# AVR463: Charging Nickel-Metal Hydride Batteries with ATAVRBC100

8-bit AVR® Microcontrollers

Application Note

## Features

- **Fully Functional Design for Charging Nickel-Metal Hydride 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 Nickel-Metal Hydride (NiMH) 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.

Rev. 8098A-AVR-09/07

# 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 NiMH Battery Technology

Nickel-Metal Hydride (NiMH) battery technology is the successor of Nickel-Cadmium (NiCd), and is considered a stepping-stone to lithium-based batteries. Its characteristics are quite similar to NiCd, but offers approximately 50% higher energy density at the cost of reduced cycle life [1] (see References page 25). NiMH does not suffer so much from the memory effect as NiCd, and therefore needs fewer exercise cycles during storage. In addition, NiMH batteries are more environmentally friendly as they contain no cadmium, and therefore pose less of a problem regarding disposal. Currently, it is being used in everything from cameras, PDAs and power tools to hybrid cars.

Although Li-Ion batteries offer almost twice the energy density, higher cell voltage and shorter charge times, they are not as stable as nickel-based batteries. Also, since nickel-based batteries have a voltage close to the regular 1.5V Zinc Carbon and Alkaline batteries, they are more suitable for use in existing equipment powered by such batteries.

Compared to NiCd, the drawbacks of NiMH batteries include a slightly higher self-discharge rate (approximately 20% per month at room temperature), lower overcharge tolerance, shorter cycle life (300-500 charge cycles) and longer charge time as it generates more heat.

### 2.1.1 Safety

Nickel-based batteries are quite stable, but might vent gas or explode if severely mishandled (short circuited, charged with the wrong charger, etc.) due to internal pressure and heat build-up. Most batteries come with a safety vent, thermal fuse/switch or pressure switch to help prevent this.

## 2.2 Charging NiMH Batteries

The most common method for recharging nickel-based batteries is with a constant current. In general, the charge current is recommended in the range 0.5 – 1.0 C[1] (fast charge) since slow charging causes crystalline formation, also known as memory effect, and lesser charge efficiency [7]. Greater currents than this, on the other hand, may cause damage. It is also more difficult to detect a full charge of NiMH batteries at low charge currents.

NiMH batteries are between 70% and 90% charge efficient, meaning you must charge for longer than one would calculate from a given charge current and battery capacity.

[1] C represents the battery's capacity per hour, i.e. mA.

### 2.2.1 Safety

Towards the end of a charge cycle, the positive electrode of the battery will reach full charge first, causing it to produce oxygen. To avoid pressure build-up in the cell, the oxygen is recombined at the negative electrode to produce water. This process does however generate heat, and it relies on the charge current to be limited so that the oxygen generation rate is not greater than the recombination rate.

Should a pressure build-up occur, most batteries have a safety vent or even a pressure switch that will cut off the charge current once the pressure gets too high. The venting of gas deteriorates the battery as electrolyte is lost. More advanced batteries have safety electronics that activate once pressure or temperature exceeds a threshold.

### 2.2.2 Battery Temperature

As mentioned, NiMH batteries generate heat during charging and it is therefore important to monitor the battery temperature to prevent damage. Suggested temperature ranges for fast charging are in the area 5 - 45°C, but temperatures between 45°C and 60 °C are acceptable for a short period at the end of a charge. [5] [6].

Naturally, great currents cause a faster increase in temperature. High temperatures decrease cell voltage and charge efficiency, and cause a higher rate of self-discharge. Cooling the batteries with a fan is sometimes done to maximize efficiency, though lower temperatures slow the oxygen recombination, possibly causing a faster pressure build-up.

### 2.2.3 Charge Time

Determining for how long to charge is not so easy since a battery's state of charge cannot be reliably determined from its voltage. In addition, it depends on the battery's capacity and the charge current.

Due to charge inefficiency, the ideal charge input lies in the area 120 – 150%, depending on if maximum cycle life (120%) or capacity (150%) is wanted. Assuming a 1.0 C charge current, a full charge of an empty battery should therefore take between 1.2 hours and 1.5 hours. Since NiMH batteries are not as tolerant of overcharge as NiCd ones, measures must be taken to detect a full charge.

### 2.2.4 Full Charge Detection

At about 60% charge input the heat generation in NiMH cells starts to increase, and may be used to detect a full charge. It is recommended to end the charge once the rate of temperature increase reaches 1 °C per minute [5] [6].

Another sign of a full charge is a drop in voltage due to the oxygen recombination, though this is not as pronounced at high temperatures or low charge currents. Suggested limits are in the range 6 - 20 mV per cell. A low limit will reduce the amount of overcharging, but also heightens the risk of prematurely ending the charge due to electrical noise or other normal drops in voltage during charging.

### 2.2.5 Typical Fast Charge Characteristics

Battery specifications should always be verified from manufacturer's data sheets. Below is a summary of typical nickel-metal hydride battery fast charge characteristics.

**Table 2-1.** Typical Charge Characteristics for Fast Charge

| Parameter | Typical Value |
|---|---|
| Charge time | 1.5 - 3 hours |
| Charge current | 0.5 – 1 C |
| Charge efficiency | 70 – 90 % |
| Charge voltage | 1.4 – 1.6 V |
| Temperature range | 5 … 60 °C (max) |

### 2.2.6 Typical Battery Characteristics

The table below summarises general data for NiMH batteries [5] [6].

**Table 2-2.** Typical NiMH Battery Characteristics

| Parameter | Typical Value |
|---|---|
| Nominal voltage | 1.2 V |
| Open circuit voltage | 1.25 – 1.35 V |
| Typical end voltage | 1.0 V |

### 2.2.7 Three Step Fast Charge

The chosen charge procedure for this application is a fast charge followed by a topping charge, as recommended in literature and by battery & charger manufacturers. This does however require that the batteries' specifications are defined at compile time since we otherwise don't know them. The procedure is as follows:

- Prequalification – Charge at low current (0.1 C) for at most 2 (TBD) minutes, until battery voltage reaches 1.0 V. Timeout means battery is quite likely damaged.
- Fast charge – Charge at high current (C) for at most 1.5 hours, until either the rate of temperature increase reaches 1 °C per minute or the voltage drops with 15 mV per cell.
- Low rate charge – Top up the battery with a low current (0.1 C) for 30 minutes, to ensure a full charge. This current is low enough for oxygen recombination to keep up, in case of an overcharge.

For safety, a temperature limit of 50 °C has been set for the last two stages, and 35 °C for the first.

It is also recommended to leave the batteries on a trickle charge (0.003 C – 0.05 C) for an indefinite duration, to counteract self-discharge. This has not been implemented, but may easily be done so by changing the end charge stage in Charge(). It would be beneficial to switch between the batteries at regular intervals, which would also allow for detection of uncharged batteries.

