AVR458: Charging Lithium-Ion Batteries with ATAVRBC100

Features

• Fully Functional Design for Charging Lithium-Ion Batteries
• High Accuracy Measurement with 10-bit A/D Converter
• Modular “C” Source Code
• Easily Adjustable Battery and Charge Parameters
• Serial Interface for Communication with External Master
• One-wire Interface for Communication with Battery EEPROM
• Analogue Inputs for Reading Battery ID and Temperature
• Internal Temperature Sensor for Enhanced Thermal Management
• On-chip EEPROM for Storage of Battery and Run-Time Parameters

1 Introduction

This application note is based on the ATAVRBC100 Battery Charger reference design (BC100) and focuses on how to use the reference design to charge Lithium-Ion (Li-Ion) batteries. The firmware is written entirely in C language (using IAR® Systems Embedded Workbench) and is easy to port to other AVR® microcontrollers.

This application is based on the ATtiny861 microcontroller but it is possible to migrate the design to other AVR microcontrollers, such as pin-compatible devices ATtiny261 and ATtiny461. Low pin count devices such as ATtiny25/45/85 can also be used, but with reduced functionality.
2 Theory of Operation

Battery charging is made possible by a reversible chemical reaction that restores energy in a chemical system. Depending on the chemicals used, the battery will have certain characteristics. A detailed knowledge of these characteristics is required in order to avoid inflicting damage to the battery.

2.1 Li-Ion Battery Technology

Lithium-Ion batteries have the highest energy/weight and energy/space ratios of modern rechargeable batteries /1/ (See “References” section on page 33). It is currently the fastest growing battery system on the market, with end applications such as notebook computers, cell phones, portable media players, Personal Digital Assistants (PDA), power tools and medical devices.

Compared to traditional, rechargeable batteries, Li-Ion batteries have low internal resistance, high cycle life, fast charge time, low self-discharge, low toxicity and no maintenance requirements. For example, lithium-ion cells with cobalt cathodes hold twice the energy of a nickel-based battery and four-times that of lead acid /2/. Lithium-ion is a low maintenance system, an advantage that most other chemistries cannot claim. There is no memory effect with lithium-ion and the battery does not require scheduled cycling to prolong its life. Lithium-ion has a low self-discharge and is environmentally friendly. Disposal causes minimal harm.

Drawbacks of Li-Ion batteries include low tolerance of overcharge and the need for embedded protection circuitry. An electrical short can result in a large current flow, a temperature rise and thermal runaway in which flaming gases are vented.

2.1.1 Safety

Lithium-ion batteries are safe, provided certain precautions are met when charging and discharging. In addition, battery manufacturers ensure a high level of reliability by adding three layers of protection, as follows:

1. The amount of active material is limited to achieve a workable equilibrium of energy density and safety.
2. Various safety mechanisms are included within each cell.
3. An electronic protection circuit is added inside the battery pack.

Cell protection devices work as follows:

- A PTC (positive temperature coefficient) device acts as a protection to inhibit high current surges.
- The CID (circuit interrupt device) opens the electrical path if an excessively high charge voltage raises the internal cell pressure.
- The safety vent allows a controlled release of gas in the event of a rapid increase in cell pressure.
The electronic protection circuit works as follows:

- A solid-state switch is opened if the charge voltage of any cell reaches a given threshold.
- A fuse cuts the current flow if the skin temperature of the cell approaches 90°C (194°F).
- The current path is cut when cell voltage drops below a given threshold. This is in order to prevent the battery from over-discharging.

Today, lithium-ion is one of the most successful and safe battery chemistries available with billions of cells being produced every year.

### 2.2 Charging Li-Ion Batteries

There is in essence only one way to charge lithium-based batteries /3/. Manufacturers of Lithium-Ion cells have very strict guidelines in charge procedures and the packs should be charged as per the manufacturers "typical" charge technique.

In Japan there are also regulations in place that define maximal charge currents and voltages for different temperature ranges. This is explained in more detail in subsection 2.2.3.

Li-Ion batteries are charged using constant voltage, with current limiter to avoid overheating in the initial stage of the charging process. Charging is terminated when the charge current drops below a threshold set by the manufacturer. The battery takes damage from overcharging and may explode if overcharged.

#### 2.2.1 Safety

Static electricity or a faulty charger may destroy the battery's protection circuit and turn solid-state switches to a permanent ON position. This may happen without the user knowing. A battery with a faulty protection circuit may function normally but does not provide protection against abuse.

Consumer grade lithium-ion batteries cannot be charged below 0°C (32°F). If charged at cold temperatures, battery packs may appear to be charging normally but chemical reactions inside the cells may cause permanent damage and can compromise the safety of the pack.

The battery will become more vulnerable to failure if subjected to impact, crush or high rate charging.

The battery must remain cool. A battery pack that gets hot during charge should not be used.

#### 2.2.2 Priming & Charge Intervals

Unlike many other types of rechargeable batteries, Lithium-Ion batteries do not need priming. The first charge of a Li-Ion battery is no different than the 10th or the 100th charge.

