Bodo´s Power Systems®

Validating Battery Management Systems with Simulation Models

Battery storage systems are critical technology for the success of electric vehicles and supplementing renewable energy systems. As important as the physical battery pack, the battery management system (BMS) ensures efficient and safe operation over the lifespan of the energy storage system.

By Tony Lennon, Market Manager, Power Electronics Control, MathWorks

When developing the software for a BMS, you need to be mindful of several operational conditions, as shown in Figure 1.

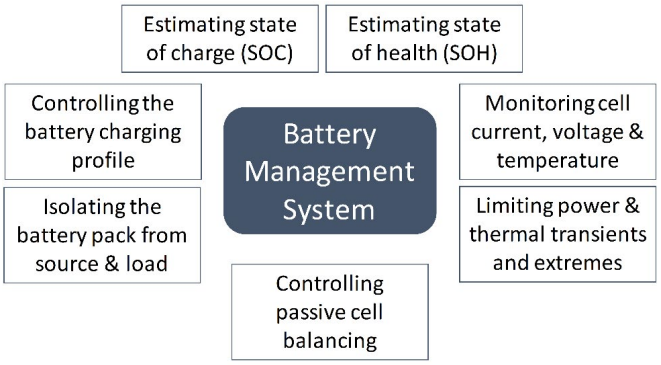

Figure 1: Functions of the battery management system.

To enable the BMS to handle these operations, you could spend time writing code, programming microcontrollers, building battery test systems, and running numerous tests. If you have written all the code perfectly, taken into account every scenario the battery system will see, and run tests for all those cases, your BMS will work as intended. The challenge is that you often cannot test all those scenarios for situations such as damaging equipment, unusual hardware faults, and the time it takes to charge and discharge a battery pack.

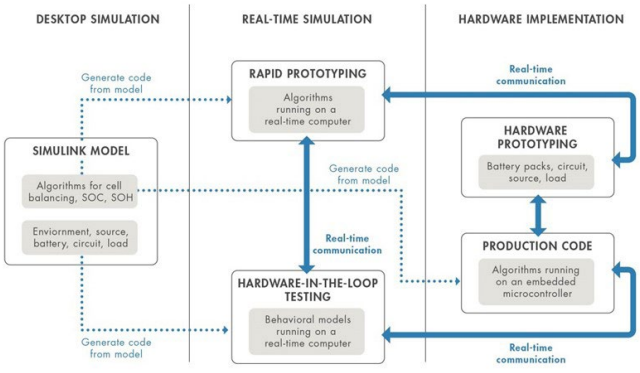

Figure 2: System-level simulation for battery management system development.

Another approach is to use system-level simulation to verify your BMS algorithms and to help validate the BMS software. This does not mean that you will not perform hardware testing. In fact, you will go into hardware testing knowing that your software will have a better chance of handling the normal and abnormal conditions specified for the battery pack, even if you can't test for them. The BMS simulation model starts with desktop simulation of the design's functional aspects, letting you perform formal verification and validation to industry standards, and progresses for use to generate code for real-time simulation and hardware implementation (Figure 2). By simulating the complete battery system before hardware testing, you gain insight into the dynamic behavior of the battery pack, explore software algorithms, and test operational cases.

Desktop Simulation: Modeling BMS Functionality

Using desktop simulation, you verify functional aspects of the BMS design, such as control and monitoring algorithms, cell charge and discharge behavior, and the sizing of passive and active electrical circuit elements. The battery, electrical circuitry, and external environmental conditions and loads are developed as lumped-parameter behavioral models. This approach lets you explore new design ideas and test multiple system architectures before committing to a hardware prototype. For example, you can compare active and passive cell balancing configurations to evaluate the suitability of each approach.

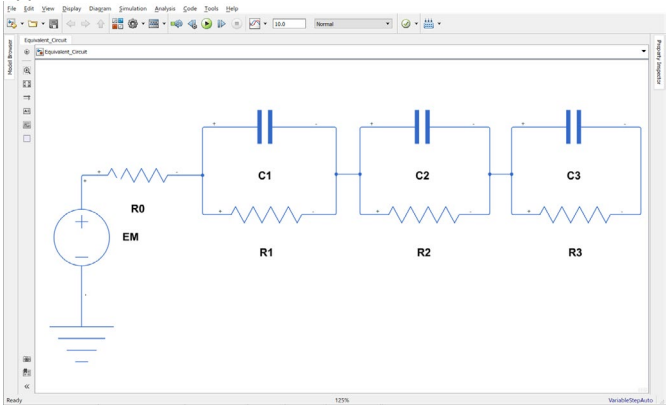

Figure 3: Equivalent circuit of a battery with three-time constants, internal resistance, and open-circuit potential.

