AVR453: Smart Battery Reference Design

8-bit AVR® Microcontrollers

Application Note

PRELIMINARY

Features

- **Support for up to 4 Li-Ion series-connected battery cells**
- **Battery protection by dedicated Hardware**
  - **Deep under voltage protection**
  - **Over-current protection during charging**
  - **Over-current protection during discharging**
  - **Short circuit protection**
- **Charging and discharging current monitoring with 18 bit ADC**
  - **Automatic Precharging after under-voltage situations**
  - **State-of-Charge and State-of-Health status**
- **SMBus communication**
  - **Full smart battery SMBus support**
  - **Support for In-System Programming through SMBus**
  - **Support for AES encrypted Firmware updates**

1 Introduction

Rechargeable Lithium-Ion (Li-Ion) batteries are widely used in portable electronics such as cell phones, digital cameras and laptop computers. This is mainly due to the high energy to weight ratio of these batteries. Maximizing the lifetime and energy storage of Li-Ion batteries requires careful monitoring and control of the charge and discharge cycles. Incorrect use may even pose a threat to safety, as Li-Ion batteries can explode under extreme conditions. For these reasons intelligent batteries – smart batteries – have been introduced.

The Atmel ATmega406 AVR microcontroller has been created with smart battery applications in mind. The feature set includes high accuracy ADCs, a TWI interface for SMBus communications, as well as independent hardware features that can protect the battery from incorrect use. This application note describes the implementation of a smart battery using the Atmel ATmega406 microcontroller.

Figure 1-1. Smart battery from the author's laptop

Rev. 2599B-AVR-09/05

2 Scope of implementation

The intent of the software associated with this application note is to provide an infrastructure for dealing with issues that will be faced when designing a battery pack. Although a functional battery pack has been implemented, there are likely many customizations and feature enhancements that can be made. The designer is encouraged to take what has been provided and personalize it.

The application note describes how the ATmega406 capabilities are employed to achieve the functionality needed when implementing a smart battery.

The application note software has not implemented battery authentication, which is desired in applications where the aftermarket represents significant revenue. This is used to ensure that only an “original” battery can be used with the product. Authentication is discussed further in section 6.7.

A Table of Contents is found on page 35.

Warning!

Incorrect handling of Li-Ion batteries poses a safety hazard: If Li-Ion batteries are mistreated they can explode. Use caution when dealing with any aspect of the design that may adversely impact safety; make sure you fully understand the behavior of the hardware and the software as a system.

3 Release Notes for preliminary release of AVR453

Note that this document and the source code are preliminary. This release targets ATmega406 rev E, older revisions of ATmega406 are not supported. For release notes, please check the Release Notes section in the doxygen documentation (`readme.html`), included with the source code.

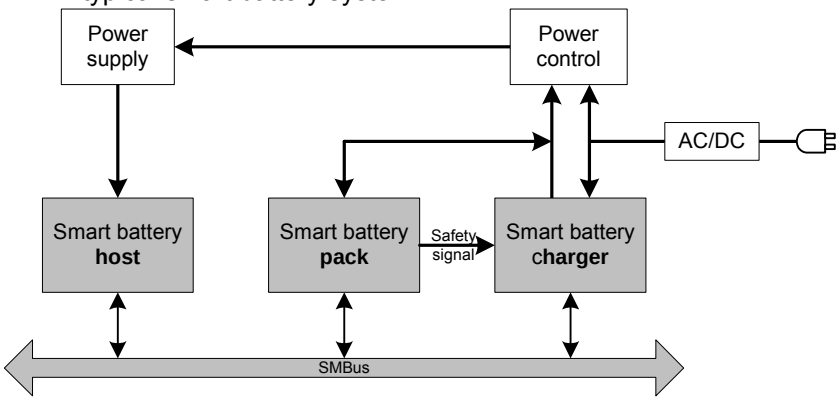
Further, please refer to Table 5-2, to see which SMBus commands that can be expected to respond correctly without modifications to the code.

4 Theory of operation

Smart battery systems consists of three elements:

- Smart battery host
- Smart battery pack
- Smart battery charger

Figure 4-1. A typical smart battery system



The smart battery host draws power from the smart battery pack (or just 'smart battery') and can obtain information about type, brand, remaining charge status and much more. Communication between the Host and the Battery is based on the System Management Bus (SMBus). The smart battery charger is a charger that can adapt its output based on the requests from the connected smart battery pack. This information is either transmitted directly from the smart battery to the charger, or retrieved directly from the smart battery pack by the charger, through the SMBus. The safety signal communicates critical errors directly from the battery to the charger.

More information about the smart battery system is available in the Smart Battery System Specification [1].

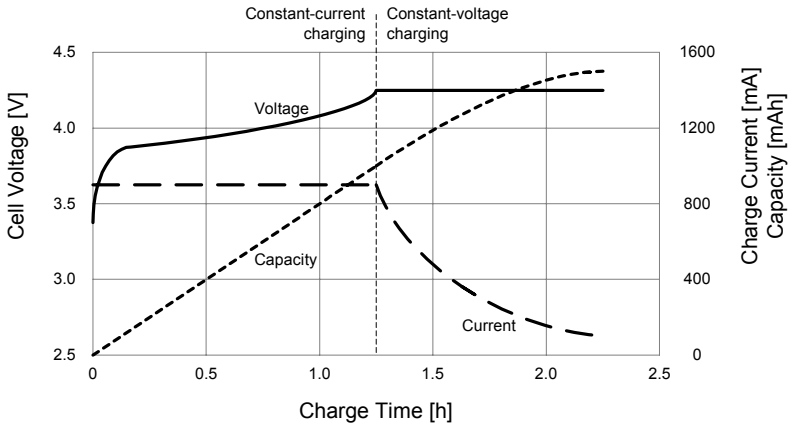
4.1 Li-Ion Battery technology

Li-Ion batteries are among the highest energy density cells on the market today, but they require careful management to yield optimum life. Specifically, overcharging and over-discharging are to be strictly avoided. Additionally, as with many battery types, excessive discharge currents can overheat the cell. Due to the lithium content of the cell, overheating is particularly dangerous and must be avoided. Further, the temperature of the cells affects the charge capacity and must be taken into account during charging, discharging and capacity estimation. By using the highly integrated ATmega406 device, component count and hence cost can be kept to a minimum while providing highly accurate charge estimates. Additionally, the presence of both EEPROM and Flash on the ATmega406 permits storing of battery history information such as temperature and current extremes that may aid in failure analysis of defective packs.

