Paper Number 07AE-304

Desktop and HIL Validation of Hybrid-Electric-Vehicle Battery-Management-System Algorithms

Gregory L. Plett
University of Colorado at Colorado Springs, and consultant to Compact Power Inc.

Robert Billings and Martin J. Klein
Compact Power Inc.

ABSTRACT

The battery management system (BMS) of a hybrid-electric-vehicle (HEV) battery pack comprises hardware and software to monitor pack status and optimize performance. One of its important functions is to execute algorithms that continuously estimate battery state-of-charge (SOC), state-of-health (SOH), and available power. The primary difficulty when validating these algorithms is that there are no sensors that can *measure* SOC, SOH, or available power, so the accuracy of the algorithms cannot be directly evaluated. To address this problem, we have developed a validation strategy based in part on a data synthesis system to provide the missing "truth" data. This paper presents the advantages and limitations of such a system, describes how it works, and gives some results.

HEV BATTERY PACK OVERVIEW

Optimizing the cost, weight, size, and reliability of major HEV systems is critical in maximizing the value of the HEV to the end customer. Since the battery pack is among the costliest and heaviest components of the HEV drive train, it is worthwhile to expend effort on a careful design, especially of those components of the pack that might impact its lifetime affordability.

Figure 1 provides a block diagram of a typical HEV battery pack. Comprising the pack are the battery cells, junction module(s), BMS, thermal management system, wiring and connectors, and the pack housing. In most applications, the cells are wired in series to develop the necessary high voltage.

The primary functions of the battery pack are to store electrical energy produced by the vehicle (via the internal combustion engine, or during regenerative braking) and to provide electrical energy for use by the vehicle particularly during acceleration or other peak energy demands. The pack needs to do so in a manner that is safe, reliable, and cost efficient. This includes not only minimizing initial purchase costs, protecting the vehicle from voltage surges or drop-outs, and preventing harmful conditions, but also minimizing operational stresses—such as excessive temperature, discharging or over-charging—that can shorten the life of the battery cells.

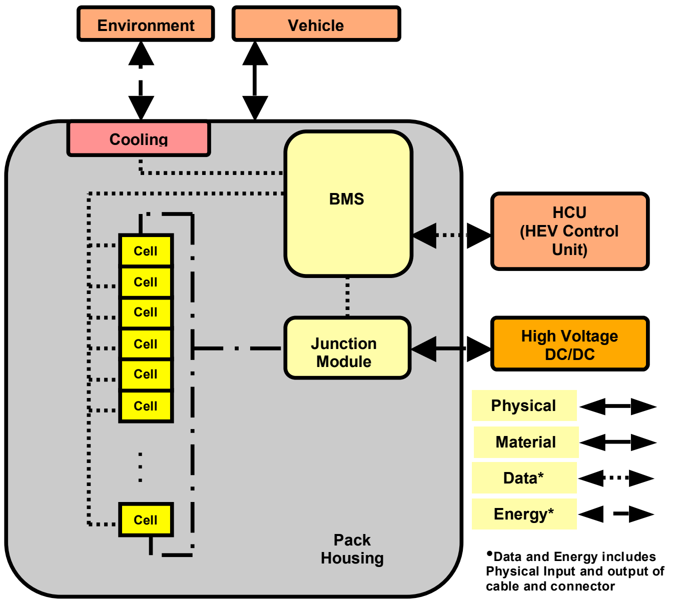


Figure 1
HEV battery pack block diagram

This is accomplished with the aid of the BMS, which manages the delivery and acceptance of electrical energy to/from the cells, as well as the operation of the cooling system and junction module. The BMS consists of a printed circuit board (PCB)—generally under the control of a microprocessor—and connectors (and housing, if necessary). It provides the following functions (among others):

- Cell state monitoring (*e.g.*, voltage, temperature);
- Dis/charge current measurement and limiting;

- Management of the cooling system;
- Necessary data conditioning, diagnostics and battery-to-host vehicle communication functions;
- High voltage relay energizing and de-energizing;
- SOC, SOH, and power estimation, including the effects of aging.

