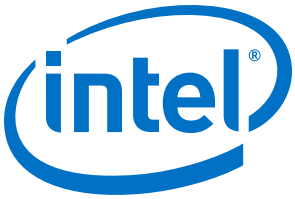


Battery Management System Reference Design

Subscribe
Send Feedback

AN-762 | 2016.04.02
Latest document on the web: PDF | HTML

Contents

1. Battery Management System Reference Design

The Altera® Battery Management System (BMS) Reference Design demonstrates battery state of charge (SOC) estimation in an FPGA-based real-time control platform that you can extend to include other BMS functionality such as battery state-of-health monitoring and charge equalization (cell balancing). It uses a dual extended Kalman filter (DEKF) algorithm to estimate SOC values for 96 cells, using a MAX® 10 development kit. The reference design's system-in-the-loop simulation runs on the MATLAB Simulink software.

A BMS is a critical component in high-value battery powered applications such as electric vehicles or energy storage. A BMS maintains the health of all the cells in the battery pack to deliver the power needed by the application. It also protects the cells from damage and maintains all the cells within the manufacturer-recommended operating conditions to prolong the life of the battery pack.

You can use an FPGA as a flexible and powerful platform for a BMS, using its high I/O count for parallel connections to many battery modules. An FPGA can accelerate processor-intensive calculations such as state-of-charge estimation.

Related Information

Improving Battery Management System Performance and Cost with Altera FPGAs

1.1. BMS Reference Design Features

- DEKF algorithm for SOC estimation and parameter identification.
- SOC value estimation for 96 cells.
- Alternative hardware implementations of SOC calculations:
  - Nios II processor with floating-point acceleration
  - Nios II with floating-point acceleration and floating-point matrix processor
  - Nios II processor and DEKF algorithm implemented in dedicated floating-point IP
- System-in-the-loop simulation runs a MATLAB Simulink model that communicates with FPGA hardware using Altera system console API.
- Compares the results from the FPGA in real-time with the results from the Simulink calculations
- Nios II processor for scheduling and communicating with MATLAB through System Console.
- Nios II software runs on µC/OS-II real-time operating system.

1.2. BMS Reference Design Getting Started

1.2.1. BMS Reference Design Software Requirements

- The Altera Complete Design Suite version 15.0, which includes:
  - The Quartus II software v15.0
  - DSP Builder v15.0
  - The Altera Nios II Embedded Design Suite (EDS) v15.0
- MATLAB R2015a

1.2.2. BMS Reference Design Hardware Requirements

The reference design requires Altera MAX 10 FPGA development kit (rev C).

Note: The reference design does not support the rev A or rev B board under default project settings, because the pinout of the rev C board is different from the other two.

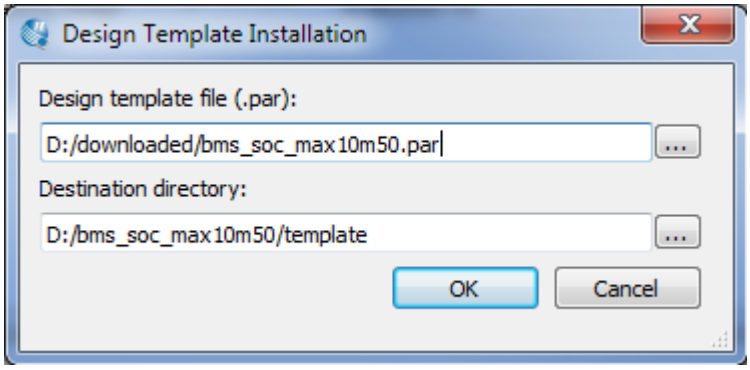
1.2.3. Downloading and Installing the BMS Reference Design

1. In the Altera Design Store, download the relevant reference design `bms_soc_max10m50.par` file for the MAX10 development kit:
   a. Select **Design Examples**, select **15.0** for the Quartus II version, then search for BMS.

   To obtain further support on the reference design, contact your local Altera sales representative.
2. In the Quartus II software, click **File ➤ New Project Wizard** .
3. Click **Next**.
4. Enter the path for project working directory and enter `bms_soc_max10m50` for the project name.
5. Click **Next**.
6. Select **Project Template**.
7. Click **Next**.
8. Click **Install the design templates**.

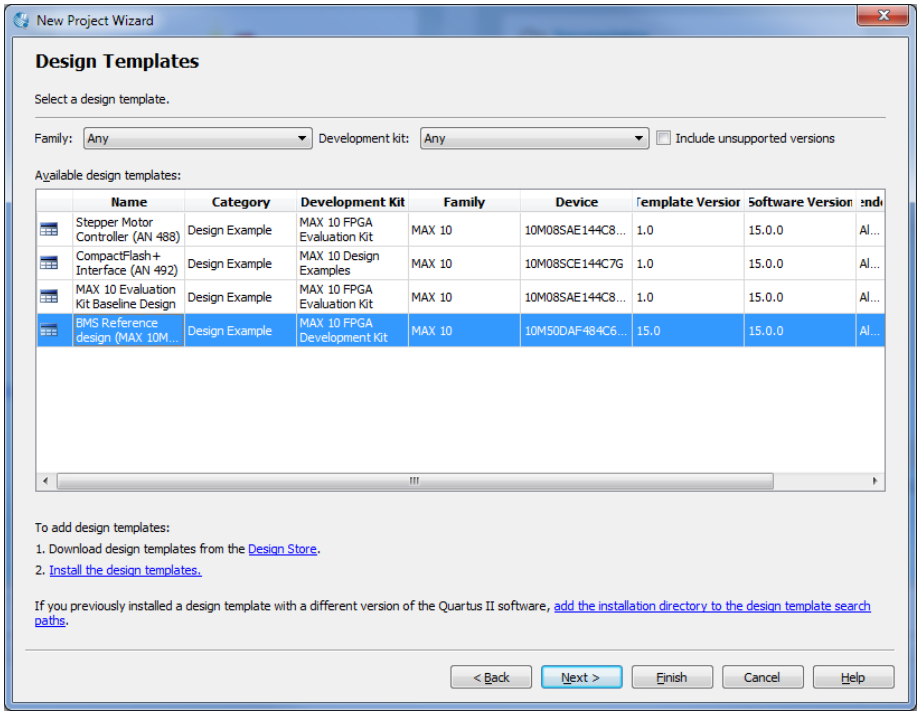
9. Browse to select the `bms_soc_max10m50.par` file for the reference design and browse to the destination directory where you want to install it.