4.1.1 Charging profile of Li-Ion batteries

A typical charging profile for Li-Ion cells is shown in Figure 4-2. In today's portable equipment, having the shortest possible charging time is often a key requirement. Although less-optimal charging methods can be used, for fastest charging generally a constant-current charge source is required until the cell reaches a defined threshold voltage. The remaining charge is supplied by use of a constant-voltage charging source. This is referred to as the Constant Current – Constant Voltage charging method (CC-CV).

Figure 4-2. Typical charging profile for a Li-Ion battery



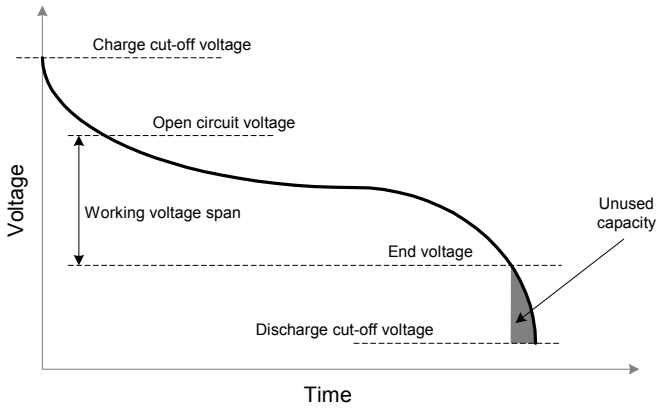
It is important that the supplied current and voltage do not exceed the manufacturer's specifications for the cells, or overheating and cell rupture may result. The ATmega406 monitors pack current and voltage independently in hardware and can disconnect the pack from the charger in the event of over-current or over-voltage. Additionally, if the designer incorporates thermal sensors the ATmega406 can sense temperature and request the charger to reduce the charging current or voltage if needed. The ATmega406 device includes an on-chip temperature sensor to provide a low-cost alternative to external sensors or to serve as a backup in the event of an external sensor failure.

A key in providing accurate state-of-charge estimates is the ability to monitor precisely the charge and discharge currents. The ATmega406 device includes a high-accuracy Coulomb Counter ADC that provides both high sensitivity and high resolution for this purpose.

4.1.2 Discharging Li-Ion batteries

Discharge of Li-Ion cells must be terminated when the cell voltages reach a defined lower limit. Discharging below this point will cause a structural change within the cell and reduce its capacity permanently. Additionally, excess current during discharge will overheat the cell and may cause a mechanical rupture.

Figure 4-3. Typical discharge profile for a Li-Ion battery.



To make best use of the available cell capacity, it is crucial that the cell voltage can be accurately measured to allow operation down to as low voltage as possible without going below the cell's discharge voltage limit. By having the bottom limit of the working voltage span as close as possible to the discharge cut-off voltage, the unused capacity is minimized (see Figure 4-3). For this reason, it is advantageous to have a high-accuracy ADC to measure the cell voltages. It is also critical to have as little series resistance as possible in the system to prevent unnecessary voltage drops. With the redundant safety features present in the ATmega406, it may be possible to eliminate one or more stages of safety circuitry from the pack, thus reducing the losses and increasing the pack output. Additionally, it is important to use a low-resistance device for current measurement. The reference design uses a 5mΩ resistor for this purpose and is able to measure current flow with a resolution better than 1mA.

4.1.3 Cell balancing

Cell balancing is the technique of adjusting the voltage of the cells in a series-connected stack to match each other. If cells have different charge and thus voltage, the cell with the highest voltage determines the charge termination point, and the cell with the lowest voltage determines the pack discharge termination point. Thus, any deviations between cells will effectively double the error by affecting both the charge and the discharge terminations. At manufacturing time, pack vendors typically match cell capacities very closely. However, due to normal manufacturing variations, after tens or hundreds of charge/discharge cycles even cells that were closely matched will diverge in terms of their operating voltage and remaining capacity. Another possibility is a weak cell, which will charge and discharge more quickly than the others in the pack due to low capacity. This forces early termination of the charge and discharge cycles, and may thus stop charging before the other cells are completely charged.

Voltage variations between cells should be kept as small as possible, preferably under 5mV, but it should also be noted that system noise in measuring cell voltages, such as may be generated by short current spikes which reduce the apparent cell voltage due to internal resistance of the cell, must be taken into account when deciding if the cells need to be balanced.

The ATmega406 device includes cell-balancing FETs across the cell voltage monitoring pins. Thus, balancing always involves reducing the voltage and charge of the highest voltage cell. These devices are rated for 2mA typical current. Although this does not seem large enough at first glance, it is adequate for balancing. The current through the balancing FET is limited by the series resistors in the filter network on each of the differential cell voltage inputs (please refer to the datasheet). Since the FET essentially shorts across the input, *it is not possible to perform accurate voltage measurements on any of the cell voltage inputs while any one of the cell balancing FETs is enabled*, as the filtering networks interact between channels to a small extent.

If cell balancing is performed during charging the balancing FET allows a small amount of the charge current to 'bypass' the cell, thus it does not receive as much charge as the other cells in the stack. During discharge, balancing increases the discharge rate for the cell. Balancing is only performed on one cell at a time, and it should always be the cell with the highest voltage that has its discharge rate increased.

4.2 Smart battery definition

There are several types of smart batteries, some smarter than others. The simplest form of a smart battery provides information about the battery technology and charge algorithm. The definition provided by the Smart Battery System Forum states that a smart battery must at least be able to provide State-of-Charge information.

A fundamental need for Li-Ion batteries is short-circuit protection. More sophisticated packs also include enhanced safety mechanisms to prevent over-charging, over-discharging, over-temperature and other conditions that are dangerous or could adversely affect battery longevity.

4.2.1 State of Charge

It is helpful to define the scope of functionality of a smart battery, both in terms of what it does as well as what it does not do. Smart battery technology provides means to track the State-of-Charge (SoC) of the battery by means of both hardware and by algorithms, which predict cell behavior. The smart battery is therefore able to calculate optimum voltage and current and send requests to the smart battery charger during the charging cycle.

Further, the smart battery also enables the Host system to manage its power usage so as to get maximum benefit from the remaining charge. Accurate measurements of cell voltages and charge/discharge currents are the basis of any prediction, so the ATmega406 device provides high-accuracy measurements and factory calibration values. The SMBus protocol defines a number of commands that provide this information to the Host system.