This latter function, in particular, is critical as accurate estimates allow the pack to be used aggressively but without causing damage, allowing for a less expensive, lighter, and more reliable battery system than a similar unit that is over-designed to compensate for poor estimates. The focus of this paper is on validating these algorithms for a production BMS.

VALIDATING BMS ALGORITHMS

Many of the BMS functions are quite straightforward, and their implementation can be validated in standard ways. The internal BMS algorithms that continuously estimate battery SOC, SOH, and available power, however, can be quite complex and pose unique challenges when attempting to qualify them for production systems.

We have previously reported BMS algorithms that perform these tasks [1–3]. Our approach has been based on a "model-based estimation" strategy that uses a mathematical model of cell behavior to predict internal states (*e.g.*, SOC, SOH) and measurable outputs (*e.g.*, voltage) based on measurable inputs (*e.g.*, current and temperature). Any deviation between the true measured output and the predicted value of the measured output can be attributed to model inaccuracy, measurement noise, and/or an error in the internal model state. An algorithm is used to adapt the internal state to balance between these effects and reduce the prediction error.

Regardless of which BMS estimation algorithm is used, the primary difficulty when validating any such algorithm on physical hardware is that the "truth" values are not known. There are no sensors that can directly measure SOC, SOH, or available power. At different points in time, laboratory tests can be performed that can be used to determine *a posteriori* what the SOC/SOH/power was at that time, but cannot determine SOC/SOH/power in real time, as the battery pack operates.

We have developed a validation methodology or system comprising two data/execution parts and a third evaluative part as one component of a strategy that overcomes this obstacle (cf. Figure 2). The first part is a software simulator of cell dynamics that can synthesize various driving, temperature, parameter, and sensor-fault profiles for the battery pack being modeled. All internal variables that are dependent on the cycling history of the cell (*e.g.*, SOC/ SOH/ power) are known to the software simulator, so "truth" values are established. The input-output behavior of this model has been tested against physical cells, and works well. We call this the "Data Generator System" (DGS).

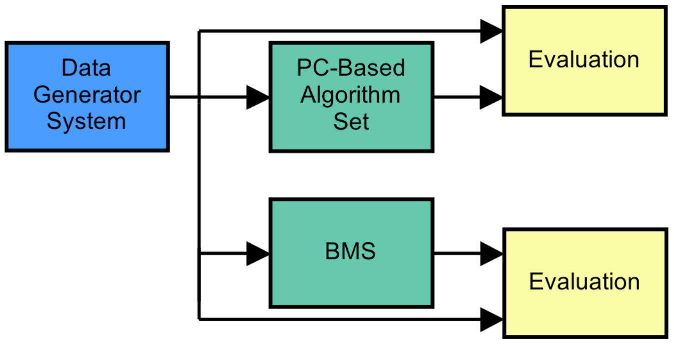


Figure 2
Main components of the validation system

The second main part of the system executes the BMS algorithms, using the synthetic cell data as input, and compares the algorithm results, in real time or accelerated time, to the "truth" values. This can be performed using the prototype algorithms on a desktop PC (which we call the BMS Algorithm Simulation System (BASS)), or by feeding the simulated voltage/ current/ temperature data to the actual BMS hardware to accomplish hardware-in-the-loop (HIL) validation. The estimates generated by either the desktop or the HIL system are compared to the truth values via an evaluation routine, and conclusions are drawn as to the quality of the estimates.

