Simscape™ Battery™

User's Guide

MATLAB®&SIMULINK®

R2022b

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

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

Simscape™ Battery™ User's Guide

Trademarks

Patents

Revision History

September 2022	Online only	New for Version 1.0 (Release 2022b)

Contents

=== Getting Started ===

1

Simscape Battery Product Description **1-2**

Battery Pack Modeling Workflows

2

Battery Modeling Workflow **2-2**
Define Battery Design **2-3**
Visualize Battery **2-3**
Define Model Resolution **2-3**
Build Battery Model **2-4**

Simulation and Analysis of Thermal Management Systems

3

Connect a Cooling Plate to a Battery Module and Parallel Assembly **3-2**

Examples

4

Build Model of Hybrid-Cell Battery Pack **4-2**

Protect Battery During Charge and Discharge for Electric Vehicle **4-10**

Peak Shaving with Battery Energy Storage System **4-28**

Thermal Analysis for New and Aged Battery Packs **4-37**

Size Resistor for Battery Passive Cell Balancing **4-41**

Battery Monitoring **4-45**

Battery Charging and Discharging **4-48**

Battery State-of-Health Estimation **4-50**

Battery State-of-Charge Estimation **4-52**

Battery Passive Cell Balancing **4-54**

Build Detailed Model of Battery Pack From Cylindrical Cells **4-56**

Build Detailed Model of Battery Pack From Pouch Cells **4-81**

Build Model of Battery Module with Thermal Effects **4-103**

Build Model of Battery Pack with Cell Aging **4-110**

Build Model of Battery Pack with Cell Balancing Circuit **4-116**

Build Model of Battery Pack for Grid Application **4-123**

Build a Simple Model of a Battery Module in MATLAB and Simscape **4-129**

Build a Simple Model of a Battery Pack in MATLAB and Simscape . . . **4-134**

1

=== Getting Started ===

Simscape Battery Product Description

Design and simulate battery and energy storage systems

Simscape™ Battery™ provides design tools and parameterized models for designing battery systems. You can create digital twins, run virtual tests of battery pack architectures, design battery management systems, and evaluate battery system behavior across normal and fault conditions.

Battery Pack Model Builder is a design tool that lets you interactively evaluate different battery pack architectures. The tool automates the creation of simulation models that match the desired pack topology and includes cooling plate connections so electrical and thermal responses can be evaluated.

Parameterized models of battery packs and battery management systems demonstrate operations, including cell balancing and state of charge estimation. You can use these examples to determine cell requirements, perform trade-off analyses and hardware-in-the-loop (HIL) testing, and generate readable and efficient C/C++ code.

2

Battery Pack Modeling Workflows

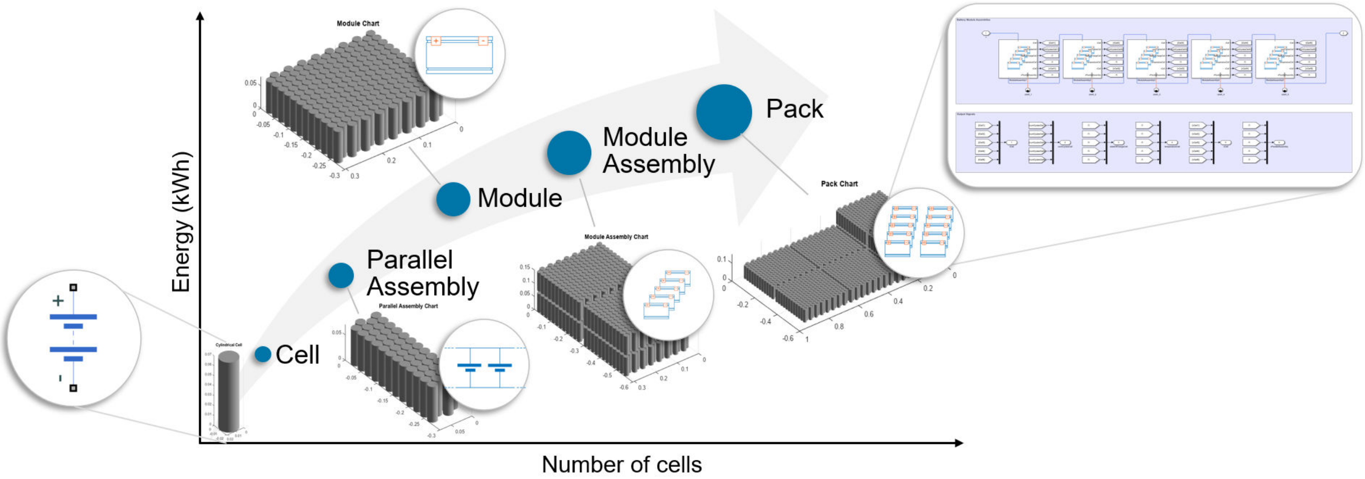
Battery Modeling Workflow

In this section...

“Define Battery Design” on page 2-3
“Visualize Battery” on page 2-3
“Define Model Resolution” on page 2-3
“Build Battery Model” on page 2-4

Simscape Battery includes MATLAB® objects and functions to automate the creation of Simscape battery models. Use these objects and functions to define your own battery design specifications, visualize your battery in a 3-D space, customize the modeling resolution during simulation, and generate a Simulink® library that contains your custom generated battery blocks. You can use these blocks to assist with virtual battery design and verification, help develop battery control algorithms using Simulink software, explore design sensitivities, and design thermal management strategies.

You can develop and test battery control strategies by simulating your custom battery blocks with the blocks in the **Battery Management System (BMS)** library of Simscape Battery. You can also thermally couple your custom battery models in Simulink with the blocks in the **Thermal Management System** library. Alternatively, you can define your own custom battery control and cooling system blocks.



By using the battery objects in Simscape Battery, you can specify several electro-thermal features that you want to model in your battery simulation. For example, you can:

- Add a cell-balancing circuit to every parallel assembly or cell for BMS control.
- Add custom thermal boundary conditions, such as thermal resistors, that represent ambient heat dissipation paths.
- Enable battery aging models in the cell-level model block.

All battery models are scaled up from a single cell model block, which by default is defined as the Battery (Table-Based) block.

You can customize the model resolution before model creation to suit the model requirements of your specific engineering problem. A larger number of equivalent circuit models of a battery provides a higher resolution. By default, the model resolution is `Lumped`, which is the lowest resolution and provides the best simulation speed and compilation time. This resolution indicates that only one "scaled-up" equivalent circuit model represents your system. If you increase the model resolution to `Grouped`, you can customize the number of electrical and thermal models required to answer your specific engineering question while increasing simulation speed. If you require a very detailed battery model, you can choose to simulate every single cell inside your battery electrically and thermally. This level of resolution comes at a great performance cost. To support real-time simulations, keep the number of equivalent circuit models equal to or less than 30. All custom Simscape Battery models support the Simscape scalable compilation feature.

To create your own battery model, follow these steps:

1. "Define Battery Design" on page 2-3
2. "Visualize Battery" on page 2-3
3. "Define Model Resolution" on page 2-3
4. "Build Battery Model" on page 2-4

Define Battery Design

Create a Simscape Battery object in your MATLAB workspace and specify its properties. These are the battery objects you can create:

- `Cell`
- `ParallelAssembly`
- `Module`
- `ModuleAssembly`
- `Pack`

You can also create these objects without any inputs and define them with the required level of detail. You can create the battery models with or without defining the geometrical characteristics of the battery cells and the battery topology. High-level models without consideration of geometry are normally used as value models early in the design stages of a prototype pack to evaluate key performance indicators. Battery mass and packaging volume are dependent properties that you can obtain by querying the `Mass` and `PackagingVolume` properties of the battery object. To determine the energy and capacity of your battery pack, you must simulate your battery model first.

You can access the MATLAB objects of Simscape Battery from the MATLAB Command Window.

Visualize Battery

The `BatteryChart` object provides a custom battery visualization function to verify the hardware specifications of your battery, such as the cell dimensions, inter-cell spacing, inter-module spacing, number of cells, selected parallel assembly topology, and many more. Geometry and cell layout are required properties to perform more detailed thermal modelling with thermal management system blocks, like the coupling of a battery module block with one of the cooling plates blocks provided in the **Thermal** library of Simscape Battery.

Define Model Resolution

Set a suitable model resolution or simulation strategy by specifying the `ModelResolution` property in the `ParallelAssembly` and `Module` levels. When you specify the resolution of your battery model, you must consider the trade-off between model resolution and model speed. To obtain optimal performance, keep the number of models to lower than or equal to 30. You can simulate specific regions or areas of your battery by using a grouped model resolution and by specifying the `SeriesGrouping` and `ParallelGrouping` properties. With this flexible approach, you can simulate specific subcomponents of your battery that exhibit the hottest and coldest temperatures, or the highest and lowest state of charge. You must capture these spreads to correctly test and develop the battery control strategy.

Build Battery Model

Use the `buildBattery` function to create a custom battery model from the `ParallelAssembly`, `Module`, `ModuleAssembly`, and `Pack` objects. This function creates one or two libraries in your current working directory that contain the necessary subsystems and variables you need to simulate the battery. The `buildBattery` function creates one library for the Simscape-level battery blocks of the object hierarchy (`ParallelAssembly` and `Module`), and another library for the Simulink-level battery subsystems, `ModuleAssembly` and `Pack`.

3

Simulation and Analysis of Thermal Management Systems

Connect a Cooling Plate to a Battery Module and Parallel Assembly

Simscape™ Battery™ includes blocks and models of battery cooling systems for simulations of battery thermal management. You can use these blocks to add detailed thermal boundary conditions and thermal interfaces to the battery Module or ParallelAssembly blocks. These cooling system blocks contain both thermal and thermal-liquid domain connections:

- To interface to or from battery blocks that include a thermal model, use the thermal domain nodes.
- To specify coolant inlet and outlet properties and operating conditions, use the thermal-liquid domain nodes.

The cooling system blocks of the **Thermal** library are flat cooling plates. These blocks support three main flow configurations: parallel channels, U-shaped rectangular channels, and edge cooling. In the edge cooling configuration, the coolant flows at one end of the flat plate and all the heat from the battery cells is transferred via conduction within the cooling plate material. You can discretize these cooling plates into elements to closely capture temperature spreads resulting from the dynamic interaction with the battery and the coolant flow.

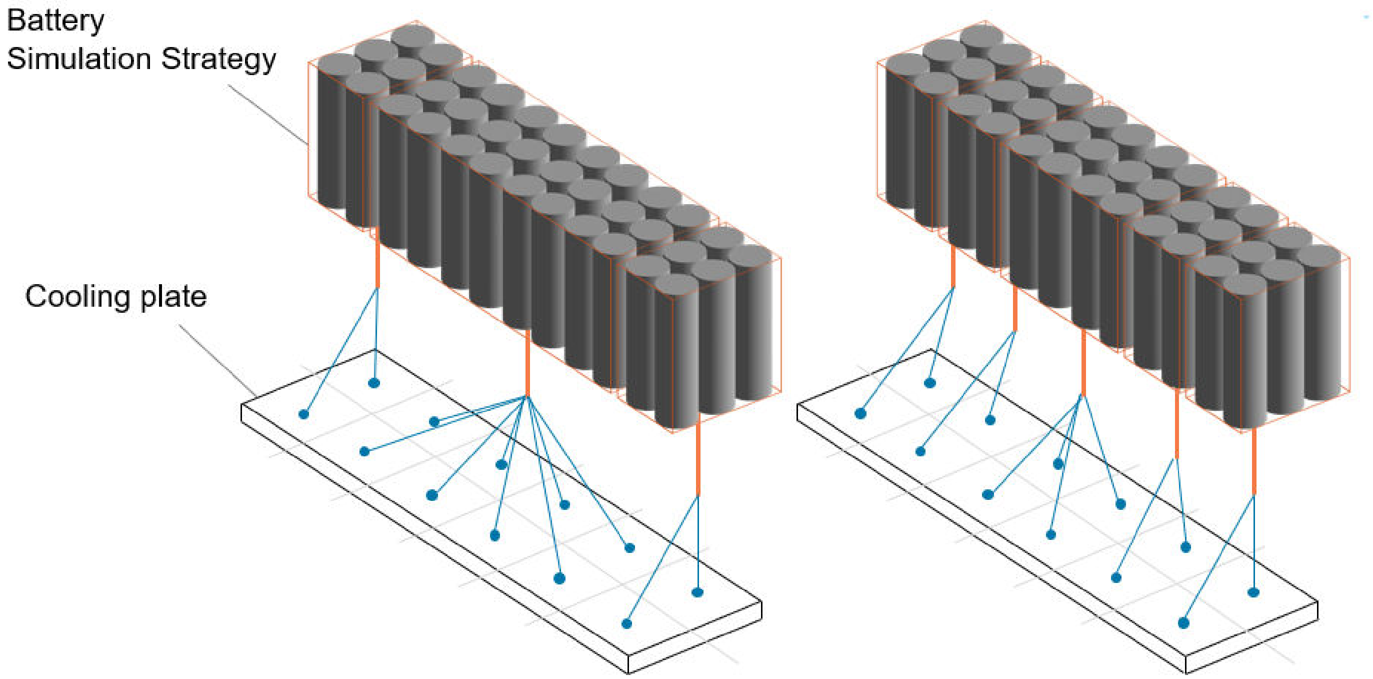
To link a cooling plate to a battery block:

1. Define your battery object and model. To display the required thermal interface characteristics for cooling plate coupling in the form of a structure, use the **ThermalNodes** property of the battery `ParallelAssembly` and `Module` objects.
2. Drag and drop your battery block and the required cooling plate block in your Simulink model and connect the thermal domain nodes of the two blocks.
3. Input the required **ThermalNodes** information into the cooling plate block. This information includes: number of nodes, 2-D location of nodes, and dimensions of nodes.

When you link a cooling plate to a battery block, the total length and width of the cooling plate are automatically fitted to that specific block.

The cooling plate linkage relies on the **Array-Of-Nodes**, a multi-dimensional or vectorized thermal domain connector. Vectorized thermal domain connectors facilitate the element-wise coupling of battery thermal models to the cooling plate components. Vectorized connections are necessary in the detailed thermal modeling of battery modules that contain many different parallel assemblies or cells.

For example, consider a module that contains six parallel assemblies with six cells in parallel. You can choose to thermally simulate this module using three thermal models by setting the **SeriesGrouping** property to `[1,4,1]`. In this case the length of the thermal node array is equal to 3. Alternatively, you can increase the model resolution to five thermal models by setting the **SeriesGrouping** property to `[1,1,2,1,1]`. Here, the length of the thermal node array increases to 5. The size of the **ThermalNodes** property changes to reflect this increased level of resolution. This also changes the area and location of the thermal nodes in the battery block. This figure shows the thermal linkage that occurs when you link this battery module to one of the cooling plates from the **Thermal** library in Simscape Battery.


Battery
Simulation Strategy
Cooling plate

Examples

Build Model of Hybrid-Cell Battery Pack

This example shows how to build a Simscape™ system model of a hybrid-cell battery pack with two sets of cell run-time parameters. The generated battery pack model contains two types of battery modules, each with different battery cell components inside. Use this example to analyze the performance effects of combining different battery cells within a single battery system, such as power capability versus range.

To create the system model of a battery pack, you must first create the `Cell`, `ParallelAssembly`, `Module`, and `ModuleAssembly` objects that comprise the battery pack, and then use the `buildBattery` function. The `buildBattery` function creates a library in your working folder that contains a system model block of a battery pack that you can use as reference in your simulations. The run-time parameters for these models, such as the battery cell impedance or the battery open-circuit voltage, are defined after the model creation and are therefore not covered by the Battery Pack Builder classes. To define the run-time parameters, you can either specify them in the block mask of the generated Simscape models or use the `MaskParameters` argument of the `buildBattery` function. If you specify the `MaskParameters` argument, the function also generates a parameterization script that helps you managing the run-time parameters of the different modules and cells inside the pack.

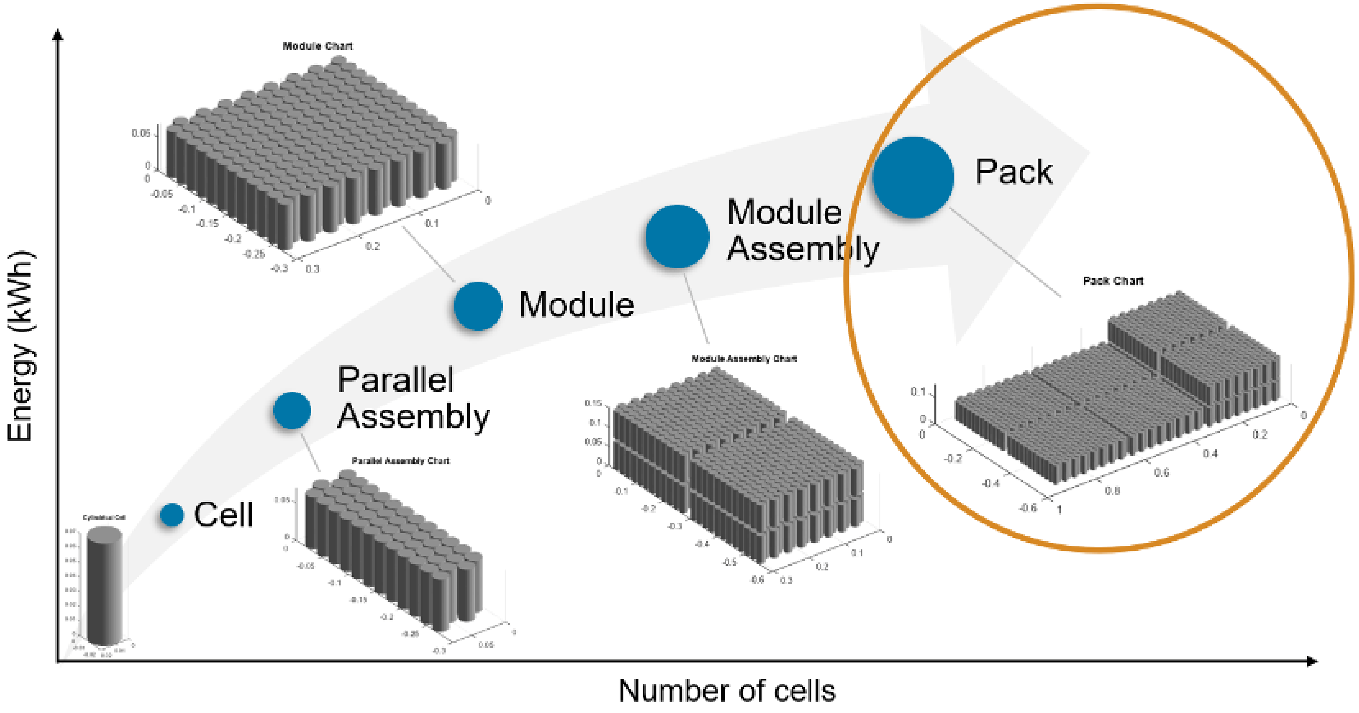
To use the functions and objects in Simscape Battery, first import the required Simscape Battery package:

```
import simscape.battery.builder.*
```

Create Battery Pack Object in MATLAB

Battery-based energy storage is a good option for integrating intermittent renewable energy sources into the grid. To create a battery pack, you must first design and create the foundational elements of the battery pack.

This figure shows the overall process to create a battery pack object in a bottom-up approach:



A battery pack comprises multiple module assemblies. These module assemblies, in turn, comprise a number of battery modules connected electrically in series or in parallel. The battery modules are made of multiple parallel assemblies which, in turn, comprise a number of battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement.

Create Cell Objects

To create the battery `Module` object, first create a Cell object of prismatic format.

```
prismaticGeometry = PrismaticGeometry(Height = simscape.Value(0.2,"m"),...
    Length = simscape.Value(0.35,"m"), Thickness =  simscape.Value(0.07,"m"));
```

The `PrismaticGeometry` object allows you to define the pouch geometrical arrangement of the battery cell. You can specify the height, length, and thickness of the cell by setting the `Height`, `Length`, and `Thickness` properties of the `PrismaticGeometry` object. For more information on the possible geometrical arrangements of a battery cell, see the `CylindricalGeometry` and `PouchGeometry` documentation pages.

Now use this `PrismaticGeometry` object to create a prismatic battery cell and assign its name.

```
batteryCell1 = Cell(Geometry = prismaticGeometry)
```

```
batteryCell1 = 
  Cell with properties:

            Geometry: [1×1 simscape.battery.builder.PrismaticGeometry]
    CellModelOptions: [1×1 simscape.battery.builder.CellModelBlock]
                Mass: [1×1 simscape.Value]

Show all properties
```

```
batteryCell1.Name = "CellChemistryType1";
```

To create a `Module` object with a different set of cell parameters, create a Cell object of prismatic format and change its name.

```
batteryCell2 = Cell(Geometry = prismaticGeometry)
```

```
batteryCell2 = 
  Cell with properties:

            Geometry: [1×1 simscape.battery.builder.PrismaticGeometry]
    CellModelOptions: [1×1 simscape.battery.builder.CellModelBlock]
                Mass: [1×1 simscape.Value]

Show all properties
```

```
batteryCell2.Name = "CellChemistryType2";
```

For more information, see the `Cell` documentation page.

Create ParallelAssembly Objects

A battery parallel assembly comprise multiple battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement. In this example, you create two parallel assemblies of one prismatic cell each.

To create the ParallelAssembly objects, use the `Cell` object you created before and specify the `NumParallelCells` property according to your design.

```
batteryParallelAssembly1 = ParallelAssembly(Cell = batteryCell1,...
    NumParallelCells = 1)
```

```
batteryParallelAssembly1 = 
  ParallelAssembly with properties:

    NumParallelCells: 1
                Cell: [1×1 simscape.battery.builder.Cell]
            Topology: "SingleStack"
                Rows: 1
     ModelResolution: "Lumped"

Show all properties
```

```
batteryParallelAssembly2 = ParallelAssembly(Cell = batteryCell2,...
    NumParallelCells = 1)
```

```
batteryParallelAssembly2 = 
  ParallelAssembly with properties:

    NumParallelCells: 1
                Cell: [1×1 simscape.battery.builder.Cell]
            Topology: "SingleStack"
                Rows: 1
     ModelResolution: "Lumped"

Show all properties
```

For more information, see the `ParallelAssembly` documentation page.

Create Module Objects

A battery module comprises multiple parallel assemblies connected in series. In this example, you create two battery modules of 4 parallel assemblies each, with an intergap between each assembly of 0.005 meters.

To create the `Module` object2, use the `ParallelAssembly` objects you created in the previous step and specify the `NumSeriesAssemblies` and `InterParallelAssemblyGap` properties.

```
batteryModule1 = Module(ParallelAssembly = batteryParallelAssembly1,...
    NumSeriesAssemblies = 4, ...
    InterParallelAssemblyGap = simscape.Value(0.005,"m"), ...
    ModelResolution = "Lumped", ...
    StackingAxis="X")
```

```
batteryModule1 = 
  Module with properties:

    NumSeriesAssemblies: 4
       ParallelAssembly: [1×1 simscape.battery.builder.ParallelAssembly]
        ModelResolution: "Lumped"
         SeriesGrouping: 4
       ParallelGrouping: 1

Show all properties
```

```
batteryModule2 = Module(ParallelAssembly = batteryParallelAssembly2,...
    NumSeriesAssemblies = 4, ...
    InterParallelAssemblyGap = simscape.Value(0.005,"m"), ...
    ModelResolution = "Lumped", ...
    StackingAxis="X")
```

```
batteryModule2 = 
  Module with properties:

    NumSeriesAssemblies: 4
       ParallelAssembly: [1×1 simscape.battery.builder.ParallelAssembly]
        ModelResolution: "Lumped"
         SeriesGrouping: 4
       ParallelGrouping: 1

Show all properties
```

For more information, see the `Module` documentation page.

Create ModuleAssembly Object

A battery module assembly comprises multiple battery modules connected in series or in parallel. The battery module assembly in this example comprises the two modules you created before twice in an alternating sequence. By default, the `ModuleAssembly` object electrically connects the modules in series.

To create the ModuleAssembly object, use the Module objects you created in the previous step and specify the InterModuleGap and NumLevels properties.

```
batteryModuleAssembly = ModuleAssembly(Module = [batteryModule1,batteryModule2,batteryModule1,bat
    InterModuleGap = simscape.Value(0.02,"m"), ...
    NumLevels = 1)
```

```
batteryModuleAssembly = 
  ModuleAssembly with properties:

    Module: [1×4 simscape.battery.builder.Module]

Show all properties
```

For more information, see the ModuleAssembly documentation page.

Create Pack Object

You now have all the foundational elements to create your hybrid battery pack. A battery pack comprises multiple module assemblies connected in series or in parallel. In this example, you create a battery pack of two module assemblies.

To create the Pack object, use the ModuleAssembly object you created in the previous step.

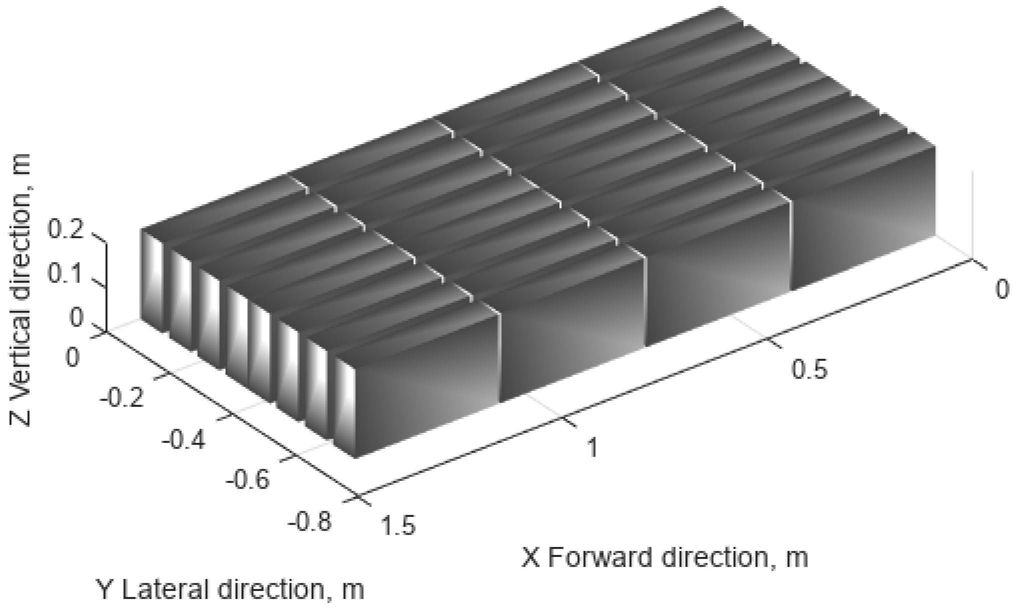
```
batteryPack = Pack(ModuleAssembly = repmat(batteryModuleAssembly,1,2),...
    StackingAxis="Y");
```

For more information, see the Pack documentation page.

Visualize Battery Pack

To visualize the battery pack before you build the system model, use the BatteryChart object. To add default axis labels to the battery plot, use the setDefaultLabels method of the BatteryChart object.

```
batteryPackChart = BatteryChart(Battery = batteryPack);
batteryPackChart.setDefaultLabels
```



For more information, see the `BatteryChart` documentation page.

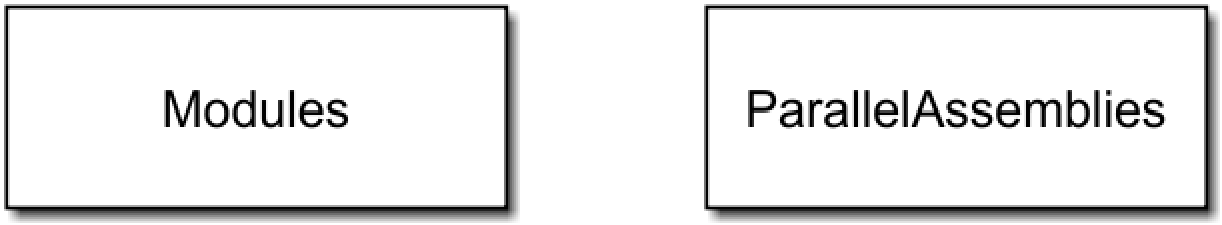
Build Simscape Model for Battery Pack Object

After you have created your battery objects, you need to convert them into Simscape models to be able to use them in block diagrams. You can then use these models as reference for your system integration and requirement evaluation, cooling system design, control strategy development, hardware-in-the-loop, and many more applications.

To create a library that contains the Simscape Battery model of the `Pack` object you created in this example, use the `buildBattery` function and set the `MaskInitialTargets` and `MaskParameters` arguments. The `MaskInitialTargets` and `MaskParameters` arguments allow you to choose between default numeric values or variable names for the parameters in each Module and Parallel Assembly block in the generated library. If you set these arguments to `"VariableNames"`, the function generates a script with all the run-time parameters and initial conditions required for simulation.

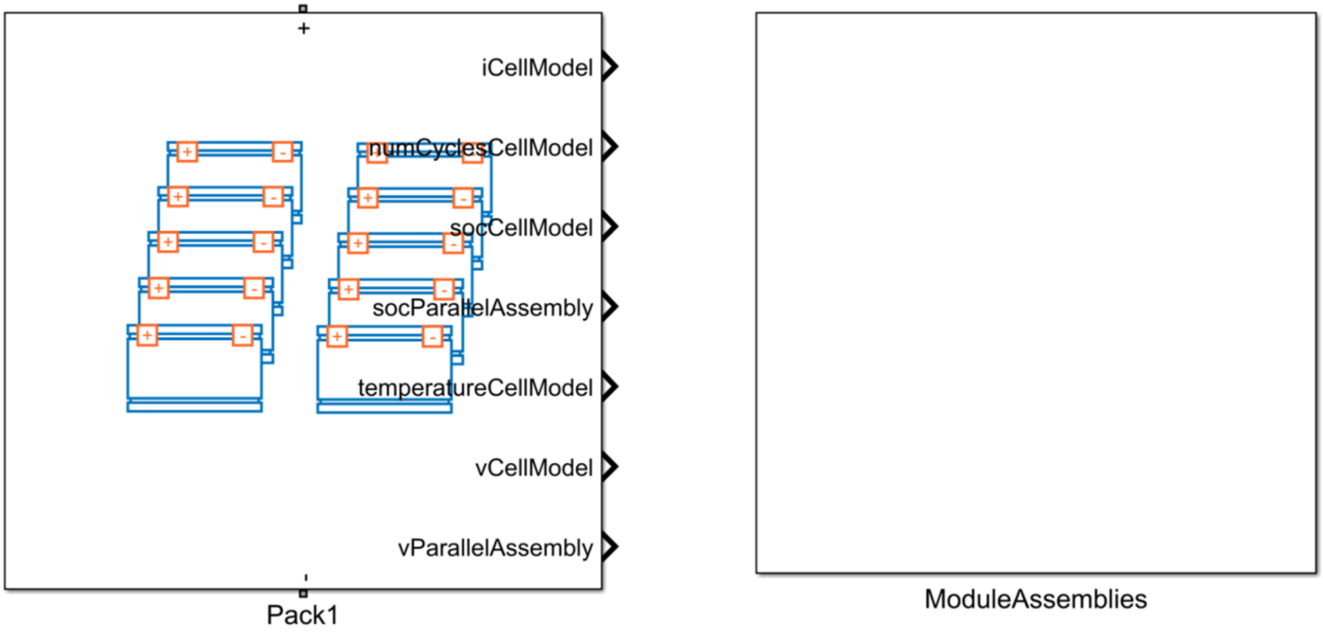
```
buildBattery(batteryPack,"LibraryName","hybridBatteryPack",...
    "MaskInitialTargets","VariableNames",...
    "MaskParameters","VariableNames")
```

The `buildBattery` function creates the `hybridBatteryPack_lib` and `hybridBatteryPack` SLX library files in your working directory. The `hybridBatteryPack_lib` library contains the Modules and ParallelAssemblies sublibraries.



To access the Simscape models of your `Module` and `ParallelAssembly` objects, open the `hybridBatteryPack_lib`. SLX file, double-click the sublibrary, and drag the Simscape blocks in your model.

The `hybridBatteryPack` library contains the Simscape models of your `ModuleAssembly` and `Pack` objects.



The Simscape models of your `ModuleAssembly` and `Pack` objects are subsystems. You can look inside these subsystems by opening the `hybridBatteryPack_lib` SLX file and double-click the subsystem.

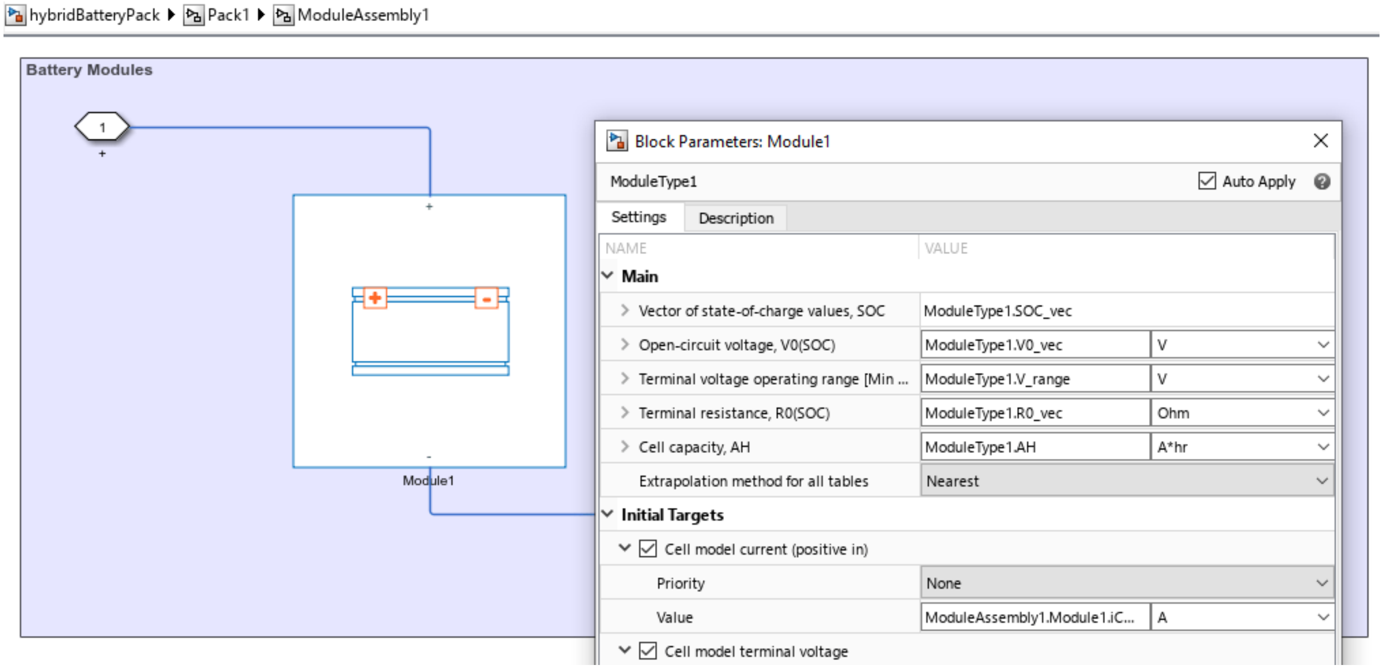
MaskParameters and MaskInitialTargets

The `MaskInitialTargets` and `MaskParameters` arguments allow you to choose between default numeric values or variable names for the parameters and initial conditions in each Module and Parallel Assembly block in the generated library.

By setting the `MaskParameters` argument to `VariableNames`, the `buildBattery` function generates a `hybridBatteryPack_param` M file where you can individually assign all the module and cell parameters, like the resistance, the open circuit voltage, and other parameters, for all the types of battery modules inside your battery pack. If you also set the `MaskInitialTargets` argument to `VariableNames`, then the generated M file contains the mask parameter definition at the beginning.

By setting the `MaskInitialTargets` argument to `VariableNames`, the `buildBattery` function generates a `hybridBatteryPack_param` M file where you can individually assign the initial temperature, state-of-charge, and other conditions, for all your battery modules in your battery pack. If you also set the `MaskParameters` argument to `VariableNames`, then the generated M file contains the initial targets definition at the end.

Check the effect of setting the `MaskParameters` and the `MaskInitialTargets` arguments to `VariableNames`. Open the `hybridBatteryPack_lib` SLX file and navigate to the ModuleAssembly1 subsystem by double-clicking the Pack1 subsystem. Double-click on the Module1 block to open the **Property Inspector**.


hybridBatteryPack ▸ Pack1 ▸ ModuleAssembly1
Battery Modules
1
+
+
-
Module1
Block Parameters: Module1
Settings
Description
NAME
VALUE

ModuleType1 ☑ Auto Apply

Main

NAME	VALUE	
Vector of state-of-charge values, SOC	ModuleType1.SOC_vec	
Open-circuit voltage, V0(SOC)	ModuleType1.V0_vec	V
Terminal voltage operating range [Min ...	ModuleType1.V_range	V
Terminal resistance, R0(SOC)	ModuleType1.R0_vec	Ohm
Cell capacity, AH	ModuleType1.AH	A*hr
Extrapolation method for all tables	Nearest	

Initial Targets

☑ Cell model current (positive in)

NAME	VALUE	
Priority	None	
Value	ModuleAssembly1.Module1.iC...	A

☑ Cell model terminal voltage

A specific variable name is associated to the values of each parameter in the **Main** section of the Module1 block. You can then easily specify these values inside the `hybridBatteryPack_param` M file without having to change them inside the model by opening the **Property Inspector** of each block individually.

Protect Battery During Charge and Discharge for Electric Vehicle

This example shows how to efficiently charge and discharge a battery for an electric vehicle (EV) and handle battery faults. Portable electronics and EV widely use li-ion batteries as power source due to their high-energy density. But li-ion batteries also have safety issues due to extreme conditions such as over-discharge, over-charge, high temperature, and low temperature. These extreme conditions can damage the battery and effect its performance in the long term. In some cases they can cause loss of stability which leads to thermal runaway.

In this example, two circuit breakers connect the positive terminal and the negative terminal of the battery to the load circuit. Two more circuit breakers connect or disconnect the charging or discharging circuits. A basic control strategy operates these circuit breakers to put the battery in charge or discharge mode and to disconnect the battery during fault conditions.

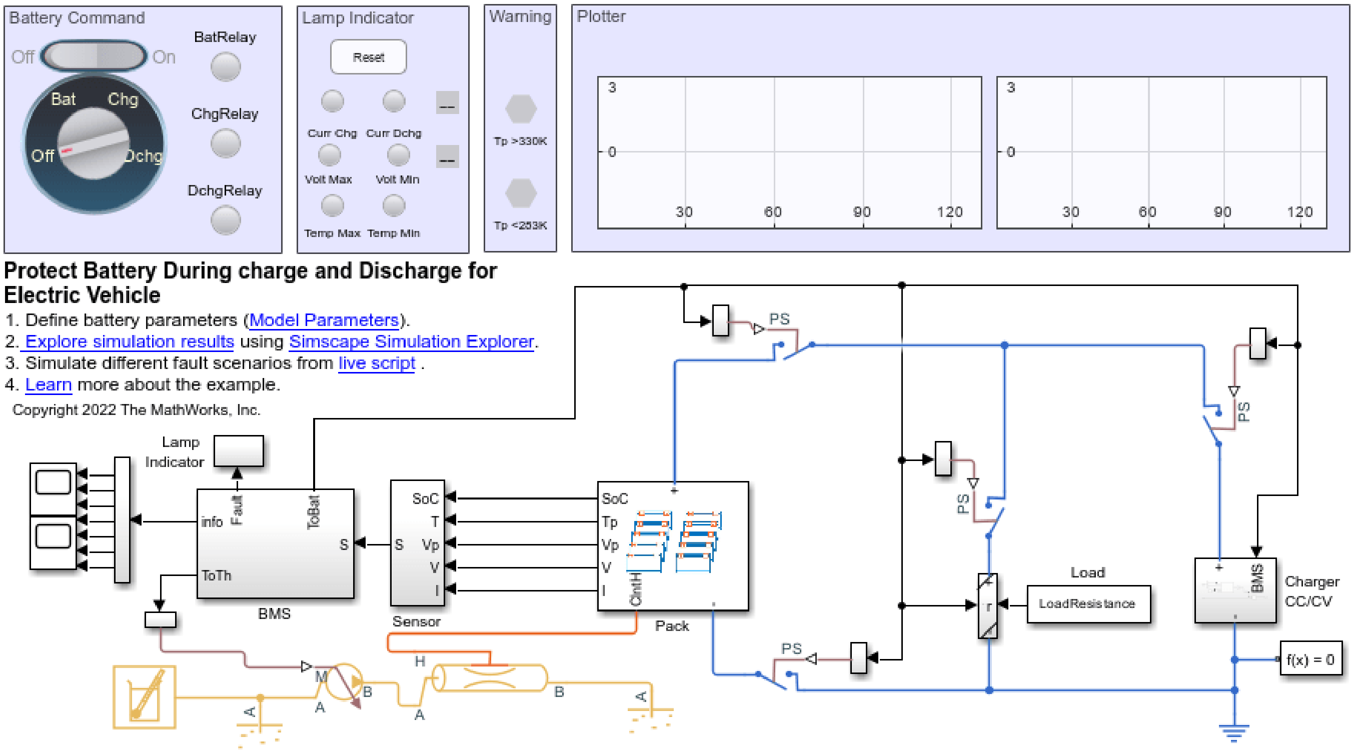
Model Overview

Open the model `batt_BatteryManagementSystem.slx`

```
clearvars
disp('Start battery plant model simulations workflow');
```

```
Start battery plant model simulations workflow
```

```
% Open the model
open_system('batt_BatteryManagementSystem.slx');
```



Protect Battery During charge and Discharge for Electric Vehicle

1. Define battery parameters (Model Parameters).
2. Explore simulation results using Simscape Simulation Explorer.
3. Simulate different fault scenarios from live script .
4. Learn more about the example.

Copyright 2022 The MathWorks, Inc.

Load parameter files

```
batt_BatteryManagementSystem_param; % Load model parameters
batt_packBTMSExampleLib_param; % Load battery pack parameters
```

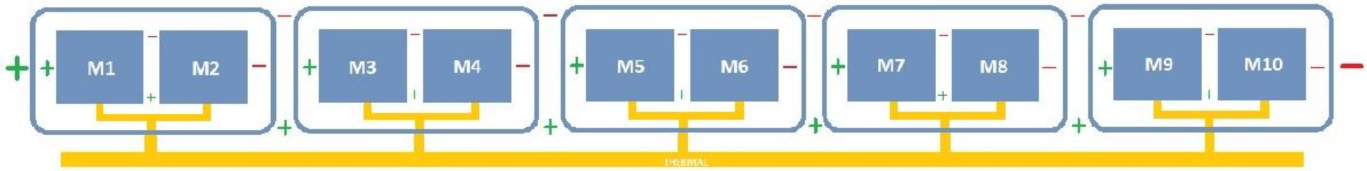
Load data for `LoadResistance` and `LoadCurrent` for the drive run

```
load('batt_BatteryManagementSystem_Drive.mat');
```

Battery Module

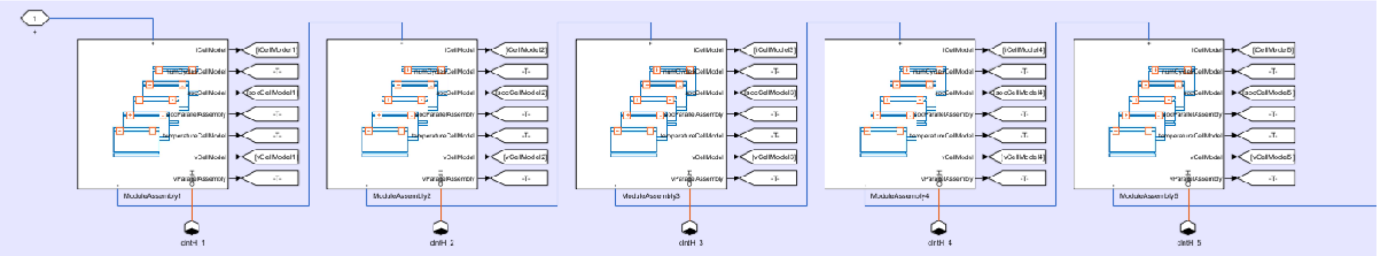
The battery comprises a battery pack of 400V, generally used in electric vehicles. Since a single cell cannot provide such voltage or power levels, multiple cells are connected in series and parallel to create the desired battery pack. The battery pack in this example comprises 10 modules, each with 11 series-connected parallel sets (p-sets). Each p-set comprises three cells in series. All modules are connected in series to form a pack of 330 cells.

To create the module used in the battery pack of this example, see the “Build Model of Battery Module with Thermal Effects” on page 4-103 example.



Open the `pack` subsystem.

```
 % Show battery Pack
open_system('batt_BatteryManagementSystem/Pack','force')
```

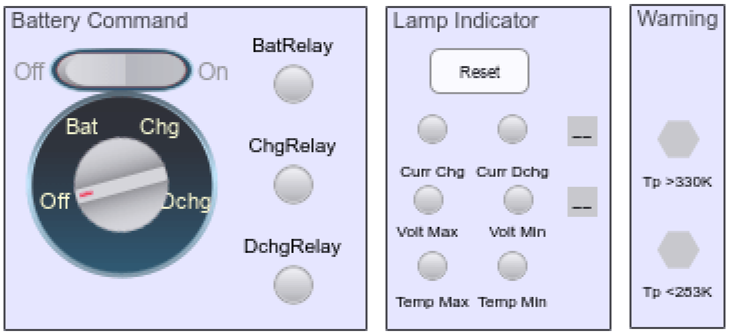


Mode Control Dashboard

In an electric vehicle, you can control the charging and discharging operations of the battery.

- To start the car, the key is turned which connects the battery circuit breakers and connects the battery to the system of the car.
- While driving, the battery is in discharge mode.
- When you connect the car to a charger, the battery is in charging mode.

In a car, the discharging and charging modes are mutually exclusive. This example emulates this scenario by implementing a charging control dashboard in the model, called Battery Command. This dashboard comprises a rotary switch for manual operations, an on-off switch for automatic operations, and indication lamps.



Use the rotary switch to choose between the charging and discharging modes manually. The position of the rotary switch affects the battery mode:

- **Off** — The battery is disconnected.
- **Bat** — The battery is connected.
- **Chg** — The battery is charging.
- **Dchg** — The battery is discharging.

Use the on-off switch to switch between modes automatically by setting the switch to On and by specifying the BatCmd variable. When the BatCmd variable is equal to:

- **0** — The battery is disconnected.
- **1** — The battery is connected.
- **2** — The battery is charging.
- **3** — The battery is discharging.

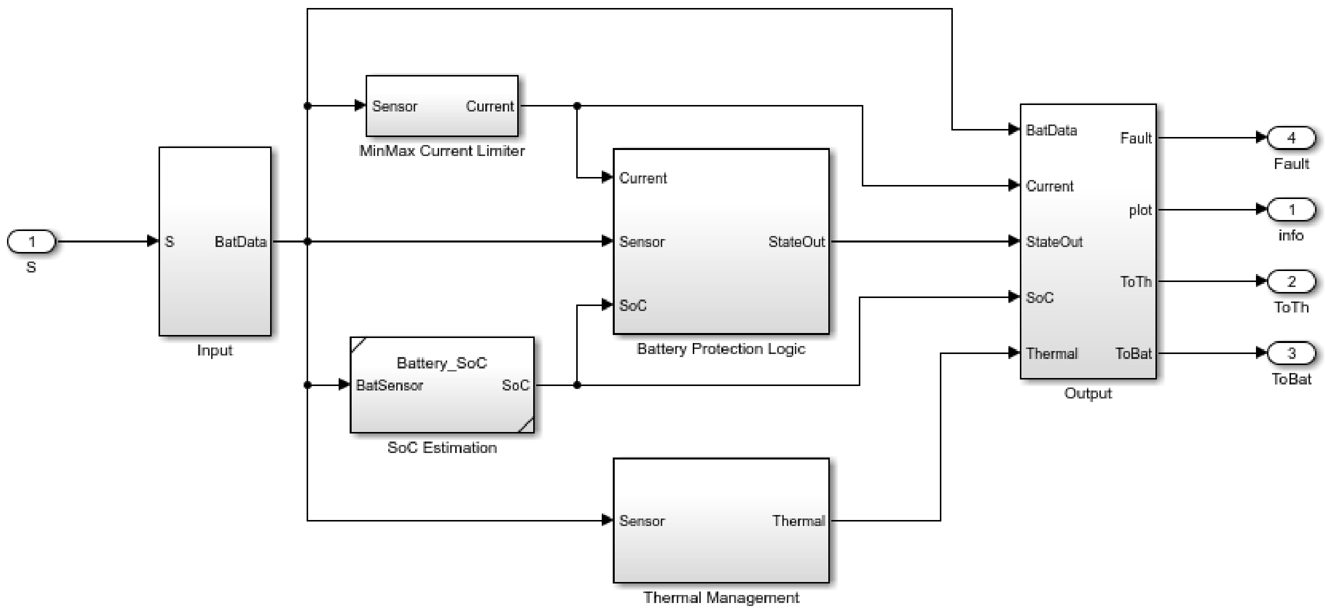
The indication lamps show which mode the battery is currently operating in. When the lamps are red, the specific mode is off. When the lamps are green, the specific mode is on. The model also contains indication lamps that track fault appearances and a Reset button to reset all the faults to zero for testing purposes. A red lamp indicates the presence of a fault.

Battery Management System

The battery management system (BMS) manages all the battery operations and keeps it within operational limits. The BMS maintains the current, voltage, and temperature of the pack within safe limits during the charging and discharging operations. In this example, the BMS controls the circuit breakers to protect the battery pack based on the pack sensor data and on estimated parameters such as the state-of charge (SOC) and the discharge and charge current limits. For temperature control, the BMS controls the flow of coolant by using an "On-Off" flow control block.

To open the BMS subsystem, at the MATLAB Command Window, enter:

```
 % Show battery BMS
open_system('batt_BatteryManagementSystem/BMS','force')
```



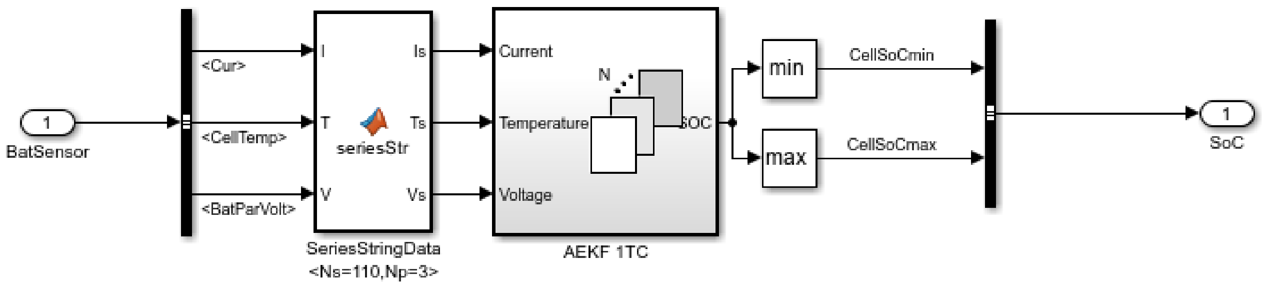
The BMS in this example comprises four different components: SoC estimation, MinMax Current Limiter, Thermal Management, and Battery Protection Logic.

SoC Estimation

A battery SOC provides the remaining charge left inside the battery. This value is an estimation based on many different parameters. There are different ways to estimate the SOC of a battery. This example uses an extended Kalman filter estimation strategy.