## 2.3 Battery Charger

This application note targets 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 core feature of the design, the buck converters, only. For

more information on the BC100, please refer to application note AVR451 - BC100 Hardware User's Guide [2].

### 2.3.1 Microcontroller

The BC100 hosts two microcontrollers: a master (ATmega644) and a slave (an ATtiny85 or ATtiny861, 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 converters on board the BC100. The default is to use a high-speed PWM output of the microcontroller to adjust the voltage and current flow to the battery. The voltage (and current) of the buck converters is 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 [3] 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 [4]. 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 tri-state pin 15 of U202 (set /OE low from the mega644). 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 alter the board unless required. Alternatively, the microcontroller can always be programmed off-board.

### 3.1.3 Jumpers

The jumpers should be configured as follows:

- J405 & J406: Set jumpers to 1/4 (max measurable voltage 10V)
- For 300 mAh battery: J401, J404 and J408 should be set to enable Buck converter C (20V / 1A)
- For 1300 mAh battery: J400, J401, J403, J408 and J407 should be set to enable Buck converter B (30V / 2.5A)

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

### 3.1.4 Battery

For testing of this application, two generic 3-cell NiMH batteries from Minamoto were used. The only data available were the specific capacity and nominal voltage. Both batteries were rated at 3.6 V, with the respective capacities 300 mAh and 1300 mAh. These batteries did not include Resistor ID, EPROM or NTC.

### 3.1.5 Battery Resistor ID

Some batteries may include a Resistor ID (RID), which is used to identify the battery type. The charger can then know data such as capacity, maximum charge current and charge time, instead of keeping these parameters static. If RID is used, the resistor values and the associated data are available from the battery manufacturer. This application supports but does not require RID, and if used, the RID lookup-table should be updated. The resistor should be connected as follows:

**Table 3-1.** Connecting battery resistor ID to charger

| Battery Pin | Charger Connector |
|---|---|
| RID | SCL |
| GND | BATTERY- |

If a Resistor ID is not connected, or its value causes ADC saturation, default battery data may be used. See Configuration on page 22.

### 3.1.6 NTC

Temperature measurement during fast charge of NiMH batteries is vital, and some manufacturers therefore often include a thermistor in their batteries. Usually, these have a negative temperature coefficient (NTC).

For this application, an RH16-3H103FB NTC from Mitsubishi was employed, and the NTC lookup-table populated accordingly. The NTC should be connected as follows:

**Table 3-2.** Connecting NTC thermistor to charger

| Battery Pin | Charger Connector |
|---|---|
| NTC | NTC/RID |
| GND | BATTERY- |

In this application, if an NTC is not connected, the ADC will be saturated and temperature registered as -1 °C. This will halt the charging, unless a sub zero minimum battery temperature has been set.

### 3.1.7 Data EPROM

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

**Table 3-3.** Connecting external EPROM DS2502 to charger

| EPROM Pin | Charger Connector |
|---|---|
| DATA | 1-WIRE/SDA |
| GND | BATTERY- |

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

### 3.1.8 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 voltage will typically max out at 4.5 V during charging, so the recommended supply voltage is 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™. 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 will probably need to be rewritten.

The table below lists the files that are relevant to the compiler project.

**Table 4-1.** Project files

| File | Type | Note |
|---|---|---|
| ADC.c | C source code | Functions related to A/D conversion |
| ADC.h | Header file | |
| battery.c | C source code | Functions related to battery control & data acquisition, and definition of default battery data |
| battery.h | Header file | |
| chargefunc.c | C source code | Functions related to charging |
| chargefunc.h | Header file | |
| enums.h | Header file | Enumerations for time.c and USI.c, which are used in both Slave and Master (SPI communication) |
| main.c | C source code | Main program / Program entry point |
| main.h | Header file | |
| menu.c | C source code | State machine definitions |
| menu.h | Header file | |
| NIMHcharge.c | C source code | The charge state function for charging of NiMH batteries |
| charge.h | Header file | |
| NIMHspecs.h | Header file | Definitions of NiMH cell & battery specifications |
| OWI.c | C source code | Functions related to one-wire interface |
| OWI.h | Header file | |
| PWM.c | C source code | Functions related to generating pulse-width modulated output |
| PWM.h | Header file | |
| statefunc.c | C source code | The different state functions |
| statefunc.h | Header file | |
| structs.h | Header file | Structs for ADC.c and battery c, which are used in both Slave and Master (SPI communication) |
| time.c | C source code | Functions related to timekeeping and measurement of time |
| time.h | Header file | |
| USI.c | C source code | Functions related to serial interface (SPI communcation) |
| USI.h | Header file | |

## 4.1 Overview

The firmware integrates all functions required to charge two NiMH 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.

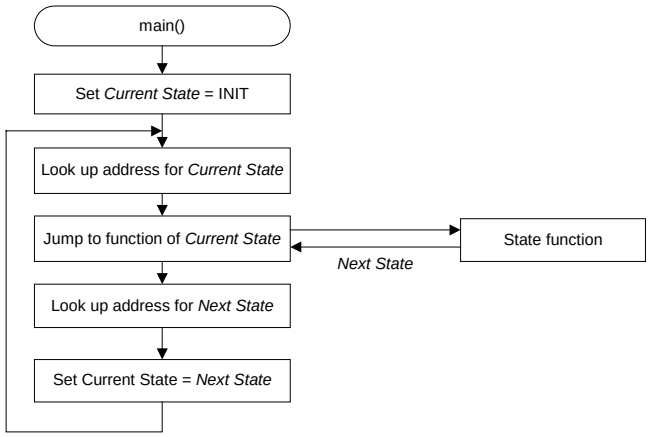
**Table 4-2.** Memory requirements of firmware

| Build option | Memory | Approximate value |
|---|---|---|
| Debug | CODE (Flash) | 5800 bytes |
| | DATA (SRAM) | 270 bytes |
| | XDATA (EEPROM) | 130 bytes |
| Release | CODE (Flash) | 3900 bytes |
| | DATA (SRAM) | 270 bytes |
| | XDATA (EEPROM) | 130 bytes |

## 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.

**Figure 4-1.** Flow chart of main function, including the state machine




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 4-3.** State machine codes (see source code, menu.h)

| Label [1] | Related Function [2] | Description |
|---|---|---|
| INIT | Initialize() | Entry state |
| BATCON | BatteryControl() | Check hardware and batteries |
| PREQUAL | Charge() | Raise battery voltage, safety check. |
| SLEEP | Sleep() | Low power consumption mode |
| FASTCHARGE | Charge() | Charge with constant current at 1.0 C. |
| TRICKLECHARGE | Charge() | Charge with constant current at 0.1 C. |
| ENDCHARGE | Charge() | End of successful charge |
| DISCHARGE | Discharge() | |
| ERROR | Error() | Resolve error, if possible |