Figure 1. Design Template Installation



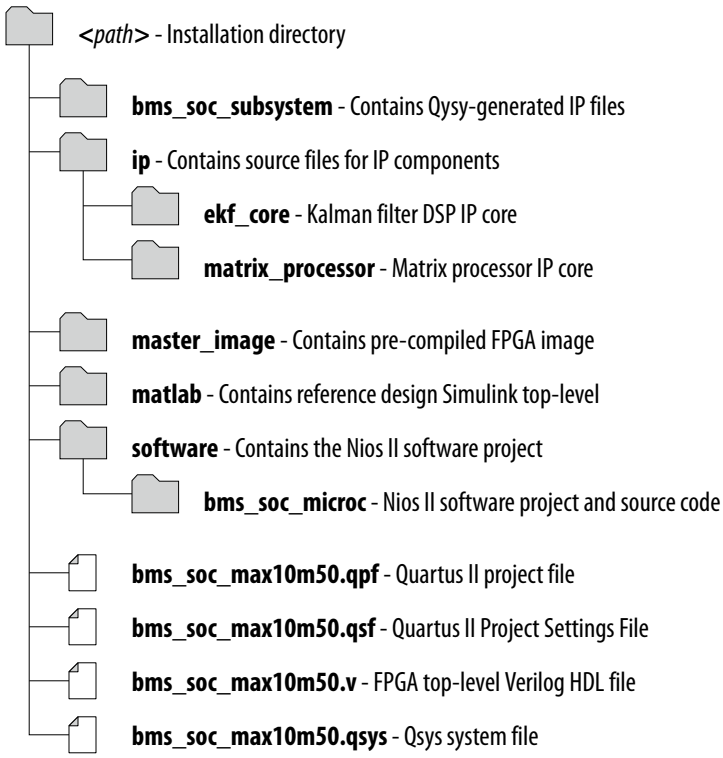
10. Click **OK** on the design template installation message.
11. Select the **BMS Reference design** design example.

Figure 2. Design Template



12. Click **Next**.
13. Click **Finish**.
The Quartus II software expands the archive and sets up the project, which may take some time.

Figure 3. Directory Structure



Related Information

Altera Design Store

1.2.4. Setting Up the MAX 10 Development Board

1. Connect the USB-Blaster II connector (J12) the development board to your computer using a USB cable.
2. Apply power to the development board.

Related Information

MAX 10 FPGA Development Kit User Guide
The FPGA development board provides a hardware platform for evaluating the performance and features of the Intel® MAX 10 device.

1.2.5. Compiling the FPGA Hardware Design for the BMS Reference Design

You can compile your design or use the Altera-provided pre-compiled `.sof` and `.pof` from the `/master_image` directory of your reference design

1. Launch Quartus II software.
2. Open project `bms_soc_max10m50.qpf`.
3. Click **Processing ➤ Start Compilation**.

   Note: You may edit the reference design project in Qsys.

1.2.6. Compiling the Nios Software for the BMS Reference Design

You can either compile your design or use the Altera-provided pre-compiled `.elf` from the `software/ bms_soc_microc` directory of your reference design.

1. Start Nios II EDS by clicking**Start ➤ Altera ➤ Nios II EDS ➤ Nios II Software Build Tools**
2. Specify the `\software` folder as the workspace by browsing to the reference design `/software` directory.
3. Click **OK** to create the workspace.
4. Import application and board support package (BSP) projects:
   a. Click **File ➤ Import**.
   b. Expand **General** and click **Existing Projects into Workspace**.
   c. Click **Next**.
   d. Browse to `\software\bms_soc_microc` and click **OK**.
   e. Click **Finish**.
   f. Repeat steps a4.a on page 7. to e4.a on page 7. for `bms_soc_microc_bsp`.
5. Rebuild the BSP project:
   a. Right click `bms_soc_microc_bsp` project
   b. Point to Nios II
   c. Click **Generate BSP**
6. Build the application project:
   a. Right-click `bms_soc_microc` project
   b. Click **Build Project**.

   Note: On Windows operating systems, the first time you build, the project might take up to one hour.
7. Note: build and rebuild the project if you make any changes to the Qsys project.

1.2.7. Programming the BMS Reference Design Hardware onto the Device

1. In the Quartus II software, click **Tools ➤ Programmer**.
2. In the **Programmer** pane, select **USB-Blaster II** under **Hardware Setup** and **JTAG** under **Mode**.
3. Click **Auto Detect** to detect devices.
4. Select any **10M50** device.
5. Right click **10M50** device, select **Edit**, and click **Change file**.
6. Select the `output_files/<project_name>.sof` or `output_files/<project name>.pof` and click Open.
7. Turn on **Program/Configure**.
8. Click **Start**.

Note: If you program using `.sof` file, reprogram the device after removing power from the board. If you program using `.pof` file, on-chip flash keeps the image, and you do not need to program the device after removing power from the board. It takes longer to program the device initially using the `.pof` file.

1.2.8. Downloading the BMS Reference Design Nios II Software to the Device

1. Start Nios II EDS, by clicking **Start ➤ Altera ➤ Nios II EDS ➤ Nios II Command Shell**
2. In the command shell, go to the software project folder by entering:`cd <reference_design_path>/software/bms_soc_microc`
3. In the command shell, download the software by entering: `nios2-download -r –g bms_soc_microc.elf`
   When the downloadis successful, you see the following message in the terminal:

```
Using cable “USB-BlasterII on <computername> [USB-1]”, device 1, instance
0x00
Resetting and pausing target processor: OK
Initializing CPU cache <if present>
OK
Downloaded 65KB in 0.0s
Verified OK
Starting Processor at address 0x00000190
```