To open the SoC Estimation subsystem, at the MATLAB Command Window, enter:

```
% Show SoC estimator
open_system('batt_BatteryManagementSystem/BMS/SoC Estimation','force')
```

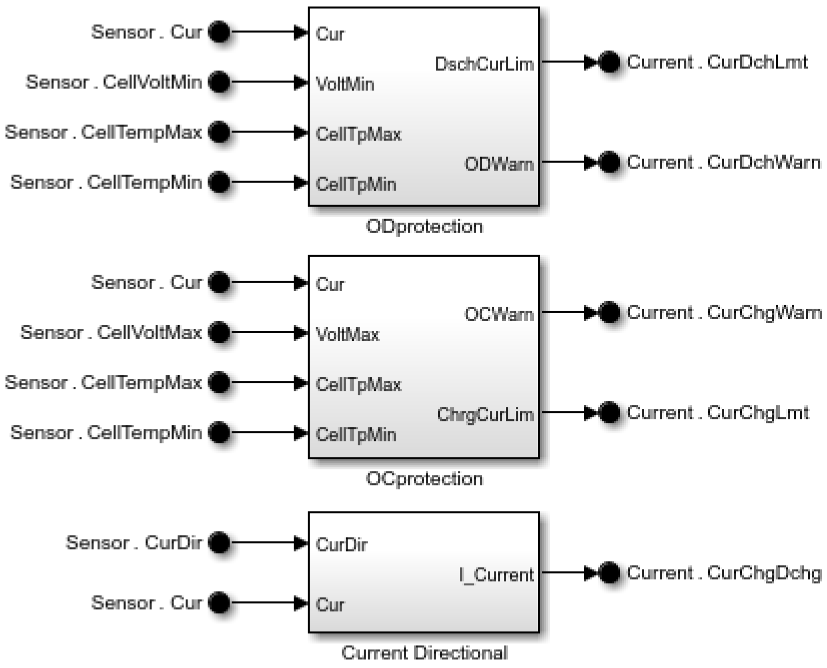


Current Limiter

The current state of the battery, such as the battery voltage and temperature, defines the over-discharge and over-charge current limits of the battery for protection of the pack. For example, while discharging, if the temperature is high, you must reduce the current that the electric vehicle withdraws from the battery. If the voltage of the battery is low and the vehicle withdraws too much current from the battery, it can cause damage to the cells and must be limited.

To open the Current Limiter subsystem, at the MATLAB Command Window, enter:

```
% Show Current limit calculation
open_system('batt_BatteryManagementSystem/BMS/MinMax Current Limiter','force')
```



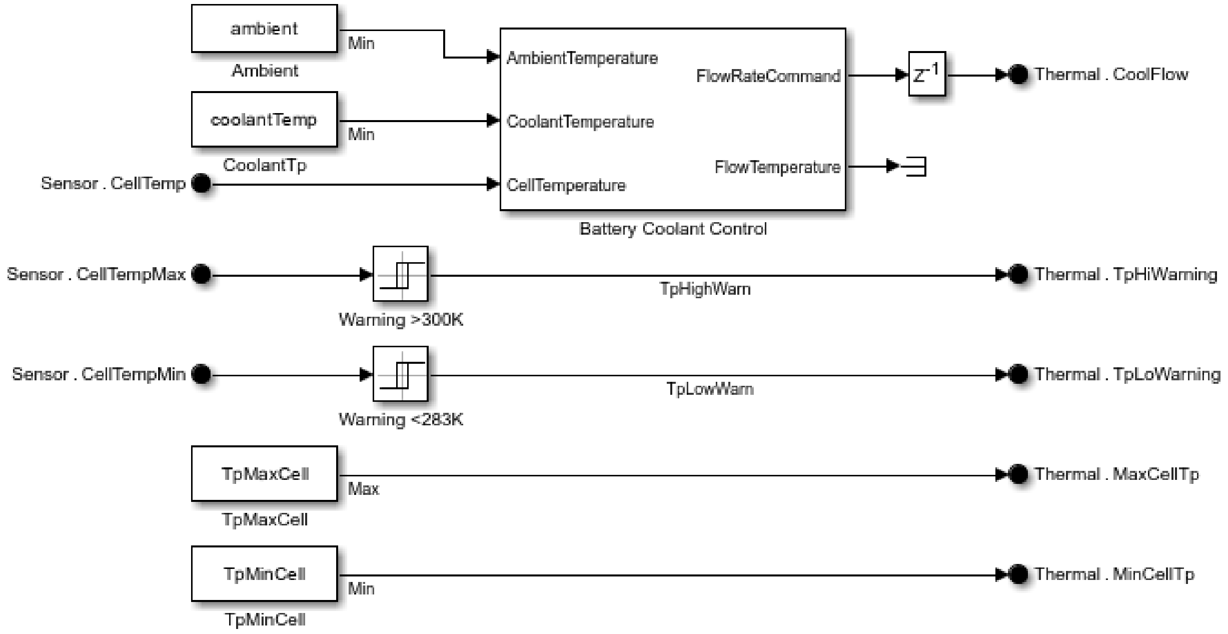
Thermal Control

For a safe operation of the battery, the battery temperature must be within specified limits. To control the temperature, you must remove the excess heat by using a coolant circuit. The flow of the coolant controls how much heat you can remove from the battery pack. In this example, an on-off control block manages the coolant circuit.

If the temperature is greater than a threshold value, the pump switches on. When the temperature is lower than the lowest threshold value, the pump switches off.

To open the Thermal Management subsystem, at the MATLAB Command Window, enter:

```
% Show Thermal management control
open_system('batt_BatteryManagementSystem/BMS/Thermal Management','force')
```

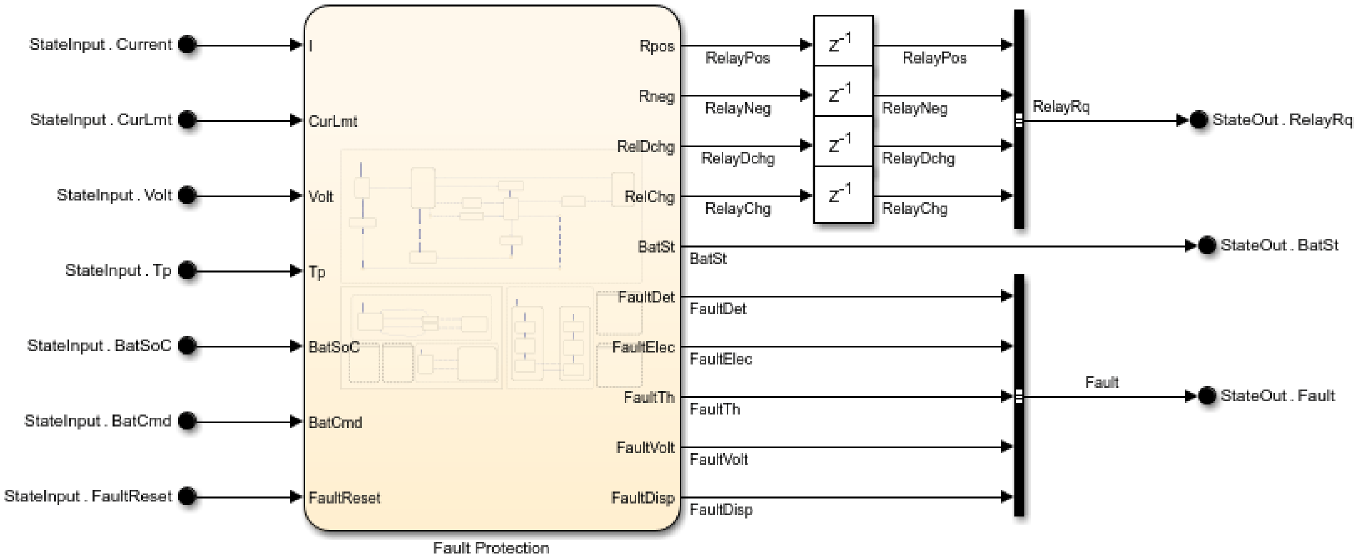


Battery Protection Logic

The battery protection logic is a state-flow logic that takes the battery parameters, sensor data, and user input from the charging control dashboard to generate the signal for the relay operation, state of the battery, and fault analysis of the battery.

To open the Battery Protection Logic subsystem, at the MATLAB Command Window, enter:

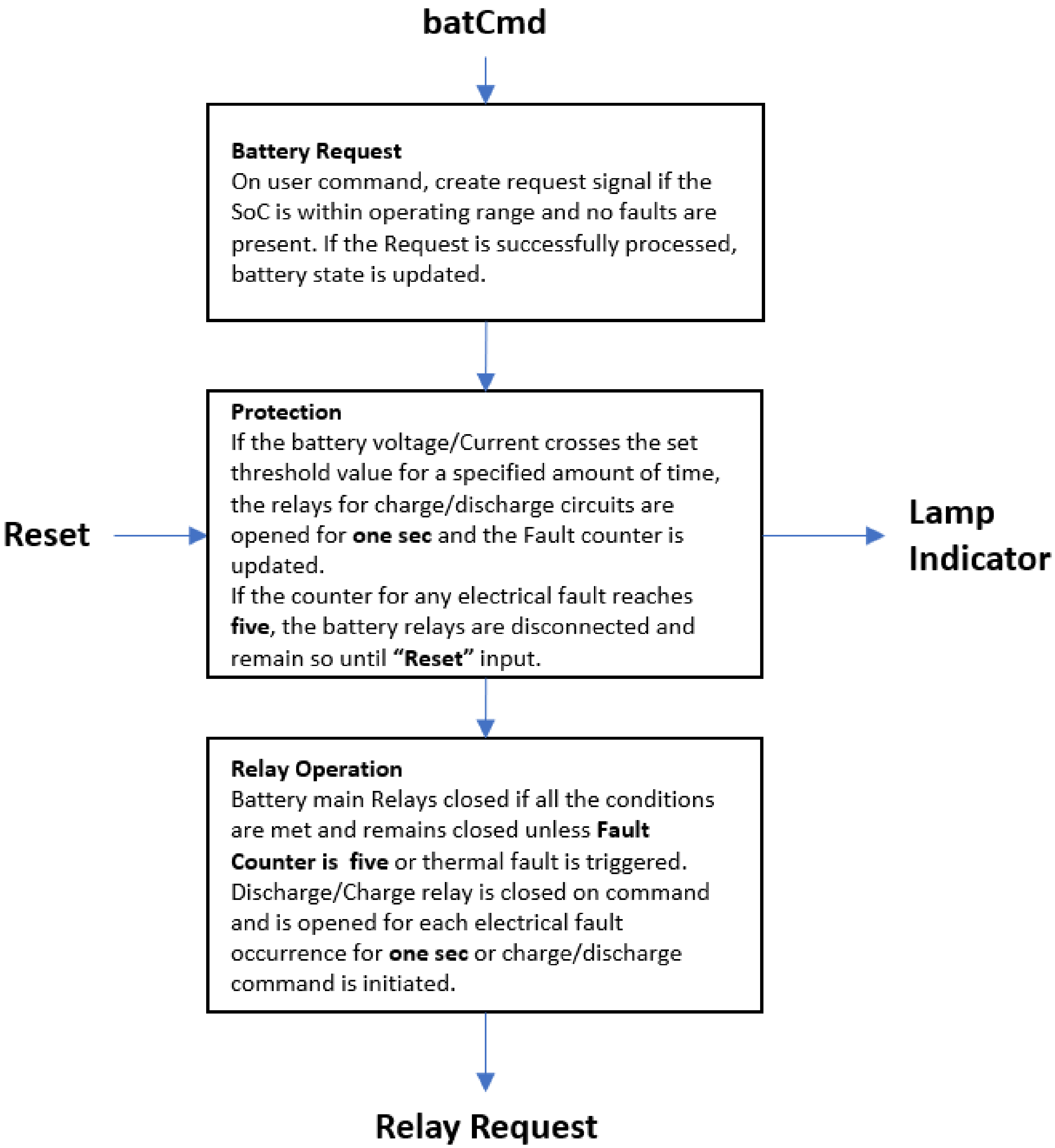
```
% Show protection stateflow
open_system('batt_BatteryManagementSystem/BMS/Battery Protection Logic/State Flow' ...
    ,'force')
```



For fault protection, a counter records the triggering of current and voltage faults. When there are more than five faults, the battery protection logic disconnects the battery from the load until you reset the count. The counter does not record thermal faults. The battery is disconnected at the first thermal fault appearance. While discharging, if the battery SOC is lower than a specific limit, the protection logic disconnects the battery. While charging, if the battery SoC is greater than the upper threshold, the protection logic disconnects the battery from the charging circuit.

This flowchart shows the logic inside Fault Protection stateflow block. The logic follows these steps:

- Battery Request — Put the battery in ideal, charge, or discharge mode according to the received input.
- Protection — Check if the battery parameter (Current, Voltage and Temperature) crosses the threshold and generate faults.
- Relay Operation — Operate the battery, charge, and discharge relays based on the request and fault status.



Battery Request
On user command, create request signal if the SoC is within operating range and no faults are present. If the Request is successfully processed, battery state is updated.

Protection
If the battery voltage/Current crosses the set threshold value for a specified amount of time, the relays for charge/discharge circuits are opened for **one sec** and the Fault counter is updated.
If the counter for any electrical fault reaches **five**, the battery relays are disconnected and remain so until **"Reset"** input.

Relay Operation
Battery main Relays closed if all the conditions are met and remains closed unless **Fault Counter is five** or thermal fault is triggered.
Discharge/Charge relay is closed on command and is opened for each electrical fault occurrence for **one sec** or charge/discharge command is initiated.

Fault Simulations

Battery faults occur when the battery is put in extreme scenarios.

Open the model `batt_BatteryManagementSystem.slx`

```
open_system('batt_BatteryManagementSystem/','force')
```

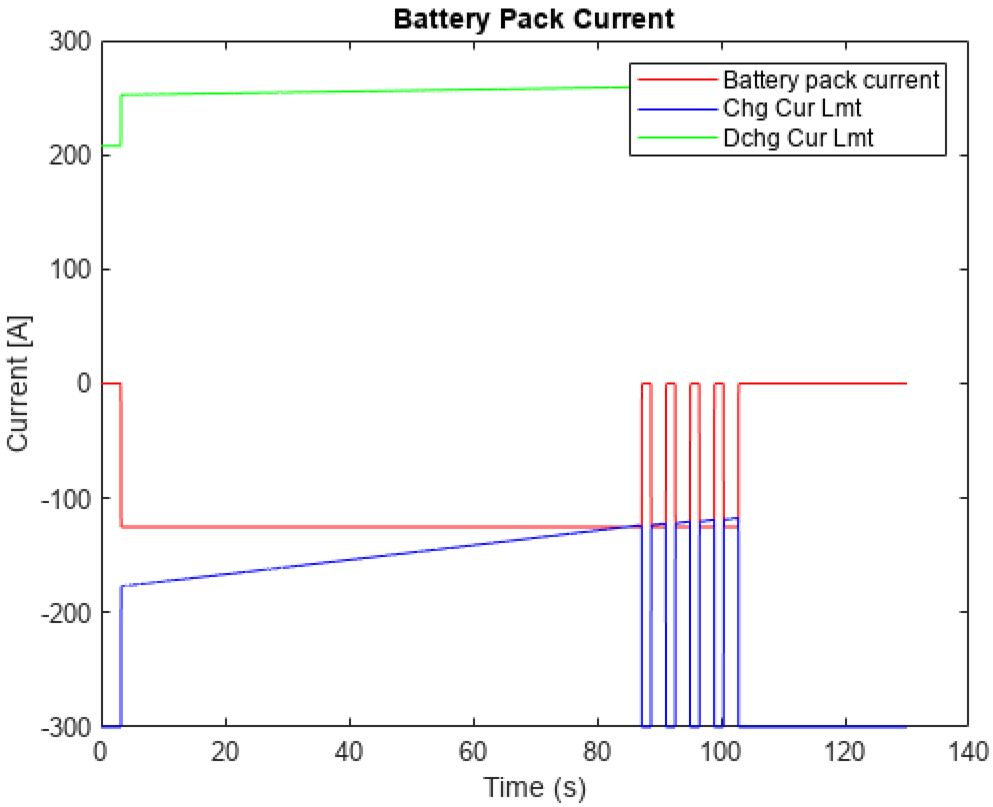
Fault During Battery Charging

In charge mode, a battery can experience these faults:

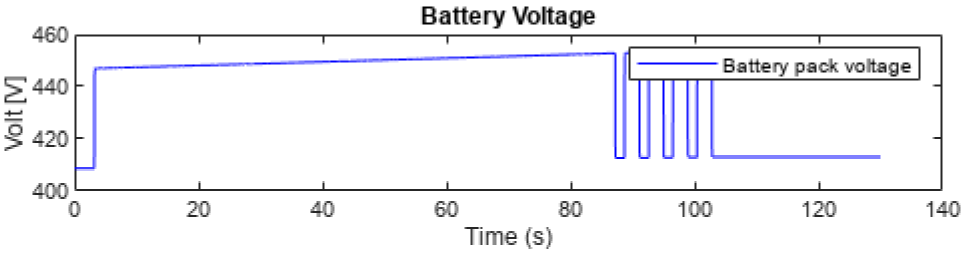
- Overvoltage fault — An incompatible device charges the battery beyond its rated voltage.
- Overcurrent fault — A current higher than the allowed limit charges the battery.

While charging, as the voltage and temperature of the battery increase, the charging current limit decreases. If the current limit goes below the charging current, a charging fault triggers and the charging circuit is disconnected from the battery for protection. After five fault occurrences, the battery circuit breakers disconnect for the rest of the simulation.

```
batt_BatteryManagementSystem_param; % Load Simulation parameters
ChargerCC_A = 125; % charger max charging current (Ah)
initialPackSOC = 0.5; % Initial SoC of the pack set to low
BatCmdData=timeseries([0;0;1;1;2;2],[0;1.99;2;2.99;3;130],'Name','BatCmdData'); % Battery input
batt_packBTMSExampleLib_param; % Load battery parameters with the new data
sim('batt_BatteryManagementSystem.slx'); % Simulate the model
%% Plot for comparison
% Plot for current
figure
plot(logsout_batt_BatteryManagementSystem.get("CurDisp").Values,'r-');
hold on
plot(logsout_batt_BatteryManagementSystem.get("<CurChgLmt>").Values  ,'b-');
plot(logsout_batt_BatteryManagementSystem.get("<CurDchgLmt>").Values  ,'g-');
hold off;legend('Battery pack current','Chg Cur Lmt','Dchg Cur Lmt');
xlabel('Time (s)');ylabel('Current [A]');title('Battery Pack Current');
```

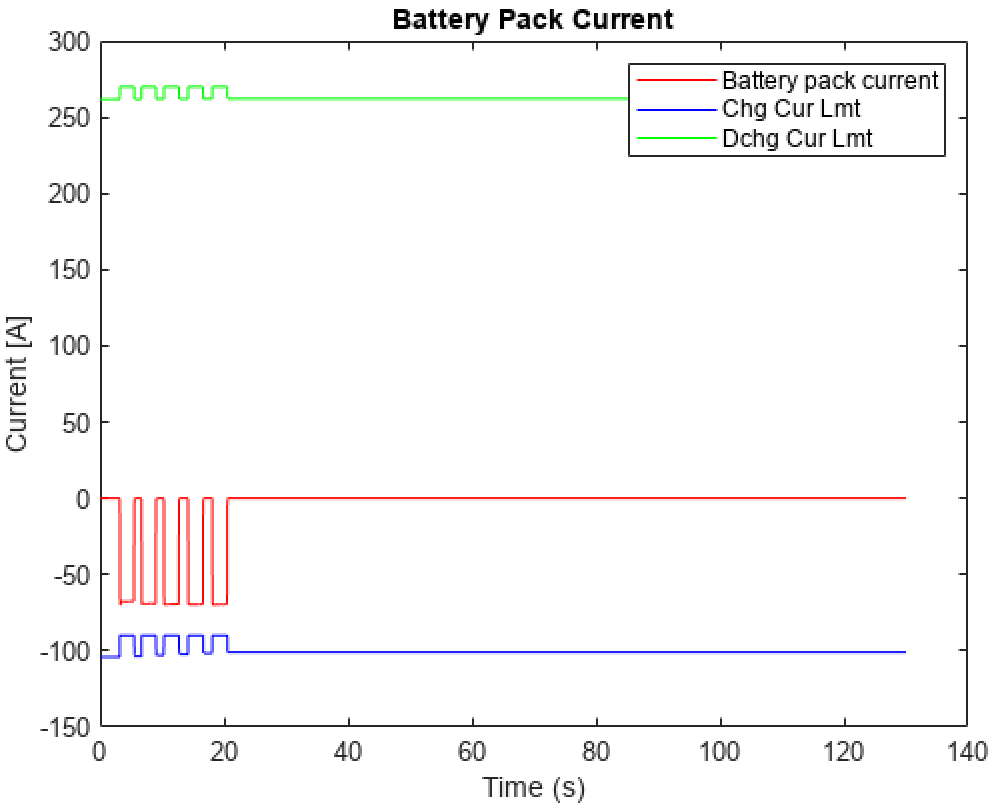


```
%% Plot Voltage
figure
simlog_handles(2) = subplot(3, 1, 2);
plot(logsout_batt_BatteryManagementSystem.get("<BatVolt>").Values,'b-');
legend('Battery pack voltage');
xlabel('Time (s)');ylabel('Volt [V]');title('Battery Voltage');
```

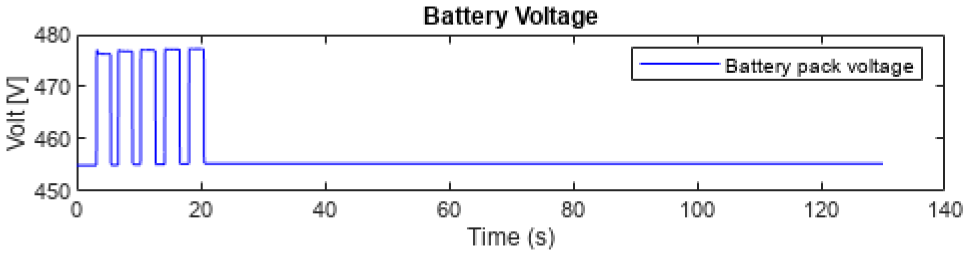


For higher values of SOC, the cell voltage is closer to the full charge voltage. A high charging current can overcharge the battery or increase the battery voltage too much, which triggers an overvoltage fault. After five fault occurrences, the battery circuit breakers disconnects for rest of the simulation.

```
batt_BatteryManagementSystem_param; % Load Simulation parameters
MaxVoltLmt = 4.2; % cell max voltage limit
ChargerCC_A = 70; % charger max charging current
initialPackSOC = 0.95; % Initial SoC of the pack set to high
BatCmdData=timeseries([0;0;1;1;2;2],[0;1.99;2;2.99;3;130],'Name','BatCmdData'); % Battery input
batt_packBTMSExampleLib_param; % Load battery parameters with the new data
sim('batt_BatteryManagementSystem.slx'); % Simulate model
%% Plot for comparison
% Plot for current
figure
plot(logsout_batt_BatteryManagementSystem.get("CurDisp").Values,'r-');
hold on
plot(logsout_batt_BatteryManagementSystem.get("<CurChgLmt>").Values  ,'b-');
plot(logsout_batt_BatteryManagementSystem.get("<CurDchgLmt>").Values  ,'g-');
hold off;legend('Battery pack current','Chg Cur Lmt','Dchg Cur Lmt');
xlabel('Time (s)');ylabel('Current [A]');title('Battery Pack Current');
```



```
%% Plot Voltage
figure
simlog_handles(2) = subplot(3, 1, 2);
plot(logsout_batt_BatteryManagementSystem.get("<BatVolt>").Values,'b-');
legend('Battery pack voltage');
xlabel('Time (s)');ylabel('Volt [V]');title('Battery Voltage');
```



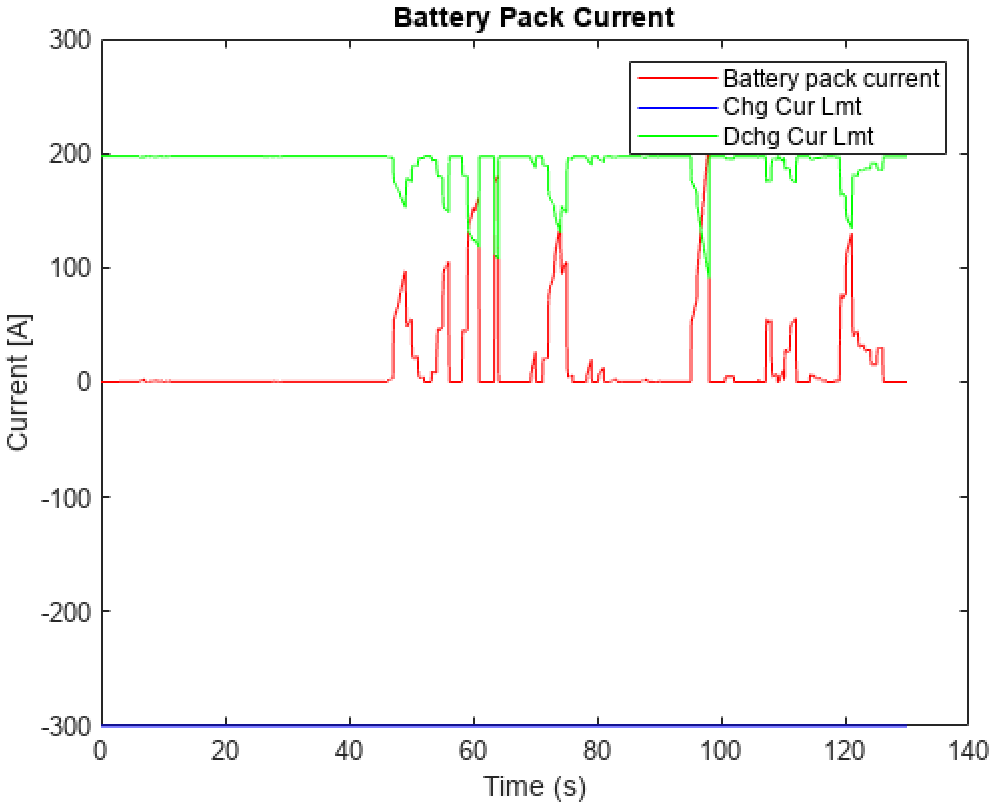
Fault During Battery Discharging

In discharge mode, a battery can experience these faults:

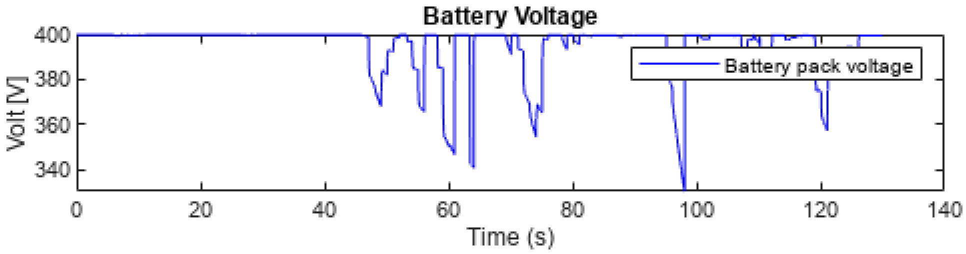
- Undervoltage fault — The battery discharges beyond its minimum rated voltage.
- Overcurrent fault — A current higher than the allowed limit discharges the battery.

While discharging, as the battery voltage decreases and the battery temperature increases, the discharging current limit decreases. If the current limit goes below the discharging current, a discharging fault triggers. If the cell voltage goes below the minimum voltage limit, a voltage fault triggers. For any of these faults the discharging circuit is disconnected from the battery for protection. After five fault occurrences, the battery circuit breakers disconnect for the rest of the simulation.

```
batt_BatteryManagementSystem_param; % Load Simulation parameters
load('batt_BatteryManagementSystem_Drive.mat') % load drive data
MinVoltLmt=3.2; % battery minimum voltage limit
initialPackSOC = 0.25; % Initial SoC of the pack set to low
BatCmdData=timeseries([0;0;1;1;3;3],[0;1.99;2;2.99;3;130],'Name','BatCmdData'); % Battery input
batt_packBTMSExampleLib_param; % Load battery parameters with the new data
sim('batt_BatteryManagementSystem.slx'); % Simulate model
%% Plot for comparison
% Plot for current
figure
plot(logsout_batt_BatteryManagementSystem.get("CurDisp").Values,'r-');
hold on
plot(logsout_batt_BatteryManagementSystem.get("<CurChgLmt>").Values  ,'b-');
plot(logsout_batt_BatteryManagementSystem.get("<CurDchgLmt>").Values  ,'g-');
hold off;legend('Battery pack current','Chg Cur Lmt','Dchg Cur Lmt');
xlabel('Time (s)');ylabel('Current [A]');title('Battery Pack Current');
```



```
%% Plot Voltage
figure
simlog_handles(2) = subplot(3, 1, 2);
plot(logsout_batt_BatteryManagementSystem.get("<BatVolt>").Values,'b-');
legend('Battery pack voltage');
xlabel('Time (s)');ylabel('Volt [V]');title('Battery Voltage');
```



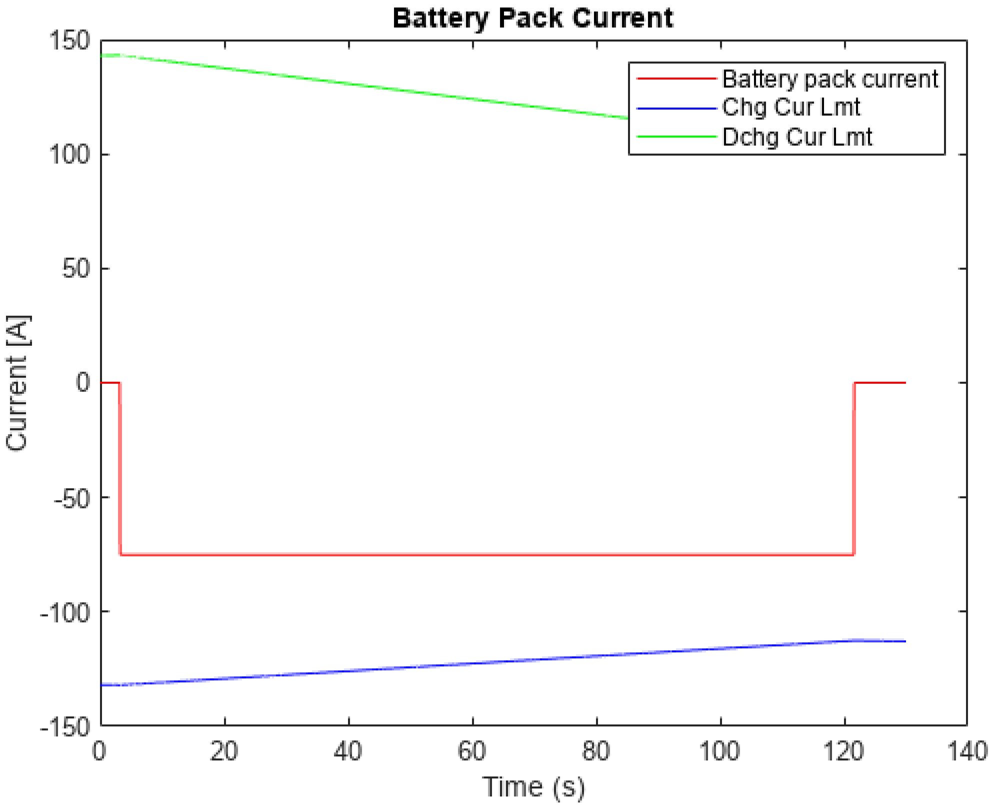
Battery Thermal Fault

Thermal faults trigger if the battery temperature goes beyond the safe operating range. A simple on-off strategy controls the flow of coolant in the thermal circuit to manage the battery temperature.

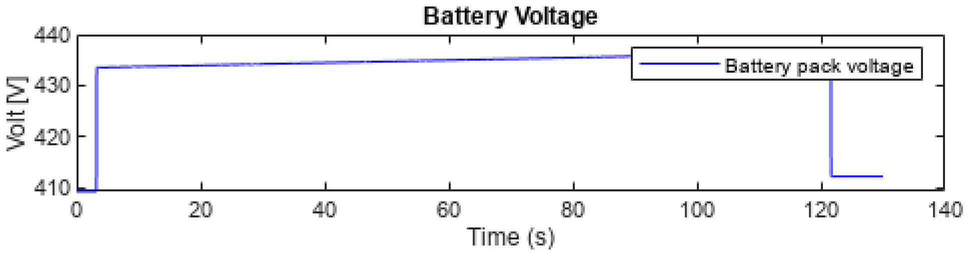
To simulate a thermal fault, this example first turns off the coolant control so that the temperature in the battery is unregulated. The initial temperature of the battery is high. A high current charges the battery and brings its temperature to values beyond the safe operating range. This triggers the thermal fault and the battery circuit breakers disconnect for the rest of the simulation.

```
batt_BatteryManagementSystem_param; % Load Simulation parameters
% Temperature parameter to "switch on" flow for thermal control
CoolantSwitchOnTp = MaxThLmt+5; % switch on temp set to 338.15 K (65 deg C)
% "Switch on" temperature for coolant flow set five degrees more than the max allowed temperatur
initialBattTemp=328.15; % Initial temperature set to high(K) - 55 deg Celcius
ChargerCC_A = 75; % charger max charging current
initialPackSOC = 0.5; % Initial SoC of the pack set to low
BatCmdData=timeseries([0;0;1;1;2;2],[0;1.99;2;2.99;3;130],'Name','BatCmdData'); % Battery input
batt_packBTMSExampleLib_param; % Load battery parameters with the new data
sim('batt_BatteryManagementSystem.slx'); % Simulate the model
%% Plot for comparison
% Plot for current
figure
plot(logsout_batt_BatteryManagementSystem.get("CurDisp").Values,'r-');
hold on
plot(logsout_batt_BatteryManagementSystem.get("<CurChgLmt>").Values  ,'b-');
plot(logsout_batt_BatteryManagementSystem.get("<CurDchgLmt>").Values  ,'g-');
```

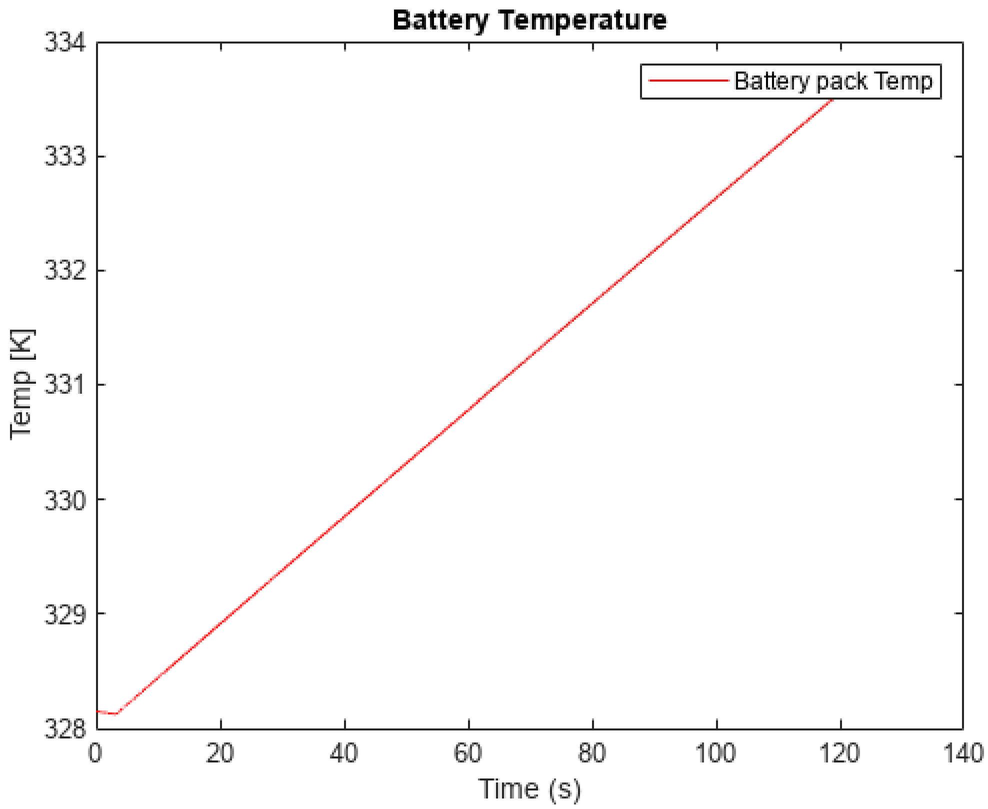
```
hold off;legend('Battery pack current','Chg Cur Lmt','Dchg Cur Lmt');
xlabel('Time (s)');ylabel('Current [A]');title('Battery Pack Current');
```



```
%% Plot Voltage
figure
simlog_handles(2) = subplot(3, 1, 2);
plot(logsout_batt_BatteryManagementSystem.get("<BatVolt>").Values,'b-');
legend('Battery pack voltage');
xlabel('Time (s)');ylabel('Volt [V]');title('Battery Voltage');
```



```
%% Plot for Temp
figure
plot(logsout_batt_BatteryManagementSystem.get("<TpMax>").Values,'r-')
legend('Battery pack Temp');xlabel('Time (s)');ylabel('Temp [K]');
title('Battery Temperature')
```


Battery Temperature
Battery pack Temp
334
333
332
331
330
329
328
Temp [K]
0
20
40
60
80
100
120
140
Time (s)

Peak Shaving with Battery Energy Storage System

Introduction

This example shows how to model a battery energy storage system (BESS) controller and a battery management system (BMS) with all the necessary functions for the peak shaving. The peak shaving and BESS operation follow the IEEE Std 1547-2018 and IEEE 2030.2.1-2019 standards.

In this example, an average converter, an output filter, and associated control model the BESS. The BESS can operate in grid-forming control and it receives setpoint from the operator control room for power dispatch. The BESS also receives the powerflow measurements from point of common coupling (PCC) and changes cotrol mode for peak shaving.

Description of BESS controller

The BESS controller receives commands and setpoint from the control room operator as well as various measurements and status from different sources and loads connected to the feeder. The BESS in this model comprises these functions:

1. Reference frequency generation
2. Reference voltage generation
3. Receive setpoint and command from operator
4. Change control mode. According to the powerflow measurement at PCC, the BESS starts peak shaving or enables to charging mode

Implementation of Photovoltaic (PV) model

The model represents a three-phase grid-connected photovoltaic (PV) system that injects power with unity power factor (UPF) without using an intermediate DC-DC converter. The transformerless configuration simulates leakage currents. To track the maximum power point (MPP), the example uses these maximum power point tracking (MPPT) techniques:

- Incremental conductance
- Perturbation and observation

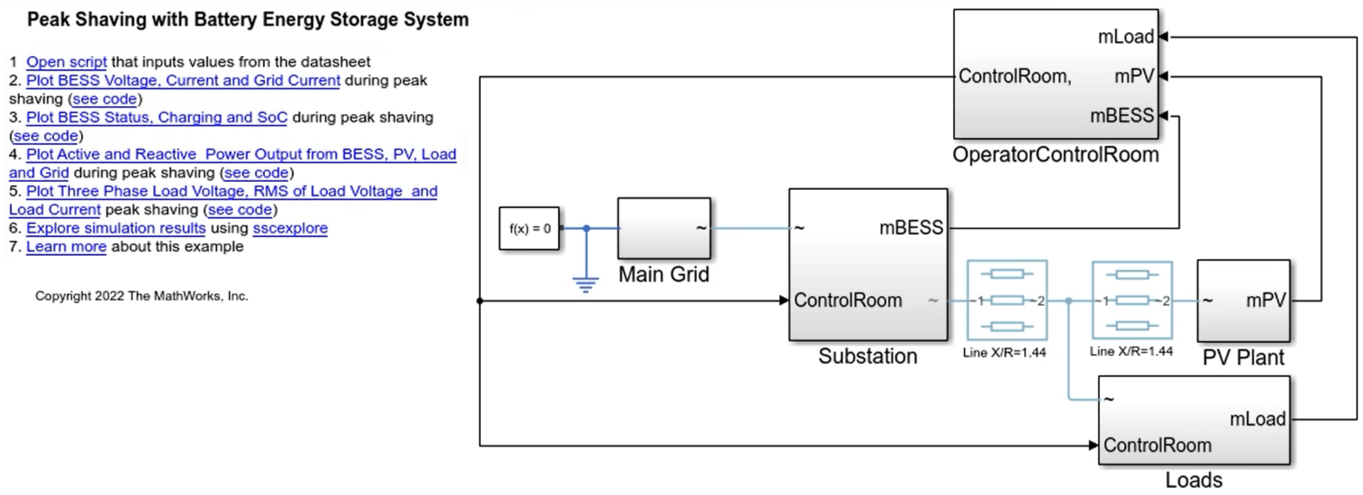
Build Model for BESS Peak Shaving

Model Overview

Open the model `sscv_peak_shaving.slx`.

```
mdl = "sscv_peak_shaving";
open_system(mdl)
```

The Substation subsystem connects the BESS and the feeder to the main grid. This subsystem comprises a connecting breaker, disconnectors, and transformers to connect the main grid to the BESS and the outgoing feeder. The substation also contains the BESS controller and the BMS.



Peak Shaving with Battery Energy Storage System

1 Open script that inputs values from the datasheet
2. Plot BESS Voltage, Current and Grid Current during peak shaving (see code)
3. Plot BESS Status, Charging and SoC during peak shaving (see code)
4. Plot Active and Reactive Power Output from BESS, PV, Load and Grid during peak shaving (see code)
5. Plot Three Phase Load Voltage, RMS of Load Voltage and Load Current peak shaving (see code)
6. Explore simulation results using sscexplore
7. Learn more about this example

Copyright 2022 The MathWorks, Inc.

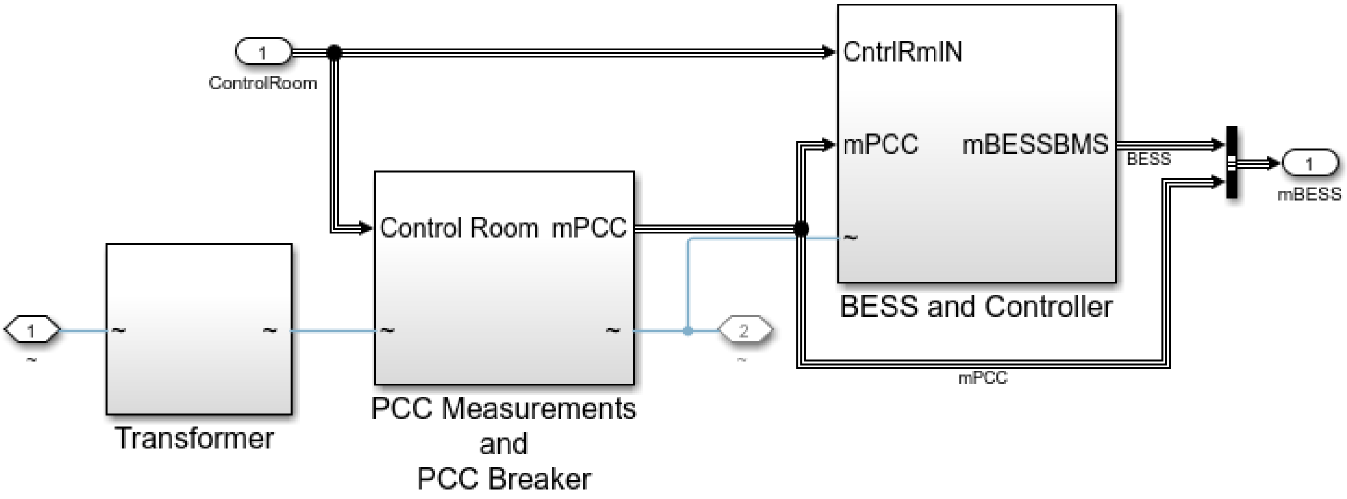
Building Components for Peak Shaving with BESS

This example comprises these main components:

1. Substation
2. BESS System
3. Battery Management System (BMS)
4. Battery Module
5. Operator Control Room

Substation

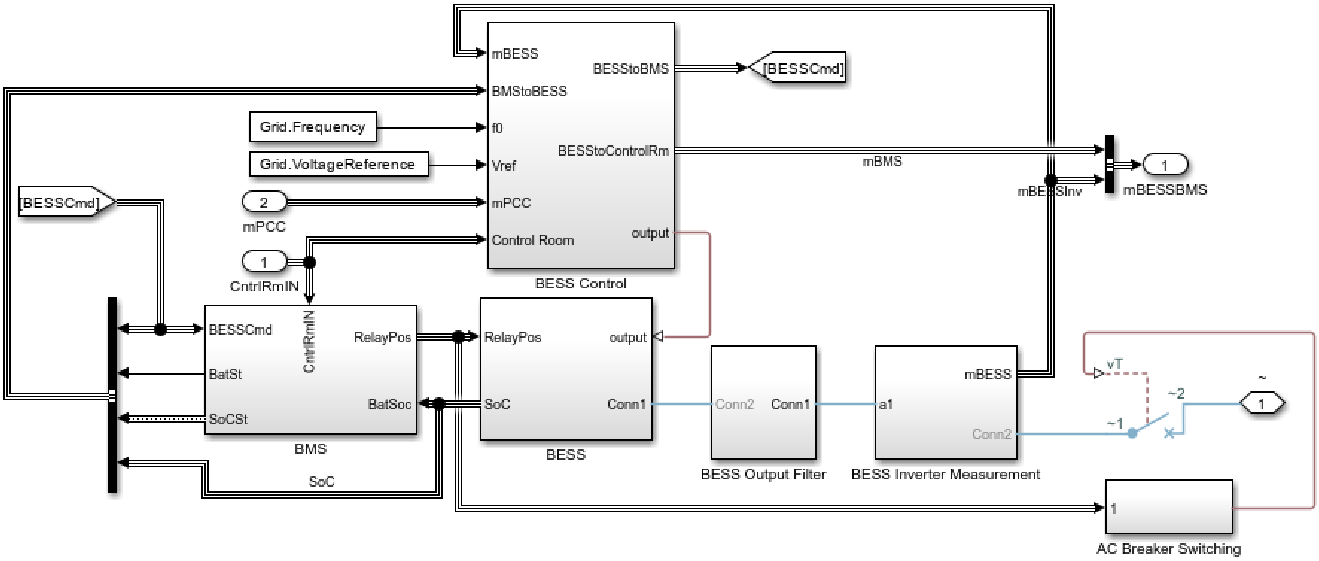
The Substation subsystem connects the BESS and the feeder to the main grid by using a connecting breaker, disconnectors, and transformers. The substation also contains the BESS controller and the BMS.



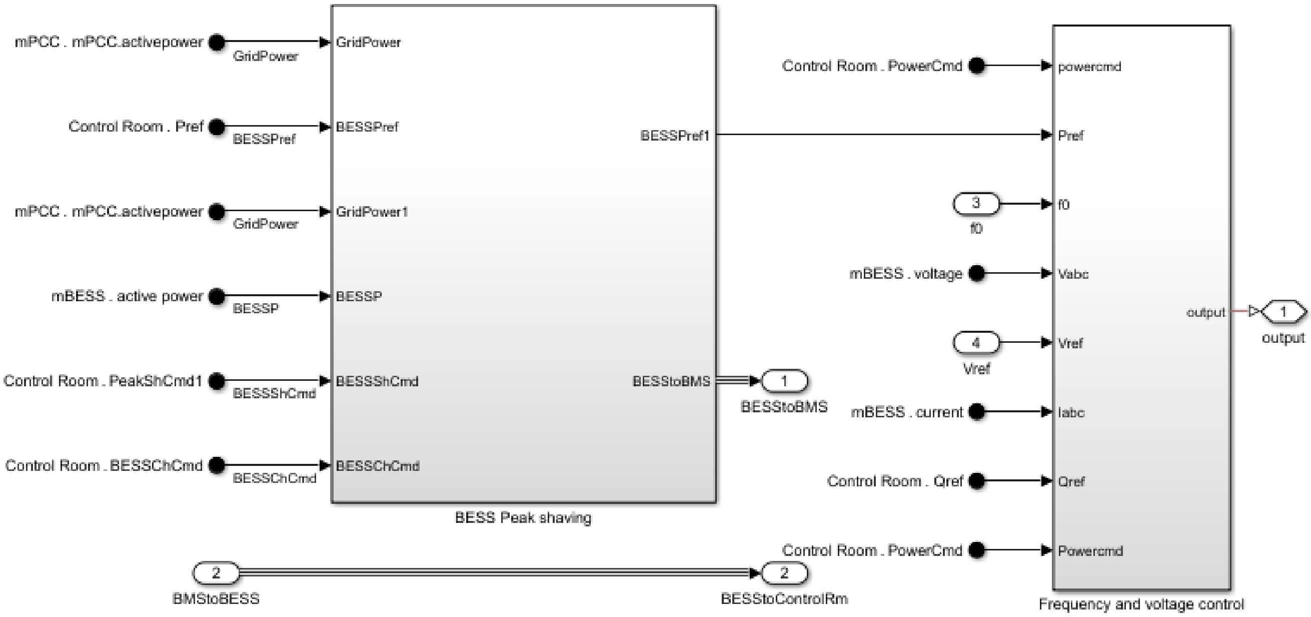
BESS System

The BESS system comprises:

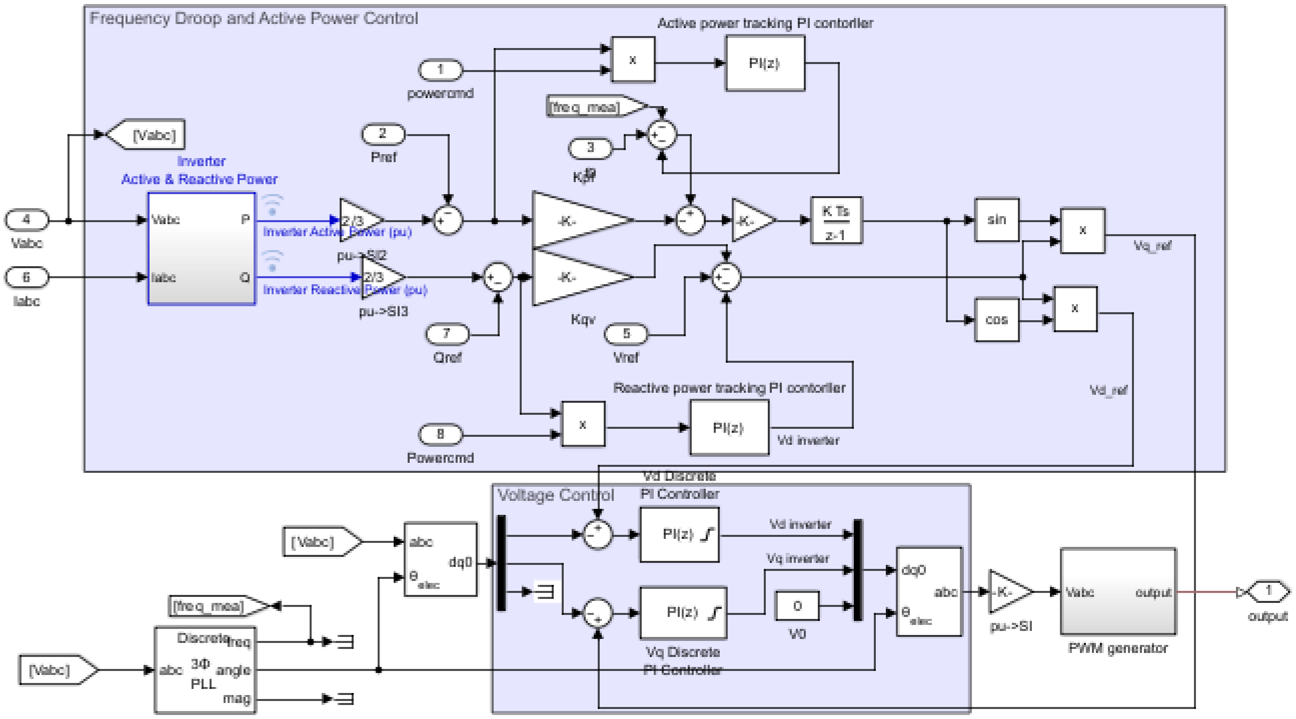
1. Grid side converter, filter, measurement, and control
2. Battery management system (BMS)
3. Battery Module



The BESS converter connects the battery modules to the grid and controls the power flow through the converter. The BESS controller implements the peak shaving function.