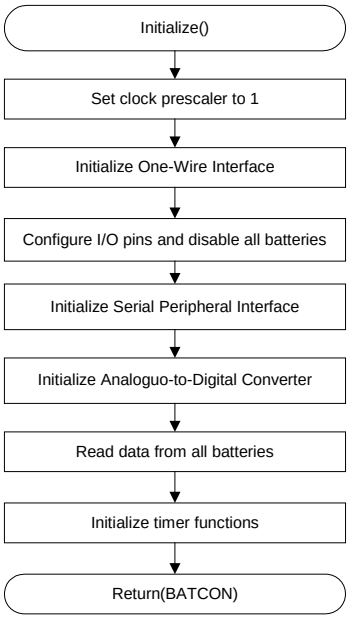
Notes: 1. Name of label, excluding leading “ST_”
2. Function name, as declared in source code

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 4-2.** Flow chart of initialisation function

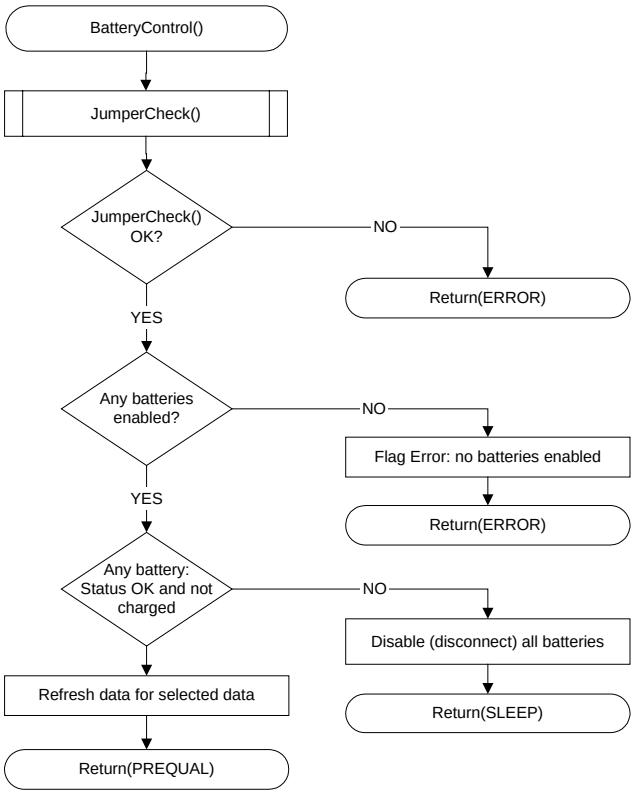



The initialisation function always exits with the same return code, pointing to the state function for battery control.

### 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.

**Figure 4-3.** Flow chart of battery control function



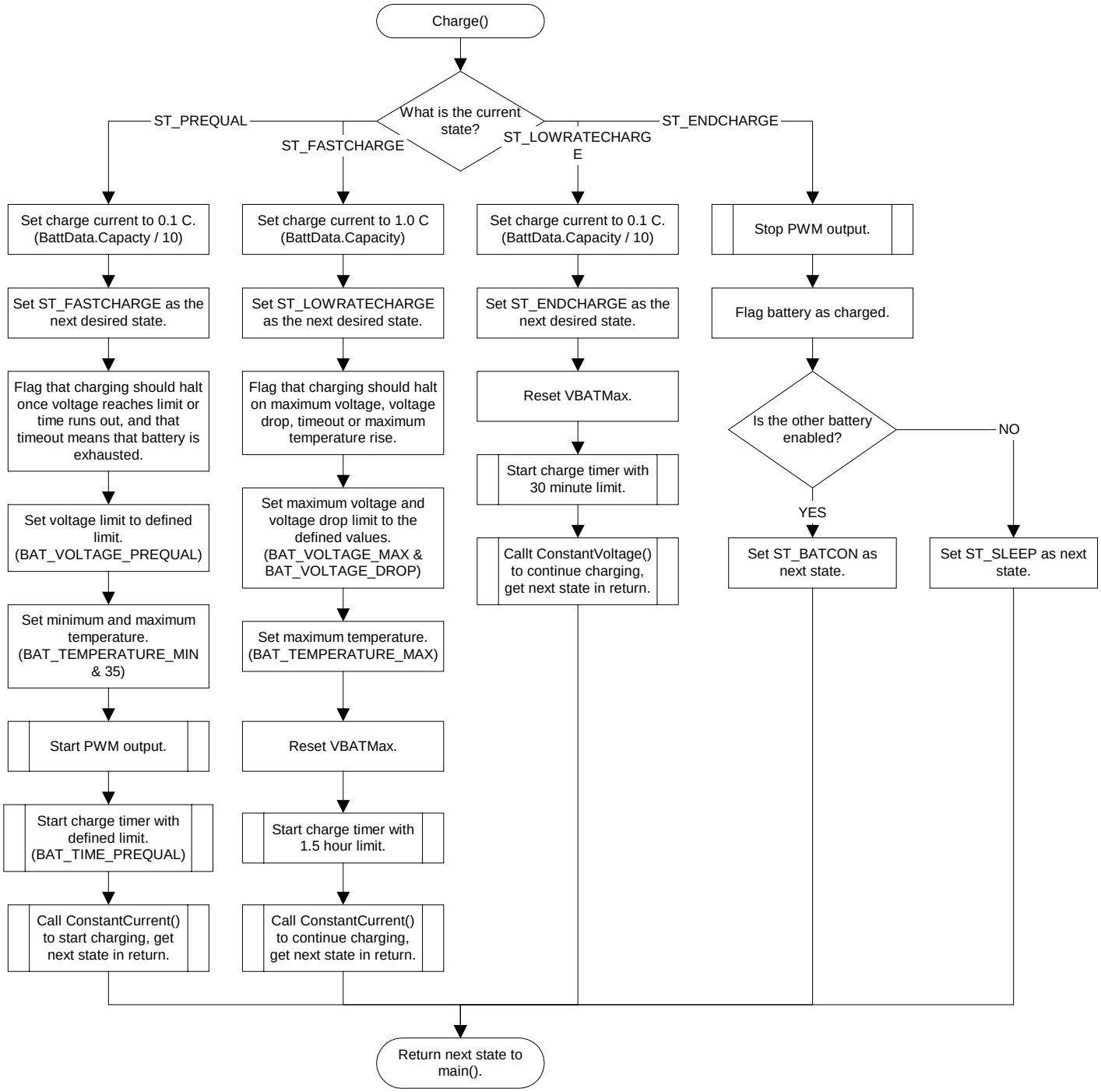

### 4.2.3 Charge()

The charge function contains the charging algorithm divided into stages. For this application, it has four stages:

- Prequalification – If the battery voltage is between 0.8 V and 1.0 V, charge at a low rate like 0.1 C for 2 minutes. If the voltage doesn't rise to 1.0 V within the time limit, the battery is likely damaged. Limit max. temperature to 35 °C.
- Fast charge – Charge at 1.0 C for 1.5 hours at maximum, and terminate when voltage drops 10 – 15 mV per cell or the rate of temperature increase reaches 1 °C per minute. Limit maximum temperature to 50 °C.
- Top-up charge – Top the battery up with a 0.1 C charge current for 30 minutes. Limit maximum temperature to 50 °C.
- End charge – Decide 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. After each stage, the function returns the next desired state to main(). The program flow of this state function is illustrated in the figure below.