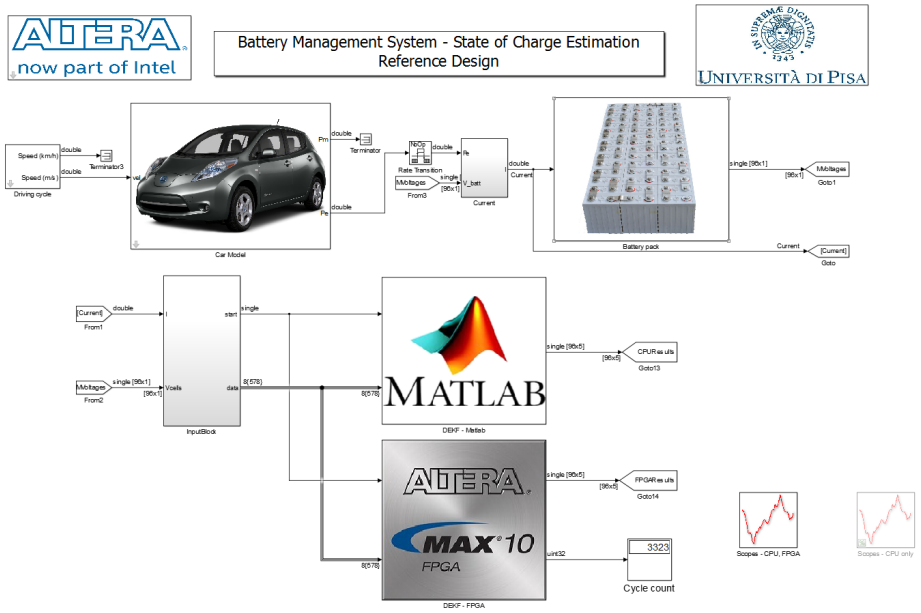
4. In the Nios II EDS, click **Run ➤ Run configurations...** .
5. Double click **Nios II Hardware** to generate a new run configuration.
6. Click **New_configuration**.
7. On the **Project** tab select the **bms_soc_microc** project in **Project name**.
8. Turn on **Enable browse for file system ELF file**.
9. Browse to the `software\ bms_soc_microc` and select `bms_soc_microc.elf`.
10. On the **Target Connection** tab, click **Refresh Connections**.
    The software finds the USB-Blaster cable.
11. Click **Apply** to save changes, optionally specifying a name for the new configuration.
12. Click **Run** to start the software.
13. Check that the terminal console display shows the message:

```
Start hardware interaction
```

1.2.9. MATLAB Simulink Top-Level Design for the BMS Reference Design

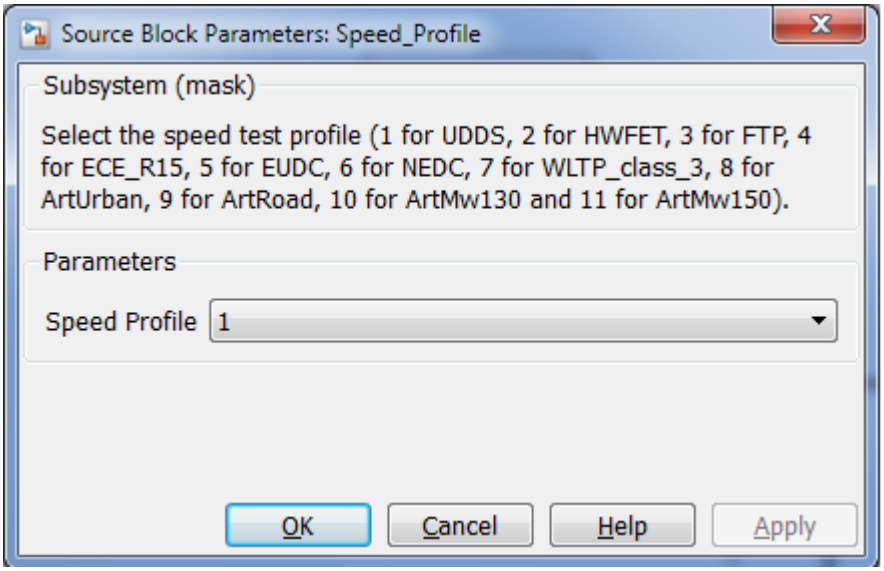
The reference design uses the Simulink model `demo_top.slx`, which calls `setup_demo_top.m` file to initialize all parameters.

Figure 4. Simulink Top-Level Design



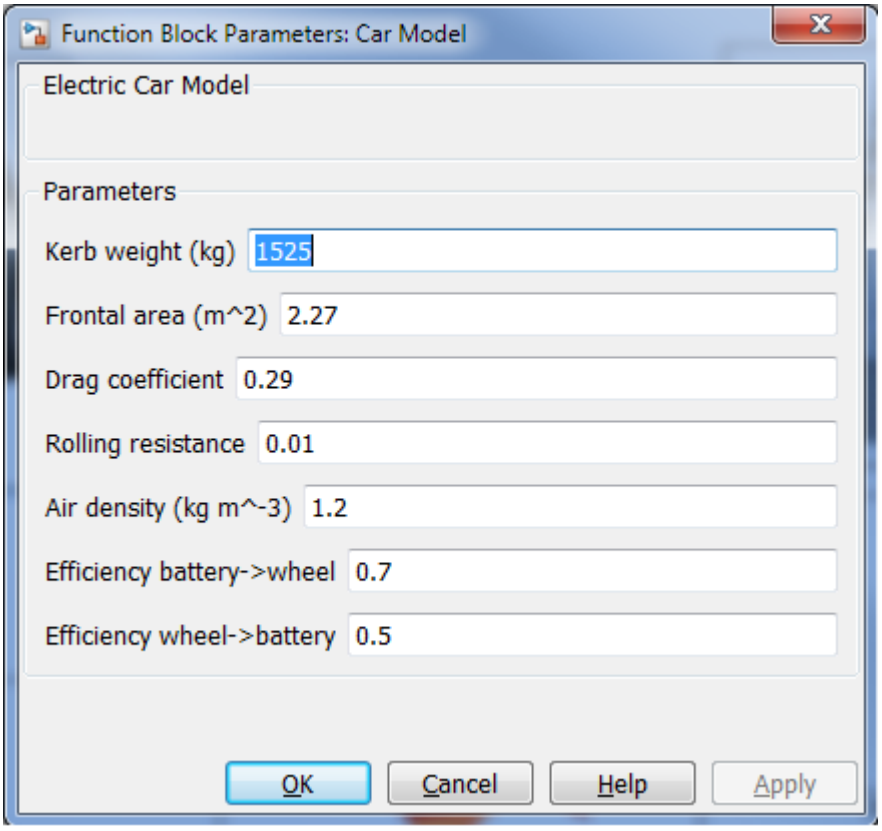
The driving cycle block allows you to select the speed profile.