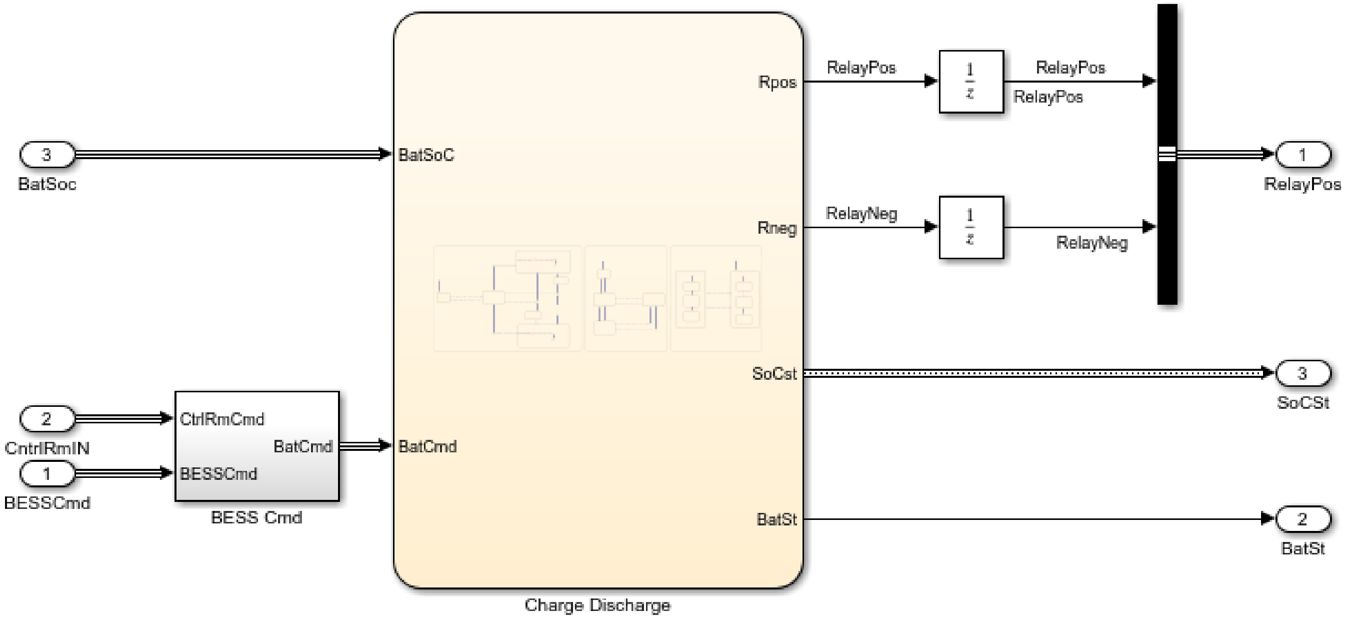


The power measurement at PCC detects high loading of the main grid at the substation and activates the peak shaving function. The peak shaving function limits the power from the main grid to the maximum rated power while the BESS system provides the rest of the power requirement.



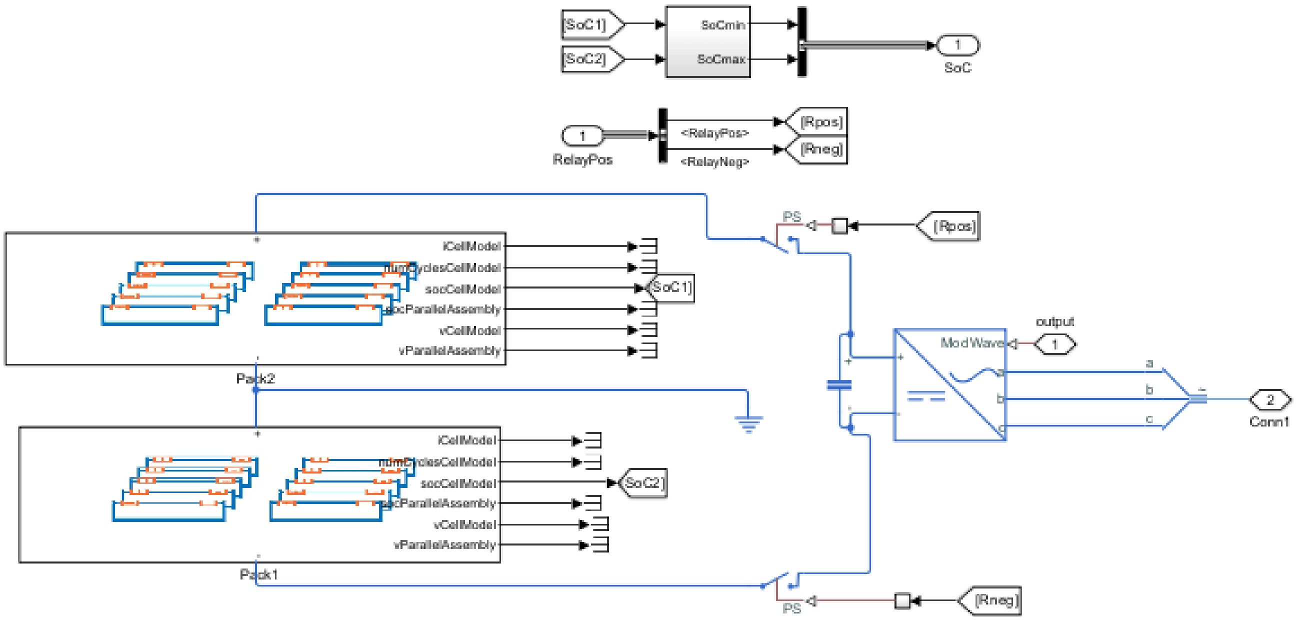
Battery Management System (BMS)

The BMS receives the request from the grid-side converter on power requirement. The BMS also monitors the state-of-charge (SOC) of the battery module. In this example, the BMS disconnects the battery if the SOC is above the high SOC threshold and the battery is discharging. Similarly the BMS disconnects the battery if the SOC is below the low SOC threshold and the battery is charging. Once the battery opens from the DC side, the AC-side breaker also opens within one cycle.



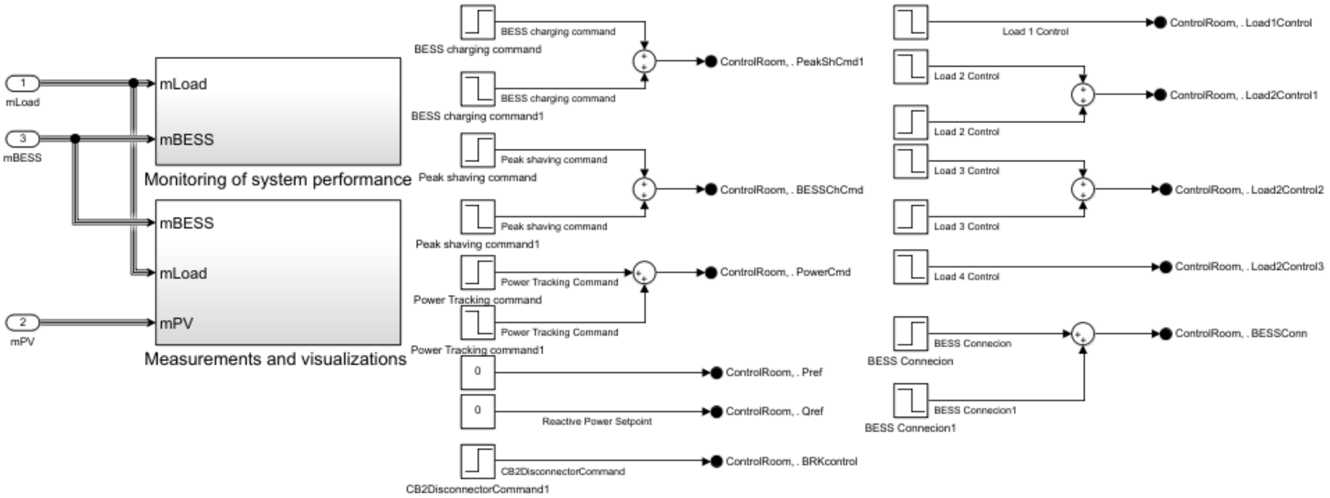
Battery Module

The battery module is connected to the DC side of the BESS converter. Two battery packs are connected in series and grounded at the midpoint.The DC breakers can disconnect the battery module.



Operator Control Room

The Operator Control Room subsystem sends all the setpoints and commands. It also plots the measured quantities and the system performance analysis.



Define Parameters & Run Simulations

Initialize the BESS, grid, and PV parameters. At the MATLAB Command Window, enter:

```
run("sscv_peak_shaving_BESS_data.mlx");
```

Initialize Battery parameters

The battery module in this example is generated by using the objects and functions in the Battery Pack Model Builder. For more information on how to build a battery pack, see the “Build a Simple Model of a Battery Pack in MATLAB and Simscape” on page 4-134 example.

```
run("sscv_peak_shaving_param.m");
Ns=1500/25;
Np=round(150*1000/(59*Ns*25));
load('sscv_peak_shaving_data.mat')
```

Run Simulation

Simulate the model.

```
run("sscv_peak_shaving");
```

Plot Simulation Results

These plots show:

1. Voltage and current of BESS.
2. Active and reactive power output of BESS, PV, load, and main grid.
3. Voltage, current, and power consumption of loads.
4. Status, discharge, charge, and SOC of BESS.

This plot shows the three-phase voltage and current output of the BESS, as well as the grid current during peak shaving and BESS disconnection.

```
run('sscv_peak_shaving_plot_BESS_VI.m')
```

The plot shows the measured values around the start of peak shaving around 3.0 s and the BESS disconnection at 4.97 s. A stable voltage and current output from BESS verifies a good peak shaving. The disconnection of BESS happens due to low SOC.

This plot shows the active and reactive power of BESS, PV, main grid, and loads.

```
run('sscv_peak_shaving_plot_PQ.m')
```

The stable active and reactive power output verifies the efficacy of the peak shaving method.

This plot shows the voltage and current at the loads.

```
run('sscv_peak_shaving_plot_Load_VI.m')
```

The load voltage and load current remain steady during peak shaving and BESS disconnection.

This plot shows the charge, discharge, BESS status, and SOC of the BESS.

```
run('sscv_peak_shaving_plot_BMS_SoC.m')
```

The discharge status during peak shaving and the disconnection of the BESS due to low SOC matches with

the results from the AC-side output. This also validates the BMS functions for BESS SOC monitoring.

Evaluate System Performance

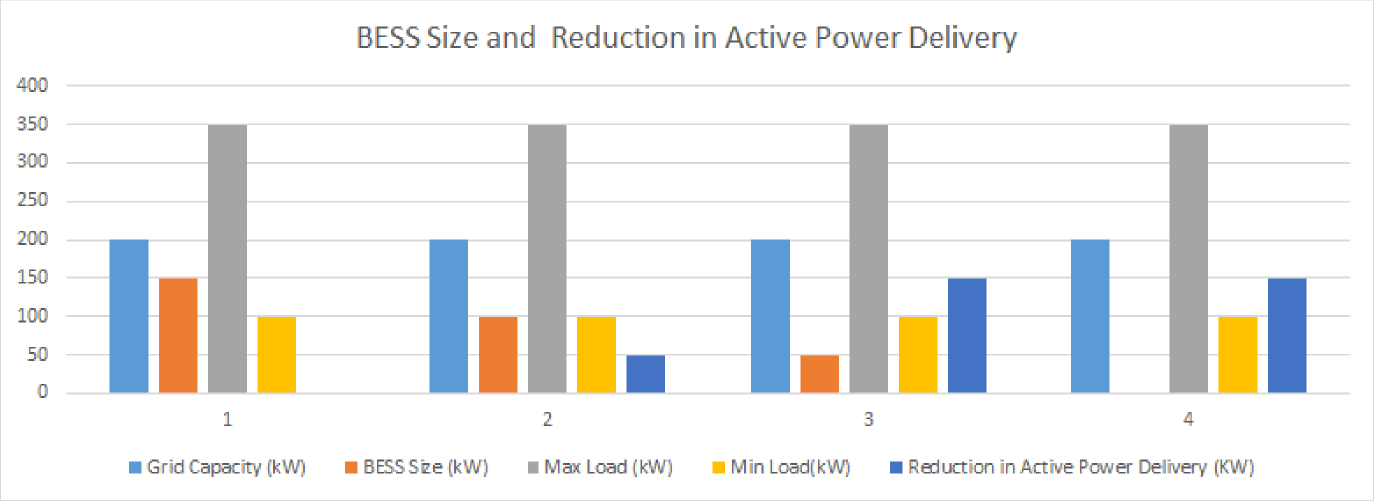
These plots show the results of the system performance and the impact of the peak shaving function.

These performance indices include:

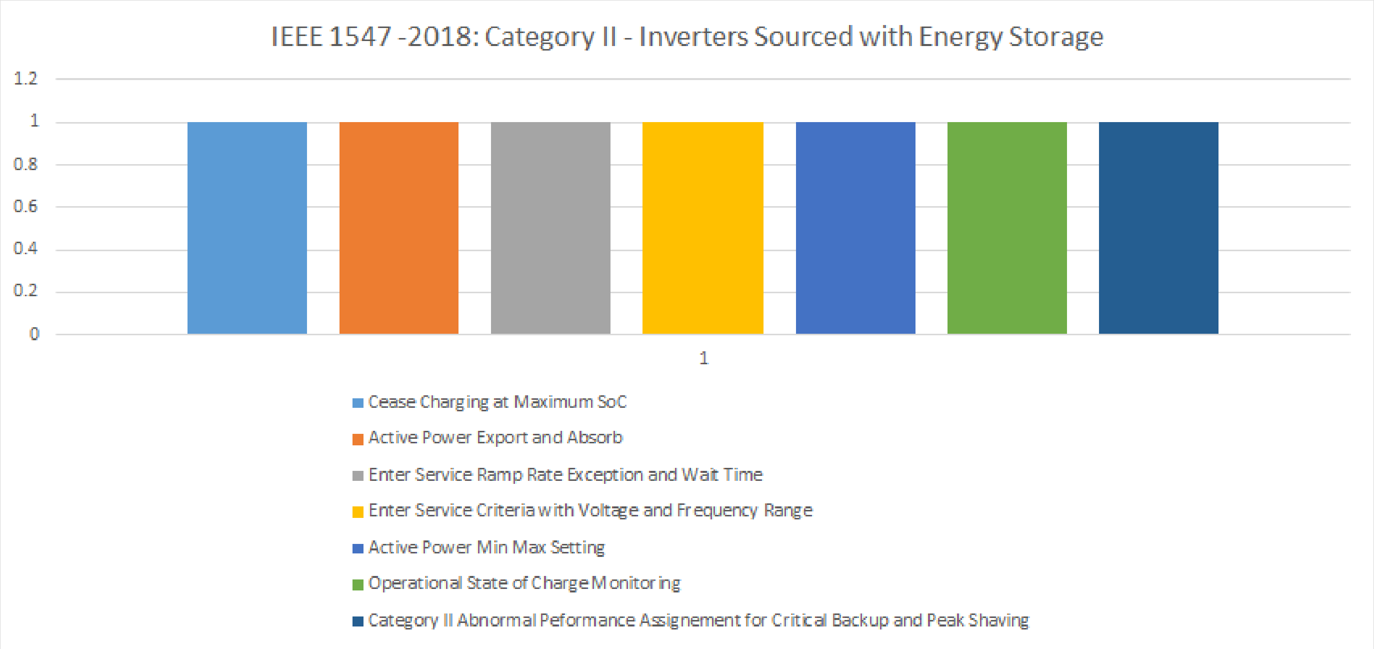
1. Active Power Delivery and BESS Sizing.
2. IEEE 1547 -2018: Category II - Inverters Sourced with Energy Storage Mapping.
3. IEEE 2030.2.1-2019 Guide for Design, Operation, and Maintenance of Battery Energy Mapping.
4. Impact of peak shaving function time delay

This plot shows the loss in active power delivery with variation in BESS sizes. The grid capacity and

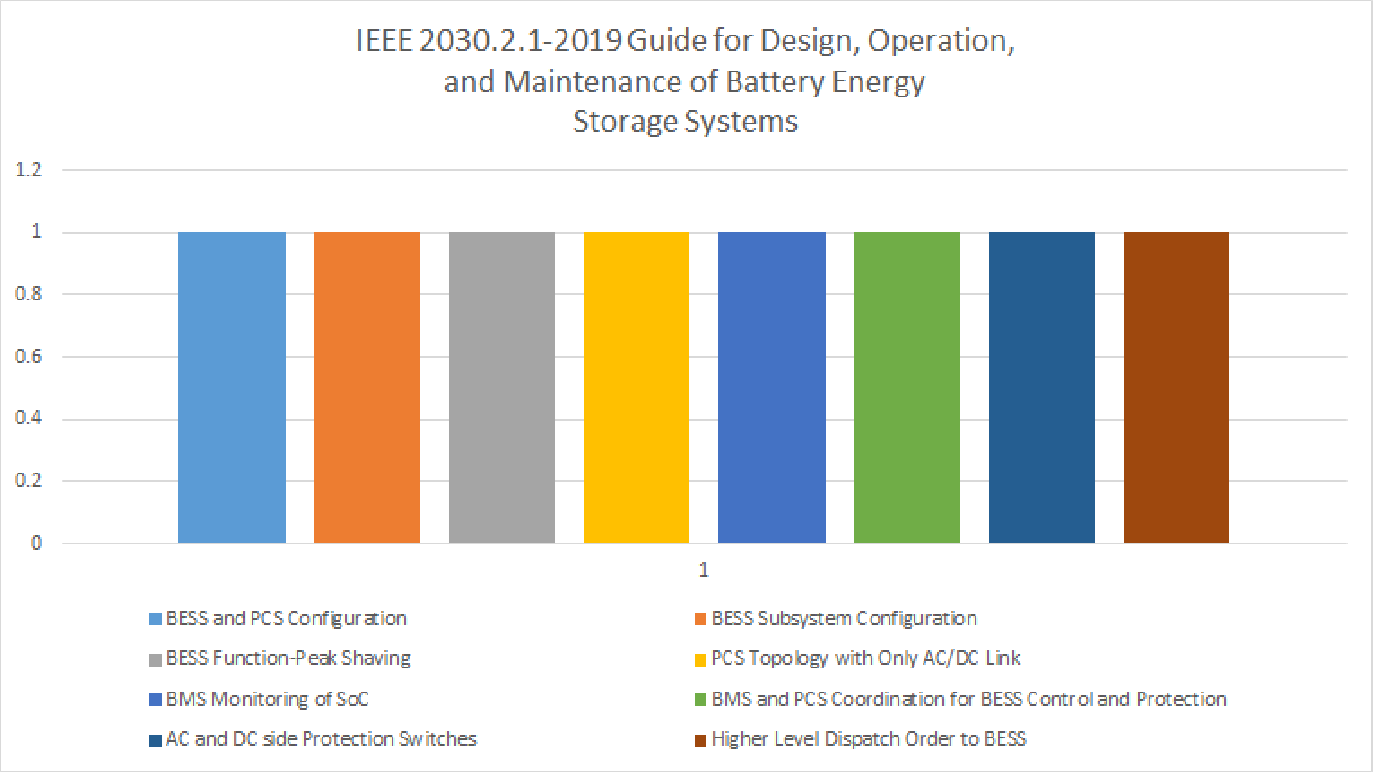
load variation are constant.



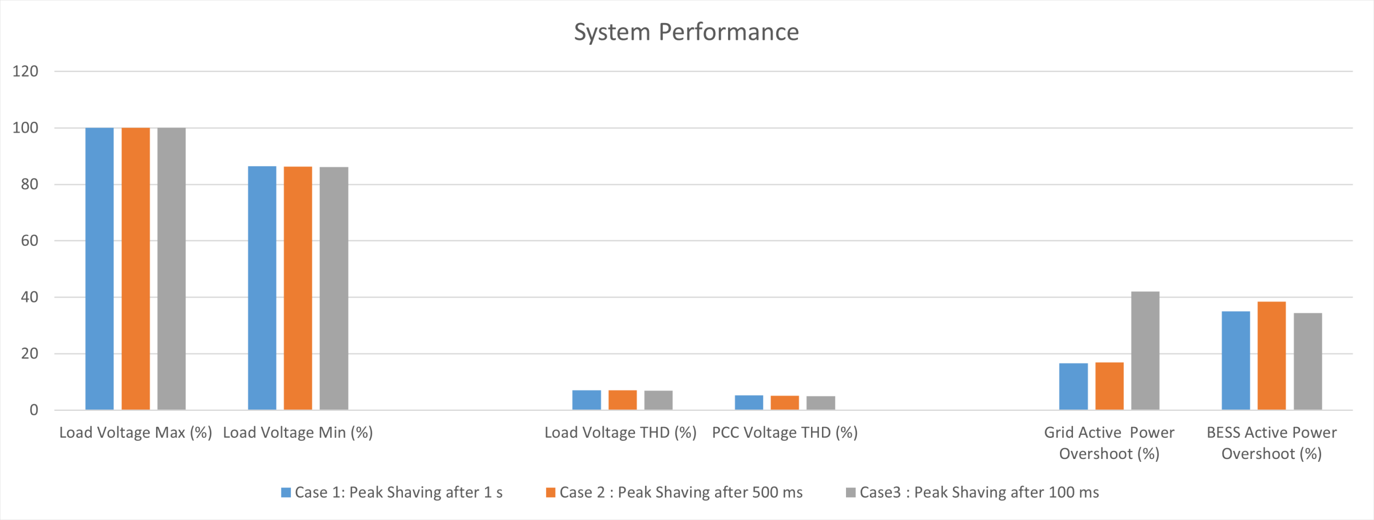
This plot shows the indices for the BESS system implemented in this model following the IEEE 1547 -2018: Category II - Inverters Sourced with Energy Storage standard.



This plot shows the indices for the BESS system operation and maintenance implemented in this model following the IEEE 2030.2.1-2019 Guide for Design, Operation, and Maintenance of Battery.



This plot shows the impact of peak shaving function time delay after grid power crosses the threshold.



The time delay of the peak shaving function has more impact on the overshoots of the active power from the grid and the BESS.

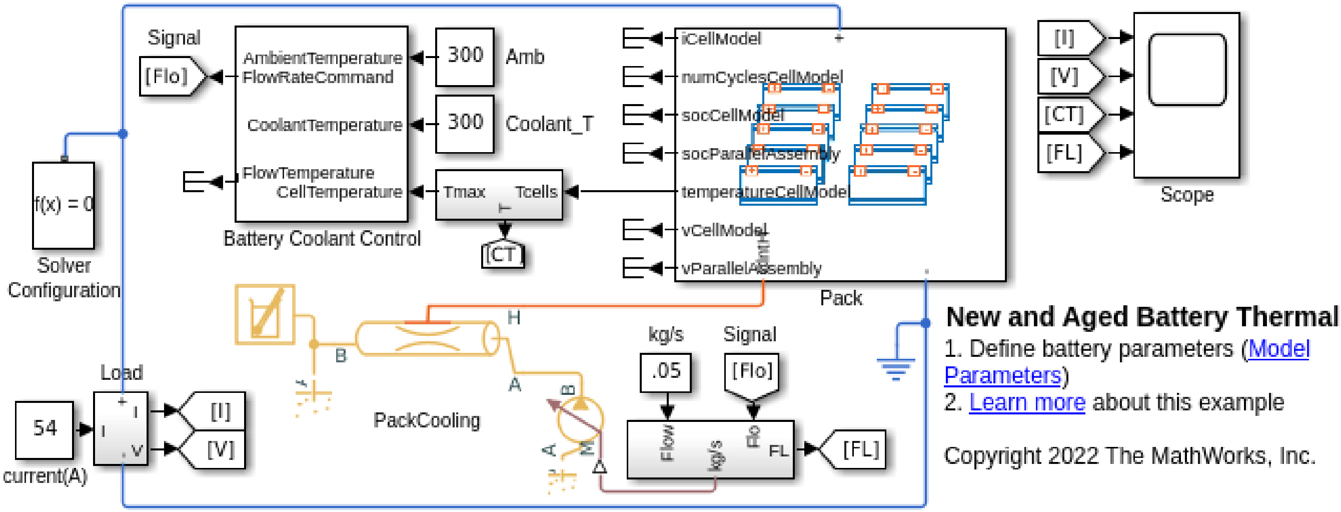
There are no significant impact on the load voltage and total harmonic distortion (THD) values.

Thermal Analysis for New and Aged Battery Packs

This example shows how to evaluate a new and end-of-life (EOL) lithium-ion battery pack. With cell usage and time, the capacity of the cell degrades and the resistance increases due to the formation of a solid-electrolyte-interface (SEI), a passivation layer over the anode surface. You must design battery pack components to meet warranty criteria at EOL time from power, performance, and packaging perspectives. This example analyzes a 400V battery pack for EOL thermal performance based on its packaging.

Build Battery Pack

To build the battery pack used in this example, follow the steps in the “Build Model of Battery Pack with Cell Aging” on page 4-110 example and generate the `batt_PackCellAgingModelLib` SLX file in your working directory. This SLX file contains the battery pack model for cell aging applications. This battery pack comprises five module assemblies. Each module assembly comprises five modules. Each battery module has 12 cells. The EOL for the battery pack is equal to 1000 cycles.



New and Aged Battery Thermal

1. Define battery parameters (Model Parameters)
2. Learn more about this example

Copyright 2022 The MathWorks, Inc.

A Pipe block cools the battery pack modules and the Battery Coolant Control block controls the coolant flowrate. To analyze the worst case scenario, the circuit receives a constant 2C-rate current of 54 A for 30 minutes.

Define Parameters and Run Simulations

Initialize the battery parameters. At the MATLAB Command Window, enter:

```
run("batt_PackCellAgingModel_param.m");
```

Simulate a new battery pack, for a constant discharging current at 2C rate.

```
batt_PackCellAgingModelData = sim("batt_PackCellAgingModelSim.slx");
% Post-process data
newPack_Temp = batt_PackCellAgingModelData.batt_PackCellAgingResults.get(3).Values.Data;
newPack_Time = batt_PackCellAgingModelData.batt_PackCellAgingResults.get(3).Values.Time;
newPack_Volt = batt_PackCellAgingModelData.batt_PackCellAgingResults.get(2).Values.Data;
```

```
newPack_Curr = batt_PackCellAgingModelData.batt_PackCellAgingResults.get(1).Values.Data;
newPack_Flow = batt_PackCellAgingModelData.batt_PackCellAgingResults.get(4).Values.Data;
```

Simulate an EOL battery pack. Every 100 cycles, the **Terminal Resistance, R0** parameter of the cell decreases by 5%. In each module, set the **Change in terminal resistance after N discharge cycles (%)** parameter to 5 and the **Number of discharge cycles, N** parameter to 100.

```
ModuleType1.N0 = 100;
ParallelAssemblyType1.N0 = 100;
ModuleType1.dR0 = 5;
ParallelAssemblyType1.dR0 = 5;
```

The thermal resistance of the battery pack, between the cells and the cooling system, degrades with time. The value of the thermal resistance increases from 1.2 for the new pack to 5 for the aged pack.

```
ModuleType1.CoolantThermalPathResistance = 5;
ParallelAssemblyType1.CoolantThermalPathResistance = 5;
```

Initialize the battery pack close to the EOL cycle (999).

```
end_of_life_cycles = 999;
run("batt_PackCellAgingModel_param_EOL.m")
```

Simulate the EOL battery pack for a constant discharging current at 2C rate.

```
batt_PackCellAgingModelData = sim("batt_PackCellAgingModelSim.slx");
% Post-process data
agedPack_Temp = batt_PackCellAgingModelData.batt_PackCellAgingResults.get(3).Values.Data;
agedPack_Time = batt_PackCellAgingModelData.batt_PackCellAgingResults.get(3).Values.Time;
agedPack_Volt = batt_PackCellAgingModelData.batt_PackCellAgingResults.get(2).Values.Data;
agedPack_Curr = batt_PackCellAgingModelData.batt_PackCellAgingResults.get(1).Values.Data;
agedPack_Flow = batt_PackCellAgingModelData.batt_PackCellAgingResults.get(4).Values.Data;
```

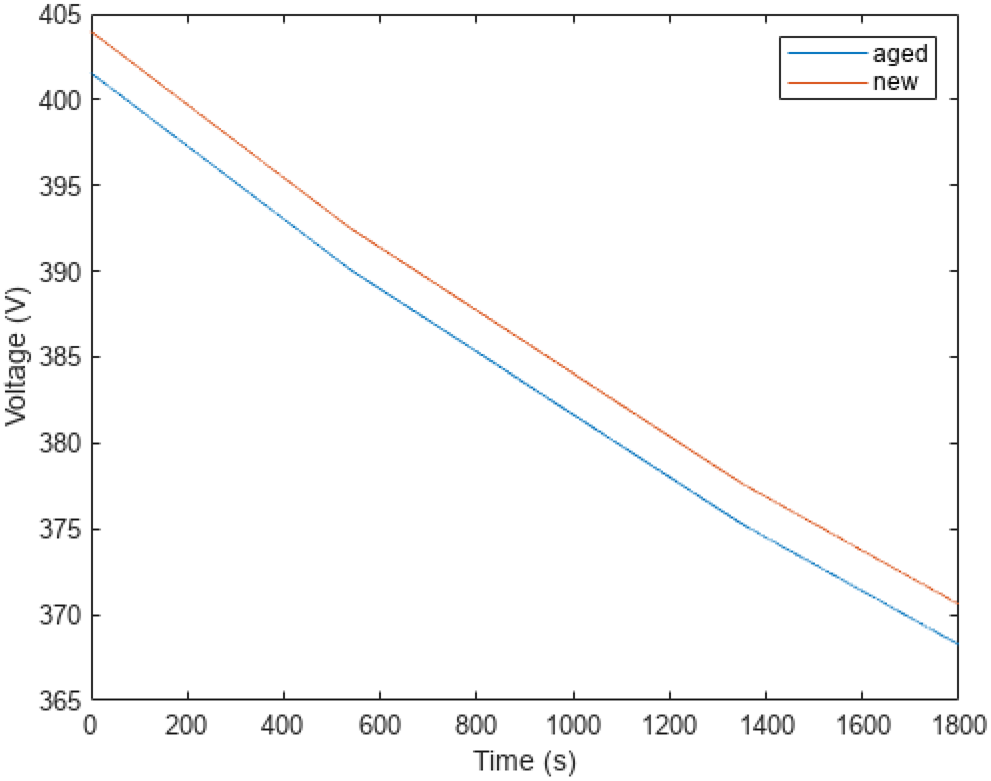
Analyze Results

Load the `batt_PackWithCellBalancingResults` MAT file and run the `batt_PackWithCellBalancingPlot` M file. At the MATLAB Command Window, enter:

```
max_temp_diff = round(max(squeeze(agedPack_Temp))-max(squeeze(newPack_Temp)),1);
disp(['Battery EOL max. cell temperature is ~ ',num2str(max_temp_diff),...
    ' higher compared to max. cell temperature in a new pack'])
```

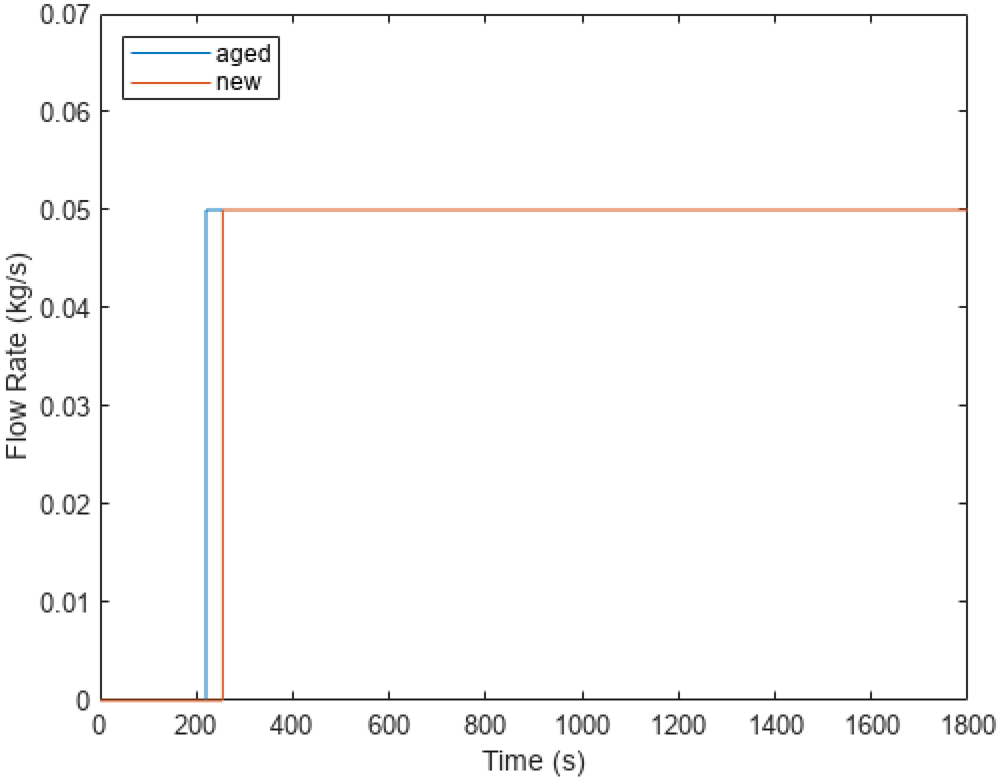
```
Battery EOL max. cell temperature is ~ 7.1 higher compared to max. cell temperature in a new pack
```

```
figure(1)
plot(agedPack_Time,squeeze(agedPack_Volt));hold on;
plot(newPack_Time,squeeze(newPack_Volt));hold off;
legend('aged','new','Location','northeast')
ylabel('Voltage (V)')
xlabel('Time (s)')
```



For the aged cells, the maximum cell temperature is almost 7 degree Celsius higher than the maximum cell temperature of a new pack. The voltage of the aged pack is slightly lower than the voltage of the new pack. These values show that the battery pack design is thermally safe from EOL perspective.

```
figure(2)
plot(agedPack_Time,squeeze(agedPack_Flow));hold on;
plot(newPack_Time,squeeze(newPack_Flow));hold off;
legend('aged','new','Location','northwest')
ylabel('Flow Rate (kg/s)')
xlabel('Time (s)')
ylim([0 0.07])
```



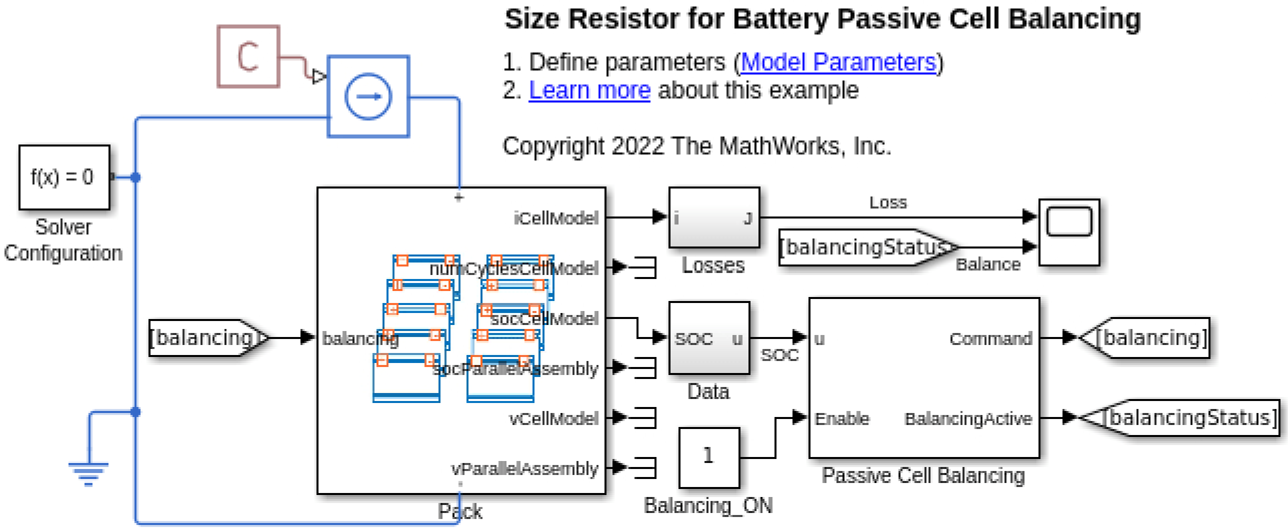
This plot shows the coolant flow switch-on times for a new and an aged battery pack. Inside the aged battery pack, as the cells heat up more than in a new battery pack, the battery controller switches on the coolant pump earlier. The pump power consumption is higher due to the earlier activation of the coolant pump.

Size Resistor for Battery Passive Cell Balancing

This example shows how to implement a passive cell balancing for a lithium-ion battery pack. Cell-to-cell differences in the battery module create imbalances in the cell state-of-charge (SOC) and voltages. In this example, the balancing algorithm triggers when the battery pack is idle and the difference in the cell SOC is greater than a certain predefined value. The passive balancing shunt resistor is sized based on power loss and balancing time considerations.

Build Battery Pack

To build the battery pack used in this example, follow the steps in the “Build Model of Battery Pack with Cell Balancing Circuit” on page 4-116 example and generate the `batt_PackWithCellBalancingLib` SLX files in your working directory. This SLX file contains the battery pack model for cell balancing applications. This battery pack comprises two module assemblies. Each module assembly comprises two modules. Each battery module has 16 cells. Open the `batt_PackWithCellBalancingLib` SLX file, drag and drop the Pack subsystem to your model, and connect it to the Passive Cell Balancing block. The Passive Cell Balancing block uses the cell SOC as balancing parameter.



Size Resistor for Battery Passive Cell Balancing

1. Define parameters (Model Parameters)
2. Learn more about this example

Copyright 2022 The MathWorks, Inc.

Define Parameters

Initialize the battery parameters

```
run("batt_PackWithCellBalancing_param.m");
```

In this example, the balancing threshold is equal to 0.1% of the SOC.

```
threshold_balancing_SOC = 1e-3;
```

For both the modules inside ModuleAssembly1 object, define all the 16 initial cell SOC.

```
ModuleAssembly1.Module1.socCellModel =...
    [0.69;0.69;0.69;0.69;...
    0.715;0.715;0.715;0.715;...
    0.7;0.7;0.7;0.7;...
    0.7;0.7;0.7;0.7];
ModuleAssembly1.Module2.socCellModel =...
    ModuleAssembly1.Module1.socCellModel;
```

Do the same for both modules inside the ModuleAssembly2 object.

```
ModuleAssembly2.Module1.socCellModel =...
    [0.69;0.69;0.69;0.69;...
     0.715;0.715;0.715;0.715;...
     0.7;0.7;0.7;0.7;...
     0.7;0.7;0.7;0.7];
ModuleAssembly2.Module2.socCellModel =...
    ModuleAssembly2.Module1.socCellModel;
```

Specify the shunt resistor options that you want to evaluate.

```
balancingResistor_options = [2 3 4 5 6]; % all Resistances in Ohm
```

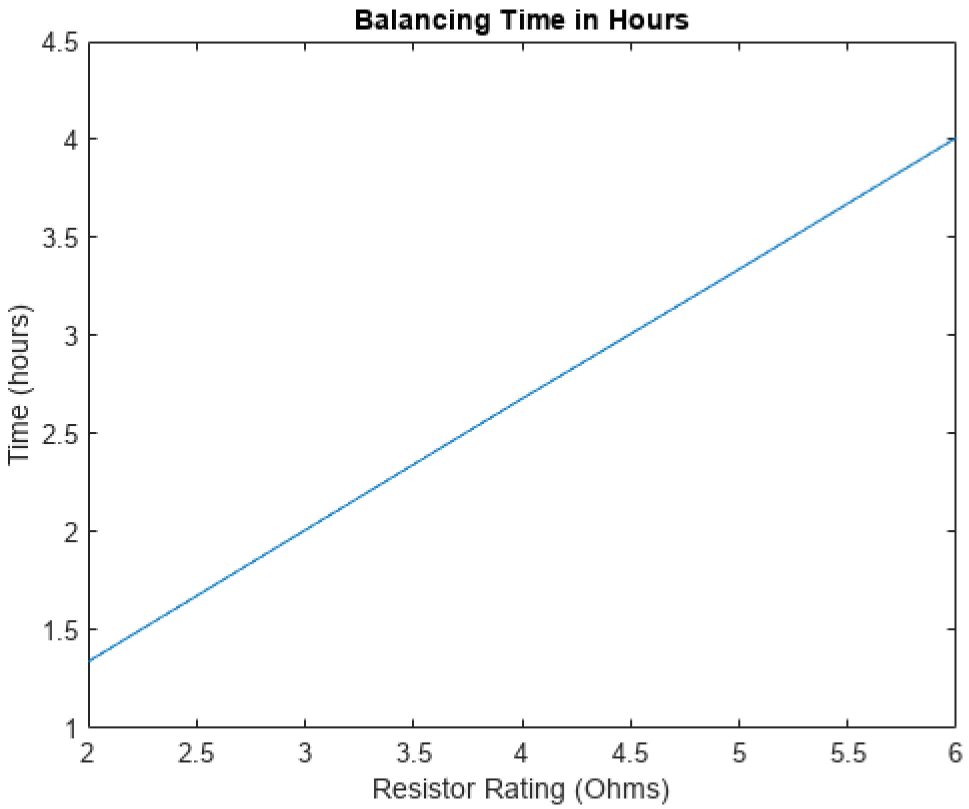
Run Simulations

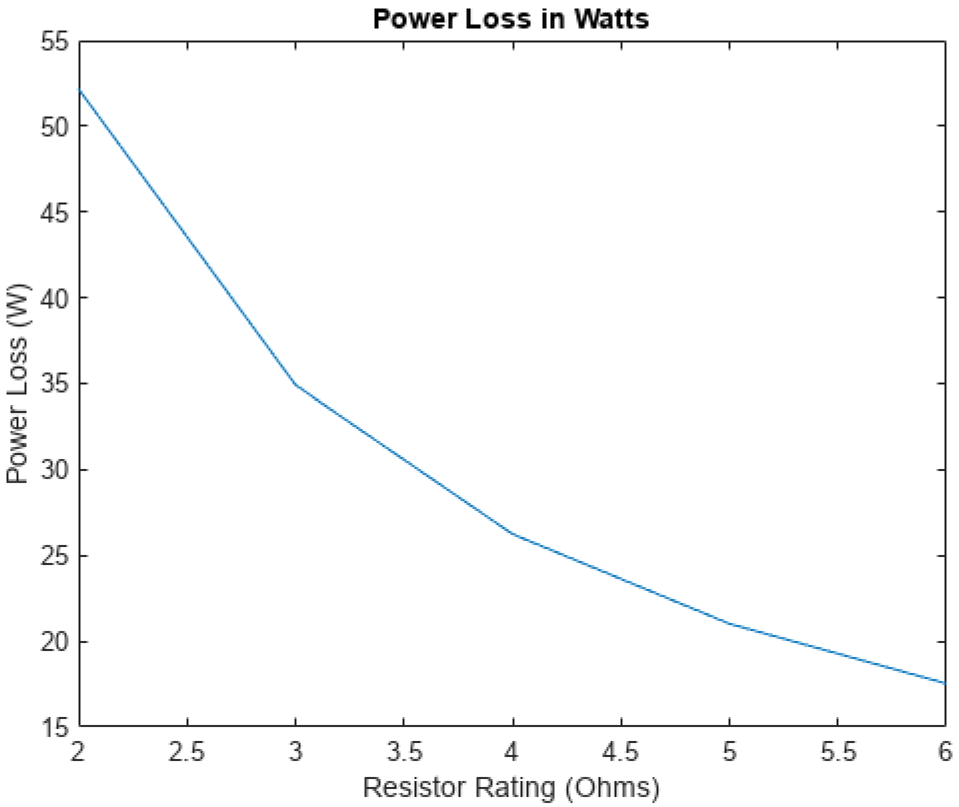
Simulate the model for all the balancing resistor options specified in the `balancingResistor_options` variable. At the MATLAB Command Window, run the `batt_PackWithCellBalancingSimulate` M file. The file runs simulation for all the balancing resistor options and stores the output result in a `batt_PackWithCellBalancingResults` MAT file.

Analyze Results

Load the `batt_PackWithCellBalancingResults` MAT file, in the MATLAB Command Window, enter:

```
run("batt_PackWithCellBalancingPlot.m")
```


Balancing Time in Hours
Time (hours)
4.5
4
3.5
3
2.5
2
1.5
1
2
2.5
3
3.5
4
4.5
5
5.5
6
Resistor Rating (Ohms)



The first plot shows the balancing time, in hours, for each resistor rating. For a pack resistor of 4 Ohm, the battery SOC balances in around 2.5 hours.

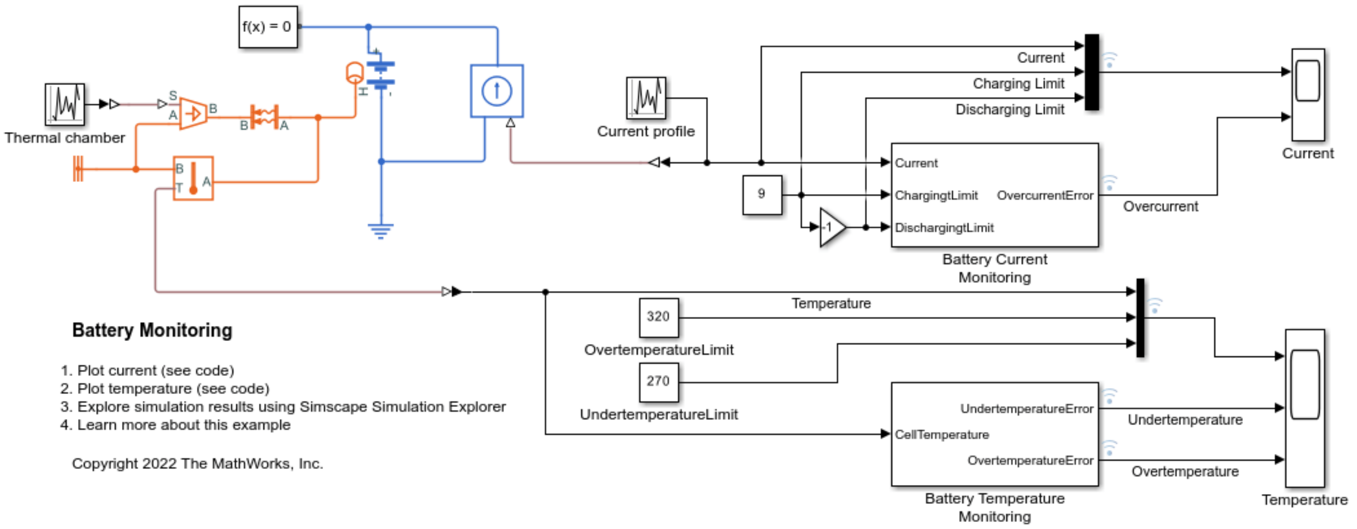
The second plot shows the power loss, in Watts, for each resistor rating. A resistor of 4 Ohm produces a power loss equal to almost 25 W.

The 4 Ohm resistor is a good trade-off for the final hardware.

Battery Monitoring

This example shows how to use battery management system blocks to monitor the current and temperature of a battery. A random current and temperature profile is applied to the battery which is then simulated for 6 hours.

Model

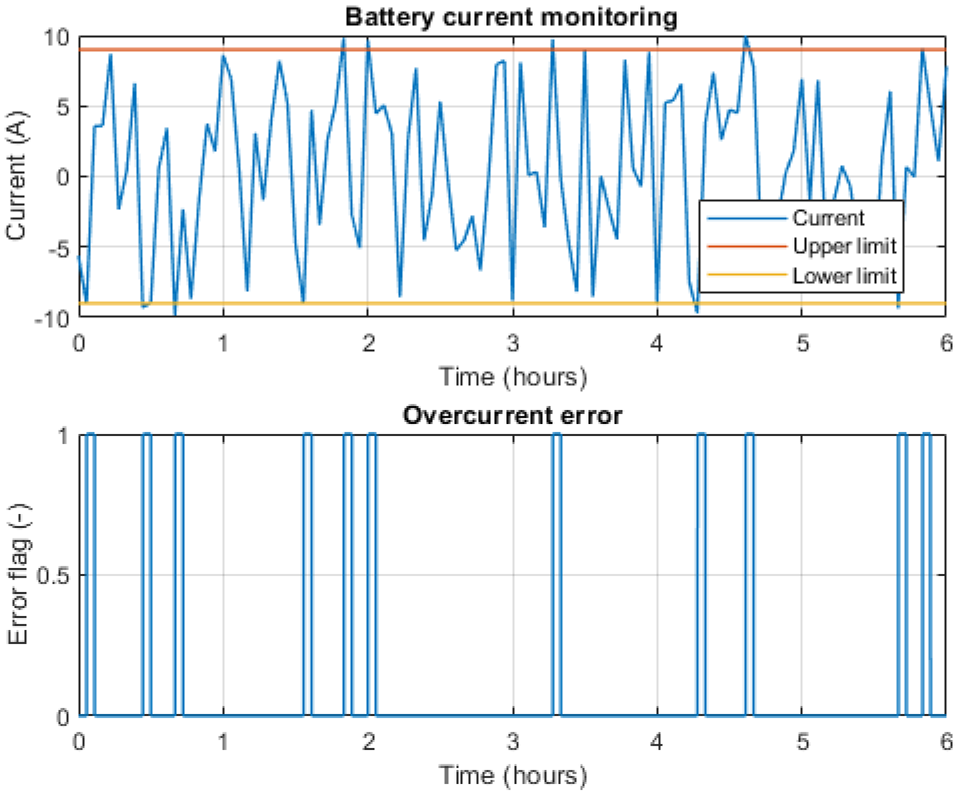


Battery Monitoring

1. Plot current (see code)
2. Plot temperature (see code)
3. Explore simulation results using Simscape Simulation Explorer
4. Learn more about this example

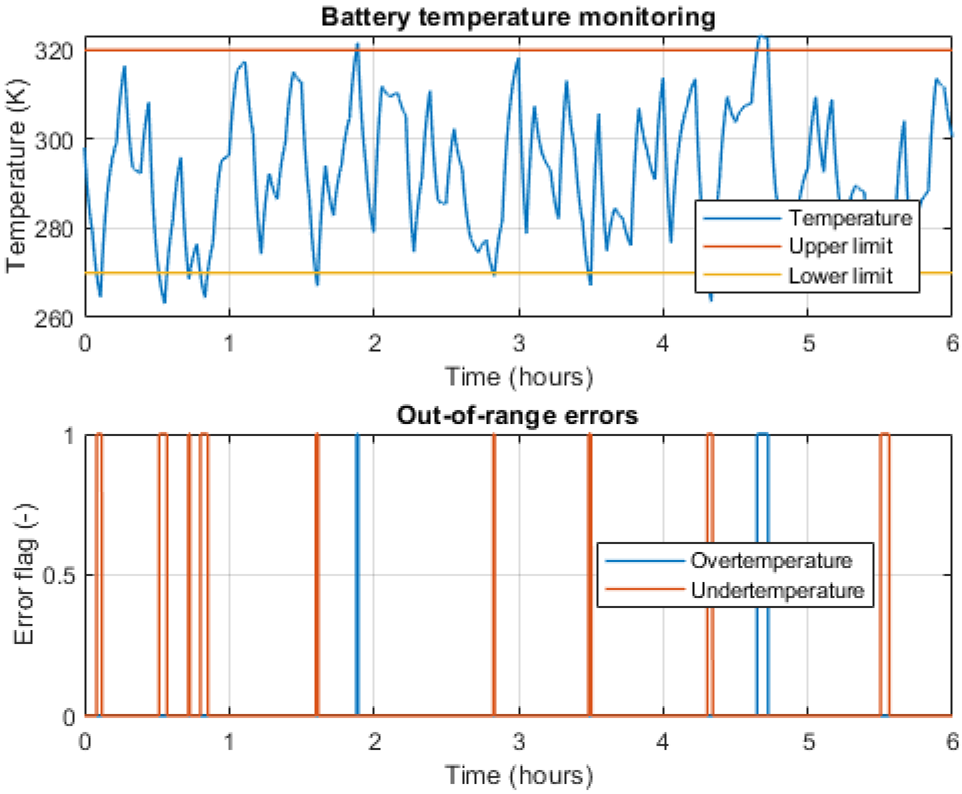
Current Monitoring Results

The plot below shows the battery current and overcurrent error.