**Figure 4-4.** Flow chart of the charge state function

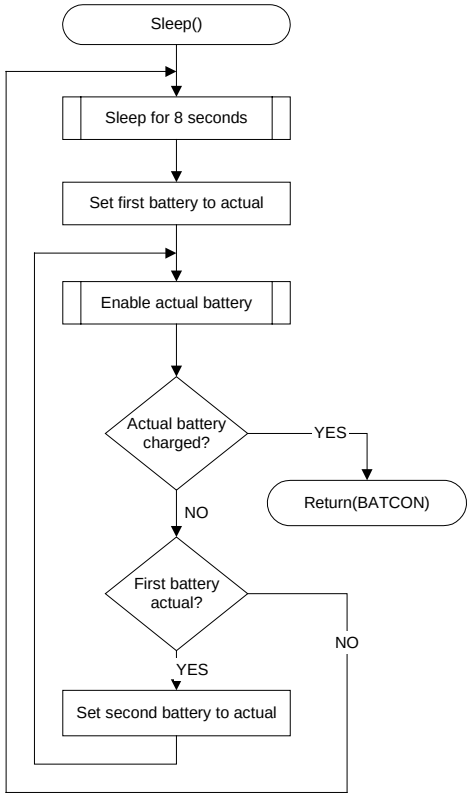



### 4.2.4 Sleep()

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 4-5.** Flow chart of sleep function

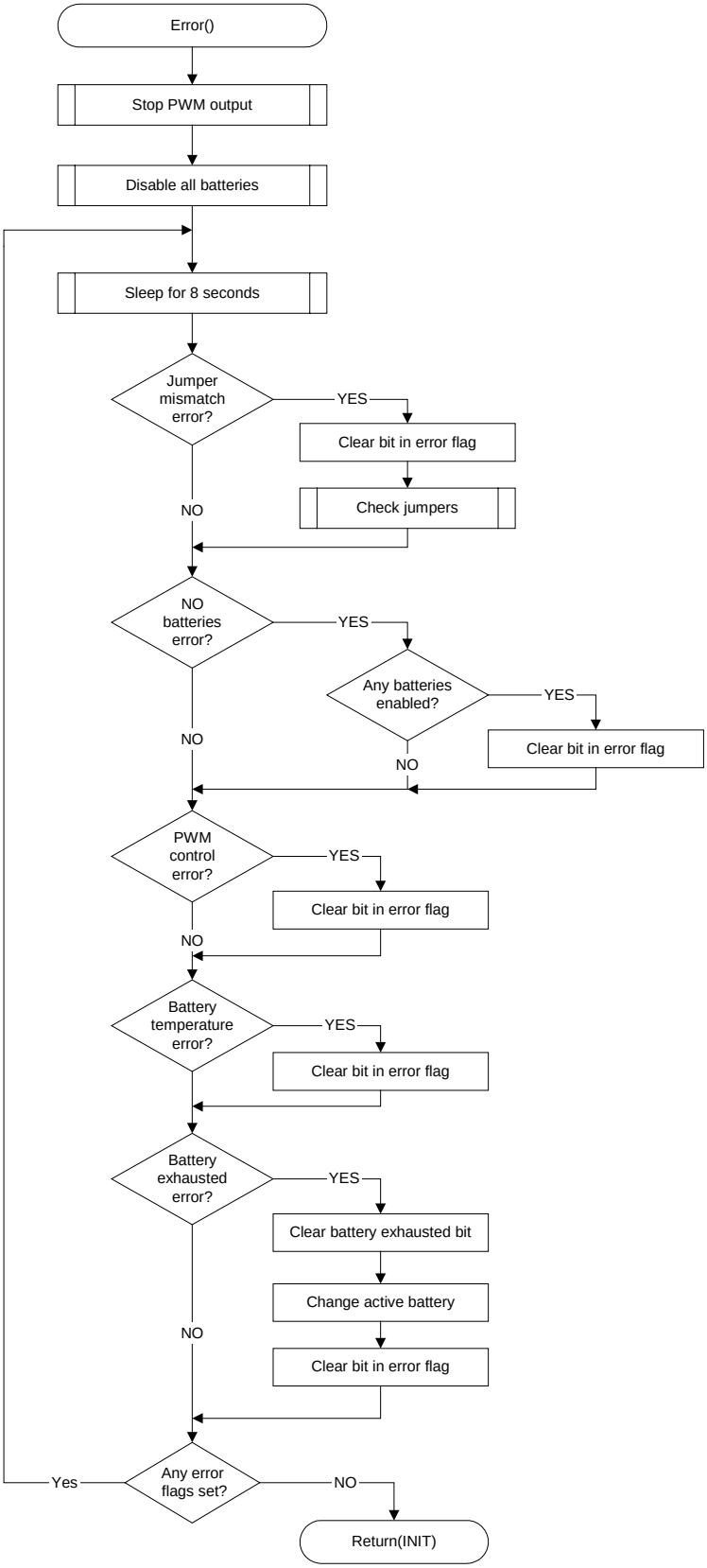



### 4.2.5 Error()

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 4-6.** Flow chart of error handler