Figure 5. Driving Cycle



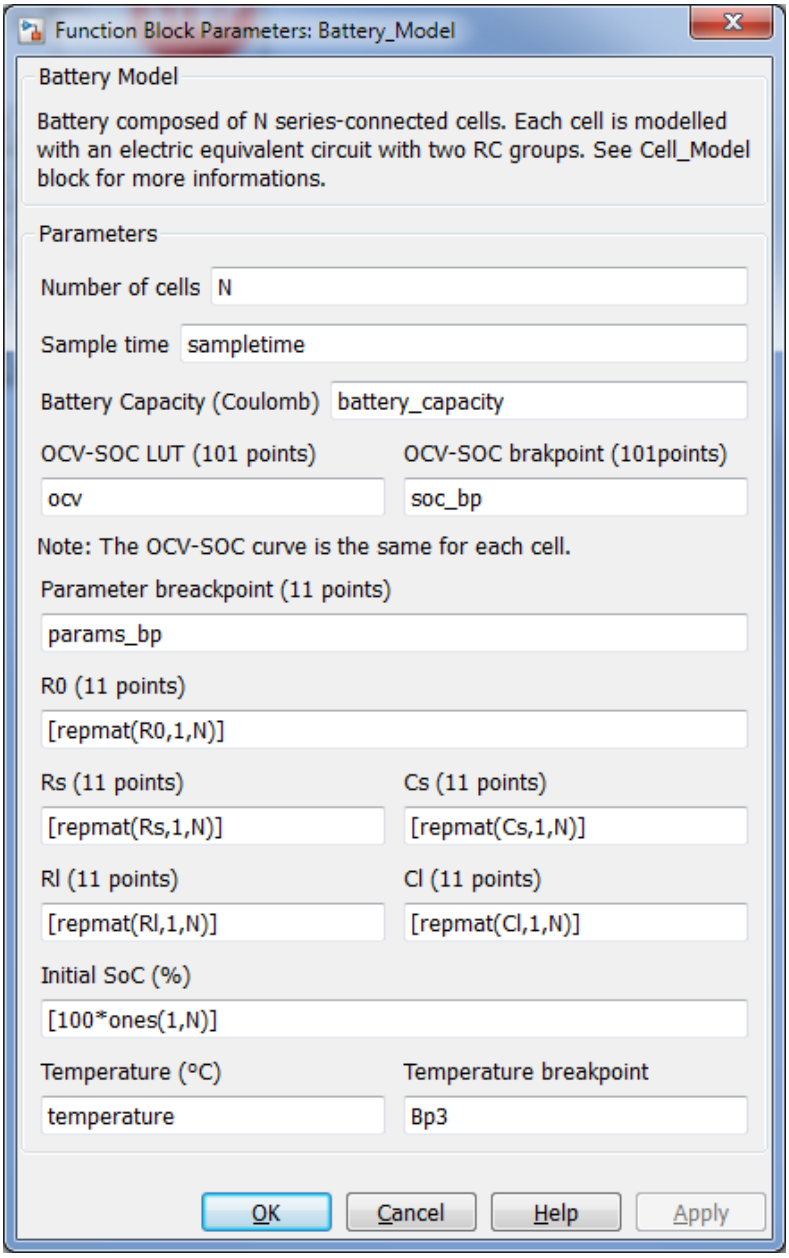
The car model block allows you to select the car model parameters.

Figure 6. Car Model



The battery pack contains the battery model block, which allows you to select the battery parameters.

Figure 7. Battery Model



The DEKF-FPGA block communicates with the FPGA. It sends input, including initial values, voltage and current data, to the FPGA, and receives SOC estimation, and updated battery model parameters from the FPGA. Three scope windows, including cell 1 and 2 information, and SOC value for the first 12 cells, appear by default when you open the Simulink top level design. You can add more scopes in the floating scope block.

