

Day 1

- Introduction
- Getting started with a battery cell
- Cell characterization
- Battery pack modeling

Day 2

- State Estimation
- Battery management system
- Fault monitoring and current limit computation
- Conclusion

Appendices

- Kalman Filter
- Create battery objects by scripting
- Exercises



Battery Modeling and Algorithm Development with
Simulink®

Getting Started with a Battery Cell

Getting Started with a Battery Cell

Outline

- Battery terminologies
- Equivalent Circuit Model
- Charge and discharge a cell
- Modeling cell thermal effects
- Modeling cell degradation

The following names are trademarks of The MathWorks, Inc.:

MATLAB®; Simulink®; Simscape™; Simscape™ Battery™

Getting Started with a Battery Cell

Chapter Learning Outcomes

The attendee will be able to:

- Define terms used in a typical battery component
- Create a battery model using the Battery (Table-Based) block
- Discharge a battery cell at a given C-rate
- Charge a cell using the Constant Current-Constant Voltage method
- Add thermal and fading characteristics to the cell

Getting Started with a Battery Cell

Terms and Definitions

Throughout the course, you will encounter several terms that are used commonly while working on battery models or designing algorithms for the battery management system (BMS)

- Cell nominal capacity
- Cell nominal voltage
- C-rate
- Open-circuit voltage
- Terminal voltage
- State of charge

In this chapter, these terms are introduced and defined.

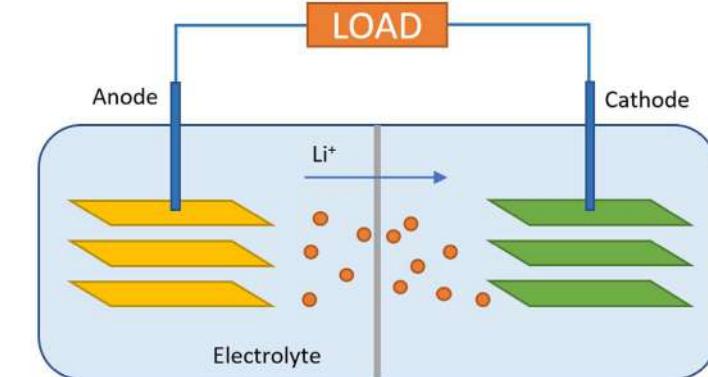
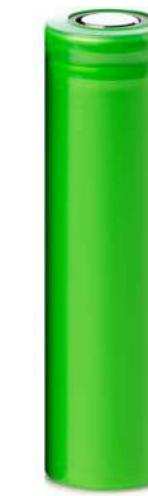
Getting Started with a Battery Cell

Cell

A cell is a single-unit energy source that produces electrical energy from chemical reactions. Typically, a cell contains a cathode(+) and an anode(-) being separated by the separator. They are placed in the electrolyte that assists the movement of the ions. When a cell is connected to an outer circuit (i.e. a resistive load), electrons go through the circuit while ions are transmitted through the electrolyte.

Given the different chemical constituents of a cell, the chemical reactions define the electrode potential. For example, a nickel-based cell gives a voltage of around 1.2 V, while a lithium-based cell provides an average voltage of around 3.6-3.7 V. In this course, the cell used is assumed to have lithium-based chemistry unless specified otherwise.

A battery is made up of cells connected in series or parallel configurations. The terms are often used interchangeably, but this course will use them distinctly: a cell is a singular source of energy, whereas a battery is a component made up of several such individual cells.



Getting Started with a Battery Cell

Nominal Cell Capacity and Voltage

- Nominal Cell Capacity:**

Cell capacity specifies the amount of charge a cell can hold. It is normally expressed with a unit of Ampere-hour (Ah). For example, a cell capacity of 250 mAh implies that the cell can supply a current of 250 mA for one hour. Typically, you can find the nominal cell capacity provided by the manufacturer on the cell packaging. It indicates the average capacity you can get from a fully charged cell under specified load conditions and temperature.

- Nominal Cell Voltage:**

The nominal cell voltage is the average voltage that can be obtained from the cell. The voltage provided by the cell is higher than the nominal voltage when fully charged, and lower than the nominal voltage when fully discharged.

Try

What is the nominal capacity and voltage in the lithium-ion cells shown below?

Combining the nominal cell capacity and voltage, you can generally gauge the average energy a cell can provide. But do note that these values are measured under nominal conditions. For example, a cell might not provide the same amount of energy when the temperature is too low. It is important to consider the environment and the load conditions when sizing the system.



Getting Started with a Battery Cell

C-rate

- C-rate**

The C-rate is computed as the ratio of charge or discharge current to the rated capacity of the cell.

$$\text{C-rate} = \frac{I}{q_{nom}}$$

Where

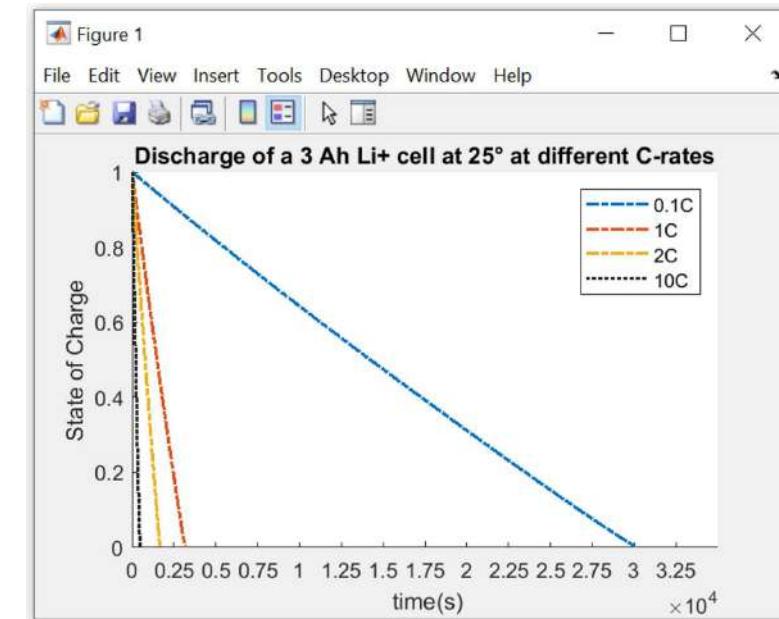
I – current going through the cell

q_{nom} – nominal cell capacity in Ah

If a cell is rated to have 2 Ah capacity and it is charged using a current of 2 A, then the cell is said to have charged at 1 C rate. If the same cell is discharged at 0.5 C rate, it implies that the cell is providing a current of 1 A for a duration of two hours to the load.

Try

What's the discharge current when discharging a 10 Ah cell with 0.1C?



Getting Started with a Battery Cell

State of Charge

- State of Charge

The State of Charge (SOC) of a cell is a virtual gauge indicating the amount of capacity remaining in the cell. SOC is defined as the ratio of current charge to nominal cell capacity.

$$SOC = \frac{q(t)}{q_{nom}}$$

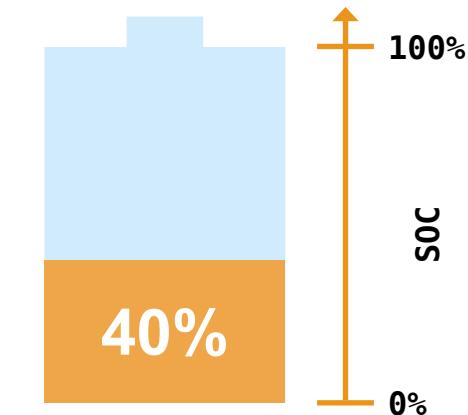
where

$q(t)$ – present cell charge

q_{nom} – nominal cell capacity in Ampere-seconds

SOC is a unitless value. It varies from 0 to 1 or 0% to 100%, which represent fully discharged and fully charged states respectively.

Ideally, the SOC of a cell can be calculated by measuring the concentration of the ions at one electrode. But it is hard to implement outside the lab. In reality, SOC is estimated based on measurable values such as the terminal voltage and current. We are going to discuss SOC estimation in the State Estimation chapter.



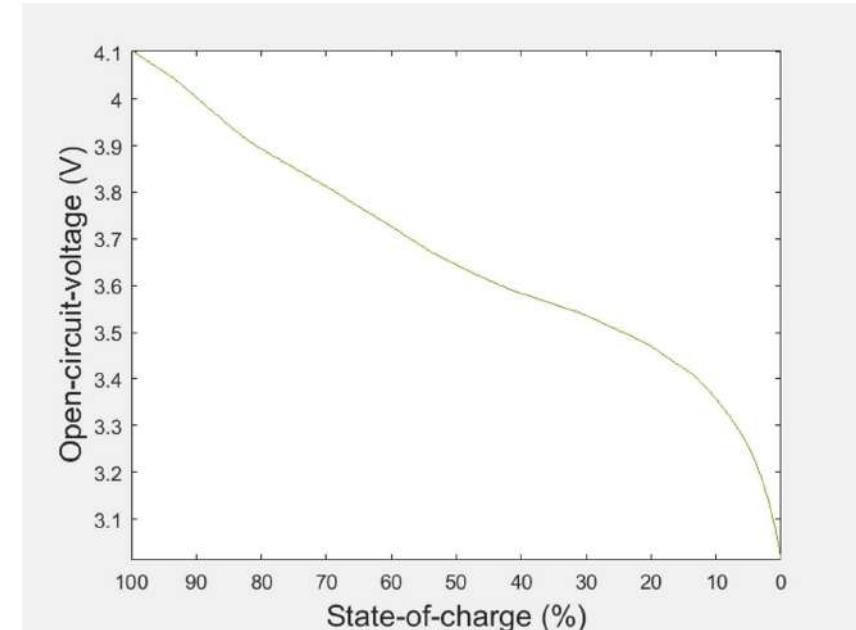
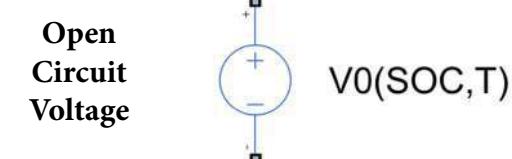
Getting Started with a Battery Cell

Equivalent Circuit Model, Open-Circuit Voltage, and Terminal Voltage

To accurately estimate the states of a cell, you need to have a good plant model that closely approximates the real cell behaviors. There are multiple ways to model the dynamics of a cell, such as modeling the electrochemistry of the cell. In this training, you are going to use the Equivalent Circuit Model (ECM) to model the cell. The ECM represents the cell's input/output relationship by using electric circuits. Compared with other methods, ECM has a good balance between simulation speed and accuracy which is more suitable for industrial applications.

- **Open Circuit Voltage**

A cell can be modeled as simply as a voltage source, where the voltage is a function of SOC and temperature. Typically, when a cell is not connected to any load, the voltage of a battery is a fixed value at a specific SOC and temperature. This voltage is recognized as the Open Circuit Voltage (OCV) and its value is often provided by the manufacturer. On the right is the SOC-OCV curve of a LiNiMnCo (NMC) cell.



Getting Started with a Battery Cell

Equivalent Circuit Model, Open-Circuit Voltage, and Terminal Voltage (Continued)

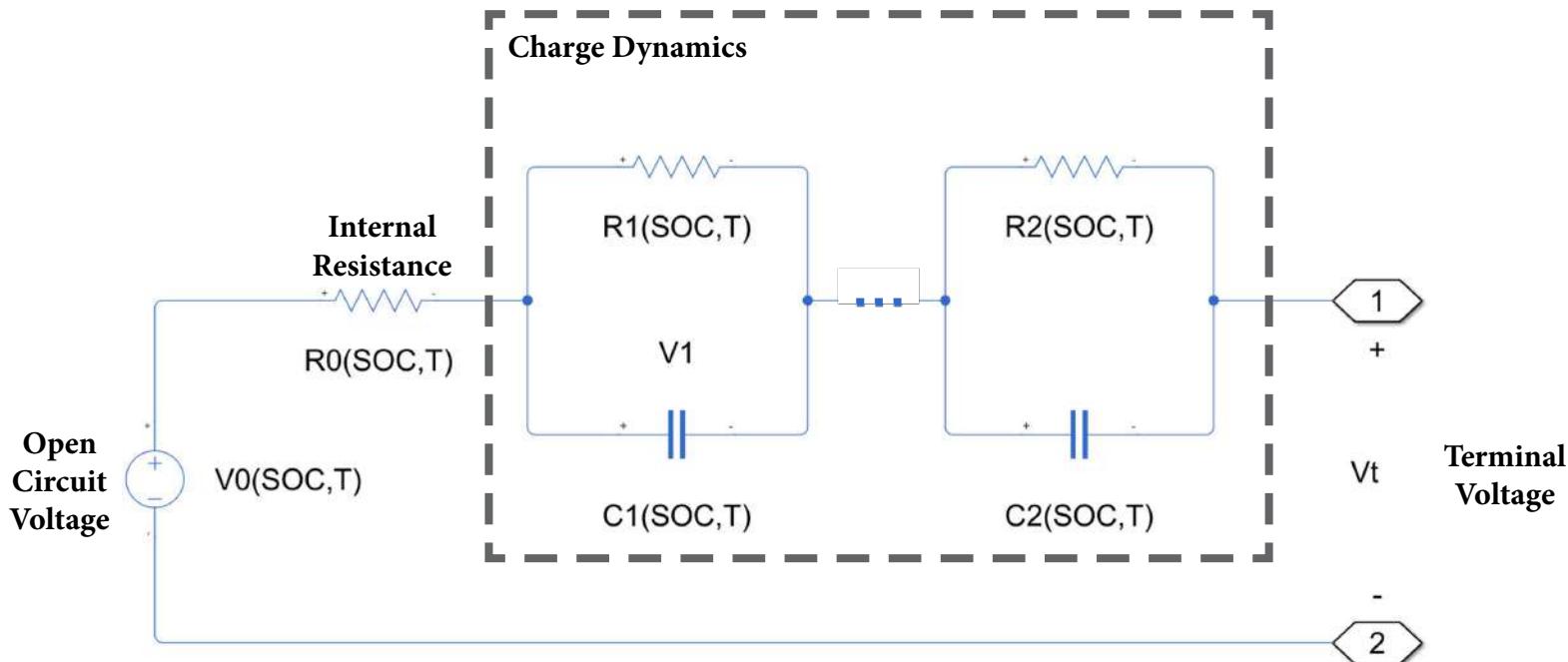
- Terminal Voltage

The terminal voltage of a battery is the measurable voltage difference between the two electrodes. When a cell is not connected to a load, the terminal voltage of the battery equals OCV. However, when the cell is in use, the measurable terminal voltage is subject to the cell's internal resistance and the diffusion dynamics of the ions.

- Equivalent Circuit Model

To model the cell behavior with more detail, below is the Thevenin ECM of a cell. The series resistor R_0 represents the Ohmic loss when current flows through the cell. The series of RC pairs mimic the diffusion of the ions within the cell. The modeler must decide the number of RC branches by considering the tradeoff between model accuracy and model complexity.

Note that the open-circuit voltage, V_0 , the ohmic resistance, R_0 , and the RC branches parameters are dependent on cell SOC and temperature.



Getting Started with a Battery Cell

Battery (Table-Based) Block

In Simulink, the Battery (Table-based) block from the Simscape > Battery > Cells library represents the high-fidelity equivalent circuit model described previously. The block calculates parameters (V_o , R_o , etc.) as a function of SOC and optional temperature using lookup tables.



Under the **Main** tab, you can use the **Tabulate parameters over temperature** setting to decide whether to tabulate parameters over temperatures or not. When the selection is **No**, V_o , R_o are 1D vectors with the same number of elements as the SOC vector. When the selection is **Yes**, V_o , R_o are 2D vectors with the same dimensions of (SOC, T).

Main	
> Vector of state-of-charge values, SOC	[0, .1, .25, .5, .75, .9, 1]
Tabulate parameters over temperature	Yes
Current directionality	Disabled
> Vector of temperatures, T	[278, 293, 313] K
> Open-circuit voltage, V0(SOC,T)	[3.49, 3.5, 3.51; 3.55, 3.57, 3.56... V

Temperature 

SOC 

V0(V)	5°C	20°C	50°C
0%	3.49	3.50	3.51
50%	3.71	3.71	3.72
100%	4.19	4.19	4.19

Try

```
>> cellDischarge01_start
```

Add a Battery (Table-Block) from the Simscape > Battery > Cells library into the model.

You can select the charge dynamics topology under the **Dynamics** tab. To use one RC branch for charge dynamics, select **One time-constant dynamics** as **Charge dynamics**.

Dynamics	
Charge dynamics	One time-constant dynamics
> First polarization resistance, R1(SOC,T)	[.0109, .0029, .0013; .0069, .00... Ohm
> First time constant, tau1(SOC,T)	[20, 36, 39; 31, 45, 39; 109, 10... s

The initial conditions of the battery can be set under the **Initial Targets** tab. By default, the initial cell SOC is set to 1. You can override the **Value** to start at a different SOC.

Initial Targets	
> <input type="checkbox"/> Current (positive in)	
> <input type="checkbox"/> Terminal voltage	
> <input checked="" type="checkbox"/> State of charge	
Priority	High
Value	1

Getting Started with a Battery Cell

Block Parameterization Manager

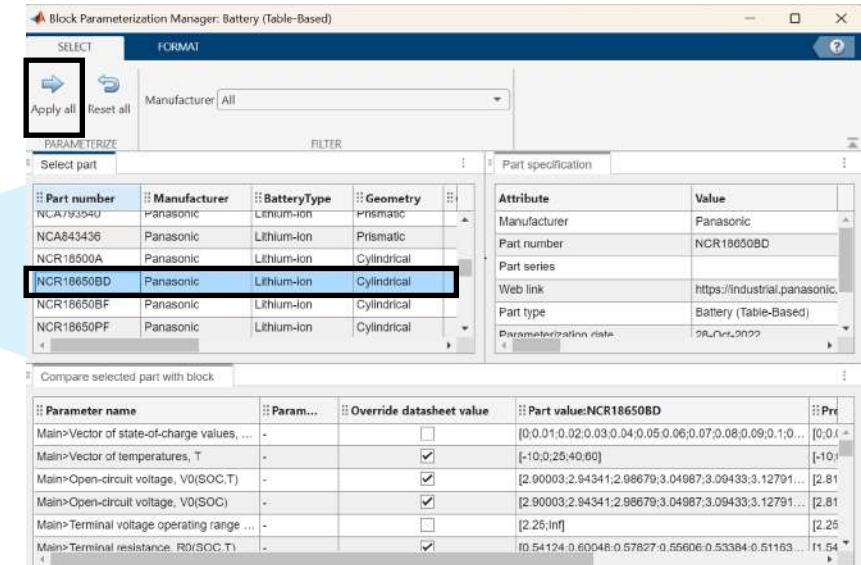
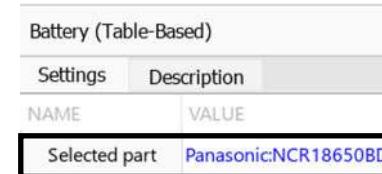
To ensure the Battery (Table-Based) block can accurately represent the real battery, it is crucial to provide accurate parameters. The upcoming chapter will introduce you to the parameter estimation workflow, which involves collecting experimental data and performing parameter estimation. This process can be time-consuming, as battery experiments often span days to complete. Alternatively, when you don't have battery parameters ready, you have the option to utilize pre-parameterized battery data available through the **Block Parameterization Manager**.

1. Click on the hyperlink under the **Selected part** to open the **Block Parameterization Manager**.
2. Select NCR18650BD from the **Select part** table.
3. Click **Apply all** to update the Battery (Table-Based) block parameter.

Try

Follow the steps to update the cell parameters using **Block Parameterization Manager**.

These parameters are collected from manufacturer data sheets. After selecting a part, you can find details about the parameters under Compare selected part with block. If the Override datasheet value is checked for a field, the corresponding Part value will override the block value after Apply all. For Battery (Table-Based) block, the provided parameters are Open circuit voltage V_0 , Terminal resistance R_0 , Cell capacity AH, and temperature independent fading table for open-circuit voltage and cell capacity.



Getting Started with a Battery Cell

Create a Discharge Circuit with Simscape

Next, you are going to connect the battery with a resistive load to perform a simple discharge.

1. Add a Resistor block from the **Simscape > Foundation Library > Electrical > Electrical Elements** library. Set the **Resistance** to 1 Ohm.
2. Turn the Resistor to vertical by pressing **Ctrl+R**. Connect the Battery (Table-Block) and the Resistor together to form a circuit.
3. To monitor the voltage across the load, add a Voltage Sensor block from **Simscape > Foundation Library > Electrical > Electrical Sensors** library. Connect the + and – terminals of the Voltage Sensor across the resistor.

In Simscape, the line connected between blocks represents the physical connections which is different from Simulink. You cannot directly connect a Simscape signal to a Simulink block. Instead, you need to convert the Simscape signal to a Simulink signal.

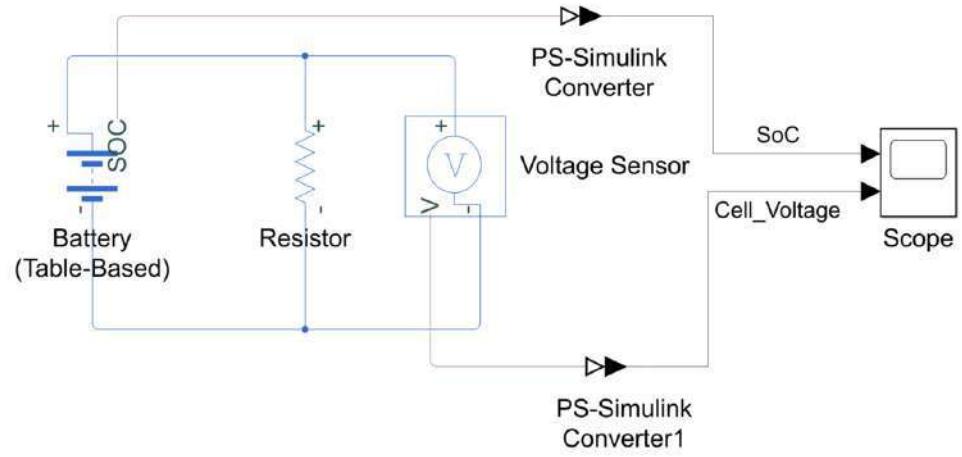
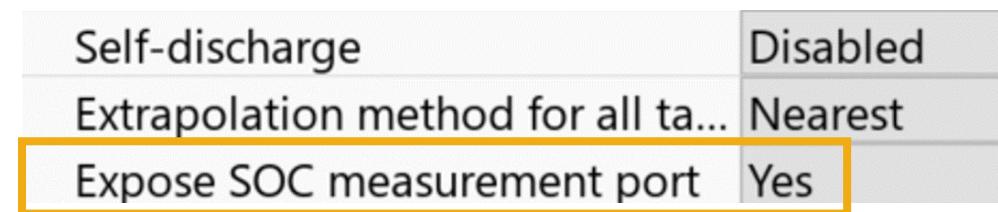
4. Add a PS-Simulink Converter block from the **Simscape > Utilities** library.
5. It is recommended that you specify the unit of a signal when converting. Set the **Output signal unit** of the PS-Simulink Converter to **V**.
6. Connect the V port from the Voltage Sensor block to the PS-Simulink Converter block, and the output to the scope.

The Battery (Table-based) block also provides an instrument to monitor the cell SOC. To enable this functionality:

7. Change **Expose SOC measurement port** to **Yes** in the **Block Parameters** dialog.
8. Connect the SOC port to a PS-Simulink Converter, and the output to the scope.

Try

Follow the steps to build the discharge circuit.



PS-Simulink Converter	
Settings	Description
NAME	VALUE
Vector format	inherit
Output signal unit	V

Getting Started with a Battery Cell

Battery (Table-Based) – Discharging

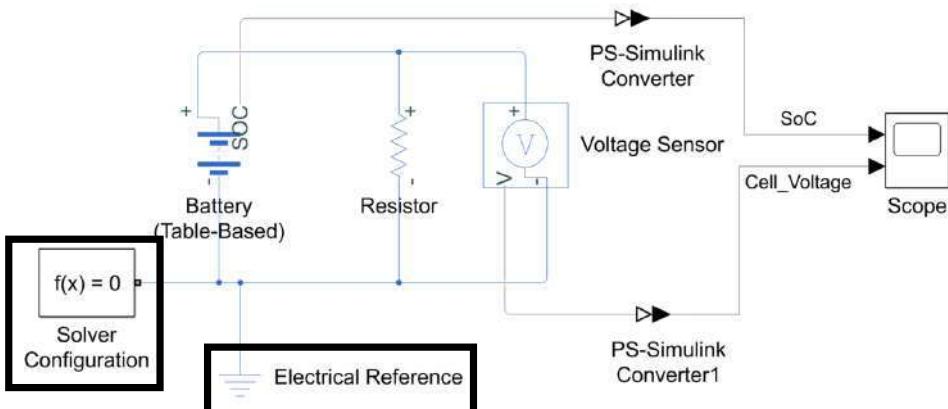
Other than the blocks building up the circuit, Simscape requires reference and utility blocks to facilitate the simulation.

For each Simscape domain, at least one reference block is required to establish a reference level for physical quantities. For example, if the discharge circuit is in the electrical domain, an Electrical Reference block should be added.

9. Connect an Electric Reference block from the **Simscape > Foundation Library > Electrical > Electrical Elements** library to the network.

Simscape requires a Solver Configuration block for each Simscape network. The block provides additional solver options and settings that are required before starting a simulation. The Solver Configuration block can be connected anywhere in the Simscape network.

10. Add a Solver Configuration block from the **Simscape > Utilities** library. Connect the block to a point in the circuit.



Try

```
>> cellDischarge02_soln
```

Normally there's no need to change the settings in the Solver Configuration block. By default, the Simscape diagram is solved by the global solver selected in the model's Configuration Parameters. Optionally you can select to use a local solver for the Simscape network separately from the rest of the Simulink model. In cases such as hardware-in-the-loop (HIL) where the simulation speed is a concern, using a local solver could potentially free other parts of the Simulink model to run faster (i.e., use a discrete global solver when there's no continuous state in the rest of the model).

11. Simulate the model. The cell slowly discharges through the resistor. Cell SOC gradually reduces from 1 to 0 and the cell voltage drops accordingly.

Note that if you extend the simulation time to 3600s, the cell will become over-discharged and you will get a warning message stating that “State of charge must be greater than or equal to zero”. When SOC is out of range, the voltage value will either hold the nearest or extrapolate linearly depending on your settings in the **Extrapolation method for all tables** in the **Block Parameters**. It is the responsibility of the Battery Management System (BMS) to limit the battery discharge within a minimum voltage cut-off range to avoid damaging the cell. You will learn to build a BMS later in the course.

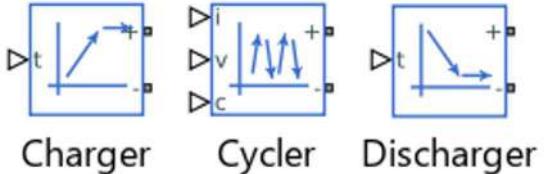
Getting Started with a Battery Cell

Constant Current – Constant Voltage (CC-CV) Charging

Constant Current – Constant Voltage (CC-CV) charging is one of the commonly used methods to charge a lithium-based cell. It is used to prevent the battery from being overcharged while charging it to its maximum capacity.

In the CC-CV charging scheme, the cell is charged using a constant current first until its terminal voltage reaches a predetermined maximum voltage. However, the OCV hasn't reached the maximum voltage yet since there are current-dependent voltage drops across the resistors. To further charge the cell while avoiding overvoltage, the charger switches to constant voltage mode. As OCV continues to increase, less current is required to keep the terminal voltage constant. When the charging current drops around zero, the battery is fully charged.

Simscape Battery simplifies the process of building charging, discharging, and cycling circuits by offering readily available blocks. These blocks are located under **Simscape > Battery > Cyclers** library.



You will use the Charger block to implement the CC-CV logic.

1. Open model **cellCharge01_start.slx**
2. Add a Charger block and connect it with the circuit accordingly.
3. Add a PS-Constant block with a value of **1** and connect it to the **t** port of the Charger.
4. Set up the **Block Parameters** of the Charger as shown in the figure.

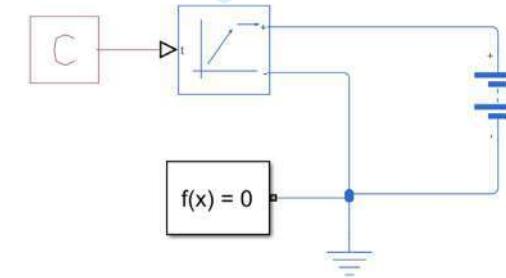
Try

Follow the steps to charge the battery using CC-CV logic.

>> cellCharge01_start

Parameters

Constant charging current	27	A
Voltage threshold	4.08	V
Constant charging voltage	4.08	V
Current threshold	CC_current/30	0.9 A

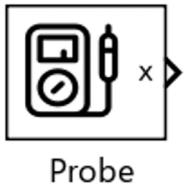


Field **Constant charging current** denotes the current used in the CC phase. The **Voltage threshold** indicates the voltage to switch from CC to CV. Then the charger keeps the voltage constant at **Constant charging voltage**. In the CV phase, the required current to keep the same constant voltage will gradually decrease. When it drops below the **Current threshold**, charging is considered completed and all current is cut off from the circuit. Typically a **Current threshold** of $C/30$ is considered small enough to end the CV charging.

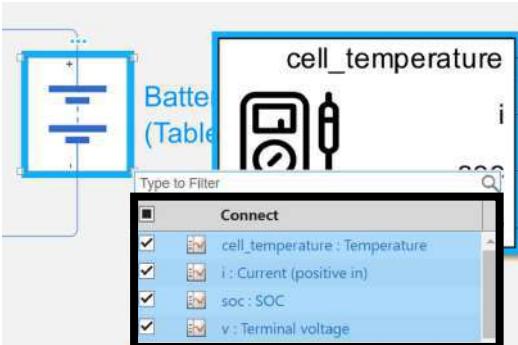
Getting Started with a Battery Cell

Probing Signals

There are multiple ways to collect measurements from a Simscape diagram. In the previous exercise, you explicitly added voltage measurement to the circuit. As an alternative way, the Probe block from the **Simscape > Utilities** library lets you select variables from another block in the model and output the measurements as Simulink signals.