4.2.2 State of Health

A battery pack's state of health is not a measure of its state of charge, but rather its ability to accept and retain a charge as well as its current capacity. Due to aging, number of charge/discharge cycles and other factors, a cell's capacity will naturally diminish over its lifetime. It is useful to be able to assess the condition of the battery pack so that it can be determined if and when a replacement may be required. The ATmega406 device is capable of providing enhanced measurements and performing user-defined computations to aid in this determination. Pack capacity is continuously recalibrated in the supplied software whenever the pack reaches the full charge or full discharge state. Additionally the fast-responding Voltage ADC peripheral, when combined with the Coulomb Counter ADC, makes cell impedance measurements possible. The on-chip EEPROM also contributes by permitting the permanent storage of historical data.

4.3 Smart batteries and SMBus

The SMBus is the protocol used with smart batteries. This Bus and protocol architecture provides a means for keeping hardware costs low while also providing flexible functionality in a modular way. SMBus is a protocol that allows multiple nodes to respond to unique addresses. The Protocol is designed to handle multiple master devices being connected to the bus (arbitration control) and to ensure that a node will never lock the SMBus. The integrity of data can be verified by using Packet Error Checking. Details and specifications for SMBus can be found at www.smbus.org.

An SMBus device can provide manufacturer information, tell the system what its model/part number is, save its state for a suspend event, report different types of errors, accept control parameters, and return its status. A smart battery can manage

its own charging, report errors, inform the Host of low-charge conditions, predict remaining run-time, provide temperate, voltage and current information and continuously self-correct to maintain prediction accuracy. The SMBus specification also allows for five separate manufacturer-defined commands. One of these is used in this reference implementation for performing in-system programming of the ATmega406 device over SMBus. Another command is used to initiate, delete or check status of calibration.

4.4 A Very smart battery controller – ATmega406

The features of the ATmega406 that pertain especially to use in smart battery applications are as follows:

- Two Wire Interface (TWI) for SMBus communications
- High Resolution, high sensitivity ADC (referred to as a Coulomb Counter ADC)
- Multi-channel 12-bit voltage-measurement ADC (referred to as VADC)
- High-accuracy, calibrated voltage reference
- CPU-independent battery protection circuitry
- Integrated on-chip calibrated temperature sensor
- High-voltage-capable input and output pins
- High-voltage FET drivers for external charge, discharge and precharge FETs
- Integrated Cell-balancing FETs
- Wake-up timer
- Independent Watchdog timer
- On-chip low quiescent current voltage regulator

4.4.1 Two Wire Interface and SMBus

The communication mechanism of a smart battery system is the SMBus, which is fundamentally based on the Two-Wire Interface (TWI). One primary difference from TWI is that the SMBus specifies a minimum clock speed of 10kHz and a SMBCLK low timeout of 35ms, mainly to identify Slave-device faults and to allow recovery from such lockup conditions.

Another difference is that SMBus devices are expected to identify and report flaws in the communication on the fly. Two mechanisms are provided in the SMBus specification: the flaw is signaled either by withholding the ACK after the flawed byte is received, or by holding the clock line low for more than 25ms. This latter is the mechanism that is employed in this design.

Additional hardware has been provided to detect when the battery pack has been removed or reinserted into the target system. This Bus Connect/Disconnect function includes a timed filter to delay the indication of pack removal (Disconnect) in accordance with the SMBus specification.

4.4.2 Analog to digital converters

The ATmega406 includes two separate ADCs: the 18-bit Coulomb Counter ADC (CCADC) and the 12-bit Voltage ADC (VADC). Both use an internal high accuracy calibrated voltage reference.

The CCADC is used to very accurately measure the current that flows in and out of the battery pack, enabling reliable monitoring of the battery's SoC (state of charge).

Factory calibration of the CCADC offset must be performed to ensure maximum accuracy.

The VADC is used to monitor the individual cell voltages, the chip temperature from the internal temperature sensor, and can be used to monitor the temperature of the batteries.

More details about the capabilities and the operation of these ADCs can be found in the ATmega406 data sheet.

4.4.3 CPU-independent battery protection

The Battery Protection circuitry in the ATmega406 device provides CPU-independent hardware monitoring of the pack voltage and current levels, and can disconnect the pack from the load or charger to prevent critical failure by shutting off the two primary control FETs. Specifically, there are four conditions that are monitored: deep under-voltage, over-current during both charging and discharging, and short circuit. Short circuit is distinct from discharge over-current in that it reacts more quickly and has a higher threshold, whereas over-current protection is intended to watch for a continuous condition that exceeds the cell ratings.

4.4.4 High Voltage tolerant I/O

Many pins of the ATmega406 are capable of accepting high voltage without damage. This allows the ATmega406 to detect charger presence, monitor the pack and directly control the external high-current pack-protection FETs using very few external components.

The SMBus standard also requires an additional redundant Safety Signal line that can communicate critical pack state information to the charger. Since this line connects outside of the pack, a high-voltage line (PC0) is used for this purpose since it requires no special protection circuitry to guard it from the voltage on the pack's positive terminal.

4.4.5 Integrated cell-balancing FETs

The integrated cell balancing FETs in ATmega406 save valuable board space and help to achieve maximum pack performance for its entire life. Hardware ensures that no more than one balancing FET is enabled at any given time. The software in this design automatically disables and re-enables the balancing FET to yield correct readings when performing cell voltage measurements.

