

# Battery Pack Modeling, Simulation, and Deployment on a Multicore Real Time Target

Javier Gazzarri, Nishant Shrivastava, Robyn Jackey, and Craig Borghesani  
MathWorks

## ABSTRACT

Battery Management System (BMS) design is a complex task requiring sophisticated models that mimic the electrochemical behavior of the battery cell under a variety of operating conditions. Equivalent circuits are well-suited for this task because they offer a balance between fidelity and simulation speed, their parameters reflect direct experimental observations, and they are scalable. Scalability is particularly important at the real time simulation stage, where a model of the battery pack runs on a real-time simulator that is physically connected to the peripheral hardware in charge of monitoring and control. With modern battery systems comprising hundreds of cells, it is important to employ a modeling and simulation approach that is capable of handling numerous simultaneous instances of the basic unit cell while maintaining real time performance.

In previous publications we presented a technique for the creation of a battery cell model that contains the electrochemical fingerprints of a battery cell based on equivalent circuit model fitting to experimental data. In this work we extend our previous model to represent a battery pack, featuring cell creation, placement, and connection using automation scripts, thus facilitating the design of packs of arbitrary size and electrical topology. In addition, we present an assessment of model partitioning schemes for real time execution on multicore targets to ensure efficient use of hardware resources, a balanced computational load, and a study of the potential impact of the calculation latencies inherent to distributed systems on solver accuracy. Prior to C code generation for real time execution, a model profiler assesses the model partitioning and helps determine the multicore configuration that results in the lowest average turnaround time, the time elapsed between task start and finish.

The resulting model is useful in the generation of multiple operating scenarios of interest in the design of charging, balancing, and safety related procedures.

**CITATION:** Gazzarri, J., Shrivastava, N., Jackey, R., and Borghesani, C., "Battery Pack Modeling, Simulation, and Deployment on a Multicore Real Time Target," *SAE Int. J. Aerosp.* 7(2):2014, doi:10.4271/2014-01-2217.

---

## INTRODUCTION

Many battery applications, especially those of high power levels, require the series or parallel connection of several unit battery cells, with a topology dictated by the electrical characteristics of the load. As the battery pack size increases, so does the computational demand on the solver for the simulation of the system. This problem becomes more acute if the model needs to run in real-time for hardware-in-the-loop (HIL) testing. Modern computing hardware offers capabilities such as parallel execution that can increase simulation speed, but it is generally not easy to fully take advantage of these capabilities, largely due to the significant challenges posed by multithreading the model execution process.

Real time execution on multicore targets would appear to be suitable for battery pack simulation because the repetitive nature of a battery pack suggests a relatively straightforward way to evenly partition the model. This apparent simplicity notwithstanding, the implications of partitioning a model in terms of its data transfer latencies on solver accuracy is a more difficult problem to solve.

Desktop simulation is a natural first step before executing generated code on a HIL platform. This work presents a technique in which the designer uses desktop simulation to determine optimum partitions of the model prior to code generation and execution on target hardware. This step enables the designer to achieve a balanced computational load distribution across the multithreaded HIL application.

A second goal is to provide a scalable methodology capable of supporting any number of battery cell components configured in series or parallel. A MATLAB script creates, places, and connects each model component, including battery cell blocks and electrical components for the battery pack load, and partitions the model.

## MODEL PARTITIONING

As system complexity increases, so too does the computational demand placed on simulation hardware. When modeling and simulating a battery pack, thermal management and charge balancing require a model that encompasses the entire battery pack and that can be simulated in real time.

Real time capability is an essential requirement for HIL testing. For large models, real time performance can be difficult to achieve, especially if the system contains components that are numerically stiff, comprising subsystems with diverse time constants. One option for achieving real time simulation of large models is to distribute computational load by partitioning for multicore execution. With this approach, some tasks are executed concurrently, decreasing the execution time required by the solver to advance each simulation step.