## 4.3 Charging Functions

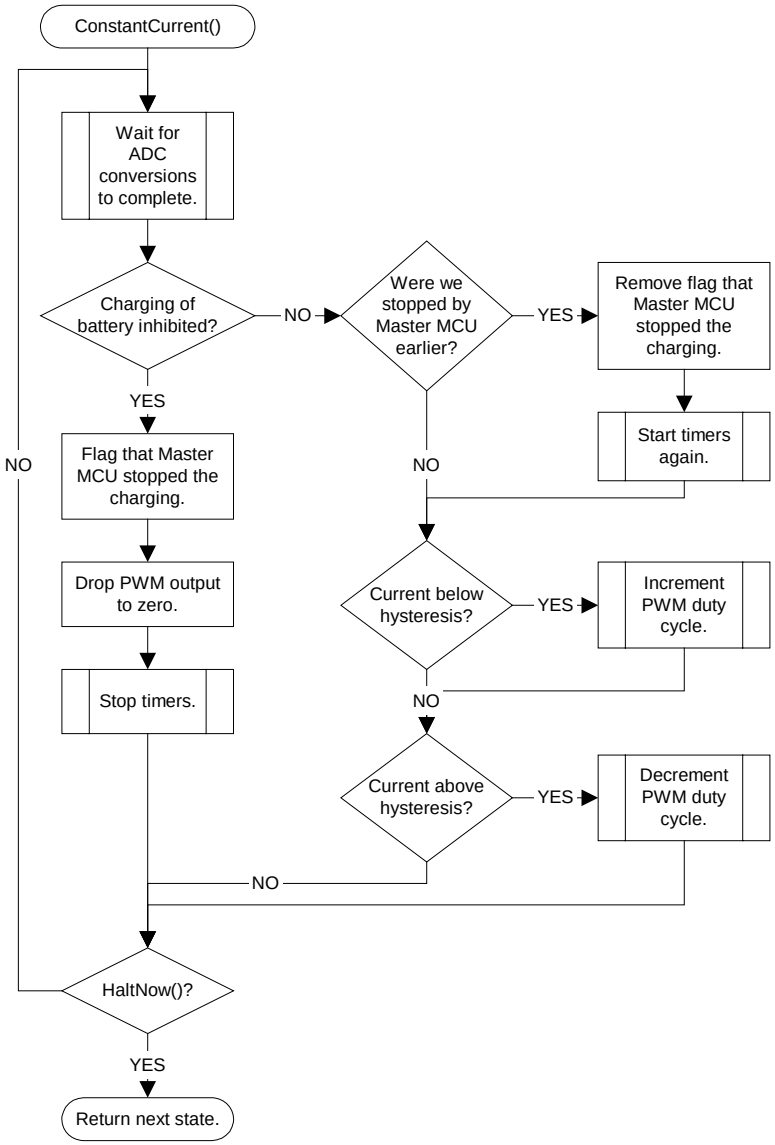
These functions are called by Charge() after all parameters have been set.

### 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.

**Figure 4-7.** Flow chart for ConstantCurrent()



### 4.3.2 Charge Halt Determination

Charge halt is determined by HaltNow(). This function is called by ConstantCurrent() and ConstantVoltage() 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 e.g. the time limit expires. 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 4-8.** Flow chart for HaltNow() part 1.