Lithium-ion batteries may be – and should be – charged often. The battery lasts longer with partial rather than full discharges. Full discharges should be avoided because of wear.

The battery loses capacity due to aging, whether used or not.
2.2.3 Charge Stages

There are two charge stages of a Lithium-Ion battery, as follows:

1. Constant current. Charging of a Li-Ion battery starts with applying constant current to the battery. The size of the charge current is battery-dependent and given by the manufacturer. This stage is complete when battery voltage has reached the threshold given by the manufacturer.

2. Constant voltage. After battery threshold voltage has been reached the charger will switch from supplying constant current to supplying constant voltage. This stage is complete when charge current has dropped below the threshold given by the manufacturer.

The below figure illustrates voltage and current of a lithium-ion battery during charging.

**Figure 1-1.** Charge stages and limits of a Varta PoLiFlex® cell

In the figure above, “Overcharge” is the level at which cell protection circuitry cuts in and opens a solid-state switch and discontinues the charge current path. After this, battery voltage typically needs to drop several hundred millivolts before the current path is restored. “Overdischarge” is the level at which the current path is cut in order to prevent the battery from over-discharging. Recommended battery operating voltage is typically a margin away from overcharge and discharge limits.

Note that for compliance with Japanese regulations, the charge current and voltage may not exceed a set of maxima defined for four temperature ranges, as shown in Table 1-1. If the temperature is out of the specified ranges, i.e., below 0 or above 60, charging must stop. This behaviour is implemented in this application note, but is not enabled by default. See section 4.5.1 for help on configuration.
Table 1-1: Maximum charge current and voltage, according Japanese regulations.

<table>
<thead>
<tr>
<th>Range Identifier</th>
<th>Temperature Range</th>
<th>Maximum Current</th>
<th>Maximum Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>0 – 10 °C</td>
<td>1.0 C</td>
<td>4.10 V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5 C</td>
<td>4.25 V</td>
</tr>
<tr>
<td>T1</td>
<td>10 – 45 °C</td>
<td>1.0 C</td>
<td>4.25 V</td>
</tr>
<tr>
<td>T2</td>
<td>45 – 50 °C</td>
<td>1.0 C</td>
<td>4.15 V</td>
</tr>
<tr>
<td>T3</td>
<td>50 – 60 °C</td>
<td>1.0 C</td>
<td>4.10 V</td>
</tr>
</tbody>
</table>

T0 is a special case, with two different sets of maxima. For the fastest charging in this temperature range, a charging algorithm like the following should be used:

1. Constant current (1.0 C) until voltage reaches 4.10 V.
2. Constant voltage (4.10 V) until current sinks to 0.5 C.
3. Constant current (0.5 C) until voltage reaches 4.25 V.
4. Constant voltage (4.25 V) until current sinks below threshold.

Naturally, the charger must also be able to handle transitions between the different temperature ranges. If the battery manufacturer specifies more restrictive limits, they should of course be used instead of these.

2.2.4 Typical Charge Characteristics

Battery specifications should always be verified from manufacturer’s data sheets. Below is a summary of typical lithium-ion battery charge characteristics. Actual parameters may vary.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Typical Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charge time</td>
<td>3 hours</td>
</tr>
<tr>
<td>Charge current</td>
<td>1 C</td>
</tr>
<tr>
<td>Charge efficiency</td>
<td>99.9 %</td>
</tr>
<tr>
<td>Charge current threshold</td>
<td>0.03 C</td>
</tr>
<tr>
<td>Charge voltage</td>
<td>4.20 V</td>
</tr>
<tr>
<td>Charge voltage tolerance (per cell)</td>
<td>± 0.05 V</td>
</tr>
<tr>
<td>Temperature range</td>
<td>0 … +45 °C</td>
</tr>
<tr>
<td>Humidity range</td>
<td>65 ± 20 RH</td>
</tr>
</tbody>
</table>

2.2.5 Typical Battery Characteristics

The table below summarises manufacturer’s data for the batteries types used in this application. Other types of batteries may be used, but may require adjustments to software and/or hardware.

Table 1-3. Manufacturer’s data for Varta PoLiFlex range of lithium-ion batteries /4/
<table>
<thead>
<tr>
<th>Parameter</th>
<th>PLF 443441</th>
<th>PLF 383562</th>
<th>PLF 503562</th>
<th>2P/PLF 503562</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominal voltage</td>
<td>3.70</td>
<td></td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>Operating voltage range</td>
<td>2.75 … 4.20</td>
<td></td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>Charge voltage</td>
<td>4.20</td>
<td></td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>Charge voltage tolerance</td>
<td>± 50</td>
<td></td>
<td></td>
<td></td>
<td>mV</td>
</tr>
<tr>
<td>Charge current</td>
<td>520</td>
<td>720</td>
<td>955</td>
<td>955</td>
<td>mA</td>
</tr>
<tr>
<td>Charge cut-off time</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>hours</td>
</tr>
<tr>
<td>Charge cut-off current</td>
<td>10</td>
<td>14</td>
<td>19</td>
<td>38</td>
<td>mA</td>
</tr>
<tr>
<td>RID (resistor ID)</td>
<td>3.9</td>
<td>6.8</td>
<td>10</td>
<td>24</td>
<td>kΩ</td>
</tr>
<tr>
<td>NTC</td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>kΩ</td>
</tr>
<tr>
<td>B-value</td>
<td></td>
<td></td>
<td></td>
<td>3435</td>
<td>K</td>
</tr>
<tr>
<td>Overcharge detection</td>
<td></td>
<td></td>
<td></td>
<td>4.35</td>
<td>V</td>
</tr>
<tr>
<td>Overdischarge detection</td>
<td></td>
<td></td>
<td></td>
<td>2.20</td>
<td>V</td>
</tr>
</tbody>
</table>
2.3 Battery Charger