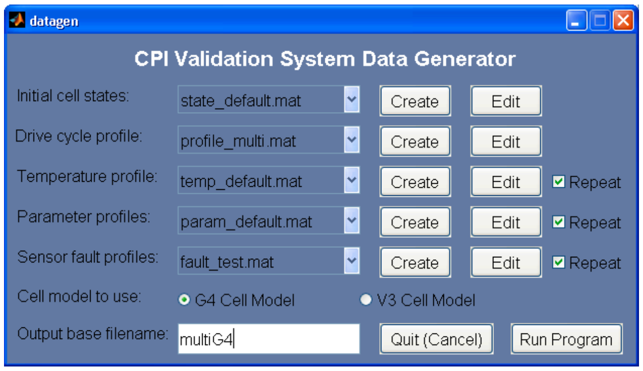
Some benefits of this approach are:

1) Standardizes validation of major releases and incremental algorithm changes;
2) Low cost—does not require expensive pack cyclers or environmental chambers;
3) Identifies areas of sensitivity to direct focused efforts in future development;
4) Provides quantitative measures of rate of convergence of estimates;
5) Assesses impact of cell-to-cell variations and the effect of aged cells—pack refurbishing and matching problems.

The following sections describe the cell model used to synthesize the data, the Data Generator System, the BMS Algorithm Simulation System, and the HIL Validation System. We give some lessons learned from testing and conclude.

THE DATA GENERATOR SYSTEM

The data-generator system is written in the MATLAB^{TM} scientific computing language. It comprises a main computational core, along with a graphical user interface (GUI) to allow simple control over the system. In this section, we first describe the primary element of the



Initial cell states: state_default.mat
Drive cycle profile: profile_multi.mat
Temperature profile: temp_default.mat ✓ Repeat
Parameter profiles: param_default.mat ✓ Repeat
Sensor fault profiles: fault_test.mat ✓ Repeat
Cell model to use: ● G4 Cell Model ○ V3 Cell Model
Output base filename: multiG4

Figure 3
Screen capture of the DGS main GUI

core—the cell model used to synthesize truth data—and then describe features of the GUI.

CELL MODEL USED TO SYNTHESIZE TRUTH

In order for the proposed validation scheme to be effective, the DGS must be able to synthesize cell input-output and state information that matches that of a physical cell operating under the same conditions with very high fidelity. If the accuracy of this step cannot be guaranteed, then it is impossible to interpret the results of the algorithms meaningfully.

Our DGS uses a mathematical model of the cells used in our BMS—called the "Enhanced Self-Correcting" (ESC) cell model—that has been carefully crafted over several years, and reported in a number of places [4,5]. To summarize, the inputs to the ESC are: cell current, cell temperature, and cell parameters, and the output is cell voltage. The model has internal states that keep track of polarization voltage(s), hysteresis voltage, and state-of-charge, and the output voltage is computed as the sum of the polarization voltages, ohmic voltage, OCV, and hysteresis. The parameters of the cell model include: cell capacity, cell resistance, time constants, and mixing factors for the polarization voltages, and maximum hysteresis.

The ESC has been optimized to predict performance of the cells in our BMS under highly dynamic conditions expected in an HEV environment [5]. Voltage estimation errors are zero-mean, with typical RMS error in the range of 5–10mV over the 10%–90% SOC range, for temperatures greater than –10°C (lower-temperature performance improvement of the model is the subject of present and future work).

THE MAIN DGS GUI

In order to simulate cell states and outputs when generating data using the ESC model, the data generator needs a fair amount of information. First, the initial values of all states for all cells are required. Secondly, the drive-cycle (current) profile[1] is needed. Third, the temperature of all cells as a time profile must be given. Fourth, all cell model parameters for all cells as a function of time must be entered. Finally, sensor faults and noises must be given.

Data for each of these five basic categories of information are entered via a number of GUI windows, are stored in individual files, and are later combined when synthesizing the drive cycle. The individual DGS GUIs are managed by a main MATLAB GUI, shown in Figure 3.

One of the sub-GUIs allows entry of the initial cell state information (polarization voltages, SOC value, hysteresis voltage) for all of the cells in the battery pack. Another allows entry of the drive cycle as a current-versus-time profile. Such current profiles comprise a sequence of sub-profiles, which may be either: dynamic drive cycles (e.g., UDDS, US06, HWFET, etc.), rest intervals, or constant-current intervals. The sub-profiles may be sequenced in any desired order.