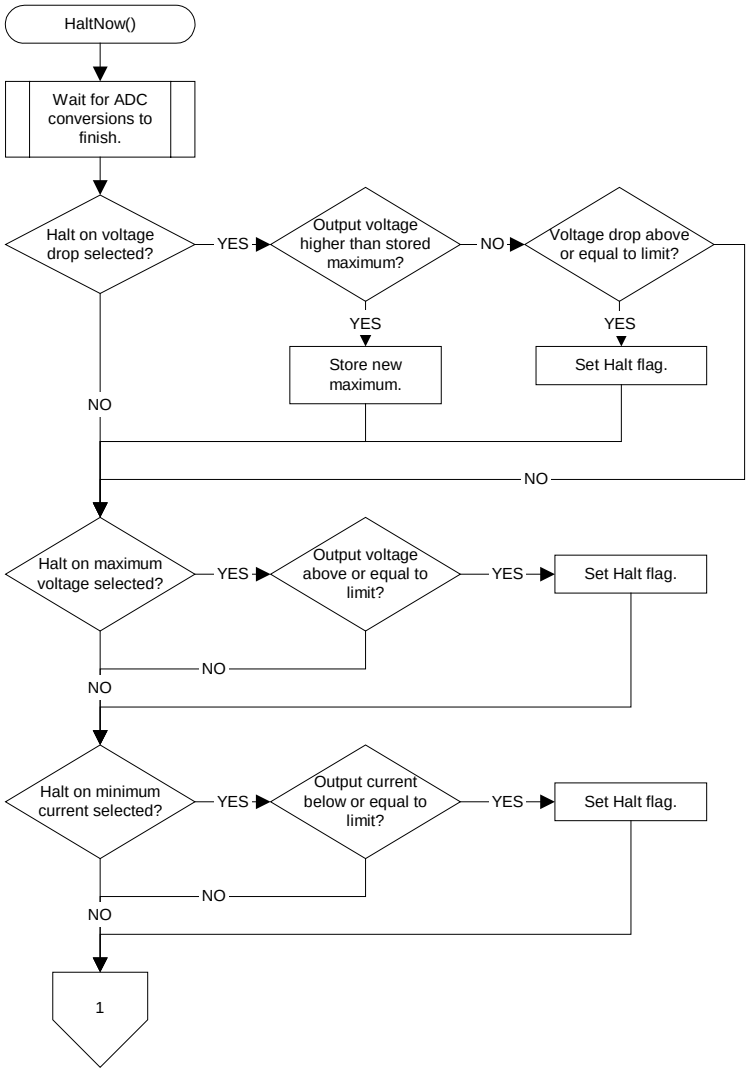


**Figure 4-9.** Flow chart for HaltNow() part 2

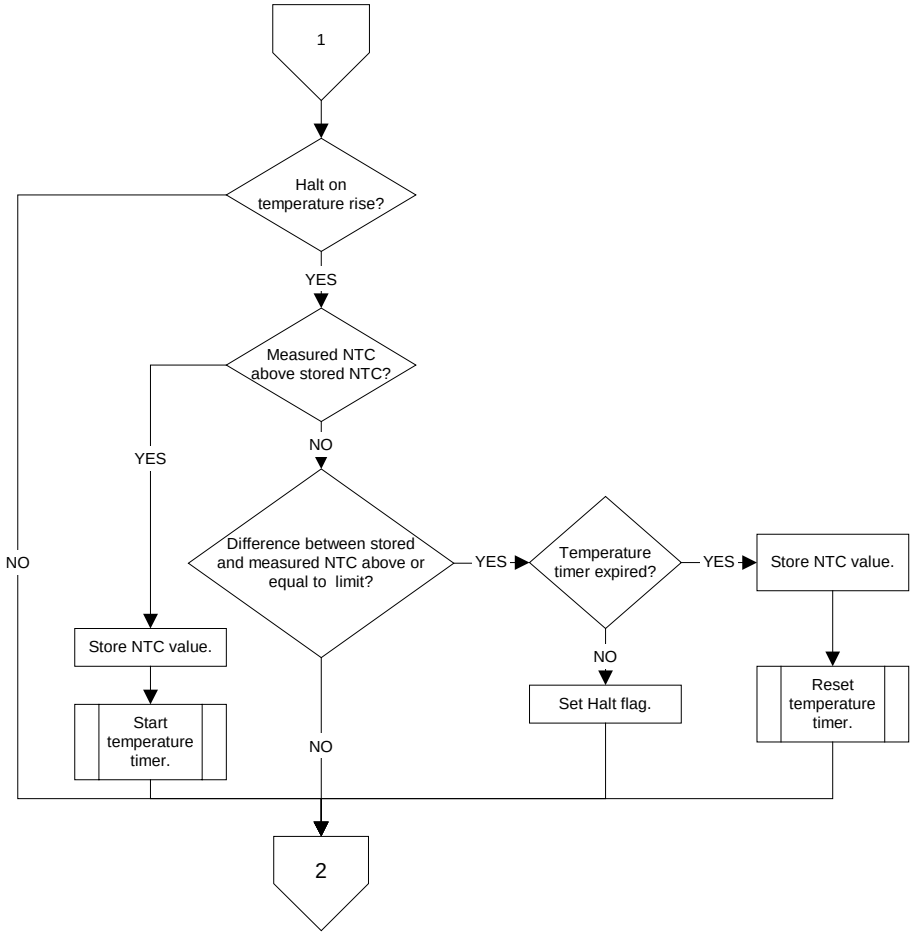


**Figure 4-10.** Flow chart for HaltNow() part 3

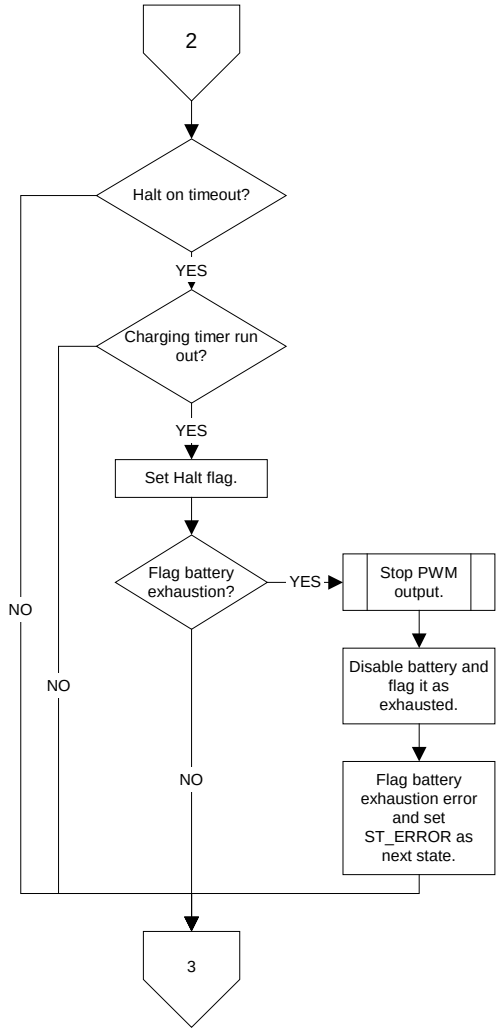
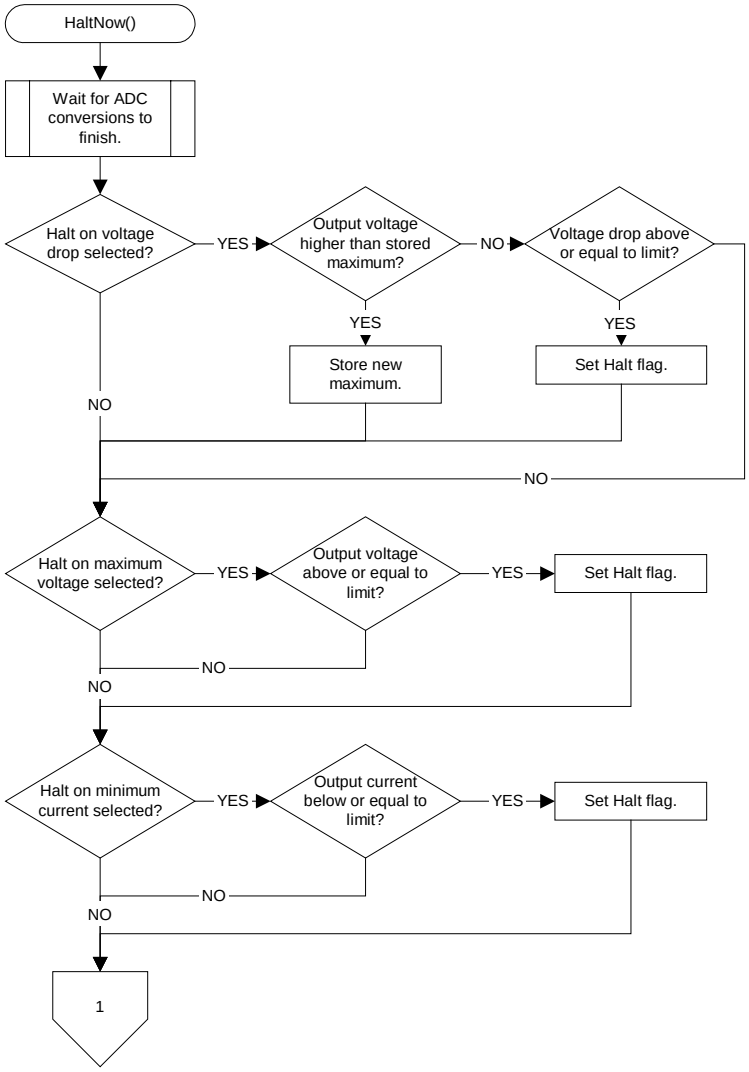


