Battery Management System Reference Design

1. Battery Management System Reference Design
   1.1. BMS Reference Design Features
   1.2. BMS Reference Design Getting Started
      1.2.1. BMS Reference Design Software Requirements
      1.2.2. BMS Reference Design Hardware Requirements
      1.2.3. Downloading and Installing the BMS Reference Design
      1.2.4. Setting Up the MAX 10 Development Board
      1.2.5. Compiling the FPGA Hardware Design for the BMS Reference Design
      1.2.6. Compiling the Nios Software for the BMS Reference Design
      1.2.7. Programming the BMS Reference Design Hardware onto the Device
      1.2.8. Downloading the BMS Reference Design Nios II Software to the Device
      1.2.9. MATLAB Simulink Top-Level Design for the BMS Reference Design
      1.2.10. Running the BMS Reference Design in a System-in-the-Loop Simulation
   1.3. BMS Reference Design Functional Description
      1.3.1. BMS Reference Design Car Model
      1.3.2. BMS Reference Design Battery Model
      1.3.3. BMS Reference Design DEKF Technique
      1.3.4. BMS Reference Design Hardware Implementation
   1.4. BMS Reference Design FPGA Resource Usage
   1.5. BMS Reference Design Benchmarking
   1.6. Acknowledgements for the BMS Reference Design
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

BMS Reference Design Software Requirements on page 4
BMS Reference Design Hardware Requirements on page 4
Downloading and Installing the BMS Reference Design on page 4
Setting Up the MAX 10 Development Board on page 6
Compiling the FPGA Hardware Design for the BMS Reference Design on page 6
Compiling the Nios Software for the BMS Reference Design on page 7
Programming the BMS Reference Design Hardware onto the Device on page 7
Downloading the BMS Reference Design Nios II Software to the Device on page 8
MATLAB Simulink Top-Level Design for the BMS Reference Design on page 8
Running the BMS Reference Design in a System-in-the-Loop Simulation on page 13

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

- `<path>` - Installation directory
  - `bms_soc_subsystem` - Contains Qsys-generated IP files
  - `ip` - Contains source files for IP components
    - `ekf_core` - Kalman filter DSP IP core
    - `matrix_processor` - Matrix processor IP core
  - `master_image` - Contains pre-compiled FPGA image
  - `matlab` - Contains reference design Simulink top-level
  - `software` - Contains the Nios II software project
    - `bms_soc_micro` - Nios II software project and source code
  - `bms_soc_max10m50.qpf` - Quartus II project file
  - `bms_soc_max10m50.qsf` - Quartus II Project Settings File
  - `bms_soc_max10m50.v` - FPGA top-level Verilog HDL file
  - `bms_soc_max10m50.qsys` - Qsys system file

Related Information
Alterna 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 Altera 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.
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 download is 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.
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

![Cell Information Diagram]

- SOC (State of Charge)
- Voltage
- Temperature
- Current

Detailed graphs showing variations over time for different parameters.
1. Battery Management System Reference Design

683279 | 2016.04.02

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 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

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

The reference design implements a dynamic model to simulate the behaviour of a car. You calculate the mechanical power \( P_m \) 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

\[
P_m = F_v = (Mv + \frac{1}{2} \rho_{\text{air}} SC x v^2 + \alpha R Mg) v
\]

where:
- \( M \) is the kerb weight
- \( S \) is the frontal area
- \( C_X \) is the drag coefficient
- \( R \) is the rolling resistance
- \( \rho_{\text{air}} \) is the air density
- \( g \) is the gravity acceleration
- \( v \) is the speed.

You obtain the electric power \( P_e \) from \( P_m \) where:
- \( \eta_{\text{wheel}} \) is the efficiency from the battery to the wheels
- \( \eta_{\text{reg}} \) 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 $V_i$, $SOC_i$, 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. BMS Reference Design 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, rather than joint estimation, with only one Kalman filter reduces the
state matrix dimensions and may improve the estimation robustness. Equation 2
describes the parameter evolution that with the measurement equation 4 builds the
first EKF. Equation 3 represents the state evolution that combines to the measurement
equation to form the second EKF.