Temperature Monitoring Results

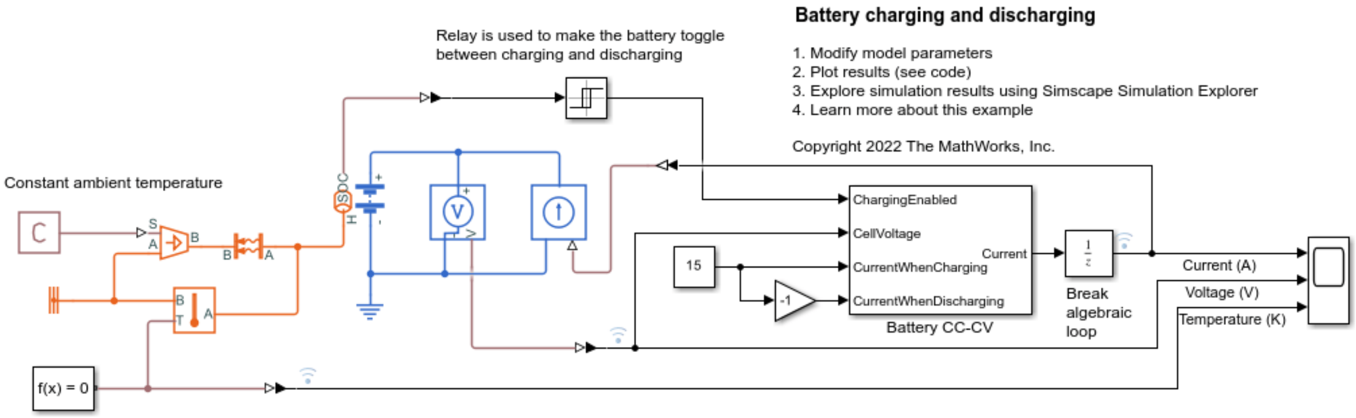
The plot below shows the battery temperature and temperature out-of-range errors.


Battery temperature monitoring
Temperature (K)
320
300
280
260
0
1
2
3
4
5
6
Time (hours)
Temperature
Upper limit
Lower limit
Out-of-range errors
Error flag (-)
1
0.5
0
0
1
2
3
4
5
6
Time (hours)
Overtemperature
Undertemperature

Battery Charging and Discharging

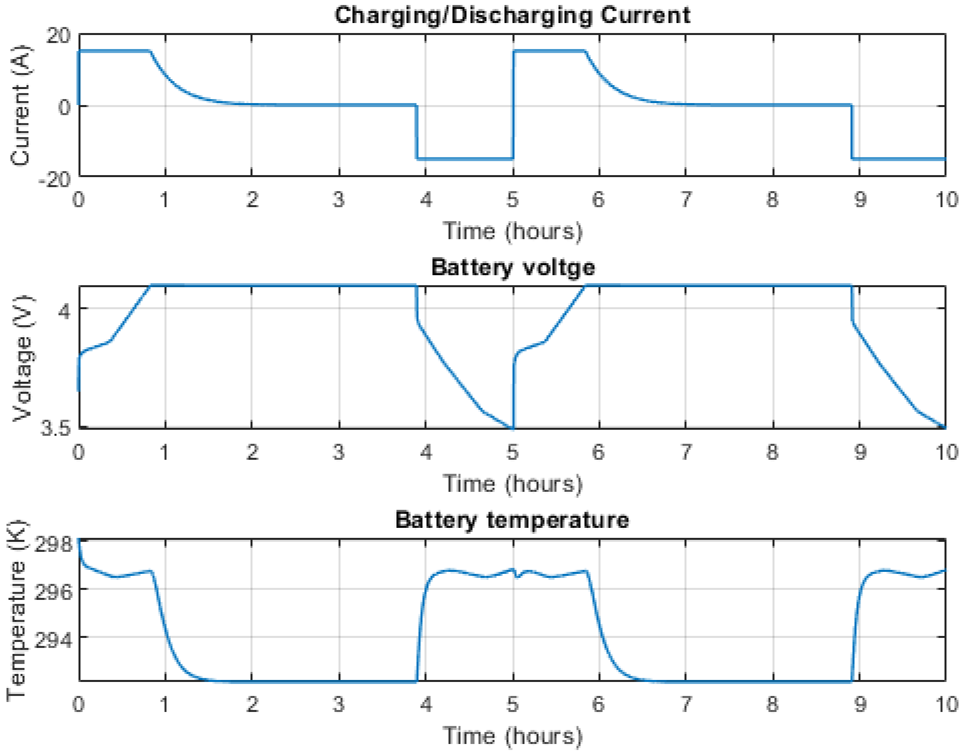
This example shows how to use a constant current and constant voltage algorithm to charge and discharge a battery. The Battery CC-CV block is charging and discharging the battery for 10 hours. The initial state-of-charge is equal to 0.3. When the battery is charging, the current is constant until the battery reaches the maximum voltage and the current decreases towards 0. When the battery is discharging, a constant current is used.

Model


Battery charging and discharging
1. Modify model parameters
2. Plot results (see code)
3. Explore simulation results using Simscape Simulation Explorer
4. Learn more about this example
Copyright 2022 The MathWorks, Inc.
Relay is used to make the battery toggle between charging and discharging
Constant ambient temperature
C
S
A
B
B
A
B
T
A
SOC
H
+
-
V
f(x) = 0
15
-1
ChargingEnabled
CellVoltage
CurrentWhenCharging
CurrentWhenDischarging
Current
Battery CC-CV
Break algebraic loop
Current (A)
Voltage (V)
Temperature (K)

Simulation Results

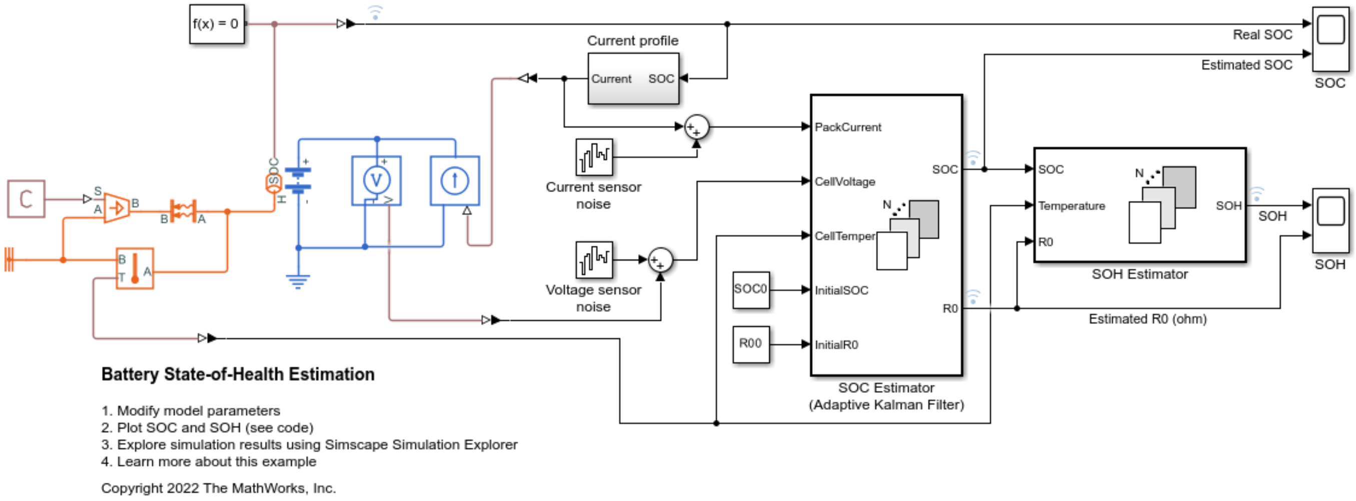
The plot below shows the current, voltage, and temperature of the battery under test.


Charging/Discharging Current
Current (A)
20
0
-20
0
1
2
3
4
5
6
7
8
9
10
Time (hours)
Battery voltge
Voltage (V)
4
3.5
Time (hours)
Battery temperature
Temperature (K)
298
296
294
Time (hours)

Battery State-of-Health Estimation

This example shows how to estimate the battery internal resistance and state-of-health (SOH) by using an adaptive Kalman filter. The initial state-of-charge (SOC) of the battery is equal to 0.6. The estimator uses an initial condition for the SOC equal to 0.65. The battery keeps charging and discharging for 10 hours. The unscented Kalman filter estimator converges to the real value of the SOC while also estimating the internal resistance. To use a different Kalman filter implementation, in the SOC Estimator (Kalman Filter) block, set the Filter type parameter to the desired value.

Model



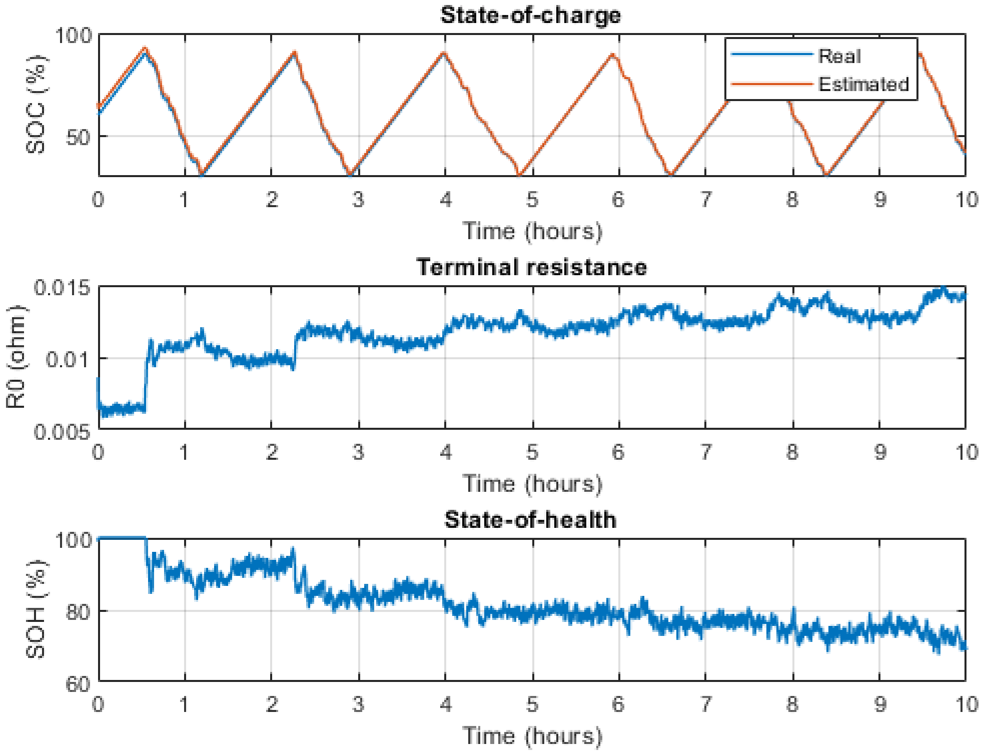
Battery State-of-Health Estimation

1. Modify model parameters
2. Plot SOC and SOH (see code)
3. Explore simulation results using Simscape Simulation Explorer
4. Learn more about this example

Copyright 2022 The MathWorks, Inc.

Simulation Results

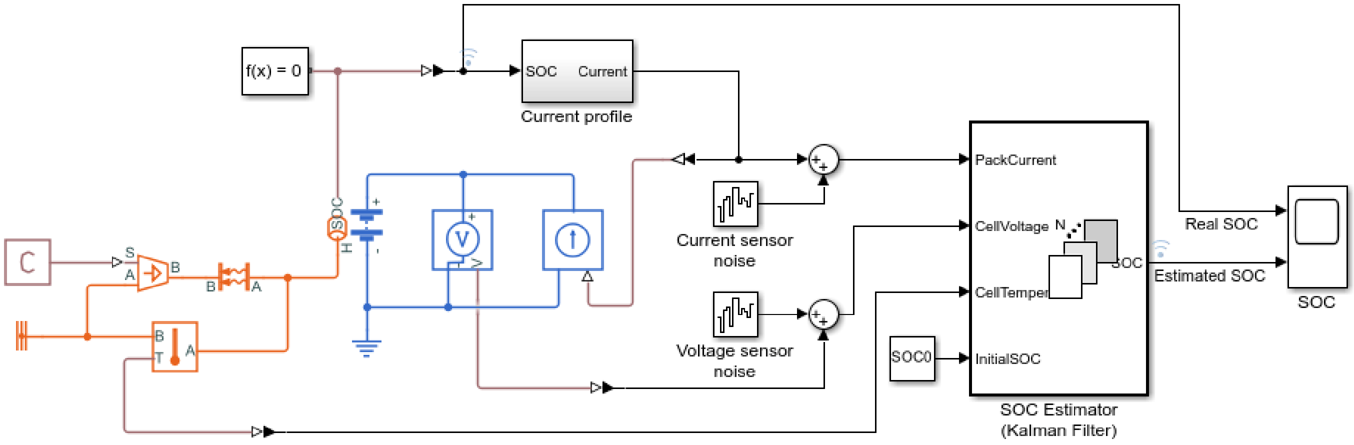
The plot below shows the real and estimated battery state-of-charge, estimated terminal resistance, and estimated state-of-health of the battery.


State-of-charge
SOC (%)
100
50
Real
Estimated
0
1
2
3
4
5
6
7
8
9
10
Time (hours)
Terminal resistance
R0 (ohm)
0.015
0.01
0.005
Time (hours)
State-of-health
SOH (%)
100
80
60
Time (hours)

Battery State-of-Charge Estimation

This example shows how to estimate the battery state-of-charge (SOC) by using a Kalman filter. The initial SOC of the battery is equal to 0.5. The estimator uses an initial condition for the SOC equal to 0.8. The battery keeps charging and discharging for 6 hours. The extended Kalman filter estimator converges to the real value of the SOC in less than 10 minutes and then follows the real SOC value. To use a different Kalman filter implementation, in the SOC Estimator (Kalman Filter) block, set the Filter type parameter to the desired value.

Model

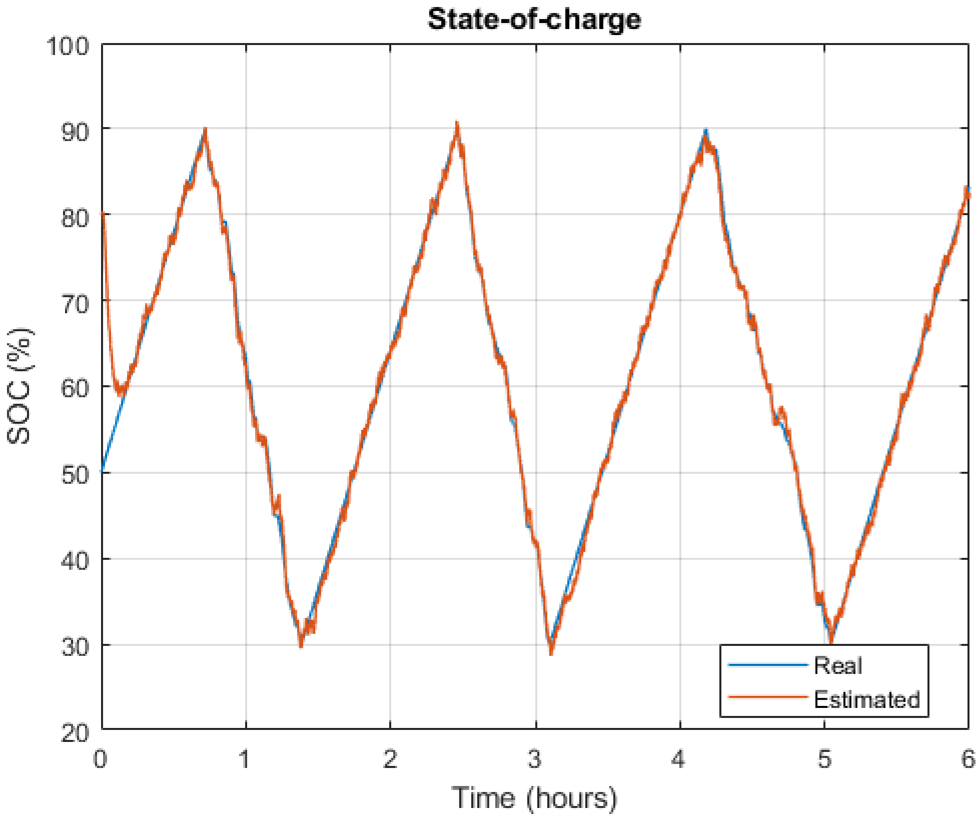


Battery State-of-Charge Estimation

1. Modify model parameters
2. Plot SOC (see code)
3. Explore simulation results using Simscape Simulation Explorer
4. Learn more about this example

Simulation Results

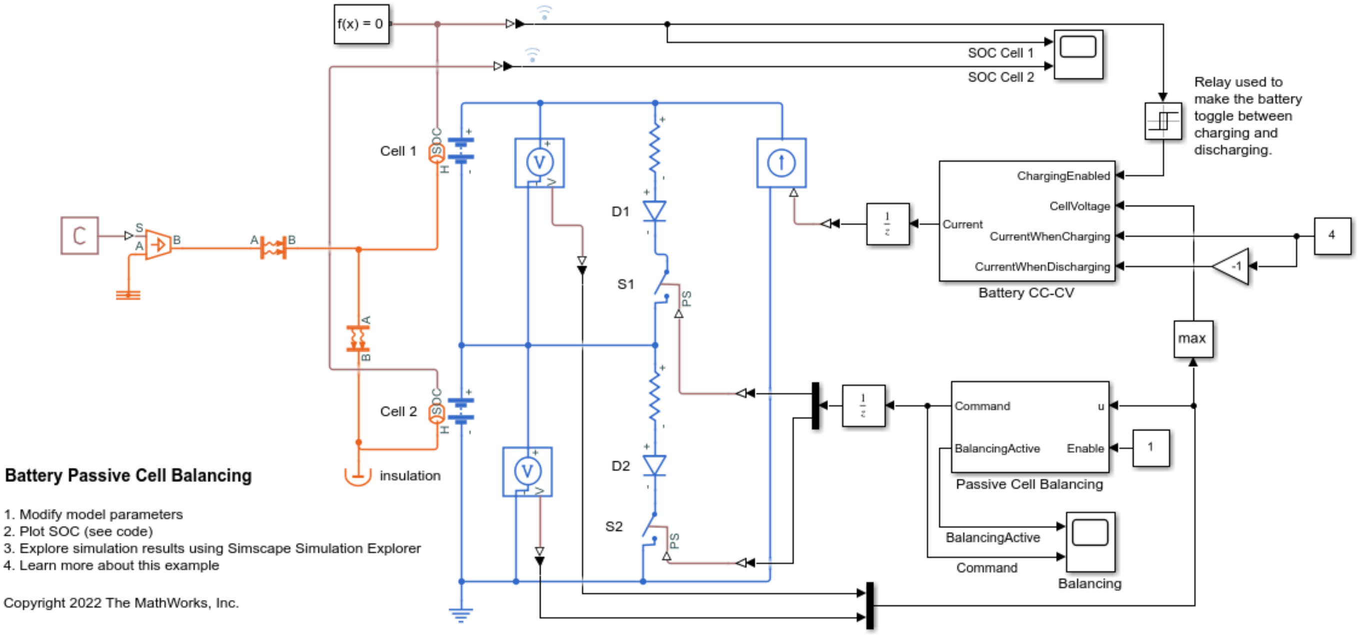
The plot below shows the real and estimated battery state-of-charge.


State-of-charge
SOC (%)
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6
Time (hours)
Real
Estimated

Battery Passive Cell Balancing

This example shows how to balance a battery with two cells connected in series by using a passive cell balancing algorithm. The initial state-of-charge (SOC) for the two cells are equal to 0.7 and 0.75. The balancing procedure depends on the cell voltages. Alternatively, you can use the SOC values for balancing. When the balancing is active, a bleeding resistor switches on to bleed the cells with higher charge. You can use the objects and functions in the Battery Pack Model Builder to generate more complex battery packs.

Model



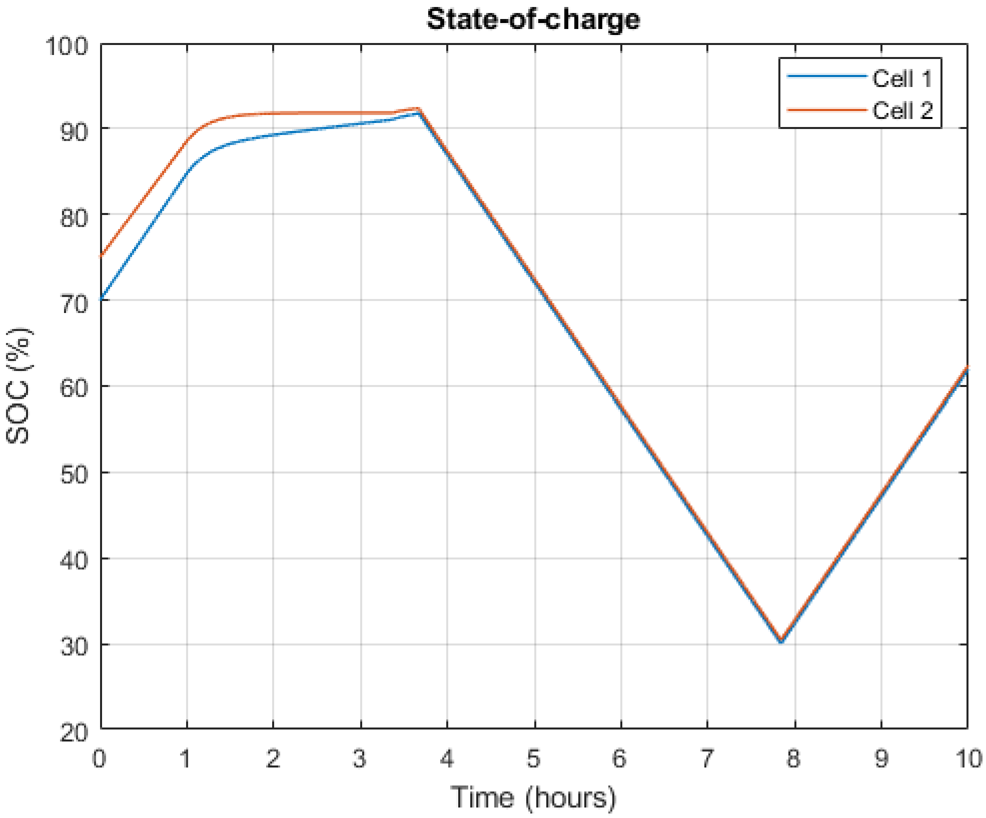
Battery Passive Cell Balancing

1. Modify model parameters
2. Plot SOC (see code)
3. Explore simulation results using Simscape Simulation Explorer
4. Learn more about this example

Copyright 2022 The MathWorks, Inc.

Simulation Results

The plot below shows the cell state-of-charge values.


State-of-charge
Cell 1
Cell 2
SOC (%)
100
90
80
70
60
50
40
30
20
0
1
2
3
4
5
6
7
8
9
10
Time (hours)

Build Detailed Model of Battery Pack From Cylindrical Cells

This example shows how to create and build Simscape™ system models for various battery designs and configurations based on cylindrical battery cells in Simscape™ Battery™. The `buildBattery` function allows you to automatically generate Simscape models for these Simscape Battery objects:

- `ParallelAssembly`
- `Module`
- `ModuleAssembly`
- `Pack`

This function creates a library in your working folder that contains a system model block of a battery pack that you can use as reference in your simulations. The run-time parameters for these models, such as the battery cell impedance or the battery open-circuit voltage, are defined after the model creation and are therefore not covered by the Battery Pack Builder classes. To define the run-time parameters, you can either specify them in the block mask of the generated Simscape models or use the **MaskParameters** argument of the `buildBattery` function.

During the first half of this example, you first define the key properties of a cylindrical battery cell and block model. You then use this cylindrical battery cell as a fundamental repeating unit inside a parallel assembly component. In the industry this component is also called a "sub-module", a "super-cell", a "P-set", or just a "cell". You later employ this parallel assembly to define a battery module, which is then used to create a module assembly and finally a battery pack. These larger battery systems all use the battery cell as a fundamental repeating unit. Throughout the workflow, you visualize the geometry and the relative positioning of these battery systems by using the `BatteryChart` object.

In the second half of the example, you modify the modeling methodology and the model resolution of the `Module`, `ModuleAssemblies`, and `Pack` objects before generating the final Simscape battery model. You can perform the geometrical aggregation or stacking of any battery object along the sequence either along the `X` or `Y` axis. These axis mirror the Vehicle Coordinate System (Z-up).

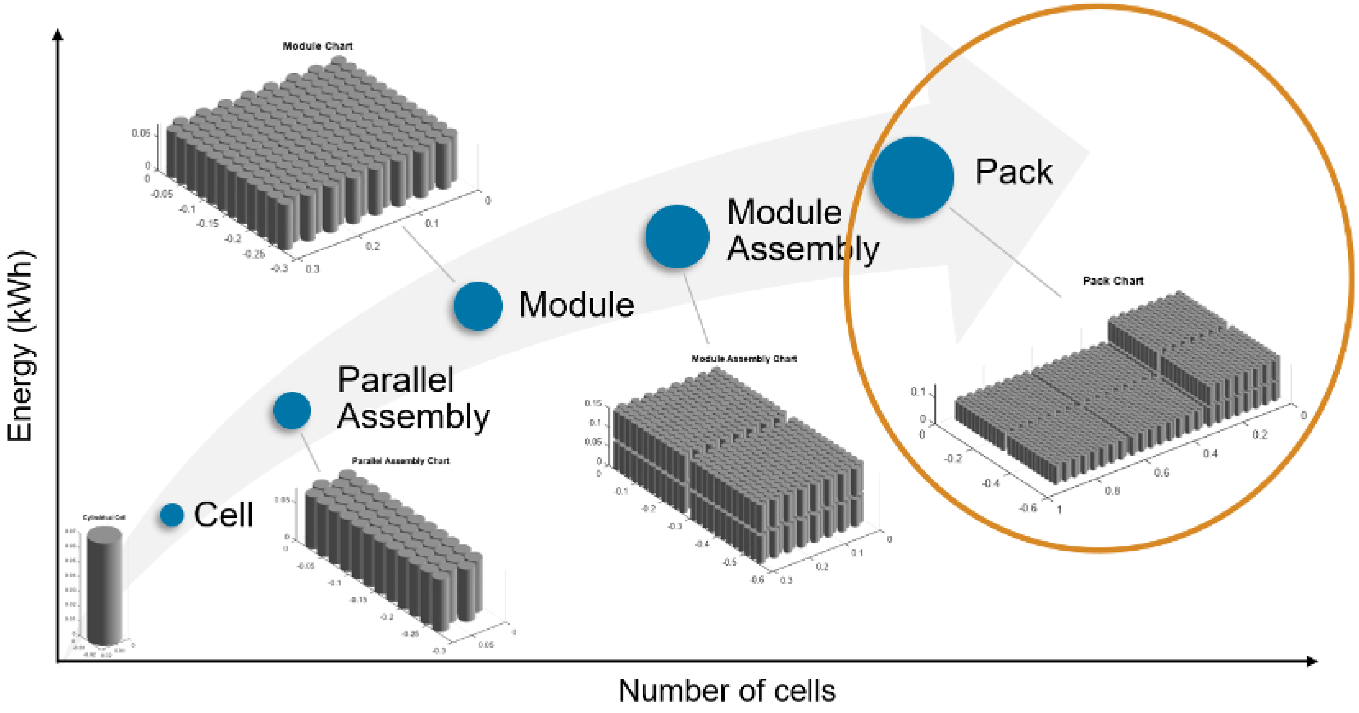
To use the functions and objects in Simscape Battery, first import the required Simscape Battery package:

```
import simscape.battery.builder.*
```

Create and Visualize Battery Objects in MATLAB

To create a battery pack, you must first design and create the foundational elements of the battery pack.

This uifigure shows the overall process to create a battery pack object in a bottom-up approach:



A battery pack comprises multiple module assemblies. These module assemblies, in turn, comprise a number of battery modules connected electrically in series or in parallel. The battery modules are made of multiple parallel assemblies which, in turn, comprise a number of battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement.

Create and Visualize Battery Cell Object

A battery cell is an electrochemical energy storage device that provides electrical energy from stored chemical energy. An electrochemical battery cell is the fundamental building block in the manufacturing of larger battery systems. To obtain the required energy and voltage levels, multiple battery cells are typically connected electrically in parallel and/or in series.

To mirror the real-world behavior, the Simscape Battery™ `Cell` object is the foundational element for the creation of a battery pack system model. You can create all battery classes without any inputs. To create a battery cell, use the `Cell` object.

```
batteryCell = Cell();
```

To meet the battery packaging and space requirements, you can arrange the battery cells in three main geometrical arrangements: cylindrical, pouch, or prismatic. To be able to visualize a single battery cell, you must first define its geometry.

Define a cylindrical geometry by using the `CylindricalGeometry` object.

```
cellGeometry = CylindricalGeometry();
```

The `CylindricalGeometry` object has two properties:

- **Radius** — Radius of the cylindrical geometry, specified as a `simscape.Value` object that represents a scalar with a unit of length.

- **Height** — Height of the cylindrical geometry, specified as a `simscape.Value` object that represents a scalar with a unit of length.

Specify custom values for the **Radius** and **Height** properties of the cylindrical geometry.

```
cellGeometry.Radius = simscape.Value(0.0105, "m");
cellGeometry.Height = simscape.Value(0.07, "m");
```

For more information on the possible geometrical arrangements of a battery cell, see the `PouchGeometry` and `PrismaticGeometry` documentation pages.

You can now link this geometry object to the battery cell by accessing the Geometry property of the `batteryCell` object.

```
batteryCell.Geometry = cellGeometry;
```

Specify a custom value for the mass of the battery cell by using the **Mass** property.

```
batteryCell.Mass = simscape.Value(0.07,"kg");
disp(batteryCell)
```

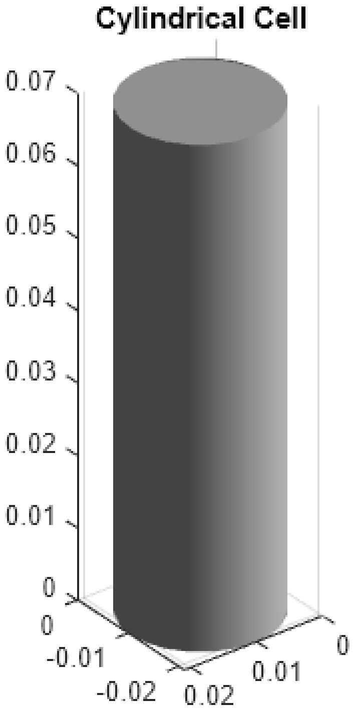
```
  Cell with properties:

            Geometry: [1x1 simscape.battery.builder.CylindricalGeometry]
    CellModelOptions: [1x1 simscape.battery.builder.CellModelBlock]
                Mass: [1x1 simscape.Value]

Show all properties
```

Visualize the battery cell by using the `BatteryChart` object. Create the uifigure where you want to visualize your battery cell.

```
f = uifigure("Color", "white");
```

Then use the `BatteryChart` object to visualize the battery cell.

```
cellChart = BatteryChart(Parent = f, Battery = batteryCell);
title(cellChart, "Cylindrical Cell")
```



For more information, see the `BatteryChart` documentation page.

By default, the Battery (Table-Based) block is the electrical and thermal model used to represent and simulate this battery cell in Simscape. When scaled up into larger battery systems like a parallel assembly or a module, this model is also scaled up accordingly depending on the model resolution. To display the information about the cell model block, use the **CellModelOptions** property of the `batteryCell` object.

```
disp(batteryCell.CellModelOptions.CellModelBlockPath);
```

```
batt_lib/Cells/Battery
(Table-Based)
```

The `Cell` object also allows you to simulate the thermal effects of the battery cell by using a simple 1-D model. To simulate the thermal effects of the battery cell, in the **BlockParameters** property of the **CellModelOptions** property of the `Cell` object, set the **thermal_port** parameter to `"model"`.

```
batteryCell.CellModelOptions.BlockParameters.thermal_port = "model";
```

You can modify all the conditional parameters of the Battery (Table-Based) block by using the **CellModelOptions** property.

```
disp(batteryCell.CellModelOptions.BlockParameters);
```

```
    T_dependence: no
    thermal_port: model
```

```
           prm_age_OCV: OCV
      prm_age_capacity: disabled
    prm_age_resistance: disabled
      prm_age_modeling: equation
               prm_dyn: off
               prm_dir: noCurrentDirectionality
              prm_fade: disabled
              prm_leak: disabled
```

Create and Visualize Battery ParallelAssembly Object

A battery parallel assembly comprises multiple battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement. You can specify the number of cells connected in parallel by using the **NumParallelCells** property.

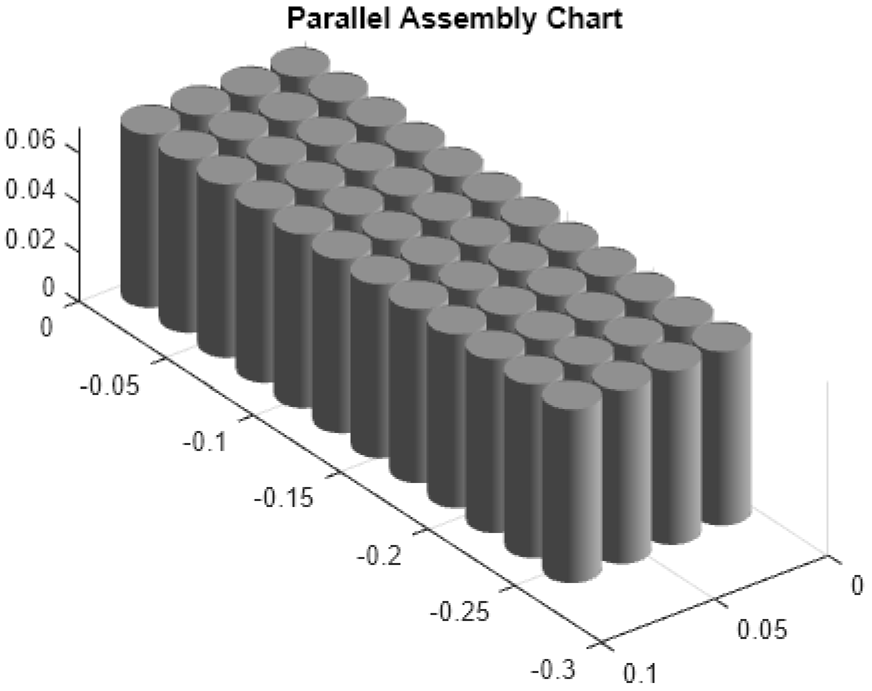
In this example, you create a parallel assembly using 48 of the cylindrical cells created in the previous step, stacked in a square topology over four rows.

```
parallelAssembly = ParallelAssembly(...
    NumParallelCells = 48, ...
    Cell = batteryCell, ...
    Topology = "Square", ...
    Rows = 4, ...
    InterCellGap = simscape.Value(0.001, "m"));
```

The **Topology** property is a function of the cell format. For cylindrical cells, the available topologies are `"Hexagonal"` and `"Square"`. By default, the ParallelAssembly object stacks the cells along the `Y` axis.

Visualize the battery parallel assembly. Create the uifigure where you want to visualize your battery parallel assembly and use the `BatteryChart` object.

```
f = uifigure("Color", "white");
parallelAssemblyChart = BatteryChart(Parent = f, Battery = parallelAssembly);
title(parallelAssemblyChart, "Parallel Assembly Chart")
```

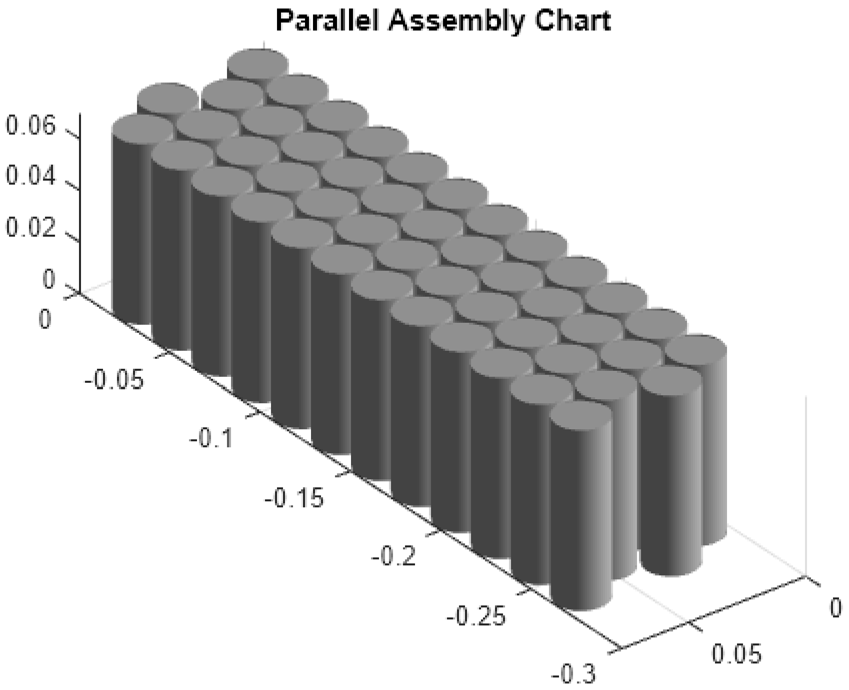


You can modify all the public properties inside the parallel assembly after its creation. For example, you can set the topology of the parallel assembly to the more space-efficient hexagonal configuration. Set the **Topology** property of the `ParallelAssembly` object to `"Hexagonal"`.

```
parallelAssembly.Topology = "Hexagonal";
```

Visualize the hexagonal parallel assembly.

```
f = uifigure("Color", "white");
parallelAssemblyChart = BatteryChart(Parent = f, Battery = parallelAssembly);
title(parallelAssemblyChart, "Parallel Assembly Chart")
```



You can check the cell packaging volume and the mass of any battery by accessing the **PackagingVolume** and **CumulativeMass** properties.

```
disp(parallelAssembly.PackagingVolume)
```

```
    0.0015 : m^3
```

```
disp(parallelAssembly.CumulativeMass)
```

```
    3.3600 : kg
```

Create and Visualize Battery Module Object

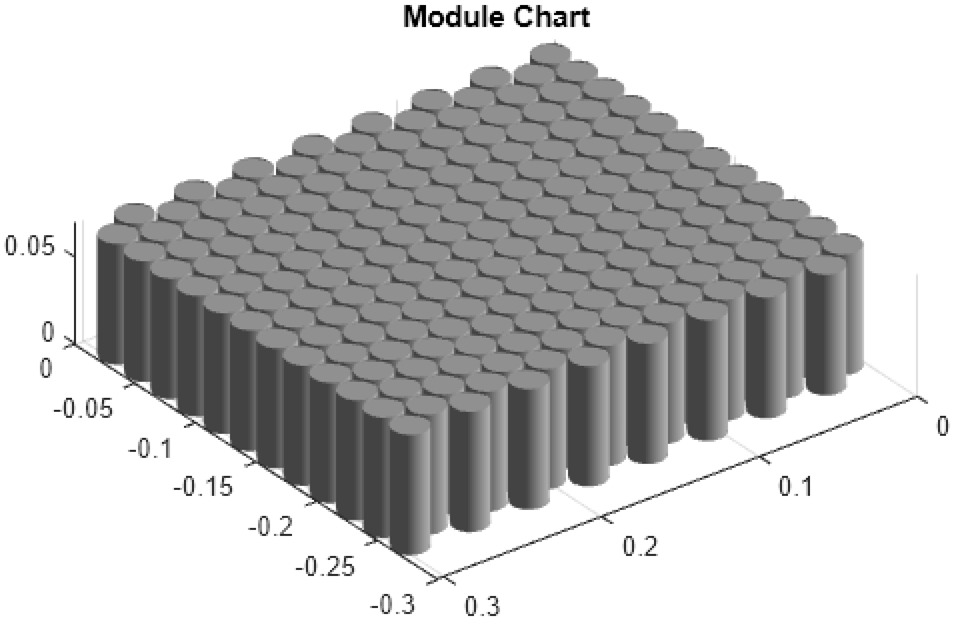
A battery module comprises multiple parallel assemblies connected in series. You can specify the number of parallel assemblies connected in series by using the **NumSeriesAssemblies** property. You can stack or geometrically assemble batteries along the X or Y axis of a Cartesian coordinate system by using the **StackingAxis** property.

In this example, you create a battery module using 4 parallel assemblies that you created in the previous step, stacked along the X axis, with an intergap between each assembly equal to 0.0001 meters.

```
module = Module(...
    ParallelAssembly = parallelAssembly, ...
    NumSeriesAssemblies = 4, ...
    StackingAxis = "X",...
    InterParallelAssemblyGap = simscape.Value(0.0001, "m"));
```

Visualize the battery `Module` object. Create the uifigure where you want to visualize your battery module and use the `BatteryChart` object.

```
f = uifigure("Color", "white");
moduleChart = BatteryChart(Parent = f, Battery = module);
title(moduleChart, "Module Chart")
```



Display the total packacing volume and cumulative mass of your battery module.

```
disp(module.PackagingVolume)
```

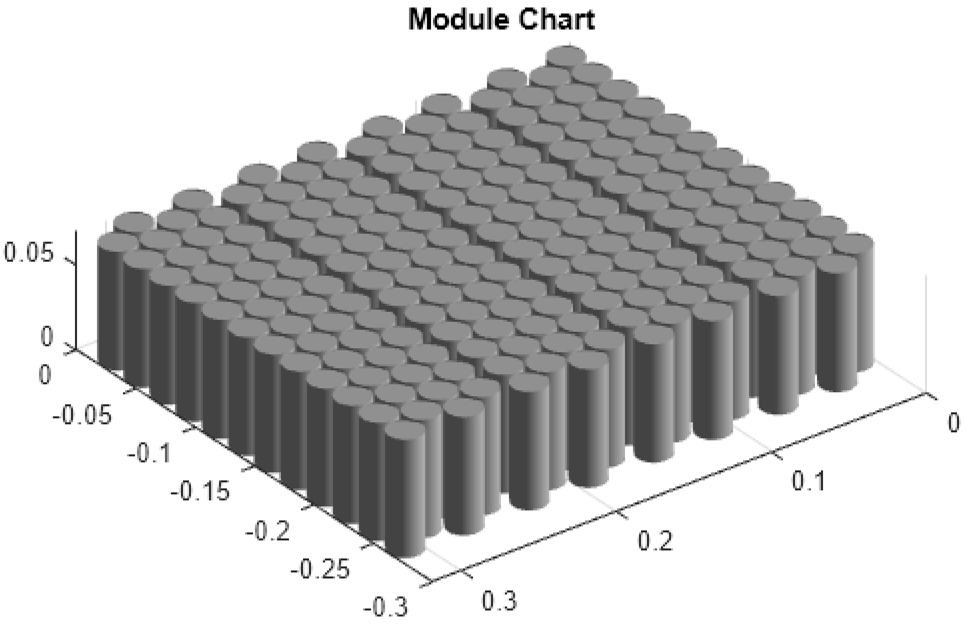
```
    0.0058 : m^3
```

```
disp(module.CumulativeMass)
```

```
   13.4400 : kg
```

You can modify all the public properties inside the module after its creation. For example, modify the gap between parallel assemblies and check how the packaging volume increases due to this change. Set the **InterParallelAssemblyGap** property of the `Module` object to `0.005` m and visualize the object.

```
module.InterParallelAssemblyGap = simscape.Value(0.005, "m");

f = uifigure("Color", "white");
moduleChart = BatteryChart(Parent = f, Battery = module);
title(moduleChart, "Module Chart")
```



Now check the new packaging volume of your battery module.

```
disp(module.PackagingVolume)
```

```
    0.0061 : m^3
```

The packaging volume increased due to the increase in gap between parallel assemblies.

Reset the **InterParallelAssemblyGap** property back to its original value.

```
module.InterParallelAssemblyGap = simscape.Value(0.001,"m");
```

Create and Visualize Battery ModuleAssembly Object

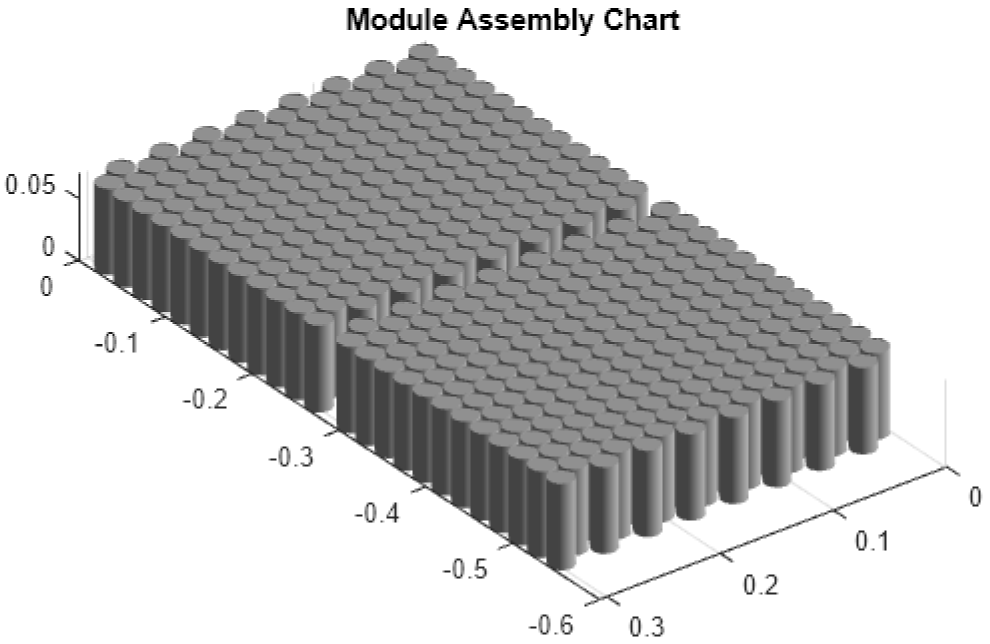
A battery module assembly comprises multiple battery modules connected in series or in parallel. You can define the number and types of modules by using the **Module** property. If a module assembly comprises many identical modules, use the repmat function. Otherwise use an array of distinct modules.

In this example, you create a battery module assembly by using two identical modules of the `Module` object you created in the previous step, stacked along the Y axis, with an intergap between each module equal to 0.005 meters. By default, the `ModuleAssembly` object electrically connects the modules in series.

```
moduleAssembly = ModuleAssembly(...
    Module = repmat(module,1,2), ...
    StackingAxis = "Y",...
    InterModuleGap = simscape.Value(0.005, "m"), ...
    CircuitConnection = "Series");
```

Visualize the battery ModuleAssembly object. Create the uifigure where you want to visualize your battery module assembly and use the BatteryChart object.

```
f = uifigure("Color", "white");
moduleChart = BatteryChart(Parent = f, Battery = moduleAssembly);
title(moduleChart, "Module Assembly Chart")
```



All battery objects, including modules, have a **Name** property. The ModuleAssembly object automatically assigns a unique name to all of its modules. To display the name of each module in your ModuleAssembly object, use the **Name** property.

```
disp(moduleAssembly.Module(1).Name);
```

```
Module1
```

```
disp(moduleAssembly.Module(2).Name);
```

```
Module2
```

You can modify the Name property to rename any of the modules inside a module assembly. Specify a new name for the two modules in your battery module assembly.

```
moduleAssembly.Module(1).Name = "MyModuleA";
moduleAssembly.Module(2).Name = "MyModuleB";
```

These names are the names of the subsystem block that you generate when you run the buildBattery function at the end of this example.

```
disp(moduleAssembly.Module(1).Name);
```

```
MyModuleA
```

```
disp(moduleAssembly.Module(2).Name);
```

```
MyModuleB
```

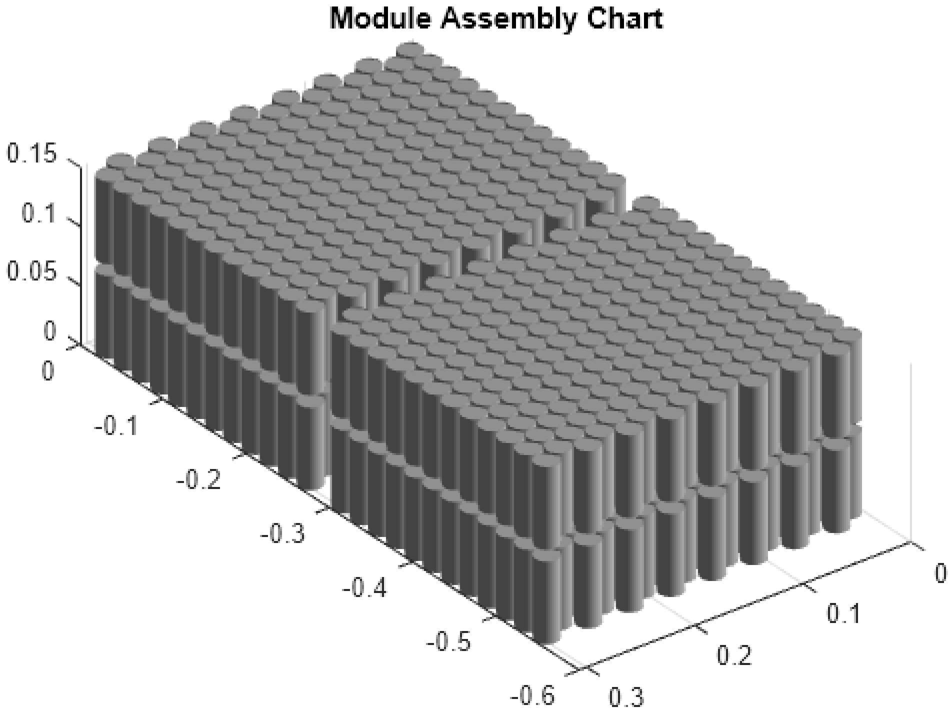
A `ModuleAssembly` battery object also allows you to stack the modules along the Z axis. To stack modules along the Z axis, use the **NumLevels** property.The NumLevels property defines the number of levels, tiers, or floors of the module assembly. The `ModuleAssembly` object stacks the modules symmetrically according to the number of levels and modules in the assembly.