4.4.6 Low power operation

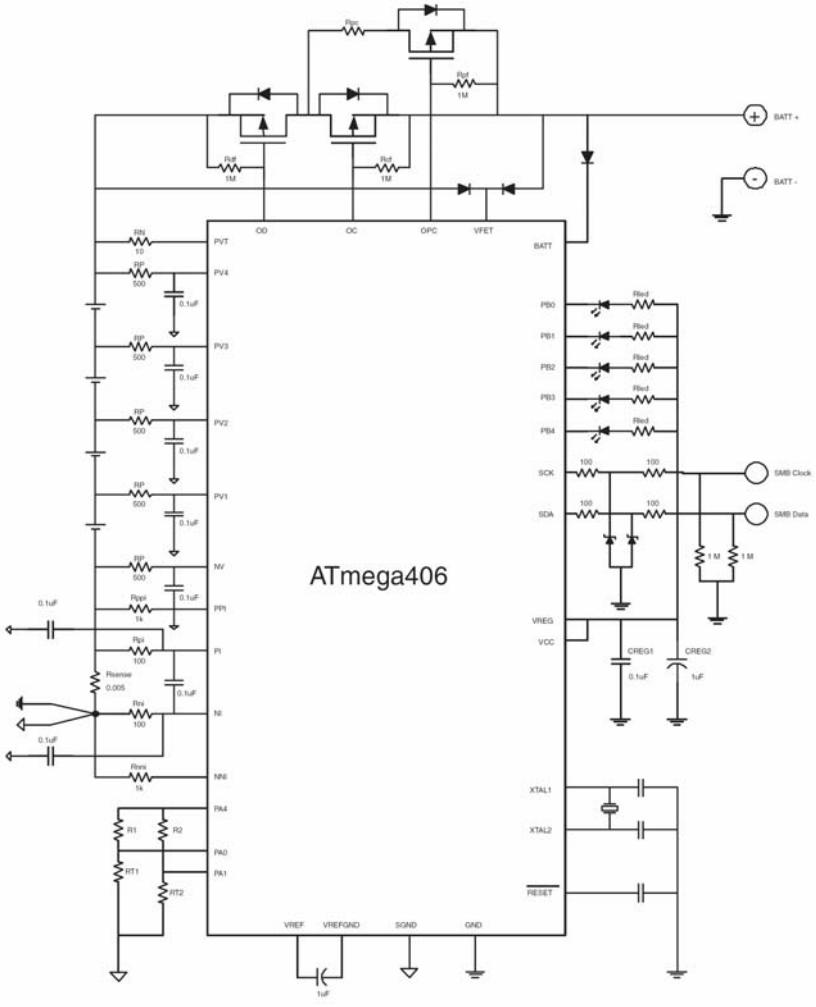
The combination of low power CPU clock modes, low quiescent current voltage regulator, low power Wake-up timer and oscillator, and low-power watch dog timer enables the ATmega406 to provide full functionality while drawing the absolute minimum power. Its low power modes ensure that pack shelf life is determined primarily by self-discharge rather than circuitry power consumption.

5 Implementation of smart battery

The software for this application is documented in the doxygen documentation (`readme.html`), which is downloaded with the source. Please see the Compilation info section first for details on complier(s) and settings. The software is targeted specifically to the ATAVRSB100 development board hardware, which can be purchased through Atmel sales channels and Atmel Distributors. The hardware is described separately in application note AVR454. Both this application note, and the source code for this implementation are available from the Atmel web site (http://www.atmel.com/products/avr/).

The following description of the implementation assumes that the reader is somewhat familiar with SMBus, smart battery and Rechargeable Batteries.

Figure 5-1. Typical operating circuit for the ATmega406 (from datasheet).



5.1 Overview of the software implementation

The software implementation consists of two separate projects: a bootloader, and the main battery application. The bootloader includes a subset (only enough code to perform programming tasks over SMBus) of the application program's SMBus code, but implemented in a polled manner rather than using interrupts (no interrupts are used in the bootloader). It also includes a small command interpreter and various low-level memory programming functions. Optionally, AES or other encryption algorithms may be added to protect the memory image during transfer.

The application program is more complex. In general, timer interrupts trigger the VADC to perform periodic scans, and computations are performed on the results to generate the data required to support the individual SMBus commands. The CCADC produces its own interrupts when its charge-monitoring data is ready, and calculations are performed on those results when they become available. SMBus communications are handled almost entirely by the TWI interrupt's state machine. Timer0 provides a periodic timer tick that allows up to eight generic timer event users, and also controls the duty-cycling and scanning of the LEDs.

The following table details the usage of each peripheral and the interrupts it uses.

Table 5-1. Interrupt Usage

Peripheral	Interrupt	Source File	Usage
Battery Protection	BP	safety.c	Shut down pack due to fault condition
External interrupts	INT0-3	Gpio.c	Unused
Pin Change	PCINT0	Smbus.c	PA6 monitoring SMBCLK for bus idle before Master Transmit mode
	PCINT1	Smbus.c	Unused
Watchdog	WDT	Timer.c	Software safety
Wake-up timer	WAKEUP	Timer.c	Periodic wake-up during pack 'Standby' operating mode
Timer1	TIMER1_COMP	Timer.c	Unused
	TIMER1_OVF	Timer.c	Unused
Timer0	TIMER0_COMPA	Timer.c	LED duty-cycle and multiplexing
	TIMER0_COMPB	Timer.c	PWM for main FETs; not presently used
	TIMER0_OVF	Timer.c	2.048mS timer tick
TWI	TWICD	Smbus.c	Pack insertion/removal notification
	TWI	Smbus.c	SMBus protocol state machine
VADC	ADC	Analog.c	Automatic scanning of analog sources
CCADC	CCCONV	Analog.c	Quick-response, non-accumulating current measurement
	CCREG	Analog.c	Triggers change from sampling to integrating current measurement
	CCACC	Analog.c	Indicates that a new Accumulated-current result is ready
EEPROM	EE_READY	n/a	Unused
Flash	SPM_READY	n/a	Unused

5.1.1 Normal Code Execution

The primary code execution begins with a hardware initialization. All peripherals and interrupt sources are set up, and interrupts are then enabled. SRAM variables are initialized, and normal execution flow begins.

In addition to normal activity invoked by interrupts, there are two primary mechanisms used to affect main loop code execution: generic timers and action flags.

The Timer0 Overflow interrupt provides eight generic software timers. When these timers expire, a corresponding function is called (from within the ISR, so its execution must be kept very short). This function may restart the timer, enable or disable a peripheral, or take other small actions. If a larger task must be performed, this function will assert an Action Flag.

Action flags are monitored in the main loop. When an action flag is asserted, a set of actions will be taken and the flag will be cleared. To provide longer duration timers than those available from the Generic Timers directly, one of the Generic Timer channels asserts an action flag, which in turn tracks longer intervals within the main loop. Based on these longer timeouts, other more infrequent actions are taken.

One such activity is the initiation of SMBus Master transactions as required by the SMBus standard. Another activity is the initiation of VADC scanning for measuring cell voltages, on-chip temperature, and thermistor reading. After a complete scan is done, the VADC conversion results are used to recalculate these parameters, and this in turn result in updates to the various SMBus variables.