In Simulink, the user configures a model for on-target concurrency by assigning separate *tasks* to different parts of the model at the top level. Each task requires an associated sampling time that must be an integer multiple of the top-level model step size. This partitioning is done irrespective of the target hardware specifications, with a scheduler assigning the tasks to CPU cores in an optimal way without the need of user intervention. This detachment of model development from hardware mapping makes this approach scalable and easy to employ with increasingly powerful targets. In Figure 1, model blocks are assigned tasks that the Simulink scheduler later uses to generate multithreaded C code for on-target execution.

Although concurrency is only possible at the on-target execution stage, Simulink offers a profiling tool to assess the performance of the partitioned model on the desktop. This tool simulates the effects of model partitioning in terms of computational load balance and shows quantitative information on maximum and average turnaround times for the execution of each task. We use this tool to assess the partitioning strategy of a cell pack.

The often homogeneous architecture of a battery pack makes it suitable for concurrency since the model can be partitioned into similar units that should result in a balanced load on the computing hardware. It is important, however, to validate the partitioned model to ensure that delays and data transfer latencies introduced by the partitioning do not undercut solver performance. The first part of the Results section describes this validation step in detail. Subsequent parts of the Results section describe the process used to choose the optimal partitioning scheme for a battery pack architecture of eight lithium ion cells in series.

## RESULTS

### Baseline Model

The baseline model used to validate partitioning strategies consists of eight cells in series. Each battery cell block contains an equivalent circuit with state-of-charge and temperature dependent parameters, as described in [1,2,3]. Figure 2 shows the benchmark model to be compared against partitioned models. Battery cell, voltage sensor, current source, and heat transfer blocks are part of the Simscape library, and they are interconnected forming a physical (acausal) network. In this simulation, a sequence of 25A discharge pulses partially drains the initially fully charged battery pack during one hour.

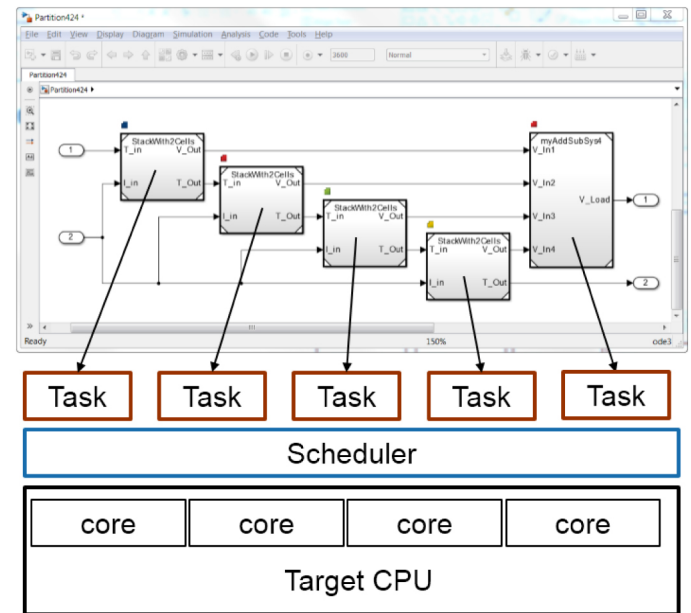


Figure 1. Partitioning schematic for concurrent execution in Simulink. The user assigns different tasks to different parts of the model and a scheduler finds an optimal hardware distribution of the computational load [4].

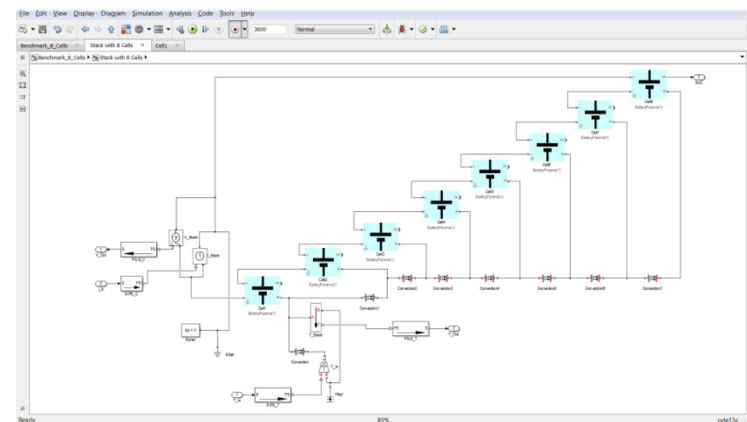


Figure 2. Baseline model with eight lithium ion battery cells in series. Thermal connection blocks model convective heat transfer between adjacent cells. The baseline model uses the variable step implicit solver (ode15s) recommended for numerically stiff systems.