For example, create a new module assembly object that comprises 4 identical modules stacked along the Z axis on two levels.

```
zStackedModuleAssembly = ModuleAssembly(...
    Module = repmat(module,1,4), ...
    StackingAxis = "Y",...
    NumLevel = 2,...
    InterModuleGap = simscape.Value(0.01, "m"));
```

Visualize the `ModuleAssembly` object, `zStackedModuleAssembly`.

```
f = uifigure("Color", "white");
moduleAssemblyChart = BatteryChart(Parent = f, Battery = zStackedModuleAssembly);
title(moduleAssemblyChart, "Module Assembly Chart")
```



Create and Visualize Battery Pack Object

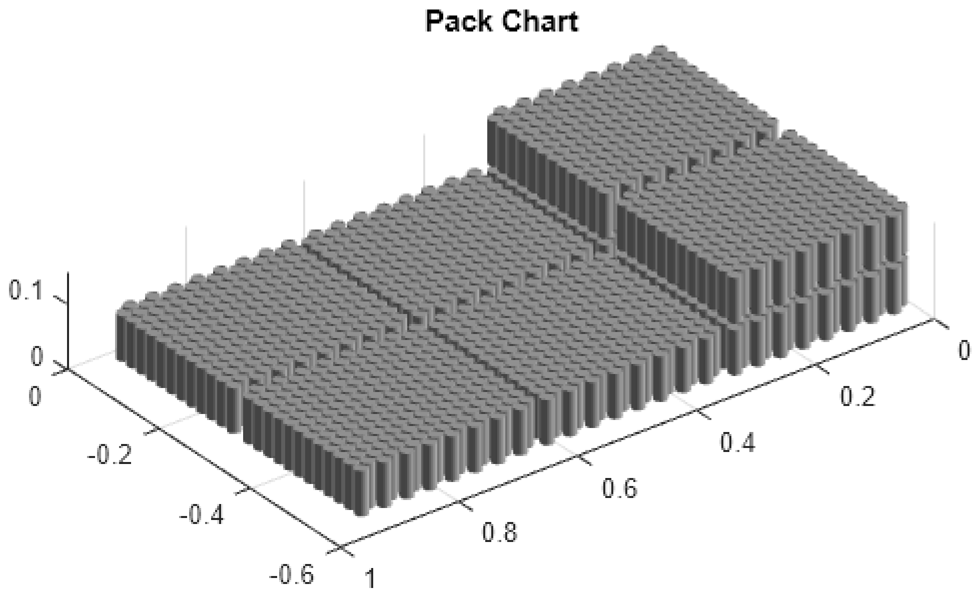
You now have all the foundational elements to create your battery pack. A battery pack comprises multiple module assemblies connected in series or in parallel. You can define the number and types of module assemblies by using the **ModuleAssembly** property. If a pack comprises many identical module assemblies, use the `repmat` function. Otherwise use an array of distinct module assemblies.

In this example, you create a battery pack of 3 module assemblies. The first module assembly is the module assembly stacked along the Z axis, zStackedModuleAssembly. The other two module assemblies are two identical module assemblies that you created in the previous step.

```
batteryPack2 = Pack(...
    ModuleAssembly = [zStackedModuleAssembly, repmat(moduleAssembly,1,2)], ...
    StackingAxis = "X",...
    InterModuleAssemblyGap = simscape.Value(0.005, "m"));
```

Visualize the battery `Pack` object. Create the uifigure where you want to visualize your battery pack and use the `BatteryChart` object.

```
f = uifigure("Color", "white");
packChart = BatteryChart(Parent = f, Battery = batteryPack2);
title(packChart, "Pack Chart")
```



The Pack object automatically assigns a unique name to all of its module assemblies upon creation. To display the name of each module assembly in your Pack object, use the **Name** property.

```
disp(batteryPack2.ModuleAssembly(1).Name);
```

```
ModuleAssembly1
```

```
disp(batteryPack2.ModuleAssembly(2).Name);
```

```
ModuleAssembly2
```

You can use a Pack object to define a common cell balancing strategy for all the modules inside the pack by specifying the **BalancingStrategy** property.

```
batteryPack2.BalancingStrategy = "Passive";
```

Modifying this property at this level automatically modifies the same property inside all of the underlying module components in the battery pack. Check the balancing strategy of the modules inside your battery pack.

```
disp(batteryPack2.ModuleAssembly(1).Module(1).BalancingStrategy);
```

```
Passive
```

```
disp(batteryPack2.ModuleAssembly(1).Module(2).BalancingStrategy);
```

```
Passive
```

The **BalancingStrategy** property of each module in the pack updated to reflect the change you have applied to the **BalancingStrategy** property of your `Pack` object.

Use the **PackagingVolume** and **CumulativeMass** properties to display the cumulative pack mass and packaging volume of your battery pack.

```
disp(batteryPack2.PackagingVolume)
```

```
    0.0471 : m^3
```

```
disp(batteryPack2.CumulativeMass)
```

```
  107.5200 : kg
```

Modify Model Resolution of Battery Objects

`ParallelAssembly` and `Module` objects have a **ModelResolution** property that allows you to set the level of fidelity of the generated Simscape model used in simulations. You can specify the ModelResolution property to either:

- `Lumped` — Lowest fidelity. The battery object uses only one electrical model with parameters scaled up according to the number of cells in series and in parallel. To obtain the fastest compilation time and running time, use this value.
- `Detailed` — Highest fidelity. The battery object uses one electrical model and one thermal model for each battery cell.
- `Grouped` — Custom simulation strategy, available only to `Module` objects.

You can view the simulation strategy by using the **SimulationStrategyVisible** property of the `BatteryChart` object.

Modify Model Resolution for ParallelAssembly Object

A `ParallelAssembly` object uses a single battery `Cell` object as foundational repeating unit upon its creation. The single cell model is scaled up depending on the model resolution of the parallel assembly. If you set the **ModelResolution** property of the parallel assembly to `"Lumped"`, all relevant model parameters (such as the resistance) are scaled up by using the number of cells connected in parallel, `P`, as a multiplication factor:

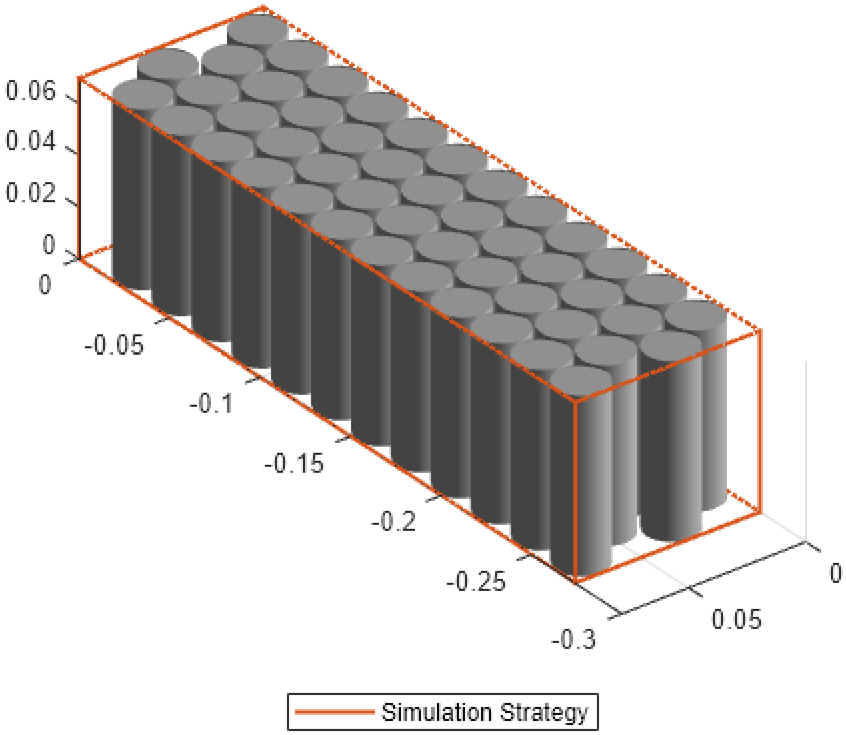
- Cell-to-ParallelAssembly open-circuit voltage (V) — $\mathrm{V0}_{\mathrm{ParallelAssembly}} = \mathrm{V0}_{\mathrm{cell}}$
- Cell-to-ParallelAssembly resistance (Ohm) — $\mathrm{R0}_{\mathrm{ParallelAssembky}} = \mathrm{R0}_{\mathrm{cell}} / P$, $\mathrm{R1}_{\mathrm{ParallelAssembky}} = \mathrm{R1}_{\mathrm{cell}} / P$
- Cell-to-ParallelAssembly capacitance (F) — $\mathrm{C1}_{\mathrm{ParallelAssembly}} = \mathrm{C1}_{\mathrm{cell}} * P$
- Cell-to-ParallelAssembly capacity (Ah) — $\mathrm{AH}_{\mathrm{ParallelAssembly}} = \mathrm{AH}_{\mathrm{cell}} * P$
- Cell-to-ParallelAssembly thermal mass (J/K) — $\mathrm{thermal\_mass}_{\mathrm{ParallelAssembly}} = \mathrm{thermal\_mass}_{\mathrm{cell}} * P$

Create a new `ParallelAssembly` object with the battery cell that you created at the beginning of this example. By default, the **ModelResoultion** property of a `ParallelAssembly` object is set to "Lumped".

```
lumpedPSet = ParallelAssembly(...
    NumParallelCells = 48, ...
    Cell = batteryCell, ...
    Rows = 4, ...
    InterCellGap = simscape.Value(0.001, "m"));
```

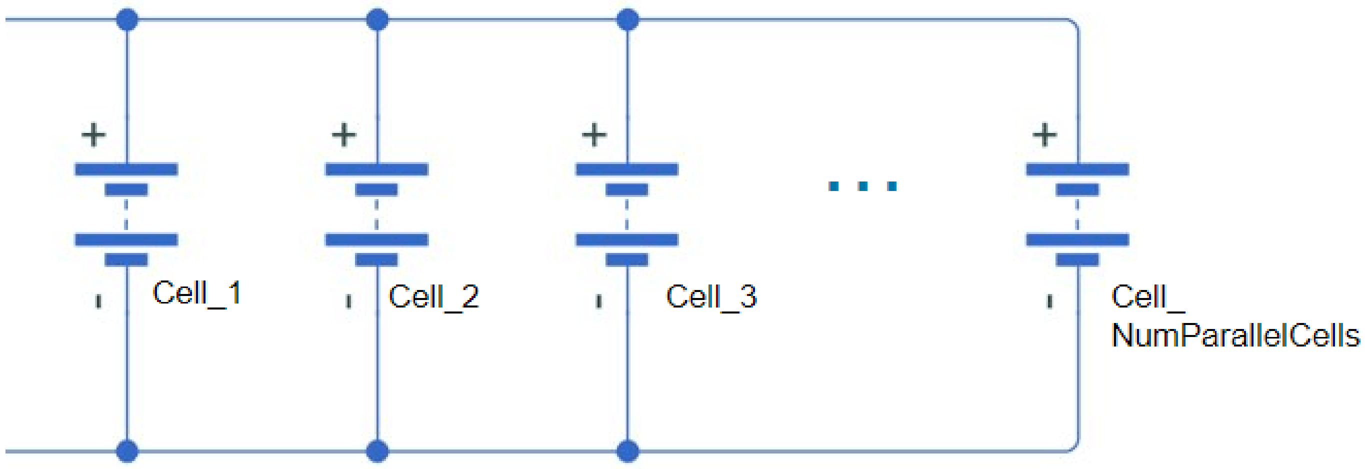
Visualize the `ParallelAssembly` object and check the model resolution by setting the **SimulationStrategyVisible** property to `"on"`.

```
f = uifigure("Color", "white");
paralllelAssemblyChartLumped = BatteryChart(Parent = f, Battery = lumpedPSet, SimulationStrategyV
```



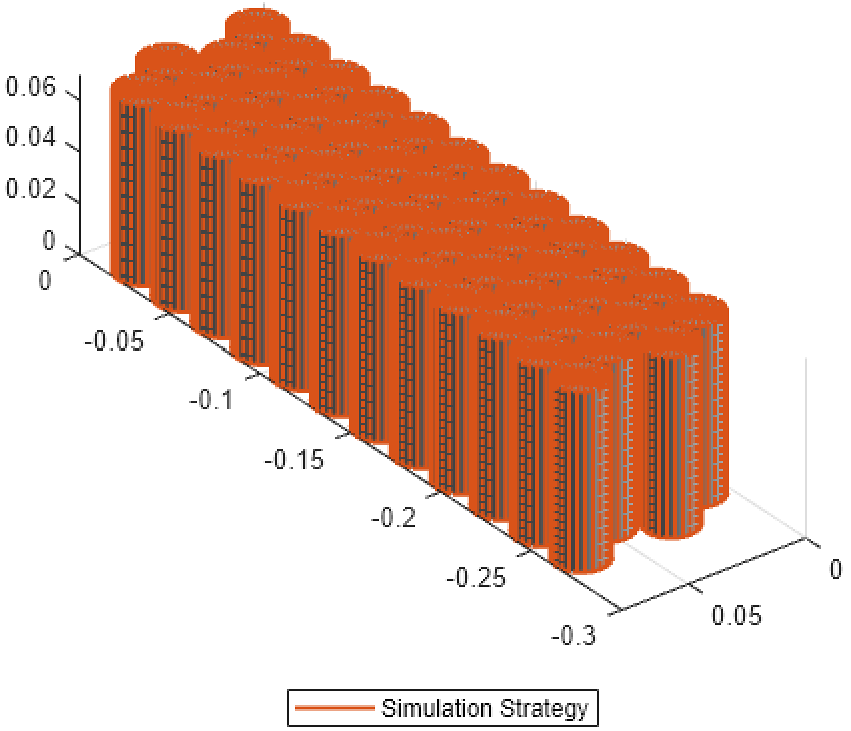
Only one single cell model block represents all the cell components inside the orange box.

If you set the **ModelResolution** property of the parallel assembly to `"Detailed"`, the `ParallelAssembly` object instantiates a number of cell model blocks equal to the value of the **NumParallelCells** property and connects them electrically in parallel in Simscape.



Change the model resolution of the previous `ParallelAssembly` object to "`Detailed`" and visualize it by using the `BatteryChart` object and by setting the **SimulationStrategyVisible** property to "`on`".

```
detailedPset = lumpedPSet;
detailedPset.ModelResolution = "Detailed";
f = uifigure("Color", "white");
paralllelAssemblyChartDetailed = BatteryChart(Parent = f, Battery = detailedPset, SimulationStrat
```



A number of cell model blocks equal to the value of the **NumParallelCells** property represents each cell component.

Modify Model Resolution for Module Object

Lumped Module Resolution

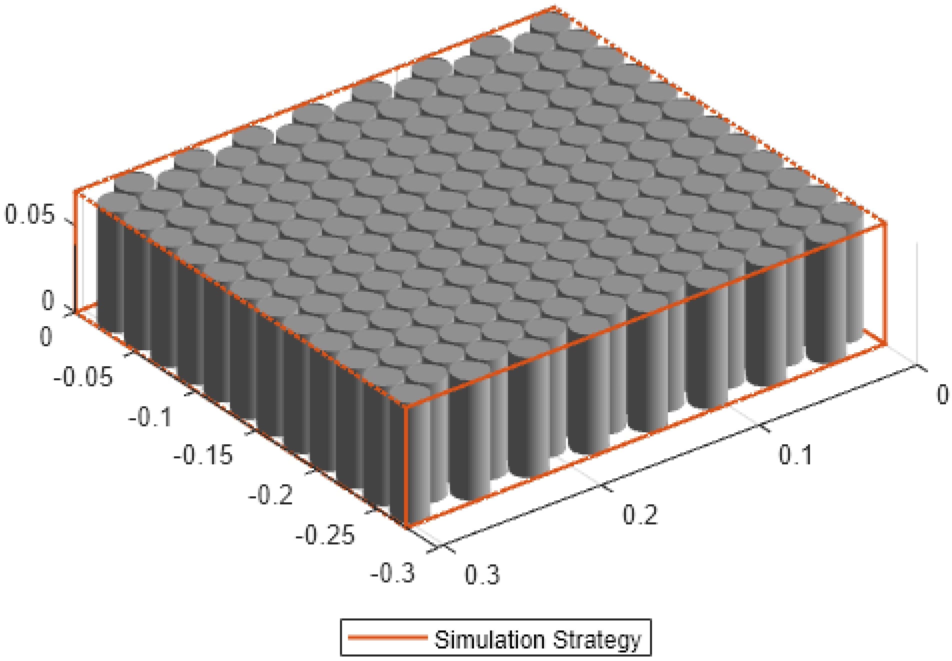
By default, the model resolution in modules and parallel assemblies is set to "Lumped". This means that the generated battery model in Simscape only uses one electrical model to electrically simulate all the battery cells within that system.

Check how the lumped module resolution works in Module objects. Create a Module object that comprises four parallel assemblies stacked along the X axis.

```
lumpedModule = Module(...
    ParallelAssembly = lumpedPSet, ...
    NumSeriesAssemblies = 4, ...
    StackingAxis = "X",...
    InterParallelAssemblyGap = simscape.Value(0.0001, "m"));
```

Visualize the Module object and check the model resolution by setting the **SimulationStrategyVisible** property to "on".

```
f = uifigure("Color", "white");
moduleChartLumped = BatteryChart(Parent = f, Battery = lumpedModule, SimulationStrategyVisible =
```



One electrical cell model simulates all the cells contained in the dotted orange box. The cell model parameters are scaled up using the number of parallel assemblies connected in series, S, and the number of cells connected in parallel, P, as a multiplication factor:

- Cell-to-Module open-circuit voltage (V) — $\mathrm{V0}_{\mathrm{ParallelAssembly}} = \mathrm{V0}_{\mathrm{cell}} * S$
- Cell-to-ParallelAssembly resistance (Ohm) — $\mathrm{R0}_{\mathrm{ParallelAssembky}} = \mathrm{R0}_{\mathrm{cell}} * S / P$, $\mathrm{R1}_{\mathrm{ParallelAssembky}} = \mathrm{R1}_{\mathrm{cell}} * S / P$
- Cell-to-ParallelAssembly capacitance (F) — $\mathrm{C1}_{\mathrm{ParallelAssembly}} = \mathrm{C1}_{\mathrm{cell}} * P / S$
- Cell-to-ParallelAssembly capacity (Ah) — $\mathrm{AH}_{\mathrm{ParallelAssembly}} = \mathrm{AH}_{\mathrm{cell}} * P$
- Cell-to-ParallelAssembly thermal mass (J/K) — $\mathrm{thermal\_mass}_{\mathrm{ParallelAssembly}} = \mathrm{thermal\_mass}_{\mathrm{cell}} * S * P$

Add thermal boundary conditions to your module. To define a thermal path to ambient, set the **AmbientThermalPath** property to `"CellBasedThermalResistance"`.

```
moduleLumped.AmbientThermalPath = "CellBasedThermalResistance";
```

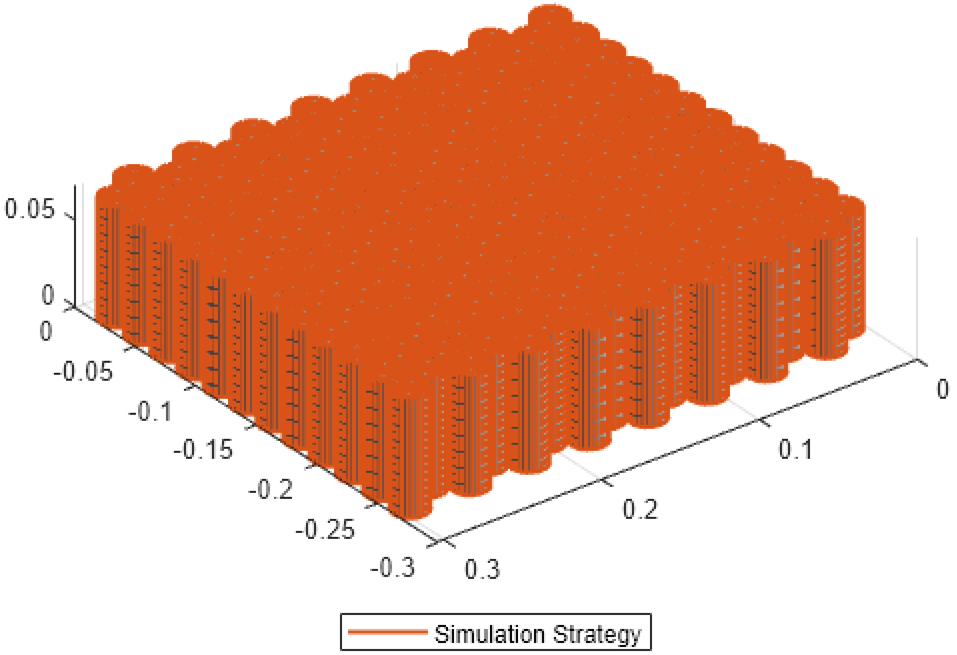
Detailed Module Resolution

Now change the model resolution of the previous `Module` object to `"Detailed"` and visualize it by using the `BatteryChart` object and by setting the **SimulationStrategyVisible** property to `"on"`.

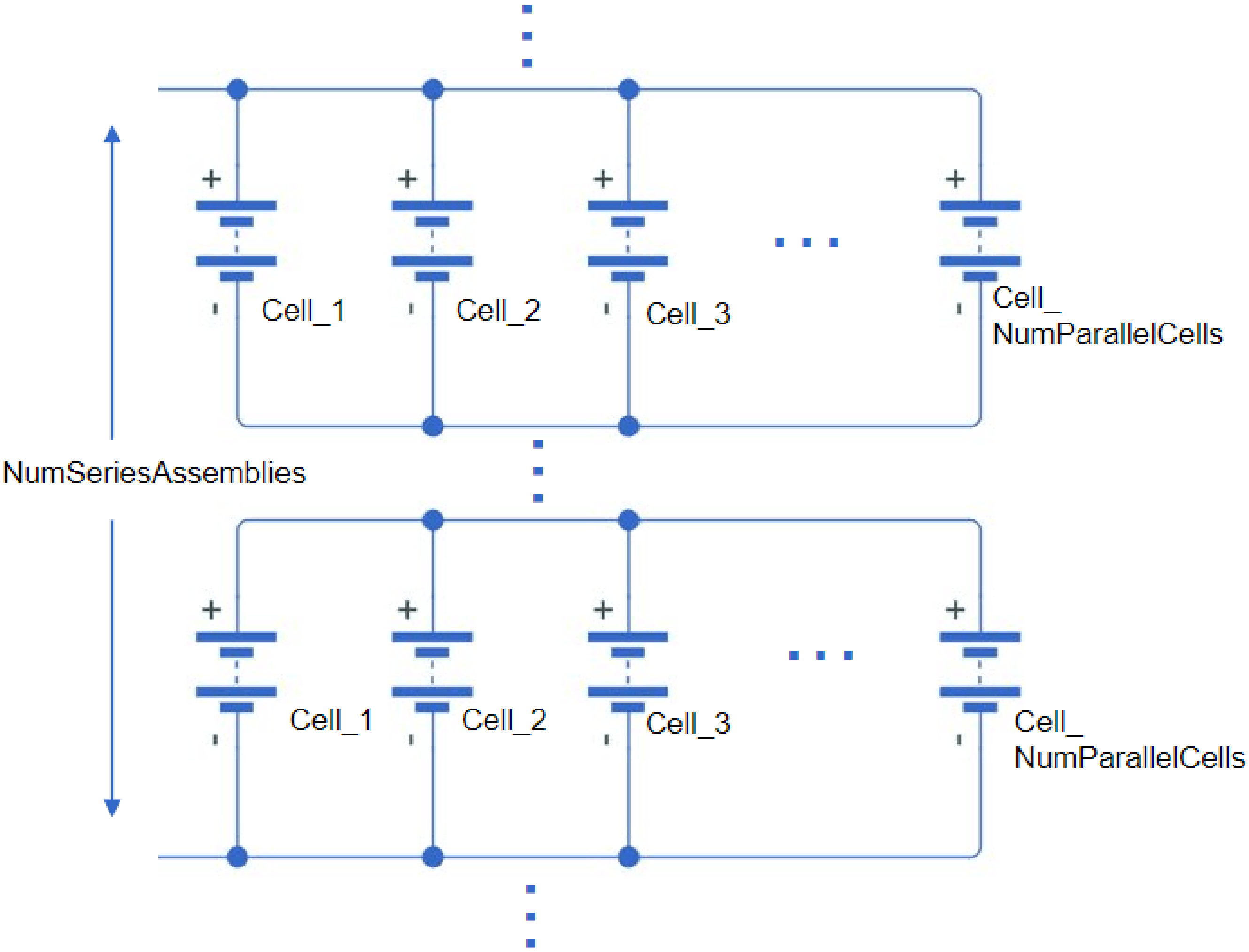
```
detailedModule = lumpedModule;
detailedModule.ParallelAssembly.ModelResolution = "Detailed";
detailedModule.ModelResolution = "Detailed";
```

For cylindrical modules, the **"Detailed"** model resolution is not recommended as many cells are present and it is important to keep the total number of models between 30 and 50.

```
f = uifigure("Color", "white");
moduleChartDetailed = BatteryChart(Parent = f, Battery = detailedModule, SimulationStrategyVisib
```



A number of cell model blocks equal to the value of the **NumParallelCells** property represents each cell component.



Add thermal boundary conditions to your detailed battery module. To define the location of a cooling plate, set the **CoolingPlate** property to `"Bottom"`.

```
detailedModule.CoolingPlate = "Bottom";
```

Grouped Module Resolution

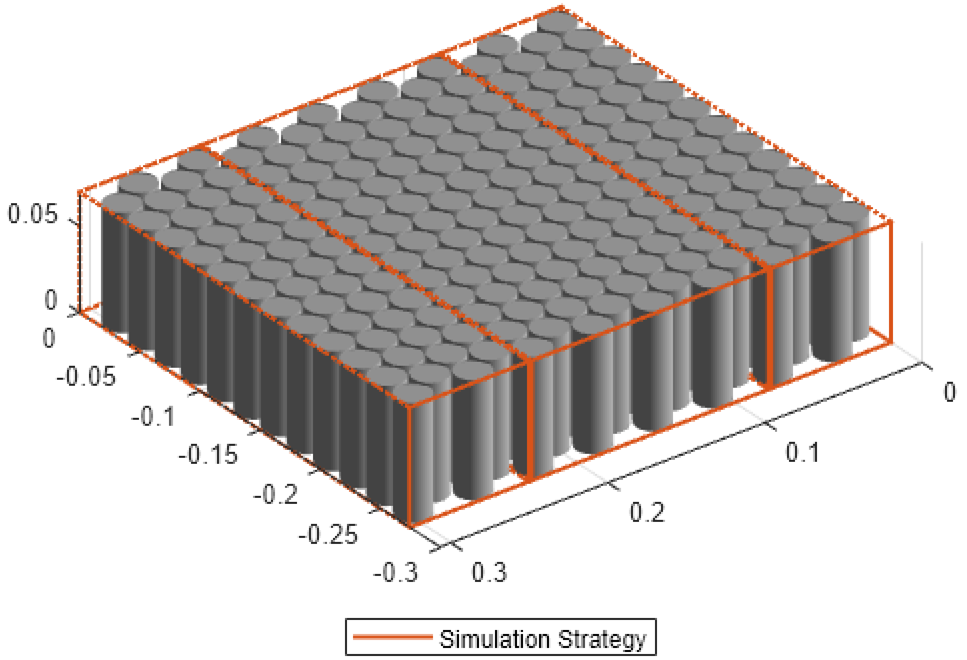
For battery modules, you can also set the **ModelResolution** property to `"Grouped"`. This simulation strategy helps increasing the model performance.

```
module.ModelResolution = "Grouped";
```

When you set the **ModelResolution** property of a module to "Grouped", you can define an additional simulation strategy by using the **SeriesGrouping** and **ParallelGrouping** properties:

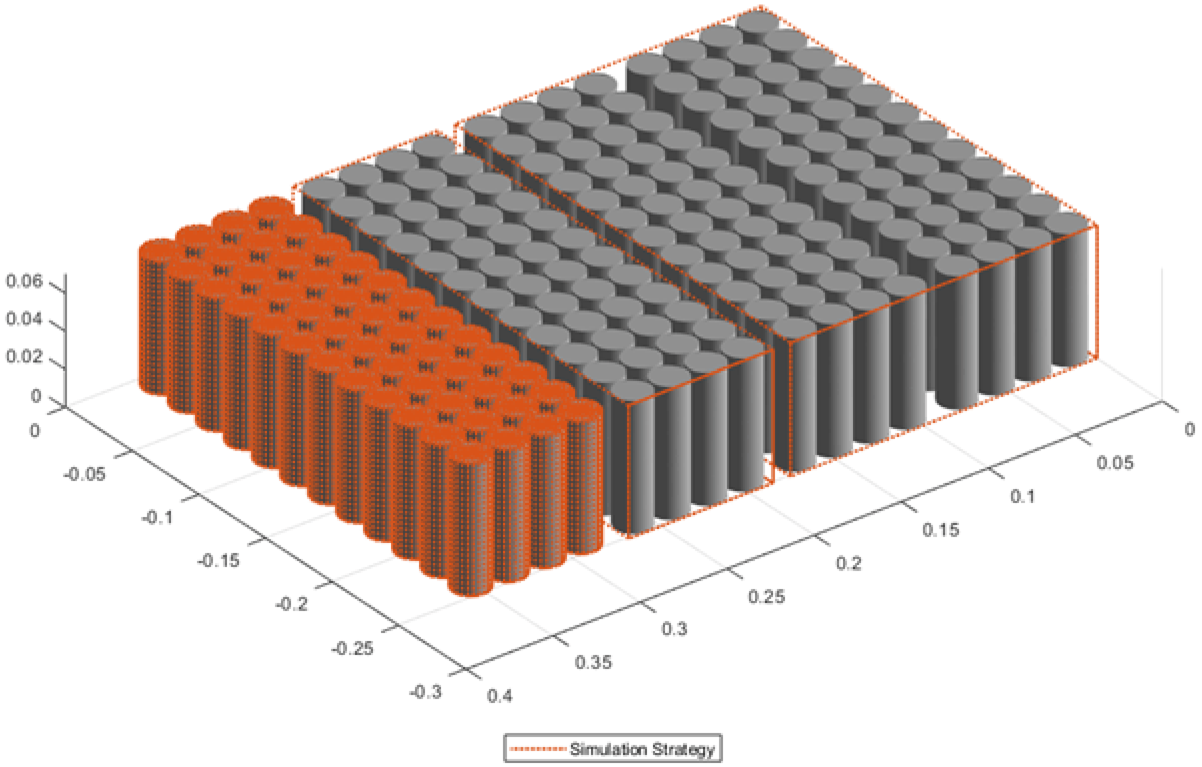
- **SeriesGrouping** — Custom modeling strategy for the module along the series connections, specified as a strictly positive array of doubles. The length of the array of this property specifies the number of individual electrical models required. Each element value of this array specifies how many parallel assemblies are lumped within the specified electrical model. The sum of the elements in the array must be equal to value of the NumSeriesAssemblies property. For example, if your module comprises four parallel assemblies (**NumSeriesAssemblies** = 4) and you set this property to `[2 1 1]`, the module is discretized in three individual electrical models where the first model comprises two of the original parallel assemblies.

```
module.SeriesGrouping = [1,2,1];
f = uifigure("Color", "white");
moduleChartGrouped = BatteryChart(Parent = f, Battery = module, SimulationStrategyVisible = "on"
```



- **ParallelGrouping** — Custom modeling strategy for the module for every parallel assembly defined in the SeriesGrouping property, specified as a strictly positive array of doubles. The length of the array of this property must be equal to the length of the array of the SeriesGrouping property. Each element of this array specifies the number of individual electrical models for every element in the array of the SeriesGrouping property. The values of the elements of this array can be equal only to either 1 or the value of the NumParallelCells property. For example, if your module comprises four parallel assemblies (**NumSeriesAssemblies** = 4), 48 cylindrical cells for each parallel assembly (**NumParallelCells** = 48), and three individual electrical models where the first model comprises two of the original parallel assemblies (**SeriesGrouping** = [2 1 1]), then if you set this property to `[1 1 48]`, the module is discretized in 50 individual electrical models where each cell of the fourth parallel assembly has an electrical model.

-



Assign Model Resolution for ModuleAssembly Object

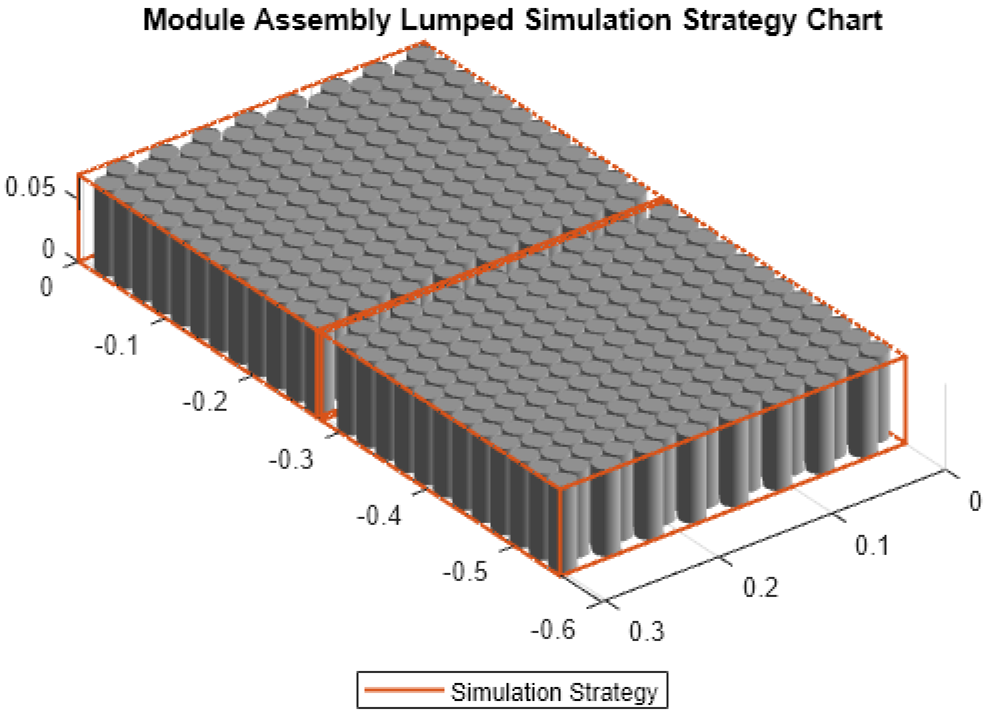
A `ModuleAssembly` object inherits the model resolution of its battery modules.

Create a `ModuleAssembly` object by using the lumpedModule `Module` object that you created in the previous step.

```
moduleAssemblyLumped = ModuleAssembly(...
    Module = repmat(lumpedModule,1,2), ...
    StackingAxis = "Y",...
    InterModuleGap = simscape.Value(0.005, "m"));
```

Then visualize the `ModuleAssembly` object and check the model resolution by setting the **SimulationStrategyVisible** property to `"on"`.

```
f = uifigure("Color", "white");
lumpedModuleAssemblyChart = BatteryChart(Parent = f, Battery = moduleAssemblyLumped , SimulationS
title(lumpedModuleAssemblyChart, "Module Assembly Lumped Simulation Strategy Chart"  )
```



The **ModelResolution** property of the `ModuleAssembly` object you just created is automatically set to `"Lumped"` because the **ModelResolution** properties of its modules are set to `"Lumped"`.

Assign Model Resolution for Pack Object

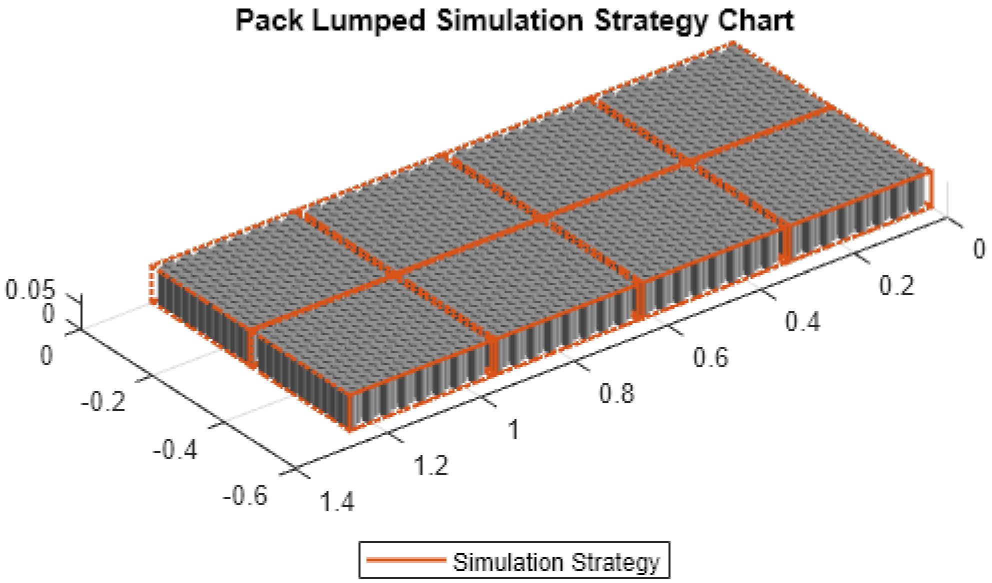
A `Pack` object inherits the model resolution of its battery module assemblies.

Create a `Pack` object by using the moduleAssemblyLumped `ModuleAssembly` object that you created in the previous step.

```
packLumped = Pack(...
    ModuleAssembly = repmat(moduleAssemblyLumped,1,4), ...
    StackingAxis = "X",...
    InterModuleAssemblyGap = simscape.Value(0.01, "m"));
```

Then visualize the `Pack` object and check the model resolution by setting the **SimulationStrategyVisible** property to `"on"`.

```
f = uifigure("Color", "white");
packLumpedChart = BatteryChart(Parent = f, Battery = packLumped , SimulationStrategyVisible = "o
title(packLumpedChart, "Pack Lumped Simulation Strategy Chart")
```


Pack Lumped Simulation Strategy Chart

The **ModelResolution** property of the `Pack` object you just created is automatically set to `"Lumped"` because the **ModelResolution** properties of its module assemblies are set to `"Lumped"`.

Build Simscape Model for the Battery Objects

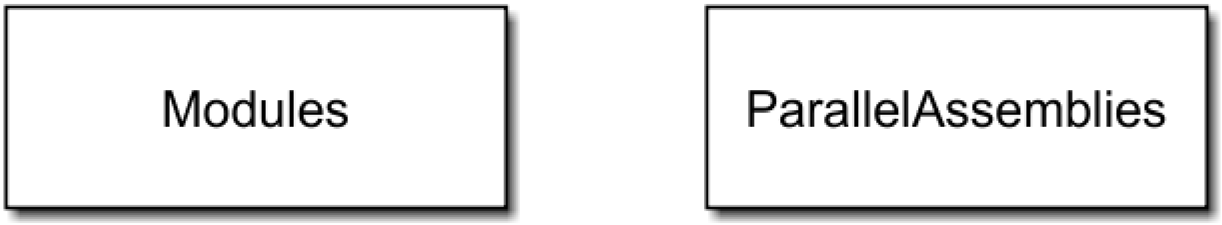
After you have created your battery objects, you need to convert them into Simscape models to be able to use them in block diagrams. You can then use these models as reference for your system integration and requirement evaluation, cooling system design, control strategy development, hardware-in-the-loop, and much more.

To create a library that contains the Simscape Battery model of all the batteries object you created in this example, use the `buildBattery` function.

```
buildBattery(packLumped,"LibraryName","cylindricalPackExample");
```

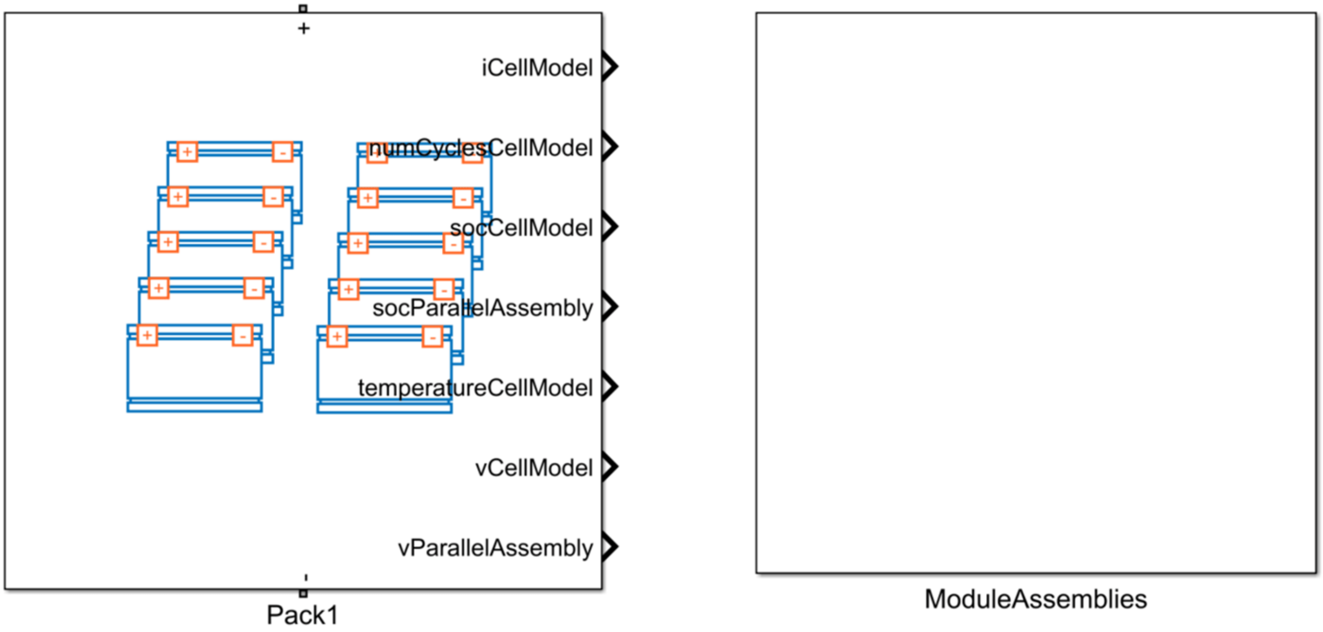
```
Generating Simulink library 'cylindricalPackExample_lib' in the current directory 'C:\TEMP\Bdoc22
```

This function creates the `cylindricalPackExample_lib` and `cylindricalPackExample` SLX library files in your working directory. The `cylindricalPackExample_lib` library contains the Modules and ParallelAssemblies sublibraries.



To access the Simscape models of your `Module` and `ParallelAssembly` objects, open the `cylindricalPackExample_lib`. SLX file, double-click the sublibrary, and drag the Simscape blocks in your model.

The `cylindricalPackExample` library contains the Simscape models of your `ModuleAssembly` and `Pack` objects.



The Simscape models of your `ModuleAssembly` and `Pack` objects are subsystems. You can look inside these subsystems by opening the `packLibrary` SLX file and double-click the subsystem.

For more information, see the `buildBattery` documentation page.

Build Detailed Model of Battery Pack From Pouch Cells

This example shows how to create and build Simscape™ system models for various battery designs and configurations based on pouch battery cells in Simscape™ Battery™. The `buildBattery` function allows you to automatically generate Simscape models for these Simscape Battery objects:

- `ParallelAssembly`
- `Module`
- `ModuleAssembly`
- `Pack`

This function creates a library in your working folder that contains a system model block of a battery pack that you can use as reference in your simulations. The run-time parameters for these models, such as the battery cell impedance or the battery open-circuit voltage, are defined after the model creation and are therefore not covered by the Battery Pack Builder classes. To define the run-time parameters, you can either specify them in the block mask of the generated Simscape models or use the **MaskParameters** argument of the `buildBattery` function.

During the first half of this example, you first define the key properties of a pouch battery cell and block model. You then use this pouch battery cell as a fundamental repeating unit inside a parallel assembly component. In the industry this component is also called a "sub-module", a "super-cell", a "P-set", or just a "cell". You later employ this parallel assembly to define a battery module, which is then used to create a module assembly and finally a battery pack. These larger battery systems all use the battery cell as a fundamental repeating unit. Throughout the workflow, you visualize the geometry and the relative positioning of these battery systems by using the `BatteryChart` object.

In the second half of the example, you modify the modeling methodology and the model resolution of the `Module`, `ModuleAssemblies`, and `Pack` objects before generating the final Simscape battery model. You can perform the geometrical aggregation or stacking of any battery object along the sequence either along the `X` or `Y` axis. These axis mirror the Vehicle Coordinate System (Z-up).

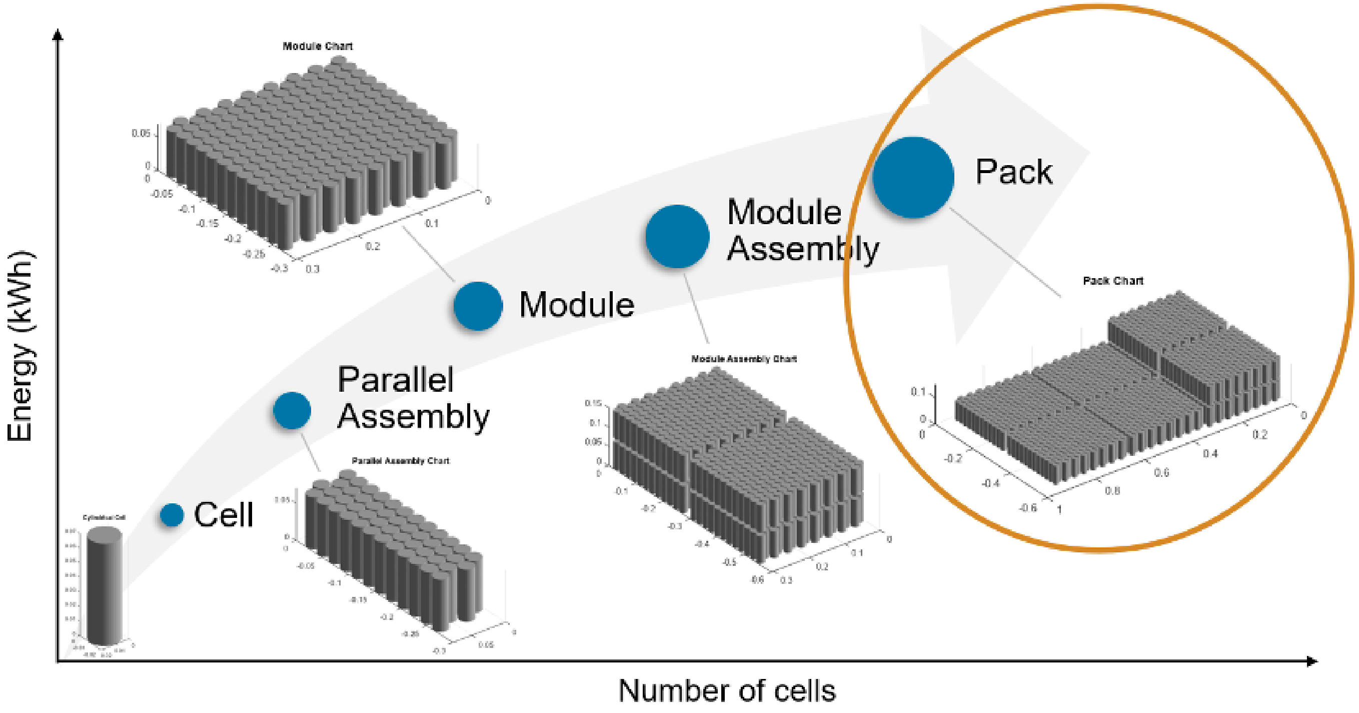
To use the functions and objects in Simscape Battery, first import the required Simscape Battery package:

```
import simscape.battery.builder.*
```

Create and Visualize Battery Objects in MATLAB

To create a battery pack, you must first design and create the foundational elements of the battery pack.

This uifigure shows the overall process to create a battery pack object in a bottom-up approach:



A battery pack comprises multiple module assemblies. These module assemblies, in turn, comprise a number of battery modules connected electrically in series or in parallel. The battery modules are made of multiple parallel assemblies which, in turn, comprise a number of battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement.

Create and Visualize Battery Cell Object

A battery cell is an electrochemical energy storage device that provides electrical energy from stored chemical energy. An electrochemical battery cell is the fundamental building block in the manufacturing of larger battery systems. To obtain the required energy and voltage levels, multiple battery cells are typically connected electrically in parallel and/or in series.

To mirror the real-world behavior, the Simscape Battery `Cell` object is the foundational element for the creation of a battery pack system model. You can create all battery classes without any inputs. To create a battery cell, use the `Cell` object.

```
batteryCell = Cell();
```

To meet the battery packaging and space requirements, you can arrange the battery cells in three main geometrical arrangements: cylindrical, pouch, or prismatic. To be able to visualize a single battery cell, you must first define its geometry.

Define a pouch geometry by using the `PouchGeometry` object.

```
cellGeometry = PouchGeometry();
```

The `PouchGeometry` object has six properties:

- **Length** — Length of the pouch geometry, specified as a `simscape.Value` object that represents a scalar with a unit of length.

- **Thickness** — Thickness of the pouch geometry, specified as a `simscape.Value` object that represents a scalar with a unit of length.
- **Height** — Height of the pouch geometry, specified as a `simscape.Value` object that represents a scalar with a unit of length.
- **TabLocation** — Location of the tabs of a pouch battery cell, specified as either `Standard` or `Opposed`.
- **TabWidth** — Width of the tab of a pouch battery cell, specified as a `simscape.Value` object that represents a scalar with a unit of length.
- **TabHeight** — Height of the tab of a pouch battery cell, specified as a `simscape.Value` object that represents a scalar with a unit of length.

Specify custom values for the Length, **Height**, **TabWidth**, and **TabLocation** properties of the pouch geometry.

```
cellGeometry.Length = simscape.Value(0.36, "m");
cellGeometry.Height = simscape.Value(0.13, "m");
cellGeometry.TabWidth = simscape.Value(0.05, "m");
cellGeometry.TabLocation = "Opposed";
```

For more information on the possible geometrical arrangements of a battery cell, see the `CylindricalGeometry` and `PrismaticGeometry` documentation pages.

You can now link this geometry object to the battery cell by accessing the Geometry property of the `batteryCell` object.

```
batteryCell.Geometry = cellGeometry;
```

Specify a custom value for the mass of the battery cell by using the **Mass** property.

```
batteryCell.Mass = simscape.Value(0.8,"kg");
disp(batteryCell)
```

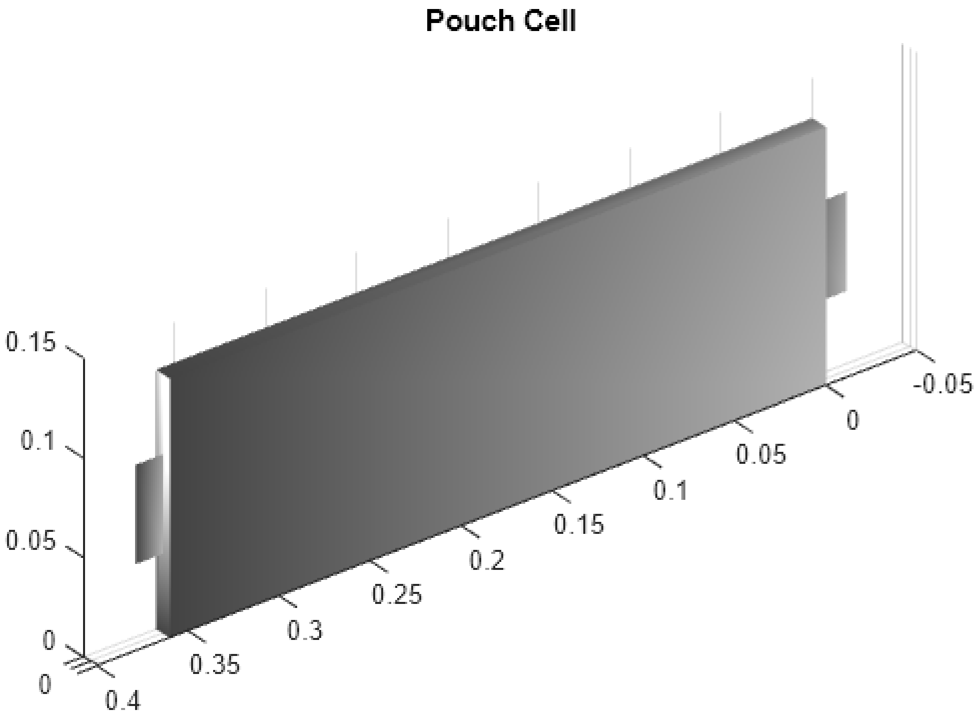
```
  Cell with properties:

            Geometry: [1x1 simscape.battery.builder.PouchGeometry]
    CellModelOptions: [1x1 simscape.battery.builder.CellModelBlock]
                Mass: [1x1 simscape.Value]

Show all properties
```

Visualize the battery cell by using the `BatteryChart` object. Create the uifigure where you want to visualize your battery cell.

```
f = uifigure("Color", "white");
```

Then use the `BatteryChart` object to visualize the battery cell.

```
cellChart = BatteryChart(Parent = f, Battery = batteryCell);
title(cellChart, "Pouch Cell")
```



By default, the Battery (Table-Based) block is the electrical and thermal model used to represent and simulate this battery cell in Simscape. When scaled up into larger battery systems like a parallel assembly or a module, this model is also scaled up accordingly depending on the model resolution. To display the information about the cell model block, use the **CellModelOptions** property of the `batteryCell` object.

```
disp(batteryCell.CellModelOptions.CellModelBlockPath);
```

```
batt_lib/Cells/Battery
(Table-Based)
```

The `Cell` object also allows you to simulate the thermal effects of the battery cell by using a simple 1-D model. To simulate the thermal effects of the battery cell, in the **BlockParameters** property of the **CellModelOptions** property of the `Cell` object, set the **thermal_port** parameter to `"model"`.

```
batteryCell.CellModelOptions.BlockParameters.thermal_port = "model";
```

You can modify all the conditional parameters of the Battery (Table-Based) block by using the **CellModelOptions** property.

```
disp(batteryCell.CellModelOptions.BlockParameters);
```

```
          T_dependence: no
          thermal_port: model
           prm_age_OCV: OCV
```

```
      prm_age_capacity: disabled
    prm_age_resistance: disabled
      prm_age_modeling: equation
               prm_dyn: off
               prm_dir: noCurrentDirectionality
              prm_fade: disabled
              prm_leak: disabled
```

Create and Visualize Battery ParallelAssembly Object

A battery parallel assembly comprises multiple battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement. You can specify the number of cells connected in parallel by using the **NumParallelCells** property.