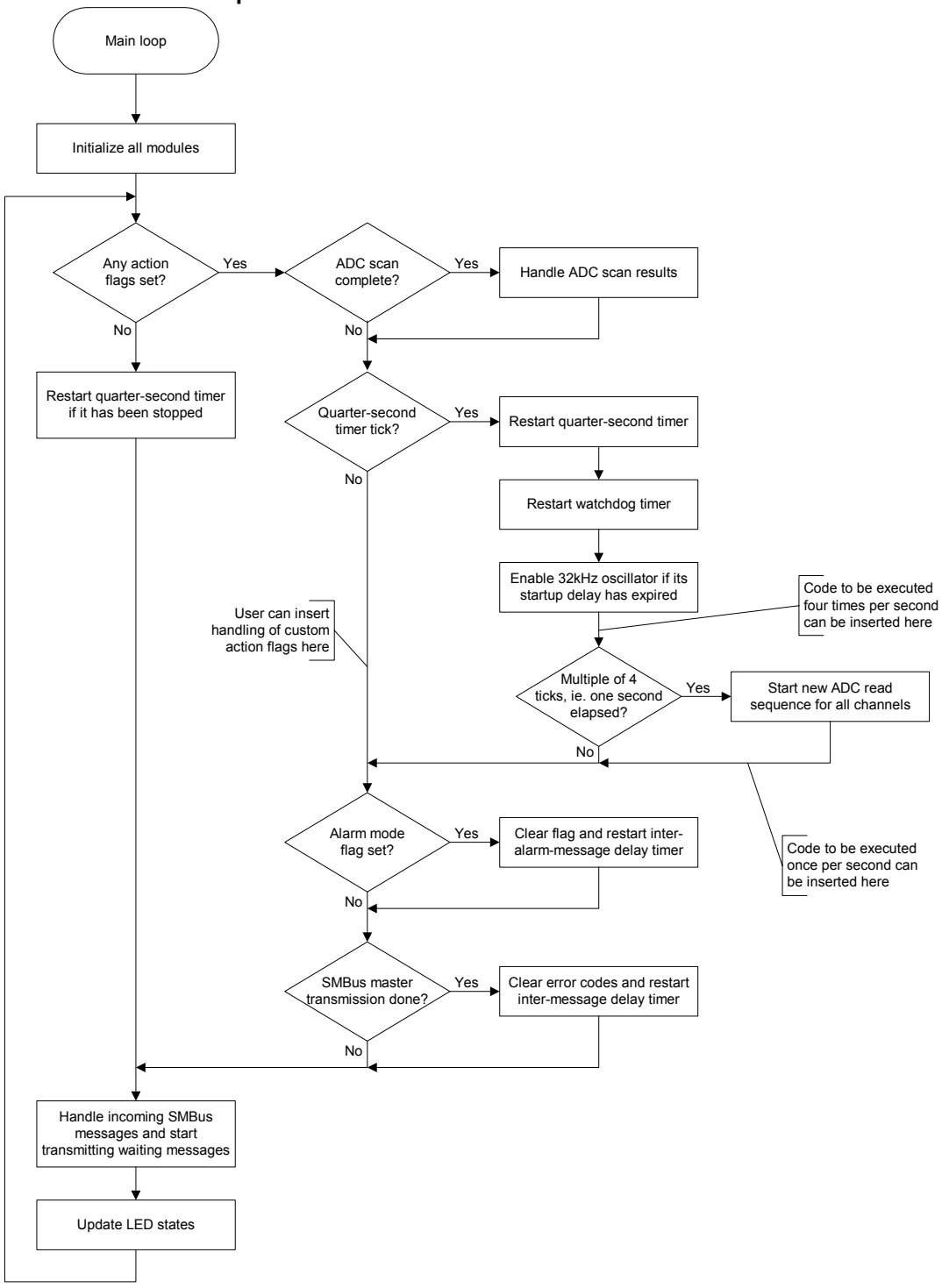
A mechanism has been established in the VADC Conversion Complete ISR to handle automatic scanning of all ten VADC channels. Besides taking readings, this scan automatically manages the disabling and re-enabling of the cell balancing FETs. Since filters are used on the cell inputs, the cell balancing FETs must be disabled early enough in the scan to allow the filters to reach full voltage before a cell reading is taken. The present implementation of the software scans all other channels before scanning the cells, thus allowing maximum recovery time for the filter's R-C time constant. See section 4.1.3 for more details on cell balancing theory.

The Coulomb Counter ADC (CCADC) utilizes several different interrupts, depending on the pack's operating mode. Foreground code determines which mode to operate the Coulomb Counter in, and this further influences the choice of CCADC operating modes for the purpose of power management. Additionally, the 32kHz Crystal oscillator supplies clock for the Wakeup timer, which is used in lower-power operating modes.

As shown in the flowchart in Figure 5-2, the application first initializes all modules and then enters an eternal loop. In every iteration, the loop first checks if any action flags are set by the interrupt controlled parts of the application and acts accordingly. Four times per second the quarter-second flag is set and the loop performs its regular tasks. Inside the quarter-second flag check, the user can insert custom code to be executed four times per second. Every fourth time the quarter-second flag is set, once per second, there is a similar place to insert custom code to be executed once per second.

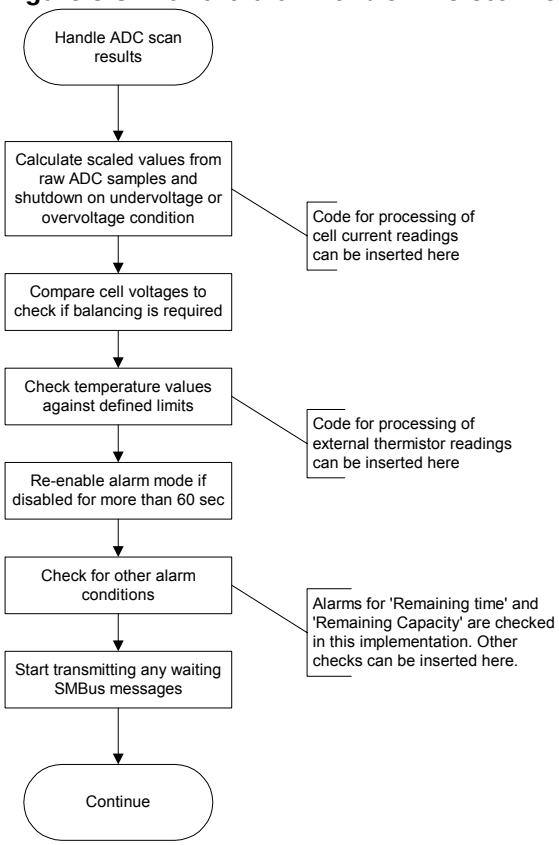
When starting the 32kHz oscillator, it takes up to 2 seconds for it to stabilize. Therefore, when the oscillator is started, a startup delay counter is initialized. The box 'Enable 32kHz oscillator...' in the flowchart takes care of updating this delay counter and enabling the real-time clock when the startup delay has elapsed.