Inputs to the model are the discharge current and the ambient temperature, and outputs are the pack voltage and temperature. The baseline model uses a variable-step solver for maximum accuracy. Subsequent versions of the model are configured with a fixed-step solver to enable real-time simulation. It is important to compare the variable step results and the fixed step results to identify any possible losses in accuracy, particularly during rapid changes in voltage and current.

Figure 3 shows the voltage (V), temperature (°C), current (A), and state of charge (or SoC) traces as a result of the electrical and thermal stimuli.

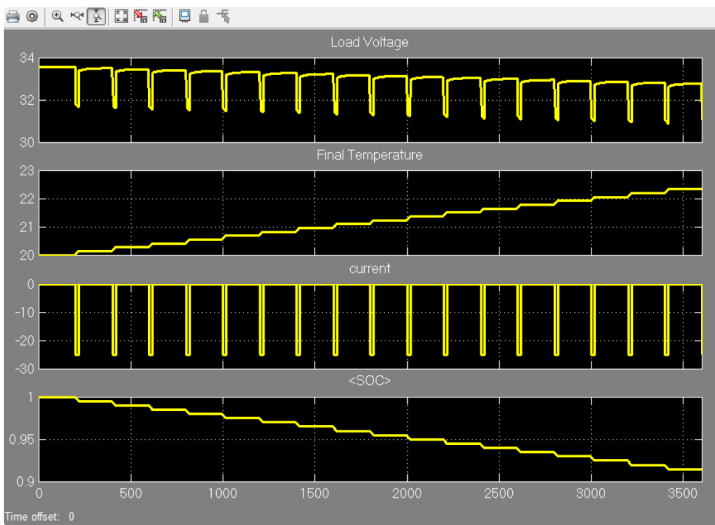


Figure 3. Pack voltage (V), temperature (°C), discharge current (A), and SoC (-) for the benchmark model.

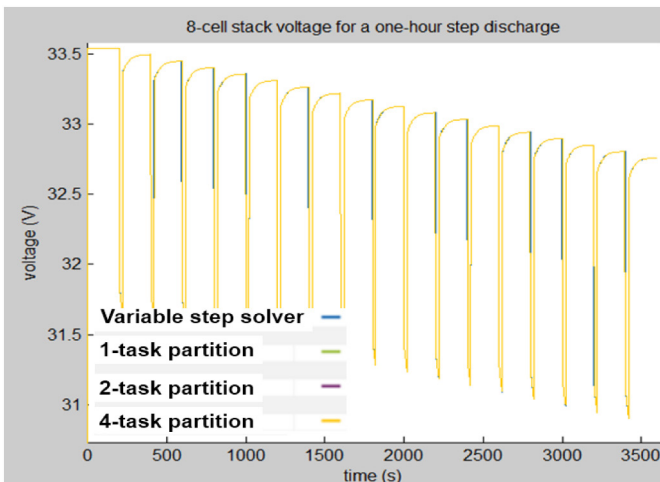


Figure 4. Eight-cell pack voltage response for the baseline simulation and three partitioning schemes: 1) No partition with variable step solver, 2) No partition fixed step solver, 3) Partition in two tasks with four cells each, and 4) Partition in four tasks with two cells each.