Figure 8. Cell Information

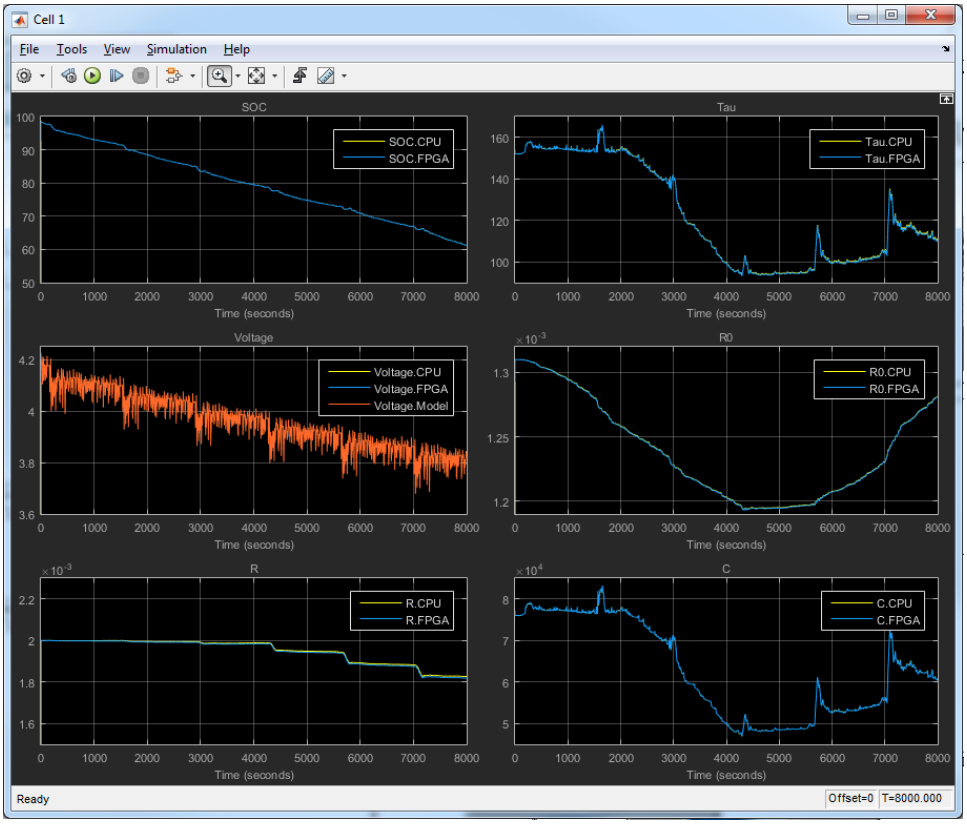
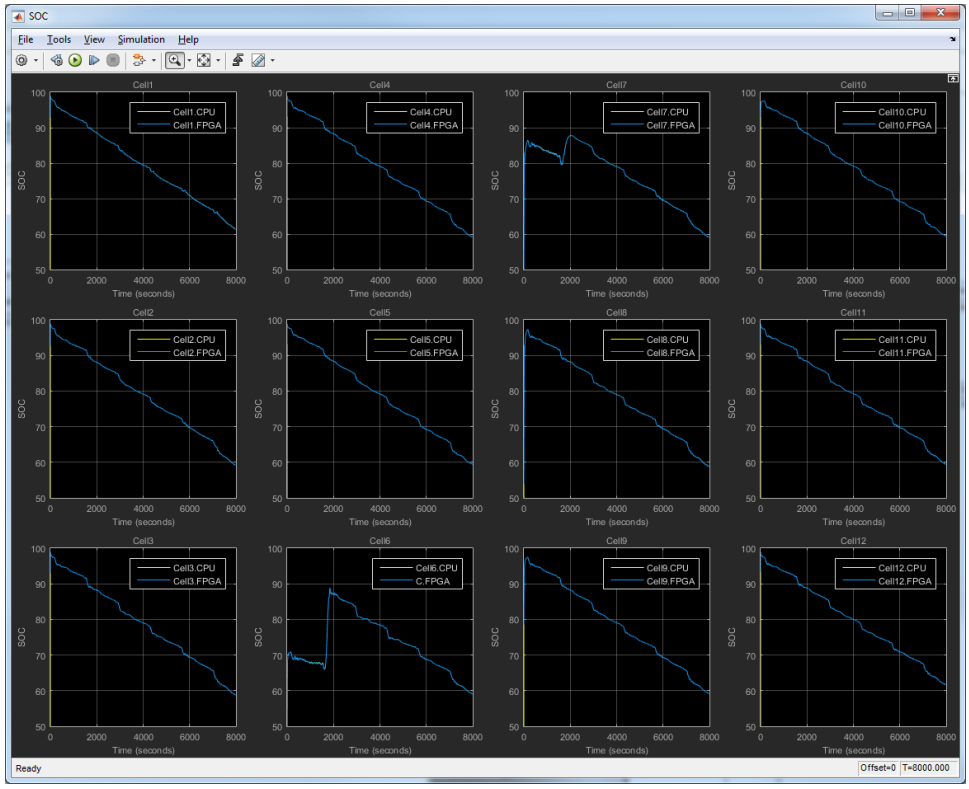


Figure 9. SOC values for 12 Cells



1.2.10. Running the BMS Reference Design in a System-in-the-Loop Simulation

1. Start DSP Builder in MATLAB (see related links).
2. In MATLAB, change the working directory to *<reference_design_path>*`\matlab\bms_soc_application`
3. Open `demo_top.slx`
4. Check that you configured, programmed, and started to run the software on the MAX 10 FPGA.
5. In Simulink, click **Play**.

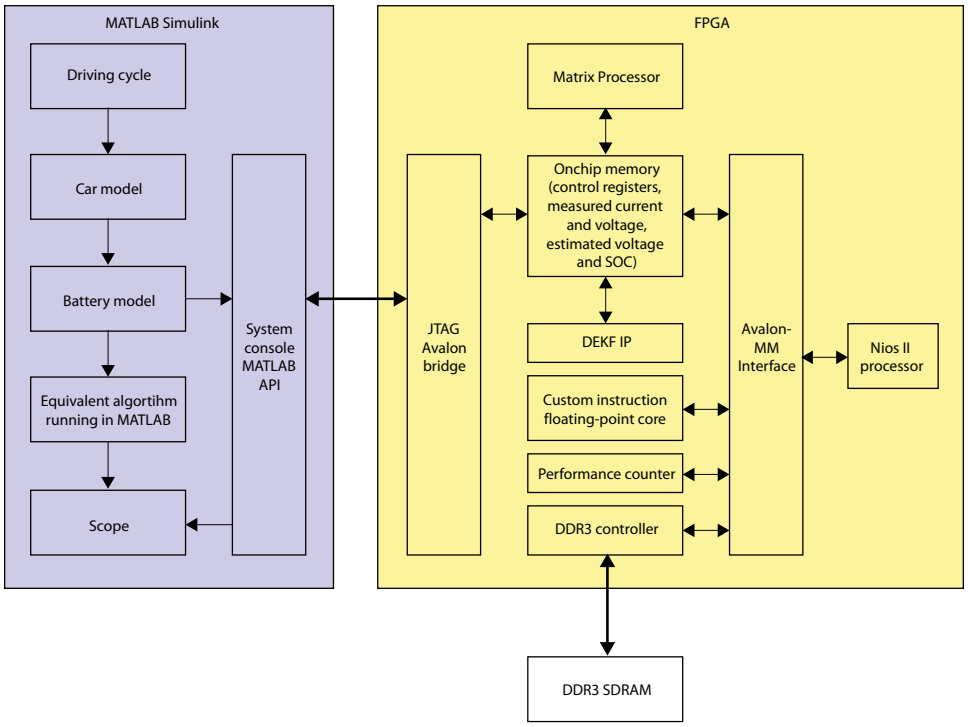
Related Information

Starting DSP Builder in MATLAB

1.3. BMS Reference Design Functional Description

Figure 10. Block Diagram

Note: The design only uses DDR3 SDRAM for Nios II processor storage. You can replace it with external flash or another memory block.



1.3.1. BMS Reference Design Car Model

The car model computes the electric power at the battery's terminals, so that the vehicle speed follows a driving cycle. You can select from 11 standard driving cycles.

The Urban Dynamometer Driving Schedule (UDDS), the Highway Fuel Economy Test (HWFET) and the Federal Test Procedure (FTP) are defined by the U.S. Environmental Protection Agency. The New European Driving Cycle (NEDC), the Extra-Urban Driving Cycle (EUDC) and the Economic Commission for Europe urban driving cycle (ECE R15) are maintained by the United Nations Economic Commission for Europe (UNECE). The Common Artemis Driving Cycles consists of the Urban cycle (ArtUrban), the Rural road cycle (ArtRoad) and the Motorway cycles (ArtMw130 and ArtMw150, with a maximum speed of 130 and 150 km/h, respectively). The Worldwide harmonized Light vehicles Test Procedures (WLTP) Class 3 is developed the following the guidelines of UNECE World Forum for Harmonization of Vehicle Regulations. The various cycles differ in the average speed and electric power required from the traction battery.