1. Add a Probe block from the **Simscape > Utilities** library.
2. Double-click on the Probe block to enable selection. Then single-click on the Battery (Table-Based) block.
3. In the drop-down menu, select **cell temperature**, **i**, **SOC**, and **v** as shown in the figure.
4. Connect the Probe to the Scope.
5. Simulate the model.



After selecting a block, the Probe context menu contains all the externally accessible variables of the selected block. Note that before 23a, only the variables exposed in the Initial Targets section of the Block Parameters are available for probing.

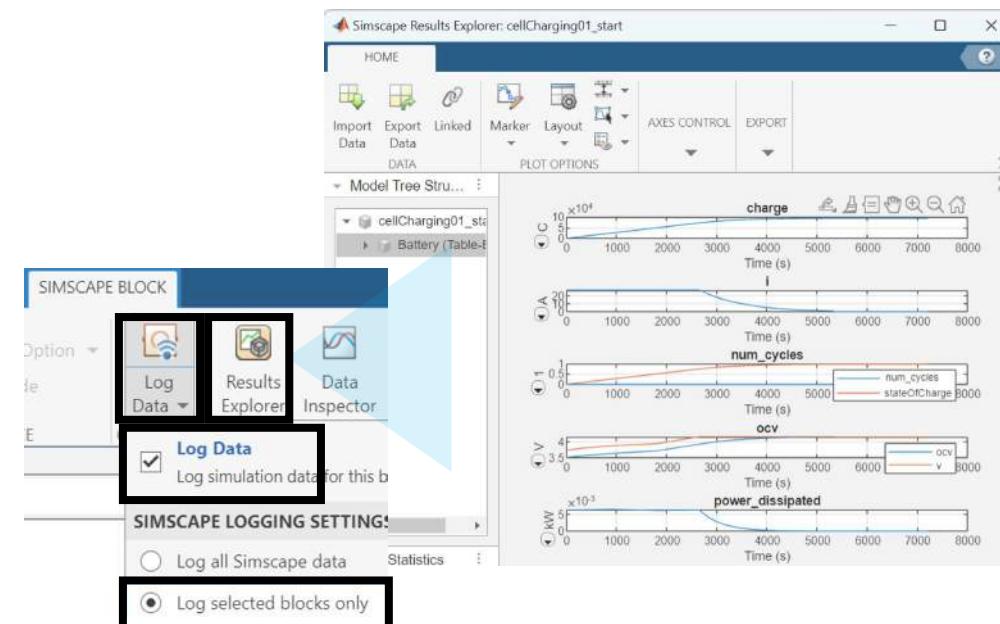
Try

Follow the steps to add a Probe block to the model.

```
>> cellCharge02_probe
```

Lastly, **Simscape Results Explorer** offers a third way of probing Simscape signals without adding blocks to the model.

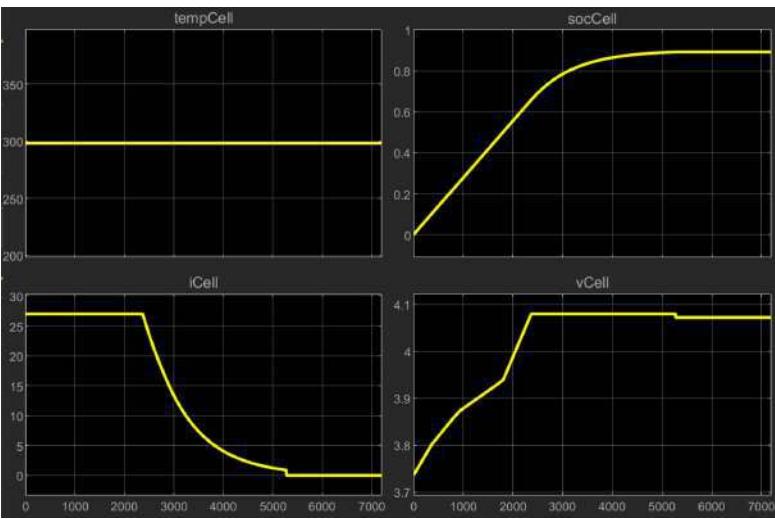
1. Select the Battery (Table-Based) block and go to the **SIMSCAPE BLOCK** tab of the block.
2. Under **Log Data**, check the checkbox of **Log Data** and select **Log selected blocks only**.
3. Simulate the model. Click on the **Results Explorer** to view the results.



Getting Started with a Battery Cell

Modeling Thermal Effect

In the CC phase, the charging current is held and voltage starts to increase. During this time, the SOC increases linearly. Once the terminal voltage hits the threshold, the charging switches over to CV mode. The voltage is kept as a constant and the current gradually decreases to the current threshold.



Note that the following convention is applied for the current:

- A positive indicates charging.
- A negative indicates discharging.

The simulation temperature of the cell is held constant currently. However, due to the internal resistance of the battery, heat is generated when current flows through the cell. The battery temperature is determined by the summation of all the ohmic losses included in the cell model.

$$M_{th}\dot{T} = \sum_i \frac{V_{T,i}^2}{R_{T,i}}$$

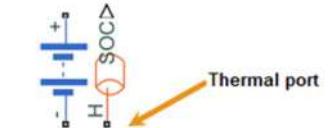
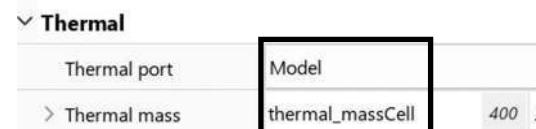
Try

Enable the **Thermal port** of the Battery (Table-Based) block and set the **Thermal mass** parameter.

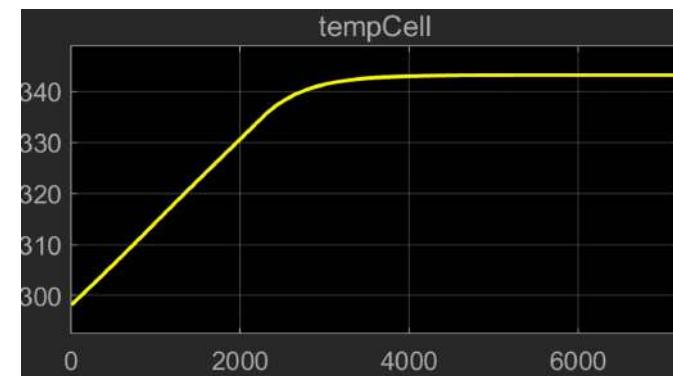
M_{th} is the thermal mass of the battery. It equals the mass of the cell times the specific heat capacity of the cell. Specific heat capacity has the unit of J/kgK. It describes how much heat is needed for one unit of mass to increase the temperature for one unit. The specific heat capacity of the cell should be determined by thermal experiment.

To model and measure the cell temperature changes, you need to enable the thermal port of the Battery(Table-based) block.

1. Go under the **Thermal** tab of the **Block Parameters**. Set the **Thermal port** to **Model**.
2. Change the Thermal Mass of the battery to **thermal_massCell**.
3. Simulate the model.



By enabling the thermal model, cell temperature continuously rises during charging. The temperature rise is proportional to the charging current.



Getting Started with a Battery Cell

Modeling Thermal Effect (Continue)

You can further connect the **Thermal** port of the Battery (Table-Based) block with additional thermal circuits to model the cell heat exchange with the ambient environment. Assume a simple case where the ambient temperature is the average air temperature surrounding the cell. The battery exchanges heat with the ambient environment through convection represented in the formula.

$$Q = kA(T_A - T_B)$$

T_A and T_B represent the temperature of the battery and the ambient environment. A is the surface area of the battery, Q is the heat flow, k is the heat transfer coefficient, and the unit of k is $\text{W}/\text{m}^2\text{K}$. Typically, the convection heat transfer coefficient for air is in the range of 2.5-25.

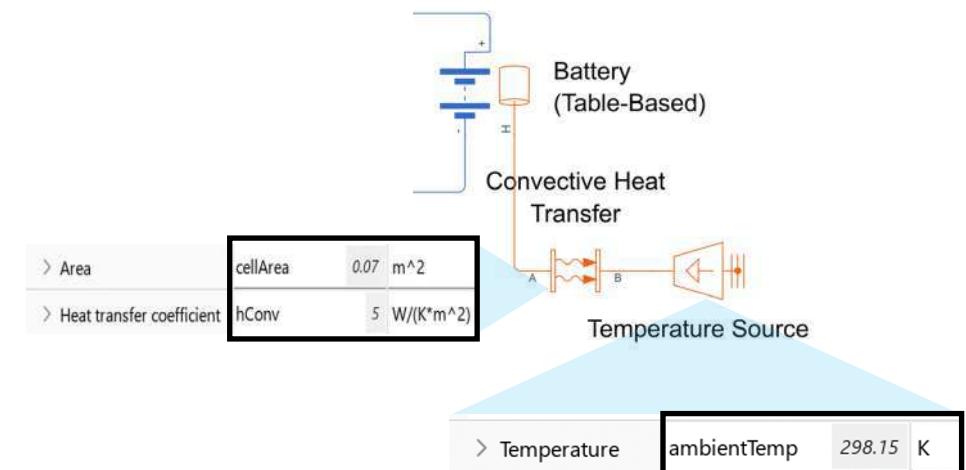
In Simulink, you can use a Temperature Source block to represent the ambient temperature and a Convective Heat Transfer block to model the heat exchange.

1. Add a Temperature Source block from the **Simscape > Foundation Library > Thermal > Thermal Sources** library into the model.
2. Add a Convective Heat Transfer block from the **Simscape > Foundation Library > Thermal > Thermal Elements** library to the model.
3. Connect the **Convective Heat Transfer** block between the **Temperature Source** block and the thermal port of the battery. Note the direction of the **Convective Heat Transfer** block does not impact the result.
4. Set the parameters of the two blocks as shown in the figure.
5. Simulate the model.

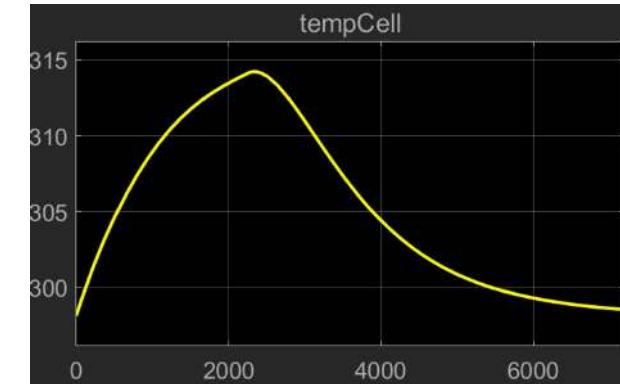
Try

Follow the steps to add a Probe block to the model.

>> cellCharge03_thermal



By modeling the ambient cooling circuit, the highest cell temperature decreases. Cell temperature still rises during the CC phase when more heat is generated compared to what can be dissipated. Less heat is generated during the CV phase compared to ambient cooling, cell temperature gradually winds down.



Getting Started with a Battery Cell

Modeling Battery Degradation

All batteries have a limited lifespan. A battery degrades during use and storage due to irreversible side reactions, such as the growth of the new solid-electrolyte interface layer, li-ion plating, etc. As a consequence, the performance of the battery continues to decline and reaches a point where it needs to be replaced. Typically, if a battery's capacity fades to 80% of its original value, it reaches the End of Life (EOL).

In the Battery (Table-Based) block, you can specify the battery fade parameters using the **Fade** tab either by using equations or look-up table. Follow the steps below to model a rapid cell fading by using the equation-based method. The cell capacity is set to fade to 80% of its nominal capacity after 10 cycles.

1. Open model `cellFade01_start.slx`.
2. In the **Block Parameters** of the Battery (Table-Based) block, under the **Fade** tab, change the **Fade characteristic** to **Equations**.
3. Change the **Number of discharge cycles, N** to **10**.
4. Change the **Change in cell capacity after N discharge cycles (%)** to **-20**.
5. Simulate the model.

The equation used to represent battery fading is derived from empirical data, indicating that the change in capacity follows a square root relationship with respect to the discharge cycles.

$$q_{fade} = q_{nom} \left(1 + \frac{\delta}{100} \sqrt{\frac{n}{N}} \right)$$

Where

- q_{fade} is the faded capacity
- q_{nom} is the nominal capacity at the beginning of life
- δ is the percent change in the capacity over N cycles
- n is the current count of the discharge cycle

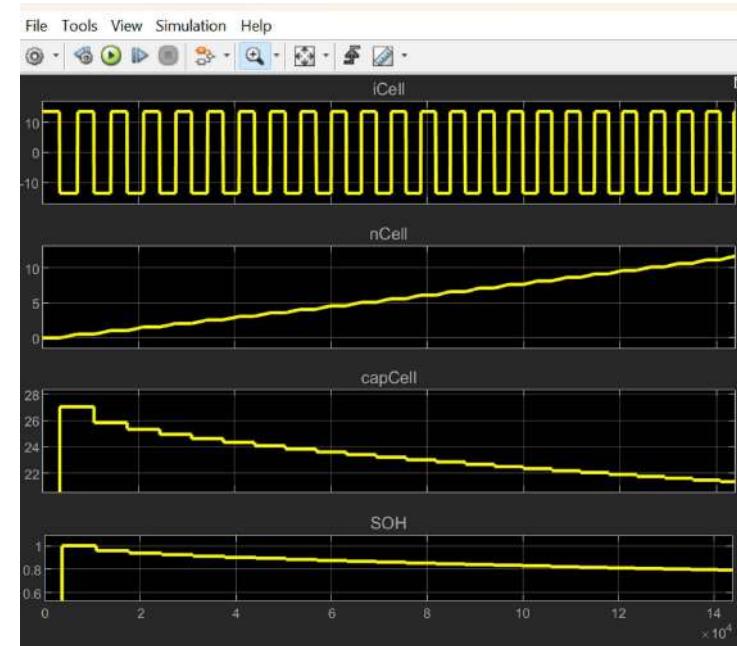
Try

Follow the steps to add battery fading to the model.

`>> cellFade01_start`

Fade	
Fade characteristic	Equations
> Number of discharge cycles, N	10
> Change in open-circuit voltage after N discharge cycles (%)	0
> Change in terminal resistance after N discharge cycles (%)	0
> Change in cell capacity after N discharge cycles (%)	-20

Note that the battery block does not provide capacity measurement as an output. An external capacity estimation is required to capture the change. In the simulation result, the cell capacity is calculated within the `ChargeDischargeControl` subsystem. You will learn more about capacity estimation in the state estimation chapter.



Getting Started with a Battery Cell

Modeling Battery Degradation (Continued)

To more accurately model the battery fade, you can model the battery fading as lookup tables and provide your data. Lookup tables can either be temperature independent or temperature dependent.

1. In the **Block Parameters** of the Battery (Table-Based) block, under the **Fade** tab, change the **Fade characteristic** to **Lookup tables (temperature dependent)**.
2. Set the fields under the Fade tab as shown below.
3. Simulate the model.

Fade	
Fade characteristic	Lookup tables (temperature dependent)
> Vector of discharge cycle values, N	N0
> Vector of temperatures for fade data, Tfade	Tfade < 1x6
> Percentage change in open-circuit voltage, ...	dV0
> Percentage change in terminal resistance, ...	dR0
> Percentage change in cell capacity, dAH(N,...)	dAH
> Percentage change in first polarization resi...	dR1

The battery fading table represents a temperature-dependent cell fading, as depicted in the figure. The percentage of battery fade increases as the cycle count and temperature rise. In practice, it is important to regulate the battery temperature to mitigate the rate of battery fading.

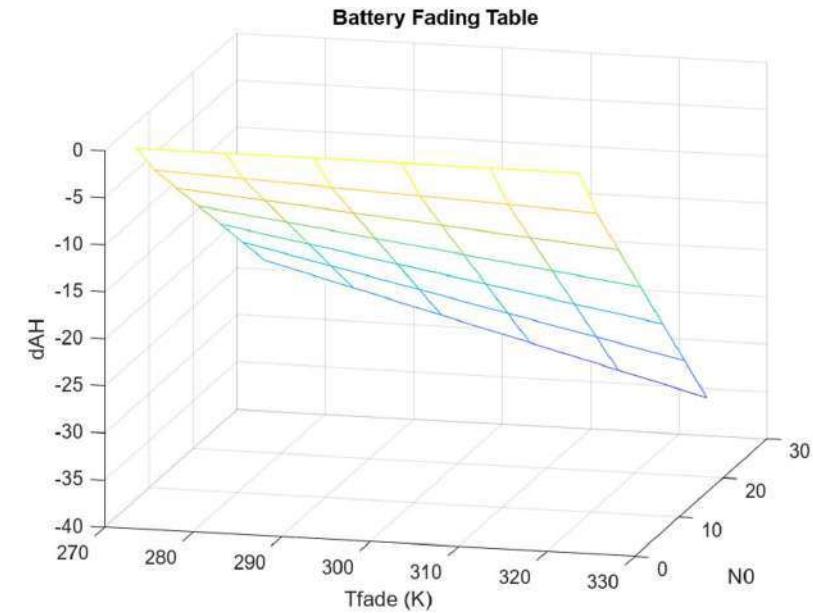
Try

Follow the steps to enable lookup table based fade modeling.

Or,

>> edit cellFadeScript

Run the script and examine the battery capacity fade under different temperatures.



Getting Started with a Battery Cell

Summary

- Battery terminologies
- Equivalent Circuit Model
- Charge and discharge a cell
- Model cell thermal effects
- Model cell degradation

Getting Started with a Battery Cell

Test Your Knowledge

1. Which of the following C rate represents discharging a 20Ah battery with 40A?
 - A. 0.2C
 - B. 0.5C
 - C. 2C
2. (T/F) The unit of SOC is Coulomb or Ampere hour.
3. (T/F) Charging a cell with CC-CV scheme till the end of the CC phase, the measured cell terminal voltage equals to the cell open-circuit voltage.

Getting Started with a Battery Cell

Answers

1. C
2. False
3. False



Battery Modeling and Algorithm Development with
Simulink®

Battery Pack Modeling

Battery Pack Modeling

Outline

- Create battery module and pack objects with the Battery Builder app
- Control the model resolution of battery objects
- Add thermal fidelity to the battery objects
- Model cooling plates attached to the battery objects

The following names are trademarks of The MathWorks, Inc.:

MATLAB®; Simulink®; Simscape™; Simscape™ Battery™

Chapter Learning Outcomes

The attendee will be able to:

- Create battery objects with Simscape Battery
- Model the thermal environment for battery simulation

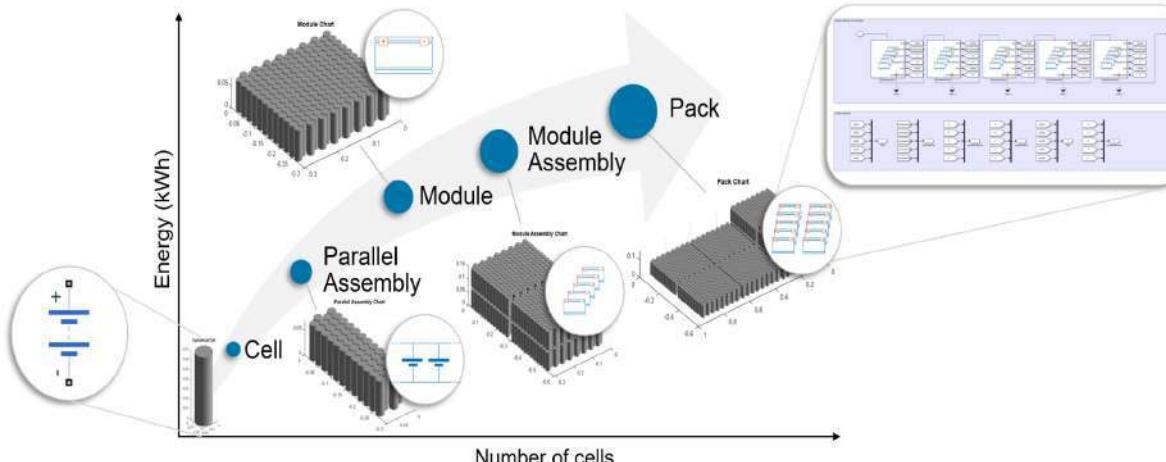
Battery Pack Modeling

Battery Builder

A lithium-ion cell has a nominal voltage of 3.7 V. To address the typical voltage and energy requirements of an electric vehicle (200-800V, 12-100kWh), multiple battery cells need to be connected to increase the voltage and energy output.

A cell module is created by wiring identical cells either in series or parallel combinations to boost the power output. It is then housed within a protective frame to protect the cells from external disturbance. Cell modules can be further combined into a battery pack to meet the energy requirements of a given application.

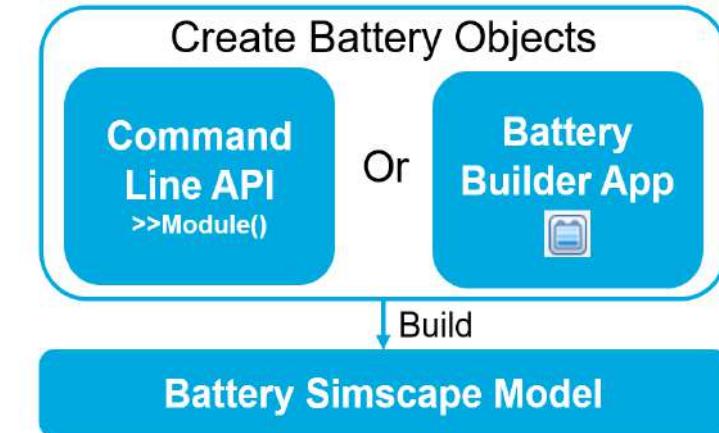
The pack modeling process often requires tedious manual connection of cells and accessories in Simulink. Simscape Battery provides an object-oriented programming way with MATLAB to automate the creation of battery packs. MATLAB objects allow you to define battery specifications, visualize batteries in 3-D spaces, assign simulation strategies, and generate customized Simulink libraries.



You can create battery objects at five different levels of hierarchies:

- **Cell**: creates a single electrochemical battery cell
- **ParallelAssembly**: creates a parallel assembly object with cells connected in parallel
- **Module**: creates a module object with parallel assemblies connected in series
- **ModuleAssembly**: creates a module assembly object with modules connected in series or parallel
- **Pack**: creates a pack object with module assemblies connected in series or parallel

These objects can either be created by scripting or using the **Battery Builder** app. In this chapter, you will follow the workflow of using the **Battery Builder** app to create battery objects in an interactive way. The scripting version can be found in Appendix *Creating Battery Objects by Scripting*.



Battery Pack Modeling

Battery Builder (Continued)

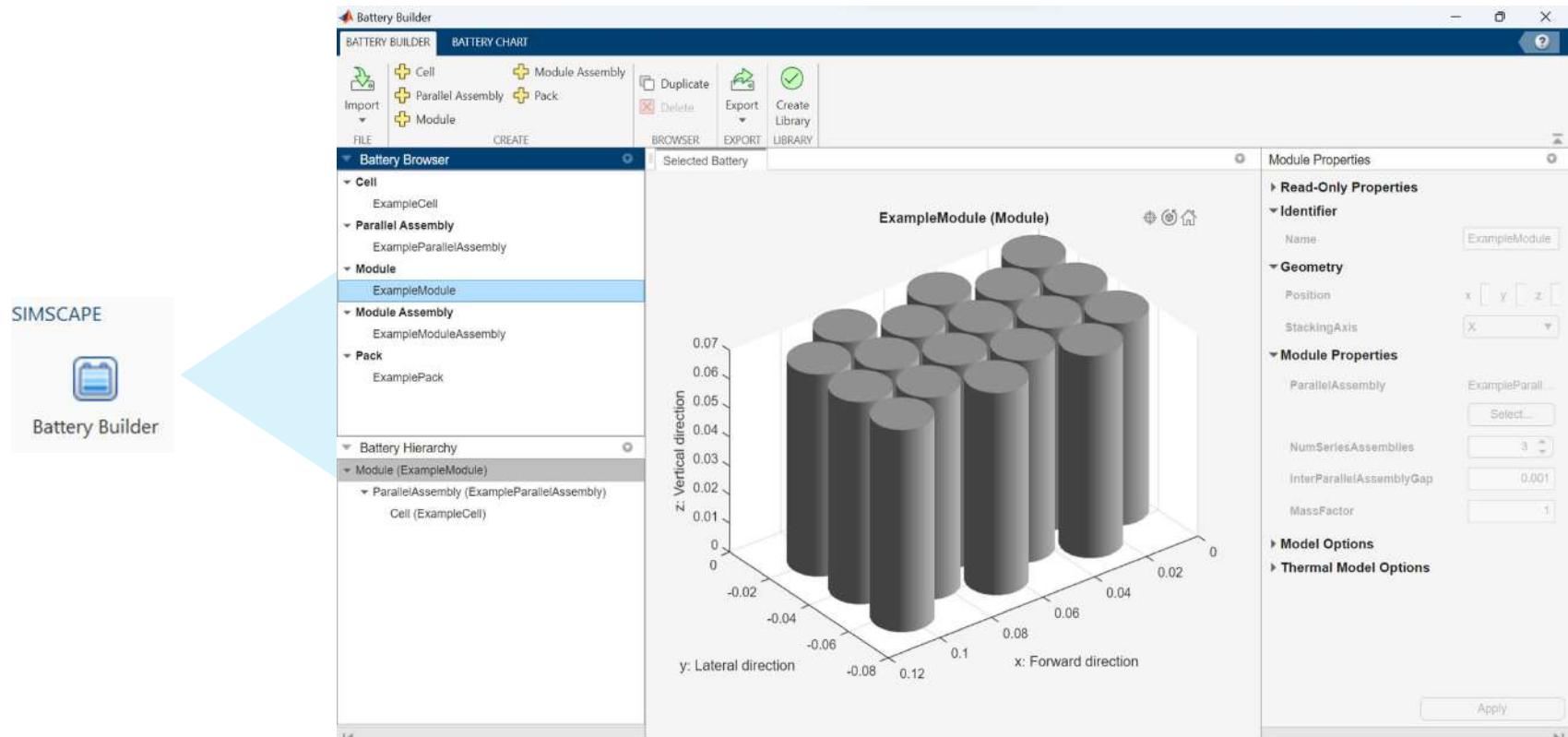
Battery Builder app provides an interactive way of creating, modifying, and visualizing battery objects. To open the **Battery Builder** app, navigate under the **APPS** tab in MATLAB, and find **Battery Builder**. You can also open the app by using the command **batteryBuilder**.

By default, **Battery Builder** app includes a set of battery objects for your reference. You can duplicate the default objects for modification or create battery objects from scratch.

Try

>> batteryBuilder

Open the **Battery Builder** app.



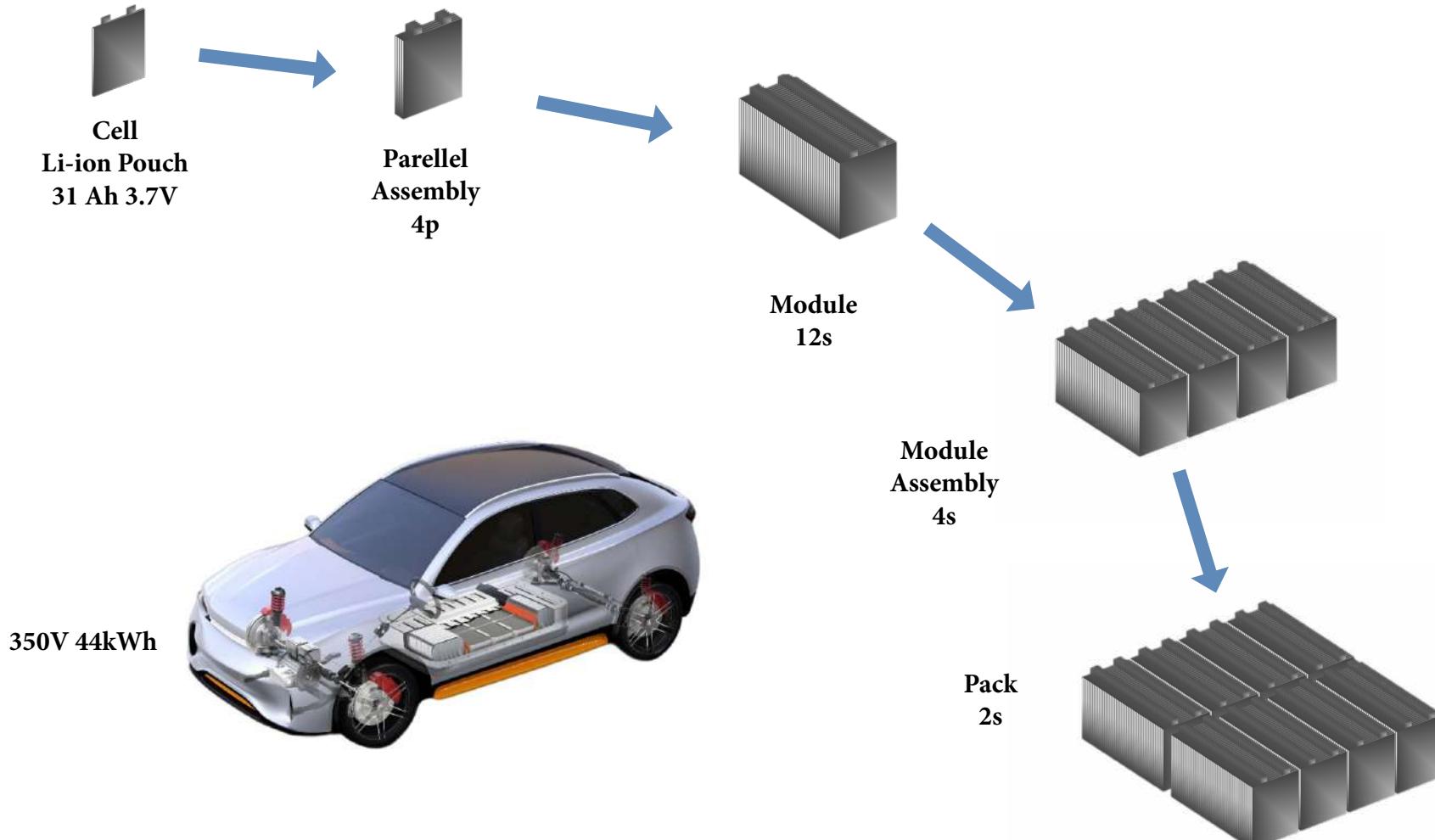
Battery Pack Modeling

Course Example – PHEV Battery Pack

As the course example, you create a battery pack of a plug-in hybrid electric vehicle with a targeting voltage of 350V and an energy level of 44kWh.