Figure 5-2. Overview of the main loop



Most of the charge and discharge control, cell balancing and thermal checks are performed once per second, when a ADC scan sequence is finished. Updated ADC readings are then available and ready for processing.

Figure 5-3. Flowchart for “Handle ADC scan results”



To prevent alarm conditions from continuously generating alarm messages, an inter-alarm-message delay counter is implemented. It is set up so that persistent alarm messages are sent only once every 10 seconds. The SMBus specification also allows a Host to disable the transmission of AlarmWarning messages temporarily for up to 60 seconds. The “Alarm mode flag set” decision box in the flowchart of Figure 5-2 takes care of starting this timer when required. When this timer expires, AlarmMode is automatically re-enabled.

When a message has been transmitted on the SMBus interface, the “SMBus master transmission done” decision box takes care of starting an inter-message delay counter to leave some space between transmitted messages on the bus.

5.2 Battery Charging and Discharging

Three factors are involved in cell charge management: voltage, current, and temperature. By monitoring current, it is possible to predict how much of the cell’s capacity remains or how long a charging operation will take. Temperature may affect the charging parameters as well as the estimates of cell capacity, and is also a safety-monitoring tool.

Three variables are used to maintain state-of-charge information. They are:

- `RunningAcc`
- `MaxTopAcc`
- `MaxBottomAcc`

RunningAcc holds the present state of charge at all times. It is possible that this value may go negative, especially if the pack has not yet been fully calibrated. In normal operation, when the pack reaches full charge this value will be reset to the difference between its present value and the MaxBottomAcc value, and MaxToppAcc will be assigned the result as well, and MaxBottomAcc will be zeroed. A similar approach is used to recalibrate at full discharge.

While this reference implementation provides all the basic data gathering and reporting functions needed for a smart battery, software algorithms must be added to provide accurate capacity estimates as well as specific charging control methods.

5.3 Voltage ADC Results

Cell voltage is measured by the VADC. Up to four cells may be independently measured. Although the range of the VADC itself is 0-1.1V, cell voltages are scaled down in hardware by a factor of approximately 5.6, allowing a range of approx. 0-6.2V. With 12 bits available to cover an input range of 0 to 1.100V, the discernable voltage increment is 1.100V/4096 = 268µV, assuming no prescaling. With the prescaling, the step corresponds to approximately 1.5mV. To ensure maximum accuracy, calibration data for correct gain is generated during factory testing at 85°C and is stored in the Signature Row (please refer to the ATmega406 datasheet for details). A function call is provided to handle reading the desired value from this storage area.

In the reference software, a VADC conversion is started every second. When this conversion completes, the ADC_INT ISR will store the conversion result, then automatically switch to the next VADC input and initiate a new conversion. This cycle will continue until all inputs have been read. During this process, the cell balancing FETs will be disabled and then re-enabled when appropriate. The point where the balancing FETs are being disabled is chosen as a multiple of the 512µs VADC conversion time, and is currently set at 512 µs. With the current circuit values of 500Ω+500Ω and 0.1µF, this corresponds to more than 5 RC time constants and the error will thus be insignificant as long as the battery has negligible internal resistance.

5.3.1 Compensation of VADC results using Signature Row Data

The ATmega406 includes factory-determined calibration values. The function ReadFactoryCalibration() is implemented to read these values into SRAM at startup and to adjust the affected peripherals. Calibration values are available for the RC oscillators, Bandgap, on-chip temperature sensor and all four cell inputs.

In some cases, such as the four cell measurement channels, the factory values must be used as part of further calculations rather than directly adjusting the peripheral.

While the cell calibration values produce mV results as required for SMBus commands through the use of simple binary math, the SMBus specification requires temperature data to be produced in 0.1°K increments rather than whole degrees. As such, the calculations for deriving temperature from the on-chip temperature sensor are not as straightforward. The routine CalculateADCresults() performs all calculations using the available calibration parameters.

Equation 5-1 shows how the cell voltages are calculated from the raw ADC reading and the Voltage ADC Cell Gain (VADCCG) Calibration Word. The resulting values are given in millivolts.

Equation 5-1. Calculation to obtain correct cell voltage in mV

$$V_{cell}[mV] = \frac{ADCreading \cdot VADCCG_{Calib}}{2^{14}}$$

Equation 5-2 shows how the internal temperature is calculated from the raw ADC reading and the Voltage Proportional to Absolute Temperature (VPTAT) Calibration Word. The equation shows how to get the temperature in Kelvin and 1/10 Kelvin.

Equation 5-2. Calculation to obtain temperature in Kelvin and 1/10 Kelvin

$$Temp.\ [1^{\circ}K] = \frac{ADCreading \cdot VPTAT_{Calib}}{2^{14}}$$

$$Temp.\ [0.1^{\circ}K] = \frac{ADCreading \cdot VPTAT_{Calib} \cdot 10}{2^{14}}$$

5.4 Coulomb Counter ADC results

Current is monitored by the CCADC. The CCADC provides three main features: Accumulated Current measurement, Instantaneous Current measurement and detection of Regular Current condition. Each of these three modes has a dedicated interrupt. All three modes are used in the reference software. The Accumulate Current conversion result is the most accurate (18 bits including sign). The Instantaneous Current conversion result has lower resolution (13 bit including sign) but provides a new output every 3.9mS, allowing very fast response to system changes. This result is used to compute the `Current` and `AverageCurrent` parameters for SMBus reporting. When the connected system is operating in a very low power mode, it is advantageous to reduce the ATmega406 operating power as well so that it does not dominate the power consumption. In these cases it is permissible to disable the CCADC and accumulate estimated current rather than measured current. The Regular Current interrupt provides a mechanism to detect if the target system has switched to a higher power operating mode so that the CCADC can be re-enabled.

The Accumulate Current conversion produces a result of 17 bits plus sign. The actual range of the converter's input signal is specified to be $\pm V_{REF}$ / 5, or ±0.22V, and the step size of the converter is therefore (0.22V / 2^{17}) = 1.678µV. However, it is not recommended to use the full input voltage range of the ADC due to linearity issues near the upper end of the range. Therefore, the useable range has been specified as ±0.15V, which is approximately $0.22V / \sqrt{2}$. An example follows.