Temperature profiles are entered in a similar way. They comprise constant-temperature intervals and ramping temperature intervals, sequenced in any desired order. Different temperature profiles may be entered for each cell, if desired.

Parameter profiles may be entered for every parameter of every cell in the battery pack. The profiles are generated in a way that allows either the default value of the parameter (temperature dependent, as specified in the ESC model), or a biased version of the default parameter, or simply replacing the default parameter with another value.

Sensor-fault and sensor-noise profiles may be entered for every sensor (current, voltage, temperature) in the BMS. Sensor faults include: "stuck on", "stuck off", and biases; noises are white Gaussian random variables of a given variance. Note that these sensor faults/noise profiles are not used when computing the true cell states and parameters in the DGS, but are used when computing the measured cell quantities. That is, they represent sensor faults, not cell faults. Note also that faults are applied first, and noise added to the faulted sensor.

[1] The word "profile" is used in this paper to refer to any function of time. All profiles in the DGS are discrete-time sequences with an interval of one second between samples. Note that while HEV dis/charge dynamics are considerably faster than this rate, the time constants of battery SOC and SOH are considerably slower, so we find this sample rate to be sufficient.

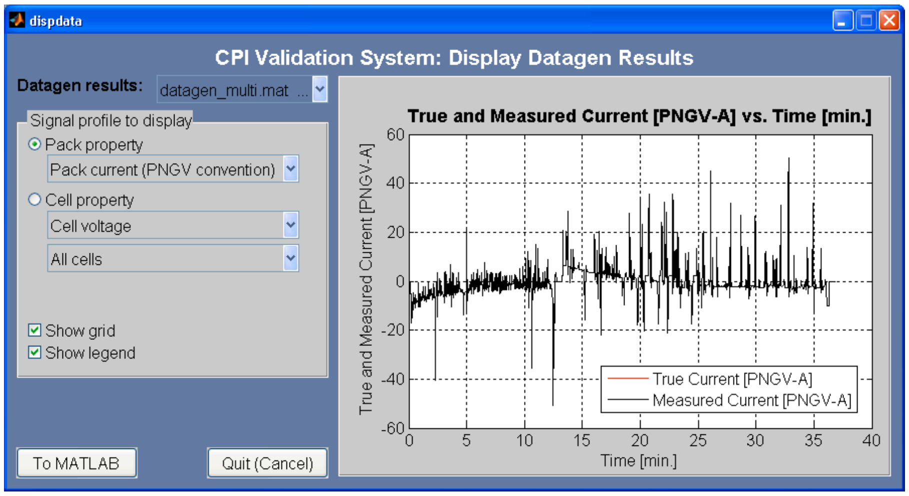

dispdata
CPI Validation System: Display Datagen Results
Datagen results: datagen_multi.mat
Signal profile to display
Pack property
Pack current (PNGV convention)
Cell property
Cell voltage
All cells
Show grid
Show legend
To MATLAB
Quit (Cancel)
True and Measured Current [PNGV-A] vs. Time [min.]
True and Measured Current [PNGV-A]
Time [min.]
True Current [PNGV-A]
Measured Current [PNGV-A]