This application note is based on the ATAVRBC100 Battery Charger reference design by Atmel®. The reference design is rather complex and has loads of features but this application focuses on the low end of the design, only. For more information on the BC100, please see AVR451 - BC100 Hardware User’s Guide /5/.

2.3.1 Microcontroller

The BC100 hosts two microcontrollers; a master (ATmega644, by default) and a slave (an ATtiny25/45/85 or ATtiny261/461/861, by default). The master microcontroller is outside the scope of this application but it may be noted that the microcontrollers are capable of communicating with each other such that the master may request data from the slave at any time.

The slave microcontroller is fully capable of handling all tasks related to battery charging and it does not require a master microcontroller to be present. It constantly scans the connectors for batteries and, if found, charges them when required. The slave microcontroller also constantly monitors the hardware for any anomalies.

2.3.2 Power supply

This application note does not focus on the power supply. It may, however, be noted that the firmware constantly monitors the input voltage levels in order to make sure operation is reliable.

2.3.3 Buck switches

The firmware on the slave microcontroller controls any of the three buck switches on board the BC100. The default is to use a high-frequency PWM output of the microcontroller to adjust the voltage and current flow to the battery. The voltage (and current) of the buck switches are directly proportional to the duty cycle of the PWM signal.
3 Battery Charger Hardware

This application note is based on the ATAVRBC100 Battery Charger reference design. A detailed hardware description will not be provided in this document. Please see AVR451 - BC100 Hardware User's Guide for detailed information.

3.1 Configuration

The ATAVRBC100 Battery Charger reference design must be configured as detailed below.

3.1.1 Microcontroller

The hardware should be populated as follows:

- Make sure socket SC300 is empty
- Populate socket SC301 with an ATtiny861

It is possible to use other AVR microcontrollers but this application has been optimised for using ATtiny861. Pin compatible replacements such as ATtiny261 and ATtiny461 /6/ may be used if the compiled code size is decreased. This can be done by increasing the optimisation of the compiler and by removing unwanted features from the firmware.

Other microcontroller options include ATtiny25, ATtiny45 and ATtiny85 /7/. These (as well as other 8-pin AVR microcontrollers) use the SC300 socket on BC100. It should be noted that due to reduced pin count the 8-pin microcontrollers provide less features than the default 20-pin.

3.1.2 Programming Connector

The microcontroller can be programmed via 6-pin connector J301, using either SPI or debugWIRE.

Please note that in some hardware revisions of BC100 it may be necessary to remove R303 and disconnect pin 15 of U202. This procedure frees the /RESET line for use by external programmer or debugger but removes the possibility for the master microcontroller to reset the slave. Do not engineer the board unless required. Alternatively, the microcontroller can always be programmed off-board.

3.1.3 Jumpers

The jumpers should be configured as follows:

- J400, J401, J407 & J408: Set jumpers to use Buck Switch C (20V / 1A)
- J405 & J406: Set jumpers to 1/4 (max measurable voltage 10V)

Other configurations are possible, but may require firmware changes. See variable VBAT_RANGE in file ADC.h.
3.1.4 Battery

This application uses a particular type of lithium-ion batteries and all configurations presented here are based on manufacturer’s data. Other lithium-ion batteries may naturally be used but it is up to the user to look up battery data from manufacturer’s data sheets and make sure necessary adjustments are done to firmware and hardware. See section 4.5.1 and file battery.h.

The figure below illustrates connection pads of the lithium-ion batteries used in this application.

**Figure 1-2.** Connection pads of a Varta PoLiFlex cell.

The battery is connected to the battery charger as follows.

<table>
<thead>
<tr>
<th>Battery Connector</th>
<th>Charger Connector</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>- (minus)</td>
<td>BATTERY-</td>
<td></td>
</tr>
<tr>
<td>NTC</td>
<td>NTC/RID</td>
<td>Battery temperature measurement</td>
</tr>
<tr>
<td>ID</td>
<td>SCL</td>
<td>RID, Battery identification resistor</td>
</tr>
<tr>
<td>+ (plus)</td>
<td>BATTERY+</td>
<td></td>
</tr>
</tbody>
</table>

3.1.5 Data EPROM

Some batteries are equipped with an embedded EPROM for storing charge and manufacturing data. This application supports the use of EPROM via a one-wire interface. The default is a DS2502 EPROM connected as follows.