The selected cell has a nominal capacity of 31Ah and a nominal voltage of 3.7V. To meet the voltage and energy criteria, the topology of the pack is 96s4p.

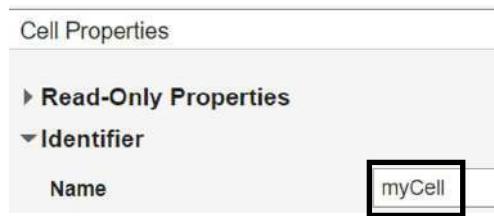
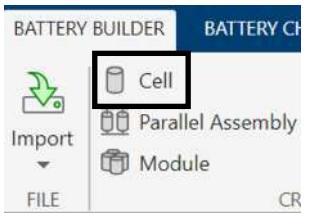
To ensure the battery operates in the desired temperature range, a liquid cooling plate with parallel channels is placed underneath the pack.



Battery Pack Modeling

Creating a Battery Cell Object

1. Instantiate a battery cell object by clicking the **Cell** button under **BATTERY BUILDER > CREATE**.
2. In the **Cell Properties** panel, change the **Name** of the cell to **myCell**.



Cell Geometry

The geometry of a cell impacts the sizing and thermal design of a battery system. Simscape Battery supports three types of cell geometries: cylindrical, prismatic, and pouch. Each geometry requires distinct dimensions for its construction.



Cylindrical Cell

Radius
Height



Prismatic Cell

Length
Thickness
Height



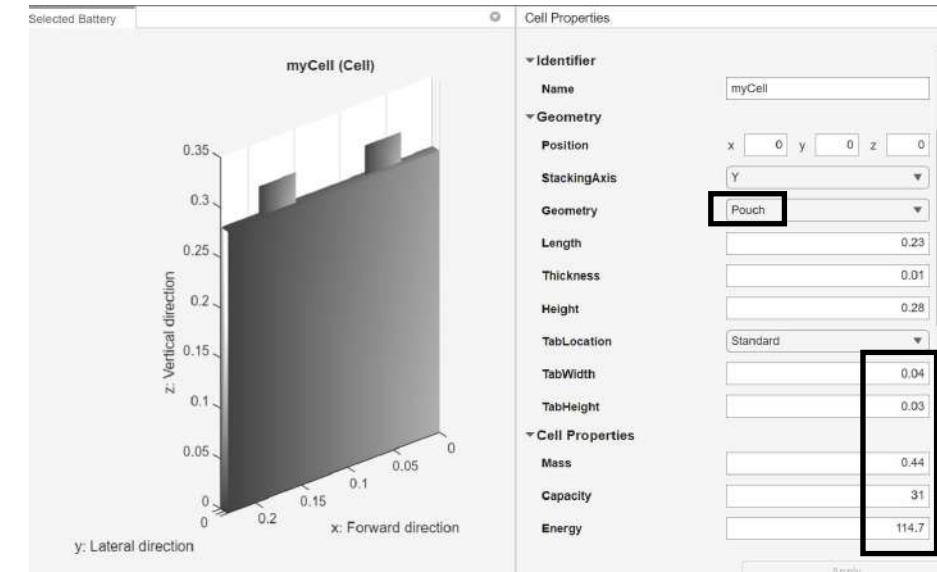
Pouch Cell

Length
Thickness
Height
TabLocation
TabWidth
TabHeight

Try

Follow the steps to create a batter cell object with Pouch geometry.

3. Select **Geometry** as **Pouch** under **Cell Properties**. Click **Apply** to examine the visualization of the pouch cell.



Cell Properties

Along with the battery cell object, you can further specify the mass, capacity, and energy of the cell as **Cell Properties**. These values will be carried over to higher level of hierarchies to calculate the cumulative values for the entire pack.

4. Specify Cell Properties as follow:

- **Mass = 0 . 44** (kg)
- **Capacity = 31** (Ah)
- **Energy = 114 . 7** (Wh)

5. Click **Apply** to apply the changes.

Battery Pack Modeling

Cell Model Options

The **Cell Model Options** controls how the cell appears in Simulink. By default, the Battery (Table-Based) block is selected as the fundamental building block given by **CellModelBlockPath**.

You can specify **CellModelBlockPath** to point to any battery block in the Simulink library. If the existing battery blocks do not meet your requirements, you can write your own block with the Simscape language and generate a Simulink block with **ssc_build**. Additional information about coding with the Simscape language is available in the course *Modeling Physical Systems with Simscape*.

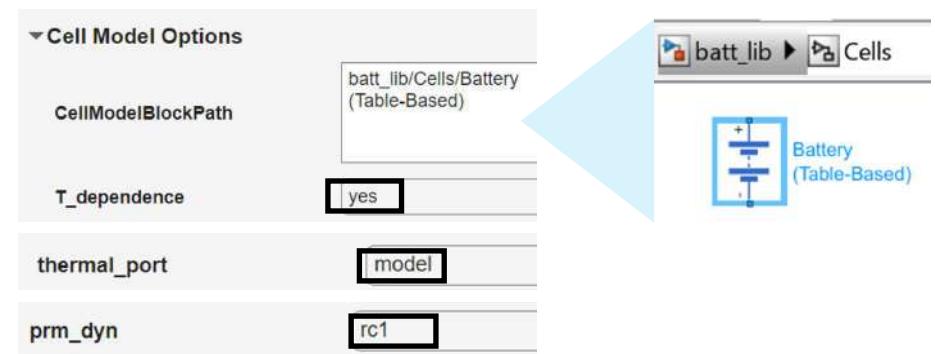
The remaining fields within the **Cell Model Options** align with the **Block Parameters** of the selected battery block. Please refer to the figure for a mapping between the options and the **Block Parameters** dialog.

Note **Cell Model Options** cannot be modified after block creation. The selected options determine which fields appear in the **Block Parameters** dialog, while the unselected options are excluded from the generated block.

1. To enable the thermal modeling options, under **Cell Model Options**, change **T_dependence** to **yes** and **thermal_port** to **model**.
2. To select a 1RC model for the cell, change **prm_dyn** to **rc1** to model one RC dynamics.
3. Click **Apply** to apply the changes.

Try

Follow the steps to enable RC dynamics and thermal modeling options for the cell.



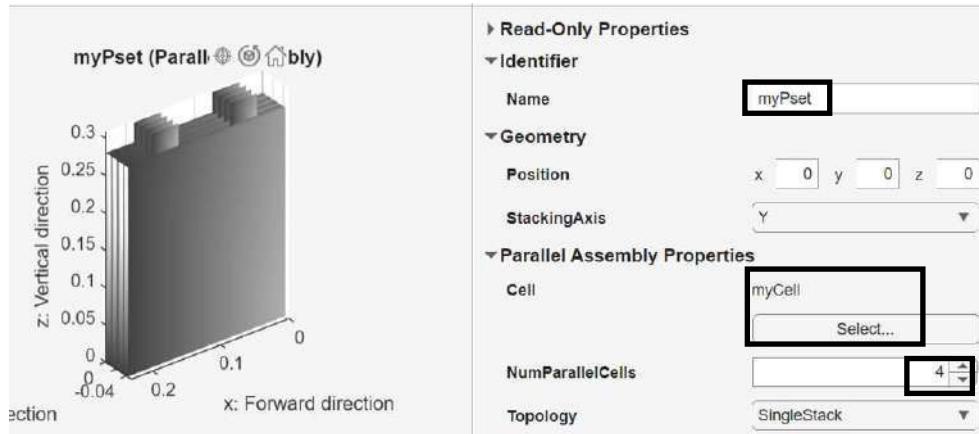
Main	
> Vector of state-of-charge values, SOC	[0, .1, .25, .5, .75, .9, 1]
Temperature dependent tables	Yes T_dependence
Current directionality prm_dir	Disabled
> Vector of temperatures, T	[278, 293, 313] K
> Open-circuit voltage, V0(SOC,T)	[3.49, 3.5, 3.51; ...] V
> Terminal voltage operating range [Min Max]	[0, inf] V
> Terminal resistance, R0(SOC,T)	[.0117, .0085, .0... Ohm
> Cell capacity, AH	27 A*hr
Self-discharge prm_leak	Disabled
Extrapolation method for all tables	Nearest
Expose SOC measurement port	No
Dynamics	
Charge dynamics prm_dyn	No dynamics
Fade	
Fade characteristic prm_fade	Disabled
Calendar Aging	
Internal resistance calendar aging	Disabled
Capacity calendar aging	Disabled
Thermal	
Thermal port thermal_port	Omit
> Simulation temperature	298.15 K

Battery Pack Modeling

Creating a Parallel Assembly Object

After creating the cell object, you connect four cells in parallel to create a parallel assembly.

1. Click on the **Parallel Assembly** button to create a new parallel assembly.
2. Change the **Name** of the parallel assembly to **myPset**.
3. Select **myCell** as the **Cell** for the parallel assembly (Note that a reselect is required if **myCell** is modified.)
4. Change the **NumParallelCells** to **4**.
5. Click **Apply** to apply the changes.



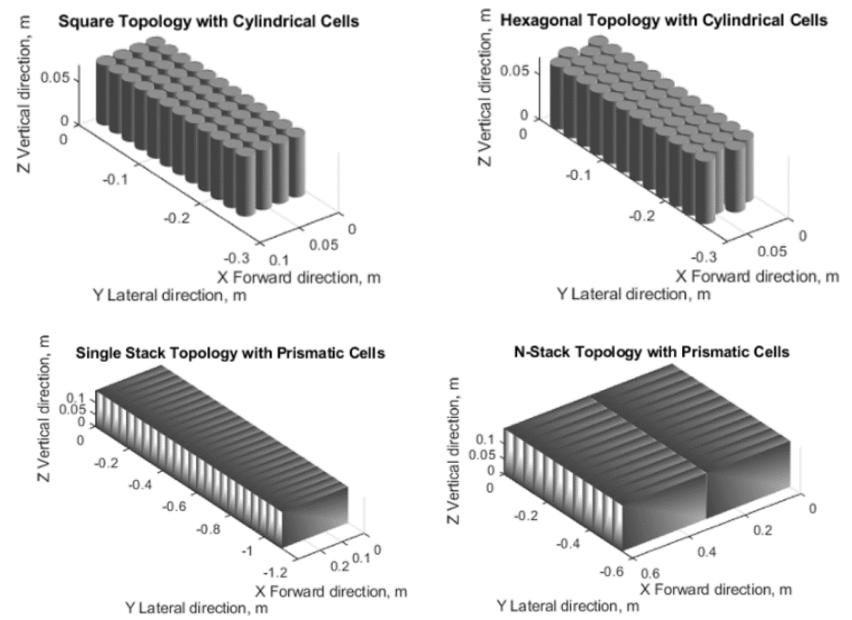
Geometry-wise, you can incorporate **InterCellGap** and **Topology** into your battery pack design. The **Topology** specifies the geometrical arrangement of cells relative to the cell format:

- **Pouch cell, prismatic cell:** Single Stack, N-Stack
- **Cylindrical cell:** Square, Hexagonal

Please refer to documentation **Simscape Battery > Battery Pack Modeling > ParallelAssembly** for more information.

Try

Follow the steps to create a parallel assembly with four cells connected in parallel.



With more than one cell, the **Battery Builder** automatically calculates the cumulative mass, volume, and capacity of the parallel assembly based on the given cell level information. You can find these values under **Read-Only Properties** of the parallel assembly.

Read-Only Properties					
Type	ParallelAssembly				
NumModels	1				
CumulativeMass	1.76 kg				
PackagingVolume	0.0030659 m³				
CumulativeCellCapacity	124 A·h				
CumulativeCellEnergy	458.8 W·h				
Layout	<table border="1"> <tr> <td>1</td><td>2</td><td>3</td><td>4</td></tr> </table>	1	2	3	4
1	2	3	4		

Battery Pack Modeling

Creating a Battery Module Object

Next, you create a module object that comprises multiple parallel assemblies in series.

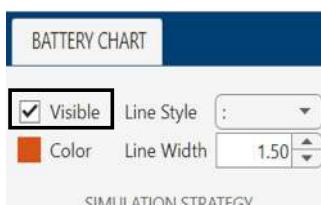
1. Click on the **Module** button to create a new module.
2. Change the **Name** of the parallel assembly to **myModLmp**.
3. Select **myPset** as the **ParallelAssembly** for the module.
4. Change the **NumSeriesAssemblies** to 12.
5. Click **Apply** to apply the changes.

Model Resolution

For parallel assemblies and modules containing multiple cells, you can set the **ModelResolution** to be either **Lumped**, **Grouped**, or **Detailed**. With **Detailed** resolution, each individual cell will be represented by a separate cell model (one set of equivalent circuit model.) While for a **Lumped** module, only one cell model will be implemented to represent the entire module. The parameters are scaled accordingly based on the number of cells in series(S) and in parallel(P).

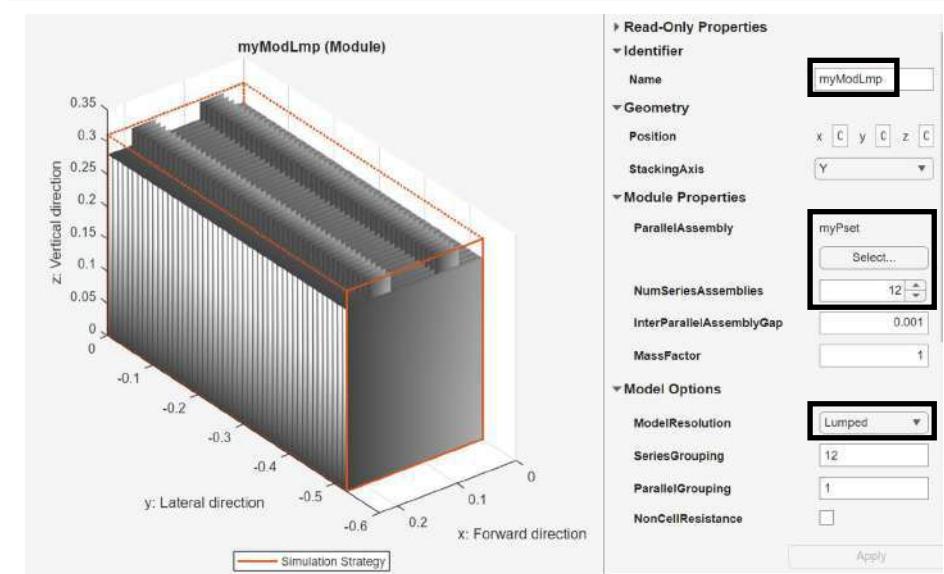
- Open Circuit Voltage: $V_{O_{Lmp}} = V_{O_{cell}} * S$
- Resistances: $R_{O_{Lmp}} = R_{O_{cell}} * S/P; R_{I_{Lmp}} = R_{I_{cell}} * S/P$
- Capacitance: $C_{I_{Lmp}} = C_{I_{cell}} * P/S$
- Capacity: $AH_{Lmp} = AH_{cell} * P$
- Thermal mass: $TM_{Lmp} = TM_{cell} * S * P$

You can visualize the model resolution by checking the **Visible** checkbox under **BATTERY CHART > SIMULATION STRATEGY**. Each orange box indicates one cell model block. When the Model Resolution is **Lumped**, only one orange box is wrapped around the entire model.



Try

Follow the steps to create a lumped module object.



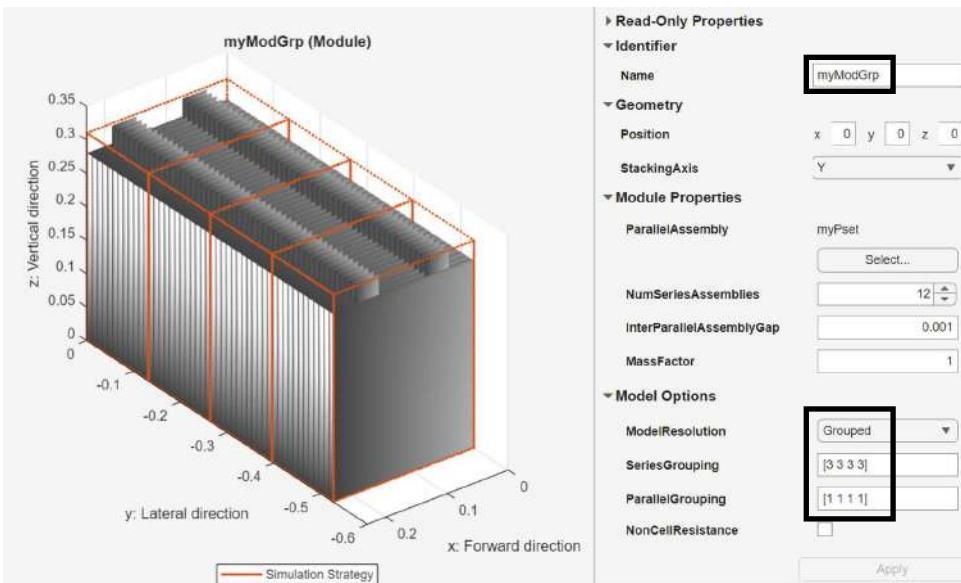
To balance simulation accuracy and speed, it is important to choose an appropriate model resolution. Desktop simulation can generally handle 60-80 cell models, whereas for hardware-in-the-loop (HIL), this number shrinks to around 30-40 cell models. To optimize your simulation, it is best to avoid creating models with over 100 cell models unless necessary.

Battery Pack Modeling

Creating a Battery Module Object (Continued)

To provide more flexibility when selecting model resolution, Simscape Battery allows users to define **Grouped** model resolution to combine custom number of cells in to one cell model. Follow the steps to divide the lumped module into four cell models, each with three parallel assemblies connected in series.

1. Duplicate `myModLmp` and rename the new module as `myModGrp`.
2. Change the **ModelResolution** of `myModGrp` to **Grouped**.
3. Change the **SeriesGrouping** to `[3 3 3 3]`.
4. Click **Apply** to apply the changes.



Try

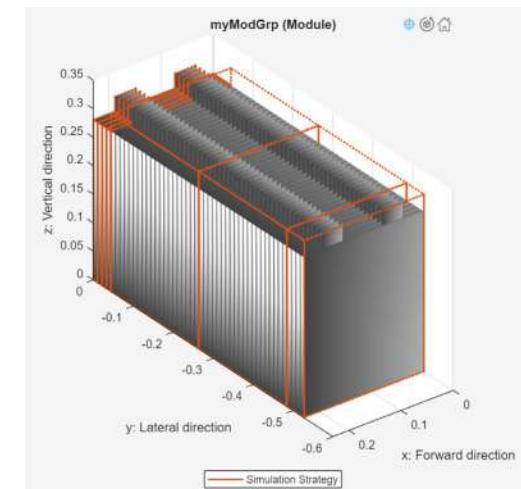
Follow the steps to create a module with grouped model resolution.

SeriesGrouping controls how you divide the series connections into multiple cell models. It is a vector whose sum of the elements must equal **NumSeriesAssemblies**.

ParallelGrouping controls the modeling strategy of parallel assemblies within the cell models defined by the **SeriesGrouping** property. **ParallelGrouping** should be a vector with the same length as **SeriesGrouping**. Each entry of the **ParallelGrouping** vector can be either 1 (lumped) or **NumParallelCells** (detailed.) Note that if the entry in **SeriesGrouping** is bigger than one, the **ParallelGrouping** can only be 1 (lumped). Here is an example of using **SeriesGrouping** and **ParallelGrouping** combined.

SeriesGrouping: `[1 5 5 1]`

ParallelGrouping: `[4 1 1 1]`



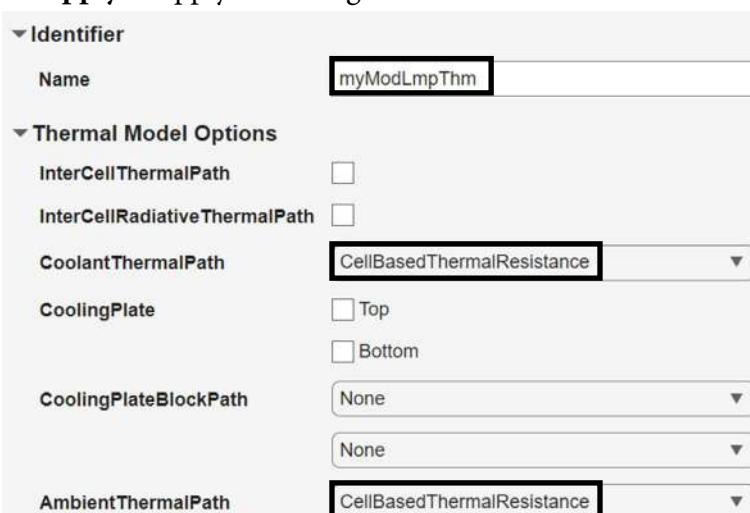
Battery Pack Modeling

Adding Thermal Model Options

Other than connecting cells in series and parallel electrically, Simscape Battery also supports you in constructing the thermal circuit along the way. Based on the selected model resolution, each cell model is represented by a thermal mass in 1D. Then to calculate the heat flow through the thermal mass, you can use the settings under **Thermal Model Options** to specify the thermal boundary conditions. These boundary conditions define the specific heat transfer mechanisms between the cell model and its surroundings, such as cell-to-ambient, cell-to-coolant, and cell-to-cell.

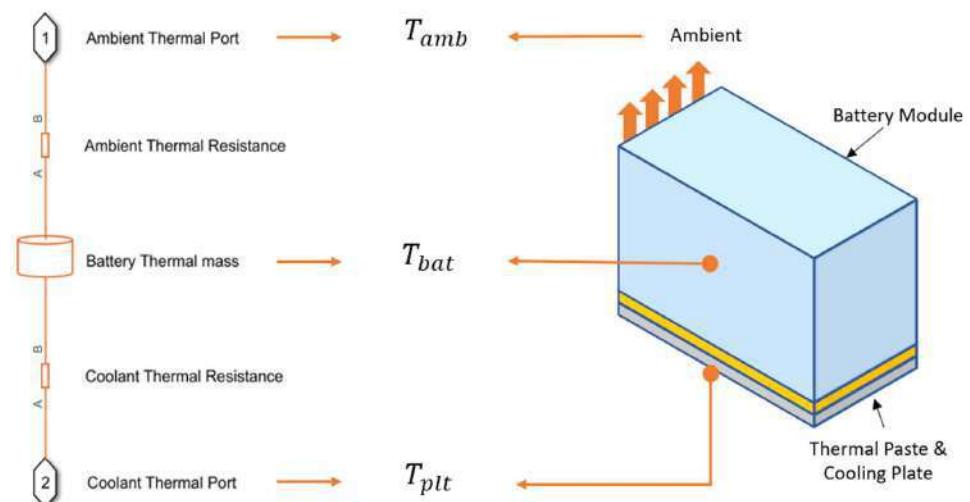
Consider the simplest case where the entire battery module is lumped as one cell model. Assume the module is exchanging heat with the ambient environment through convection while placed on a cooling plate. Simscape Battery allows you to incorporate thermal resistances to represent these heat exchanges. This is enabled by using **AmbientThermalPath** and **CoolantThermalPath**.

1. Duplicate **myModLmp** and rename it as **myModLmpThm**.
2. Change the **AmbientThermalPath** and **CoolantThermalPath** to **CellBasedThermalResistance**.
3. Click **Apply** to apply the changes.



Try

Follow the steps to add thermal model options to a lumped battery module.

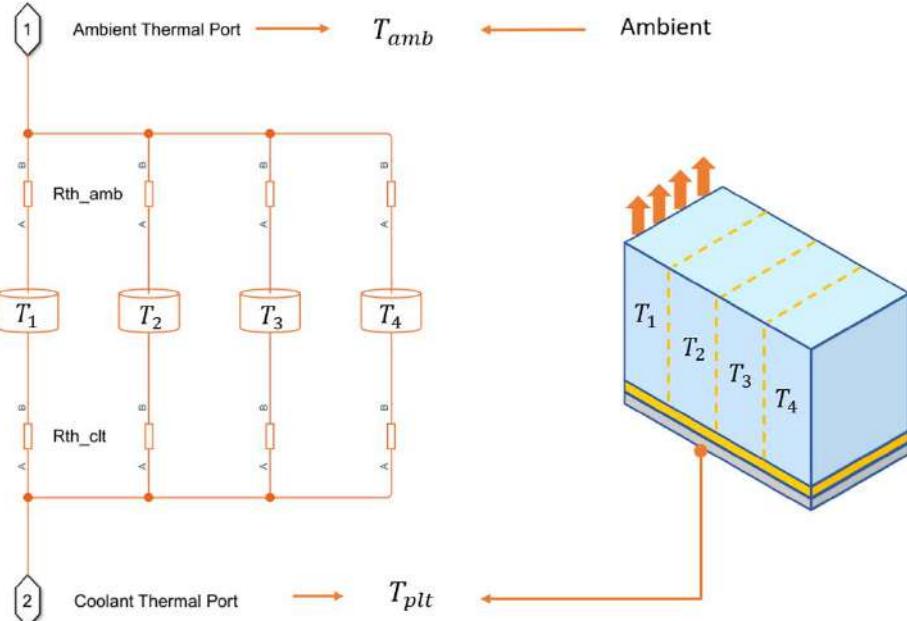


Battery Pack Modeling

Adding Thermal Model Options (Continued)

Lumped battery module utilize one thermal node to characterize the temperature of the entire module, thereby disregarding variations in temperature caused by locations. If **AmbientThermalPath** and **CoolantThermalPath** are enabled for a grouped module, thermal resistances are inserted between each cell model thermal mass and the thermal ports. You can vary the values of the thermal resistances to create a spread of temperature.

1. Duplicate **myModGrp** and rename it as **myModGrpThm**.
2. Change the **AmbientThermalPath** and **CoolantThermalPath** to **CellBasedThermalResistance**.

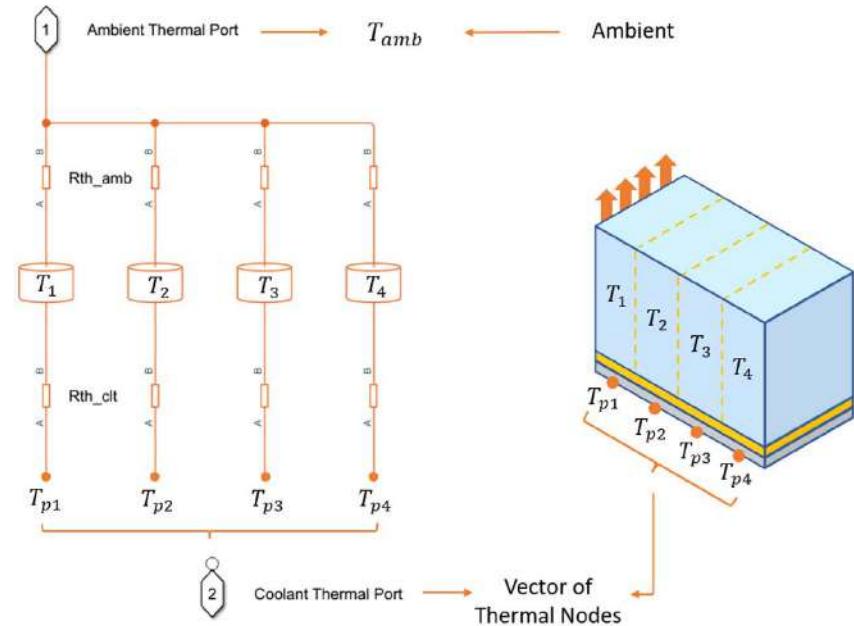


Try

Follow the steps to add thermal model options to a grouped battery module.

While a grouped battery module uses a separate thermal mass per cell model, by default it comes with a scalar thermal port, which suggests the exterior temperature is lumped as a single thermal node. Instead of using a single thermal node for the thermal boundary condition, you can enable the **CoolingPlate** option to output an array of thermal nodes.

3. Check the **Bottom** checkbox for option **CoolingPlate**.
4. Click **Apply** to apply the changes.



Battery Pack Modeling

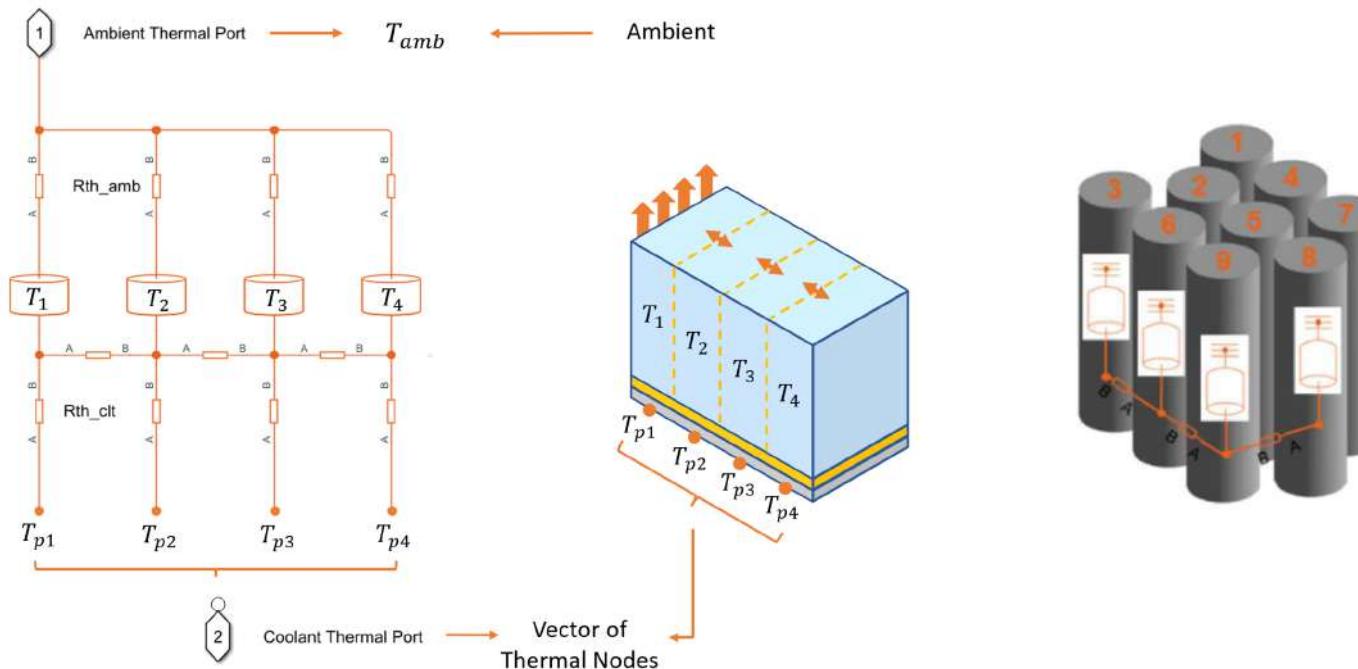
Adding Thermal Model Options (Continued)

Lastly, as the cells are sitting next to each other in the module, you should also consider including the heat transfer between cells.