Figure 4
Screen capture of the data display GUI

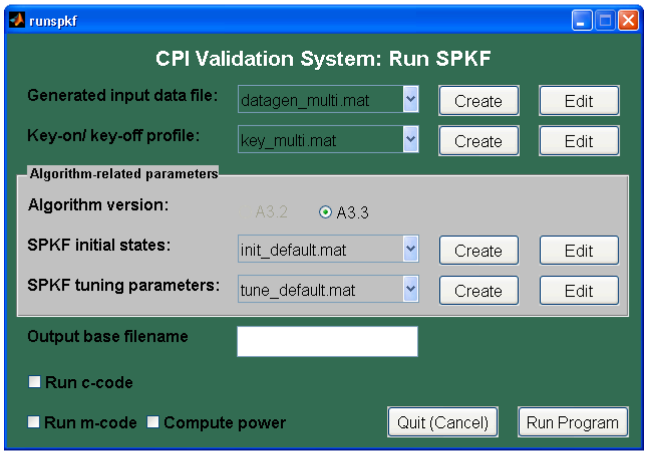
Temperature-, parameter-, and fault profiles need not be the same length (in time) as the drive-cycle profile. If a profile is longer than the drive-cycle profile chosen, it is simply truncated and the first portion of the profile is used. If it is shorter than the drive-cycle profile, then two options exist. The default is that the temperature (etc.) profile is only applied for the first portion of the drive cycle; after that, the default temperature (25 °C), default parameter, or default sensor fault/noise (no fault, no noise) is used. However, if the associated “Repeat” checkbox is selected, the temperature (etc.) profile is repeated over and over to fill the entire drive cycle.

When all inputs are specified, the data generator may be run. The output of the data generation is stored in a file for later input to the BASS. This file comprises all true sensor values, all measured sensor values, all true parameters, all true states, and maximum power capability of the pack. The contents of this output file may be viewed by a separate GUI, shown in Figure 4. (Note that in this particular example there was no sensor noise, so the “true current” is identical to the “measured current”. Had sensor noise been included in the data synthesized by the DGS for this example it would have been immediately evident in the GUI.)

THE BMS ALGORITHM SIMULATION SYSTEM

After “truth” and measured data has been generated by the DGS, they may be used as input to the BMS algorithms in order to test how well they estimate cell states, parameters, and available power. A MATLAB GUI has been written to facilitate this task, and is shown in Figure 5. The GUI allows entering the desired truth data file (generated by the DGS, as just discussed), a key-on/ key-off profile, the version of the algorithms to be used in the simulation, the initial states and parameters assumed by the BMS algorithms, and which version of code to execute.

The second row of the main BASS GUI allows associating a key-on value (i.e., when in time relative to the drive cycle the vehicle and the BMS is turned on) and/or a key-off value (i.e., when in time the vehicle and the BMS is turned off, and (presumably) the battery pack is allowed to rest) with every time-step in the data file, to allow testing the algorithm initialization and shut-down procedures. Note that for events nominally occurring during the same time interval, key-on events happen first, the algorithms then execute if the key is in the “on” state, and key-off events happen last.


runspkf

CPI Validation System: Run SPKF

Generated input data file: datagen_multi.mat
Key-on/ key-off profile: key_multi.mat

Algorithm-related parameters
Algorithm version: A3.2 [✓ A3.3]
SPKF initial states: init_default.mat
SPKF tuning parameters: tune_default.mat

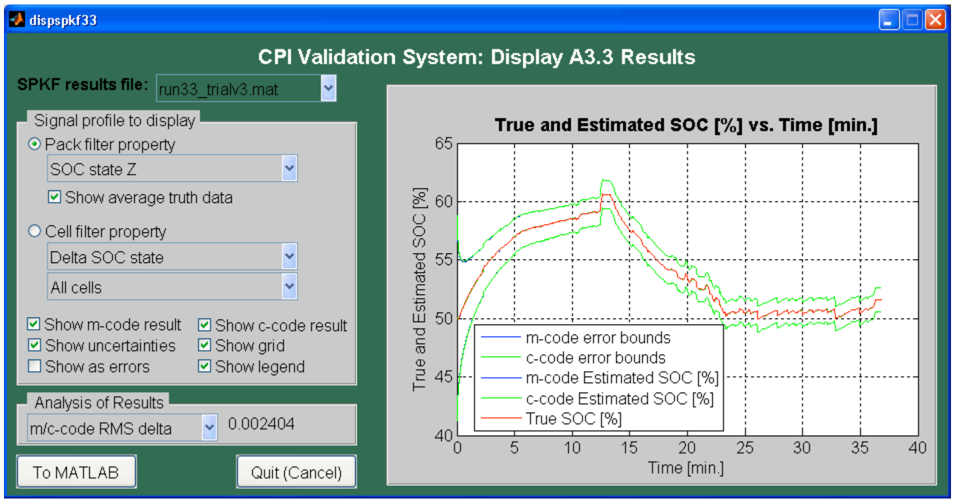
Output base filename:

☐ Run c-code
☐ Run m-code ☐ Compute power

Figure 5
Screen capture of the main BASS GUI

BASS allows using different versions of the estimation algorithms, and allows different cell models to be used as well. (The cell version chosen when computing the truth data in DGS is automatically used so does not need to be re-entered here). The algorithm version may then be selected; for example, the figure shows the version of algorithm that we call A3.3 to be selected.

The different algorithm versions require different information in order to run. Generally, however, they all need some state/parameter initialization, and they need



CPI Validation System: Display A3.3 Results

SPKF results file: run33_trialv3.mat

Signal profile to display

[✓ Pack filter property] SOC state Z

[✓] Show average truth data

[ ] Cell filter property: Delta SOC state; All cells

[✓] Show m-code result [✓] Show c-code result

[✓] Show uncertainties [✓] Show grid

[ ] Show as errors [✓] Show legend

Analysis of Results

m/c-code RMS delta 0.002404

Figure 6
Screen capture of the display results GUI

some tuning parameters, which control the amount of sensor/process noise that the algorithms expect.

All states and parameters may either be set to default values or be replaced by some constant that the user enters. For example, in a hard-reset key-on event, the default SOC is computed using open-circuit voltage and temperature. However, to test the robustness of the algorithms to poor initialization, the user may enter any value of the initial SOC estimate here. It is then possible to see how quickly the algorithm converges to the correct SOC from its bad initial estimate.

When the initial state/parameter and tuning information has been entered, it is time to run the algorithms. The prototype MATLAB "m-code" algorithms, and/or the hand-coded compiled C-language ("c-code") algorithms may be run.

When the algorithms have completed, the results may be displayed in a separate GUI (cf. Figure 6). This GUI allows powerful data visualization. All traces of state/parameter profiles—both truth and estimate—may be viewed. The state/parameter may be viewed in its natural units and compared to the truth value, or may be viewed as an estimation error, where the error is computed as truth minus estimate. Error bars (three-sigma uncertainties as estimated by the algorithms) may be overlaid, and grid and legend may be added. An analysis pane automatically computes either the RMS estimation error of the present trace or the percentage of time the error bounds correctly encompass the truth value. Notice that in this particular example the MATLAB m-code and the c-code produce results that are too similar to distinguish by eye. In fact, the analysis pane shows that the RMS difference between the two curves is 0.002404(%). We also notice that the difference between "true SOC" and estimated SOC is indistinguishable by eye (which is typical for the advanced SOC estimation methods we use when everything is initialized properly). The present figure may also be copied into a standard MATLAB figure window, allowing zooming, editing, saving, printing, and so forth.

THE HIL SYSTEM

Our algorithm development cycle has several phases. The algorithms are first prototyped in MATLAB (for ease and speed of development), then hand-coded in the C programming language (for speed of execution, and less demanding final processor requirements) on a desktop platform, and finally included in the software that executes on the BMS hardware. The preceding discussion has shown how we can validate the MATLAB and desktop C code, and here we discuss how the same method can be extended to validate the C code executing on the BMS hardware.