<table>
<thead>
<tr>
<th>EPROM Pin</th>
<th>Charger Connector</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA</td>
<td>1-WIRE/SDA</td>
</tr>
<tr>
<td>GND</td>
<td>BATTERY-</td>
</tr>
</tbody>
</table>

If an EPROM is not connected to the battery charger the application will simply disregard its absence.
3.1.6 Supply Voltage

The higher the supply voltage, the higher the minimum current the buck switches can provide. For example, if supply voltage is about 9 V and buck charger C is used to charge a battery at 4.20 V then the minimum attainable current is about 80 mA. At this point the smallest decrease in PWM duty cycle (i.e. reducing the contents of OCR1B by 1) will effectively turn off the current to the battery.

It is recommended to use a supply voltage some three volts above battery charge voltage. In this application the battery is being charged at 4.20 V and the recommended supply voltage is therefore 7.5 V.

Another method to lower the minimum charge current the hardware can provide is to use a buck switch with a large inductor. In BC100 this means Buck Switch A.
4 Battery Charger Software

The firmware is written in C language using IAR Systems Embedded Workbench, version 4.20. Since the firmware has been written entirely in C, it should not be a difficult task to port it to other AVR C-compilers. Some compiler specific details may, however, need to be rewritten.

In the table below are listed the files that are relevant to the compiler project.

<table>
<thead>
<tr>
<th>File</th>
<th>Type</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC.c</td>
<td>C source code</td>
<td>Functions related to A/D converter</td>
</tr>
<tr>
<td>ADC.h</td>
<td>Header file</td>
<td></td>
</tr>
<tr>
<td>battery.c</td>
<td>C source code</td>
<td>Resistor ID and NTC lookup-tables, and functions related to battery control &amp; data acquisition</td>
</tr>
<tr>
<td>battery.h</td>
<td>Header file</td>
<td></td>
</tr>
<tr>
<td>charfunc.c</td>
<td>C source code</td>
<td>Functions that do the actual charging, according to specified parameters</td>
</tr>
<tr>
<td>charfunc.h</td>
<td>Header file</td>
<td></td>
</tr>
<tr>
<td>enums.h</td>
<td>Header file</td>
<td>Enumerations used in ADC.c and time.c</td>
</tr>
<tr>
<td>LIIONcharge.c</td>
<td>C source code</td>
<td>The charging algorithm</td>
</tr>
<tr>
<td>charge.h</td>
<td>Header file</td>
<td></td>
</tr>
<tr>
<td>LIIONspecs.h</td>
<td>Header file</td>
<td>Cell &amp; battery specific definitions</td>
</tr>
<tr>
<td>main.c</td>
<td>C source code</td>
<td>Main program / Program entry point</td>
</tr>
<tr>
<td>main.h</td>
<td>Header file</td>
<td></td>
</tr>
<tr>
<td>menu.c</td>
<td>C source code</td>
<td>State machine definitions</td>
</tr>
<tr>
<td>menu.h</td>
<td>Header file</td>
<td></td>
</tr>
<tr>
<td>OWI.c</td>
<td>C source code</td>
<td>Functions related to one-wire interface</td>
</tr>
<tr>
<td>OWI.h</td>
<td>Header file</td>
<td></td>
</tr>
<tr>
<td>PWM.c</td>
<td>C source code</td>
<td>Functions related to generating pulse-width modulated output</td>
</tr>
<tr>
<td>PWM.h</td>
<td>Header file</td>
<td></td>
</tr>
<tr>
<td>statefunc.c</td>
<td>C source code</td>
<td>Functions related to the states defined in menu.c</td>
</tr>
<tr>
<td>statefunc.h</td>
<td>Header file</td>
<td></td>
</tr>
<tr>
<td>structs.h</td>
<td>Header file</td>
<td>Declarations of various structs used throughout the project</td>
</tr>
<tr>
<td>time.c</td>
<td>C source code</td>
<td>Functions related to timekeeping</td>
</tr>
<tr>
<td>time.h</td>
<td>Header file</td>
<td></td>
</tr>
<tr>
<td>USI.c</td>
<td>C source code</td>
<td>Functions related to serial interface</td>
</tr>
<tr>
<td>USI.h</td>
<td>Header file</td>
<td></td>
</tr>
</tbody>
</table>

4.1 Overview

The firmware integrates all functions required to charge two lithium-ion batteries. Batteries are connected to separate ports such that one may be charged while the other is idle. The firmware is fully automated and capable of stand-alone battery
monitoring and charging but it may also be used together with a master microcontroller, such as the one implemented in BC100.