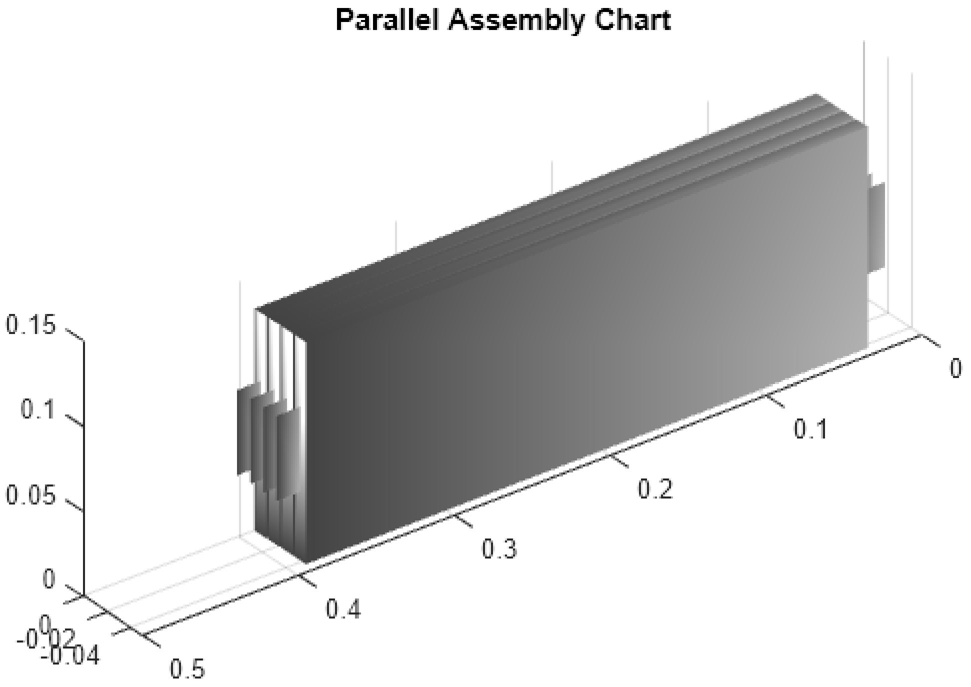
In this example, you create a parallel assembly using four of the cylindrical cells created in the previous step, stacked in a single stack topology with a gap between the cells equal to 0.001 meters.

```
parallelAssembly = ParallelAssembly(...
    NumParallelCells = 4, ...
    Cell = batteryCell, ...
    Topology = "SingleStack", ...
    InterCellGap = simscape.Value(0.001, "m"));
```

The **Topology** property is a function of the cell format. For pouch cells, the onlyavailable topology is `"SingleStack"`. By default, the ParallelAssembly object stacks the cells along the `Y` axis.

Visualize the battery parallel assembly. Create the uifigure where you want to visualize your battery parallel assembly and use the `BatteryChart` object.

```
f = uifigure("Color", "white");
parallelAssemblyChart = BatteryChart(Parent = f, Battery = parallelAssembly);
title(parallelAssemblyChart, "Parallel Assembly Chart")
```



You can modify all the public properties inside the parallel assembly after its creation.

You can check the cell packaging volume and the mass of any battery by accessing the **PackagingVolume** and **CumulativeMass** properties.

```
disp(parallelAssembly.PackagingVolume)
```

```
    0.0022 : m^3
```

```
disp(parallelAssembly.CumulativeMass)
```

```
    3.2000 : kg
```

Create and Visualize Battery Module Object

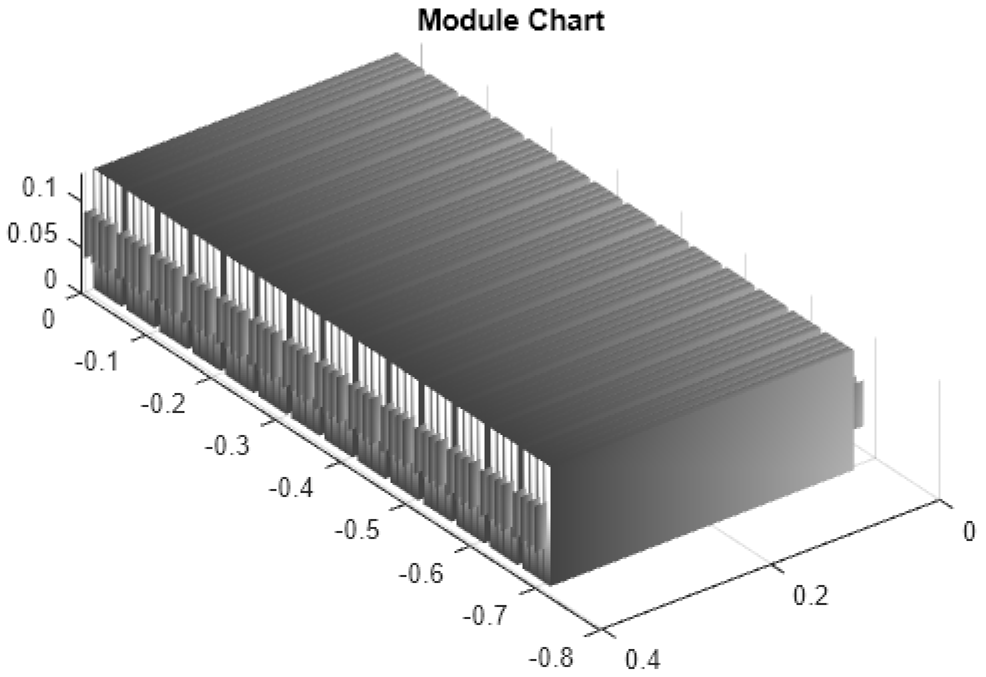
A battery module comprises multiple parallel assemblies connected in series. You can specify the number of parallel assemblies connected in series by using the **NumSeriesAssemblies** property. You can stack or geometrically assemble batteries along the X or Y axis of a Cartesian coordinate system by using the **StackingAxis** property.

In this example, you create a battery module using 14 parallel assemblies that you created in the previous step with an intergap between each assembly equal to 0.008 meters.

```
module = Module(...
    ParallelAssembly = parallelAssembly, ...
    NumSeriesAssemblies = 14, ...
    InterParallelAssemblyGap = simscape.Value(0.008, "m"));
```

Visualize the battery `Module` object. Create the uifigure where you want to visualize your battery module and use the `BatteryChart` object.

```
f = uifigure("Color", "white");
moduleChart = BatteryChart(Parent = f, Battery = module);
title(moduleChart, "Module Chart")
```



Display the total packacing volume and cumulative mass of your battery module.

```
disp(module.PackagingVolume)
```

```
    0.0358 : m^3
```

```
disp(module.CumulativeMass)
```

```
   44.8000 : kg
```

You can modify all the public properties inside the module after its creation.

Create and Visualize Battery ModuleAssembly Object

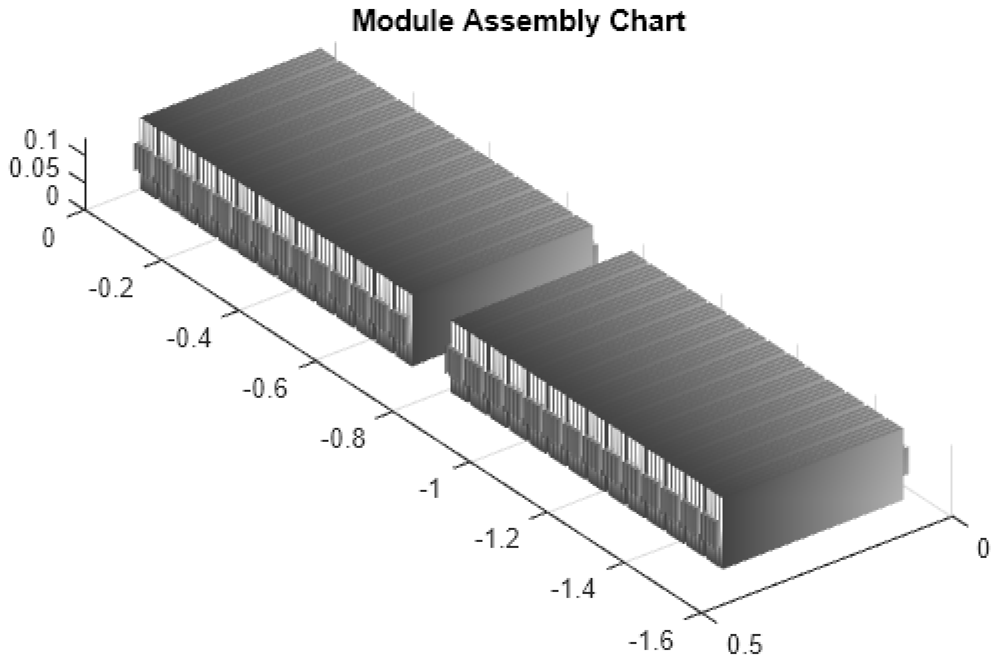
A battery module assembly comprises multiple battery modules connected in series or in parallel. You can define the number and types of modules by using the **Module** property. If a module assembly comprises many identical modules, use the repmat function. Otherwise use an array of distinct modules.

In this example, you create a battery module assembly by using two identical modules of the `Module` object you created in the previous step, with an intergap between each module equal to 0.1 meters. By default, the `ModuleAssembly` object electrically connects the modules in series.

```
moduleAssembly = ModuleAssembly(...
    Module = repmat(module,1,2), ...
    InterModuleGap = simscape.Value(0.1, "m"));
```

Visualize the battery `ModuleAssembly` object. Create the uifigure where you want to visualize your battery module assembly and use the `BatteryChart` object.

```
f = uifigure("Color", "white");
moduleChart = BatteryChart(Parent = f, Battery = moduleAssembly);
title(moduleChart, "Module Assembly Chart")
```



All battery objects, including modules, have a **Name** property. The ModuleAssembly object automatically assigns a unique name to all of its modules. To display the name of each module in your `ModuleAssembly` object, use the **Name** property.

```
disp(moduleAssembly.Module(1).Name);
```

```
Module1
```

```
disp(moduleAssembly.Module(2).Name);
```

```
Module2
```

You can modify the Name property to rename any of the modules inside a module assembly. Specify a new name for the two modules in your battery module assembly.

```
moduleAssembly.Module(1).Name = "MyModuleA";
moduleAssembly.Module(2).Name = "MyModuleB";
```

These names are the names of the subsystem block that you generate when you run the buildBattery function at the end of this example.

```
disp(moduleAssembly.Module(1).Name);
```

```
MyModuleA
```

```
disp(moduleAssembly.Module(2).Name);
```

```
MyModuleB
```

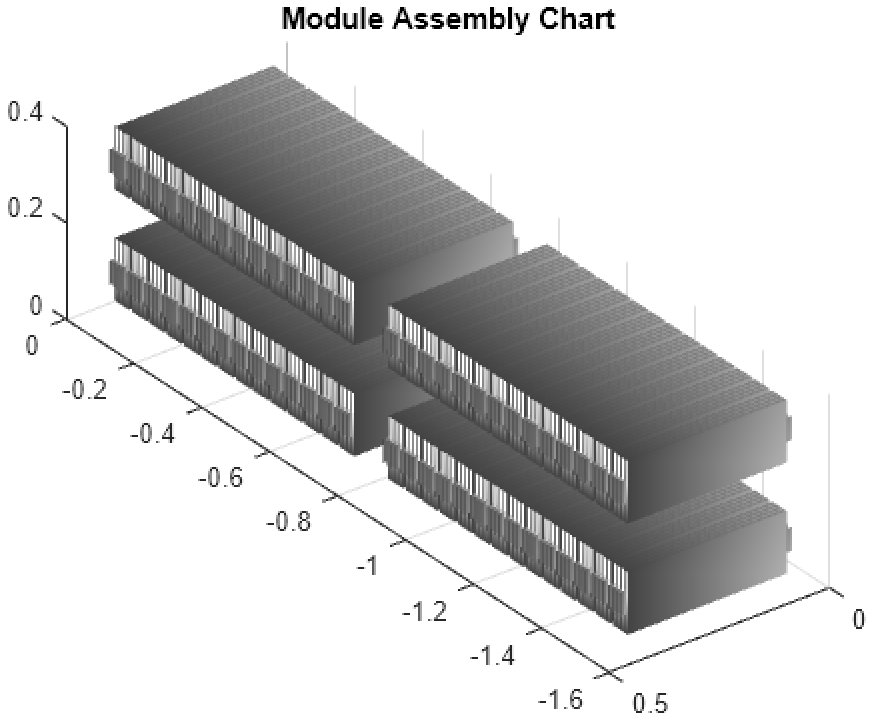
A `ModuleAssembly` battery object also allows you to stack the modules along the Z axis. To stack modules along the Z axis, use the **NumLevels** property.The NumLevels property defines the number of levels, tiers, or floors of the module assembly. The `ModuleAssembly` object stacks the modules symmetrically according to the number of levels and modules in the assembly.

For example, create a new module assembly object that comprises 4 identical modules stacked along the Z axis on two levels.

```
zStackedModuleAssembly = ModuleAssembly(...
    Module = repmat(module,1,4), ...
    NumLevel = 2,...
    InterModuleGap = simscape.Value(0.1, "m"));
```

Visualize the `ModuleAssembly` object, `zStackedModuleAssembly`.

```
f = uifigure("Color", "white");
moduleAssemblyChart = BatteryChart(Parent = f, Battery = zStackedModuleAssembly);
title(moduleAssemblyChart, "Module Assembly Chart")
```



Create and Visualize Battery Pack Object

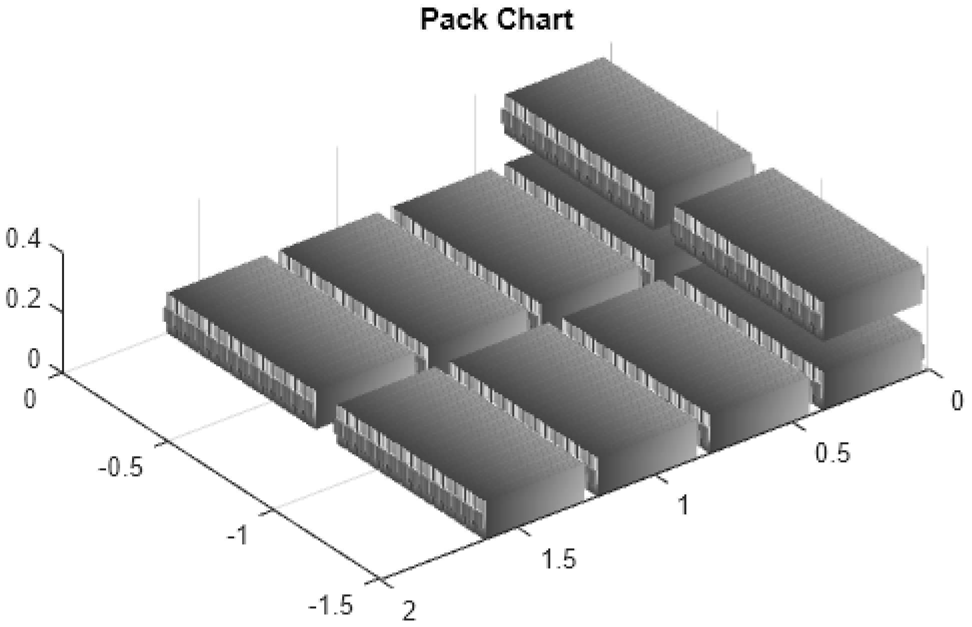
You now have all the foundational elements to create your battery pack. A battery pack comprises multiple module assemblies connected in series or in parallel. You can define the number and types of module assemblies by using the **ModuleAssembly** property. If a pack comprises many identical module assemblies, use the `repmat` function. Otherwise use an array of distinct module assemblies.

In this example, you create a battery pack of 4 module assemblies. The first module assembly is the module assembly stacked along the Z axis, zStackedModuleAssembly. The other three module assemblies are three identical module assemblies that you created in the previous step.

```
batteryPack2 = Pack(...
    ModuleAssembly = [zStackedModuleAssembly, repmat(moduleAssembly,1,3)], ...
    StackingAxis = "X",...
    InterModuleAssemblyGap = simscape.Value(0.005, "m"));
```

Visualize the battery `Pack` object. Create the uifigure where you want to visualize your battery pack and use the `BatteryChart` object.

```
f = uifigure("Color", "white");
packChart = BatteryChart(Parent = f, Battery = batteryPack2);
title(packChart, "Pack Chart")
```



The Pack object automatically assigns a unique name to all of its module assemblies upon creation. To display the name of each module assembly in your Pack object, use the **Name** property.

```
disp(batteryPack2.ModuleAssembly(1).Name);
```

```
ModuleAssembly1
```

```
disp(batteryPack2.ModuleAssembly(2).Name);
```

```
ModuleAssembly2
```

```
disp(batteryPack2.ModuleAssembly(3).Name);
```

```
ModuleAssembly3
```

```
disp(batteryPack2.ModuleAssembly(4).Name);
```

```
ModuleAssembly4
```

You can use a Pack object to define a common cell balancing strategy for all the modules inside the pack by specifying the **BalancingStrategy** property.

```
batteryPack2.BalancingStrategy = "Passive";
```

Modifying this property at this level automatically modifies the same property inside all of the underlying module components in the battery pack. Check the balancing strategy of the modules inside your battery pack.

```
disp(batteryPack2.ModuleAssembly(1).Module(1).BalancingStrategy);
```

```
Passive
```

```
disp(batteryPack2.ModuleAssembly(2).Module(1).BalancingStrategy);
```

```
Passive
```

```
disp(batteryPack2.ModuleAssembly(3).Module(1).BalancingStrategy);
```

```
Passive
```

```
disp(batteryPack2.ModuleAssembly(4).Module(1).BalancingStrategy);
```

```
Passive
```

The **BalancingStrategy** property of each module in the pack updated to reflect the change you have applied to the **BalancingStrategy** property of your Pack object.

Use the **PackagingVolume** and **CumulativeMass** properties to display the cumulative pack mass and packaging volume of your battery pack.

```
disp(batteryPack2.PackagingVolume)
```

```
    0.3579 : m^3
```

```
disp(batteryPack2.CumulativeMass)
```

```
   448 : kg
```

Modify Model Resolution of Battery Objects

ParallelAssembly and Module objects have a **ModelResolution** property that allows you to set the level of fidelity of the generated Simscape model used in simulations. You can specify the **ModelResolution** property to either:

- Lumped — Lowest fidelity. The battery object uses only one electrical model with parameters scaled up according to the number of cells in series and in parallel. To obtain the fastest compilation time and running time, use this value.
- Detailed — Highest fidelity. The battery object uses one electrical model and one thermal model for each battery cell.
- Grouped — Custom simulation strategy, available only to Module objects.

You can view the simulation strategy by using the **SimulationStrategyVisible** property of the BatteryChart object.

Modify Model Resolution for ParallelAssembly Object

A ParallelAssembly object uses a single battery Cell object as foundational repeating unit upon its creation. The single cell model is scaled up depending on the model resolution of the parallel assembly. If you set the **ModelResolution** property of the parallel assembly to "Lumped", all relevant model parameters (such as the resistance) are scaled up by using the number of cells connected in parallel, P, as a multiplication factor:

- Cell-to-ParallelAssembly open-circuit voltage (V) — $\mathrm{V0}_{\mathrm{ParallelAssembly}} = \mathrm{V0}_{\mathrm{cell}}$
- Cell-to-ParallelAssembly resistance (Ohm) — $\mathrm{R0}_{\mathrm{ParallelAssembky}} = \mathrm{R0}_{\mathrm{cell}} / P$, $\mathrm{R1}_{\mathrm{ParallelAssembky}} = \mathrm{R1}_{\mathrm{cell}} / P$

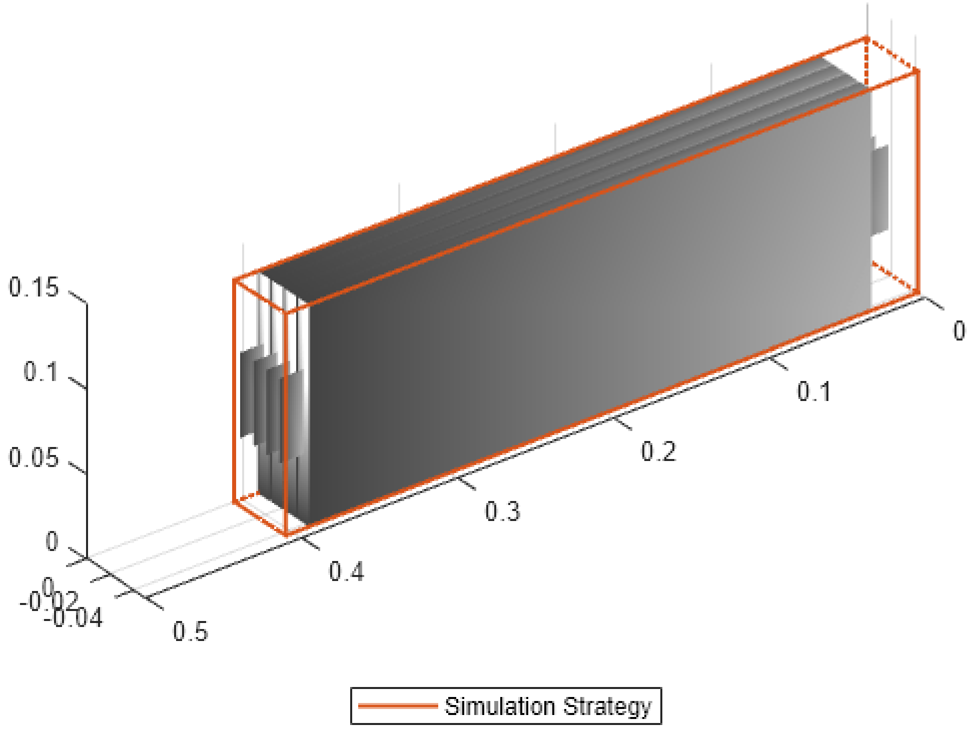
- Cell-to-ParallelAssembly capacitance (F) — $C1_{ParallelAssembly} = C1_{cell} * P$
- Cell-to-ParallelAssembly capacity (Ah) — $AH_{ParallelAssembly} = AH_{cell} * P$
- Cell-to-ParallelAssembly thermal mass (J/K) — $thermal\_mass_{ParallelAssembly} = thermal\_mass_{cell} * P$

Create a new `ParallelAssembly` object with the battery cell that you created at the beginning of this example. By default, the **ModelResoultion** property of a `ParallelAssembly` object is set to "Lumped".

```
lumpedParallelAssembly = ParallelAssembly(...
    NumParallelCells = 4, ...
    Cell = batteryCell, ...
    Topology = "SingleStack", ...
    InterCellGap = simscape.Value(0.001, "m"));
```

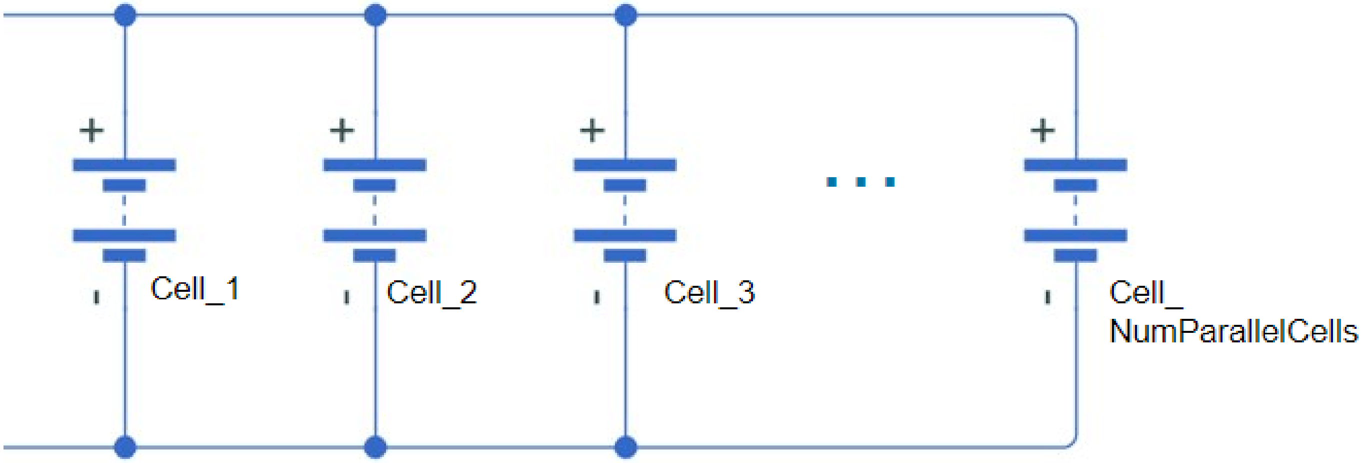
Visualize the `ParallelAssembly` object and check the model resolution by setting the **SimulationStrategyVisible** property to `"on"`.

```
f = uifigure("Color", "white");
paralllelAssemblyChartLumped = BatteryChart(Parent = f, Battery = lumpedParallelAssembly, Simula
```



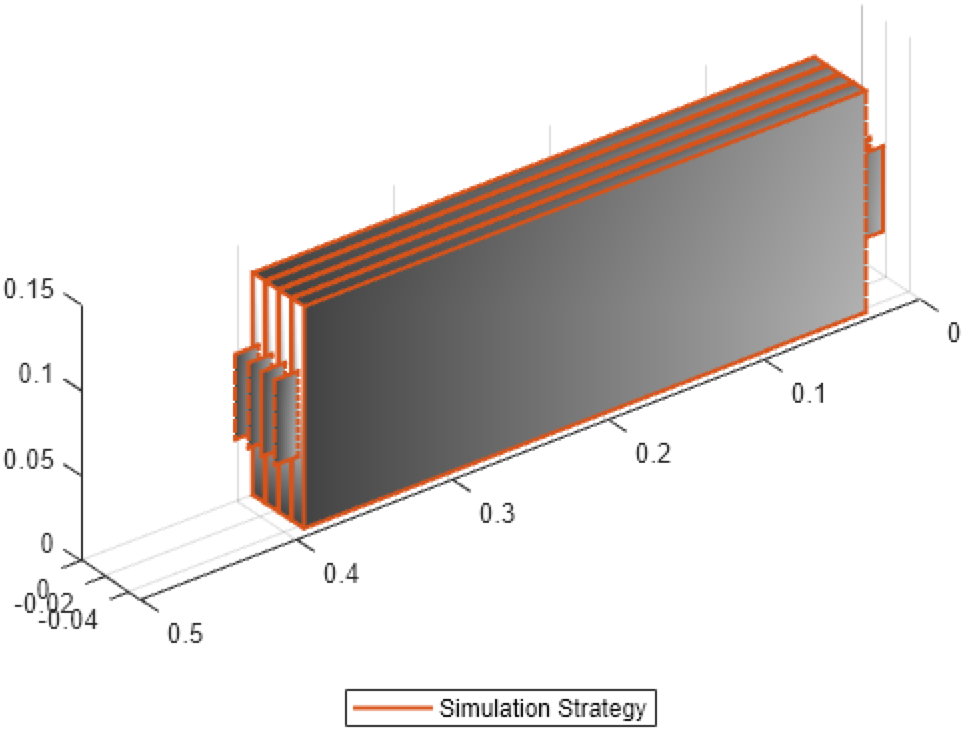
Only one single cell model block represents all the cell components inside the orange box.

If you set the **ModelResolution** property of the parallel assembly to `"Detailed"`, the `ParallelAssembly` object instantiates a number of cell model blocks equal to the value of the **NumParallelCells** property and connects them electrically in parallel in Simscape.



Change the model resolution of the previous ParallelAssembly object to "Detailed" and visualize it by using the BatteryChart object and by setting the **SimulationStrategyVisible** property to "on".

```
detailedPset = lumpedParallelAssembly;
detailedPset.ModelResolution = "Detailed";

f = uifigure("Color", "white");
paralllelAssemblyChartDetailed = BatteryChart(Parent = f, Battery = detailedPset, SimulationStrat
```



A number of cell model blocks equal to the value of the **NumParallelCells** property represents each cell component.

Modify Model Resolution for Module Object

Lumped Module Resolution

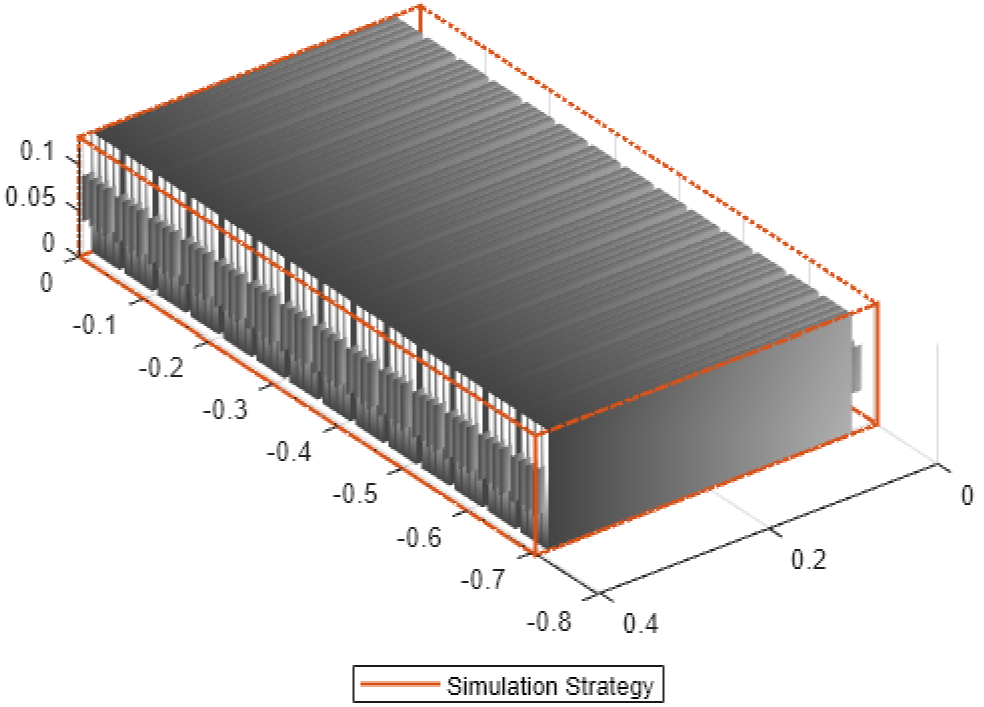
By default, the model resolution in modules and parallel assemblies is set to `"Lumped"`. This means that the generated battery model in Simscape only uses one electrical model to electrically simulate all the battery cells within that system.

Check how the lumped module resolution works in `Module` objects. Create a `Module` object that comprises 14 parallel assemblies.

```
lumpedModule = Module(...
    ParallelAssembly = parallelAssembly, ...
    NumSeriesAssemblies = 14, ...
    InterParallelAssemblyGap = simscape.Value(0.008, "m"), ...
    ModelResolution = "Lumped");
```

Visualize the `Module` object and check the model resolution by setting the **SimulationStrategyVisible** property to `"on"`.

```
f = uifigure("Color", "white");
moduleChartLumped = BatteryChart(Parent = f, Battery = lumpedModule, SimulationStrategyVisible =
```



One electrical cell model simulates all the cells contained in the dotted orange box. The cell model parameters are scaled up using the number of parallel assemblies connected in series, S, and the number of cells connected in parallel, P, as a multiplication factor:

- Cell-to-Module open-circuit voltage (V) — $\mathrm{V0}_{\mathrm{ParallelAssembly}} = \mathrm{V0}_{\mathrm{cell}} * S$
- Cell-to-ParallelAssembly resistance (Ohm) — $\mathrm{R0}_{\mathrm{ParallelAssembky}} = \mathrm{R0}_{\mathrm{cell}} * S / P$, $\mathrm{R1}_{\mathrm{ParallelAssembky}} = \mathrm{R1}_{\mathrm{cell}} * S / P$
- Cell-to-ParallelAssembly capacitance (F) — $\mathrm{C1}_{\mathrm{ParallelAssembly}} = \mathrm{C1}_{\mathrm{cell}} * P / S$
- Cell-to-ParallelAssembly capacity (Ah) — $\mathrm{AH}_{\mathrm{ParallelAssembly}} = \mathrm{AH}_{\mathrm{cell}} * P$
- Cell-to-ParallelAssembly thermal mass (J/K) — $\mathrm{thermal\_mass}_{\mathrm{ParallelAssembly}} = \mathrm{thermal\_mass}_{\mathrm{cell}} * S * P$

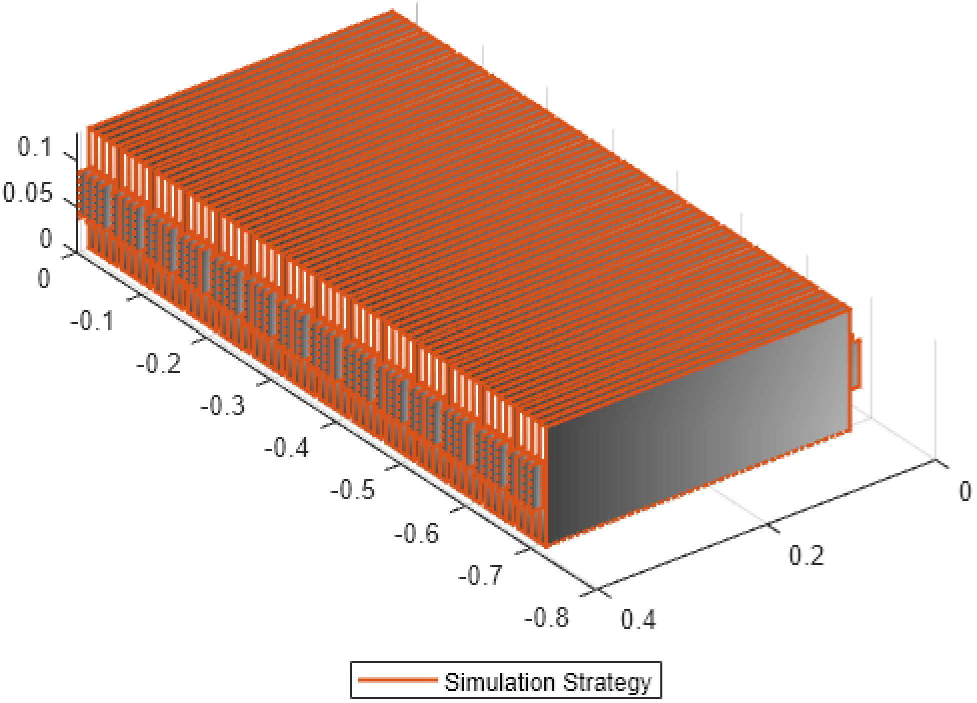
Detailed Module Resolution

Now change the model resolution of the previous `Module` object to `"Detailed"` and visualize it by using the `BatteryChart` object and by setting the **SimulationStrategyVisible** property to `"on"`.

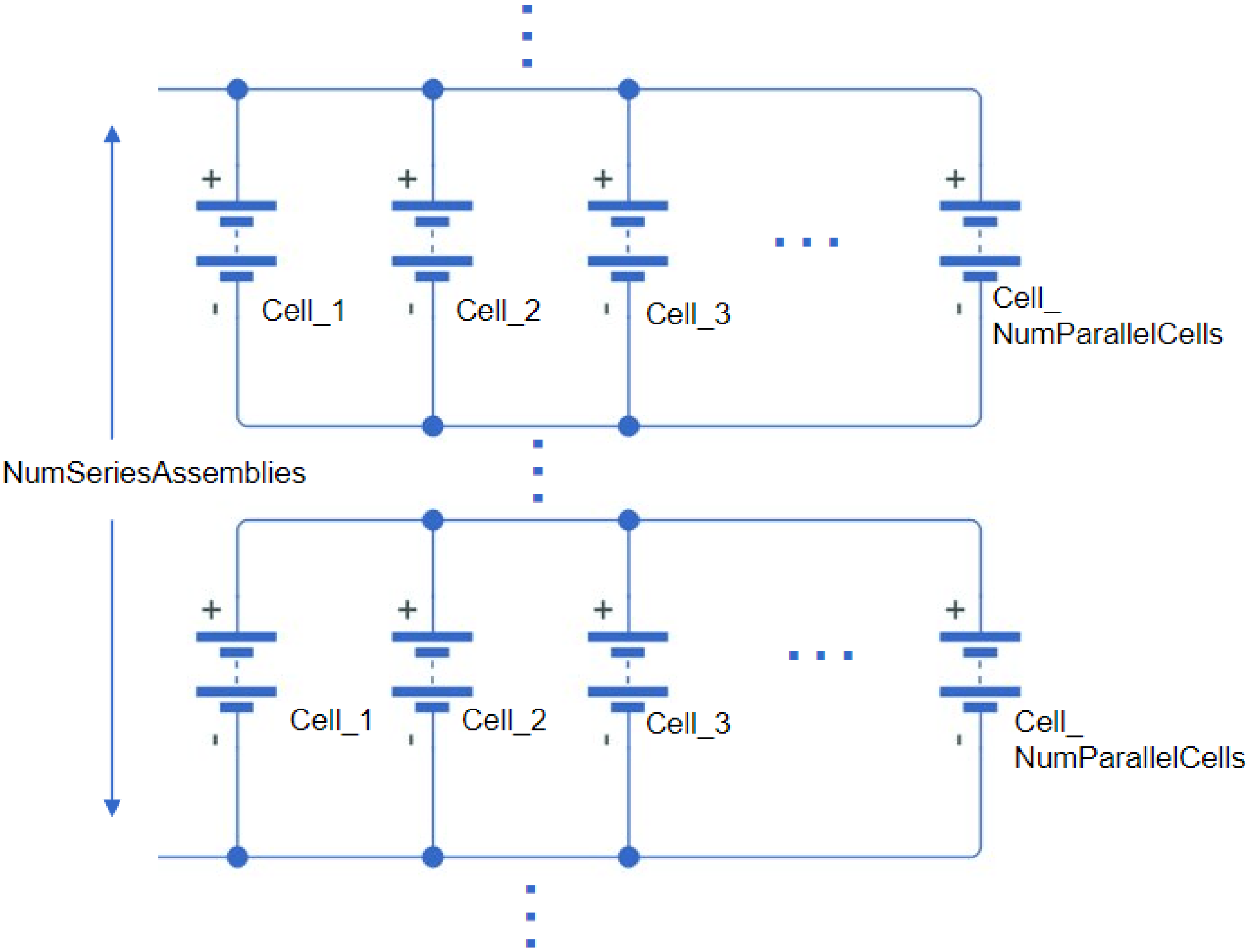
```
detailedModule = lumpedModule;
detailedModule.ParallelAssembly.ModelResolution = "Detailed";
detailedModule.ModelResolution = "Detailed";
```

For pouch modules, the **"Detailed"** model resolution is not recommended as many cells are present and it is important to keep the total number of models between 30 and 50.

```
f = uifigure("Color", "white");
moduleChartDetailed = BatteryChart(Parent = f, Battery = detailedModule, SimulationStrategyVisib
```



A number of cell model blocks equal to the value of the **NumParallelCells** property represents each cell component.



Grouped Module Resolution

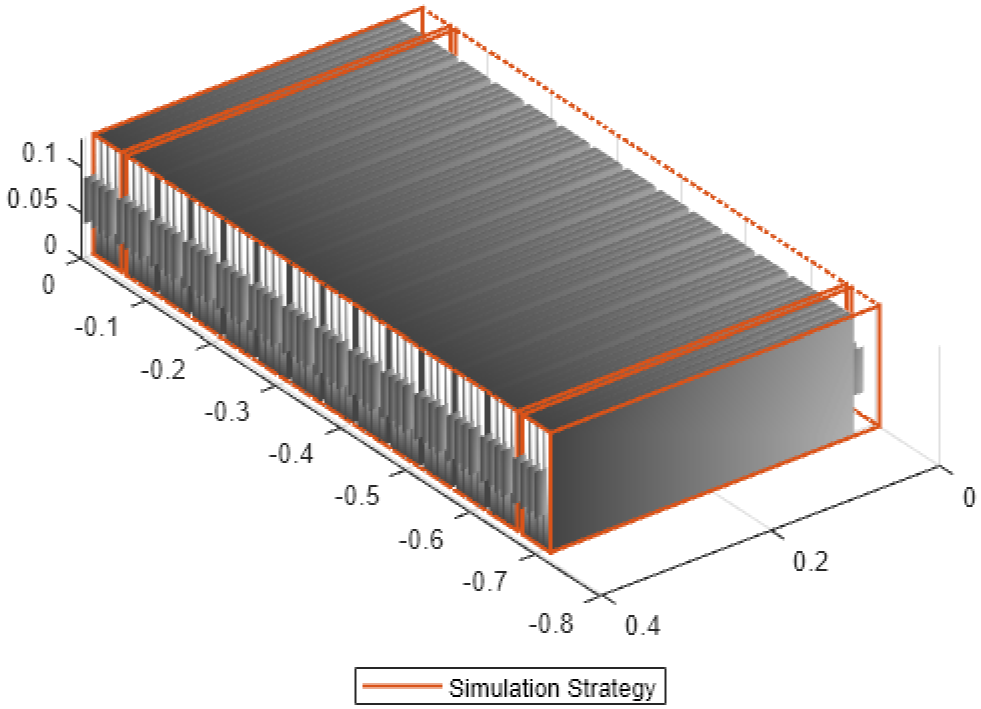
For battery modules, you can also set the **ModelResolution** property to `"Grouped"`. This simulation strategy helps increasing the model performance.

```
module.ModelResolution = "Grouped";
```

When you set the **ModelResolution** property of a module to "Grouped", you can define an additional simulation strategy by using the **SeriesGrouping** and **ParallelGrouping** properties:

- **SeriesGrouping** — Custom modeling strategy for the module along the series connections, specified as a strictly positive array of doubles. The length of the array of this property specifies the number of individual electrical models required. Each element value of this array specifies how many parallel assemblies are lumped within the specified electrical model. The sum of the elements in the array must be equal to value of the NumSeriesAssemblies property.

```
module.SeriesGrouping = [1,12,1];
f = uifigure("Color", "white");
moduleChartGrouped = BatteryChart(Parent = f, Battery = module, SimulationStrategyVisible = "on"
```



- **ParallelGrouping** — Custom modeling strategy for the module for every parallel assembly defined in the SeriesGrouping property, specified as a strictly positive array of doubles. The length of the array of this property must be equal to the length of the array of the SeriesGrouping property. Each element of this array specifies the number of individual electrical models for every element in the array of the SeriesGrouping property. The values of the elements of this array can be equal only to either 1 or the value of the NumParallelCells property. For example, if your module comprises four parallel assemblies (**NumSeriesAssemblies** = 4), 48 pouch cells for each parallel assembly (**NumParallelCells** = 48), and three individual electrical models where the first model comprises two of the original parallel assemblies (**SeriesGrouping** = [2 1 1]), then if you set this property to `[1 1 48]`, the module is discretized in 50 individual electrical models where each cell of the fourth parallel assembly has an electrical model.

Assign Model Resolution for ModuleAssembly Object

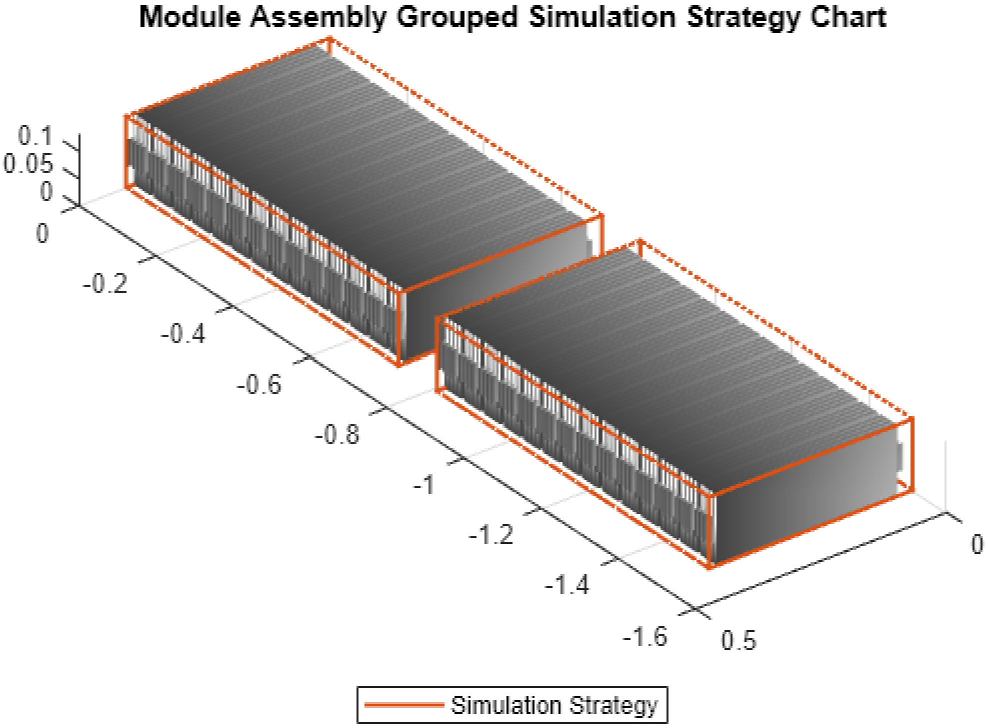
A `ModuleAssembly` object inherits the model resolution of its battery modules.

Create a `ModuleAssembly` object by using the lumpedModule `Module` object that you created previously.

```
module.ModelResolution = "Lumped";
moduleAssemblyLumped = ModuleAssembly(...
    Module = repmat(module,1,2), ...
    InterModuleGap = simscape.Value(0.1, "m"));
```

Then visualize the ModuleAssembly object and check the model resolution by setting the **SimulationStrategyVisible** property to "on".

```
f = uifigure("Color", "white");
moduleAssemblyChart = BatteryChart(Parent = f, Battery = moduleAssemblyLumped , SimulationStrateg
title(moduleAssemblyChart, "Module Assembly Grouped Simulation Strategy Chart"  )
```



The **ModelResolution** property of the ModuleAssembly object you just created is automatically set to "Lumped" because the **ModelResolution** properties of its modules are set to "Lumped".

Assign Model Resolution for Pack Object

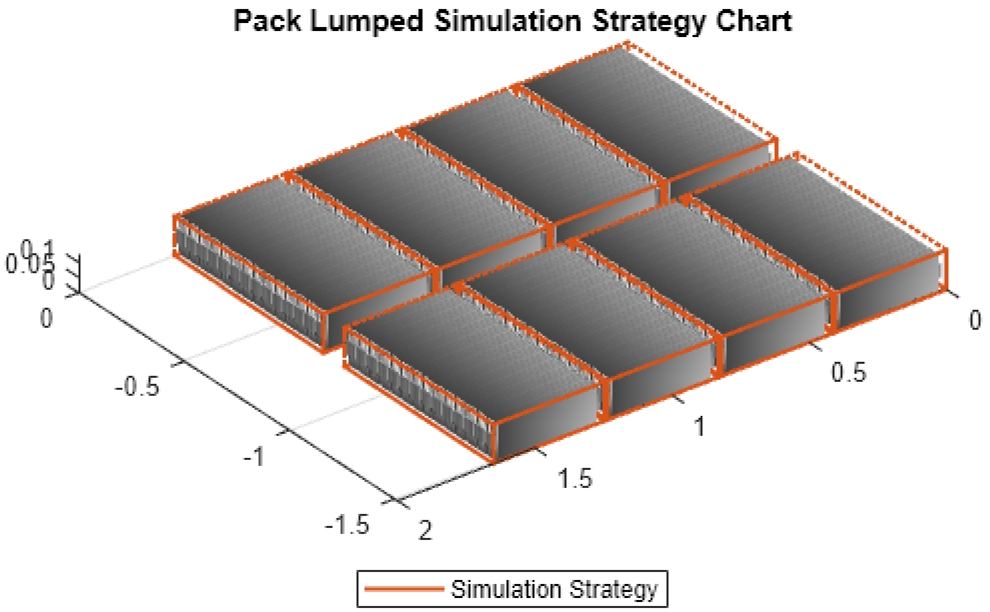
A Pack object inherits the model resolution of its battery module assemblies.

Create a Pack object by using the moduleAssemblyLumped ModuleAssembly object that you created in the previous step.

```
packLumped = Pack(...
    ModuleAssembly = repmat(moduleAssemblyLumped,1,4), ...
    StackingAxis = "X",...
    InterModuleAssemblyGap = simscape.Value(0.01, "m"));
```

Then visualize the Pack object and check the model resolution by setting the **SimulationStrategyVisible** property to "on".

```
f = uifigure("Color", "white");
packLumpedChart = BatteryChart(Parent = f, Battery = packLumped , SimulationStrategyVisible = "o
title(packLumpedChart, "Pack Lumped Simulation Strategy Chart")
```



The **ModelResolution** property of the `Pack` object you just created is automatically set to `"Lumped"` because the **ModelResolution** properties of its module assemblies are set to `"Lumped"`.

Build Simscape Model for the Battery Objects

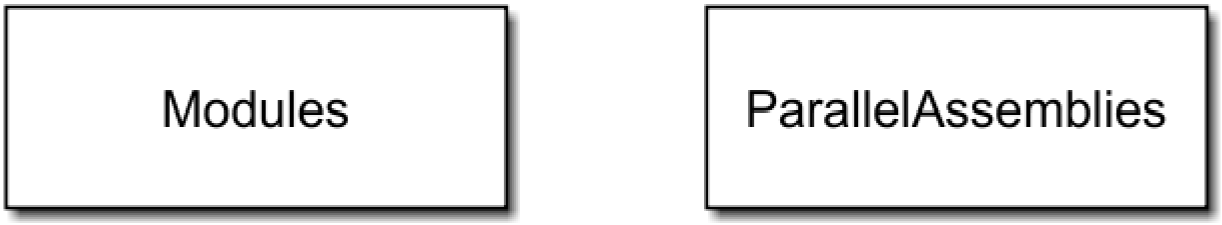
After you have created your battery objects, you need to convert them into Simscape models to be able to use them in block diagrams. You can then use these models as reference for your system integration and requirement evaluation, cooling system design, control strategy development, hardware-in-the-loop, and much more.