The idea of HIL battery algorithm validation is certainly not unique to us (see, for example, [6]). Our enhancement to this idea, however, is that we have intentionally designed the HIL system to interface with the desktop system, so that a further level of validation can be performed on the same data set, with the same expected results. The desktop c-code validation system and the HIL c-code validation system intentionally use the same algorithm code-base to reduce the chance of introducing programming errors. The main difference between the two systems is that the desktop system reads measurements from files, while the HIL system reads measurements from slave processors connected via a serial bus. In the production BMS, the slave processors in turn measure cell voltages, cell temperatures, and pack current using appropriate sensors, but in the HIL system we have introduced a method to inject synthetic data created by the DGS into the main BMS via the slave processor serial bus. This is achieved by first converting the MATLAB cell model from the DGS into Simulink^{TM}—a platform for multi-domain

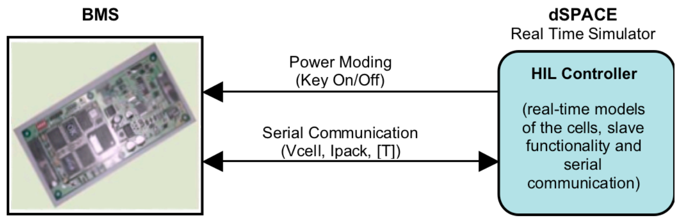


Figure 7
Block diagram of HIL system.

simulation of dynamic systems. The slave functionality is modeled in Simulink and couples the cell model to a current versus time drive cycle. A model of the serial bus is also used for transmission of the cell voltages, current and temperature matrix to the BMS. These Simulink models are then converted to a form that is run on a real-time platform. In our case, this platform is dSPACE^{TM} (cf. Figure 7 for a block diagram). The BMS receives the appropriate information from the simulator as it normally would from actual slave processors.

With the real-time models of the cell and slave behavior, the validation benefits of the desktop are greatly enhanced. The actual implementation of specific algorithms, such as SOC, can be evaluated on the BMS. This can illuminate many issues that can be masked by running algorithms in the desktop environment with double precision data. Timing constraints and fixed-point errors are a few. In the case of the SOC algorithm, the evaluation routine compares the known truth values from the simulator to the BMS estimated output and is able to validate the performance in real time.

We have also devised a method whereby algorithm tuning parameters may be downloaded to the BMS independently of the main BMS software itself. Therefore, the HIL system may be tested using different tuning information without the need to recompile software every time a new test is attempted. This also allows us to intentionally inject bad initialization data to the algorithms to test robustness to conditions we do not expect to occur in practice, but to which we must be able to respond.

The HIL system is designed but is still in progress of being implemented. It will be able to execute all of the tests described in the next section when completed.

THE TEST SUITE

The DGS, BASS, and HIL system just described provide a very flexible framework that can be used to execute an immense variety of tests on the algorithms. CPI has contracted with a third party, Emmeskay [7], whose personnel have helped in defining the tests that should be run to properly validate the algorithms.

The present suite of tests for use in the desktop-validation system comprises 729 different scenarios. Some of these test the normal operation of the algorithms, and others test the robustness of the algorithms to improper initialization, sensor failures, and the like. A range of static and time-varying ambient temperatures are considered, and a variety of driving profiles are also included. If these tests were implemented in real time, a number of months would be required to run the entire test suite (neglecting the time required to set thermal chamber temperatures, etc.). However, the entire set of tests can be run on the desktop system in either two hours (the hand-optimized "c-code" or in 24 hours (the MATLAB "m-code"), approximately. We find it valuable to test both code sets from time to time to make sure that the results agree, as they should.

The HIL system operates in real time, so it is not feasible to implement all of these 729 scenarios. We are presently pursuing a design-of-experiments approach to construct a more practical set of tests to run on this system. We expect to report on these results in the future. The HIL system is also ideal for Design Failure Modes and Effects Analysis (DFMEA).