By default, the firmware fits into an ATtiny861 (build option: debug) or an ATtiny461 (build option: release). Memory requirements of the firmware are summarised in the table below.
### Table 1-7. Memory requirements of firmware

<table>
<thead>
<tr>
<th>Build option</th>
<th>Memory</th>
<th>Approximate value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>CODE (Flash)</td>
<td>5800 bytes</td>
</tr>
<tr>
<td></td>
<td>DATA (SRAM)</td>
<td>270 bytes</td>
</tr>
<tr>
<td></td>
<td>XDATA (EEPROM)</td>
<td>130 bytes</td>
</tr>
<tr>
<td>Release</td>
<td>CODE (Flash)</td>
<td>3900 bytes</td>
</tr>
<tr>
<td></td>
<td>DATA (SRAM)</td>
<td>270 bytes</td>
</tr>
<tr>
<td></td>
<td>XDATA (EEPROM)</td>
<td>130 bytes</td>
</tr>
</tbody>
</table>

### 4.2 State Machine

The state machine is rather simple and resides in the `main()` function. It simply looks up the address of the next function to execute and then jumps to that function. The flow chart of the state machine is illustrated in the figure below.

![Flow chart of main function, including the state machine](attachment:flow_chart.png)

Upon return, the state machine expects the function to indicate the next state as a return argument. The recognised return codes are described in the table below.

### Table 1-8. State machine codes (see source code, menu.h)

<table>
<thead>
<tr>
<th>Label</th>
<th>Related Function (2)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>Initialize()</td>
<td>Entry state</td>
</tr>
<tr>
<td>BATCON</td>
<td>BatteryControl()</td>
<td>Check hardware and batteries</td>
</tr>
<tr>
<td>PREQUAL</td>
<td>Charge()</td>
<td>Raise battery voltage, safety check</td>
</tr>
<tr>
<td>SLEEP</td>
<td>Sleep()</td>
<td>Low power consumption mode</td>
</tr>
<tr>
<td>CCURRENT</td>
<td>Charge()</td>
<td>Charge with constant current</td>
</tr>
<tr>
<td>CVOLTAGE</td>
<td>Charge()</td>
<td>Charge with constant voltage</td>
</tr>
<tr>
<td>MAXVOLTCURR</td>
<td>Charge()</td>
<td>Charge with maximum voltage or current</td>
</tr>
<tr>
<td>ENDCONSTRAIGHT</td>
<td>Charge()</td>
<td>End of successful charge</td>
</tr>
<tr>
<td>DISCHARGE</td>
<td>Discharge()</td>
<td></td>
</tr>
<tr>
<td>ERROR</td>
<td>Error()</td>
<td>Resolve error, if possible</td>
</tr>
</tbody>
</table>
State functions are described in the following sections.

4.2.1 Initialize()

The initialisation function is the first state function that will be executed after device reset. The flow chart of the function is shown in the figure below.

**Figure 1-4. Flow chart of initialisation function**

```
Initialize()
  ↓
Set clock prescaler to 1
  ↓
Initialize One-Wire Interface
  ↓
Configure I/O pins and disable all batteries
  ↓
Initialize Serial Peripheral Interface
  ↓
Initialize Analog-to-Digital Converter
  ↓
Read data from all batteries
  ↓
Initialize timer functions
  ↓
Return(BATCON)
```

The initialisation function always exits with the same return code, pointing to the state function for battery control.

4.2.2 BatteryControl()

The battery control function verifies that jumpers are set correctly and then checks to see if there are any enabled batteries present that require charging. The program flow is illustrated in the figure below.
**4.2.3 Charge()**

In the current implementation, the charge function has two different modes of operation, depending on if it is configured to comply with Japanese regulations or not.

The non-compliant mode is the simplest and consists of four stages:

- **Prequalification** - during which the battery is charged with a constant current until a sufficient charge voltage is reached. If this happens within a given time limit, the battery is considered good and the charger may continue on the next stage. If time runs out before the voltage is reached, or battery temperature goes out of limits, the battery is considered bad and charging is halted.

- **Constant current charge** - during which the battery is charged with a higher, battery-specific current until the battery voltage reaches its maximum. If this happens within the battery’s maximum charge time limit, the charger goes to the next stage. If the time limit expires, or battery temperature goes out of limits, the battery is considered bad and charging is halted.

- **Constant voltage charge** – during which the battery is charged at the maximum battery voltage until the charge current sinks beneath a battery-specific cut-off limit, or the maximum charge time limit expires. Here too, charging is halted if battery temperature goes out of limits.

- **End charge** – in which the charger decides whether to go into the sleep state, or to attempt a charge of the other battery.
ChargeParameters and HaltParameters are central variables in this function. The program flow of this state function is illustrated in the figure below.

**Figure 1-6. Flow chart of the charge state function**

The compliant mode has combined the second (ST_CCURRENT) and third (ST_CVOLTAGE) state into one: ST_MAXVOLTCURR. This function essentially makes sure that both the charge voltage and the current are within the limits for the temperature range.
4.2.4 Discharge()

This function has not been implemented.

4.2.5 Sleep()

The application enters sleep mode when all batteries have been fully charged. It wakes up at regular intervals to check the current status of the batteries. Sleep mode is terminated as soon as any battery requires charging.

Sleep mode is illustrated in the flow chart below.