Figure 4 shows the voltage response of all partitioned schemes superimposed on the baseline results. The similarity of the results validates the three partitioned versions of the model:

- 1). 1-task partition: One task for the entire eight cell pack. Fixed step solver
- 2). 2-task partition: Two-task partition with four cells each in each task. Fixed step solver
- 3). 4-task partition: Four-task partition with two cells in each task. Fixed step solver

### Partitioning Procedure

For a given battery pack size, several partitioning schemes can be conceived for real time concurrent execution. A four-core real time computer suggests, for example, a partitioning of four units with two cells each. When partitioning, it is important to take into consideration the performance and accuracy implications of imposing execution boundaries on the model, since this arrangement also imposes potential data transfer latencies between any two adjacent partitions. This compromise between execution time improvement and latency management lends itself to an optimization scheme.

The automated workflow of model creation streamlines the task of exploring for optimal partitioning schemes. The workflow enables the user to evaluate several configurations and to assess the benefits of multicore computing. The user can also evaluate trade-offs in performance imposed by scheduler overhead and the solver performance implications of inter-task data transfer latencies. Simulink provides a profiling tool to perform this trade-off evaluation.

To support an automated and convenient method for exploring different partitioning architectures, we create the battery pack model using MATLAB scripts. This approach enables the creation of a battery pack model and its partitioning into tasks with a simple call to a function. The script combines unit blocks according to arguments provided to it. The base function is named `CE_CreateModel` and it has seven arguments.

```
CE_CreateModel(ModelName, n_Stacks,
               n_Cells, n_Tasks, base_rate, f, choice)
```

The arguments to this function are: the desired model name, the number of stacks into which the battery pack will be divided, the number of cells in series inside each stack, the number of tasks into which the model is to be partitioned, the top-level solver sampling rate, an integer factor that multiplies the base sample rate if any part of the model needs to be solved at a slower frequency, and a choice of desktop simulation or real time simulation.

The `CE_CreateModel` function needs the following blocks to be predefined:

- 1). Battery cell. This block defines the battery type in the form of a lookup-table based equivalent circuit, including thermal effects.
- 2). Load. This block specifies the battery charge/discharge cycle.
- 3). Input. This block specifies the ambient temperature.
- 4). Output. This block contains the output blocks such as plotting scopes.

Using these blocks, the MATLAB script automatically constructs a battery pack model by placing and connecting any number of battery cell blocks electrically and thermally, connects the battery pack to its load, and specifies configuration preferences and partitions. Functions such as `add_block` and `add_line` are used for the script-level manipulation of Simulink blocks. The loops below show examples of how we create a stack of `n_Cells` battery cells and make the series electrical connections between adjacent cells.

```

% Placement of n_Cells battery cell blocks and
creation of a handle to its port designation
for i = 1:n_Cells
    myCells(i) = add_block('LiBatteryBlocks/Lithium
Cell', [StackName '/Cell', num2str(i)]);
    myCellPorts(i) = get_param(myCells(i),
'PortHandles');
end
% Series connection of battery cell blocks
for i = 1:n_Cells-1
    add_line(StackName, myCellPorts(i).LConn(1),
myCellPorts(i+1).LConn(2));
end
    
```

The MATLAB script is also responsible for the task mapping, i.e. assigning model blocks to different tasks so that the scheduler can multithread them for distributed execution. The following example code shows how tasks are assigned to battery cell blocks. The vector `myBlock` is a handle to each stack for which the user specifies a separate task.

```

k = 1;
l = 1;
for i = 1:n_Tasks
    for j = k : 1*blocks_per_task;
        taskConfig.map(myBlock(j),myTask(i));
    end
    k = k + blocks_per_task;
    l = l+1;
end
    
```

Figure 5 shows an example pack created using this technique. Two battery cells in series are inside each of the Stack1, Stack2, Stack3, and Stack4 blocks. The small rectangle above each model reference block designates the task to which it is mapped.