Modeling and Characterizing the Battery Cell

For system-level simulation of a battery pack, a common approach is to use an equivalent circuit that simulates the thermoelectric behavior of a cell. As shown in Figure 3, the voltage source provides the open-circuit voltage (OCV), a series resistor models internal resistance, and one or more resistor-capacitor pairs in parallel represent the time-dependent behavior of the cell. These elements are temperature and state of charge (SOC) and are unique to each battery's chemistry, requiring they be determined using test data.

Modeling Power Electronics, Passive Components, Sources, and Loads

Having a complete model of the electrical system lets you understand how the BMS interacts with the battery pack. For example, the simulation can contain a photovoltaic system model to represent a variable charging source for testing the BMS algorithms under changing operating conditions, including fault scenarios. An electrical load on the battery pack, such as an interior permanent magnet synchronous motor in an electric vehicle, can be simulated for standard drive cycles. The remainder of the battery system simulation is made up of active and passive electrical components. These models can vary from simple linear elements to having more complex nonlinear behavior.

Developing Supervisory Control Algorithms

Simulation models make it easy to develop supervisory control algorithms using state machines and flow charts to model combinatorial and sequential decision logic for fault detection and management, charge and discharge power limitation, temperature control, and cell balancing. You can see how the BMS supervises the battery system as it reacts to events, time-based conditions, and external input signals. For example, for constant current, constant voltage (CCCV) charging, you can develop and test the logic that controls when the cell transitions from current charging mode to voltage charging mode.

Estimating State of Charge

Open-circuit voltage (OCV) measurement and current integration (coulomb counting) are traditional SOC estimation techniques for older battery chemistries. Modern battery chemistries that have flat OCV-SOC discharge signatures require a different approach for SOC estimation. Extended Kalman filtering is shown to provide accurate results for a reasonable computational effort. This technique often includes a nonlinear model of the battery and uses the current and voltage measured from the cell as inputs, as well as a recursive algorithm that calculates the internal states of the system (SOC among them).

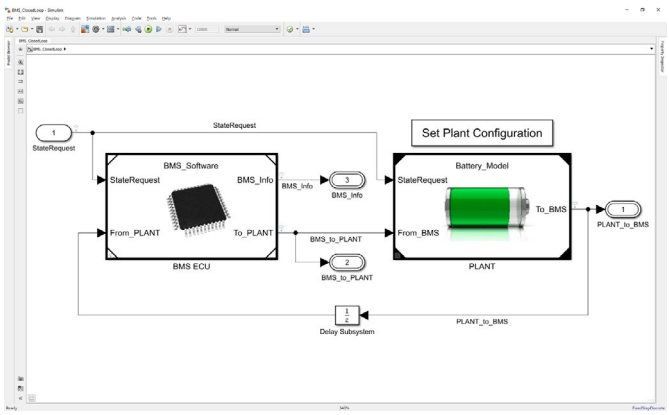

Figure 4: BMS algorithms and plant dynamics, including battery pack and load modeled in Simulink.

Estimating State of Health

Batteries degrade due to calendar life and cycling, increasing internal resistance and losing reserve capacity. The increase in internal resistance is a straightforward estimate using short time estimates. Calculating loss of capacity is challenging because it requires a full charge or discharge excursion for an accurate estimation. Unlike with SOC, there is no standardized agreement on how state of health (SOH) should be estimated. The practice is to use your organization's specific interpretation of battery health.

Testing with Desktop Simulation

As stated previously, fully testing a BMS using a hardware prototype for all use cases may not be practical or safe. Desktop simulation lets you verify BMS algorithms using test cases to exercise all possible branches of logic and closed-loop control. When the battery system must meet safety requirements, you use formal test methods in accordance with standards such as IEC 61508, IEC 61851, and ISO 26262. The simulation model serves as an executable specification driving both the design and testing of the BMS (Figure 4).

Real-Time Simulation: Validating BMS Software

As a step in validating the BMS algorithms, you can use desktop simulation models to generate C and HDL code for real-time simulation, for both rapid prototyping (RP) and hardware-in-the-loop (HIL) testing. With RP, you emulate the BMS controller, letting you begin validating algorithms before implementing code on a microcontroller or FPGA. HIL simulation emulates the balance of the battery system and is used for testing a BMS controller before hardware prototypes are used.