**Figure 1-7. Flow chart of sleep function**

![Flow chart of sleep function](image)

4.2.6 Error()

Program flow is diverted here when an error has occurred. The error handler contains some simple algorithms that try to resolve the most common problems. Program execution will exit the error handler when all sources of error have been cleared.

The program flow is illustrated in the figure below.
Figure 1-8. Flow chart of error handler

1. Error()
2. Stop PWM output
3. Disable all batteries
4. Sleep for 8 seconds
5. Jumper mismatch error?
   - YES: Clear bit in error flag
   - NO: Check jumpers
6. NO batteries error?
   - YES: Any batteries enabled?
     - YES: Clear bit in error flag
     - NO: Clear bit in error flag
   - NO: Clear bit in error flag
7. PWM control error?
   - YES: Clear bit in error flag
   - NO: Clear bit in error flag
8. Battery temperature error?
   - YES: Clear bit in error flag
   - NO: Clear bit in error flag
9. Battery exhausted error?
   - YES: Clear battery exhausted bit
   - NO: Change active battery
   - NO: Clear bit in error flag
10. Any error flags set?
    - YES: Return(INIT)
    - NO: Clear bit in error flag
4.3 Charging Functions

These functions are called by Charge() after all parameters have been set.

4.3.1 Constant Current/Voltage

These two functions are similar, apart from what ADC measurements they try to keep within limits. Therefore, only the flow chart for ConstantCurrent() is illustrated in the figure below. They both make use of the variable ChargeParameters.

If a Master microcontroller is present, it may temporarily stop the charging by flagging a charge inhibit. This is to prevent battery damage during prolonged serial transfers.
4.3.2 Maximum Voltage and Current

Due to Japanese regulations for battery chargers, a third charging function is supplied: MaxVoltageAndCurrent(). This function continually monitors the battery’s temperature, and makes sure the charging parameters (both current and voltage) are within the limits defined by the regulations, shown earlier in Table 1-1. In essence, it is a temperature dependent version of the two other charging functions combined. Figure 1-10 shows a somewhat simplified flowchart for the function.
Figure 1-10: Flowchart for MaxVoltageAndCurrent()

MaxVoltageAndCurrent()

Wait for ADC conversions to complete.

Do NTC-lookup.

Has temperature range changed?

Is current range T0?

Set charge parameters according to new range.

Set charge parameters according to T0.

Is either voltage or current above charge parameters?

Increment PWM duty cycle.

Decrement PWM duty cycle.

Are both voltage and current below charge parameters?

Yes

NO

HaltNow()? YES

Return next state.

NO

Charging of battery inhibited?

Flag that Master MCU stopped the charging.

Stop timers.

NO

Charging inhibited?

NO

Were we stopped by Master MCU earlier?

NO

Remove flag that Master MCU stopped the charging.

Start timers again.

YES

NO

If charging is inhibited?

NO

Drop PWM output to zero.

Note that this function does not stop charging if the temperature is out of bounds (below 0°C or above 60°C) since charge halts are decided by a dedicated function, described next.

4.3.3 Charge Halt Determination

Charge halt is determined by HaltNow(). This function is called by ConstantCurrent(), ConstantVoltage() and MaxVoltageAndCurrent() every time they loop, to decide if a stage of charging is done.

With the variable HaltParameters the user can specify at what terms the charging should be halted, and if an error should be flagged if, for example, the time limit expires. Note
that if the Master MCU inhibits the charging, a drop in voltage or the charge current falling below the minimum threshold will not trigger a halt. An error flag will also result in ST_ERROR being set as the next state, thereby aborting the charge. If no errors are flagged, the next desired state, set earlier in Charge(), will apply.

Lastly, the function checks if temperature is within limits, if the battery is OK and if mains voltage is above minimum. Should any of these tests fail, the next state is set to an appropriate error handler (ST_ERROR, ST_INIT or ST_SLEEP) and charging is aborted.
Figure 1-11. Flow chart for HaltNow() part 1.

HaltNow()

Wait for ADC conversions to finish.

Do NTC lookup.

- Halt on voltage drop selected?
  - YES
  - YES
  - NO
  - NO

- Halt on maximum voltage selected?
  - YES
  - NO

- Halt on minimum current selected?
  - YES
  - NO

Output voltage higher than stored maximum?

Output voltage above or equal to limit?

Charge inhibited by master?

Voltage drop above or equal to limit?

Store new maximum.

Set Halt flag.

Set Halt flag.

Set Halt flag.

Set Halt flag.
Figure 1-12. Flow chart for HaltNow() part 2

1. Halt on temperature rise?
   - YES
     1.1. Measured NTC above stored NTC?
        - YES
           1.1.1. Store NTC value.
        - NO
           1.1.2. Start temperature timer.
   - NO
     1.2. NO

2. NO
   - NO
     2.1. Set Halt flag.
   - YES
     2.2. Reset temperature timer.

3. Temperature timer run out?
   - YES
     3.1. Store NTC value.
   - NO
     3.2. Reset temperature timer.
Figure 1-13. Flow chart for HaltNow() part 3

2

Halt on timeout?