To create a library that contains the Simscape Battery model of all the batteries object you created in this example, use the `buildBattery` function.

```
buildBattery(packLumped,"LibraryName","pouchPackExample");
```

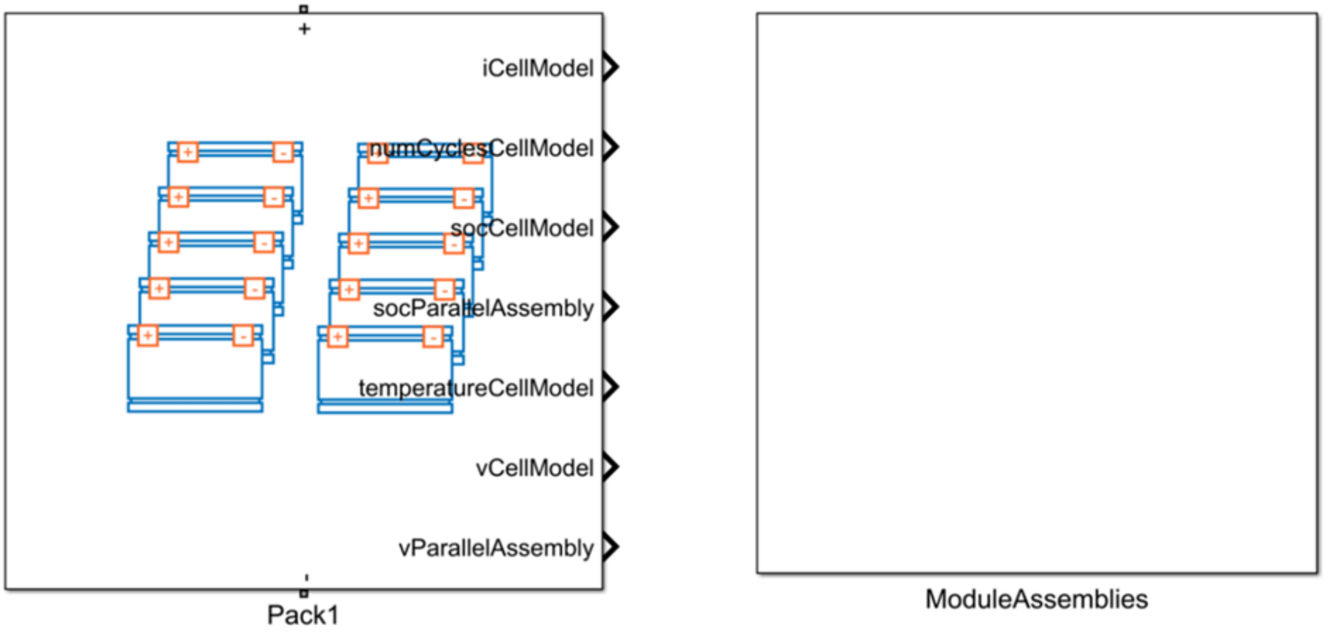
```
Generating Simulink library 'pouchPackExample_lib' in the current directory 'C:\TEMP\Bdoc22b_205
```

This function creates the `pouchPackExample_lib` and `pouchPackExample` SLX library files in your working directory. The `pouchPackExample_lib` library contains the Modules and ParallelAssemblies sublibraries.



To access the Simscape models of your Module and ParallelAssembly objects, open the pouchPackExample_lib. SLX file, double-click the sublibrary, and drag the Simscape blocks in your model.

The pouchPackExample library contains the Simscape models of your ModuleAssembly and Pack objects.



The Simscape models of your ModuleAssembly and Pack objects are subsystems. You can look inside these subsystems by opening the packLibrary SLX file and double-click the subsystem.

Build Model of Battery Module with Thermal Effects

This example shows how to create and build a Simscape™ system model of a battery module with thermal effects in Simscape™ Battery™. To create the system model of a battery module, you must first create the Cell and ParallelAssembly objects that comprise the battery module, and then use the buildBattery function. The buildBattery function allows you to automatically generate Simscape models for these Simscape Battery objects:

- ParallelAssembly
- Module
- ModuleAssembly
- Pack

This function creates a library in your working folder that contains a system model block of a battery module that you can use as reference in your simulations. The run-time parameters for these models, such as the battery cell impedance or the battery open-circuit voltage, are defined after the model creation and are therefore not covered by the Battery Pack Builder classes. To define the run-time parameters, you can either specify them in the block mask of the generated Simscape models or use the **MaskParameters** argument of the buildBattery function.

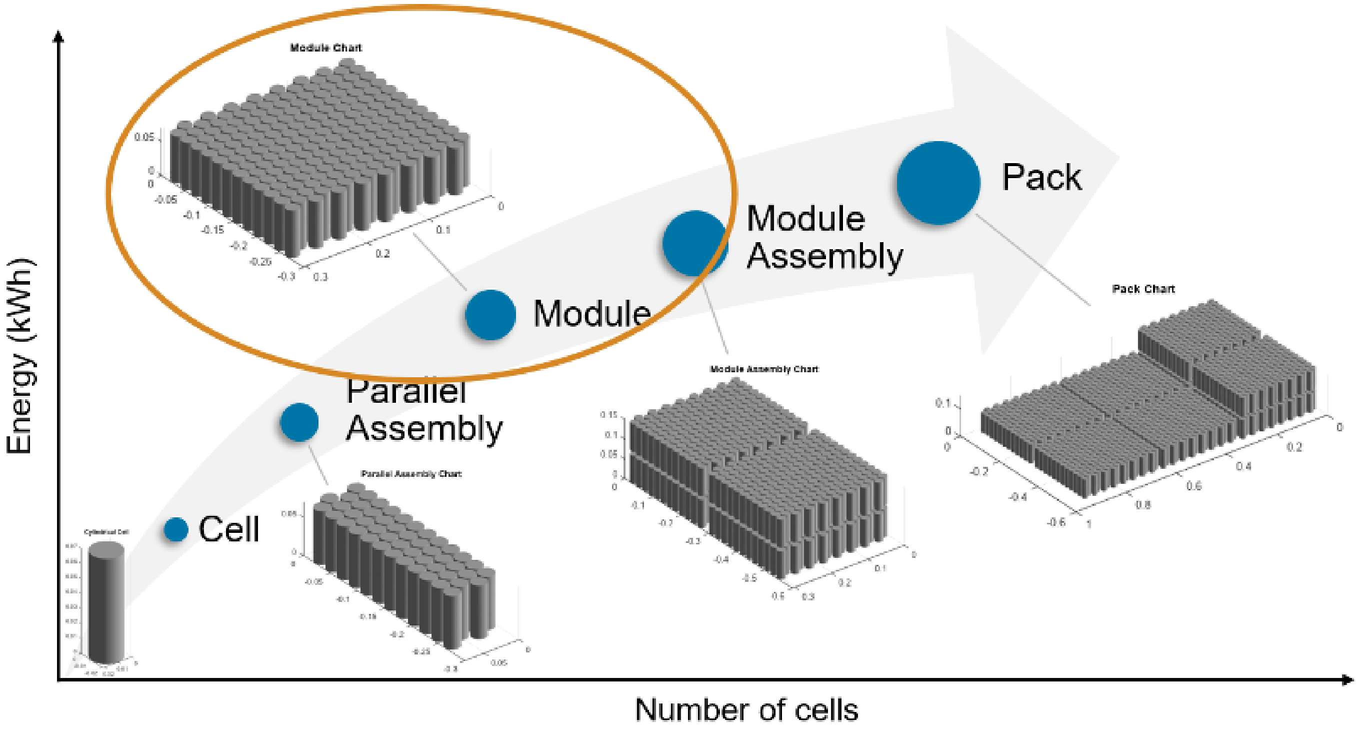
To use the functions and objects in Simscape Battery, first import the required Simscape Battery package:

```
import simscape.battery.builder.*
```

Create Battery Module Object in MATLAB

To create a battery module object, you must first design and create the foundational elements of the battery module.

This figure shows the overall process to create a battery module object in a bottom-up approach:



A battery module comprises multiple parallel assemblies. These parallel assemblies, in turn, comprise a number of battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement.

Create Cell Object

To create the battery `Module` object, first create a Cell object of pouch format.

```
pouchGeometry = PouchGeometry(Height = simscape.Value(0.1,"m"),...
    Length = simscape.Value(0.3,"m"), TabLocation = "Opposed" );
```

The `PouchGeometry` object allows you to define the cylindrical geometrical arrangement of the battery cell. You can specify the height, radius, and location of tabs of the cell by setting the **Height**, **Radius**, and **TabLocation** properties of the `PouchGeometry` object. For more information on the possible geometrical arrangements of a battery cell, see the `CylindricalGeometry` and `PrismaticGeometry` documentation pages.

Now use this `PouchGeometry` object to create a pouch battery cell.

```
batteryCell = Cell(Geometry = pouchGeometry)
```

```
batteryCell = 
  Cell with properties:

            Geometry: [1x1 simscape.battery.builder.PouchGeometry]
    CellModelOptions: [1x1 simscape.battery.builder.CellModelBlock]
                Mass: [1x1 simscape.Value]

Show all properties
```

For more information, see the `Cell` documentation page.

The Cell object allows you to simulate the thermal effects of the battery cell by using a simple 1-D model. To simulate the thermal effects of the battery cell, in the **BlockParameters** property of the **CellModelOptions** property of the `Cell` object, set the **thermal_port** property to `"model"` and the **T_dependence** property to `"yes"`.

```
batteryCell.CellModelOptions.BlockParameters.thermal_port = "model";
batteryCell.CellModelOptions.BlockParameters.T_dependence = "yes";
```

You can define the thermal boundary conditions for battery parallel assemblies and modules only if you have previously defined a thermal model at the cell level.

Create ParallelAssembly Object

A battery parallel assembly comprise multiple battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement. In this example, you create a parallel assembly of three pouch cells.

To create the `ParallelAssembly` object, use the `Cell` object you created before and specify the **NumParallelCells** property.

```
batteryParallelAssembly = ParallelAssembly(Cell = batteryCell,...
    NumParallelCells = 3, ...
     ModelResolution = "Detailed");
```

For more information, see the `ParallelAssembly` documentation page.

Create Module Object

You now have all the foundational elements to create your battery module. A battery module comprises multiple parallel assemblies connected in series. In this example, you create a battery module of 14 parallel assemblies with an intergap between each assembly of 0.005 meters. You also define the model resolution of the module.

To create the `Module` object, use the `ParallelAssembly` object you created before and specify the **NumSeriesAssemblies**, **InterParallelAssemblyGap**, and **ModelResolution** properties.

```
detailedBatteryModule = Module(ParallelAssembly = batteryParallelAssembly,...
    NumSeriesAssemblies = 14, ...
    InterParallelAssemblyGap = simscape.Value(0.005,"m"), ...
    ModelResolution = "Detailed");
```

For more information, see the `Module` documentation page.

Define Thermal Boundary Conditions

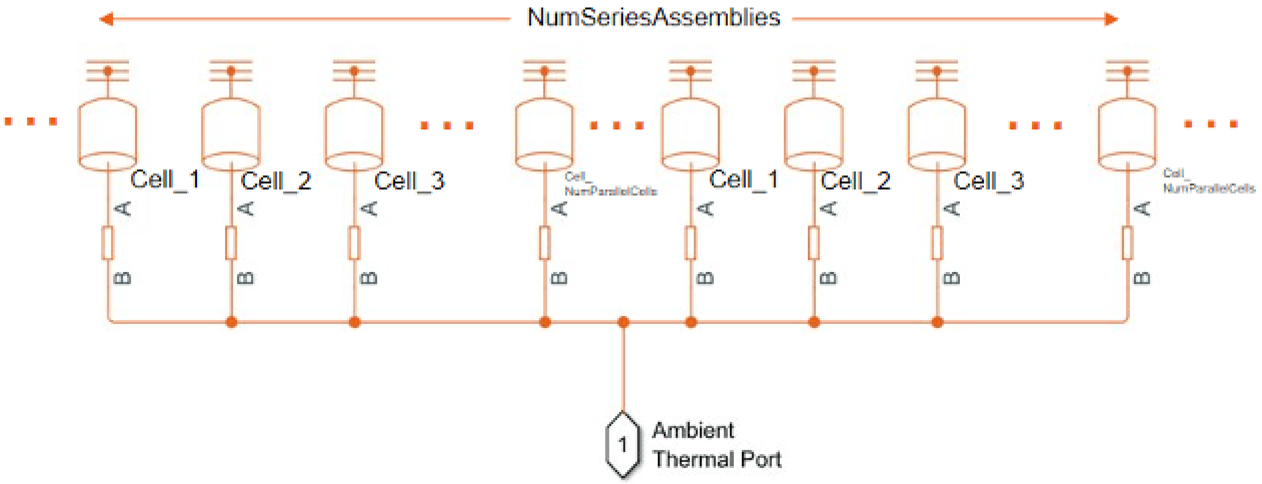
For your `Module` object, you can define the thermal paths to the ambient, the coolant, and the location of the cooling plate by using the **AmbientThermalPath**, **CoolantThermalPath**, and **CoolingPlate** properties.

Define Ambient Thermal Path

To define a thermal path to ambient, set the **AmbientThermalPath** property to `"CellBasedThermalResistance"`.

```
detailedBatteryModule.AmbientThermalPath = "CellBasedThermalResistance";
```

This command adds and connects one Thermal Resistor block to every thermal port in a cell model. The other thermal ports from all resistors connect to a single thermal node. You can then connect this thermal node with a constant temperature source or other blocks in the Simscape libraries.

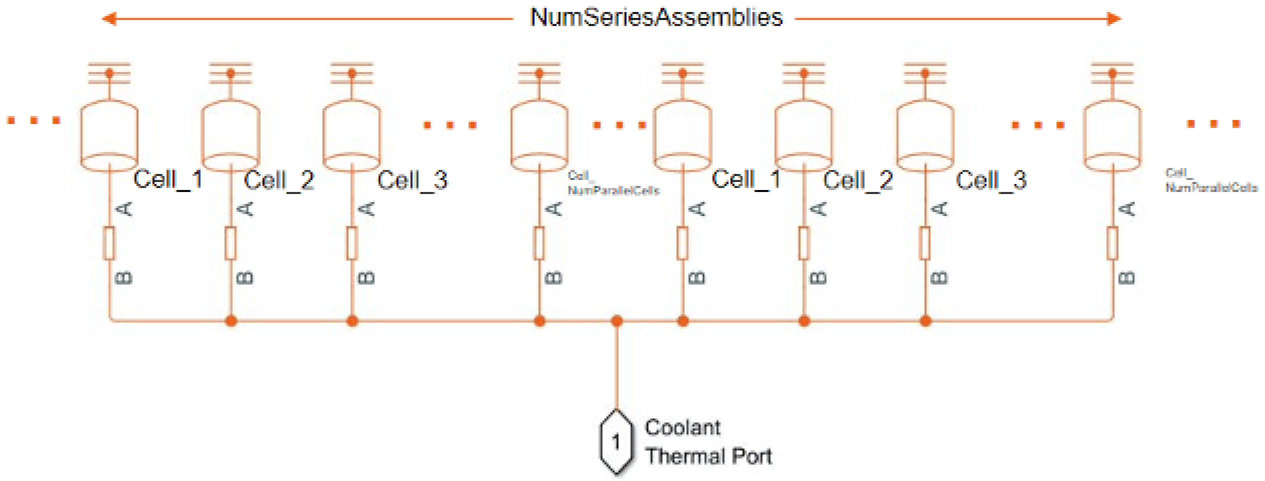


Define Coolant Thermal Path

To define a thermal path to ambient, set the **CoolantThermalPath** property to `"CellBasedThermalResistance"`.

```
detailedBatteryModule.CoolantThermalPath = "CellBasedThermalResistance";
```

This command adds and connects one Thermal Resistor block to every thermal port in a cell model. The other thermal ports from all resistors connect to a single thermal node. You can then connect this thermal node with a constant temperature source or other blocks in the Simscape libraries.



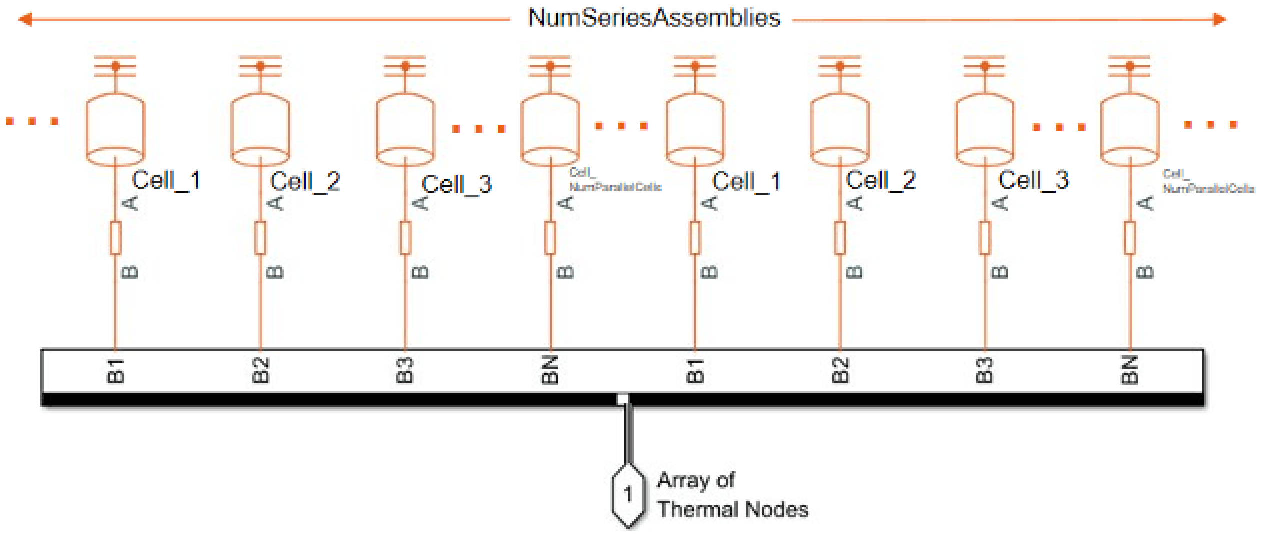
You can use the Thermal Resistor block to capture the conduction resistance relative to the cell, the thermal interface materials, and other materials along the path to the coolant. If you define a cooling system such as a cooling plate for the battery module, the other thermal port of the Thermal Resistor block is connected to an array of thermal nodes connector.

Define Cooling Plate Location

To define the location of the cooling plate on your battery module, set the **CoolingPlate** property to either `"Top"` or `"Bottom"`.

```
detailedBatteryModule.CoolingPlate = "Bottom";
```

This command connects every thermal node of each cell model in your battery module to a corresponding element inside an array of thermal nodes connector. If a **CoolantThermalPath** has been enabled, then a thermal resistance will be added between each battery model and its corresponding element inside the arrary of thermal nodes.



The array of thermal nodes is exposed at the **Module** level as a single connector but is multi-dimensional. You can connect an array of thermal nodes only to another array of thermal nodes of the same size. You can add a Cooling Plate block from the Simscape Battery library as heat sink.

To facilitate multi-dimensional thermal domain connections, you can use the **ThermalNodes** property of your `Module` object as input to the Cooling Plate block. You can view the number of thermal nodes, dimensions, and locations of the thermal nodes of the underlying cell models by accesing the **ThermalNodes** property.

```
disp(detailedBatteryModule.ThermalNodes);
```

```
        Bottom: [1x1 struct]
     Locations: [42x2 double]
    Dimensions: [42x2 double]
      NumNodes: 42
```

Visualize Battery Module and Check Model Resolution

To obtain the number of Simscape Battery Battery(Table-based) blocks used for the pack simulation, use the **NumModels** property of your `Module` object.

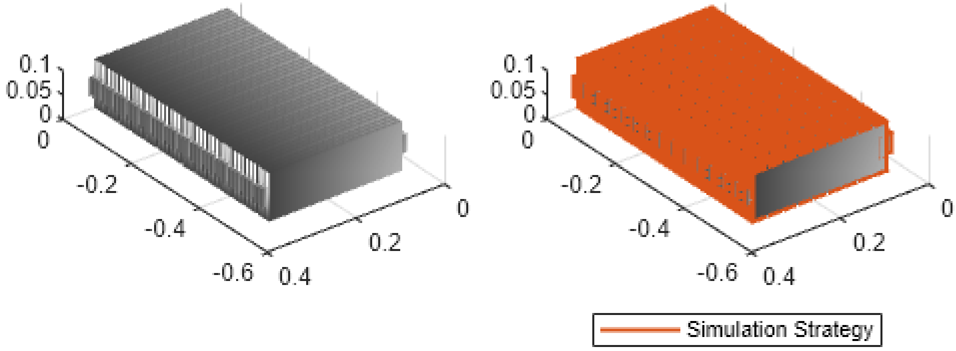
```
disp(detailedBatteryModule.NumModels);
```

```
    42
```

To visualize the battery module before you build the system model and to view its model resolution, use the `BatteryChart` object. Create the figure where you want to visualize your battery module.

Then use the `BatteryChart` object to visualize the battery module. To view the model resolution of the module, set the **SimulationStrategyVisible** property of the `BatteryChart` object to `"On"`.

```
f = uifigure(Color="w");
tl = tiledlayout(1,2,"Parent",f,"TileSpacing","Compact");
nexttile(tl)
batteryModuleChart1 = BatteryChart(Parent = tl, Battery = detailedBatteryModule);
nexttile(tl)
batteryModuleChart2 = BatteryChart(Parent = tl, Battery = detailedBatteryModule, SimulationStrate
```



For more information, see the `BatteryChart` documentation page.

Build Simscape Model for the Battery Module Object

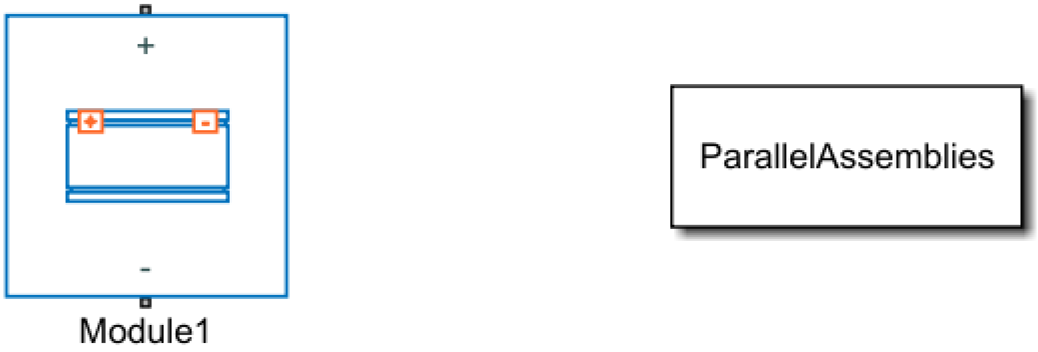
After you have created your battery objects, you need to convert them into Simscape models to be able to use them in block diagrams. You can then use these models as reference for your system integration and requirement evaluation, cooling system design, control strategy development, hardware-in-the-loop, and much more.

To create a library that contains the Simscape Battery model of the Module object you created in this example, use the buildBattery function.

```
buildBattery(detailedBatteryModule,"LibraryName","moduleBTMSExample")
```

```
Generating Simulink library 'moduleBTMSExample_lib' in the current directory 'C:\TEMP\Bdoc22b_205
```

This function creates a library named moduleBTMSExample_lib in your working directory. This library contains the Simscape models of your Module and ParallelAssembly objects.



To build a more detailed model of a battery pack, see the “Build Detailed Model of Battery Pack From Pouch Cells” on page 4-81 example.

To see an application of a battery thermal effects model with a coolant thermal path, see the “Protect Battery During Charge and Discharge for Electric Vehicle” on page 4-10 example.

Build Model of Battery Pack with Cell Aging

This example shows how to create and build a Simscape™ system model of a battery pack that includes cell aging in Simscape™ Battery™. Predicting the lifetime of battery cells under a specific application is fundamental to assess warranty risk, develop second-life applications, and perform virtiual design verification.

To create the system model of a battery pack, you must first create the `Cell`, `ParallelAssembly`, `Module`, and `ModuleAssembly` objects that comprise the battery pack, and then use the `buildBattery` function. The `buildBattery` function allows you to automatically generate Simscape models for these Simscape Battery objects:

- `ParallelAssembly`
- `Module`
- `ModuleAssembly`
- `Pack`

This function creates a library in your working folder that contains a system model block of a battery pack that you can use as reference in your simulations. The run-time parameters for these models, such as the battery cell impedance or the battery open-circuit voltage, are defined after the model creation and are therefore not covered by the Battery Pack Builder classes. To define the run-time parameters, you can either specify them in the block mask of the generated Simscape models or use the **MaskParameters** argument of the `buildBattery` function.

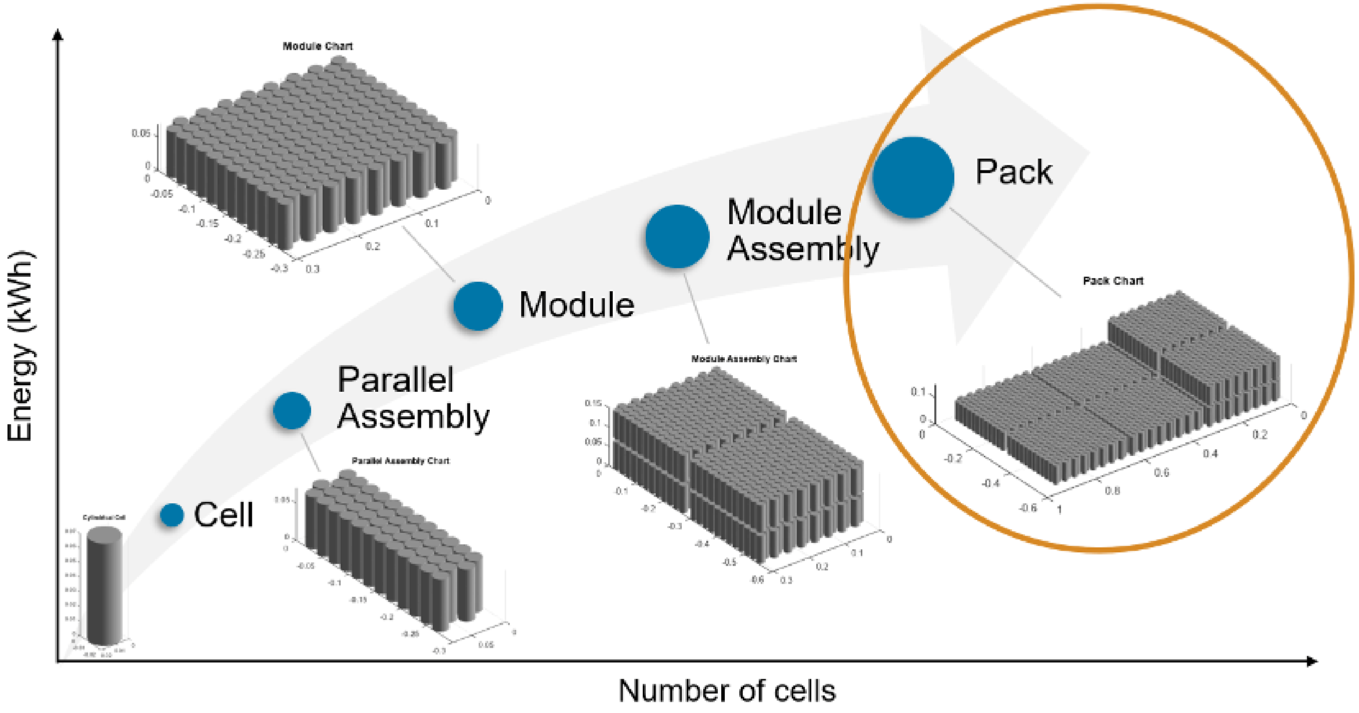
To use the functions and objects in Simscape Battery, first import the required Simscape Battery package:

```
import simscape.battery.builder.*
```

Create Battery Pack Object in MATLAB

To create a battery pack, you must first design and create the foundational elements of the battery pack.

This figure shows the overall process to create a battery pack object in a bottom-up approach:



A battery pack comprises multiple module assemblies. These module assemblies, in turn, comprise a number of battery modules connected electrically in series or in parallel. The battery modules are made of multiple parallel assemblies which, in turn, comprise a number of battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement.

Create Cell Object and Specify Aging Effects

To create the battery `Module` object, first create a `Cell` object of pouch format.

```
pouchGeometry = PouchGeometry()
```

```
pouchGeometry = 
  PouchGeometry with properties:

         Length: [1x1 simscape.Value]
      Thickness: [1x1 simscape.Value]
    TabLocation: "Standard"
       TabWidth: [1x1 simscape.Value]
      TabHeight: [1x1 simscape.Value]
         Height: [1x1 simscape.Value]
```

The `PouchGeometry` object allows you to define the pouch geometrical arrangement of the battery cell. For more information on the possible geometrical arrangements of a battery cell, see the `CylindricalGeometry` and `PrismaticGeometry` documentation pages.

Now use this `PouchGeometry` object to create a pouch battery cell.

```
batteryCell = Cell(Geometry = pouchGeometry)
```

```
batteryCell = 
  Cell with properties:
```

```
           Geometry: [1x1 simscape.battery.builder.PouchGeometry]
   CellModelOptions: [1x1 simscape.battery.builder.CellModelBlock]
               Mass: [1x1 simscape.Value]

Show all properties
```

For more information, see the `Cell` documentation page.

The Cell object allows you to simulate the aging effects of the battery cell by specifying these properties:

- prm_age_capacity — Capacity calendar aging. This property allows you to decide whether to model the calendar aging effects on the capacity of a battery cell.
- prm_age_resistance — Internal resistance calendar aging. This property allows you to decide whether to model the calendar aging effects on the internal resistance of a battery cell.
- prm_age_modeling — Modeling option. This property allows you to specify how to mathematically model the aging effects on the capacity and internal resistance of a battery cell.

To simulate the cycling aging effects of the battery cell, in the **BlockParameters** property of the **CellModelOptions** property of the `Cell` object, set the **prm_fade** property to `"equations"`.

```
batteryCell.CellModelOptions.BlockParameters.prm_fade = "equations";
```

The Cell object also allows you to simulate the thermal effects of the battery cell by using a simple 1-D model. To simulate the thermal effects of the battery cell, in the **BlockParameters** property of the **CellModelOptions** property of the `Cell` object, set the **thermal_port** parameter to `"model"`.

```
batteryCell.CellModelOptions.BlockParameters.thermal_port = "model";
```

Create ParallelAssembly Object

A battery parallel assembly comprise multiple battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement. In this example, you create a parallel assembly of three pouch cells.

To create the ParallelAssembly object, use the `Cell` object you created before and specify the **NumParallelCells** property according to your design.

```
batteryParallelAssembly = ParallelAssembly(Cell = batteryCell,...
    NumParallelCells = 3, StackingAxis = "X");
```

For more information, see the `ParallelAssembly` documentation page.

Create Module Object

A battery module comprises multiple parallel assemblies connected in series. In this example, you create a battery module of 4 parallel assemblies stacked along the X axis, with an intergap between each assembly of 0.005 meters.

To create the `Module` object, use the `ParallelAssembly` object you created in the previous step and specify the **NumSeriesAssemblies, InterParallelAssemblyGap**, and **StackingAxis** properties.

```
batteryModule = Module(ParallelAssembly = batteryParallelAssembly,...
    NumSeriesAssemblies = 4, ...
    InterParallelAssemblyGap = simscape.Value(0.005,"m"), ...
    StackingAxis = "X");
```

For more information, see the `Module` documentation page.

Create ModuleAssembly Object

A battery module assembly comprises multiple battery modules connected in series or in parallel. In this example, you create a battery module assembly of five identical modules with an intergap between each module equal to 0.1 meters. By default, the `ModuleAssembly` object electrically connects the modules in series.

To create the `ModuleAssembly` object, use the `Module` object you created in the previous step and specify the **InterModuleGap** and **StackingAxis** properties.

```
batteryModuleAssembly = ModuleAssembly(Module = repmat(batteryModule,1,5),...
    InterModuleGap = simscape.Value(0.1,"m"), ...
    StackingAxis = "Y");
```

For more information, see the `ModuleAssembly` documentation page.

Create Pack Object

You now have all the foundational elements to create your battery pack. A battery pack comprises multiple module assemblies connected in series or in parallel. In this example, you create a battery pack of 5 identical module assemblies with an intergap between each module assembly of 0.01 meters and a coolant thermal path.

To create the `Pack` object, use the `ModuleAssembly` object you created in the previous step and specify the **InterModuleAssemblyGap** and **CoolantThermalPath** properties.

```
batteryPack = Pack(ModuleAssembly = repmat(batteryModuleAssembly,1,5),...
    InterModuleAssemblyGap = simscape.Value(0.01,"m"),...
    CoolantThermalPath = "CellBasedThermalResistance");
```

For more information, see the `Pack` documentation page.

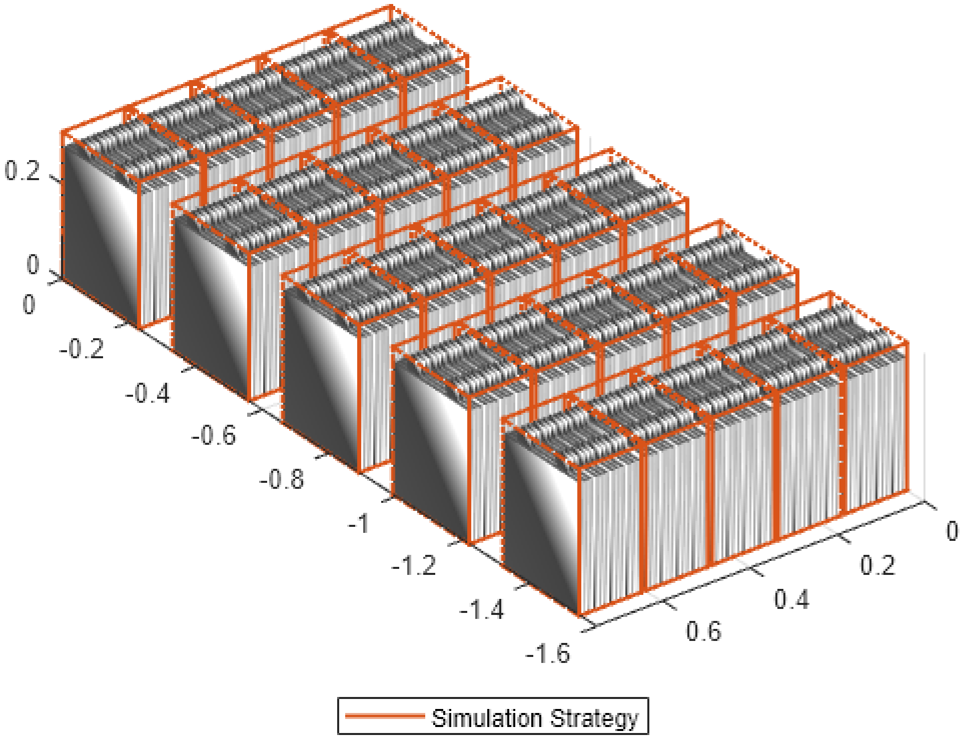
Visualize Battery Pack and Check Model Resolution

To visualize the battery pack before you build the system model and to view its model resolution, use the `BatteryChart` object. Create the figure where you want to visualize your battery pack.

```
f = uifigure(Color="w");
```

Then use the `BatteryChart` object to visualize the battery pack. To view the model resolution of the module, set the **SimulationStrategyVisible** property of the `BatteryChart` object to `"On"`.

```
batteryPackChart = BatteryChart(Parent = f, Battery = batteryPack, ...
    SimulationStrategyVisible = "on");
```



To add default axis labels to the battery plot, use the `setDefaultLabels` method of the `BatteryChart` object.

For more information about the BatteryChart object, see the `BatteryChart` documentation page.

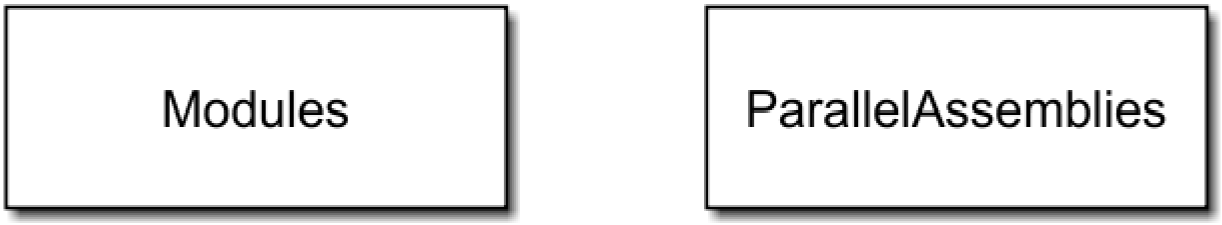
Build Simscape Model for the Battery Pack Object

After you have created your battery objects, you need to convert them into Simscape models to be able to use them in block diagrams. You can then use these models as reference for your system integration and requirement evaluation, cooling system design, control strategy development, hardware-in-the-loop, and much more.

```
buildBattery(batteryPack,"LibraryName","packAgingExample")
```

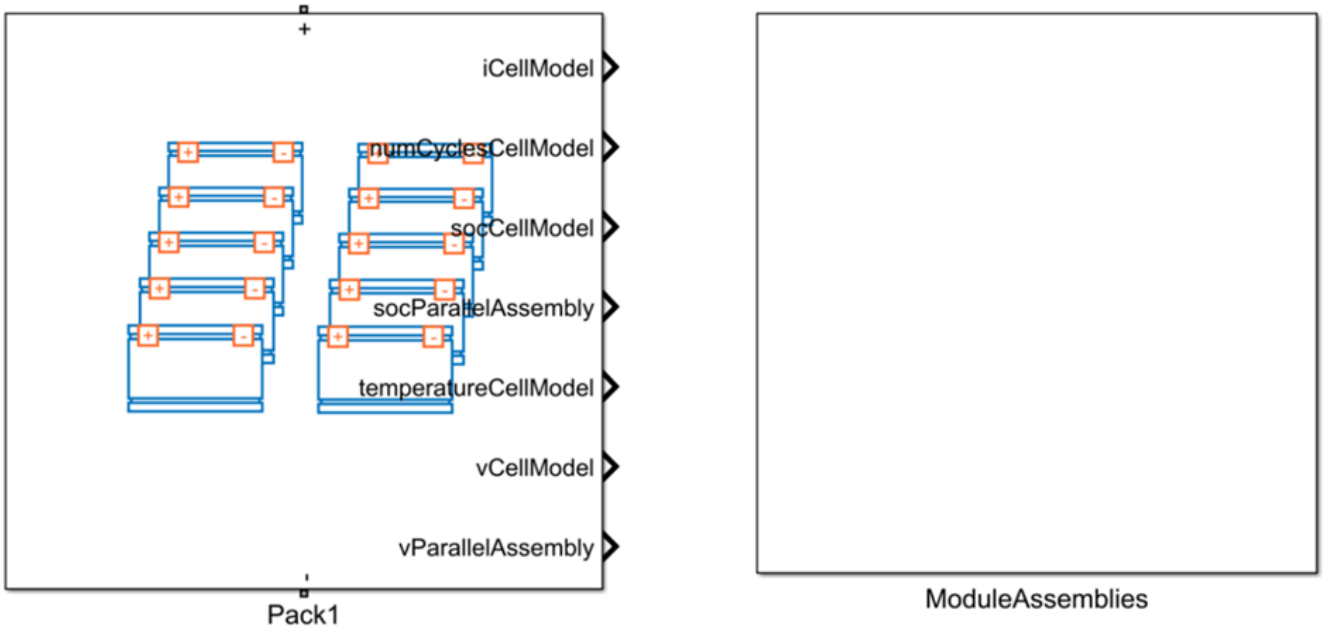
```
Generating Simulink library 'packAgingExample_lib' in the current directory 'C:\TEMP\Bdoc22b_2054
```

This function creates the `packAgingExample_lib` and `packAgingExample` SLX library files in your working directory. The `packAgingExample_lib` library contains the Modules and ParallelAssemblies sublibraries.



To access the Simscape models of your `Module` and `ParallelAssembly` objects, open the `packAgingExample_lib`. SLX file, double-click the sublibrary, and drag the Simscape blocks in your model.

The `packAgingExample` library contains the Simscape models of your `ModuleAssembly` and `Pack` objects.



The Simscape models of your `ModuleAssembly` and `Pack` objects are subsystems. You can look inside these subsystems by opening the `packLibrary` SLX file and double-click the subsystem.

To see how to evaluate a new and end-of-life (EOL) lithium-ion battery pack, see the “Thermal Analysis for New and Aged Battery Packs” on page 4-37 example.

Build Model of Battery Pack with Cell Balancing Circuit

This example shows how to create and build a Simscape™ system model of a battery pack with cell balancing circuits in Simscape™ Battery™. High voltage (> 60V) battery pack systems typically consist of multiple parallel assemblies or cells connected electrically in series. In these systems, the state-of-charge of individual parallel assemblies or cells often becomes unbalanced over time due to a variety of causes.

To create the system model of a battery pack, you must first create the `Cell`, `ParallelAssembly`, `Module`, and `ModuleAssembly` objects that comprise the battery pack, and then use the `buildBattery` function. The `buildBattery` function allows you to automatically generate Simscape models for these Simscape Battery objects:

- `ParallelAssembly`
- `Module`
- `ModuleAssembly`
- `Pack`

This function creates a library in your working folder that contains a system model block of a battery pack that you can use as reference in your simulations. The run-time parameters for these models, such as the battery cell impedance or the battery open-circuit voltage, are defined after the model creation and are therefore not covered by the Battery Pack Builder classes. To define the run-time parameters, you can either specify them in the block mask of the generated Simscape models or use the **MaskParameters** argument of the `buildBattery` function.

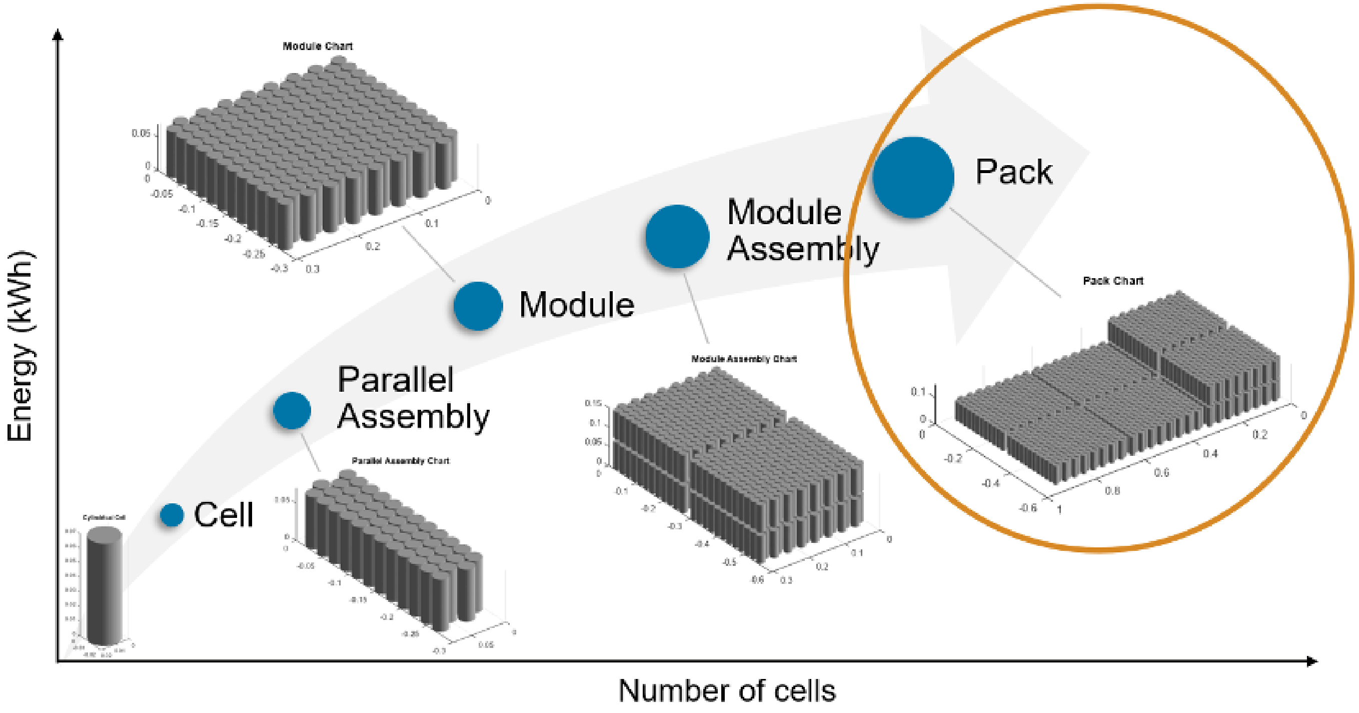
To use the functions and objects in Simscape Battery, first import the required Simscape Battery package:

```
import simscape.battery.builder.*
```

Create Battery Pack Object in MATLAB

To create a battery pack, you must first design and create the foundational elements of the battery pack.

This figure shows the overall process to create a battery pack object in a bottom-up approach:



A battery pack comprises multiple module assemblies. These module assemblies, in turn, comprise a number of battery modules connected electrically in series or in parallel. The battery modules are made of multiple parallel assemblies which, in turn, comprise a number of battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement.

Create Cell Object

To create the battery `Module` object, first create a Cell object of cylindrical format.

```
cylindricalGeometry = CylindricalGeometry(Height = simscape.Value(0.07,"m"),...
    Radius = simscape.Value(0.0105,"m"));
```

The `CylindricalGeometry` object allows you to define the cylindrical geometrical arrangement of the battery cell. You can specify the height and radius of the cell by setting the **Height** and **Radius** properties of the `CylindricalGeometry` object. For more information on the possible geometrical arrangements of a battery cell, see the `PouchGeometry` and `PrismaticGeometry` documentation pages.

Now use this `CylindricalGeometry` object to create a cylindrical battery cell.

```
batteryCell = Cell(Geometry = cylindricalGeometry)
```

```
batteryCell = 
  Cell with properties:

            Geometry: [1x1 simscape.battery.builder.CylindricalGeometry]
    CellModelOptions: [1x1 simscape.battery.builder.CellModelBlock]
                Mass: [1x1 simscape.Value]

Show all properties
```

For more information, see the `Cell` documentation page.

Create ParallelAssembly Object

A battery parallel assembly comprise multiple battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement. In this example, you create a parallel assembly of four cylindrical cells stacked in a square topology over four rows.

To create the ParallelAssembly object, use the `Cell` object you created before and specify the **NumParallelCells**, **Rows**, and **Topology** properties according to your design.

```
batteryParallelAssembly = ParallelAssembly(Cell = batteryCell,...
    NumParallelCells = 4, ...
    Rows = 4, ...
    Topology = "Square", ...
    ModelResolution = "Detailed");
```

For more information, see the `ParallelAssembly` documentation page.

Create Module Object

A battery module comprises multiple parallel assemblies connected in series. In this example, you create a battery module of 4 parallel assemblies with an intergap between each assembly of 0.005 meters. You also define the model resolution of the module and add an ambient thermal boundary condition.

To create the `Module` object, use the `ParallelAssembly` object you created in the previous step and specify the **NumSeriesAssemblies**, **InterParallelAssemblyGap**, and **ModelResolution** properties.

```
batteryModule = Module(ParallelAssembly = batteryParallelAssembly,...
    NumSeriesAssemblies = 4, ...
    InterParallelAssemblyGap = simscape.Value(0.005,"m"), ...
    ModelResolution = "Detailed");
```

For more information, see the `Module` documentation page.

Create ModuleAssembly Object

A battery module assembly comprises multiple battery modules connected in series or in parallel. In this example, you create a battery module assembly of two identical modules with an intergap between each module equal to 0.005 meters. By default, the `ModuleAssembly` object electrically connects the modules in series.

To create the `ModuleAssembly` object, use the `Module` object you created in the previous step and specify the **InterModuleGap** property.

```
batteryModuleAssembly= ModuleAssembly(Module = repmat(batteryModule,1,2),...
    InterModuleGap = simscape.Value(0.005,"m"));
```

For more information, see the `ModuleAssembly` documentation page.

Create Pack Object

You now have all the foundational elements to create your battery pack. A battery pack comprises multiple module assemblies connected in series or in parallel. In this example, you create a battery pack of 2 identical module assemblies with an intergap between each module assembly of 0.005 meters.

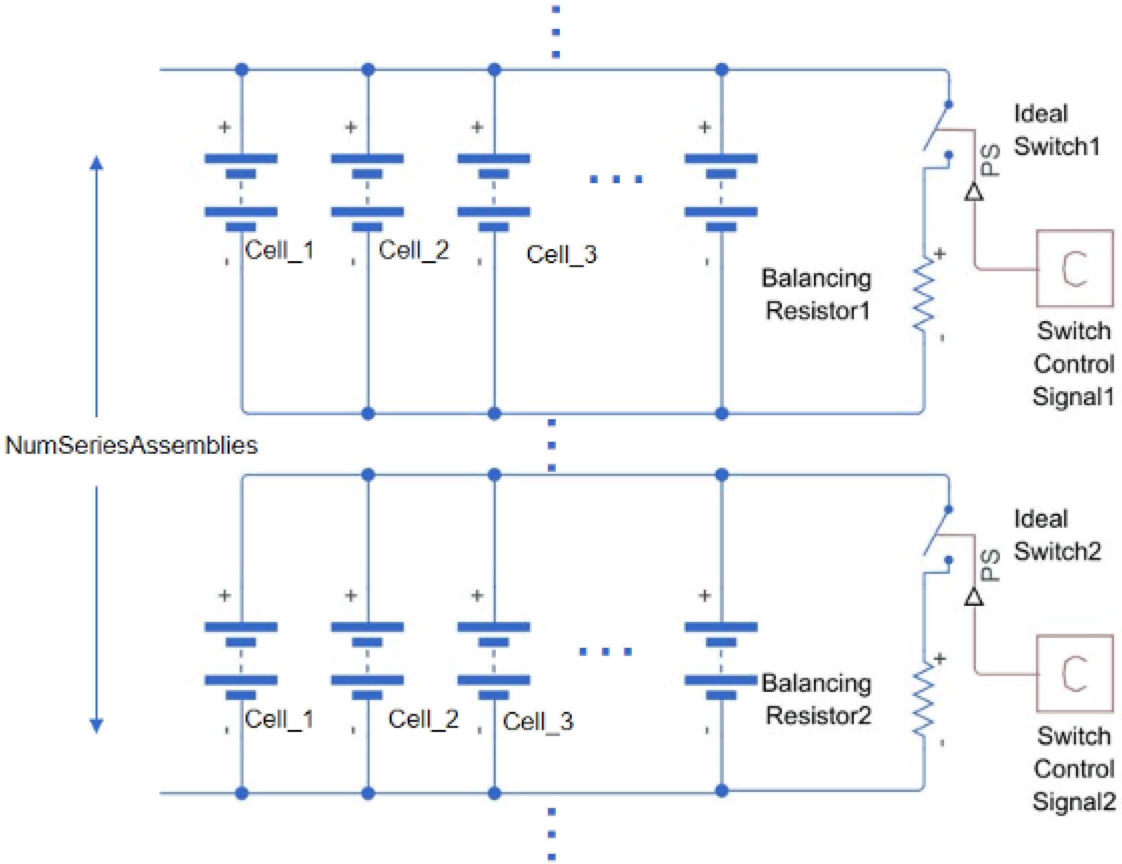
To create the Pack object, use the ModuleAssembly object you created in the previous step and specify the **InterModuleAssemblyGap** property.

```
batteryPack= Pack(ModuleAssembly = repmat(batteryModuleAssembly,1,2),...
    InterModuleAssemblyGap = simscape.Value(0.005,"m"));
```

For more information, see the Pack documentation page.

Define Cell Balancing Strategy

The Pack object allows you to define a cell balancing strategy. Specifying a balancing strategy adds an ideal passive balancing circuit to every parallel assembly inside the battery pack. The balancing circuit consists of a balancing resistor connected in series to a signal controlled switch.