**Figure 4-11.** Flow chart for HaltNow() part 4



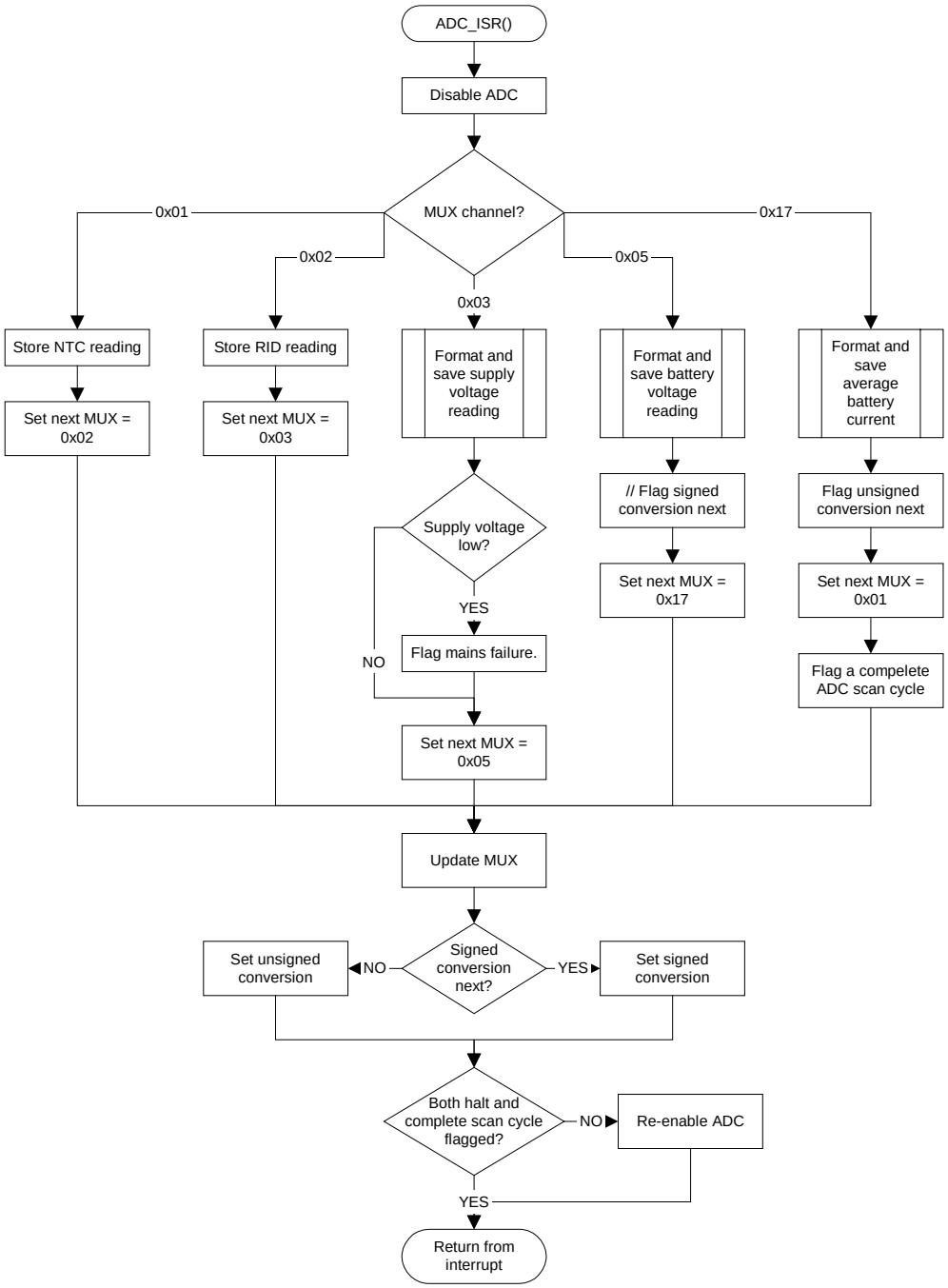

## 4.4 Other Functions

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.

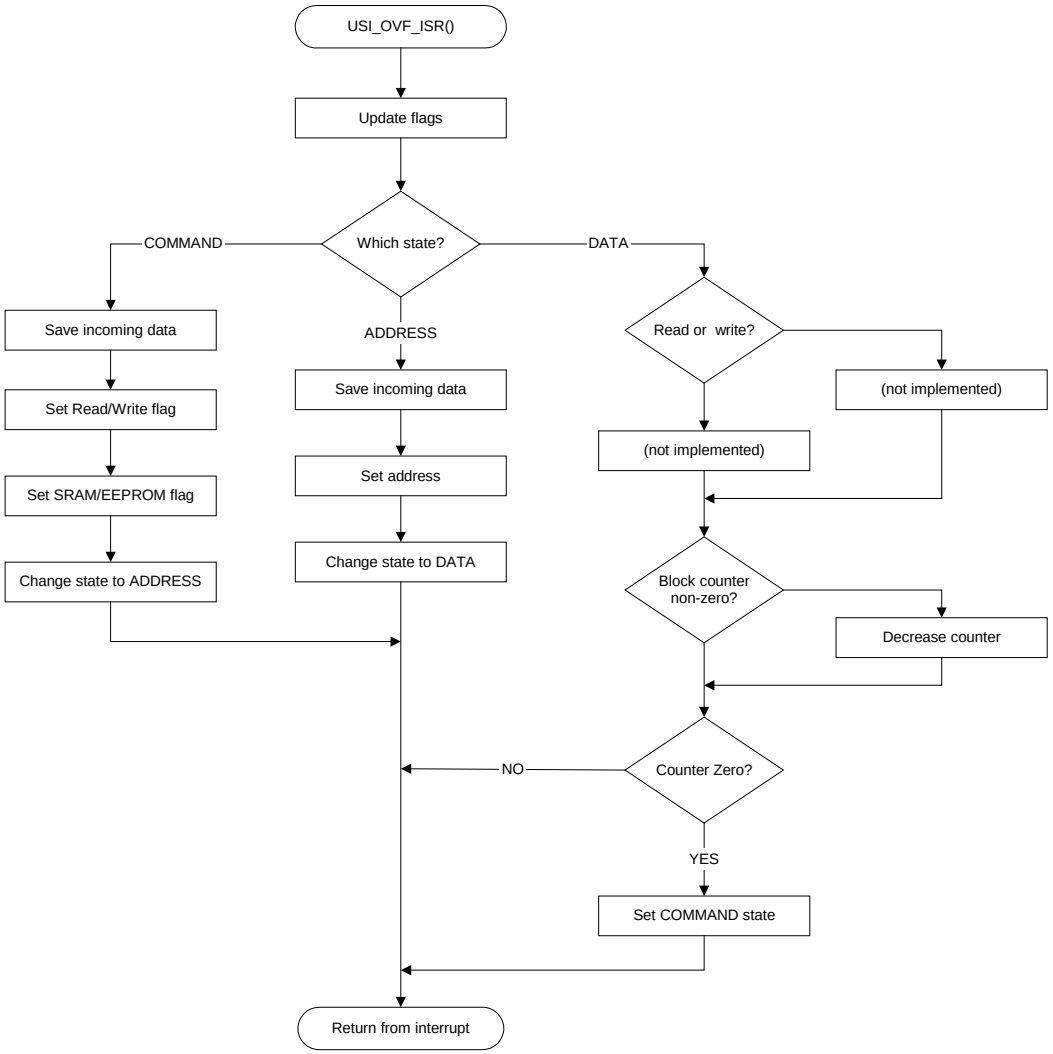
**Figure 4-12.** Flow chart of ADC interrupt service routine




### 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 4-13.** Flow chart of USI overflow interrupt service routine.



# 5 Quick Start Guide

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

### 5.1.1 Configuration

The most important compile-time constants are listed in the table below.