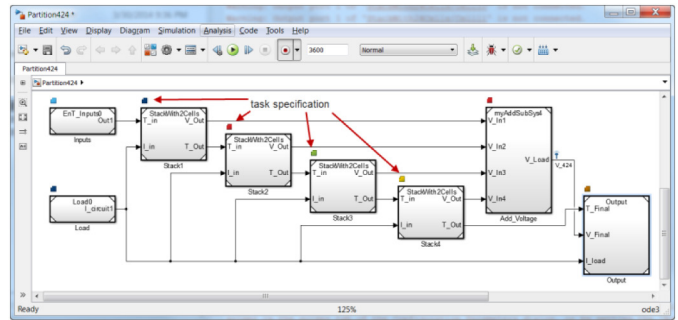


Figure 5. Top level of the 8-cell pack model partitioned in four 2-cell stacks and mapped into four tasks.

Although all blocks at the top level require a task assignment, in this example the computational load of the tasks corresponding to the four blocks outside of the battery pack is negligible since these blocks consist of very simple elements. Figure 6 shows the simple two-cell circuit inside each Stack block.

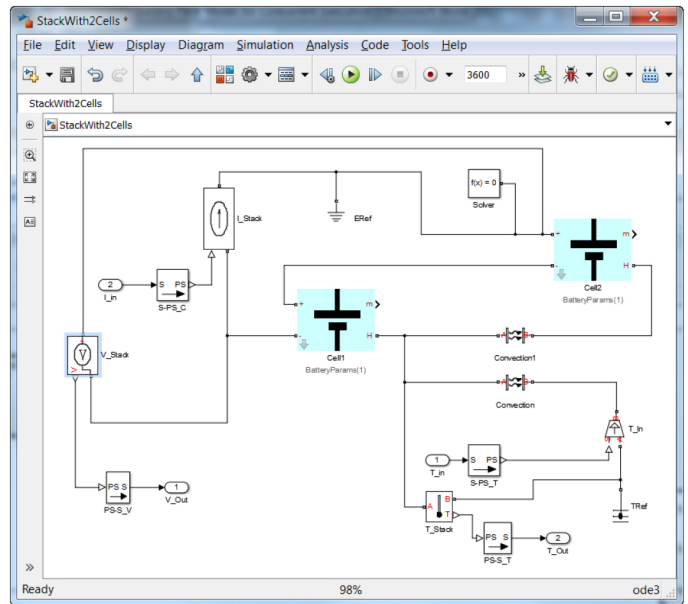


Figure 6. Battery blocks inside each Stack block of Figure 5. The two cells are connected in series both electrically and thermally.

### Desktop Profiling

Simulink provides a profiling tool that can be used in desktop simulation, before a model is downloaded to the real time computer to assess computational load balancing. The results of this profiler include a chart showing the load distribution, the average and maximum turnaround times per task assigned in the model, and a diagram showing the task-to-core mapping. The following sections illustrate the use of this tool on the battery model, using a Lenovo laptop with an Intel® Core™ i7-2620M Dual Core CPU running at 2.7GHz, hyperthreading on. Operating system: Windows 7 6.1.

**Case 1 - No Partitioning (or one-task partition), Fixed Step Solver @10 ms (one stack with eight cells and one task)**

This case is equivalent to the baseline case except it is configured to use a fixed step solver. There is no concurrency because the eight battery cells are all mapped onto a single task. The average turnaround time for the stack block is 209µs (Figure 7). Load balance is, of course, not applicable in this case since there is no partitioning. It is important to consider that since this report is the result of performance metrics measured on a non-real time desktop computer, its results should not be taken as a quantitative prediction of what the real time simulation performance will be, since that simulation will occur on different hardware.

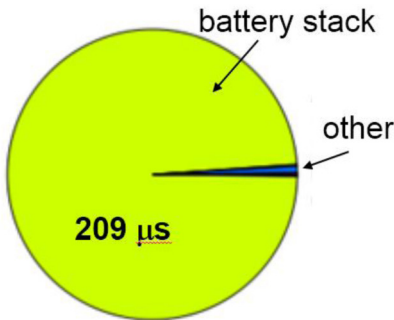


Figure 7. Profiler report for no concurrency, fixed-step solver @10ms. The average turnaround time was 209 µs for the task in charge of solving the battery stack. The blue portion corresponds to non-stack components such as source and sink Simulink blocks.