Table 1. Driving Cycle Details

The duration, Distance and average speed of each cycle

Driving Cycle	Duration (min)	Distance (km)	Average speed (km/h)
UDDS	23	12.0	31.5
HWFET	13	16.5	77.5
FTP	31	17.8	34.1
EUDC	7	6.5	58.6
NEDC	20	8.3	25.4
ECE R15	3	0.9	16.5
WLTP class 3	30	23.2	46.5
ArtUrban	17	4.9	17.6
ArtRoad	18	17.3	57.4
ArtMw130	18	28.7	96.8
ArtMw150	18	29.5	99.5

The reference design implements a dynamic model to simulate the behaviour of a car. You calculate the mechanical power *Pm* as the sum of three contributions: one linked to the acceleration, one because of the air resistance and another because of the rolling resistance:

Figure 11. Car Model Equation

$$P_m = Fv = (Mv + \frac{1}{2}\rho_{air} SC_x v^2 + \alpha_R Mg)v$$

where:

- M is the kerb weight
- S is the frontal area
- C_X is the drag coefficient
- R is the rolling resistance
- ρ_{air} is the air density
- g is the gravity acceleration
- v is the speed.

You obtain the electric power P_e from P_m where:

- η_{wheel} is the efficiency from the battery to the wheels
- η_{reg} is the efficiency in the opposite direction, i.e., during the regenerative breaking.

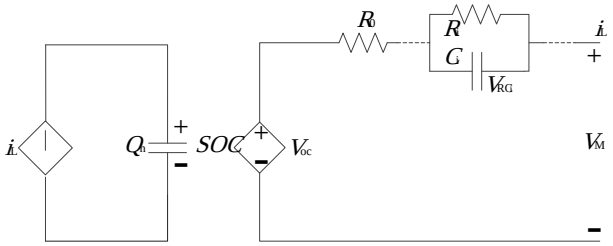
To obtain the battery current, divide the electric power by the sum of the cell voltages calculated by the battery model.

1.3.2. BMS Reference Design Battery Model

The battery model can simulate a number of series-connected cells.

The only input is the battery current, which is the same for all the series-connected cells. For the output, at each time step, the model generates the arrays of the cell voltages V_i, *SOC*, and the current values of the model parameters.

Figure 12. Battery cell equivalent circuit model



The left hand side models the cell capacity Q_n and evaluates the *SOC* as the voltage across a linear capacitor with a capacity equal to Q_n (expressed in Coulomb) divided by 1V. The cell voltage v_M is the sum of the open-circuit voltage V_{OC} and a dynamic term, which accounts for the internal Ohmic resistance R_0 and the double layer effect (V_{RC1}) and diffusion (V_{RC2}) of the Li-ion during charging and discharging (two RC branches). The model parameters change with manufacturing variations, ageing and operating conditions, such as temperature and state of charge. To model the temperature and SOC dependences, the reference design stores the parameter values in two-dimensional LUTs. You see the variability of the cell behaviour by setting the model parameters, temperature, and capacity of each cell individually

1.3.3. DEKF Technique

In the DEKF technique, the reference design simultaneously executes two cooperating Kalman filters for nonlinear systems: one for the state and the other for the parameters estimation.

Dual estimation, $x_k = \begin{bmatrix} SOC_k \\ V_{RC_k} \end{bmatrix}$ rather than joint estimation, with only one Kalman filter reduces the state matrix dimensions and may improve the estimation robustness.

Equation 1. Equation 1

Equation 1 describes the parameter evolution that with the measurement equation builds the first EKF.

$$p(k+1) = p(k) + \chi(k)$$

Equation 2. Equation 2

Equation 2 represents the state evolution that combines with the measurement equation to form the second EKF.

$$x(k+1) = F\left(x(k), i_L(k), p(k)\right) + \xi(k)$$

Equation 3. Measurement Equation

$$v_T(k) = G\left(x(k), i_L(k), p(k)\right) + \psi(k)$$

The measurement equation is the same for both filters. In the above equations:

- *k* is the discrete time
- *p* is parameters vector
- x = [SOC; V_{RC1}] is the battery state vector
- χ, ξ and ψ are the parameters, the state and measurement noise, with zero mean and covariance matrix Σχ, Σξ and Σψ, respectively.

Equation 4. Circuit Equations

The circuit equation describes the actual circuit.

$$x_k = f\left(SOC_k, V_{RC_k}, i_{L_k}\right) = \begin{bmatrix} SOC_k \\ V_{RC_k} \end{bmatrix} = \begin{bmatrix} SOC_{k-1} - \frac{T}{Q_r} i_{L_k} \\ v_{RC_{k-1}} e^{-T/\tau} + R\left(1 - e^{-T/\tau}\right) i_{L_k} \end{bmatrix}$$

$$v_{T_k} = g\left(SOC_k, V_{RC_k}, i_{L_k}\right) = OCV\left(SOC_k\right) - R_0 i_{L_k} - v_{RC_k}$$

$$OCV(SOC) = P_1 SOC^7 + P_2 SOC^6 + P_3 SOC^5 + P_4 SOC^4 + P_5 SOC^3 + P_6 SOC^2 + P_7 SOC^1 + P_8$$

Equation 5. DEKF Matrix Equations

The matrix equations are derived from the circuit equations.