1. Check the **InterCellThermalPath** checkbox.
2. Click **Apply** to apply the changes.



When **InterCellThermalPath** is enabled, Simscape Battery inserts thermal resistance between adjacent cell models. If you have more than one row of cells, the paths between rows are also included.

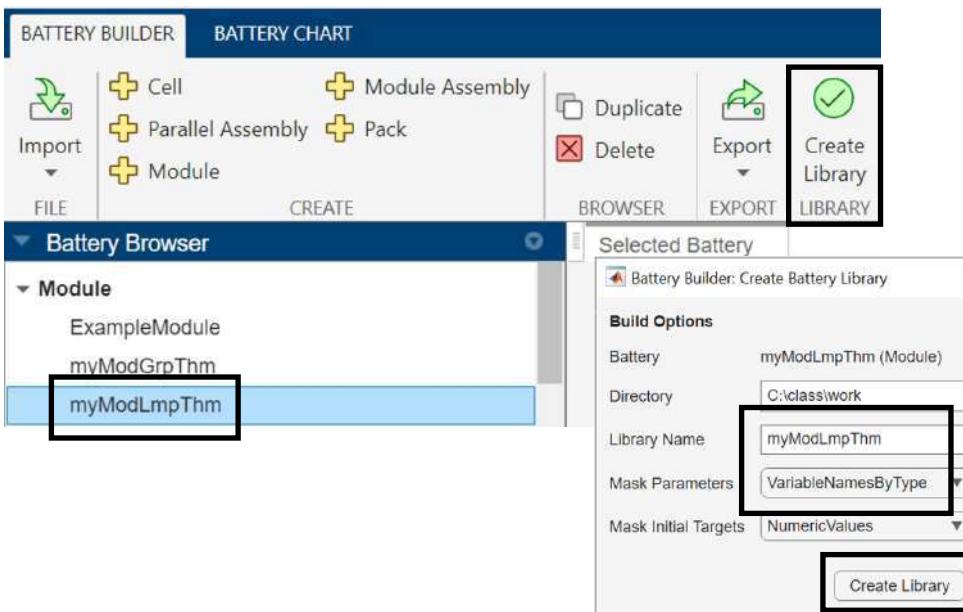


Battery Pack Modeling

Generating Simscape Battery Blocks

After creating the battery objects, you use the **Create Library** option to generate custom Simscape components corresponding to these objects.

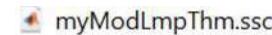
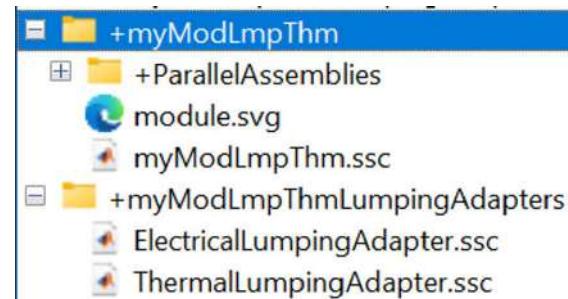
1. Select **myModLmpThm** from the **Battery Browser** and click on **Create Library**.
2. Change the **Library Name** to **myModLmpThm**.
3. Change **Mask Parameters** to **VariableNamesByType**.
4. Click on **Create Library** to generate the files.



During the build process, two folders are created within the build directory. These folders contain various **.SSC** files that define the components with the Simscape Language. The folder named **+LibraryName** includes the **.SSC** files for the module and its subcomponent parallel assembly. The selected cell model serves as the fundamental building block, and is then connected to electrical and thermal lumping adapters for scaling purposes. The lumping adapters are defined in the folder **+LibraryNameLumpingAdapters**.

Try

Follow the steps to create the Simscape Battery block for **myModLmpThm**. Examine the generated **.ssc** file.



```
components(ExternalAccess=observe)
myCell = batterycm.table_battery(SOC_vec = SOC_vecCell, ...
    T_vec = T_vecCell,V0_mat = V0_matCell,V_range = V_rangeCell,
    AH = AHCell,extrapolation_option = extrapolation_optionCell,
    thermal_mass = thermal_massCell,...
```



equations

```
CellNode.v == LumpedNode.v /CellsInSeries;
CurrentLumped == CurrentCell * (CellsInParallel-1);
end
```



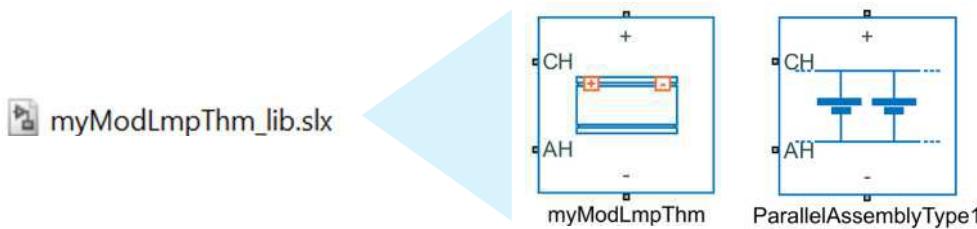
equations

```
CellNode.T == LumpedNode.T /CellsInSeries;
HeatFlowLumped == HeatFlowBattery * (CellsInParallel-1);
end
```

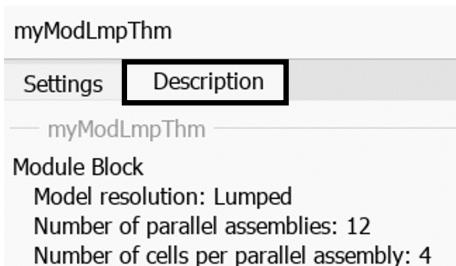
Battery Pack Modeling

Generating Simscape Battery Blocks (Continued)

The .ssc files are then transformed into Simscape blocks stored in `LibraryName_lib.slx`. This library comprises generated module and parallel assembly blocks.



The generated blocks have a similar **Block Parameters** dialog as the selected cell model. Information regarding the topology and model resolution are listed under the **Description** tab.



The content in the **Settings** tab varies based on the selected **Cell Model Options**. If **Mask Parameters** is set as **VariableNamesByType** during library creation, the parameter fields are automatically pre-filled with a data structure that bears the name of the battery object. This data structure is defined in `LibraryName_param.m`. You can use this .m file as a template and incorporate your own parameters for initialization.

Try

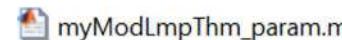
`>> myModLmpThm_lib`

Examine the generated blocks.

`>> edit myModLmpThm_param`

Examine the generated parameter initialization script.

myModLmpThm	
	Settings Description
NAME	VALUE
Main	
> Vector of state-of-charge values, SOC	myModLmpThm.SOC_vecCell
> Vector of temperatures, T	myModLmpThm.T_vecCell
> Open-circuit voltage, V0(SOC,T)	myModLmpThm.V0_matCell
> Terminal voltage operating range [Min Max]	myModLmpThm.V_rangeCell
> Terminal resistance, R0(SOC,T)	myModLmpThm.R0_matCell
> Cell capacity, AH	myModLmpThm.AHCell



`%% myModLmpThm`

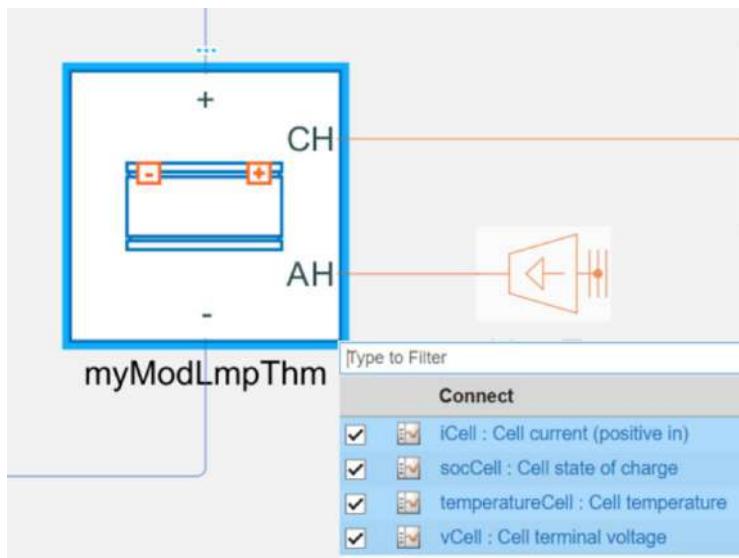
```
myModLmpThm.SOC_vecCell = [0, .1, .25, .5, .75, .9, 1]; % Vector
myModLmpThm.T_vecCell = [278, 293, 313]; % Vector of temperatures
myModLmpThm.V0_matCell = [3.49, 3.5, 3.51; 3.55, 3.57, 3.56; 3.62
myModLmpThm.V_rangeCell = [0, inf]; % Terminal voltage operating
myModLmpThm.R0_matCell = [.0117, .0085, .009; .011, .0085, .009];
```

Battery Pack Modeling

Logging Measurements From An Simscape Battery Block

Now you are ready to use the battery module in simulation.

1. Open the harness model `moduleLmp01_start.slx`.
2. Copy a `myModLmpThm` block from the library to the model.
3. Connect the + and the – terminals to the **Controlled Current Source**.
4. Double-click on the **Probe** block and select `myModLmpThm`. Check the checkboxes of `iCell`, `socCell`, `temperatureCell`, and `vCell` to expose the measurements.
5. Connect the **Probe** to the **Scope**.



Try

Follow the steps to add a battery module to the simulation harness.

```
>> moduleLmp01_start
```

Essentially, the custom battery module is a composition of multiple Simscape blocks, such as the **Battery (Table-Based) block** and the **Thermal Resistor**. Using a **Probe** block with a battery module allows you to access all Simscape nodes, variables, and intermediates within the composition.

You can find the list of measurements produced at a module level by checking the generated `.ssc` file. To prevent signal logging congestion caused by repeated values, the module block introduces variables with a **Cell** suffix (for example `vCell`) whose dimension correspond to the number of cell models. Alternatively, if your objective is to design a production BMS algorithm, where the signal dimension should mirror the actual number of measurements, the module block also includes variables with a **ParallelAssembly** suffix, whose dimensions match the number of parallel assemblies.

```
nodes
p = foundation.electrical.electrical; % +
n = foundation.electrical.electrical; % -
ClntH = foundation.thermal.thermal; % CH
AmbH = foundation.thermal.thermal; % AH
end

variables
iCell = {0,'A'}; % Cell current (positive in)
vCell = {0,'V'}; % Cell terminal voltage
socCell = {value={1,'1'},priority=priority.high}; % Cell state of charge
numCyclesCell = {value={0,'1'},priority=priority.high}; % Cell discharge cycles
temperatureCell = {value={298.15,'K'}},priority=priority.high}; % Cell temperature
vParallelAssembly = {value={repmat(0,12,1),'V'}},priority=priority.none}; % Parallel Assembly Voltage
socParallelAssembly = {value={repmat(1,12,1),'1'}},priority=priority.none}; % Parallel Assembly state of charge
end

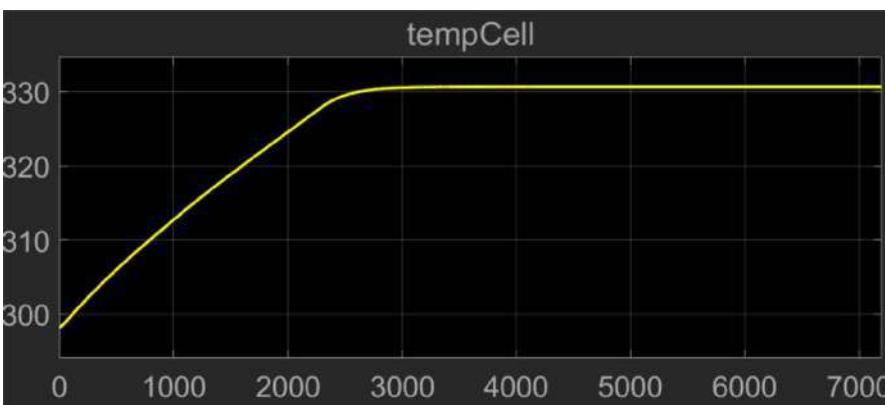
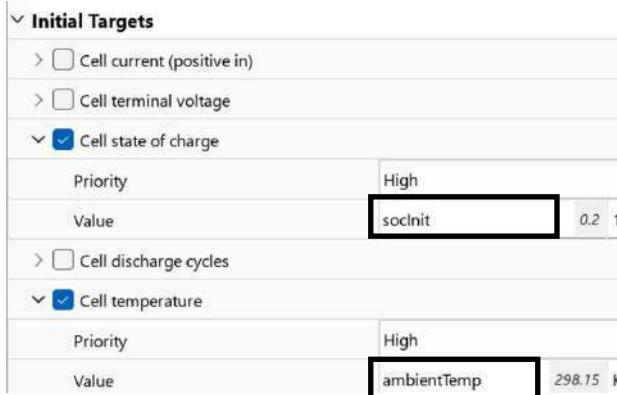
intermediates
power_dissipated = myCell.power_dissipated*((S*P)-1); % Power dissipated by (S*P)-1 cells
end
```

Battery Pack Modeling

Heat Generated By The Module

In this chapter, you analyze the battery and cooling performance under CCCV charging. The battery first goes under 1C fast charge until the terminal voltage hits 4.15V. Then, constant voltage charging is applied. The battery is assumed to be placed under an ambient environment with a constant temperature of 298.15K (25°C). After adding the module block, you will need to provide appropriate initial conditions for the simulation.

1. Go to the Block Parameters of myModLmpThm. Under **Initial Targets**, set **Cell state of charge** to **socInit** and set **Cell temperature** to **ambientTemp**.
2. Simulate the model to examine the heat generated by the cell.



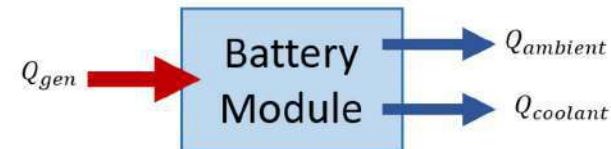
Try

Follow the steps to set the initial condition of the module. Simulate the model and examine the heat generated during fast charging.

As introduced in the cell modeling chapter, due to the internal resistance of the battery, heat is generated when current flows through the cell. The generated heat flow rate (q_{gen}) is determined by the summation of all the ohmic losses included in the cell model.

$$c_p m \dot{T} = q_{gen} = \sum_i \frac{V_{T,i}^2}{R_{T,i}}$$

When there is no heat exchange, all generated heat will be converted to a temperature rise of the battery, with a ratio that depend on the mass of the cell (m) and the specific heat capacity (c_p) of the material. Given the specific heat capacity of 964 J/kgK, the temperature rise during charging is around 31K, brining the final temperature to 330K (57°C). Next, you model the heat exchange through the ambient thermal path and the coolant thermal path to bring the temperature down to a normal operating range. The overall energy flow is represented as shown below.



In this course, the following symbols are used to denote heat transfer related terms.

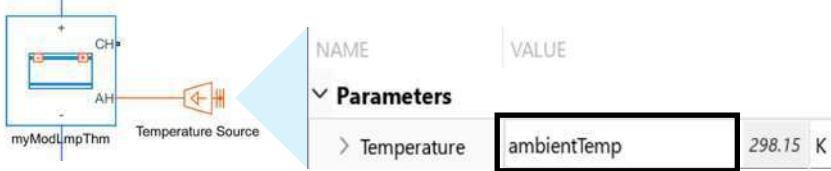
- Q : Energy transfer, J
- q : Heat transfer rate, W
- q' : Heat transfer rate per unit length W/m
- q'' : Heat flux (heat transfer rate per unit area), W/m²

Battery Pack Modeling

Adding Ambient Thermal Path

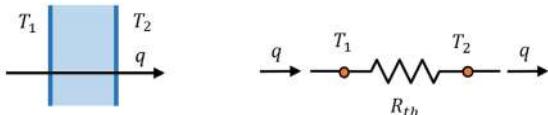
You first model the heat exchange between the module and the ambient environment.

1. Add a **Temperature Source** block from the **Simscape > Foundation Library > Thermal > Thermal Sources** library into the model. Set the **Temperature** to **ambientTemp**.
2. Connect the **Temperature source** to the **AH** port and simulate the model.



Thermal resistance

Depending on how the battery module is placed on the equipment, modeling the heat exchange between the battery and the ambient environment can be complicated. In Simscape Battery, you will use a **Thermal Resistance** to represent the overall heat transfer resistance between the ambient environment and the battery module.



Thermal resistance is a heat property that measures how effectively a component resists the flow of heat. Thermal resistance has a unit of K/W. It is defined as the ratio of temperature change per Watt of heat flow going through a specific object.

$$R_{th} = \frac{\Delta T}{q}$$

- R_{th} : Thermal resistance (K/W)
- ΔT : Temperature difference across the object (K)
- q : Rate of heat flow (W)

Try

Follow the steps to add an constant temperature ambient environment to the model. Simulate the model.

With the given equation, a known rate of heat flow is required to calculate the value of the thermal resistance. You can apply Fourier's law and Newton's law to calculate the rate of heat transfer for conduction and convection respectively. Both state that the rate of heat transfer is proportional to the temperature difference.

- Fourier's law of conduction in 1D

- Heat flux:

$$q''_x = k \frac{\Delta T}{\Delta x} = k \frac{T_1 - T_2}{L}$$

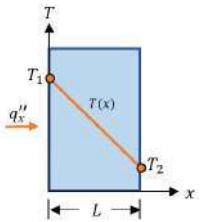
- Thermal resistance:

$$R_{cond} = \frac{\Delta T}{q_x} = \frac{L}{kA}$$

- Heat transfer rate:

$$q_x = q''_x A = \frac{kA}{L} \Delta T$$

- k : Thermal conductivity (W/m·K)
 A : Area of the plane (m^2)



- Newton's law of cooling in 1D

- Heat flux:

$$q'' = h(T_s - T_\infty) = h\Delta T$$

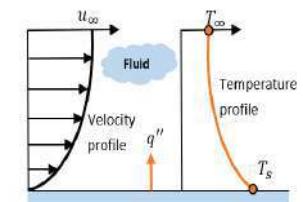
- Thermal resistance:

$$R_{conv} = \frac{\Delta T}{q} = \frac{1}{hA}$$

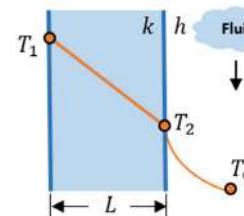
- Heat transfer rate:

$$q = q'' A = hA\Delta T$$

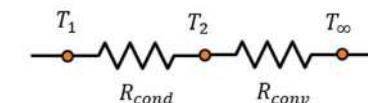
- h : Convection heat transfer coefficient (W/m²·K)
 A : Area of the plane (m^2)



Thermal resistance is analogous to electrical resistance. You can combine thermal resistances in series and in parallel to calculate the overall heat transfer if multiple layers of materials are involved.



$$R_{th} = R_{cond} + R_{conv}$$



Battery Pack Modeling

Adding Ambient Thermal Path (Continued)

Considering the overall thermal resistance of the ambient thermal path, the resistance value of each layer is heavily dependent on material properties, geometry, and interface conditions, making it challenging to be calculated accurately. Typically, this value is determined directly through experimental measurements. Below is a reference experiment set up to determine the thermal resistance.

1. Place the battery in a temperature chamber with an ambient temperature of 298.15K (25°C). Soak the battery for long enough time to make sure the battery temperature is the same as the ambient.
2. Discharge the battery with 1C until the battery temperature rises to 318.15K (45°C), stop the discharge.
3. Record how the temperature changes with time. Stop the experiment when the battery temperature drops back to ambient temperature again.
4. Calculate the thermal resistance from the cooling time constant.

In the experiment, all the heat generated by the battery is dissipated through the ambient thermal resistance. Solving the energy balance you can obtain the equation of battery temperature change with time.

- Assume all heat generated is dissipated through R:

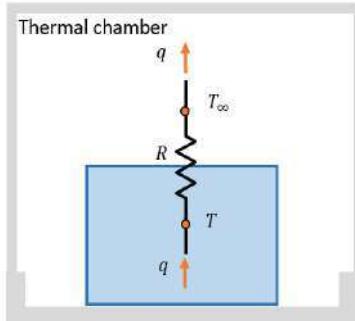
$$mc_p \frac{dT}{dt} = q = \frac{T - T_\infty}{R} \Rightarrow \frac{dT}{dt} = \frac{1}{mc_p R} (T - T_\infty)$$

- Integrate and get the analytical solution:

$$T(t) = (T_0 - T_\infty)e^{-t/\tau} + T_\infty$$

Where

- $\tau = mc_p R$ Thermal time constant
- $T_0 = T(t=0)$ Initial condition

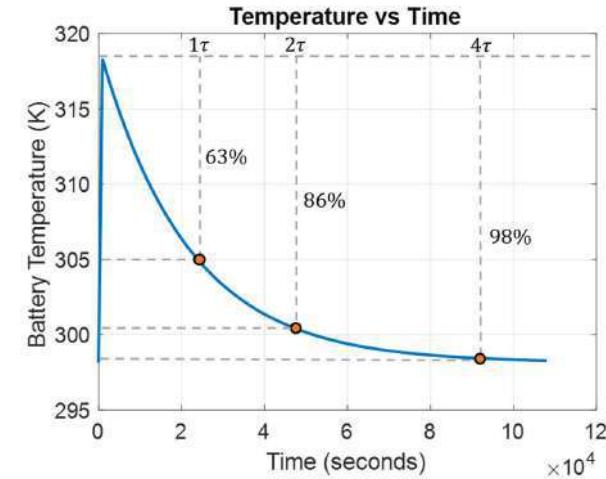


Try

`>> edit thermalResCal`

Run the script to calculate the thermal resistance.

By recording the temperature data, you can obtain the time constant from the cooling curve as shown below. Then, the thermal resistance can be calculated as $R = \tau / mc_p$.



Alternatively, you can use the **Parameter Estimator** app as introduced in chapter *cell characterization* for similar purpose. Here are some reference ambient thermal path resistances for cells with different geometries.

Cell Type	Cylindrical	Pouch	Prismatic
Capacity (Ah)	5	50	150
Thermal Resistance (K/W)	2000	250	20

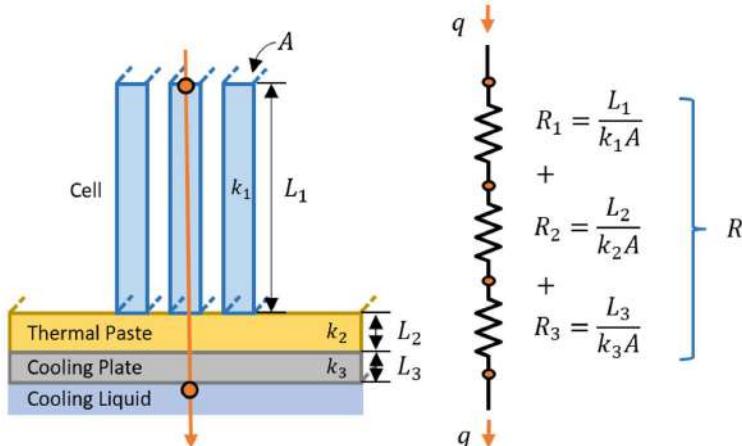
Battery Pack Modeling

Adding Coolant Thermal Path

Similar to the ambient thermal path, in Simscape Battery, the heat transfer between the battery and the coolant is also represented by a thermal resistance. Here are some reference coolant thermal path resistances accounting for the resistance between the cell and the coolant.

Cell Type	Cylindrical	Pouch	Prismatic
Capacity (Ah)	5	50	150
Thermal Resistance (K/W)	15-25	1-5	0.1-0.5

The heat transfer path to the cooling plate is relatively straightforward. Instead of experiment, you can also directly calculate the combined thermal resistance. Consider a simplified scenario where a battery is positioned on a layer of thermal paste and then attached to a cooling plate. The temperature sensor is placed on top of the cell and the coolant temperature is measured. The diagram represents the entire thermal circuit. By merging the thermal resistances of each layer, the coolant thermal path resistance can be calculated.



Try

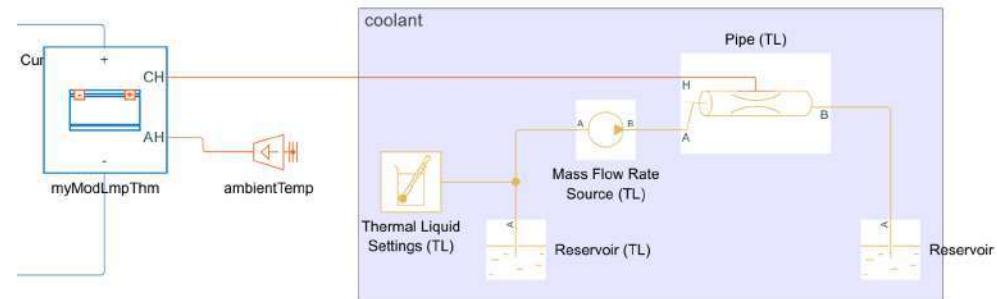
Uncomment the coolant subsystem and connect it to the CH port of the battery module. Simulate the model.

Or,

```
>> moduleLmp02_coolant
```

Modeling the coolant

While the coolant temperature can be represented as simple as a Temperature Source, in this example, you will be investigating what mass flow rate is required to lower the temperature to an acceptable level. The Simscape **Foundation Library** provides blocks for modeling the flow of the coolant and its heat exchange, which fall under the **Thermal Liquid** domain. Follow the Try box to connect the module to the **coolant** subsystem modeled with thermal liquid blocks.

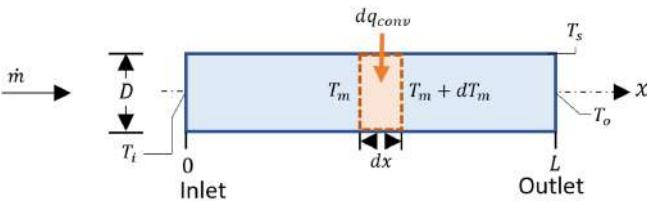


The Pipe (TL) block represents a pipeline segment with a fixed volume of liquid. The liquid experiences heat transfer due to convection between the fluid and the pipe wall. Here we are considering a pipe segment with the same length as the battery module. The liquid in the pipe is driven by a constant mass flow rate determined by the Mass Flow Rate Source (TL) block. To set the boundary conditions in the thermal liquid network, both sides of the pipe are connected to Reservoir (TL) blocks, that set the pressure and temperature of liquid assumed with infinite volume. The cross-sectional areas of all the blocks are set as the same.

Battery Pack Modeling

Pipe Internal Flow Heat Transfer Fundamentals

Consider a circular pipe with length L and diameter D. Water is flowing through the pipe with a constant mass flow rate \dot{m} . The flow is assumed fully developed which indicates the temperature distribution across the flow remains constant. T_m to denote the mean temperature over the cross-section of the tube. The convective heat transfer between the pipe wall and the fluid causes T_m to change continuously with the pipe axis.



Assume the surface temperature of the pipe is held at constant T_s , and the temperature difference between T_s and T_m decays exponentially along the pipe axis. This relationship can be derived by applying energy balance to a control volume along the tube as shown below.