To define the balancing strategy of your battery, set the **BalancingStrategy** property of the batteryPack object to "Passive".

```
batteryPack.BalancingStrategy = "Passive";
```

Visualize Battery Pack and Check Model Resolution

To obtain the number of Simscape Battery Battery(Table-based) blocks used for the pack simulation, use the **NumModels** property of your `Pack` object.

```
disp(batteryPack.NumModels);
```

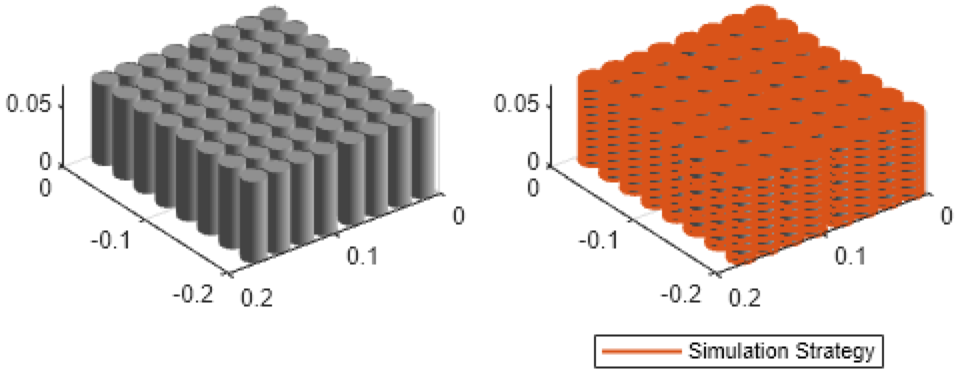
```
    64
```

To visualize the battery pack before you build the system model and to view its model resolution, use the `BatteryChart` object. Create the figure where you want to visualize your battery pack.

```
f = uifigure(Color="w");
```

Then use the `BatteryChart` object to visualize the battery module. To view the model resolution of the module, set the **SimulationStrategyVisible** property of the `BatteryChart` object to `"On"`.

```
tl = tiledlayout(1,2,"Parent",f,"TileSpacing","Compact");
nexttile(tl)
batteryPackChart = BatteryChart(Parent = tl, Battery = batteryPack);
nexttile(tl)
batteryPackChart = BatteryChart(Parent = tl, Battery = batteryPack, SimulationStrategyVisible =
```



For more information, see the `BatteryChart` documentation page.

Build Simscape Model for the Battery Module Object

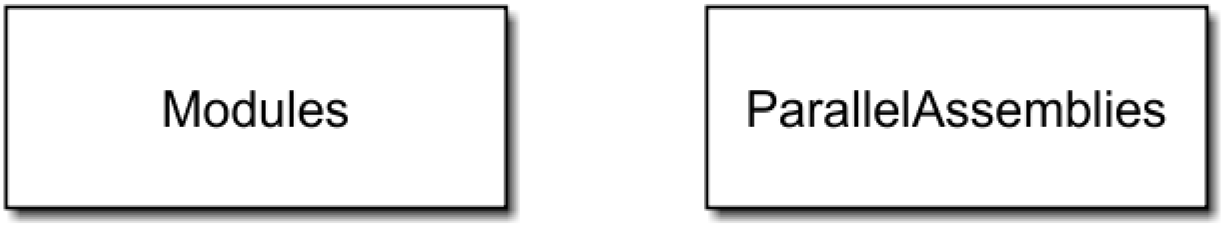
After you have created your battery objects, you need to convert them into Simscape models to be able to use them in block diagrams. You can then use these models as reference for your system integration and requirement evaluation, cooling system design, control strategy development, hardware-in-the-loop, and much more.

To create a library that contains the Simscape Battery model of the `Module` object you created in this example, use the `buildBattery` function.

```
buildBattery(batteryPack,"LibraryName","packBalancingExample");
```

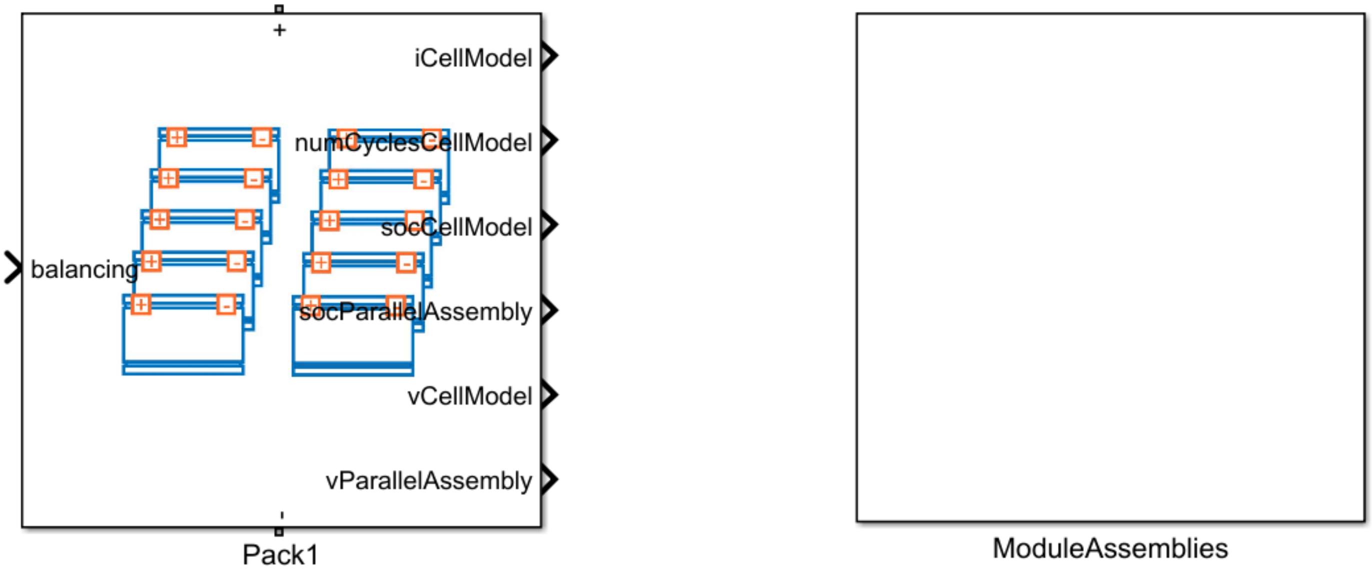
```
Generating Simulink library 'packBalancingExample_lib' in the current directory 'C:\TEMP\Bdoc22b_
```

This function creates the `packBalancingExample_lib` and `packBalancingExample` SLX library files in your working directory. The `packBalancingExample_lib` library contains the Modules and ParallelAssemblies sublibraries.



To access the Simscape models of your `Module` and `ParallelAssembly` objects, open the `packBalancingExample_lib`. SLX file, double-click the sublibrary, and drag the Simscape blocks in your model.

The `packBalancingExample` library contains the Simscape models of your `ModuleAssembly` and `Pack` objects.



The Simscape models of your `ModuleAssembly` and `Pack` objects are subsystems. You can look inside these subsystems by opening the `packLibrary` SLX file and double-click the subsystem.

To learn how to implement a passive cell balancing strategy for a lithium-ion batery pack, see the “Size Resistor for Battery Passive Cell Balancing” on page 4-41 example.

Build Model of Battery Pack for Grid Application

This example shows how to use Simscape™ Battery™ to create and build a Simscape™ system model of a battery pack from prismatic cells for grid applications. The battery pack is a 150 kWh prismatic battery for grid-level applications. To create the system model of a battery pack, you must first create the `Cell`, `ParallelAssembly`, `Module`, and `ModuleAssembly` objects that comprise the battery pack, and then use the `buildBattery` function. The `buildBattery` function allows you to automatically generate Simscape models for these Simscape Battery objects:

- `ParallelAssembly`
- `Module`
- `ModuleAssembly`
- `Pack`

This function creates a library in your working folder that contains a system model block of a battery pack that you can use as reference in your simulations. The run-time parameters for these models, such as the battery cell impedance or the battery open-circuit voltage, are defined after the model creation and are therefore not covered by the Battery Pack Builder classes. To define the run-time parameters, you can either specify them in the block mask of the generated Simscape models or use the **MaskParameters** argument of the `buildBattery` function.

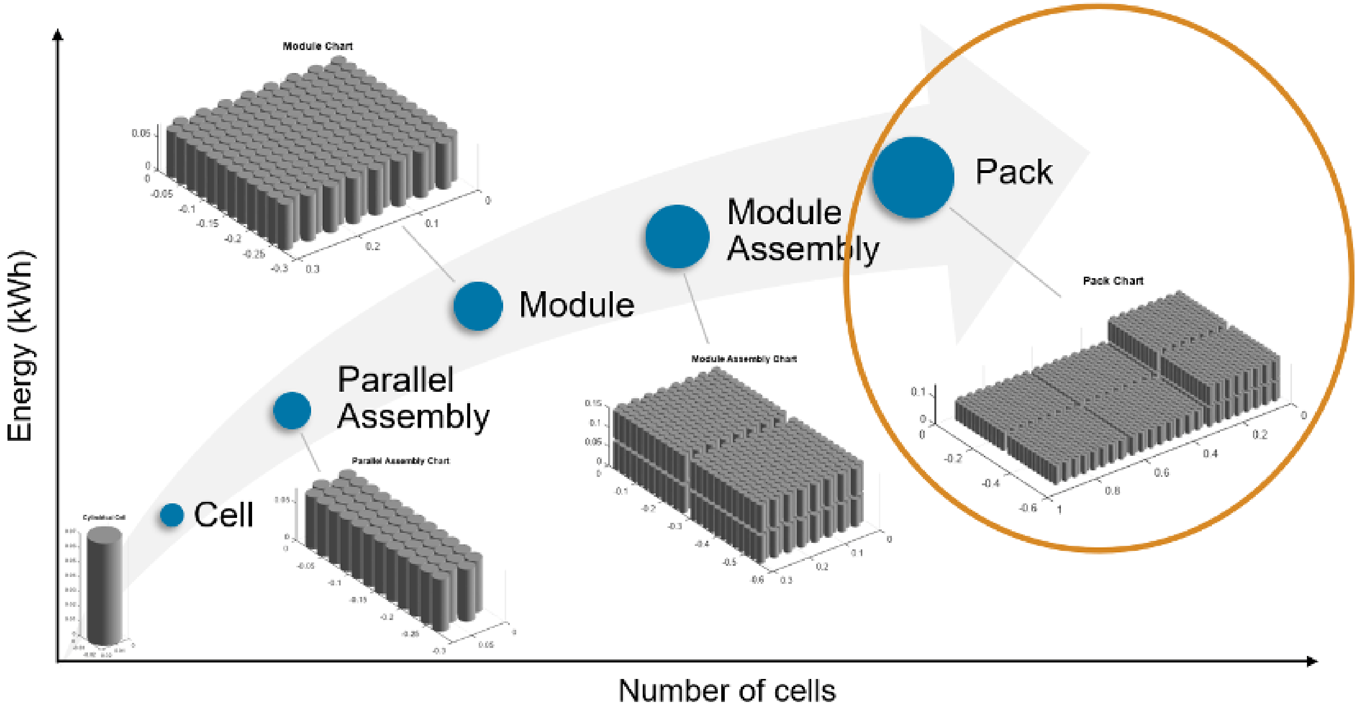
To use the functions and objects in Simscape Battery, first import the required Simscape Battery package:

```
import simscape.battery.builder.*
```

Create Battery Pack Object in MATLAB

Battery-based energy storage is a good option for integrating intermittent renewable energy sources into the grid. To create a battery pack, you must first design and create the foundational elements of the battery pack.

This figure shows the overall process to create a battery pack object in a bottom-up approach:



A battery pack comprises multiple module assemblies. These module assemblies, in turn, comprise a number of battery modules connected electrically in series or in parallel. The battery modules are made of multiple parallel assemblies which, in turn, comprise a number of battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement.

Create Cell Object

To create the battery `Module` object, first create a Cell object of prismatic format.

```
prismaticGeometry = PrismaticGeometry(Height = simscape.Value(0.2,"m"),...
    Length = simscape.Value(0.35,"m"), Thickness =  simscape.Value(0.07,"m"));
```

The `PrismaticGeometry` object allows you to define the pouch geometrical arrangement of the battery cell. You can specify the height, length, and thickness of the cell by setting the **Height**, **Length**, and **Thickness** properties of the `PrismaticGeometry` object. For more information on the possible geometrical arrangements of a battery cell, see the `CylindricalGeometry` and `PouchGeometry` documentation pages.

Now use this `PrismaticGeometry` object to create a prismatic battery cell.

```
batteryCell = Cell(Geometry = prismaticGeometry)
```

```
batteryCell = 
  Cell with properties:

            Geometry: [1x1 simscape.battery.builder.PrismaticGeometry]
    CellModelOptions: [1x1 simscape.battery.builder.CellModelBlock]
                Mass: [1x1 simscape.Value]

Show all properties
```

For more information, see the `Cell` documentation page.

The Cell object allows you to simulate the thermal effects of the battery cell by using a simple 1-D model. To simulate the thermal effects of the battery cell, in the **BlockParameters** property of the **CellModelOptions** property of the `Cell` object, set the **thermal_port** parameter to `"model"`.

```
batteryCell.CellModelOptions.BlockParameters.thermal_port = "model";
```

Create ParallelAssembly Object

A battery parallel assembly comprise multiple battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement. In this example, you create a parallel assembly of one prismatic cell.

To create the ParallelAssembly object, use the `Cell` object you created before and specify the **NumParallelCells** property according to your design.

```
batteryParallelAssembly = ParallelAssembly(Cell = batteryCell,...
    NumParallelCells = 1)
```

```
batteryParallelAssembly = 
  ParallelAssembly with properties:

    NumParallelCells: 1
                Cell: [1x1 simscape.battery.builder.Cell]
            Topology: "SingleStack"
                Rows: 1
     ModelResolution: "Lumped"

Show all properties
```

For more information, see the `ParallelAssembly` documentation page.

Create Module Object

A battery module comprises multiple parallel assemblies connected in series. In this example, you create a battery module of 22 parallel assemblies with an intergap between each assembly of 0.005 meters.

To create the `Module` object, use the `ParallelAssembly` object you created in the previous step and specify the **NumSeriesAssemblies** and **InterParallelAssemblyGap** properties.

```
batteryModule = Module(ParallelAssembly = batteryParallelAssembly,...
    NumSeriesAssemblies = 22, ...
    InterParallelAssemblyGap = simscape.Value(0.005,"m"), ...
    ModelResolution = "Lumped")
```

```
batteryModule = 
  Module with properties:

    NumSeriesAssemblies: 22
       ParallelAssembly: [1x1 simscape.battery.builder.ParallelAssembly]
        ModelResolution: "Lumped"
         SeriesGrouping: 22
       ParallelGrouping: 1
```

```
Show all properties
```

For more information, see the `Module` documentation page.

Create ModuleAssembly Object

A battery module assembly comprises multiple battery modules connected in series or in parallel. In this example, you create a battery module assembly of ten identical modules stacked on ten different levels, with an intergap between each module equal to 0.05 meters. By default, the `ModuleAssembly` object electrically connects the modules in series.

To create the `ModuleAssembly` object, use the `Module` object you created in the previous step and specify the **InterModuleGap** and **NumLevels** properties.

```
batteryModuleAssembly = ModuleAssembly(Module = repmat(batteryModule,1,10),...
    InterModuleGap = simscape.Value(0.05,"m"), ...
    NumLevels = 10)
```

```
batteryModuleAssembly = 
  ModuleAssembly with properties:

    Module: [1x10 simscape.battery.builder.Module]

Show all properties
```

For more information, see the `ModuleAssembly` documentation page.

Create Pack Object

You now have all the foundational elements to create your battery pack. A battery pack comprises multiple module assemblies connected in series or in parallel. In this example, you create a battery pack of one module assembly.

To create the `Pack` object, use the `ModuleAssembly` object you created in the previous step.

```
batteryPack = Pack(ModuleAssembly = batteryModuleAssembly)
```

```
batteryPack = 
  Pack with properties:

    ModuleAssembly: [1x1 simscape.battery.builder.ModuleAssembly]

Show all properties
```

For more information, see the `Pack` documentation page.

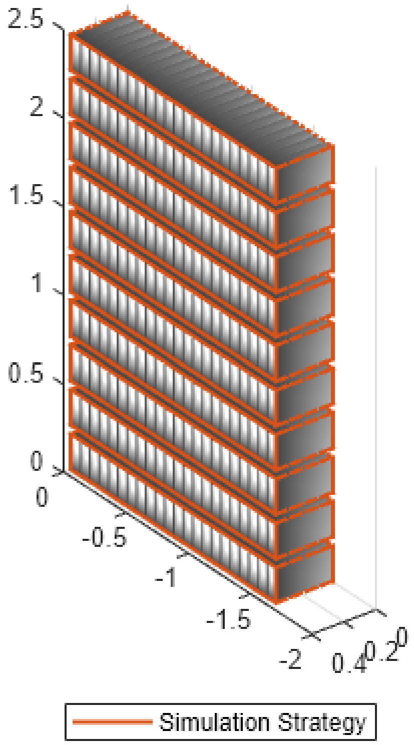
Visualize Battery Pack and Check Model Resolution

To visualize the battery pack before you build the system model and to view its model resolution, use the `BatteryChart` object. Create the figure where you want to visualize your battery pack.

```
f = uifigure(Color="w");
```

Then use the `BatteryChart` object to visualize the battery pack. To view the model resolution of the module, set the **SimulationStrategyVisible** property of the `BatteryChart` object to `"On"`.

```
batteryPackChart = BatteryChart(Parent = f, Battery = batteryPack, ...
    SimulationStrategyVisible = "on");
```



To add default axis labels to the battery plot, use the `setDefaultLabels` method of the `BatteryChart` object.

For more information, see the `BatteryChart` documentation page.

Build Simscape Model for the Battery Pack Object

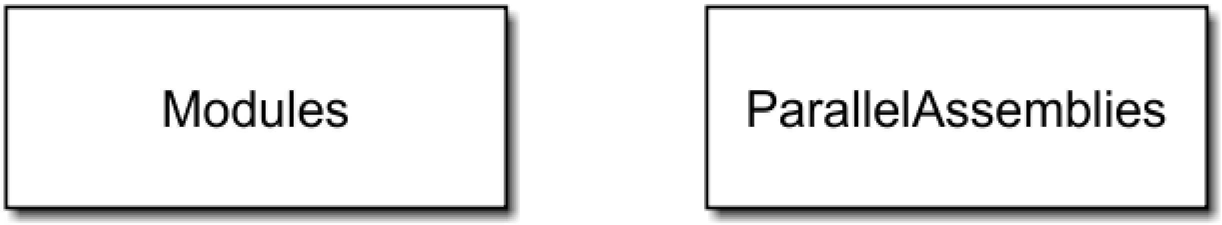
After you have created your battery objects, you need to convert them into Simscape models to be able to use them in block diagrams. You can then use these models as reference for your system integration and requirement evaluation, cooling system design, control strategy development, hardware-in-the-loop, and much more.

To create a library that contains the Simscape Battery model of the `Pack` object you created in this example, use the `buildBattery` function.

```
buildBattery(batteryPack,"LibraryName","packGridExample")
```

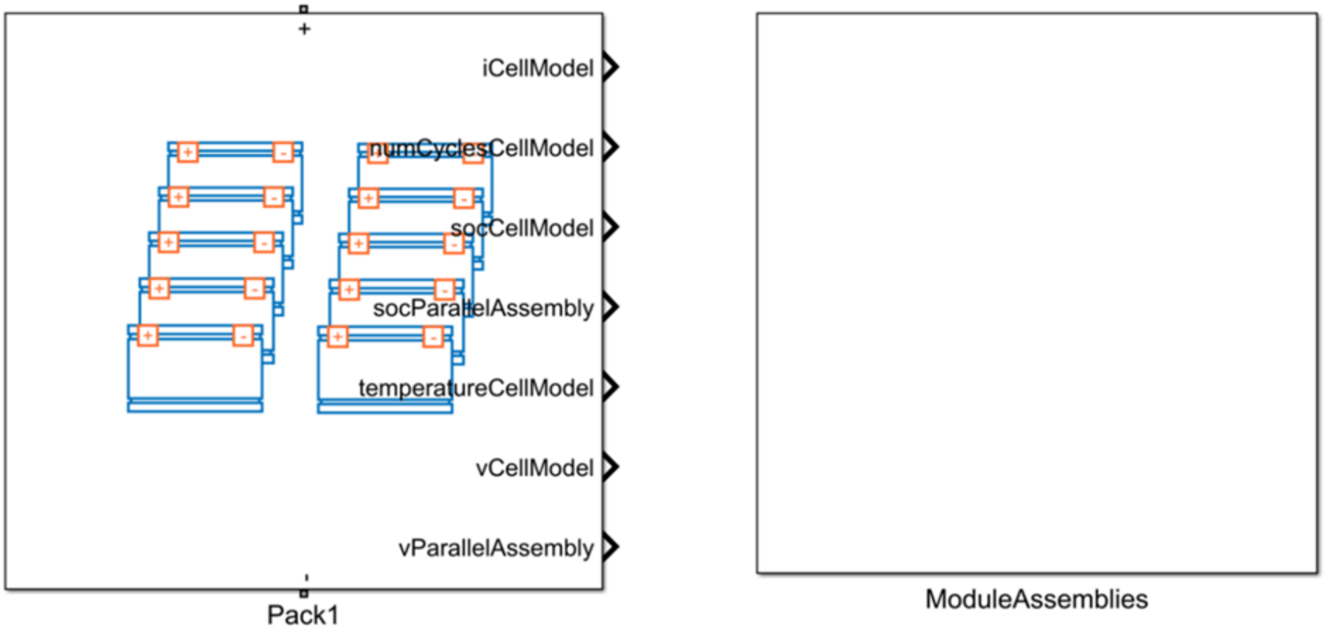
```
Generating Simulink library 'packGridExample_lib' in the current directory 'C:\TEMP\Bdoc22b_2054
```

This function creates the `packGridExample_lib` and `packGridExample` SLX library files in your working directory. The `packGridExample_lib` library contains the Modules and ParallelAssemblies sublibraries.



To access the Simscape models of your `Module` and `ParallelAssembly` objects, open the `packGridExample_lib`. SLX file, double-click the sublibrary, and drag the Simscape blocks in your model.

The `packGridExample` library contains the Simscape models of your `ModuleAssembly` and `Pack` objects.



The Simscape models of your `ModuleAssembly` and `Pack` objects are subsystems. You can look inside these subsystems by opening the `packLibrary` SLX file and double-click the subsystem.

For more information, see the `buildBattery` documentation page.

Copyright 2022 The MathWorks, Inc.

Build a Simple Model of a Battery Module in MATLAB and Simscape

This example shows how to create and build a Simscape™ system model of a battery module in Simscape™ Battery™. The battery module is a 48 V battery for an electric bike application. To create the system model of a battery module, you must first create the `Cell` and `ParallelAssembly` objects that comprise the battery module, and then use the `buildBattery` function. The `buildBattery` function allows you to automatically generate Simscape models for these Simscape Battery objects:

- `ParallelAssembly`
- `Module`
- `ModuleAssembly`
- `Pack`

This function creates a library in your working folder that contains a system model block of a battery module that you can use as reference in your simulations. The run-time parameters for these models, such as the battery cell impedance or the battery open-circuit voltage, are defined after the model creation and are therefore not covered by the Battery Pack Builder classes. To define the run-time parameters, you can either specify them in the block mask of the generated Simscape models or use the **MaskParameters** argument of the `buildBattery` function.

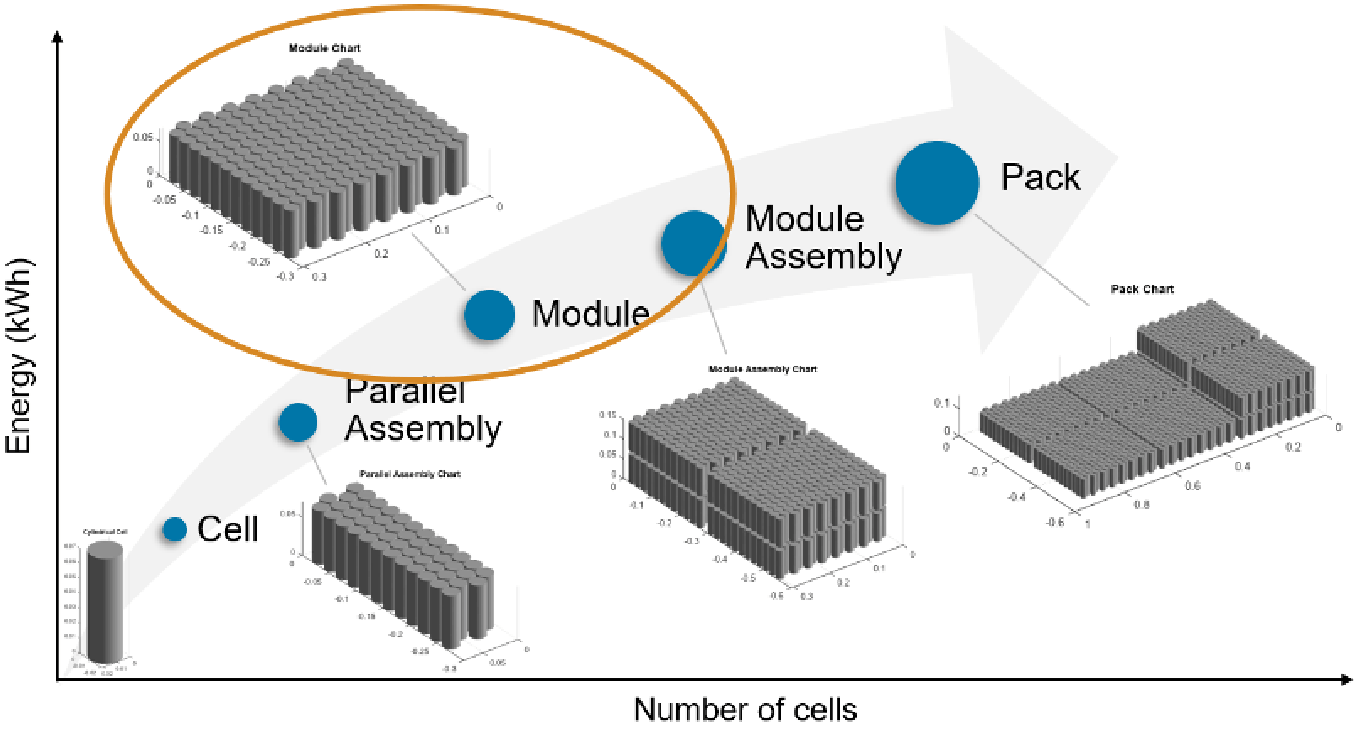
To use the functions and objects in Simscape Battery, first import the required Simscape Battery package:

```
import simscape.battery.builder.*
```

Create Battery Module Object in MATLAB

To create a battery module, you must first design and create the foundational elements of the battery module.

This figure shows the overall process to create a battery module object in a bottom-up approach:



A battery module comprises multiple parallel assemblies. These parallel assemblies, in turn, comprise a number of battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement.

Create Cell Object

To create the battery `Module` object, first create a Cell object of cylindrical format.

```
cylindricalGeometry = CylindricalGeometry(Height = simscape.Value(0.07,"m"),...
    Radius = simscape.Value(0.0105,"m"));
```

The `CylindricalGeometry` object allows you to define the cylindrical geometrical arrangement of the battery cell. You can specify the height and radius of the cell by setting the **Height** and **Radius** properties of the `CylindricalGeometry` object. For more information on the possible geometrical arrangements of a battery cell, see the `PouchGeometry` and `PrismaticGeometry` documentation pages.

Now use this `CylindricalGeometry` object to create a cylindrical battery cell.

```
batteryCell = Cell(Geometry = cylindricalGeometry)
```

```
batteryCell = 
  Cell with properties:

            Geometry: [1x1 simscape.battery.builder.CylindricalGeometry]
    CellModelOptions: [1x1 simscape.battery.builder.CellModelBlock]
                Mass: [1x1 simscape.Value]

Show all properties
```

For more information, see the Cell documentation page.

The Cell object allows you to simulate the thermal effects of the battery cell by using a simple 1-D model. To simulate the thermal effects of the battery cell, in the **BlockParameters** property of the **CellModelOptions** property of the Cell object, set the **thermal_port** parameter to "model".

```
batteryCell.CellModelOptions.BlockParameters.thermal_port = "model";
```

Create ParallelAssembly Object

A battery parallel assembly comprises multiple battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement. In this example, you create a parallel assembly of four cylindrical cells stacked in a square topology over four rows.

To create the ParallelAssembly object, use the Cell object you created before and specify the **NumParallelCells**, **Rows**, and **Topology** properties according to your design.

```
batteryParallelAssembly = ParallelAssembly(Cell = batteryCell,...
    NumParallelCells = 4, ...
    Rows = 4, ...
    Topology = "Square", ...
    ModelResolution = "Detailed");
```

For more information, see the ParallelAssembly documentation page.

Create Module Object

You now have all the foundational elements to create your battery module. A battery module comprises multiple parallel assemblies connected in series. In this example, you create a battery module of 13 parallel assemblies with an intergap between each assembly of 0.005 meters. You also define the model resolution of the module and add an ambient thermal boundary condition.

To create the Module object, use the ParallelAssembly object you created before and specify the **NumSeriesAssemblies**, **InterParallelAssemblyGap**, **ModelResolution**, and **AmbientThermalPath** properties.

```
batteryModule = Module(ParallelAssembly = batteryParallelAssembly,...
    NumSeriesAssemblies = 13, ...
    InterParallelAssemblyGap = simscape.Value(0.005,"m"), ...
    ModelResolution = "Detailed", ...
    AmbientThermalPath = "CellBasedThermalResistance")
```

```
batteryModule = 
  Module with properties:

    NumSeriesAssemblies: 13
       ParallelAssembly: [1x1 simscape.battery.builder.ParallelAssembly]
        ModelResolution: "Detailed"
         SeriesGrouping: [1 1 1 1 1 1 1 1 1 1 1 1 1]
       ParallelGrouping: [4 4 4 4 4 4 4 4 4 4 4 4 4]

Show all properties
```

For more information, see the Module documentation page.

Visualize Battery Module and Check Model Resolution

To obtain the number of Simscape Battery(Table-based) blocks used for the pack simulation, use the **NumModels** property of your `Module` object.

```
disp(batteryModule.NumModels);
```

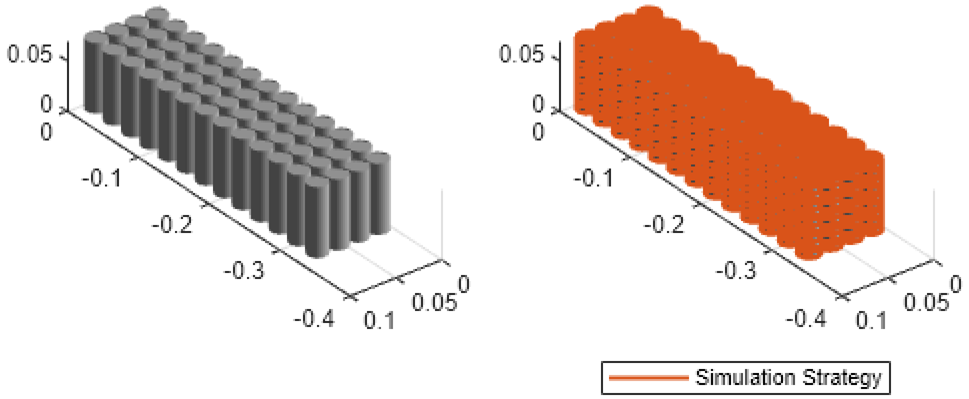
```
    52
```

To visualize the battery module before you build the system model and to view its model resolution, use the `BatteryChart` object. Create the figure where you want to visualize your battery module.

```
f = uifigure(Color="w");
tl = tiledlayout(1,2,"Parent",f,"TileSpacing","Compact");
```

Then use the `BatteryChart` object to visualize the battery module. To view the model resolution of the module, set the **SimulationStrategyVisible** property of the `BatteryChart` object to `"On"`.

```
nexttile(tl)
batteryModuleChart1 = BatteryChart(Parent = tl, Battery = batteryModule);
nexttile(tl)
batteryModuleChart2 = BatteryChart(Parent = tl, Battery = batteryModule, SimulationStrategyVisib
```



For more information, see the `BatteryChart` documentation page.

Build Simscape Model for the Battery Module Object

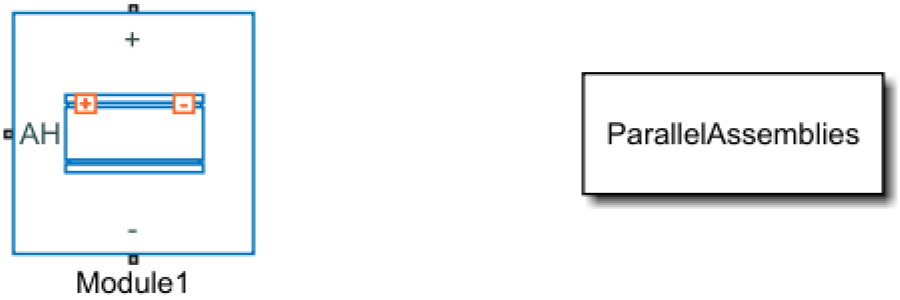
After you have created your battery objects, you need to convert them into Simscape models to be able to use them in block diagrams. You can then use these models as reference for your system integration and requirement evaluation, cooling system design, control strategy development, hardware-in-the-loop, and much more.

To create a library that contains the Simscape Battery model of the Module object you created in this example, use the buildBattery function.

```
buildBattery(batteryModule,"LibraryName","moduleLibrary");
```

```
Generating Simulink library 'moduleLibrary_lib' in the current directory 'C:\TEMP\Bdoc22b_205478
```

This function creates a library named moduleLibrary_lib in your working directory. This library contains the Simscape models of your Module and ParallelAssembly objects.



To build a battery pack model, see the “Build a Simple Model of a Battery Pack in MATLAB and Simscape” on page 4-134 example.

Build a Simple Model of a Battery Pack in MATLAB and Simscape

This example shows how to create and build a Simscape™ system model of a battery pack in Simscape™ Battery™. The battery pack is a 400 V pouch battery for automotive applications. To create the system model of a battery pack, you must first create the `Cell`, `ParallelAssembly`, `Module`, and `ModuleAssembly` objects that comprise the battery pack, and then use the `buildBattery` function. The `buildBattery` function allows you to automatically generate Simscape models for these Simscape Battery objects:

- `ParallelAssembly`
- `Module`
- `ModuleAssembly`
- `Pack`

This function creates a library in your working folder that contains a system model block of a battery pack that you can use as reference in your simulations. The run-time parameters for these models, such as the battery cell impedance or the battery open-circuit voltage, are defined after the model creation and are therefore not covered by the Battery Pack Builder classes. To define the run-time parameters, you can either specify them in the block mask of the generated Simscape models or use the **MaskParameters** argument of the `buildBattery` function.

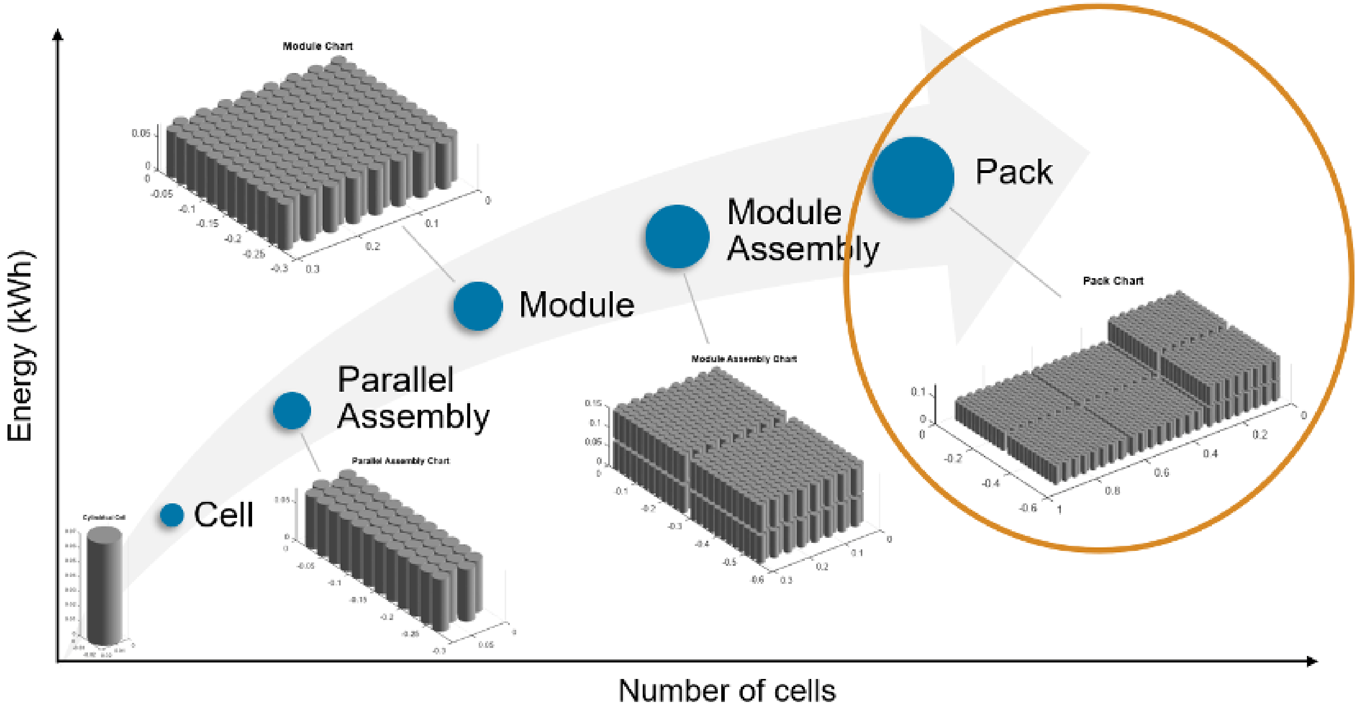
To use the functions and objects in Simscape Battery, first import the required Simscape Battery package:

```
import simscape.battery.builder.*
```

Create Battery Pack Object in MATLAB

To create a battery pack, you must first design and create the foundational elements of the battery pack.

This figure shows the overall process to create a battery pack object in a bottom-up approach:



A battery pack comprises multiple module assemblies. These module assemblies, in turn, comprise a number of battery modules connected electrically in series or in parallel. The battery modules are made of multiple parallel assemblies which, in turn, comprise a number of battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement.

Create Cell Object

To create the battery `Module` object, first create a `Cell` object of pouch format.

```
pouchGeometry = PouchGeometry(Height = simscape.Value(0.1,"m"),...
    Length = simscape.Value(0.3,"m"), TabLocation = "Opposed" )
```

```
pouchGeometry = 
  PouchGeometry with properties:

         Length: [1x1 simscape.Value]
      Thickness: [1x1 simscape.Value]
    TabLocation: "Opposed"
       TabWidth: [1x1 simscape.Value]
      TabHeight: [1x1 simscape.Value]
         Height: [1x1 simscape.Value]
```

The `PouchGeometry` object allows you to define the pouch geometrical arrangement of the battery cell. You can specify the height, length, and location of tabs of the cell by setting the **Height**, **Length**, and **TabLocation** properties of the `PouchGeometry` object. For more information on the possible geometrical arrangements of a battery cell, see the `CylindricalGeometry` and `PrismaticGeometry` documentation pages.

Now use this `PouchGeometry` object to create a pouch battery cell.

```
batteryCell = Cell(Geometry = pouchGeometry)
batteryCell = 
  Cell with properties:

            Geometry: [1x1 simscape.battery.builder.PouchGeometry]
    CellModelOptions: [1x1 simscape.battery.builder.CellModelBlock]
                Mass: [1x1 simscape.Value]

Show all properties
```

For more information, see the `Cell` documentation page.

Create ParallelAssembly Object

A battery parallel assembly comprises multiple battery cells connected electrically in parallel under a specific topological configuration or geometrical arrangement. In this example, you create a parallel assembly of three pouch cells.

To create the ParallelAssembly object, use the `Cell` object you created before and specify the **NumParallelCells** property according to your design.

```
batteryParallelAssembly = ParallelAssembly(Cell = batteryCell,...
    NumParallelCells = 3)
```

```
batteryParallelAssembly = 
  ParallelAssembly with properties:

    NumParallelCells: 3
                Cell: [1x1 simscape.battery.builder.Cell]
            Topology: "SingleStack"
                Rows: 1
     ModelResolution: "Lumped"

Show all properties
```

For more information, see the `ParallelAssembly` documentation page.

Create Module Object

A battery module comprises multiple parallel assemblies connected in series. In this example, you create a battery module of 11 parallel assemblies with an intergap between each assembly of 0.005 meters.

To create the `Module` object, use the `ParallelAssembly` object you created in the previous step and specify the **NumSeriesAssemblies** and **InterParallelAssemblyGap** properties.

```
batteryModule = Module(ParallelAssembly = batteryParallelAssembly,...
    NumSeriesAssemblies = 11, InterParallelAssemblyGap = simscape.Value(0.005,"m"))
```

```
batteryModule = 
  Module with properties:

    NumSeriesAssemblies: 11
       ParallelAssembly: [1x1 simscape.battery.builder.ParallelAssembly]
        ModelResolution: "Lumped"
         SeriesGrouping: 11
       ParallelGrouping: 1
```

```
Show all properties
```

For more information, see the Module documentation page.

Create ModuleAssembly Object

A battery module assembly comprises multiple battery modules connected in series or in parallel. In this example, you create a battery module assembly of two identical modules with an intergap between each module equal to 0.1 meters. By default, the ModuleAssembly object electrically connects the modules in series.

To create the ModuleAssembly object, use the Module object you created in the previous step and specify the **InterModuleGap** property.

```
batteryModuleAssembly = ModuleAssembly(Module = repmat(batteryModule,1,2),...
    InterModuleGap = simscape.Value(0.1,"m"))
```

```
batteryModuleAssembly = 
  ModuleAssembly with properties:

    Module: [1x2 simscape.battery.builder.Module]

Show all properties
```

For more information, see the ModuleAssembly documentation page.

Create Pack Object

You now have all the foundational elements to create your battery pack. A battery pack comprises multiple module assemblies connected in series or in parallel. In this example, you create a battery pack of 5 identical module assemblies with an intergap between each module assembly of 0.01 meters.

To create the Pack object, use the ModuleAssembly object you created in the previous step and specify the **InterModuleAssemblyGap** property.

```
batteryPack = Pack(ModuleAssembly = repmat(batteryModuleAssembly,1,5),...
    InterModuleAssemblyGap = simscape.Value(0.01,"m"))
```

```
batteryPack = 
  Pack with properties:

    ModuleAssembly: [1x5 simscape.battery.builder.ModuleAssembly]

Show all properties
```

For more information, see the Pack documentation page.

Visualize Battery Pack and Check Model Resolution

To obtain the number of Simscape Battery(Table-Based) blocks used for the pack simulation, use the **NumModels** property of your Pack object.

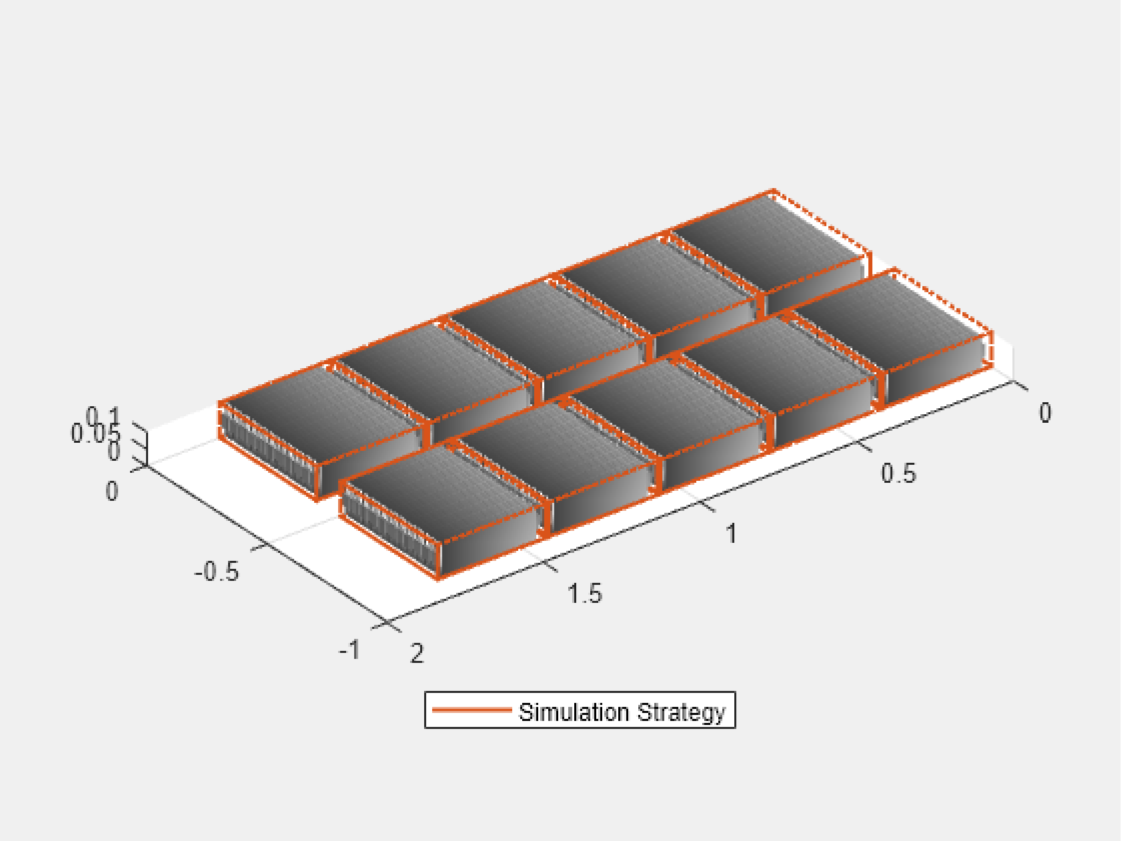
```
disp(batteryPack.NumModels);
```

```
    10
```

To visualize the battery pack before you build the system model and to view its model resolution, use the `BatteryChart` object.

Then use the `BatteryChart` object to visualize the battery pack. To view the model resolution of the module, set the **SimulationStrategyVisible** property of the `BatteryChart` object to `"On"`.

```
batteryPackChart = BatteryChart(Parent = uifigure, Battery = batteryPack, ...
    SimulationStrategyVisible = "on");
```



To add default axis labels to the battery plot, use the `setDefaultLabels` method of the `BatteryChart` object.

For more information, see the `BatteryChart` documentation page.

Build Simscape Model for the Battery Pack Object

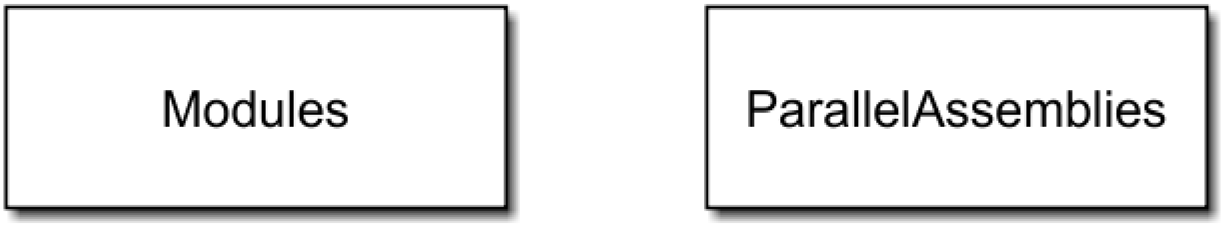
After you have created your battery objects, you need to convert them into Simscape models to be able to use them in block diagrams. You can then use these models as reference for your system integration and requirement evaluation, cooling system design, control strategy development, hardware-in-the-loop, and much more.

To create a library that contains the Simscape Battery model of the `Pack` object you created in this example, use the `buildBattery` function.

```
buildBattery(batteryPack,"LibraryName","packLibrary");
```

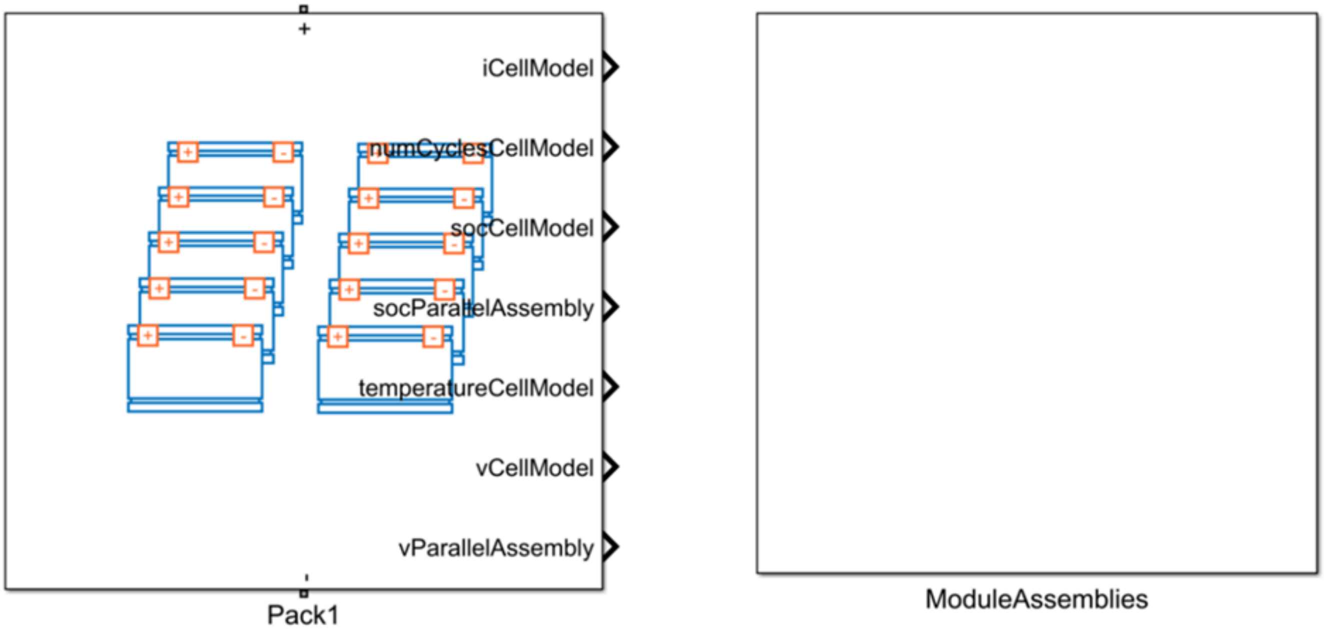
```
Generating Simulink library 'packLibrary_lib' in the current directory 'C:\TEMP\Bdoc22b_2054784_6
```

This function creates the `packLibrary_lib` and `packLibrary` SLX library files in your working directory. The `packLibrary_lib` library contains the Modules and ParallelAssemblies sublibraries.



To access the Simscape models of your `Module` and `ParallelAssembly` objects, open the `packLibrary_lib`. SLX file, double-click the sublibrary, and drag the Simscape blocks in your model.

The `packLibrary` library contains the Simscape models of your `ModuleAssembly` and `Pack` objects.



The Simscape models of your `ModuleAssembly` and `Pack` objects are subsystems. You can look inside these subsystems by opening the `packLibrary` SLX file and double-click the subsystem.

To learn how to include thermal effects in a battery pack, see the “Build Model of Battery Module with Thermal Effects” on page 4-103 example.

To build a more detailed model of a battery pack, see the “Build Detailed Model of Battery Pack From Pouch Cells” on page 4-81 example.

To learn how to model a battery energy storage system (BESS) controller and a battery management system (BMS) with all the necessary functions for the peak shaving, see the “Peak Shaving with Battery Energy Storage System” on page 4-28 example.