YES

Charging timer run out?

YES

Set Halt flag.

Flag battery exhaustion?

YES

Stop PWM output.

NO

NO

Disable battery and flag it as exhausted.

Flag battery exhaustion error and set ST_ERROR as next state.

3
4.4 Other Functions

The program flow is mainly state-based, but some processing takes place in the background. This includes A/D conversion, time keeping and serial interface handling. All of these functions are interrupt-driven.

4.4.1 A/D Conversion

The A/D converter uses the multiplexer to read in data from several channels. At the end of a conversion the ADC Interrupt Service Routine (ISR) is called, as illustrated in
the flow chart below. After the ISR is complete program execution will return to normal.

**Figure 1-15. Flow chart of ADC interrupt service routine**
4.4.2 Master-Slave Communication

This application is designed to work as stand-alone but it also supports co-operation with other microcontrollers. The Universal Serial Interface (USI) can be used for communication between microcontrollers. The basic protocol for this interface has been developed but some functions need to be finalised.

**Figure 1-16.** Flow chart of USI overflow interrupt service routine

4.5 Implementation

This section describes how to configure, create and download the software.

4.5.1 Configuration

The most important compile-time constants are mentioned in the tables below.
### Table 1-9. Battery-related compile-time constants (see source file LIIONspecs.h)

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAT_CELL_NUMBER</td>
<td>The number of cells in the battery. Each of the defined cell voltages gets multiplied by this, to define BAT_VOLTAGE_MAX, _LOW, _MIN and _PREQUAL.</td>
</tr>
<tr>
<td>CELL_VOLTAGE_SAFETY</td>
<td>In case unmatched batteries are to be charged, this constant is subtracted from CELL_VOLTAGE_MAX for every extra cell in the battery, ie. BAT_CELL_NUMBER – 1.</td>
</tr>
<tr>
<td>CELL_VOLTAGE_MAX_T0</td>
<td>The maximum cell voltage to charge to, at different temperature ranges.</td>
</tr>
<tr>
<td>CELL_VOLTAGE_MAX_T1</td>
<td></td>
</tr>
<tr>
<td>CELL_VOLTAGE_MAX_T2</td>
<td></td>
</tr>
<tr>
<td>CELL_VOLTAGE_MAX_T3</td>
<td></td>
</tr>
<tr>
<td>CELL_VOLTAGE_LOW</td>
<td>The lowest voltage at which a cell is considered charged. Charging will start when voltage drops below this level.</td>
</tr>
<tr>
<td>CELL_VOLTAGE_MIN</td>
<td>The lowest voltage at which charging may be initiated. Should generally be set to the voltage limit under which further discharge of batteries will cause damage.</td>
</tr>
<tr>
<td>CELL_VOLTAGE_PREQUAL</td>
<td>The voltage to which a cell should be charged to during prequalification.</td>
</tr>
<tr>
<td>BAT_TEMPERATURE_MAX</td>
<td>The highest battery temperature allowed. Charging will stop / not start if above this.</td>
</tr>
<tr>
<td>BAT_TEMPERATURE_MIN</td>
<td>The lowest battery temperature allowed. Charging will stop / not start if above this.</td>
</tr>
<tr>
<td>BAT_CURRENT_PREQUAL</td>
<td>Charge current during prequalification mode.</td>
</tr>
<tr>
<td>BAT_CURRENT_HYST</td>
<td>Charge current hysteresis. Current will not be adjusted when within plus or minus this value from target.</td>
</tr>
<tr>
<td>BAT_VOLTAGE_MAX_T0</td>
<td>Maximum battery voltage to charge to, calculated from BAT_CELL_NUMBER, CELL_VOLTAGE_MAX&lt;...&gt; and CELL_VOLTAGE_SAFETY.</td>
</tr>
<tr>
<td>BAT_VOLTAGE_MAX_T1</td>
<td></td>
</tr>
<tr>
<td>BAT_VOLTAGE_MAX_T2</td>
<td></td>
</tr>
<tr>
<td>BAT_VOLTAGE_MAX_T3</td>
<td></td>
</tr>
<tr>
<td>BAT_VOLTAGE_HYST</td>
<td>Charge voltage hysteresis. Current will not be adjusted when within plus or minus this value from target.</td>
</tr>
<tr>
<td>BAT_VOLTAGE_PREQUAL</td>
<td>Target voltage during prequalification stage. If this voltage is not achieved the battery will be marked as exhausted. Calculated from BAT_CELL_NUMBER and CELL_VOLTAGE_PREQUAL.</td>
</tr>
<tr>
<td>BAT_TIME_PREQUAL</td>
<td>Maximum amount of time to spend in prequalification stage.</td>
</tr>
</tbody>
</table>