**Figure 13. Equation 2**

\[ p(k+1) = p(k) + \chi(k) \]

**Figure 14. Equation 3**

\[ x(k+1) = F(x(k), i_L(k), p(k)) + \xi(k) \]

**Figure 15. Equation 4**

\[ v_T(k) = G(x(k), i_L(k), p(k)) + \psi(k) \]

**Figure 16. Measurement Equation**

\[ v_T(k) = G(x(k), i_L(k), p(k)) + \psi(k) \]
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
- \( \chi, \xi \) and \( \psi \) are the parameters, the state and measurement noise, with zero mean and covariance matrix \( \Sigma_{\chi}, \Sigma_{\xi} \) and \( \Sigma_{\psi} \), respectively.

Figure 17. Circuit Equations

\[
x_k = f(SoC_k, v_{RCk}, i_{Lk}) = \frac{SoC_k}{v_{RC_k}} = \frac{SoC_{k-1} - \frac{T}{Q_r} i_{Lk}}{v_{RC_{k-1}}} e^{\frac{T}{\tau}} + R(1 - e^{\frac{T}{\tau}}) i_{Lk}
\]

\[
v_{Ik} = g(SoC_k, v_{RCk}, i_{Lk}) = OCV(SoC_k) - R_i i_{Lk} - v_{RCk}
\]

\[OCV(SoC) = P_1 SoC^3 + P_2 SoC^2 + P_3 SoC + P_4 SoC^4 + P_5 SoC^5 + P_6 SoC^6 + P_7 SoC^7 + P_8\]

Figure 18. DEFK Matrix Equations

\[
x_k = f(SoC_k, v_{RCk}, i_{Lk}) = \frac{SoC_k}{v_{RC_k}} = \frac{SoC_{k-1} - \frac{T}{Q_r} i_{Lk}}{v_{RC_{k-1}}} e^{\frac{T}{\tau}} + R(1 - e^{\frac{T}{\tau}}) i_{Lk}
\]

\[
v_{Ik} = g(SoC_k, v_{RCk}, i_{Lk}) = OCV(SoC_k) - R_i i_{Lk} - v_{RCk}
\]

\[OCV(SoC) = P_1 SoC^3 + P_2 SoC^2 + P_3 SoC + P_4 SoC^4 + P_5 SoC^5 + P_6 SoC^6 + P_7 SoC^7 + P_8\]
Figure 19. Initialization Equation

\[ x_0, P_0, q_0, P_{q0} \]

Figure 20. Prediction Equations

\[ q_k^* = q_{k-1}^* \\
\] 
\[ P_{q_k}^* = P_{q_{k-1}}^* + Q \\n\] 
\[ x_k = f(x_{k-1}, u_{k-1}, q_{k-1}^*) \\n\] 
\[ P_k = A_k P_{k-1} A_k^T + Q \]

Figure 21. Correction Equation

\[ L_{x_k} = P_k^T C_{x_k} \left( C_{x_k} P_k^T C_{x_k}^T + R \right)^{-1} \\
\] 
\[ x_k^* = x_k + L_{x_k} \left( y_k - g(x_k, u_k, q_{k-1}^*) \right) \\n\] 
\[ P_k^* = (I - L_{x_k} C_{x_k}) P_k \\n\] 
\[ L_{q_k} = P_k^T C_{q_k} \left( C_{q_k} P_k^T C_{q_k}^T + R \right)^{-1} \\
\] 
\[ q_k^* = q_k + L_{q_k} \left( y_k - g(x_k, u_k, q_{k-1}^*) \right) \\n\] 
\[ P_{q_k}^* = (I - L_{q_k} C_{q_k}) P_{q_k} \]

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: \( D + C \times A^{-1} \times B \). The matrix multiply core can calculate the \((A \times B)\) and \((A \times B + C)\) matrix expressions.