Assume a 5000mAh battery pack is used. The pack voltage is irrelevant as we are only here concerned with current: A charge current of 1A flowing through a 5mΩ sense resistor will yield 14,898.69 counts in the CCACC result registers each second. This result consumes 15 bits (14 plus sign). Since this pack can only produce that current flow for 1 hour, or 3600 seconds, the total accumulated result would fit in (15+12) bits, or 27 bits. Thus, a 32-bit signed integer could handle a battery pack of 200,193mAh when using a 5mΩ sense resistor. The maximum current allowed to flow could not be that high due to the input voltage range of the CCADC, so the current flow would have to be limited to 30A. Further, SMBus commands are limited to allowing only up to 32,767mAh, unless scaling factors are specified in the SMBus `SpecificationInfo()` command, so the result will easily fit by a factor of 6. Clearly, the use of a 32-bit value to hold the accumulated charge is more than adequate for today's laptop systems. A higher value sense resistor could therefore be

used and would yield even more resolution on low-current readings, specifically, up to $6 \cdot 5\text{m}\Omega$ could be used without overflowing the 32-bit accumulator while still allowing the maximum possible SMBus pack capacity.

Note that any CCADC offset that is present should be removed from each sample before accumulation. Since such offset may be influenced by temperature, additional algorithms may be required. Since the resolution of the offset is limited to the step size of the converter, for smaller currents the error of only being able to use an integer value for offset becomes a larger portion of the result. This fact may determine both the value of sense resistor chosen as well as the point where a change is made from using Accumulator mode to using periodically sampled or estimated measurements.

5.4.1 CCADC result scaling

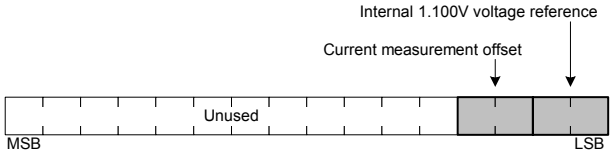
Since the CCADC accumulates steps of 1.678μV, which corresponds to 335.6μA through a 5mΩ resistor, the accumulated results must be adjusted to correspond to a 1mAh scale for reporting purposes. The scale factor is $(1/0.3356) \cdot 3600 = 10{,}727$. To convert from 0.3356mAs to 1mAh scale, the accumulated result must be divided by this number. To confirm, assume a 1A current is flowing for 1 hour. The accumulated counts will be 2979.7 per second, or 10,727,056 per hour. Dividing by 10,727 will yield 1000, which is in the mAh scale.

Performing a division of a 32-bit integer by a 16-bit integer will produce a 32-bit result. However, if the total value in the 32-bit accumulator is always lower than 336,582,624 ($2^{15} \cdot 10{,}727$) then the result of the division will not overflow. As discussed previously, SMBus commands have as an upper limit 32,767mAh, so this is not an issue when a 5mΩ sense resistor is used.

5.5 Customer calibration

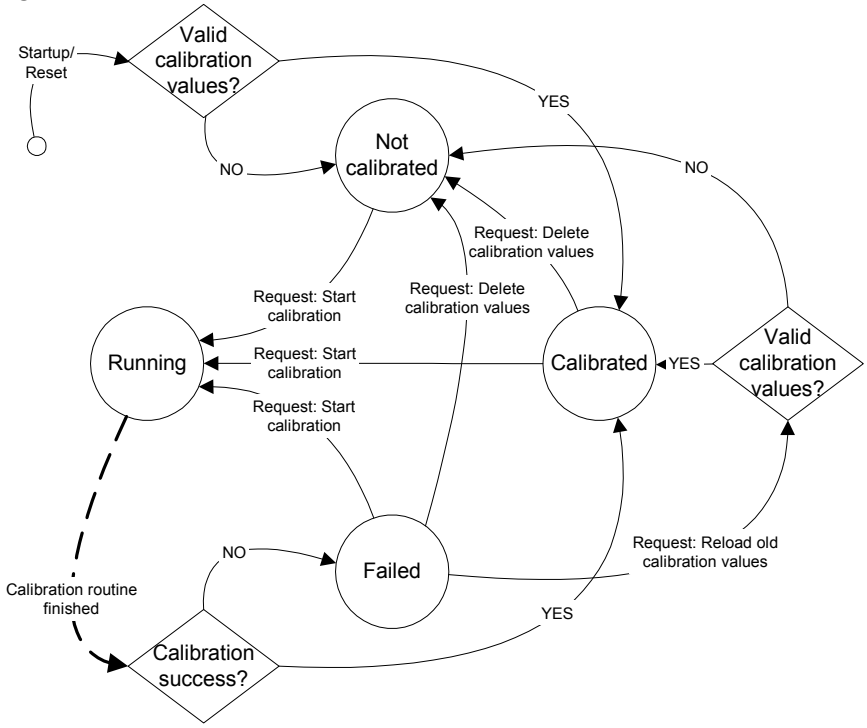
Using the SMBus command `OptionalMfgFunction4` the user controls calibration of the internal 1.100V reference voltage and the charge/discharge current measurement offset. The SMBus commands are described in section 5.9.1. The software maintains separate state machines for the two calibration operations. When using the `OptionalMfgFunction4` in a read operation, the current calibration states for both state machines are returned. However, when using the function in a write operation, the state request is stored and handled later in the main loop.

Figure 5-4. Calibration word usage



	Read state	Write request
0 0	Not calibrated	N/A
0 1	Running	Start calibration
1 0	Failed	Delete calibration
1 1	Calibrated	Reload calibration

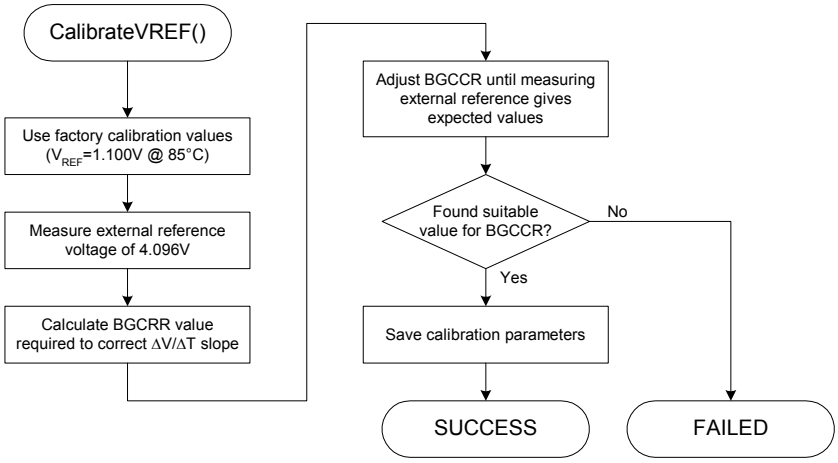
Figure 5-5. Calibration states



5.5.1 Calibrating the internal 1.100V Voltage Reference

Voltage reference calibration is implemented in the function `CalibrateVREF()` in the file `analog.h`. The function assumes that a highly accurate reference voltage of 4.096V is connected to analog input ADC0 before initiating calibration. If 4.096V is not present, calibration fails, but if a voltage close enough is present the VREF will be calibrated wrongly and all subsequent ADC measurements will be incorrect. If calibration fails, the voltage reference is reset to the factory calibration value, but the old calibration values are still in eeprom and can be reloaded.