- Apply energy balance to the control volume:

$$dq_{conv} = \dot{m}c_p dT_m = q_s P dx \quad P = \pi D$$

$$\frac{dT_m}{dx} = \frac{q_s P}{\dot{m}c_p} = \frac{P}{\dot{m}c_p} h(T_s - T_m)$$

- Define $\Delta T = T_s - T_m$, integrate over the pipe length:

$$\ln \frac{\Delta T_o}{\Delta T_i} = \frac{PL}{\dot{m}c_p} h \quad \frac{\Delta T_o}{\Delta T_i} = \frac{T_s - T_o}{T_s - T_i} = \exp \left(-\frac{PL}{\dot{m}c_p} h \right)$$

Then, the overall heat transfer over the pipe can be written as:

$$q_{conv} = \dot{m}c_p (\Delta T_i - \Delta T_o)$$

$$q_{conv} = \dot{m}c_p \Delta T_i \left(1 - \exp \left(-\frac{PL}{\dot{m}c_p} h \right) \right)$$

To calculate the heat transfer, you must calculate the convective heat transfer coefficient h . Under fully developed conditions, h can be considered as a constant. The Pipe (TL) block calculates h from the Nusselt number Nu (ratio of convective and conductive heat transfer).

$$Nu_D = \frac{\text{convective heat transfer}}{\text{conductive heat transfer}} = \frac{hD}{k}$$

k : Liquid thermal conductivity (W/m·K)

The value of the Nusselt number can be obtained theoretically or empirically depending on the flow regime. For laminar flow in a circular pipe, Nu is a constant since the flow is relatively uniform. In turbulent flow, Nu can be calculated from the empirical Gnielinski correlation as shown below.

$$Nu_D = 3.66 \quad \text{Laminar flow}$$

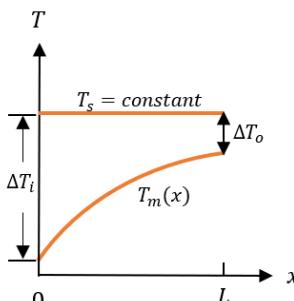
$$Nu_D = \frac{(f/8)(Re - 1000)Pr}{1 + 12.7(f/8)^{1/2}(Pr^{2/3} - 1)} \quad \text{Turbulent flow}$$

Pr : Liquid Prandtl number

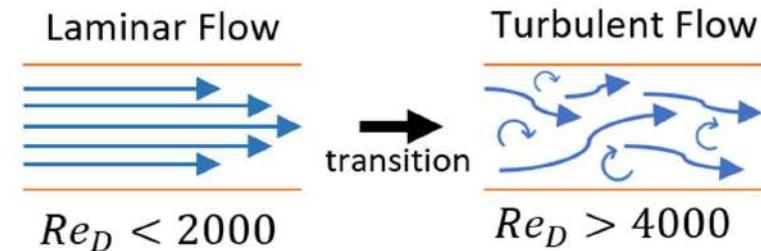
f : Darcy friction factor

The flow regime of a fluid is governed by the dimensionless Reynolds number Re . It is a ratio between inertial force and viscous force.

$$Re_D = \frac{\text{inertia forces}}{\text{viscous forces}} = \frac{uL}{\nu} = \frac{\rho u L}{\mu} = \frac{\dot{m}D}{A\mu} \quad \begin{array}{l} \rho : \text{fluid density} \quad A : \text{cross-section area} \\ u : \text{fluid velocity} \\ \nu : \text{kinematic viscosity} \\ \mu : \text{dynamic viscosity} \end{array}$$



A small Reynolds number indicates a laminar flow while a big Reynolds number indicates a turbulent flow. The chart below indicates the correlation between the flow regime to the Reynolds number.



Battery Pack Modeling

Adding Coolant Thermal Path (Continued)

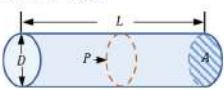
When using the Pipe (TL) block, you can specify the pipe **Geometry** in terms of the **Pipe length**, **Cross-sectional area**, and **Hydraulic diameter**. Shape plays an important factor when it comes to fluid calculation. As most of the formulas are derived based on circular geometry, if you are working with a different geometry, you must calculate the equivalent hydraulic diameter that transforms the non-circular shape into its circular representative. Hydraulic diameter is defined as four times the cross-sectional area divided by the wetted perimeter.

- Square duct:



$$D_h = \frac{4a^2}{4a} = a$$

- Circular tube:



$$D_h = \frac{4\pi D^2 / 4}{\pi D} = D$$

- Rectangular duct:



$$D_h = \frac{4ab}{2(a+b)} = \frac{2ab}{a+b}$$

$$\text{Hydraulic Diameter: } D_h = \frac{4A}{P}$$

Getting the correct hydraulic diameter is important as it impacts the calculation of the Reynolds number and the Nusselt number. Meanwhile, the shape factor also impacts your **Friction and Heat Transfer** settings. The default block parameters are based on a circular pipe. For laminar flow in a non-circular pipe, you must find the corresponding Darcy friction factor and Nusselt number from the table below. For turbulent flow, the equation mentioned on the previous page still applies.

Cross Section	b / a	Nu_D (Uniform T_s)	f
	-	3.66	64
	1.0	2.98	57
	2.0	3.39	62
	4.0	4.44	73

Try

Examine the parameters of the Pipe (TL) block.

Pipe (TL)	
NAME	VALUE
Geometry	
> Pipe length	moduleLength 0.23 m
> Cross-sectional area	crossArea 2e-05 m^2
> Hydraulic diameter	D 0.005 m
Friction and Heat Transfer	
> Aggregate equivalent length of local resistances	1 m
> Internal surface absolute roughness	15e-6 m
> Laminar flow upper Reynolds number limit	2000
> Turbulent flow lower Reynolds number limit	4000
> Laminar friction constant for Darcy friction factor	64
> Nusselt number for laminar flow heat transfer	3.66
Effects and Initial Conditions	
Fluid dynamic compressibility	Off
> Initial liquid temperature	AmbientTemp 303.15 K

Additionally, you can specify the type of liquid using a Thermal Liquid Settings (TL) block. This block includes parameters regarding liquid density, thermal conductivity, and kinematic viscosity, which are used in the pipe calculation. The default block parameters are based on water. You will need to update the values in the table if you are using other cooling liquids. Or alternatively, use the Thermal Liquid Properties (TL) block (with pre-defined tables of common liquids) from the Simscape Fluids library.



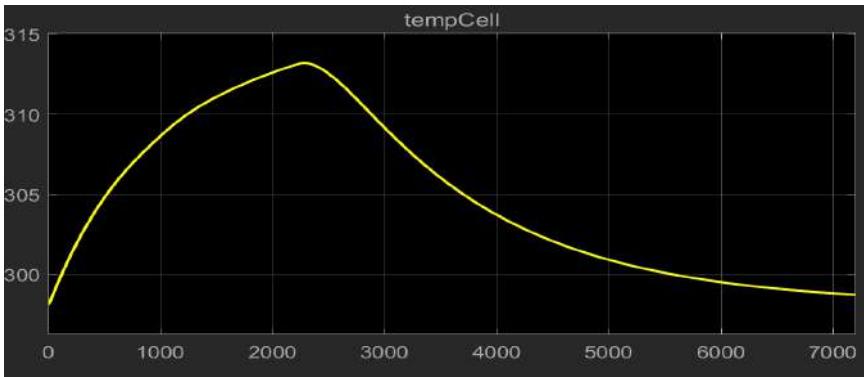
Thermal Liquid Settings (TL)

Internal Energy	
Internal energy parameterization	Specific heat vector - cp(T)
> Specific heat at constant pressure vector, c...	[4.2199; 4.1955; 4.1843; 4.180... kJ/(K*kg)
Viscosity and Conductivity	
> Kinematic viscosity vector, nu(T)	[1.7917; 1.3061; 1.0032; .8006... mm^2/s
> Thermal conductivity vector, k(T)	[561.04; 580.02; 598.44; 615.4... mW/(K*m)

Battery Pack Modeling

Simulating Battery Module with Grouped Fidelity

After simulating the lumped model, you observe that cooling brings the highest temperature down to 313K (40°C) and the end-of-charge temperature is around 298K (25°C). However, the temperature difference with respect to cell model locations is not captured.



Generate a module with grouped fidelity

To improve the simulation fidelity you use a grouped cell model to examine the temperature variance based on cell locations.

1. In the **Battery Builder** app, select **myModGrpThm** and **Create Library**. Name the library as **myModGrpThm** and change **Mask Parameters** to **VariableNamesByType**.
2. Open the harness model **moduleGrp01_start.slx**, and copy a **myModGrpThm** block from the library to the model and connect it with the existing circuit.
3. Go to the **Block Parameters** of **myGrpThm**. Under **Initial Targets**, set **Cell state of charge** to **repmat(socInit,4,1)** and set **Cell temperature** to **repmat(ambientTemp,4,1)**.

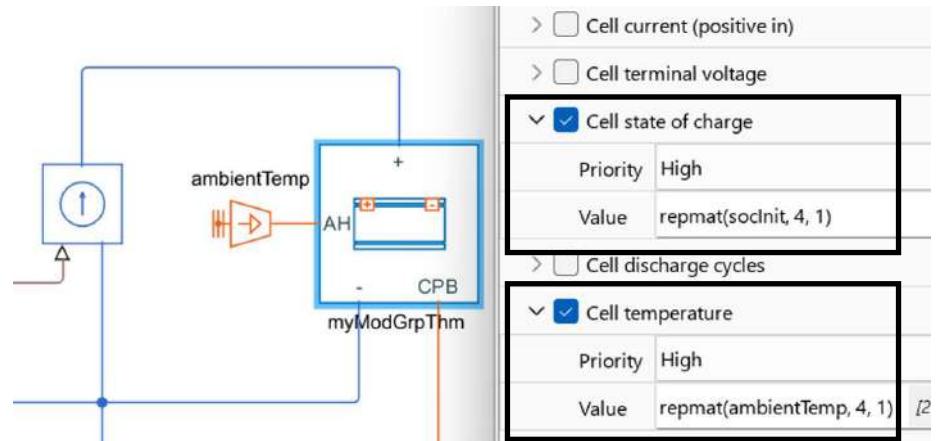
Note that, with grouped model resolution, you can provide different initial target values to each individual cell model. Function **repmat(A,m,n)** returns a matrix of copies of **A** with dimension **mxn**.

Try

```
>> moduleGrp01_start
```

Follow the steps to add a grouped module into the model.

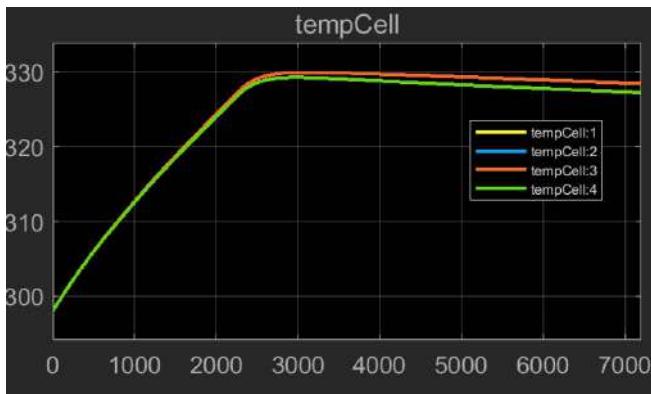
4. Setup the Probe block by exposing **iCell**, **socCell**, **temperatureCell**, and **vCell** measurements. Connect the Probe to the scope.
5. Simulate the model.



Battery Pack Modeling

Simulating Battery Module with Grouped Fidelity (Continued)

To address the temperature variance caused by location differences, you can provide different ambient thermal path resistances and coolant thermal path resistances. For example, the ambient thermal resistances of cells in the middle of a module should be bigger than the ones on the edge. Below is the simulation result with ambient thermal resistance of [100 400 400 100]. The maximum temperature difference is 1.1K.



You can use a **BatterySimulationChart** to visualize battery states from simulation as an overlay to the battery chart. Simscape simulation log (simlog) must be enabled to record the simulation data. You can turn on the setting from the command line by running

```
>> set_param(modelname,"SimscapeLogType",'all');
```

Then, create a **BatterySimulationLog** object to link the simlog with the battery object.

```
>> batterySimLog = ...
    BatterySimulationLog(batteryObj,simLog);
>> batterySimLog.SelectedVariable = "temperatureCell";
```

Try

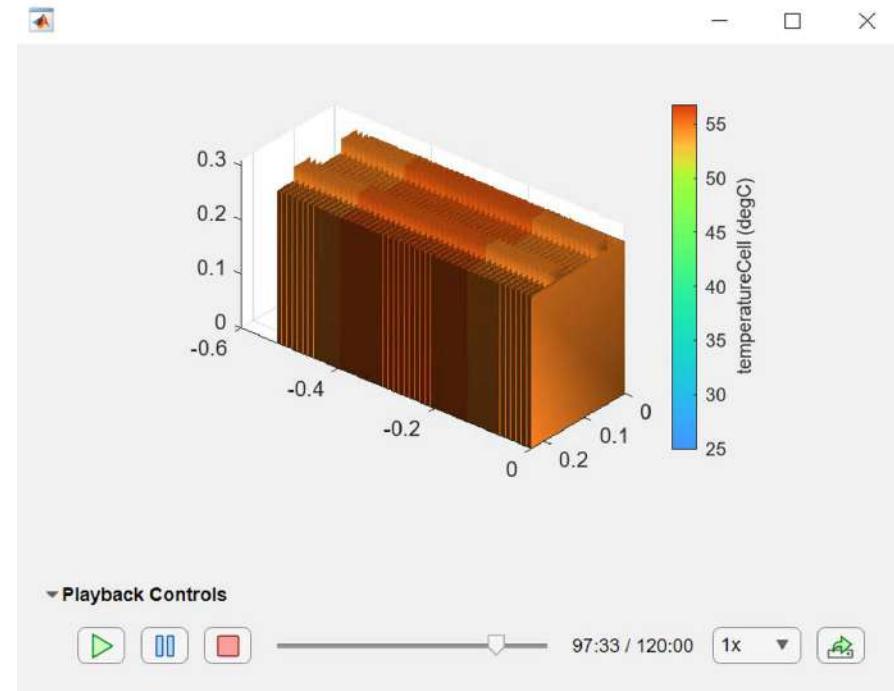
```
>> temperaturePlot_module
```

Run the script to generate a temperature plot.

Lastly, you will create a **BatterySimulationChart** to visualize the data.

```
>> moduleChart = BatterySimulationChart(..., ...
    Parent =figure, ...
    BatterySimulationLog = batterySimLog);
>> moduleChartColorBar = colorbar(moduleChart);
```

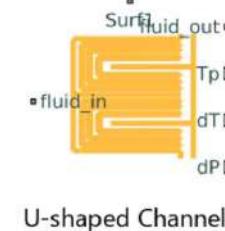
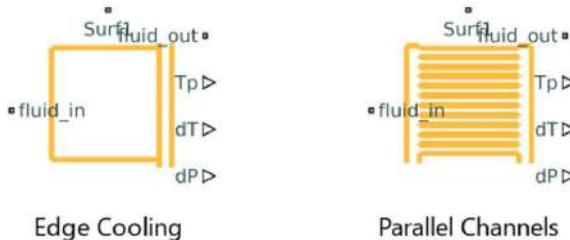
You can use the **Playback Controls** to replay the temperature change in the battery module and export a video if needed. The visualization indicates the cells in the middle have higher temperatures than the cell on the edge.



Battery Pack Modeling

Edge Cooling

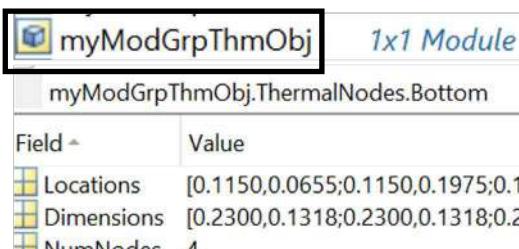
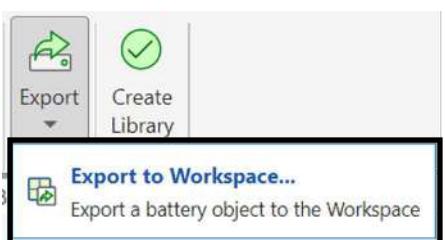
In addition to enhancing the simulation fidelity of the module, increasing the cooling plate resolution is also possible. Simscape Battery comes with a set of cooling system blocks that can be used to interface with the battery module. You will explore the simplest case edge cooling first and explore other cooling methods in later sections.



1. Add an Edge Cooling block from the **Simscape > Battery > Thermal** library. Connected the block with the existing circuit.

To accommodate the varying sizes of battery modules, the dimensions of the cooling plate should be determined based on specific module objects. When creating a battery object, Simscape Battery automatically calculates the **ThermalNodes** property that stores the dimensions and locations of the cell models. This information can then be passed to a cooling plate block for configuration.

2. In the **Battery Builder** app, export **myModGrpThm** into the base workspace, and rename the object as **myModGrpThmObj**.
3. Double-click on **myModGrpThmObj**, and examine the thermal nodes stored under **myModGrpThmObj .ThermalNodes .Bottom**.



Try

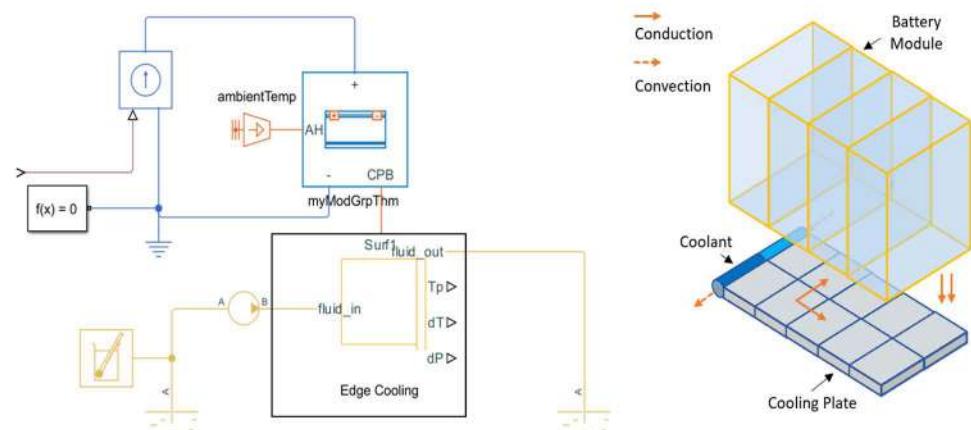
```
>> moduleGrp02_edgeCool
```

Follow the steps to add an Edge Cooling plate to the model.

4. Open the **Block Parameters** of the Edge Cooling block. Set up the **Interface** settings by using the information stored under **myModGrpThmObj**.

Interface

Battery Connectivity	Single sided
Number of battery thermal nodes (sur...	myModGrpThmObj.ThermalNodes.Bottom.NumNodes
> Dimension of battery thermal nodes (s...	myModGrpThmObj.ThermalNodes.Bottom.Dimensions
> Coordinates of battery thermal nodes ...	myModGrpThmObj.ThermalNodes.Bottom.Locations



Battery Pack Modeling

Edge Cooling (Continued)

Just like how Simscape Battery generates custom battery modules, the cooling plate blocks also employ built-in Simscape blocks to construct the thermal circuit. After determining the size of the cooling plate, it is subsequently divided into segments based on the number of partitions in X and Y dimensions. Each individual plate segment will be represented by a Thermal Mass.

Number of partitions in X dimension for the cooling plate	2
Number of partitions in Y dimension for the cooling plate	5

The heat transfer of a plate segment can be summarized into three components.

- **Conduction heat transfer between the plate segments**

In between the plate segments, the Conductive Heat Transfer blocks are in place to model the heat transfer between the plate segments. You can configure the cooling plate details under the **Plate Material** section. Follow the figure to set the parameter of the plate.

Plate Material

> Thickness of cooling plate material	2e-3	m
> Thermal conductivity of cooling plate mat...	154	W/(K*m)
> Density of cooling plate material	2730	kg/m^3
> Specific heat of cooling plate material	893	J/(K*kg)
> Initial temperature of the cooling plate an...	ambientTemp	298.15 K

- **Convective heat transfer to the coolant**

In the edge cooling configuration, the coolant flows at one end of the flat plate. Similar to the previous example, the modeling of the coolant is handled by the Pipe (TL) block. You can attach the pipes to the edge determined by **Select edge to be cooled**.

Try

```
>> moduleGrp02_edgeCool
```

Update the Edge Cooling block parameters as shown in the figures.

Design

Select edge to be cooled

Edge cooling along X axis at Y = Ymin

> Cooling channel hydraulic diameter

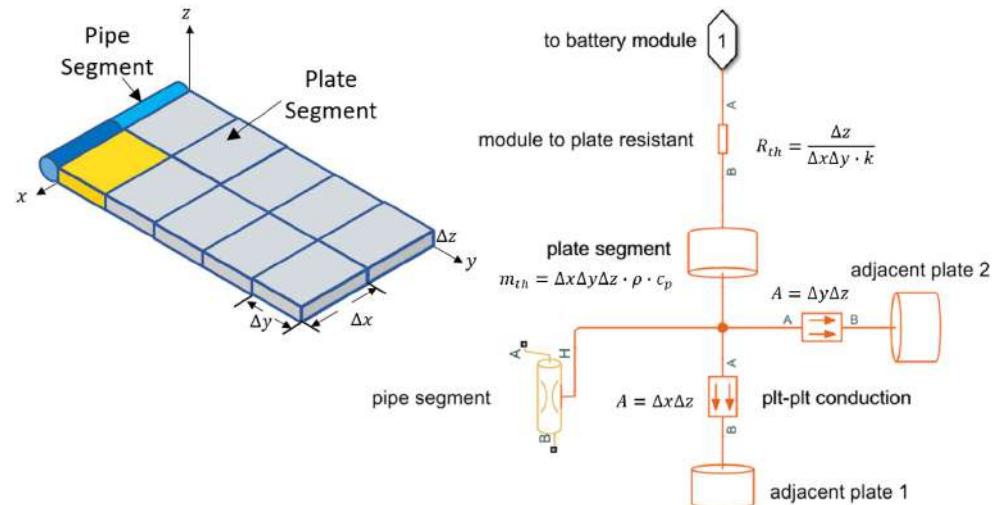
5e-3 m

> Cooling channel cross-sectional area

2e-5 m^2

> Coolant channel roughness

1.5e-05 m



- **Conduction heat transfer between the battery and the plate**

The conduction heat transfer between the battery and the plate segment is modeled by Thermal Resistance. Detailed implementation is listed on the next page.

Battery Pack Modeling

Edge Cooling (Continued)

In the implementation of battery to plate heat transfer, all cell models are assumed to be connected to all cooling plate segments. Then, the calculated heat flow is multiplied by a contact ratio between zero and one. The contact ratio is calculated based on the actual contact area of the cell model and the cooling plate segment. To illustrate, the figure below shows the heat flow between cell model k and plate segment i,j. The red number on the plate indicates the contact ratio. Eventually the total heat flow of a specific cell model or a cooling plate can be calculated by summing up related heat flow components.

- Heat flow between cell model k and plate segment i,j :

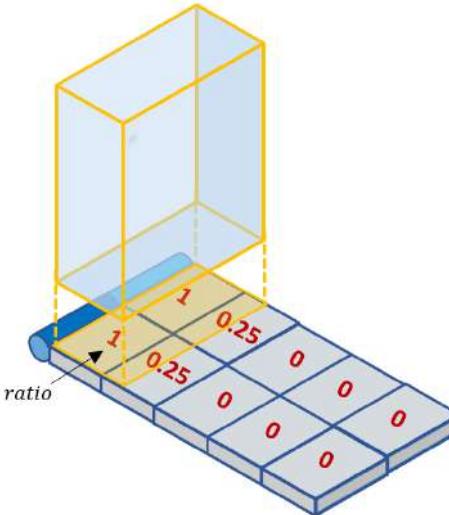
$$q_{bat(k),plt(i,j)} = \frac{T_{plt(i,j)} - T_{bat(k)}}{R_{th}} \cdot ratio$$

- Overall heat flow of plant segment i,j :

$$q_{plt(i,j)} = \sum_{k=1}^w \frac{T_{plt(i,j)} - T_{bat(k)}}{R_{th}} \cdot ratio$$

- Overall heat flow of cell model k :

$$q_{bat(k)} = \sum_{i=1}^m \sum_{j=1}^n \frac{T_{plt(i,j)} - T_{bat(k)}}{R_{th}} \cdot ratio$$



where

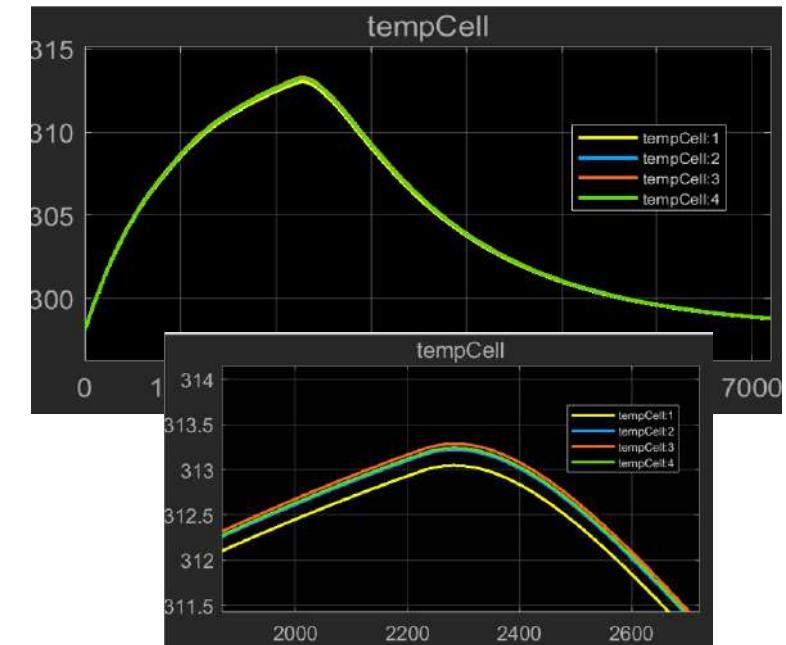
- w : number of cell models
- m,n : number of plat segemnts in x and y directions

Try

```
>> moduleGrp03_edgeCool
```

Simulate the model to explore the temperature difference between the cell models.

Simulate the model with edge cooling and zoom in to view the temperature differences of the four cell models. The simulation result shows that cell model one has the lowest temperature as it is closest to the coolant. The maximum temperature delta between the cell models is 0.3K.



Battery Pack Modeling

Creating A Module Assembly

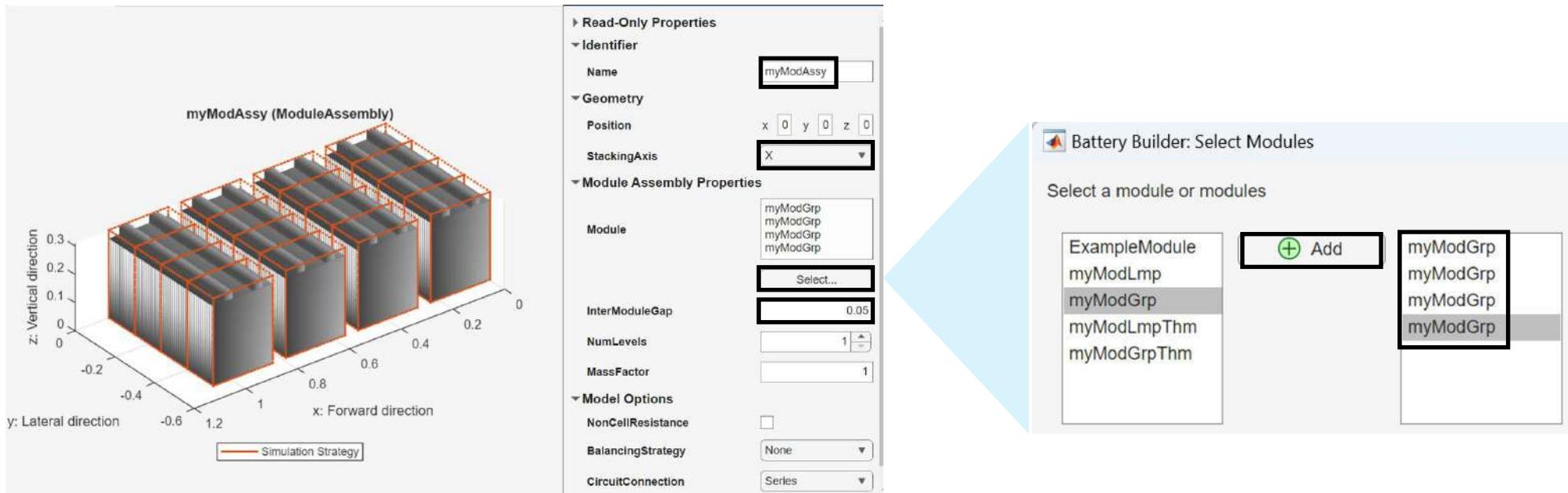
Next, you will continue your job in the **Battery Builder** app to create a module assembly and battery pack.

1. Click on the **Module Assembly** button to create a new module assembly.
2. Change the **Name** of the module assembly to **myModAssy**.
3. Click on the **Select** button and select the module for the assembly. Add four **myModGrp** to the module assembly.
4. Change the **StackingAxis** to **X**.
5. Change the **InterModuleGap** to **0 . 05**.
6. Click **Apply** to apply the changes.

Try

Follow the steps to create a module assembly.

At the module assembly level, Simscape Battery no longer asks you the number of modules to connect in series or parallel. Instead, you add modules individually and use the **CircuitConnection** property to choose between **Series** and **Parallel** connections. This setup enables you to combine different modules (that is modules with different fidelities) into a single module assembly.



Battery Pack Modeling

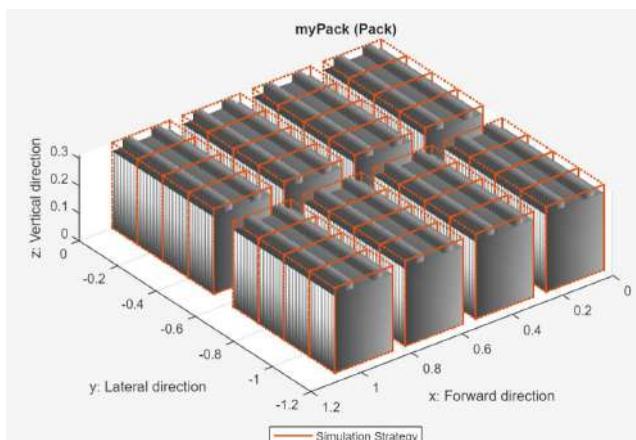
Creating Battery Pack

In the end, you create the full battery pack by connecting two module assemblies in series.

1. Click on the **Pack** button to create a new pack.
2. Change the **Name** of the pack to **myPack**.
3. Click on the **Select** button and add two **myModAssy** to the pack.
4. Change the **StackingAxis** to **Y**.
5. Change the **InterModuleGap** to **0 . 1**.

When creating battery objects using Simscape Battery, **Thermal Model Options** should always be set at the top level of hierarchy. In this example, you insert a **Parallel Channels** cooling plate at the bottom of the battery pack.

6. Change the **AmbientThermalPath** and **CoolantThermalPath** to **CellBasedThermalResistance**.
7. Check the **Bottom** checkbox for option **CoolingPlate**.
8. Under **CoolingPlateBlockPath**, change the second drop-down to **batt_lib/Thermal/Parallel Channels**.
9. Click **Apply** to apply the changes.