$$x_k = \begin{bmatrix} SOC_k \\ V_{RC_k} \end{bmatrix}$$

$$q_k = \begin{bmatrix} SOC_k \\ 1/\tau \\ R_k \end{bmatrix}$$

$$A_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{-T/\tau} \end{bmatrix}$$

$$C_{x_k} = \left[\frac{dOCV(SOC)}{dSOC} \quad -1\right]$$

$$C_{q_k} = \left[i_{L_k} \quad 0 \quad 0\right] + C_{x_k} \frac{dx_k^-}{dq}$$

$$\frac{dx_k^-}{dq} = \begin{bmatrix} 1 & 0 \\ 0 & e^{-T/\tau} \end{bmatrix} \frac{dx_{k-1}^+}{dq} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & Te^{-\frac{T}{\tau}} & \left(Ri_{L_k} - V_{RC_{k-1}}\right)\left(1 - e^{-T/\tau} i_{L_k}\right) \end{bmatrix}$$

$$\frac{dx_{k-1}^+}{dq} = \frac{dx_k^-}{dq} - L_{x_{k-1}} C_{q_{k-1}}$$

Equation 6. Initialization Equation

$$x_0, P_0, q_0, P_{q_0}$$

Equation 7. Prediction Equations

$$q_k^- = q_{k-1}^+$$

$$P_{q_k}^- = P_{q_{k-1}}^+ + Q_q$$

$$x_k^- = f\left(x_{k-1}^+, u_{k-1}, q_{k-1}^+\right)$$

$$P_k^- = A_k P_{k-1}^+ A_k^T + Q$$

Equation 8. Correction Equation

$$L_{x_k} = P_k^- C_{x_k}^T \left(C_{x_k} P_k^- C_{x_k}^T + R\right)^{-1}$$

$$x_k^+ = x_k^- + L_{x_k}\left(y_k - g\left(x_k^-, u_k, q_{k-1}^+\right)\right)$$

$$P_k^- = \left(I - L_{x_k} C_{x_k}\right) P_k^-$$

$$L_{q_k} = P_{q_k}^- C_{q_k}^T \left(C_{q_k} P_{q_k}^- C_{q_k}^T + R\right)^{-1}$$

$$q_k^+ = q_k^+ + L_{q_k}\left(y_k - g\left(x_k^-, u_k, q_{k-1}^+\right)\right)$$

$$P_{q_k}^- = \left(I - L_{q_k} C_{q_k}\right) P_{q_k}^-$$

Related Information

R. Morello et al., "Comparison of state and parameter estimators for electric vehicle batteries," Industrial Electronics Society, IECON 2015 - 41st Annual Conference of the IEEE, Yokohama, 2015, pp. 005433-005438

1.3.4. BMS Reference Design Hardware Implementation

The reference design includes three different designs to implement the DEKF algorithm:

- Design A,which has a Nios II processor with floating-point acceleration
- Design B, which has a Nios II processor with floating-point acceleration and floating-point matrix processor
- Design C, which has a Nios II processor and DEKF algorithm implemented in dedicated floating-point IP

The reference design creates the dedicated floating-point IP using Altera's DSP Builder advanced blockset. In each design, every functional component takes charge of different tasks, including system-in-the-loop communication with MATLAB Simulink, cell link list management, DEKF calculation. In the three designs, the Nios processor II controls the system-in-the-loop and cell link tasks. In design B, both the Nios II

processor and matrix processor perform the DEKF calculation, and the matrix processor processes most of the matrix calculations. Finally, in design C, DSP Builder IP processes all DEKF calculations.

To switch between different implementation methods, modify this line in source file

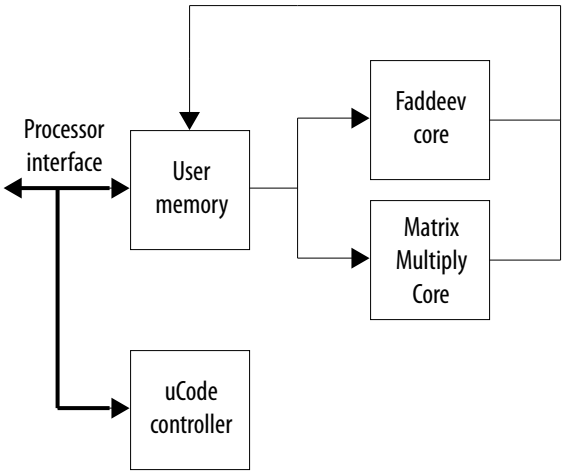
```
 \software
\bms_soc_microc
\soc_kalman.h:
// 0 - Nios2 only
// 1 - Matrix processor
// 2 - DSP Builder IP
#define ACC 1
```

1.3.4.1. BMS Reference Design Matrix Processor

The matrix processor is a generic matrix processor that Altera developed using the DSP Builder advanced blockset. It can perform sequences of different matrix operations.

You can select the maximum size of matrices to use to scale the usage of internal memory to fit the desired application. The matrix processor includes two data processing cores: Faddeev and matrix multiply cores. The Faddeev core can calculate the operation: $D + C * A^{-1} * B$. The matrix multiply core can calculate the ($A * B$) and ($A * B + C$) matrix expressions.

Figure 13. Matrix Processor Block Diagram



The matrix processor interface is the main interface between the matrix processor and the external environment. It programs the matrix processor for certain uCode, provides input matrix argumentss and reads-back results.

1.3.4.2. BMS Reference Design uCode Controller Interface

The uCode controller interface configures and controls the matrix processor.

The reference design maps it to an address region in the processor interface. The address map may change based on certain sizes and user -defined parameters of the design.

Table 2. Matrix Processor Control Registers Map

Name	Address	Bits	Description
MAT_PROC_OPCODESTART	0x320	[4:0]	Line of opCode at which the matrix processor starts uCode execution.
MAT_PROC_OPCODESTOP	0x321	[4:0]	Line of opCode at which the matrix processor stops uCode execution.
MAT_PROC_READY	0x3B3	[0:0]	Bit that indicates status of the matrix processor. The host has to test this bit before starting new uCode sequence or to poll for current sequence completion
MAT_PROC_GO	0x3B4	[0:0]	Bit asserted by host to initiate a uCode sequence execution. Processor has to assert the bit each time to start new execution.
MAT_PROC_HOSTACCESS	0x3B5	[0:0]	Use this bit to select the user memory multiplex to obtain and release user memory. Assert to obtain access from processor. Deassert to release control to matrix processor.

In addition, to control registers, the uCode interface has a page with a uCode program stored in internal memory. To configure the matrix pProcessor with your uCode program, fill values into the uCode memory. The depth of the uCode memory is a compile-time parameter.