**Case 2 - Two-Task Partitioning (two stacks with four cells each and two tasks)**

The second case under consideration features a two-task partitioning scheme, and the profiler report shows a twofold increase in average execution speed, with a 108 µs maximum (103 µs average) time. In this case, the benefit of distributing the computational load over two tasks is not offset by scheduler overhead.

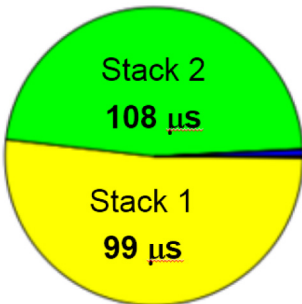


Figure 8. Profiler report for two-task partitioning, fixed-step solver @10ms. The average execution time is roughly half of that for no concurrency, revealing an almost 100% improvement in execution speed.

**Case 3 - Four-Task Partitioning (four stacks with two cells each and four tasks)**

Since the real time computer we use features a quad-core processor, partitioning the pack into four main tasks seems like a natural choice. Figure 9 shows the profiling results for this case.

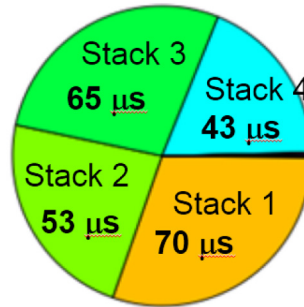


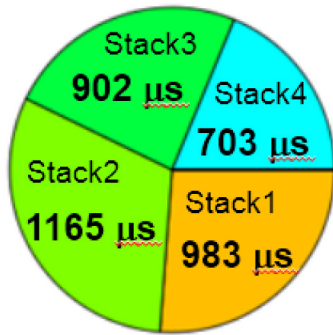
Figure 9. Profiler report for four-task partitioning, fixed-step solver @10ms.

The average turnaround time for the four-task partitioning ranged between 43 µs and 70 µs, with an average of 57 µs, slightly more than 25% of the average time for the original configuration, revealing the effect of the scheduler and data transfer overheads due to multithreading. In addition, the profiling also indicates that this partition configuration results in a slightly less balanced computational load, as revealed by the pie chart.

**Case 4 - Large Model (four stacks with 24 cells each and four tasks)**

The benefits of automating the creation and partitioning of the models are more apparent when the process is scaled up. This section describes a process equivalent to those already presented, although for a significantly larger model. In this case we simulate 96 cells in series, partitioned as four stacks of 24 cells each. The profiling tool indicates an average turnaround time ranging between 703 µs and 1165 µs, 16 times larger (on average) than the previous example, for a model 12 times larger. This larger-than-linear increase in computing time with respect to model size is an important aspect to account for when scaling-up models for real time execution.

Figure 10 illustrates one further feature of this tool: the task distribution schematic, which shows the processor core assignment (zero-based, 0 to 3 for a quad-core machine) of each task as a function of solver step. This diagram is useful in exploring load balancing as a function of simulation time.



Clock Tick / Task	0	10
Add_Voltage	2 2 3 3 3 3 2 2 2 3 1 2 1 3 1 2 2	
Bat_Stack1	1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1	
Bat_Stack2	0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3	
Bat_Stack3	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
Bat_Stack4	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	
Input	1 1 1 1 1 1 2 1 1 1 2 1 2 1 1 3 1	
Load	2 3 3 3 3 3 3 3 3 3 0 3 3 3 1 3 3	
Output	2 2 2 2 0 0 0 0 0 0 0 0 0 0 2 0 0	

Figure 10. Profiling results for a model comprising 96 cells in series. Top: The execution time is 16 times longer than the previous example, while the stack size is 12 times larger. Bottom: Task assignment to processor cores for the first 20 solver steps. Numbers indicate zero-based core identification.