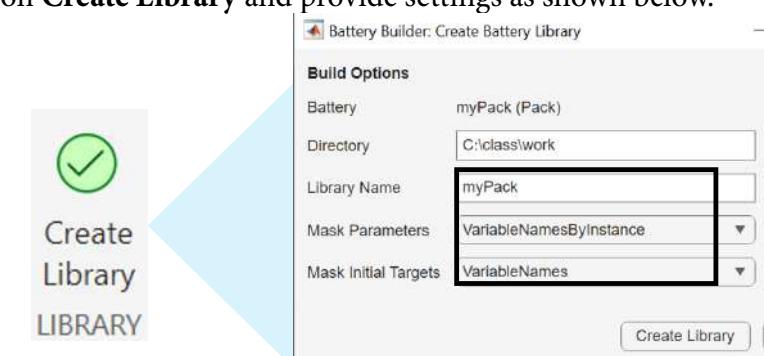
Try

Follow the steps to create a battery pack.

Identifier	
Name	<input type="text" value="myPack"/>
Geometry	
Position	x <input type="text" value="0"/> y <input type="text" value="0"/> z <input type="text" value="0"/>
StackingAxis	<input type="text" value="Y"/>
Pack Properties	
ModuleAssembly	<input type="text" value="myModAssy
myModAssy"/> <input type="button" value="Select..."/>
InterModuleAssemblyGap	<input type="text" value="0.1"/>
MassFactor	<input type="text" value="1"/>
CoolantThermalPath	<input type="text" value="CellBasedThermalResis..."/>
CoolingPlate	<input type="checkbox"/> Top <input checked="" type="checkbox"/> Bottom
CoolingPlateBlockPath	<input type="text" value="None"/> <input type="text" value="batt_lib/Thermal/Parallel..."/>
AmbientThermalPath	<input type="text" value="CellBasedThermalResis..."/>

Lastly, generate the Simulink library for the battery pack.

10. Click on **Create Library** and provide settings as shown below.



Battery Pack Modeling

Simulating a Battery Pack

Module assembly and pack objects are generated as subsystems that comprise module blocks. You can find them under the library file `LibraryName.slx`. Inside the pack or module assemblies, the subcomponents are electrically and thermally wired together based on the topology settings. Output signals are combined into vectors to be passed between layers. If the cooling plate property is specified, the corresponding cooling plate block will be inserted automatically.

Try

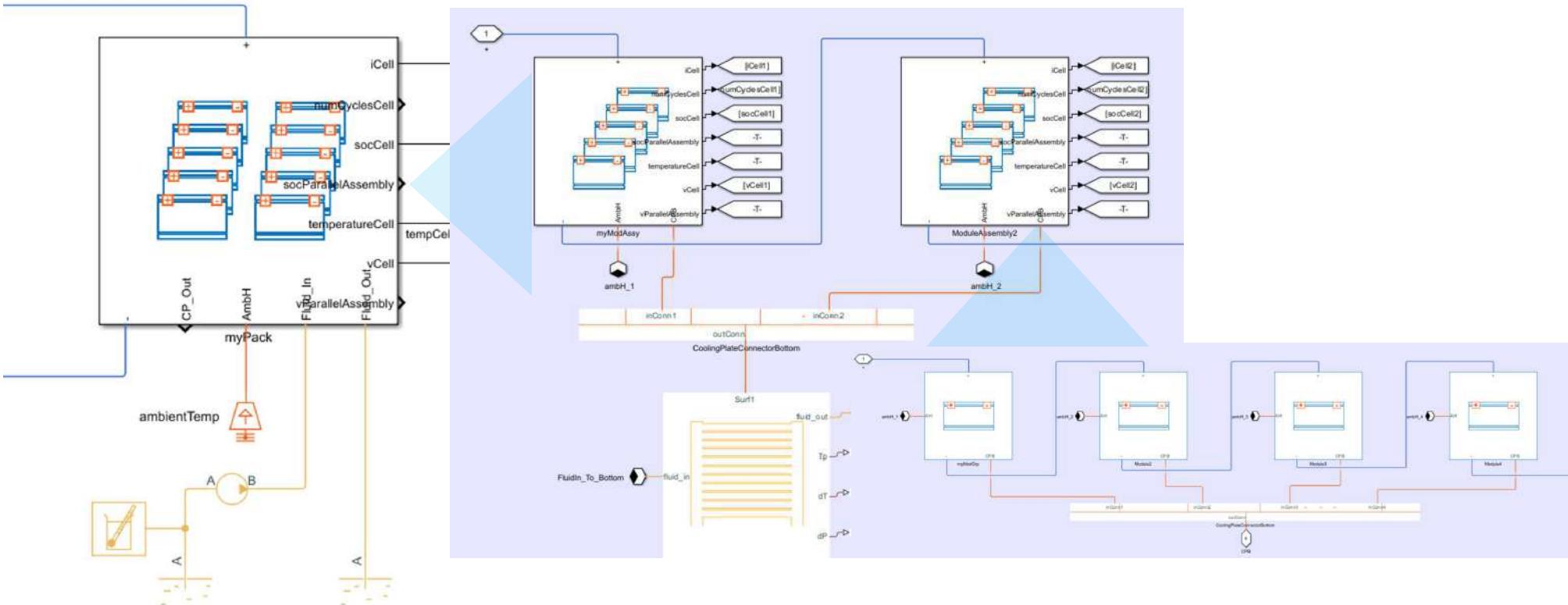
`>> myPack_lib`

Examine the generated blocks.

`>> pack01_start`

Add the battery pack into the model.

When creating the battery pack, the **Mask Parameters** (such as `V0`, `R0`) of each module inside the pack or module assembly can either be the same (`VariableNamesByType`) or per-instance specific (`VariableNamesByInstance`).



Battery Pack Modeling

Parallel Cooling

The Parallel Channels block models a cooling plate with multiple parallel channels and a pair of distributor pipes. Similar to the Edge Cooling block, the Parallel Channels block uses Thermal mass to represent the plate segment and Pipe (TL) to represent the coolant in the pipe. The number of plate segments is determined by number of partitions in X and Y dimension for the cooling plate. The number of Pipe(TL) block required for the Parallel Channels is determined by the combination of **Number of coolant channels**, **Select channel orientation direction**, and the plate segment partitions. Follow the steps to set up a parallel cooling plate with four cooling channels.

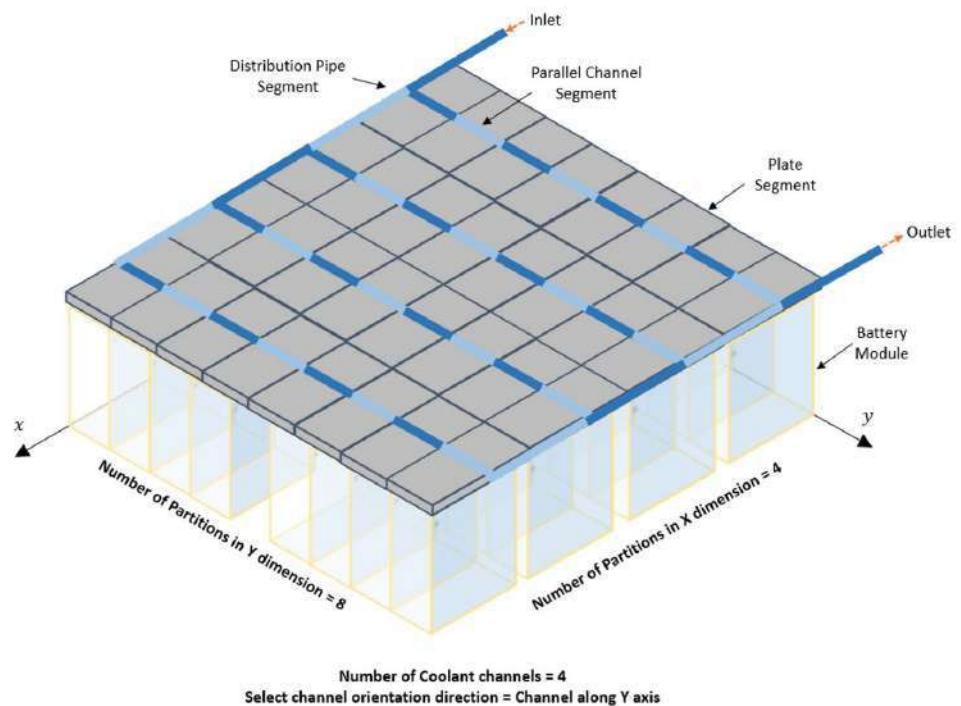
1. Go into myPack subsystem, and open the **Block Parameters** of **CoolingPlateBottom**.
2. Set the **Select channel orientation direction** as: **Channel along Y axis**.

Interface	
Number of partitions in X dimension for t...	CoolingPlateBottom.plateDimX
Number of partitions in Y dimension for t...	CoolingPlateBottom.plateDimY
Design	
Number of coolant channels	CoolingPlateBottom.nChannels
Select channel orientation direction	Channels along Y axis

In the given setup, four parallel channels go along the Y axis. Each parallel channel is divided into eight pipe segments given by Y dimension partitions. The distribution pipes go along the X axis. Each distribution pipe is divided into four pipe segments given by X dimension partitions.

Try

Follow the steps to configure the parallel cooling plate. Simulate the model.



When the number of partitions is greater than the number of coolant channels, each plate segment is connected to a separate pipe segment. When the number of partitions is smaller than the number of coolant channel, multiple pipes are mapped to the same plate segment. To get a desired temperature spread, it is important to select appropriate plate partitions to match with the number of coolant channels and the battery grouping strategies.

Battery Pack Modeling

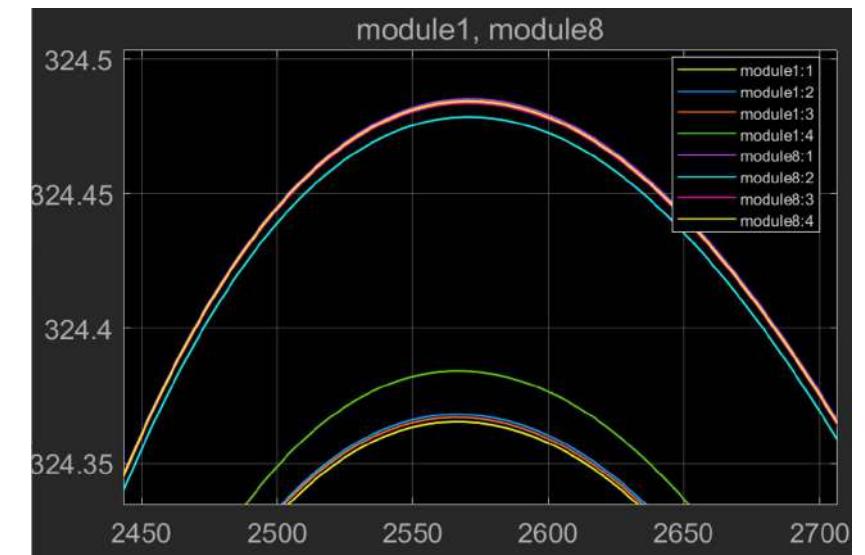
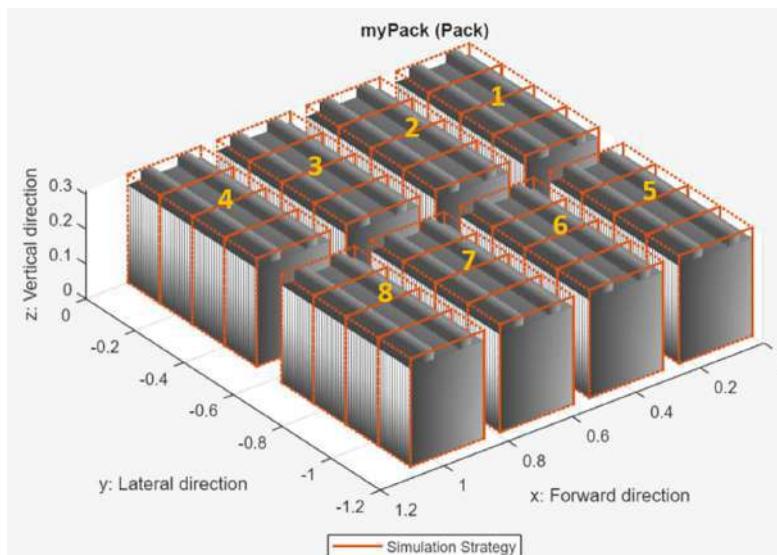
Parallel Cooling (Continued)

Simulate the model and examine the temperature differences between module one and module eight. Module one is closer to the inlet and module eight is closer to the outlet.

Try

```
>> pack02_parCool
```

Simulate the model and examine the temperature differences between modules.



Battery Pack Modeling

Summary

- Create battery module and pack objects with the Battery Builder app
- Control the model resolution of battery objects
- Add thermal fidelity to the battery objects
- Model cooling plates attached to the battery objects

Battery Pack Modeling

Test Your Knowledge

1. (T/F) When generating a battery pack with the Battery Builder app, a .ssc file is created to represent the pack.
2. (T/F) You can combine modules with different fidelities into a single module assembly.
3. (T/F) Setting the **CoolantThermalPath** option will provides you an array of cooling thermal nodes.

Answers

1. False
2. True
3. False



Battery Modeling and Algorithm Development with
Simulink®

Battery Management System

Outline

- Overview of the battery management system
- Design Stateflow logic to charge a cell using CC-CV control scheme
- Design supervisory control logic of battery management system using Stateflow
- Implement a passive cell balancing network
- Create test scenarios for battery management system using Simulink Test

The following names are trademarks of The MathWorks, Inc.:
Simulink®; Stateflow®; Simscape™ Battery™; Simulink Test™

Chapter Learning Outcomes

The attendee will be able to:

- Describe the key functionalities of a battery management system
- Model supervisory control logic with Stateflow
- Model passive cell balancing
- Define test input with the Test Sequence block

Overview of the Battery Management System

Lithium-ion battery packs are the predominant energy storage systems in aircraft, electric vehicles, portable devices, and other equipment requiring a reliable, high-energy-density, low-weight power source. The battery management system (BMS) is responsible for safe operation, performance, and battery life under diverse charge-discharge and environmental conditions.

When designing a BMS, engineers develop feedback and supervisory control that

- Monitors cell voltage, current, and temperature.
- Estimates state-of-charge and state-of-health.
- Controls the charging profile.
- Balances the state-of-charge of individual cells.
- Limits power input and output for thermal and overcharge protection.
- Isolates the battery pack from the load when necessary.

Battery Management System

Designing the BMS Algorithms

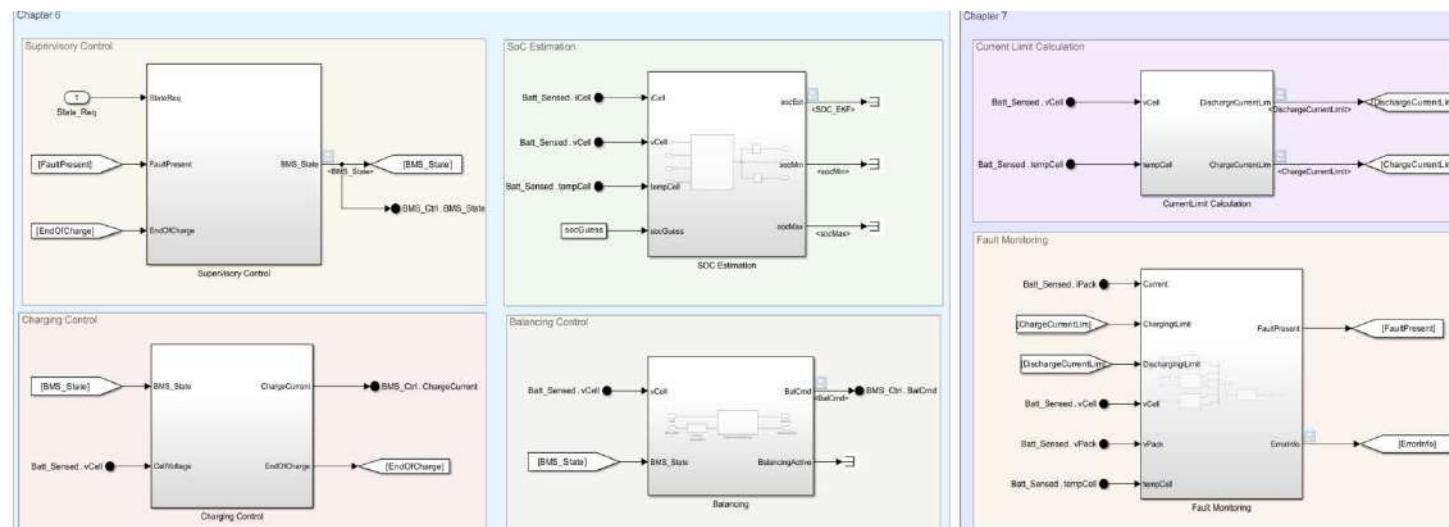
Starting from this chapter the focus will be on designing different control algorithms that are required in a BMS. The contents include:

Chapter 6 – Battery Management System

Covers the modeling of the supervisory control logic to decide the BMS operation modes and discusses how to build the charging and discharging logic with Stateflow. It also covers the design of a passive cell balancing algorithm.

Chapter 7 – Fault Diagnostics and Current Limit Calculation

Focuses on computing battery pack's charging and discharging current limits. It also covers fault detection (sensor fault and voltage fault, etc.) during the battery pack operation.



Try

Open the BMS template model and examine each component.

>> BMS01_start

Required Toolboxes

Simscape Battery - Simscape Battery provides prebuild BMS blocks for cell balancing, current management, and protection etc. You will learn how these blocks facilitate the BMS design.

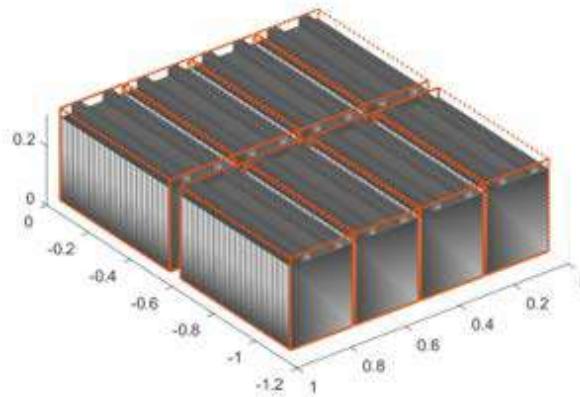
Stateflow - Instead of using prebuild blocks, Stateflow enables you to describe custom complex logic-driven systems in a graphical way. You will use Stateflow to implement the supervisory control logic.

Simulink Test - Simulink Test provides you blocks to easily create time-based test cases. You will use blocks from the Simulink Test library to provide input to the system.

Battery Management System

Battery Plant

In chapter four, you have learned how to create battery pack with different fidelities. For simplicity and avoid signal congestion, the battery pack implemented in this example is lumped with eight cell models. Each module is represented by one cell model.

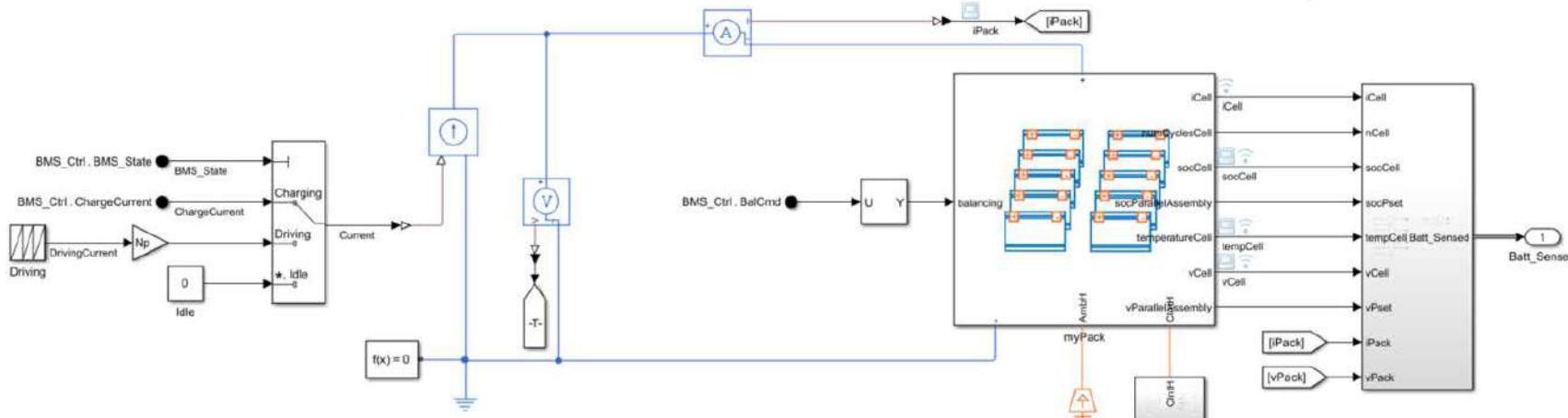


Try

Navigate to the **Plant** subsystem to examine the battery plant.

Additional to the signals produced by each cell model, pack level current and voltage are monitored. These signals are combined into the **Batt_Sensed** bus to be provided to the BMS.

Depending on the user request, the selected current profile should be given to the battery. A Multiport Switch is implemented to select the current profile based on the **BMS_State**. The driving current profile is given by the Repeating Sequence block running UDDS profile repeatedly. The charging current will be calculated by the BMS. You will learn to model the charging logic and setup **BMS_State** in the next step.



Battery Management System

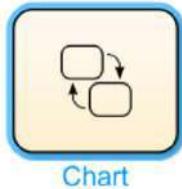
Constructing the Charging Logic with Stateflow

Unlike discharging a battery by running the load profile, the required charging current is usually decided by the BMS. Based on the SOC, voltage, etc., the BMS should control the current to prevent the battery from being overcharged, while trying to charge the battery to its full capacity.

In chapter two, you have implemented the CC-CV charging logic by using the Charger block provided by Simscape Battery. However, that is a Simscape block which is not optimal for BMS code generation. Instead, you can use Stateflow to easily set up a state machine with a finite number of modes. Go into the **Charging Control** subsystem and build the charging logic with Stateflow.

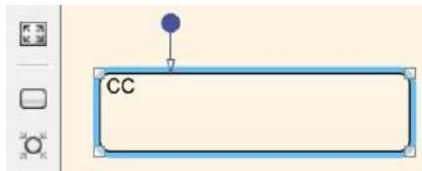
Creating a new chart

1. Add a Chart block from the Stateflow library to the model.



Adding states

2. Use the State toolbar button on the left to add two states into the model.



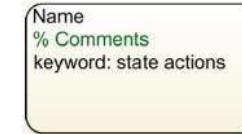
Try

Follow the steps to create the CC-CV charging logic.

Labeling the states

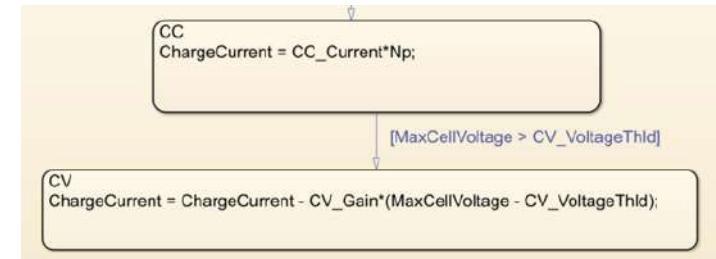
A generic state label consists of state name and state actions. The syntax is:

Note: The state action keyword controls when the actions are carried out.



Commonly used keywords are

- **entry** (or **en**) – Executes upon entering the state
 - **exit** (or **ex**) – Executes upon exiting the state
 - **during** (or **du**) – Executes when the state starts as active and remains active (no transition occurs)
3. Label the states as CC and CV.
 4. Give the corresponding state actions to the CC and CV states.
 - In the CC charging state, the charging current equals a constant of **CC_Current**.
 - In the CV charging state, the charging current is adjusted by the voltage difference between the maximum cell voltage and the voltage threshold.



Battery Management System

Constructing the Charging Logic with Stateflow (Continued)

Adding transitions

When the maximum cell voltage reaches the voltage threshold, the BMS should switch from CC charging to CV charging to prevent overvoltage. In Stateflow, you can create transitions to go from one state to another and provide transition labels to guard when the transition is valid.

5. Move your mouse to the edge of the CC state and drag it towards the edge of the CV state.
6. Double-click the transition to enable editing. Add `MaxCellVoltage > CV_VoltageThld` as the transition condition. Transition condition should be enclosed in [].

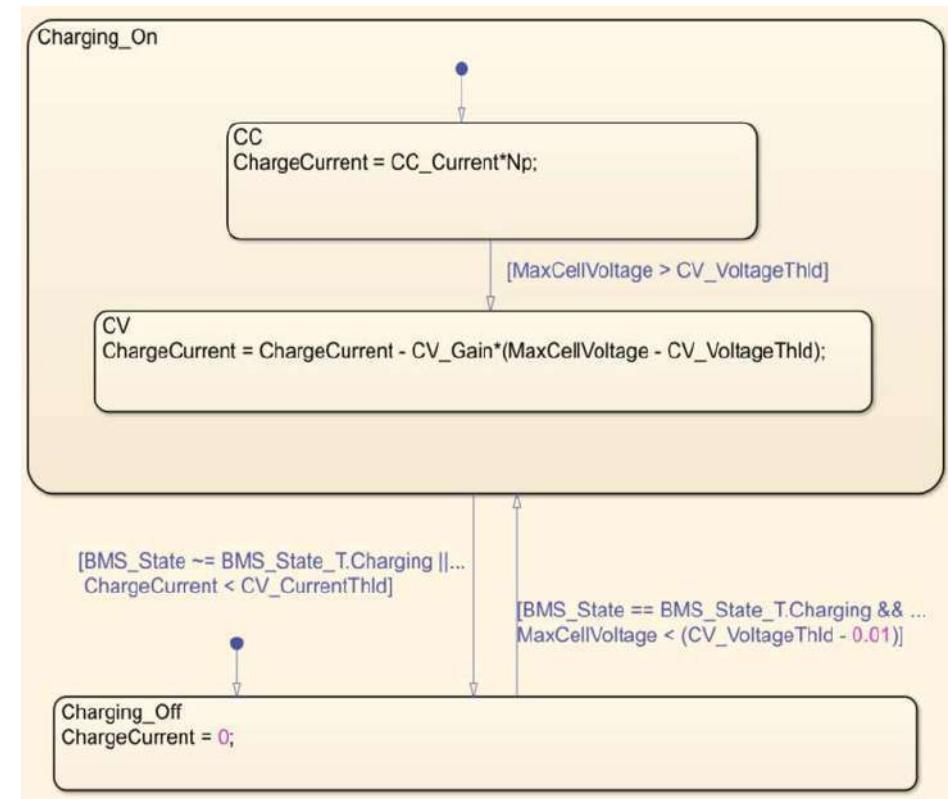
Creating super states

The CC-CV charging should only be enabled when the BMS_State is under Charging. You can add the extra layer of charging On/Off logic by adding hierarchies into the existing chart.

7. Select the CC and CV states and **Create Superstate** from the selection. Name the superstate **Charging_On**.
8. Create another state called **Charging_Off**. Give `ChargeCurrent = 0;` as the state action.
9. Add transitions between the **Charging_Off** and **Charging_On** states.
 - **Off to On:** when **BMS_State** is **Charging** and when **MaxCellVoltage** is smaller than **CV_VoltageThld - 0.01**
 - **On to Off:** when **BMS_State** is not **Charging** or when the end of CV charging is reached (`ChargeCurrent` is smaller than the cutoff current threshold).

Try

Follow the steps to create the CC-CV charging logic.



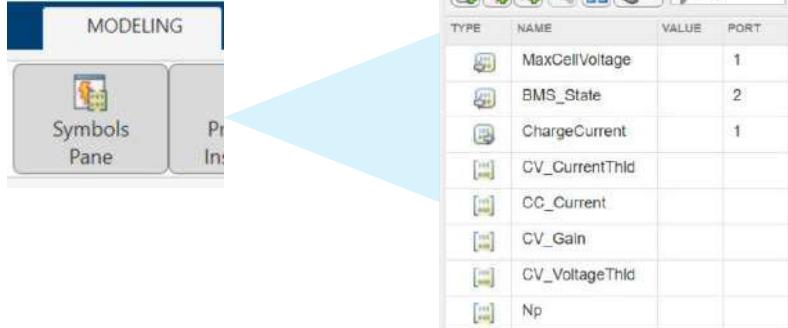
Battery Management System

Constructing the Charging Logic with Stateflow (Continued)

Defining chart data

In Stateflow, you must define the variable names used in a chart as chart data. You can use the **Symbols Pane** to create them.

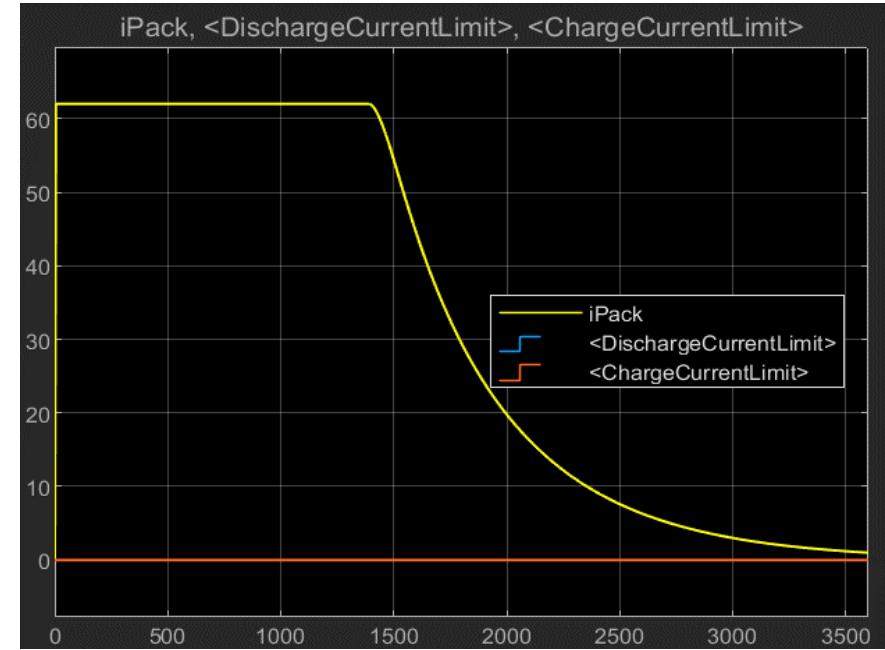
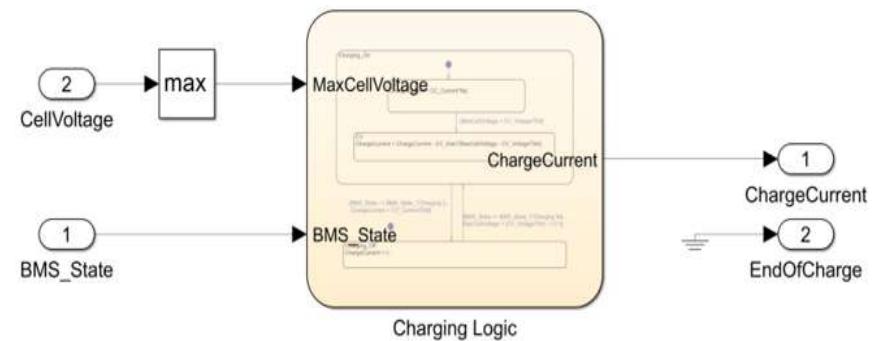
1. Open the **Symbols Pane** under the **Modeling** tab, **Design Data** section.
2. Define the data with the following scope.
 - Input Data: **MaxCellVoltage BMS_State**
 - Output Data: **ChargeCurrent**
 - Parameter Data: **CC_Current, CV_Gain, CV_VoltageThld, CV_CurrentThld, Np**
3. Connect the input/output signal lines outside the chart.
4. Simulate the model.