Figure 22. 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 arguments 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAT_PROC_OPCODESTART</td>
<td>0x320</td>
<td>[4:0]</td>
<td>Line of opCode at which the matrix processor starts uCode execution.</td>
</tr>
<tr>
<td>MAT_PROC_OPCODESTOP</td>
<td>0x321</td>
<td>[4:0]</td>
<td>Line of opCode at which the matrix processor stops uCode execution.</td>
</tr>
<tr>
<td>MAT_PROC_READY</td>
<td>0x3B3</td>
<td>[0:0]</td>
<td>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</td>
</tr>
<tr>
<td>MAT_PROC_GO</td>
<td>0x3B4</td>
<td>[0:0]</td>
<td>Bit asserted by host to initiate a uCode sequence execution. Processor has to assert the bit each time to start new execution.</td>
</tr>
<tr>
<td>MAT_PROC_HOSTACCESS</td>
<td>0x3B5</td>
<td>[0:0]</td>
<td>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.</td>
</tr>
</tbody>
</table>

In addition, to control registers, the uCode interface has a page with a uCode program stored in internal memory. To configure the matrix processor 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAT_PROC_OPCODE</td>
<td>0x323</td>
<td>[3:0]</td>
<td>.opCode to execute.</td>
</tr>
<tr>
<td>MAT_PROC_ARG5</td>
<td>0x3A3</td>
<td>[31:0]</td>
<td>Arg5 parameter of the OpCode Bits [8:0] Argument5 offset</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Valid Entries</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 | XXX             | Arg1 to Arg4 parameter of the opCode  
|                    |               | Bits [8:0] Argument offset                   |
| MAT_PROC_ARG2 – Matrix B | XXX             | Bits[31:28] Argument extension               |
| MAT_PROC_ARG3 – Matrix C | XXX             | [30:28] Valid when opCode = 1  
|                    |               | 0 – Normal (no data manipulation)            |
| MAT_PROC_ARG4 – Matrix D | XXX             | 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] Argument 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.
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.

$$
\begin{align*}
  x_k, P_0, q_0, P_0^e &= 0, \quad P_1^e = P_0^e + Q,
  x_1 = f(x_{k-1}, u_{k-1}, q_{k-1}), \\
  P_1^e &= A_k P_1^e A_k^T + Q,
  L_{n_i} &= P_C C_n^T (C_n P_C C_n^T + R)^{-1}
\end{align*}
$$

$$
\begin{align*}
  P_k^e &= (I - L_n C_n) P_k^e C_n^T (C_n P_C C_n^T + R)^{-1}, \\
  L_n &= P_C C_n^T (C_n P_C C_n^T + R)^{-1} \\
  P_{k+1}^e &= (I - L_n C_n) P_k^e \\
  x_{k+1} &= x_k + L_n (y_k - g(x_k, u_k, q_k)) \\
  q_{k+1} &= q_k + I_n (y_k - g(x_k, u_k, q_k))
\end{align*}
$$

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>
<thead>
<tr>
<th>Component</th>
<th>LE</th>
<th>M9K</th>
<th>DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEKF IP (design C only)</td>
<td>20,610</td>
<td>30</td>
<td>29</td>
</tr>
<tr>
<td>Matrix processor (design B only)</td>
<td>6,304</td>
<td>35</td>
<td>24</td>
</tr>
</tbody>
</table>

continued...
1. Battery Management System Reference Design

1.5. BMS Reference Design Benchmarking

### Table 6. Execution Time for Updating One Cell

<table>
<thead>
<tr>
<th>Implementation Method</th>
<th>Time to Update One Cell (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design A</td>
<td>44.9</td>
</tr>
<tr>
<td>Design B</td>
<td>33.8</td>
</tr>
<tr>
<td>Design C</td>
<td>16.5</td>
</tr>
</tbody>
</table>

1.6. Acknowledgements for the BMS Reference Design

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