### REAL TIME RESULTS

The ultimate goal of the procedure described above is to prepare a battery pack model for real time simulation. A typical application of real time simulation is HIL testing. Since the desktop computer hardware used for the profiling is, in general, different from the real-time computer, the results obtained from the former are not quantitatively transferable to the latter. The non-deterministic nature of the operating system running on the desktop computer reinforces this statement. However, it is best if the desktop configuration matches as closely as possible the real time computer architecture.

Real time execution on a PC (HP xw4600 with Intel Core2 Quad processor @2.5 GHz, 6.144MB RAM) running Simulink Real-Time™ kernel for Cases 1, 2, and 3 from the previous section resulted in the traces shown in Figure 11, Figure 12, Figure 13. These figures show the task execution time and its hardware core assignment (number at the lower left corner outside each bar) for a given sample time snapshot. Bar overlap in the vertical direction indicate task concurrency. A close look at the bar lengths also reveals that the execution time reduction between Case 1 and Case 2 is slightly larger than it is between Case 2 and Case 3, consistent with the desktop profiling results from the previous section.

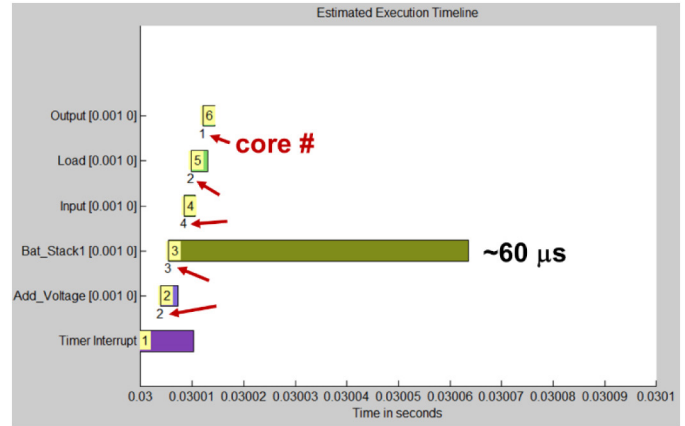


Figure 11. Snapshot of computational load distribution at 0.03sec. Case 1 - Real Time. No battery pack partitioning, fixed step solver @10ms (one stack with eight cells and one task - Bat\_Stack1). Arrows indicate processor core number.

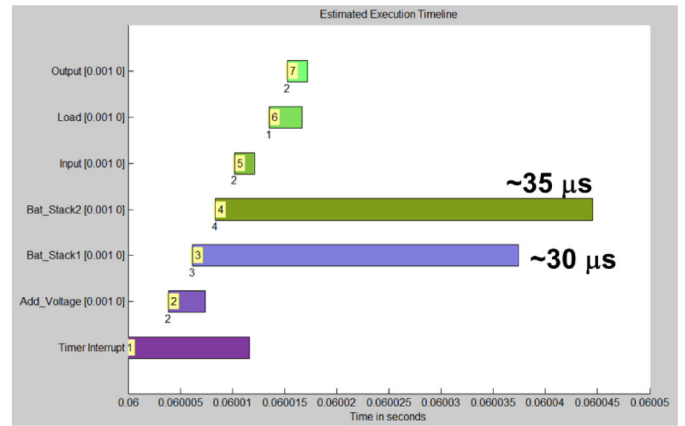


Figure 12. Snapshot of computational load distribution at 0.06sec. Case 2 - Real Time. Two-task partitioning (two stacks with four cells each and two tasks). Execution time for Bat\_Stack1 and Bat\_Stack2 was almost half of that in Case 1.