### Table 1-10: Compile-time constants for default battery, RID and NTC (battery.c/.h)

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF_BAT_CAPACITY</td>
<td>Default battery capacity.</td>
</tr>
<tr>
<td>DEF_BAT_CURRENT_MAX</td>
<td>Default maximum charge current.</td>
</tr>
<tr>
<td>DEF_BAT_TIME_MAX</td>
<td>Default maximum charge time.</td>
</tr>
<tr>
<td>DEF_BAT_CURRENT_MIN</td>
<td>Default cut-off charge current.</td>
</tr>
</tbody>
</table>
### Table 1-11: Compliant charging configuration (charge.h)

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOW_NO_RID</td>
<td>If defined, batteries without RID (or not matching the lookup-table) will cause the charger to use the above defined battery defaults. Otherwise, lack of RID means that charge is not initiated.</td>
</tr>
<tr>
<td>RID[].Low and RID[].High</td>
<td>Assume RID resistance match if value within these limits.</td>
</tr>
<tr>
<td>RID[].Capacity</td>
<td>Battery capacity for given RID.</td>
</tr>
<tr>
<td>RID[].Icharge</td>
<td>Charge current for given RID.</td>
</tr>
<tr>
<td>RID[].tCutOff</td>
<td>Maximum charge time for given RID.</td>
</tr>
<tr>
<td>RID[].IcutOff</td>
<td>Charge termination current for given RID.</td>
</tr>
<tr>
<td>NTC[].ADC</td>
<td>ADC-value for different temperatures (array index equals temperature associated with value).</td>
</tr>
<tr>
<td>NTC[].ADCsteps</td>
<td>Number of ADC steps per half degree change from measured temperature.</td>
</tr>
</tbody>
</table>

### Table 1-12: Temperature settings for compliant charging (chargefunc.h)

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMP_HYST</td>
<td>Temperature hysteresis, only used if the charger is configured to comply with Japanese regulations.</td>
</tr>
<tr>
<td>T0</td>
<td>Define the maximum temperature in the different ranges. (Keep in mind that BAT_TEMPERATURE_MAX/MIN will stop the charging.)</td>
</tr>
<tr>
<td>T1</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td></td>
</tr>
</tbody>
</table>

### 4.5.2 Compilation

Before compiling the code the following configurations should be made.

<table>
<thead>
<tr>
<th>Section</th>
<th>Tab</th>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Options</td>
<td></td>
<td>Processor configuration</td>
<td>ATtiny861 (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Memory model</td>
<td>Small</td>
</tr>
<tr>
<td>System</td>
<td></td>
<td>Data stack</td>
<td>0x40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Return address stack</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enable bit definitions</td>
<td>Selected</td>
</tr>
</tbody>
</table>
4.5.3 Programming

The compiled code is conveniently downloaded to the target device using AVR Studio® and a debugger or programming tool of choice, such as the JTAGICE mkII.

Note that the compiled code contains EEPROM data that must be loaded to the target for the software to work. Answer OK when AVR Studio asks if EEPROM contents should be loaded. This is illustrated in the figure below.

**Figure 1-17. Loading initialised data to EEPROM**

[Initialized Data]

This object file indicates initialized EEPROM data. Do you want to load this data?

[OK] [Cancel]

The program expects the use of the internal oscillator and that the clock signal is not prescaled. Some fuse bits must be programmed to ensure proper program execution. The fuse bit settings that deviate from the default are listed in the table below.

**Table 1-14. Non-default fuse bit settings**

<table>
<thead>
<tr>
<th>Fuse Bit</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKDIV8</td>
<td>1 (unprogrammed)</td>
<td>Do not divide clock by eight</td>
</tr>
<tr>
<td>CKSEL3...0</td>
<td>0010</td>
<td>Use internal oscillator</td>
</tr>
</tbody>
</table>
5 Known Limitations

Here are listed known limitations of the design.

5.1 Battery Current Measurement

Battery current is sensed using a shunt resistor with very low resistance. This means noise is easily picked up in the measured signal and that even noise with very low amplitude may disturb the measurements. As a remedy, the battery current measured is averaged over four samples.

Yet, it is not uncommon to find fluctuations in the order of 1 or 2 LSB. By default (see section 3.1.3) this means a measurement error of 7 or 14 mA (see function ScaleI() in file ADC.c). In practice, this may result in premature end of charge cycle.

The suggested solution is to optimise the size of the shunt resistor (R410: the larger, the better) and the resistor divider (R400…R410, R427, R428, R446 and R447).

5.2 RID Sensing

Battery identification resistor is sensed via pin PA2 (ADC2). The default pull-up resistor on this line (R305 in ATAVRBC100 Battery Charger reference design) is 4.7 kohm. This limits the size of the sense resistor to TBD ohm.

When using Varta PoLiFlex batteries this means the largest battery size that can be reliably sensed is 1000 mAh. For larger sense resistors / battery sizes the pull-up resistor on BC100 must be changed. In addition, the software must be updated to reflect the new pull-up resistor value.

5.3 Buck chargers

The choice of buck charger (and supply voltage) sets a limit on how low the minimum charge current may be. The higher the supply voltage and the smaller the buck switch inductor, the higher will the minimum charge current be. This means some configurations may result in premature end of charge cycle.

The remedy is to use a low supply voltage and a buck switch with a large inductor.
6 References