Some advantages of using real-time simulation for BMS design include:

- Validating algorithms before the final controller hardware is selected
- Using the flexibility of a real-time test system for rapid design iteration and testing
- Conducting HIL testing before battery system prototype hardware is available
- Exercising BMS algorithms for test cases that may be difficult, expensive, or destructive if you were to use the actual hardware

Rapid Prototyping

With RP, you generate code from your controller model and deploy it to a real-time computer that performs the functions of the production microcontroller. Code generation empowers the BMS engineer and speeds up the testing process. Algorithm changes made and verified in the desktop model can be tested on real-time hardware in hours

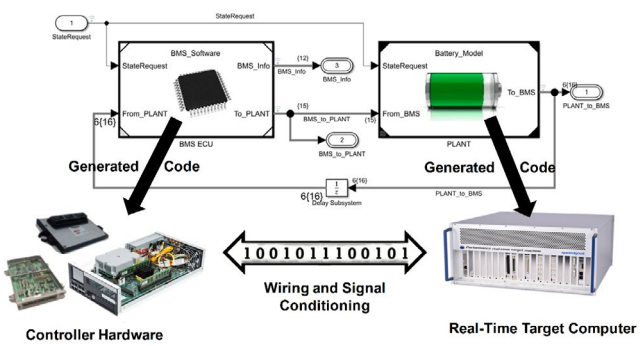

Figure 5: HIL testing of battery management system software. The BMS code is generated from BMS algorithms and deployed to a microcontroller. The battery system model generates code that is implemented on a real-time computer.

rather than the days it could take to get a software engineer to reprogram changes to a microcontroller. Further, most real-time simulation tools can interact with hardware to change algorithm parameters and log test data.

Hardware-in-the-Loop (HIL) Testing

For HIL testing, you use the battery system models rather than the control algorithm models to generate C/C++ or HDL code. This virtual real-time environment represents the dynamic behavior of the battery pack, active and passive circuit elements, loads, the charger, and other system components. When deployed to a real-time computer, you can run simulations of the hardware against your controller code before testing the controller in a battery system prototype (Figure 5). Tests developed during desktop simulation can be carried over to HIL testing, to ensure that requirements are met as the BMS design progresses. As a result, you can find and correct control design errors before they potentially damage expensive and difficult-to-replace prototype hardware. You can also uncover hardware design errors, such as incorrect component sizing.

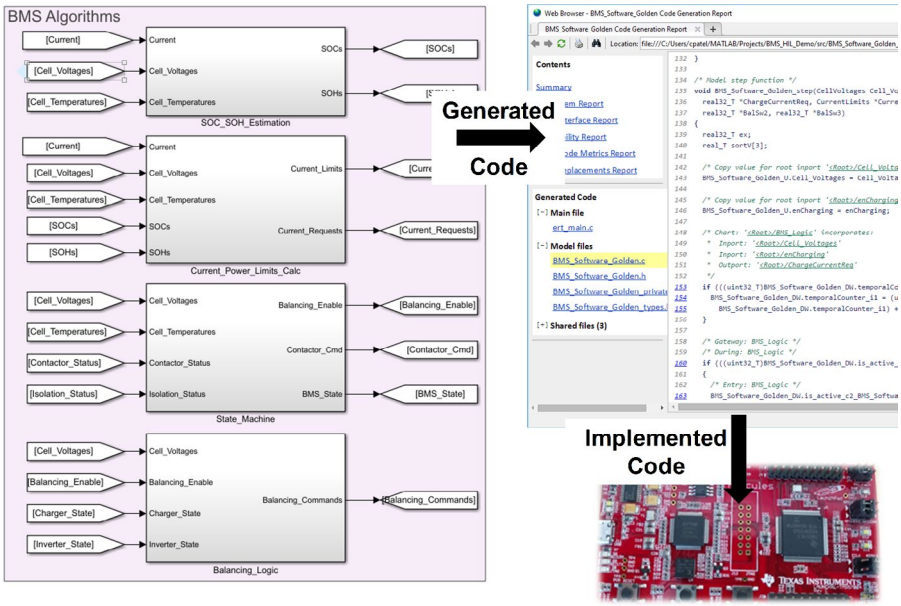

Figure 6: Automatically generating BMS production code from BMS algorithms modeled in Simulink. Code is deployed to a Texas Instruments microcontroller.

Production-Ready Code Generation

After rapid prototyping, the validated control algorithms are the basis for generating production-ready code—either optimized C/C++ code for microcontrollers or synthesizable HDL code for FPGA programming or ASIC implementation. Code generation from the simulation model eliminates manual algorithm translation errors and produces C/C++ and HDL code with numerical equivalence to the algorithms of your desktop simulation. Because you can simulate the BMS algorithms over all possible operating and fault conditions, your generated code will handle those same conditions. If hardware tests indicate that algorithm changes are needed, you can modify the algorithms in your desktop model, rerun simulation test cases to verify the correctness of the changes, and generate new, updated code (Figure 6).

www.mathworks.com