Try

Follow the steps to create the CC-CV charging logic.

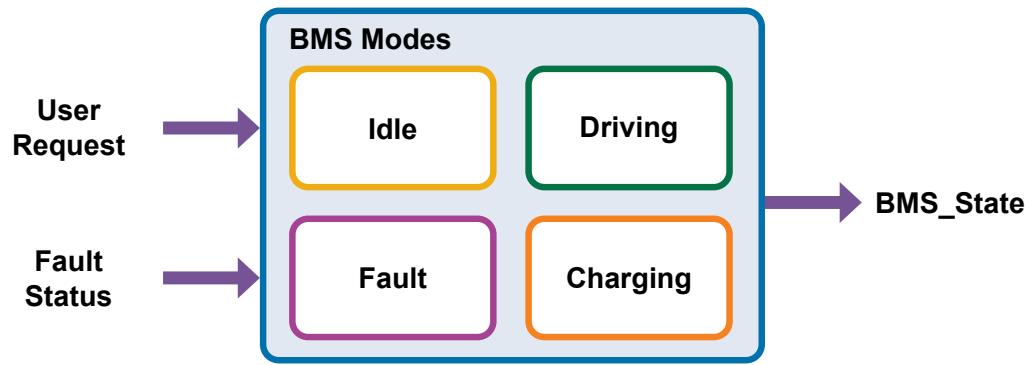
>> BMS02_charging



Battery Management System

Building the Supervisory Control Logic

Now you have discharged and charged the battery using separate control logic. It is time to combine all the functionalities together. The BMS decides which mode to operate on based on user request and current fault status. For the rest of this chapter, you will develop the supervisory control logic to decide the BMS state and trigger different BMS functionalities based on the selection.



System requirements

Depending on the **BMS_state**, a different current profile should be given to the battery. For example, when charging the battery, the charging current should be provided by the charging control logic in the BMS. When the user requests driving mode, the driving current profile should be given to the battery. If the BMS is in an idle state or at fault, all current should be cut off.

In Simulink, you can use a Multiport Switch to select the current profile based on the **BMS_State**. Instead of using numbers to represent **BMS_State**, you can use an enumeration data type to improve the readability of your model. **BMS_State_T** is a predefined enumeration data type that includes all the enumerated values for **BMS_State**.

Try

Open **BMS_State_T.m** and examine the enumeration data type.

>> edit BMS_State_T.m

```

classdef BMS_State_T < Simulink.IntEnumType
    enumeration
        Charging(1)
        Driving(2)
        Balancing(3)
        Idle(4)
        Fault(5)
    end
end

```

Battery Management System

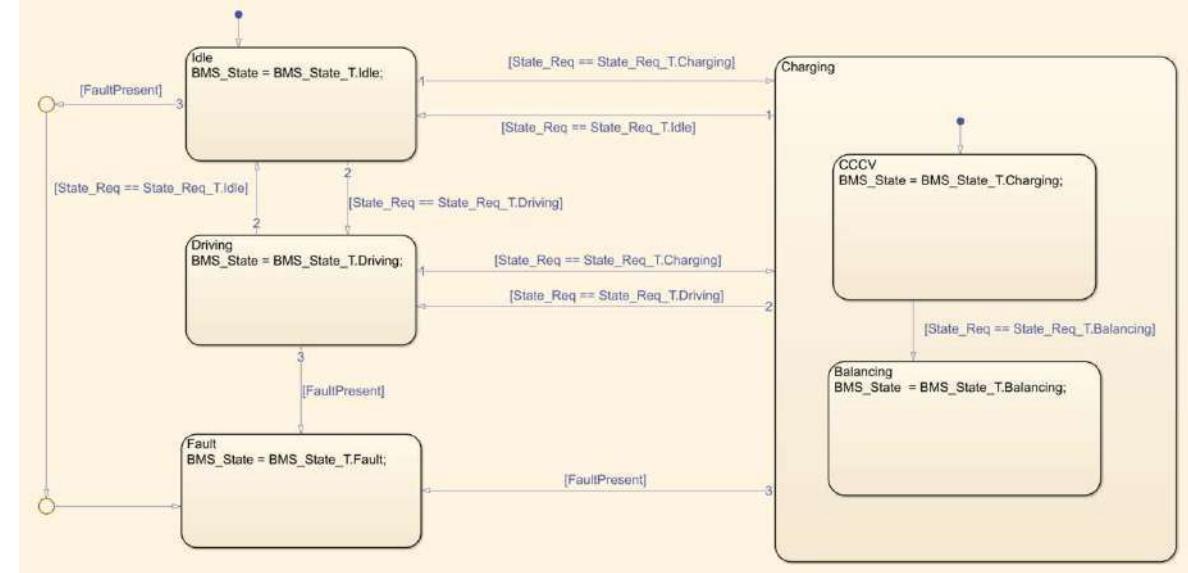
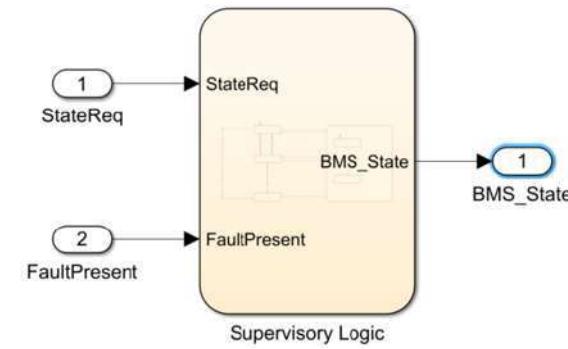
Supervisory Control State Machine

To decide the actual **BMS_State**, you will design the supervisory control state machine. The state machine takes user request **State_Req** and fault status **FaultPresent** as input, and produces **BMS_State** as output.

1. Add a Chart block into the **BMS** subsystem.
2. Add four states: **Idle**, **Driving**, **Charging**, **Fault**.
3. Inside the **Charging** state, add two substates **CCCV** and **Balancing**.
4. Set **BMS_State** values in the state actions.
5. Add the transitions based on **State_Req**, to transition between **Idle**, **Driving**, and **Charging**.
6. Add **FaultPresent** as the transition from each state to the **Fault** state.
7. Define **State_Req** and **FaultPresent** as inputs, and **BMS_State** as output in the **Symbols Pane**.
8. Connect the input/output signal lines outside the chart.

Try

Follow the steps to model the supervisory control state machine. Or, copy the prebuild chart from **supervisoryCtrl_chart.slx**



Battery Management System

Creating Test Sequence Input

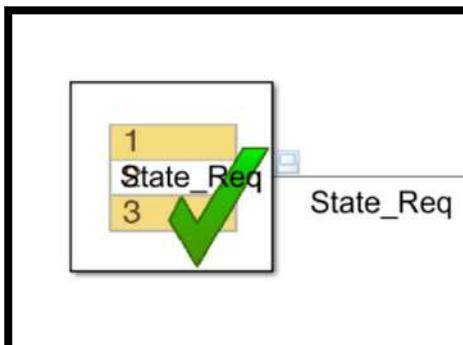
To test the supervisory control logic, you will design test cases that provide valid **State_Req** information. In Simulink, you can use the Test Sequence block from the **Simulink Test** library for the design. Instead of specifying the input at every time step, the Test Sequence block enables you to describe the test based on steps and transitions, similar to test descriptions in natural language.

You will now modify the Test Sequence block to implement the following logic to change **State_Req** between **Idle**, **Driving**, and **Charging**.

1. Standby for 60 seconds, give **State_Req** as **Idle**.
2. Transition to the **Driving** state, stay in the **Driving** state for 17500 seconds.
3. In the end, charge the battery by going into the **Charging** state.

To create a **Step**, specify the step name in the first line of the cell. Any additional lines are MATLAB® expressions known as step actions.

To add a step after the existing step, you can right-click, or place your cursor over the existing step cell to **Add step after**.

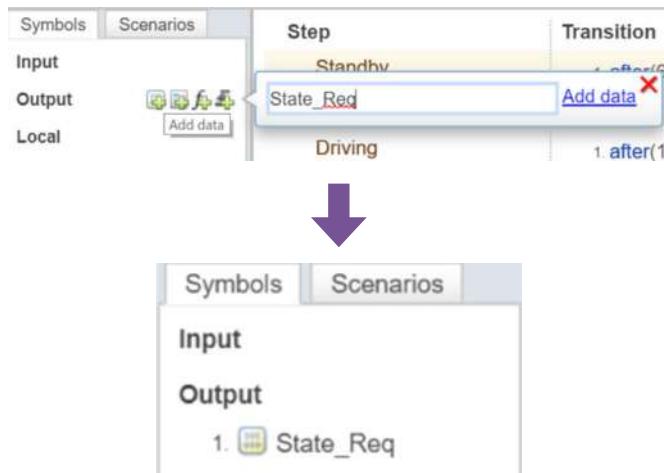


Step	Transition	Next Step	Description
: Standby State_Req = State_Req_T.Idle;	1. after(60,sec)	Driving	
Add step after • Add sub-step •			
Driving State_Req = State_Req_T.Driving;	1. after(17500,sec)	Charging	
Charging State_Req = State_Req_T.Charging			

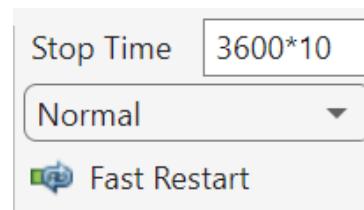
Battery Management System

Creating Test Sequence Input (Continued)

Similar to the requirement of Stateflow, you must create the symbols for input, output, parameters, etc., before using them in the Test Sequence block. You can add them from the **Symbols Pane**.



Change simulation **Stop Time** to $3600*10$. Simulate the model to test the supervisory control logic. The system goes under driving mode first, then switches to constant current charging. In the next section, the supervisory control logic is combined with the CC-CV charging logic.



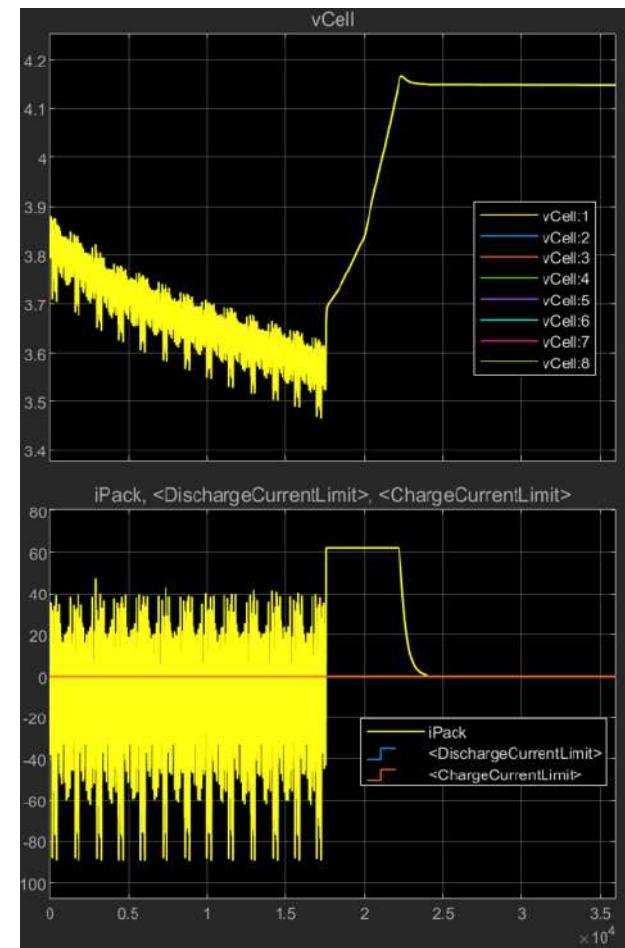
Try

Add **State_Req** as an output in the **Symbol Pane**.

Simulate the model.

Or,

>> BMS03_supervisory

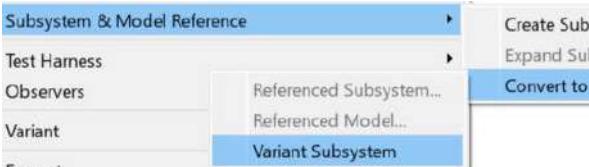


Battery Management System

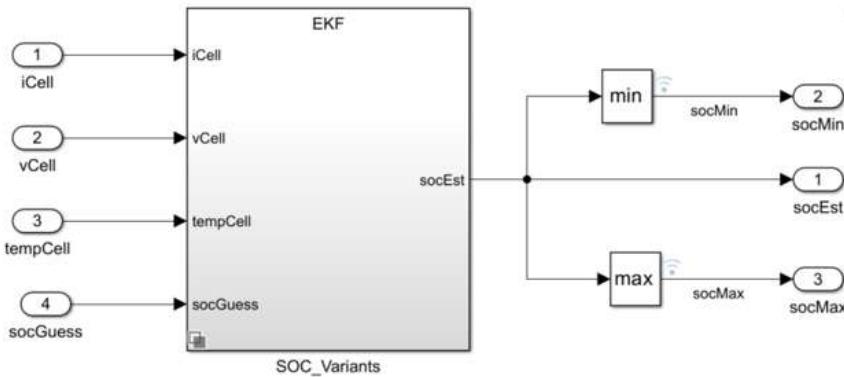
Integrating SOC Estimation

In chapter five, you have implemented coulomb counting and extended Kalman filter to estimate the SOC of a cell. When you have multiple options to complete a task, you can employ component variant to facilitate the implementation.

1. Go into the **SOC Estimation** subsystem, uncomment the **CoulombCounting** and the **EKF** subsystems.
2. Right-click on the **EKF** subsystem and go to **Subsystem & Model Reference** > **Convert to** > **Variant Subsystem**.
3. Rename the subsystem as **SOC_Variants**.



4. Cut and copy the **CoulombCounting** subsystem as another option into the variant subsystem.
5. Connect **SOC_Variants** with inputs and outputs as shown in the figure.



Try

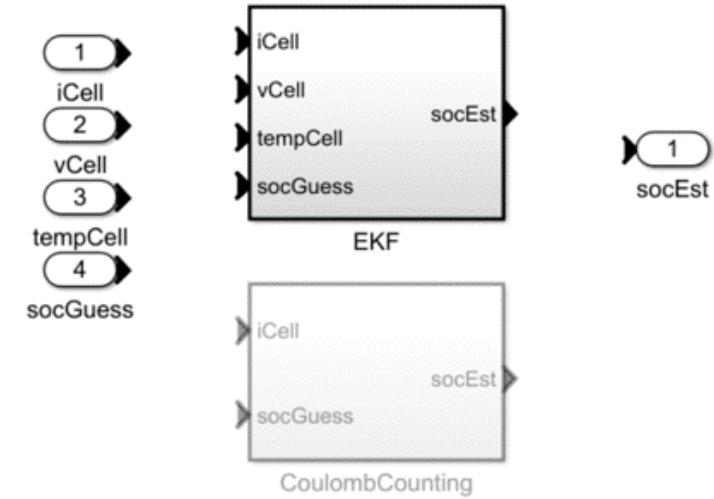
Follow the steps to create component variants with the two SOC estimation methods.

>> BMS04_socEst

In variant subsystems, signals are not explicitly connected between ports. Instead, they are mapped by the name of the ports. To select an active variant, follow the steps to control the variant selection with a predefined variable **socMethod**.

6. Right-click on the **SOC_Variants** and go to its **Block Parameters**. Change the **Variant control mode** to **expression**.
7. Provide the **Variant control expression** as shown in the figure.

Variant choices (table of variant systems)	
Name (read-only)	Variant control expression
CoulombCounting	<code>socMethod == 1</code>
EKF	<code>socMethod == 2</code>



Battery Management System

Cell Balancing

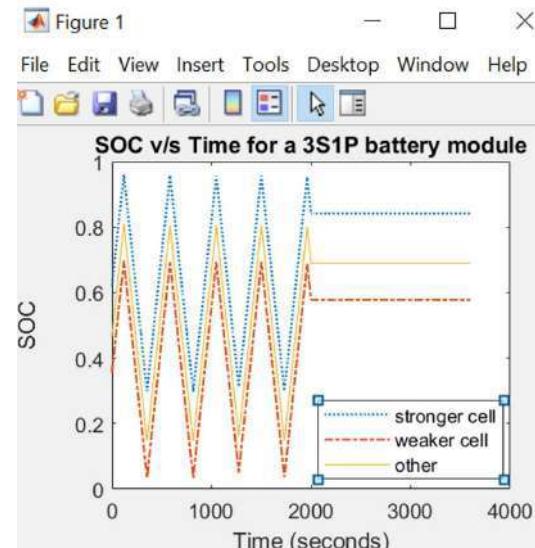
A battery pack is said to be unbalanced if the battery cells in the pack have a high difference in their SOC values.

During the battery pack operation over time, the cells have an inherent tendency to diverge to different SOC values and go out of balance. This imbalance may be caused by the difference in the coulombic efficiency of the cells, different leakage currents or self-discharge rates, the difference in the temperature gradient at different parts of the pack, or a combination of these factors.

The imbalance in the SOC levels makes the battery pack unusable or yields poor performance over time. In a series combination of cells, the cell with the lowest SOC value limits the discharging of the battery pack as it reaches the discharging limit first. Even if the other cells still have a significant charge remaining in them, they cannot be utilized. Similarly in charging, to avoid overcharging the cell with the highest SOC, other cells cannot reach a fully charged state.



The imbalance in the SOC necessitates the regulation in the charge level of individual cells. The BMS uses balancing schemes to ensure the cells are maintained at similar SOC values. You can use either passive (dissipative) or active balancing schemes for this task. In this course, a passive balancing algorithm is used to balance in SOC levels.



Battery Management System

Passive Cell Balancing

Passive balancing is a dissipative balancing technique where the excess charge of the cells with high SOC values is dissipated out to the environment in the form of heat. It makes use of basic electronic circuitry of switches and resistors as shown in the figure.

Since heat is being generated during balancing, the value of the balancing resistor should be carefully selected based on the balancing time and its power rating. A smaller balancing current is favorable to heat dissipation if a longer balancing time is allowed.

For example, assume you have 6 hours to balance the battery pack of your EV during the evening. The capacity of each cell is 30Ah and the maximum SOC difference is $dSOC = 5\%$. The desired balancing current can be calculated as:

$$I_{bal} = \frac{dSOC \times Q_{nom}}{dT} = \frac{5\% \times 30}{6} = 0.25(A)$$

If the balancing is conducted around the voltage of 4V, the required balancing resistor is:

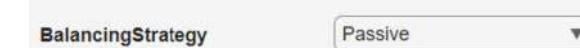
$$R_{bleed} = \frac{V_t}{I_{bal}} = \frac{4}{0.25} = 16 (\Omega)$$

Try

`>> balancingInitCondt`

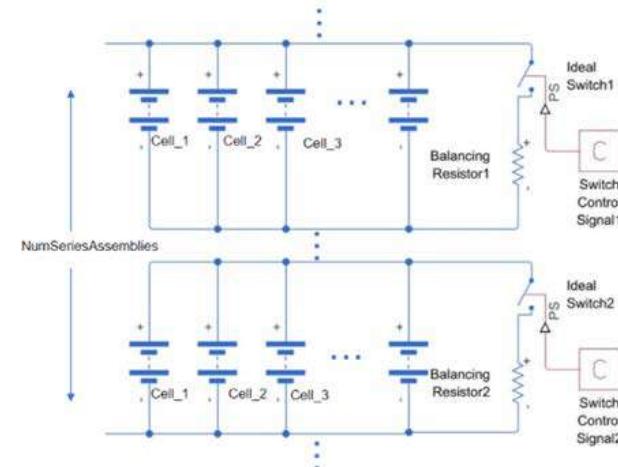
Run the script to initialize an imbalanced battery pack.

When using Simscape Battery to create custom battery pack, you can embed passive balancing resistors by setting **BalancingStrategy** to **Passive**.



The balancing resistances are added in parallel per parallel assembly. You can configure the balancing resistance through the **Block Parameters** of the module.

Cell Balancing	
> Cell balancing switch closed resistance	myModAssy.myModGr... Ohm
> Cell balancing switch open conductance	myModAssy.myModGr... 1/Ohm
> Cell balancing switch operation threshold	myModAssy.myModGrp.CellBalancing
> Cell balancing shunt resistance	myModAssy.myModGr... Ohm



Battery Management System

Building the Balancing Logic

There are multiple battery variables that can be used as balancing criteria, such as SOC and terminal voltage. If the cell SOC can be measured accurately, using SOC directly for balancing is a straightforward method. However, algorithms such as Extended Kalman Filter require large computation efforts which makes the implementation expensive. On the other hand, terminal voltage can be obtained easily through sensor measurement but it does not provide a good representation of the SOC when the battery is not settled. In real applications, voltage-based passive balancing is usually conducted by the end of CC-CV charging when the terminal voltage is close enough to the open-circuit voltage.

In this section, you will develop a voltage-based balancing logic using prebuild Simscape Battery block. Using the cell with the lowest SOC as the reference, the BMS activates the resistive network of the cells whose SOC is higher than the reference and allows the excess charge to dissipate out of the cells through the resistors in the form of heat energy.

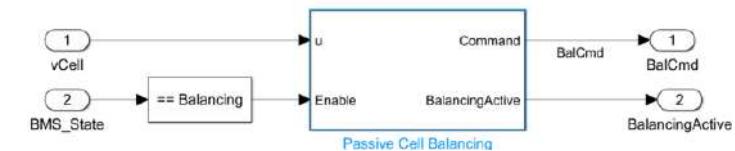
1. Go into the **Balancing** subsystem and add a Passive Cell Balancing block into the model.
2. Connect the input and output as shown in the figure.
3. Set the **Threshold** for balancing to **balThld**.

To unit test the balancing logic, you will temporary set the **BMS_State** to balancing.

4. In the **Supervisory Control** subsystem, add a Constant block and provide the value of **BMS_State_T.Balancing**.
5. Connect the Constant to port **BMS_State**.
6. Simulate the model.

Try

Add a Passive Cell Balancing block into the **Balancing** subsystem.

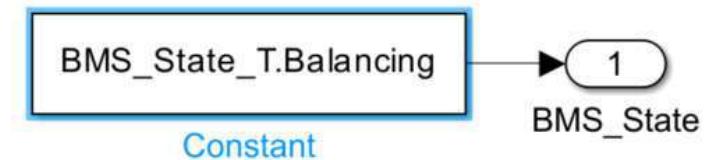


Parameters

Main Signal Attributes

Threshold for balancing

balThld

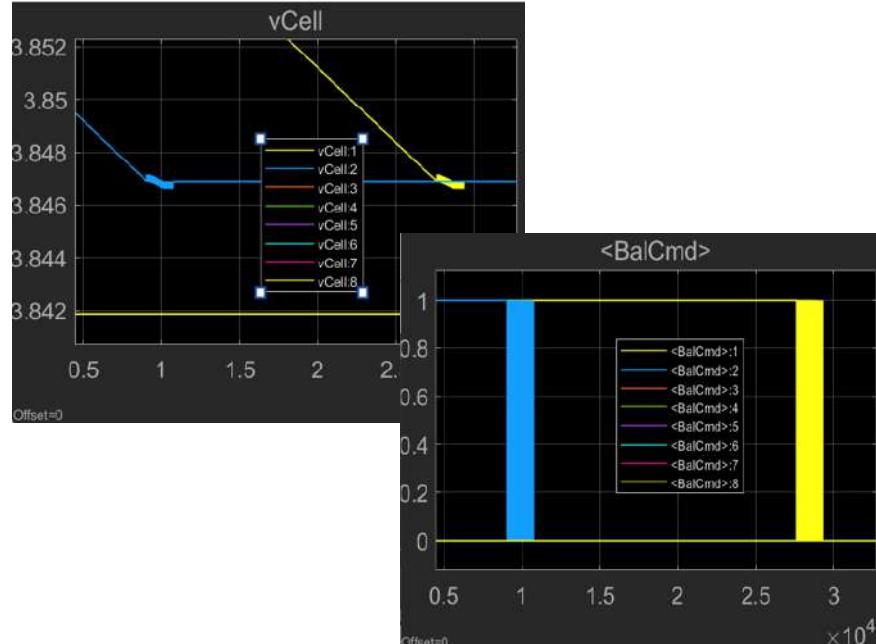


Battery Management System

Balancing Logic Solution

Starting with different SOC levels of 72%, 71%, and 70%, the cells converge to a maximum voltage difference of 0.005 V (**BalVoltageThld**). The SOC differences between the cells are minimized to around 0.5%.

However, there is a significant on-off switching event in the **BalCmd** as the SOC levels approach the threshold value. Since the balancing is done based on the terminal voltage value, cutting off the balancing current increases the terminal voltage above the threshold, which causes the switches to close again. The switching continues until the recovery voltage is smaller than the voltage threshold.



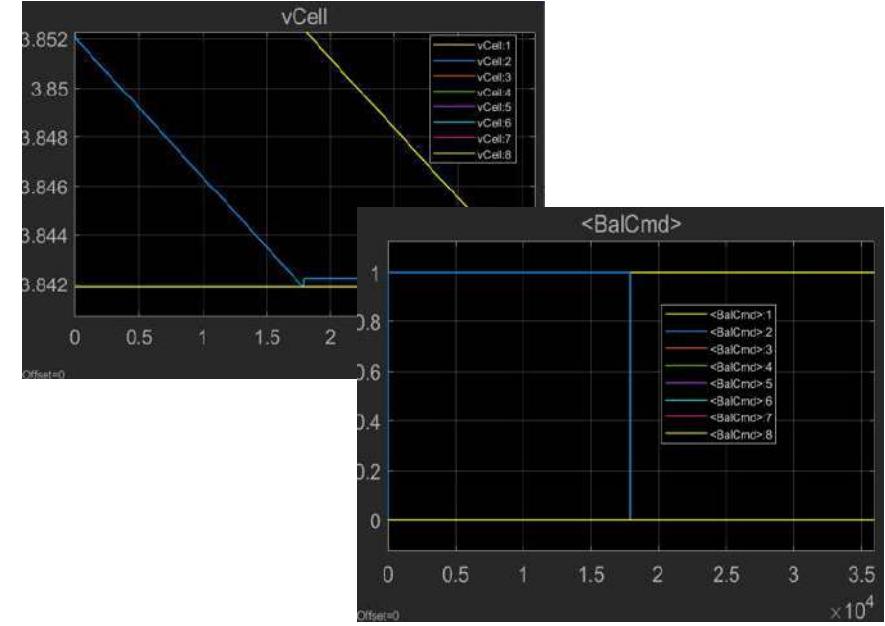
Try

Include hysteresis setting to debounce the balancing performance.

The balancing logic can be improved by adding hysteresis into the voltage threshold.

1. Set the **Hysteresis band for switching balancing off** to **balHyst**.
2. Simulate the model again.

Hysteresis band for switching balancing off
balHyst



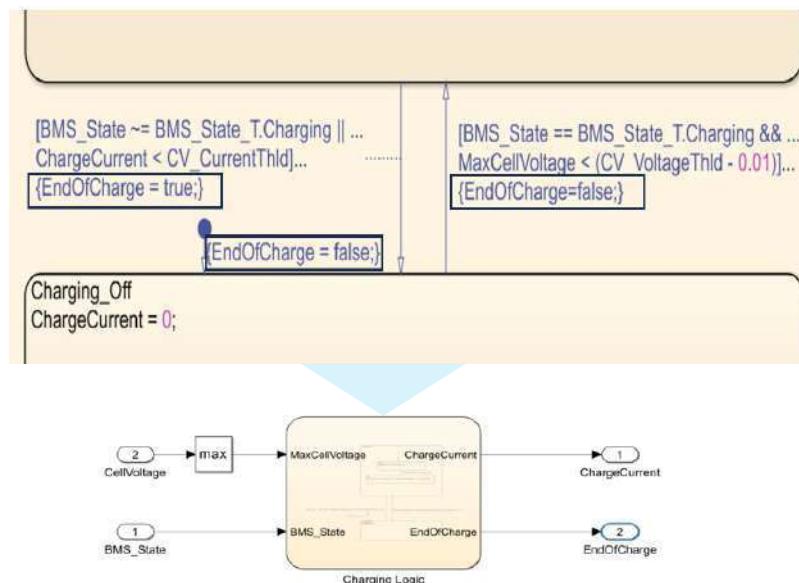
Battery Management System

Integration with Battery Management System

The balancing algorithm can now be integrated as one of the supervisory control tasks done by the BMS.

The BMS should initiate the balancing task at the end of the charging period of the battery module. To notify the supervisory control logic the battery reaches the end of charge:

1. Add an output variable **EndOfCharge** to the **Charging Logic** chart.
2. Set the default value of **EndOfCharge** as false on the **Default Transition** when entering the chart.
3. Add a condition action to change **EndOfCharge** to true while transitioning out of the **Charging_On** state, thereby indicating the end of the charging process.
4. To reset the **EndOfCharge** flag, change **EndOfCharge** to false on the transition from **Charging_Off** to **Charging_On**.