**Table 5-1.** Battery-related compile-time constants (see source files battery.c, battery.h and NIMHspecs.h)

| Label | Description |
|---|---|
| BAT_CELL_NUMBER | The number of cells in the battery. Each of the defined cell voltages gets multiplied with this, to define BAT_VOLTAGE_MAX, _LOW, _MIN and _PREQUAL. |
| CELL_VOLTAGE_SAFETY | 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. |
| CELL_VOLTAGE_MAX | The maximum voltage to which a cell should be charged. |
| CELL_VOLTAGE_LOW | The lowest voltage at which a cell is considered charged. Charging will start when voltage drops below this level. |
| CELL_VOLTAGE_MIN | 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. |
| CELL_VOLTAGE_PREQUAL | The voltage to which a cell should be charged to during prequalification. |
| BAT_TEMPERATURE_MAX | The highest battery temperature allowed. Charging will stop / not start if above this. |
| BAT_TEMPERATURE_MIN | The lowest battery temperature allowed. Charging will stop / not start if below this. |
| BAT_CURRENT_PREQUAL | Charge current during prequalification mode. |
| BAT_CURRENT_HYST | Charge current hysteresis. Current will not be adjusted when within plus or minus this value from target. |
| BAT_VOLTAGE_HYST | Charge voltage hysteresis. Current will not be adjusted when within plus or minus this value from target. |
| BAT_VOLTAGE_PREQUAL | Target voltage during prequalification stage. If this voltage is not achieved the battery will be marked as exhausted. |
| BAT_TIME_PREQUAL | Maximum amount of time to spend in prequalification stage. |
| DEF_BAT_CAPACITY | Default battery capacity. |
| DEF_BAT_CURRENT_MAX | Default maximum charge current. |
| DEF_BAT_TIME_MAX | Default maximum charge time. |
| DEF_BAT_CURRENT_MIN | Default cut-off charge current. |
| ALLOW_NO_RID | If defined, batteries without RID (or not matching the lookup-table) will cause the charger to use the battery defaults. Otherwise, charge is halted. |
| RID[].Low and RID[].High | Assume RID resistance match if value within these limits. |

| Label | Description |
|---|---|
| RID[].Capacity | Battery capacity for given RID. |
| RID[].Icharge | Charge current for given RID. |
| RID[].tCutOff | Maximum charge time for given RID. |
| RID[].IcutOff | Charge termination current for given RID. |
| NTC[] | Temperature look-up table. |

### 5.1.2 Compilation

Before compiling the code the following configurations should be made.

**Table 5-2.** Compiler configuration.

| Section | Tab | Field | Value |
|---|---|---|---|
| General Options | Target | Processor configuration | ATtiny861 (1) |
| | | Memory model | Small |
| | System | Data stack | 0x40 |
| | | Return address stack | 24 |
| | | Enable bit definitions … | Selected |
| C/C++ Compiler | Language | Require prototypes | Selected |
| Linker | Output | Format | Other: ubrof8 |
| | Extra Options | Command Line | -y(CODE)<br>-Ointel-extended,(DATA)=$EXE_DIR$\$PROJ_FNAME$_data.hex<br>-Ointel-extended,(XDATA)=$EXE_DIR$\$PROJ_FNAME$_eeprom.hex |

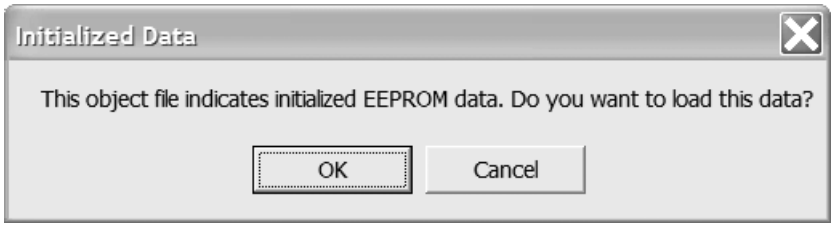
Notes: 1. Other options possible. See section 3.1.1 on page 5 for more information.

### 5.1.3 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 5-1.** Loading initialised data to EEPROM



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 5-3.** Non-default fuse bit settings

| Fuse Bit | Setting | Description |
|---|---|---|
| CKDIV8 | 1 (unprogrammed) | Do not divide clock by eight |
| CKSEL3…0 | 0010 | Use internal oscillator |

# 6 Known Limitations

Here are listed known limitations of the design.

## 6.1 Detecting Battery Presence

Currently, battery presence is detected by measuring the voltage on each battery connector. In case of fully discharged batteries, the voltage will sink to 0 Volts as soon as the relay connects it to the sensing circuitry, causing the charger not to detect them.

One solution is to charge such batteries for a couple of minutes with a power supply at about 0.1 C, to raise the voltage slightly.

It is also possible to apply a charging voltage, and flag the battery as absent if no current flows.

Note that fully discharging batteries may damage them, especially if they are multicell batteries.

## 6.2 Current Hysteresis and Voltage Drop

When charging just one or two cells, the current hysteresis may cause false full charge detection due to a voltage drop. Narrowing the hysteresis may help, but keep in mind that the Buck converter output is not of infinite resolution.

## 6.3 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).

## 6.4 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 about 14.7 kohm.

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.

### 6.5 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.

## 7 References


1. “What’s the best battery?”. Retrieved August 1, 2007, from Battery University: http://www.batteryuniversity.com/partone-3.htm

2. “AVR451 - BC100 Hardware User's Guide”. Available from Atmel web site: http://www.atmel.com/products/avr/

3. “ATtiny261/461/861 Data Sheet”. Available from Atmel web site: http://www.atmel.com/products/avr/

4. “ATtiny25/45/85 Data Sheet”. Available from Atmel web site: http://www.atmel.com/products/avr/

5. “Duracell Ni-MH Rechargeable Batteries Technical Bulletin”. Retrieved August 1, 2007 from Duracell: http://duracell.com/oem/Pdf/others/TECHBULL.pdf

6. “Handbook of Batteries, Third Edition”, from McGraw-Hill. http://www.mhprofessional.com/product.php?isbn=0071359788

7. “Charging nickel-based batteries” Retrieved August 1 2007, from Battery University: http://www.batteryuniversity.com/partone-11.htm

## Headquarters


***Atmel Corporation***
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600


## International


***Atmel Asia***
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

***Atmel Europe***
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

***Atmel Japan***
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581


## Product Contact

***Web Site***
www.atmel.com

***Technical Support***
avr@atmel.com

***Sales Contact***
www.atmel.com/contacts

***Literature Request***
www.atmel.com/literature




8098A-AVR-09/07