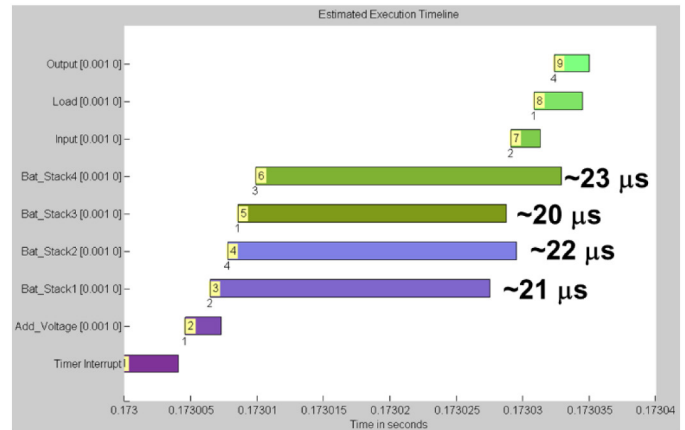


Figure 13. Case 3 - Four-task partitioning (four stacks with two cells each and four tasks). A twofold increase in partitioning with respect to Case 2 resulted in a less than twofold reduction in execution time for the tasks assigned to the battery pack.

Finally, Figure 14 shows the real time simulation of the 96-cell battery pack model for a four-task partition. The model ran in real time without overruns, with an initial latency of approximately 500 ms

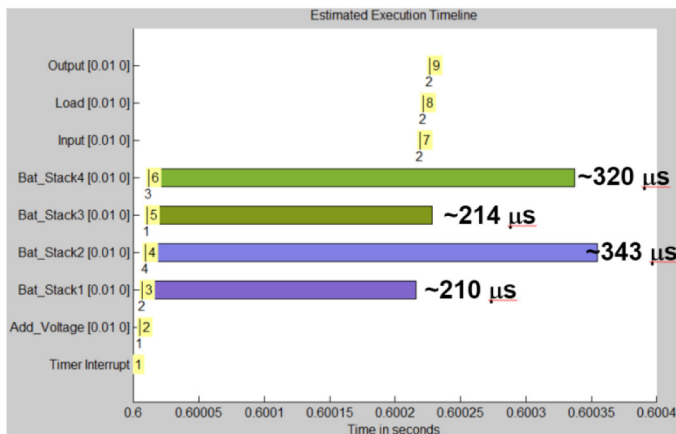


Figure 14. Snapshot of computational load distribution at 0.6sec. Case 4 - Four-task partitioning, large model (four stacks with 24 cells each and four tasks)

## CONCLUSIONS AND OUTLOOK

This work presents a flexible method for constructing battery pack models suitable for partitioning and multicore execution on real time targets. The method features automated block creation, placement, connection, and partitioning using a MATLAB script. Scripting model creation makes it easier to simulate numerous partitioning architectures and identify an architecture optimized for minimum execution time or balanced computational load distribution. The profiling tool enabled the qualitative assessment of the benefits and trade-offs of the different partitioning schemes on a desktop computer, and showed a nonlinear increase in execution time when applied to models of different size.

All models and partitions were tested in the real time environment with a sample rate of 10 milliseconds. The eight-cell model ran in real time with a performance that qualitatively agreed with the desktop profiling prediction when tested under different partitioning conditions. The four-task partition scheme was finally used on the 96-cell model, which also ran real time. As the number of partitions increased, more time was required for initialization.

The method presented in this article is applicable to cases outside of battery system design by replacing and renaming the blocks to be handled automatically by the automation scripts. The script for model creation and set up could also become the objective function of an optimization scheme to find optimal configurations in a systematic and automated way.

## REFERENCES

1. Huria, T., Ceraolo, M., Gazzarri, J., Jackey, R., "High fidelity electrical model with thermal dependence for characterization and simulation of high power lithium battery cells", *Electric Vehicle Conference (IEVC), 2012 IEEE International*, March 2012
2. Jackey, R., "A Simple, Effective Lead-Acid Battery Modeling Process for Electrical System Component Selection," SAE Technical Paper 2007-01-0778, 2007, doi:10.4271/2007-01-0778.
3. Jackey, R., Saginaw, M., Sanghvi, P., Gazzarri, J. et al., "Battery Model Parameter Estimation Using a Layered Technique: An Example Using a Lithium Iron Phosphate Cell," SAE Technical Paper 2013-01-1547, 2013, doi:10.4271/2013-01-1547
4. MathWorks Documentation - [www.mathworks.com](http://www.mathworks.com)