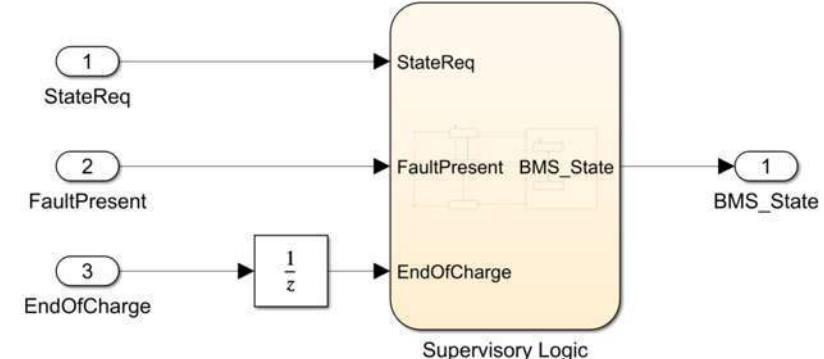


Try

>> BMS05_balancing

Follow the steps to trigger balancing by the end of CC-CV charging

5. Add **EndOfCharge** as an input to the **Supervisory_Control** chart. Use **EndOfCharge** as an Or condition to transit from the **CCCV** substate to the **Balancing** substate.
6. Restore the connection to the **BMS_State**.
7. Simulate the model.



Summary

- Overview of the battery management system
- Design Stateflow® logic to charge a cell using CC-CV control scheme
- Design supervisory control logic of battery management system using Stateflow®
- Implement a passive cell balancing network °
- Create test scenarios for battery management system using Simulink Test™

Test Your Knowledge

1. (T/F) To use the enumeration data type defined in file A.m, you need to run the .m file before using it in Simulink®.
2. (T/F) Test Sequence blocks must be configured to have at least one input and one output.
3. (T/F) In a Test Sequence block, the value you specified in the transition column must be a logical expression.

Answers

1. False
2. False
3. True

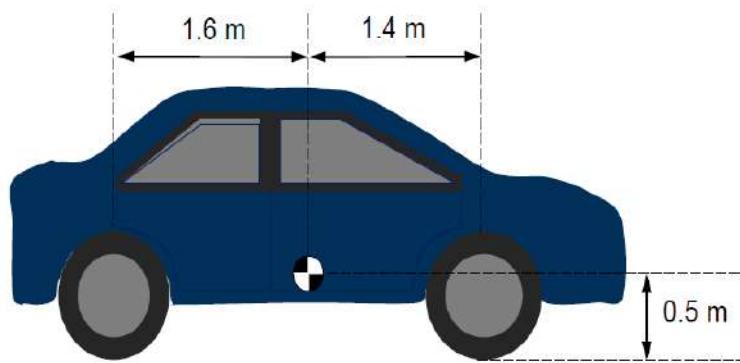
Vehicle modeling using Simscape

Getting started

In EVs, modeling vehicle dynamics is essential to calculate the load torque acting on the motor. The load torque translates into the motor current which majorly contributes to the losses in motor as well as inverter and the discharging of battery. Even, for sizing the motor, inverter, dc-link, and the battery, it is important to capture all the essential dynamics of the vehicle.

By using Simscape Driveline, we can model the longitudinal dynamics of the vehicle which includes vehicle mass, mass distribution, tires (longitudinal), terrain (road slope), wind effects etc. We can also mode the mechanical powertrain that includes gears, differentials, shaft compliance, and mechanical brakes as required.

For this exercise, we will modeling the vehicle with the following specifications.



Parameters

Vehicle body:

Vehicle mass: 1500 kg

Wheels per axle: 2

Frontal area: 3 m²

Drag coefficient: 0.4

Distances are shown on the diagram below.

Tires (magic formula):

Rolling radius: 16 in (406.4 mm)

Tire stiffness: 200000 N/m

Tire damping: 1000 N/(m/s)

Tire inertia: 4.129 kg*m²

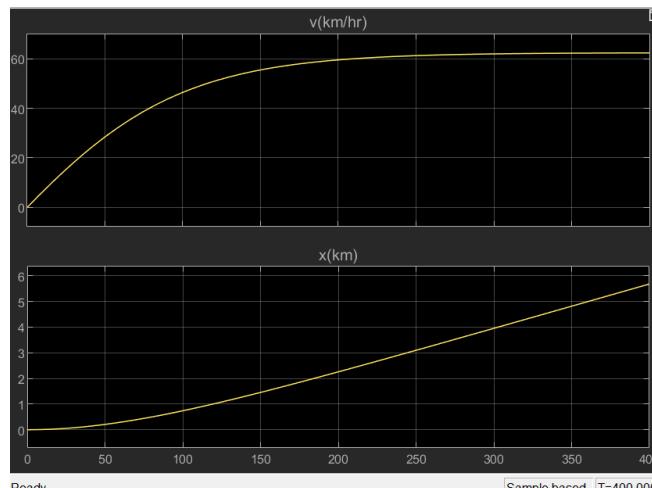
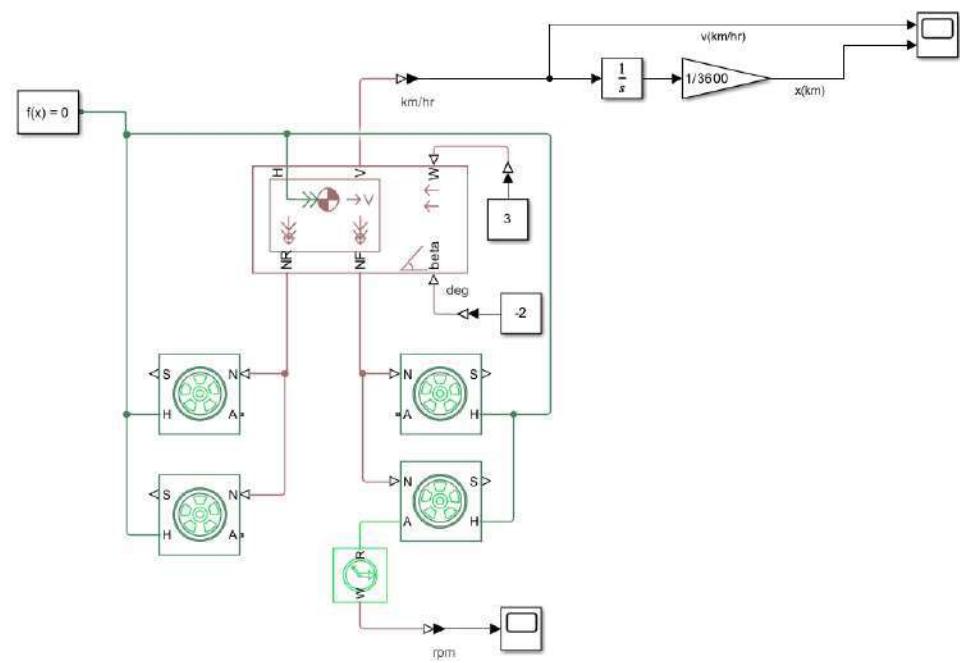
Rolling resistance: 0.015

To get started, we will be modeling the scenario where the vehicle freely moves on the inclined road with a slope of 2° downwards. We will also include the headwind speed of 3 m/s. We are interested in calculating the vehicle velocity (in km/hr), the distance travelled by the vehicle (in km).

Open a new Simulink model. Get ‘Vehicle Body’ block from Simscape Driveline library. Read the description of the block to understand the physical conserving ports and the signal ports of the block. Edit the parameters as given in the above list. Get four ‘Tire (magic formula) blocks in the model and edit the parameters for the same. Connect the Vehicle Body and the Tires through port H.

To provide the slope, connect a ‘Constant’ block (with parameter -2) through ‘Simulink-PS converter’ to ‘beta’ port of ‘Vehicle Body’. Set the input unit as ‘deg’. In a similar manner, to provide headwind speed, connect a ‘Constant’ block (with parameter 3) through ‘Simulink-PS’ converter to ‘w’ port of ‘Vehicle Body’.

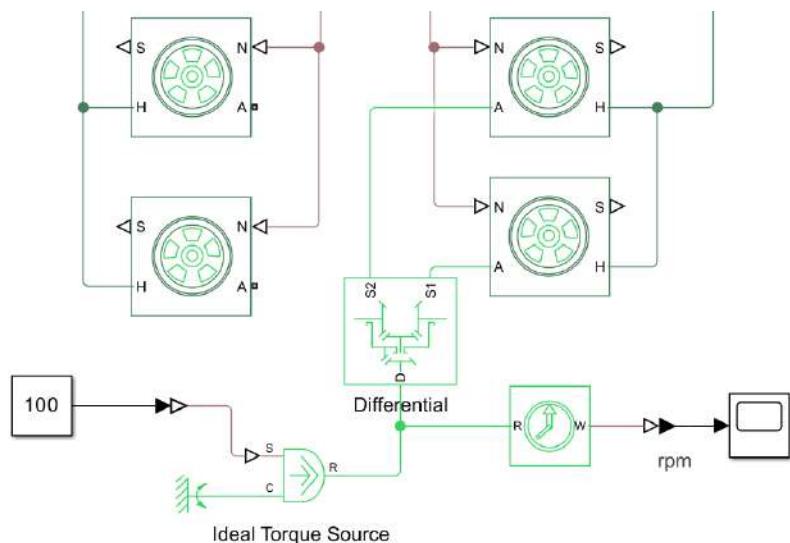
For measuring vehicle speed, we can use ‘V’ port in the ‘Vehicle Body’. Use ‘PS-Simulink’ converter to use ‘Scope’ to view the variation of speed with time. Set the unit as ‘km/hr’. Distance can be calculated by integrating the velocity over the time. To measure tire rpm, connect the ‘Ideal Rotational Motion Sensor’ block from Simscape to the ‘A’ port of any of the tires. Simulate the model for 400 seconds to see the significant dynamics of the systems.



Applying torque to the axle of the vehicle

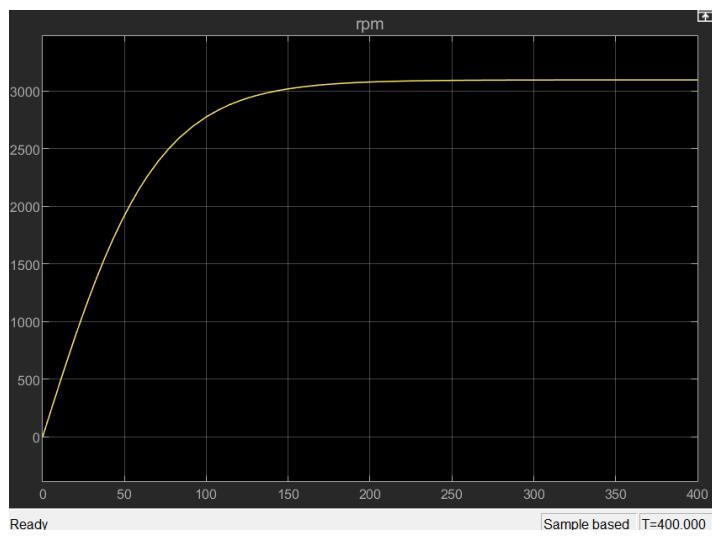
In vehicles, the power needs to be transmitted from the source (engine or motor) to the axle of the vehicle. In conventional vehicles, it is done through a quite complicated transmission system which includes the gear and the clutch assembly, belt drives, differentials etc. In EVs, this powertrain is relatively simple as the motor can provide a constant torque over a wide range of vehicle speeds. In this exercise, to keep it simple, we are only including a differential gear. We can use the block from Simscape Driveline and use the default parameters.

To analyze how the vehicle accelerates at a constant torque (100 Nm), let us make the road gradient (beta) as well as the headwind speed (w) zero. The constant torque can be provided by using ‘Ideal Torque Source’ block from Simscape. Considering the front wheel drive, we are connecting a differential to the front wheels.



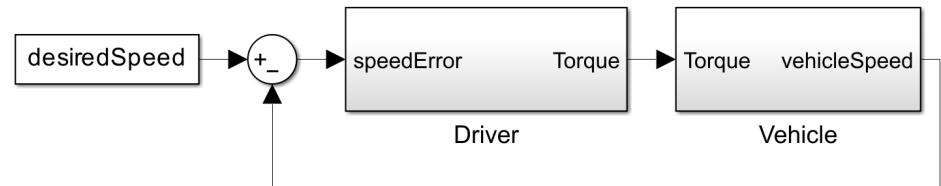
To measure the required motor speed, the ‘Ideal Rotational Motion Sensor’ block is connected to the ‘D’ port of the differential.

After running the simulation, we can see the vehicle velocity and the motor speed required for the same.



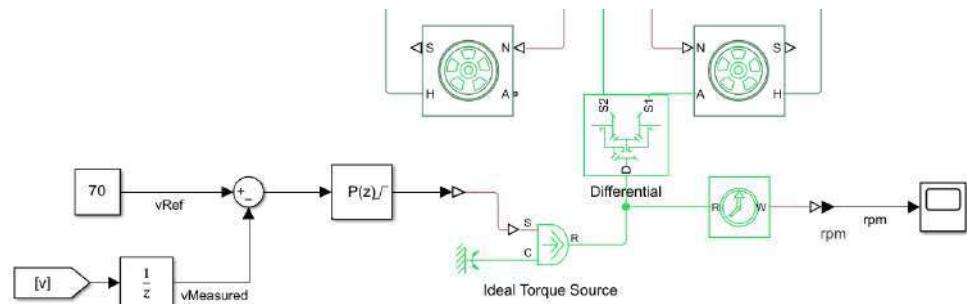
Accelerating the vehicle to the desired speed

In any vehicle, the torque provided by the source (engine/motor) is not constant. The torque profile is dependent on the desired and the current vehicle speed. The driver adjusts the pedal and accordingly controls the torque that the source (motor in EVs) provides to the vehicle. For sizing the motor for the vehicle, the required torque and the power profile must be computed for a given drive cycle. This can be easily conceptualized by adding a driver in a closed loop as shown below.



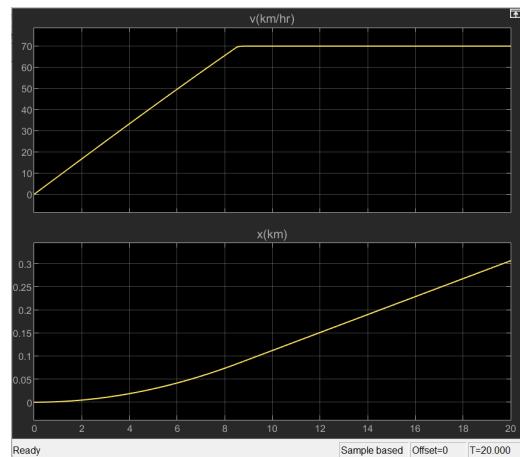
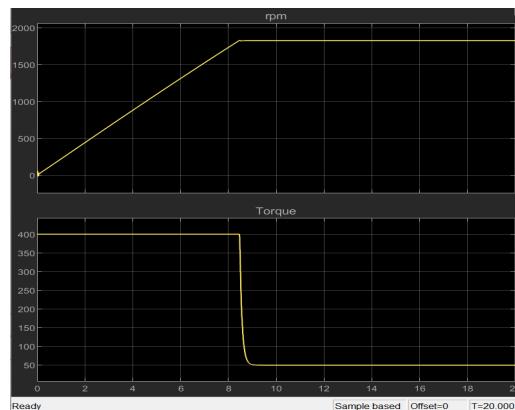
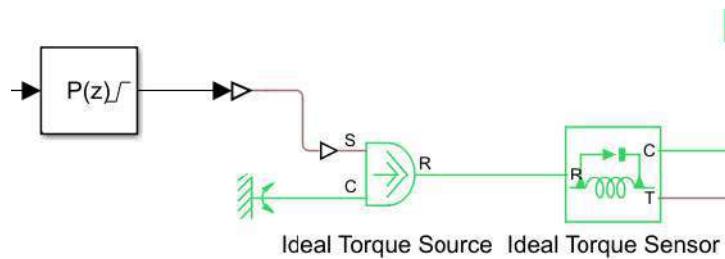
Here, we already have a vehicle model to which we can provide an input torque using ‘Ideal Torque Source’ block. The driver model will be control algorithm which calculates the required torque depending upon the speed error.

Set the desired speed as 70 km/hr. Add a PID controller block to the model. Provide the vehicle velocity as a feedback. Also, to account for the sensor update, connect a unit delay block in the feedback loop.



The sample time of the discrete controller and the unit delay block is set as 0.01 seconds. The PID block is configured as a proportional controller with $P = 400$. The output of the controller (torque) is limited to 400 Nm (this depends upon

the motor design limits). After running the simulation for 20 seconds, we can see that the vehicle accelerates to 70 km/hr. By connecting ‘Ideal Torque Sensor’ in the drive path, we can measure the torque required for the same.



To check if the driver model works well even on a different terrain, let us change the road inclination during the simulation. Let us consider a case where the road inclination angle ‘beta’ is a function of a distance travelled by the vehicle.

If $2 < x < 2.5$, $\text{beta} = 3^\circ$; otherwise $\text{beta} = 0^\circ$

There are multiple ways of implementing this algorithm in Simulink. One of them is by using a ‘MATLAB Function’ block with the help of which, we can use MATLAB language to model a functionality in Simulink.

Get MATLAB Function block inside the model. Once you double click on it, you find an environment where an algorithm can be written as a MATLAB function.

```
function beta = Slope(x)
```

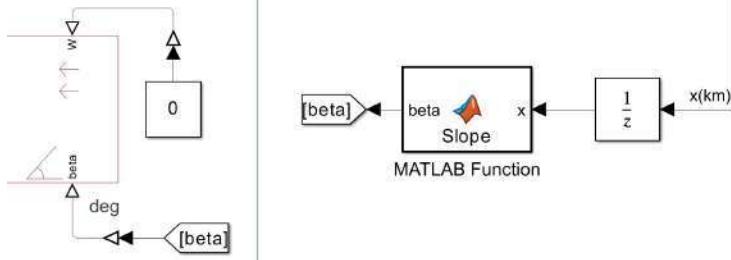
```
if x > 2 && x < 2.5
```

```
    beta = 3;
```

```
else
```

```
    beta = 0;
```

```
end
```

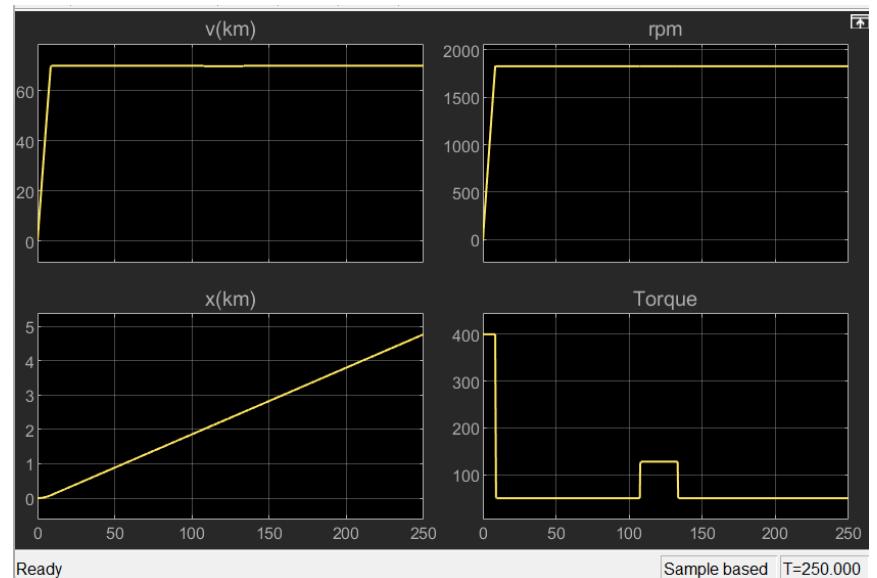


After running the simulation for 250 seconds, we get the following results.

We can see from the results that to keep the vehicle running with the same speed even at a positive slope, the torque requirement increases.

Although the driver model works substantially well for the changes in the load, but there is a slight dip in the vehicle speed in the power requirement from the vehicle increases. This is a limitation of a proportional control that it cannot guarantee zero steady state error. Hence, to improve the driver model, let us change the configuration of the controller to PI, with $I = 30$. Also, set the anti-windup method

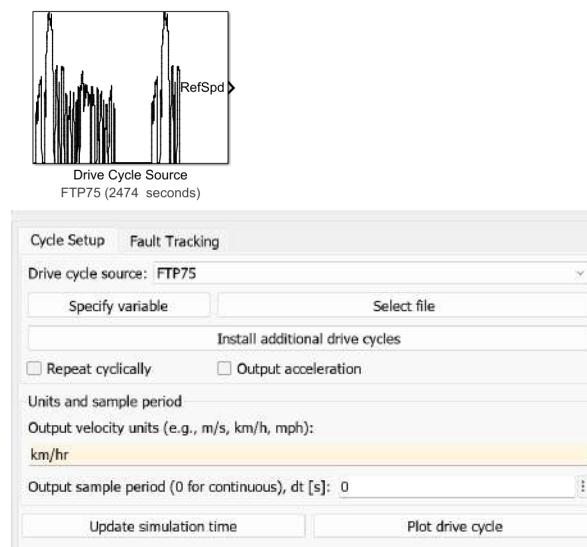
in the Saturation tab to ‘clamping’ to prevent the integrator windup.



Torque profile for a standard drive cycle

Once the controller (driver model) is able to ensure that the vehicle can maintain the desired speed, we can compute the torque profile for the drive cycle of our choice. The required torque and the power profile help the designer in sizing motors, inverters and batteries. Also, calculating required motor current from the torque helps in analyzing the thermal behavior of the EV powertrain and thus provide good insights in designing the cooling systems.

From Powertrain Blockset, we can make use of the block ‘Drive Cycle Source’. This block can output the variation of speed with time for some of the standard drive cycles.

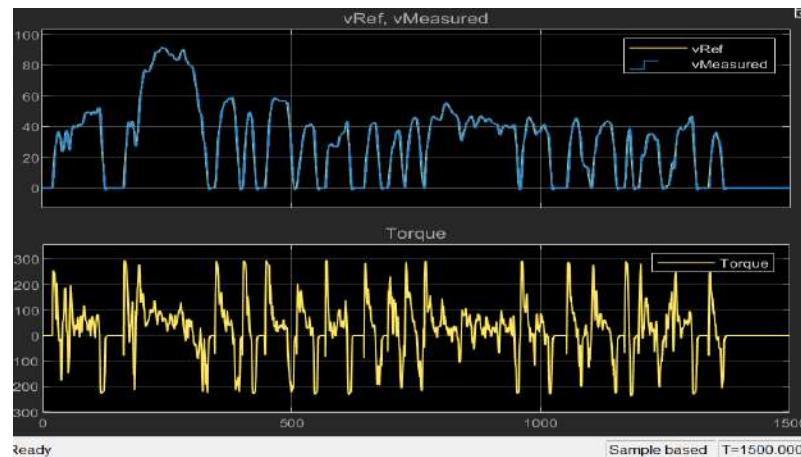


We will use this block to provide the reference speed. Set the output velocity unit as km/hr. Run the simulation for 1500 seconds. Ensure that the vehicle speed tracks the reference. The torque measured is the required torque that the motor needs to provide for the given drive cycle.

In this scenario, the braking is also being performed by the source as the controller limits are set from -400 to 400. It means that 100% of the braking is generative. In actual situation, the braking action will be a combination of

regenerative and the mechanical braking. Check the model ‘VehicleDynamics_withBrakeClosedLoop.slx’.

The torque and the motor rpm data generated from this model can be used to test the motor control algorithm that has been developed in the earlier part of this course.



System level motor

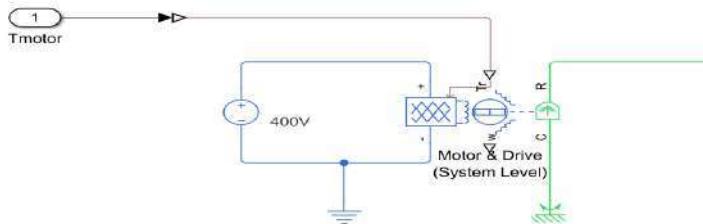
The motor model along with the torque controller will be integrated with the vehicle model. But even before that, if we want to find out the approximate current requirement to generate the required torque, we can use ‘Motor & Drive’ block from Simscape Driveline (or Simscape Electrical).

In this block we can provide the motor characteristics such as torque-speed, efficiency at maximum torque, etc. Depending on the load requirements on the mechanical and the torque reference (T_r), it consumes the equivalent current

from the source, i.e. dc link voltage. With this block, we can find the current requirement by the motor of specific characteristics for a given vehicle and drive cycle.

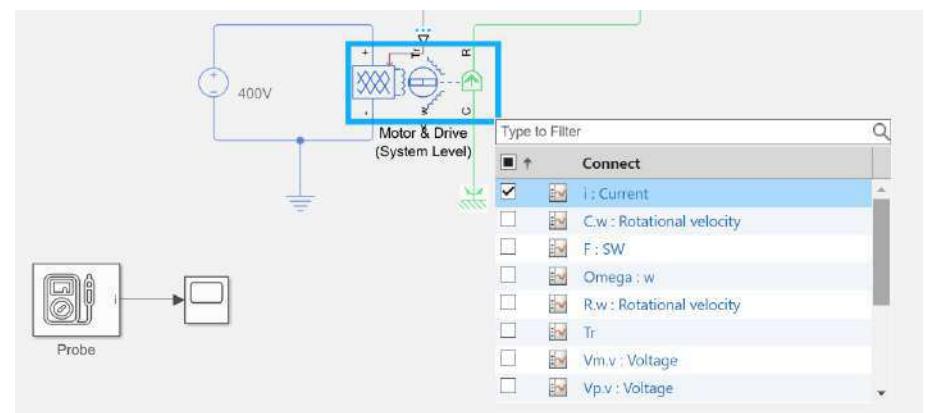
In the model ‘VehicleDynamics_withBrakeClosedLoop.slx’, let us use this block to compare the battery discharging for motor with and without regenerative braking. In the Vehicle subsystem, replace the Ideal Torque Source with ‘Motor & Drive (System-Level)’ block from Simscape Electrical. Add a dc voltage source of 400 V (dc link) and connect it to the electrical terminals as shown in the figure.

Parameterize the block with the following details.

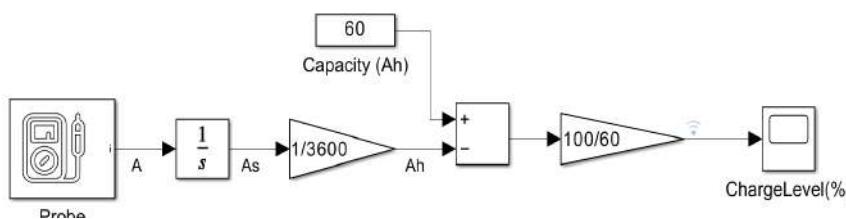


The easiest way to find the SoC consumed is by the load is by calculating charge by integrating the current over the time. To measure the current consumed by the motor, we can either connect current sensor in the motor circuit or use a ‘Probe’ block from Simscape Utilities. After double clicking on the Probe, we can select the Motor & Drive and select Current from the available variables.

NAME	VALUE	
Modeling option	No thermal port	
Electrical Torque		
Parameterize by	Maximum torque and power	
Allow intermittent over-torque	No	
> Continuous operation maximum torque	400	N*m
> Continuous operation maximum power	60000	W
> Torque control time constant, Tc	0.02	s
<input type="checkbox"/> Enable supply switch		
Electrical Losses		
Parameterize losses by	Single efficiency measurement	
> Motor and driver overall efficiency (percent)	95	
> Speed at which efficiency is measured	1500	rpm
> Torque at which efficiency is measured	400	N*m

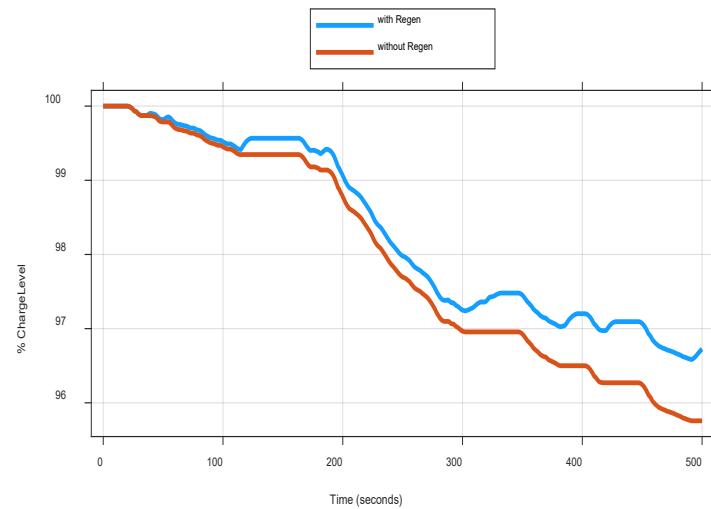
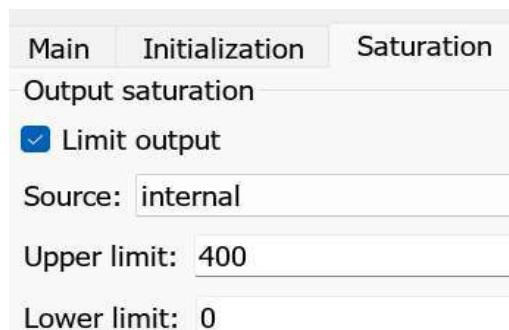


If a battery capacity is known (say 60 Ah), the available charge level in the battery can be easily calculated by the given mathematical expression.



For the sake of comparison, we can also enable the data logging for the signal that is capturing the charge level.

To analyse the model without regenerative braking, set the saturation limits in the speed controller from 0 to 400. To analyse the same with 100 % regenerative braking, change the limits from -400 to 400.





Accelerating the pace of engineering and science

mathworks.com/training

© 2020 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc.

See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.