Table 3. uCode Program Memory Map

Name	Address Line 0	Bits	Description
MAT_PROC_OPCODE	0x323	[3:0]	opCode to execute.
MAT_PROC_N	0x333	[3:0]	*N* size parameter of the opCode.
MAT_PROC_M	0x343	[3:0]	*M* size parameter of the opCode.
MAT_PROC_W	0x353	[3:0]	*W* size parameter of the opCode.
MAT_PROC_ARG1	0x363	[31:0]	*Arg1* parameter of the opCode Bits [8:0] Argument1 offset Bits[31:28] Argument 1 extension
MAT_PROC_ARG2	0x373	[31:0]	*Arg2* parameter of the opCode Bits [8:0] Argument2 offset Bits[31:28] Argument 4 extension
MAT_PROC_ARG3	0x383	[31:0]	*Arg3* parameter of the opCode Bits [8:0] Argument3 offset Bits[31:28] Argument 4 extension
MAT_PROC_ARG4	0x393	[31:0]	*Arg4* parameter of the opCode Bits [8:0] Argument4 offset Bits[31:28] Argument 4 extension
MAT_PROC_ARG5	0x3A3	[31:0]	*Arg5* parameter of the opCode Bits [8:0] Argument5 offset

To configure the matrix processor to execute an algorithm, the CPU host must program it. To program, fill data into the uCode program area and provide run-time configuration parameters.

Table 4. Matrix Processor opCode Arguments

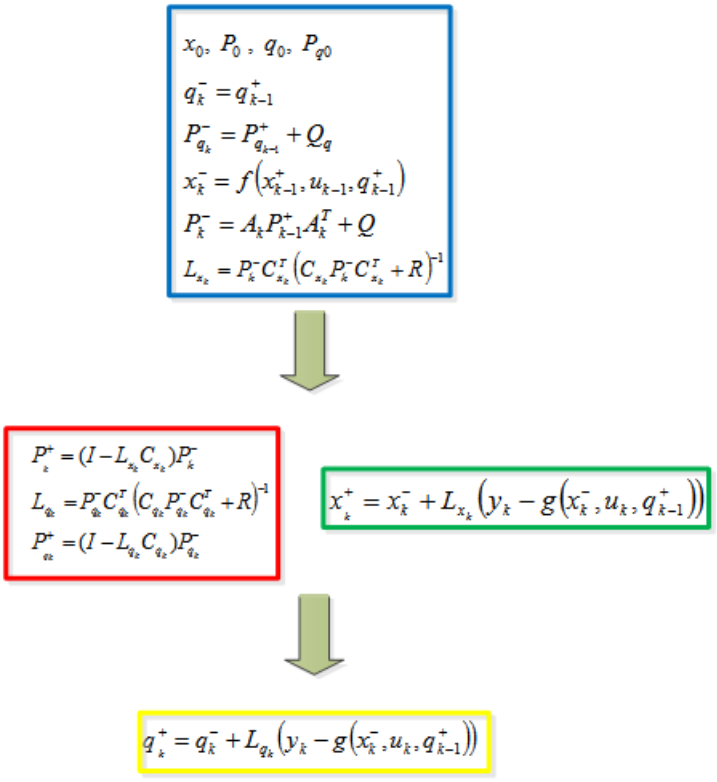
The uCode program structure has these fields that you must supply to the uCode controller. The fields are opCode dependent. The matrix processor has three operation modes. .

Name	Valid Entries	Description
MAT_PROC_OPCODE	1, 3, 4	Operation mode: 1 – D + C * A-1 * B 3 – A * B 4 – A * B + C
MAT_PROC_N	2-MAX	*N* size parameter of the opCode *MAX* is compile time parameter
MAT_PROC_M	1-MAX	*M* size parameter of the opCode *MAX* is compile time parameter
MAT_PROC_W	1-MAX	*W* size parameter of the opCode *MAX* is compile time parameter
MAT_PROC_ARG1 – Matrix A MAT_PROC_ARG2 – Matrix B MAT_PROC_ARG3 – Matrix C MAT_PROC_ARG4 – Matrix D	XXX	*Arg1* to *Arg4* parameter of the opCode Bits [8:0] Argument offset Bits[31:28] Argument extension [30:28] Valid when opCode = 1 0 – Normal (no data manipulation) 1 – Negate elements 2 – Zero elements Matrix 3 – Identity matrix 4 – Negative identity matrix [31] 0 – Do not transpose matrix 1 – Transpose matrix
MAT_PROC_ARG5	XXX	Result matrix *Arg5* parameter of the opCode Bits [8:0] Argument5 offset

In the matrix processor solution, the reference design accelerates part of the matrix operation using the matrix processor. Meanwhile, the Nios II processor can also be doing calculations.

Figure 14. Task Scheduling

The Nios II processor calculates the steps in the blue box. Then the matrix processor starts calculating the equations in the red box. Meanwhile, the Nios II processor can process the steps in the green box. Finally, the Nios II processor finishes the calculation in the yellow box.



1.3.4.3. BMS Reference Design DEKF IP (Design C)

The DEKF IP includes all calculations in the DEKF.

The Nios II processor only communicates with the host, sends inputs, and receives results. The reference design includes the model file `Ekf_Core.slx`. A DSP Builder-generated direct implementation of the algorithm in the FPGA fabric executes quickly but uses too many FPGA resources for a low-cost FPGA device. However, the DSP Builder ALU folding automatically generates a design which reuses FPGA logic.The ALU folder saves around 90% of the FPGA resources in this design.

1.4. BMS Reference Design FPGA Resource Usage

Use the resource usage to estimate the size of your design.

Table 5. Resource Usage

Component	LE	M9K	DSP
DEKF IP (design C only)	20,610	30	29
Matrix processor (design B only)	6,304	35	24

continued...

Component	LE	M9K	DSP
Nios II processor	2,916	32	6
Nios II floating-point custom instruction core	2,204	3	9
DDR controller	4,785	12	0
Avalon-MM interconnect	5,897	0	0
JTAG-Avalon-MM bridge	799	1	0
Miscellaneous	1,292	23	0
Design A total	~14,000	57	15
Design B total	~24,000	92	39
Design C total	~33,000	84	35
Total	~45,000/50,000	136/182	68

1.5. BMS Reference Design Benchmarking

Table 6. Execution Time for Updating One Cell

Implementation Method	Time to Update One Cell (µs)
Design A	44.9
Design B	33.8
Design C	16.5

1.6. Acknowledgements for the BMS Reference Design

Altera wants to acknowledge the help of Federico Baronti of the University of Pisa with the models and algorithms used in this reference design.