LESSONS LEARNED TO DATE

The multi-level validation system that we have presented in this paper has provided a number of benefits and lessons learned. We itemize several here:

- The speedup of the DGS versus a real-time HIL implementation allowed testing many scenarios in accelerated time. This would not have been possible otherwise, and helped give confidence in which aspects of the algorithm were working (and which needed more attention).
- The ability of the DGS/BASS to instantly plot any desired internal signal from a file containing data from a specific test was invaluable to understanding which factors were important and which were marginal. Aspects of the algorithms that we "knew" were working in fact needed more attention.
- Being able to execute the prototype m-code and production c-code with the same stimulus (which should have produced the same output) helped us discover and eliminate a number of very subtle programming errors.
- Generally, it will be impossible to meet all specifications all the time (especially if testing robustness and intentionally trying to confuse the algorithms). It is necessary to prioritize the importance of each test to be performed to balance nominal operation against robustness.

CONCLUSION

This paper presents an approach to validating algorithms, such as SOC estimation, for a battery management system. Several levels of validation are proposed: First, validation of the prototype m-code, next validation of the desktop c-code, after that, validation of the production software. To overcome the lack of any sensor that can measure "truth" for the quantities of interest in real time, all of these levels of testing use synthetic test data generated from a cell model, and benefit from the "truth" being known by construction. The proposed approach also benefits from being inexpensive, fast, and can give a good indication of how well the algorithms will function in practice.

This method is no substitute for actual testing of real cells. However, it can help to minimize the amount of this testing that is required if a careful design-of-experiments approach is taken. A balanced overall validation strategy therefore comprises a large suite of desktop validation tests and a smaller suite of real-time tests on physical battery packs.

REFERENCES

1. G. Plett, Sigma-point Kalman filtering for battery management systems of LiPB-based HEV battery packs: Part 1. Introduction and state estimation, International Journal of Power Sources, Vol. 152, No. 2, October 2006, pp. 1356–68.
2. G. Plett, Sigma-point Kalman filtering for battery management systems of LiPB-based HEV battery packs: Part 2. Simultaneous state and parameter estimation, International Journal of Power Sources, Vol. 152, No. 2, October 2006, pp. 1369–84.
3. G. Plett and M. Klein, Advances in HEV Battery Management Systems, Proc. SAE Convergence 2006, Paper Number 2006-21-0060.
4. G. Plett, LiPB dynamic cell models for Kalman-filter SOC estimation, in: CD-ROM Proceedings of the 19th Electric Vehicle Symposium (EVS19), (Busan, Korea: October 2002).
5. G. Plett, Results of Temperature-Dependent LiPB Cell Modeling for HEV SOC Estimation, in: CD-ROM Proceedings of the 21st Electric Vehicle Symposium (EVS21), (Monaco: April 2005).
6. C. Massey, A. Bekaryan, P. Liu, A. Parulian, L. Turner, D. Frisch, T. Weber, and M. Verbrugge, Hardware-in-the-Loop Testing for Electrochemical Cells in Hybrid Electric Vehicles, SAE Technical Paper Series, Commercial Vehicle Engineering Congress and Exhibition, Chicago, Illinois, November 1–3, 2005, Paper 2005-01-3500.
7. www.emmeskay.com. Accessed 9/1/2006.

CONTACT

Gregory Plett is Associate Professor of Electrical Engineering at the University of Colorado at Colorado Springs and consultant to Compact Power Inc. He may be reached at:

Dept. of Electrical and Computer Engineering,
University of Colorado at Colorado Springs,
1420 Austin Bluffs Parkway, P.O. Box 7150,
Colorado Springs, CO 80933–7150 USA
Tel: +1–719–262–3468, Fax: +1–719–262–3589,
E-mail: glp@eas.uccs.edu,
URL: http://mocha-java.uccs.edu

Robert Billings is Electrical/Electronics Engineer with Compact Power Inc. He may be reached at:

Compact Power Inc.,
1857 Technology Drive, Troy, MI 48083 USA
Tel: +1–248–291–2381
E-mail: rbillings@compactpower.com
URL: http://www.compactpower.com

Martin Klein is Director of Engineering at Compact Power Inc. He may be reached at:

Compact Power Inc.,
1857 Technology Drive, Troy, MI 48083 USA
Tel: +1–248–291–2379
E-mail: mklein@compactpower.com
URL: http://www.compactpower.com