Figure 5-6. Flowchart for `CalibrateVREF()`



Please refer to the ATmega406 datasheet for more details on the Bandgap Reference Calibration registers, BGCRR and BGCCR.

5.5.2 CCADC offset calibration

Calibration for current measurement offset, or CCADC offset, is implemented in the function `CalibrateCCoffset()` in the file `analog.h`. The function assumes that the current through the sense resistor has been zero for at least one second before initiation calibration. The result from the CCADC is then used to update the offset calibration values. If the measured offset is outside predefined limits, the calibration process fails and the offset calibration value is set to a default value of zero.

5.5.3 Storage of calibration values

If the calibration routine(s) are successful, the resulting values are stored in eeprom at the addresses given in `ee.h`. When the Atmega406 is reset it checks if there are valid values, and if so, uses them and updates the calibration state to reflect this.

5.6 Battery Protection

The Battery Protection Interrupt (BPINT) is used to indicate that a fault condition was detected by the Battery Protect hardware. Tripping the safety mechanism forces the pack into Power-Off Mode, but first a status flag is written to EEPROM to indicate the reason for entering Power-Off Mode, to aid in debugging. This is implemented in the routine `DoShutdown()`, and the Reason codes are defined in the file `pwrmgmt.h`. In this implementation, no error messages are sent on the SMBus when battery protection is triggered.

5.7 Pack Configuration

The header file `pack.h` and other related header files define critical parameters of the pack, such as over-voltage, under-voltage, over-current, and thermal parameters, as well as the number of stacked cells.

During debugging, it may be helpful to modify these values to prevent unintentional triggering of hardware and software error detection mechanisms. Note that if such mechanisms are tripped, the software will typically force the AVR into a Sleep mode, resulting in a loss of control over the CPU via JTAG until a Reset is issued.

5.8 LED Control

The OC0A PWM output is used to control the brightness of five external LEDs. These LEDs can be used for any purpose, but are typically used as charge indicators. Inside the Timer 0 Compare Match A ISR, a counter variable cycles between each LED's corresponding control signal, forcing the output pin low if the LED has been enabled in the LEDflags global variable. Thus, the battery capacity indicator function requires no mainline code for its operation, other than that of deciding which LEDs to enable or disable.

5.9 SMBUS Protocol Implementation

The reference software fully implements the required command set from Smart Battery Data Specification version 1.1. AlarmWarning messages (command 0x16), ChargingCurrent (0x15) and ChargingVoltage (0x15) are the only messages generated by the software in SMBus Master Mode. All other commands are handled by the pack as an SMBus Slave device. As a Slave, it will either accept data or provide data as a response to a command originated from a smart battery host or a smart battery charger.

All word-size data maintained for SMBus commands are kept in the union variable SV. The header file `smbus.h` includes two different ways to access this data, either through the `SMBvariables[][]` array which provides byte-level access, or the `SMBvar_int[]` array which provides word-level access. This approach reduces code size and improves speed when dealing with flags and byte-wide data.

Table 5-2 provides details on the default value of each of these variables.

Table 5-2. SMBus Commands and Default data values

Command (ID)	Data Direction & Size	Default Value	Data Source
ManufacturerAccess (0x00)	R/W Word	0x4060	Initialized at startup
RemainingCapacityAlarm (0x01)	R/W Word	`PACK_DESIGNCAPTYP`(1) / 10	From battery specs
RemainingTimeAlarm (0x02)	R/W Word	10	Per sbdat110, section 4.4.1
BatteryMode (0x03)	R/W Word	0	Per sbdat110, section 5.1.4
AtRate (0x04)	R/W Word	Calculated as needed	Per sbdat110, section 5.1.5
AtRateTimeToFull (0x05)	Read Word	Calculated as needed	Per sbdat110, section 5.1.6
AtRateTimeToEmpty (0x06)	Read Word	Calculated as needed	Per sbdat110, section 5.1.7
AtRateOK (0x07)	Read Word	Calculated as needed	Per sbdat110, section 5.1.8
Temperature (0x08)	Read Word	Calculated as needed	On-chip sensor
Voltage (0x09)	Read Word	Calculated as needed	VADC readings
Current (0x0A)	Read Word	Calculated as needed	CCADC Instantaneous
AverageCurrent (0x0B)	Read Word	Calculated as needed	CCADC Instantaneous, avg'd
MaxError (0x0C)	Read Word	Calculated as needed	Per sbdat110, section 5.1.13
RelativeStateOfCharge (0x0D)	Read Word	Calculated as needed	Per sbdat110, section 5.1.14
AbsoluteStateOfCharge (0x0E)	Read Word	Calculated as needed	Per sbdat110, section 5.1.15
RemainingCapacity (0x0F)	Read Word	Calculated as needed	Per sbdat110, section 5.1.16
FullChargeCapacity (0x10)	Read Word	Calculated as needed	Per sbdat110, section 5.1.17
RunTimeToEmpty (0x11)	Read Word	Calculated as needed	Per sbdat110, section 5.1.18
AverageTimeToEmpty (0x12)	Read Word	Calculated as needed	Per sbdat110, section 5.1.19
AverageTimeToFull (0x13)	Read Word	Calculated as needed	Per sbdat110, section 5.1.20
ChargingCurrent (0x14)	Read Word or Write to Charger	Calculated as needed	Per sbdat110, section 5.2.1
ChargingVoltage (0x15)	Read Word or Write to Charger	Calculated as needed	Per sbdat110, section 5.2.2
BatteryStatus (0x16)	Read Word or Write to Host	0x0080	Per sbdat110, section 4.4.1
CycleCount (0x17)	Read Word	0	Per sbdat110, section 4.4.1
DesignCapacity (0x18)	Read Word	Calculated as needed	From battery specs
DesignVoltage (0x19)	Read Word	Calculated as needed	From battery specs
SpecificationInfo (0x1A)	Read Word	0x0031	Per sbdat110, section 5.1.25
ManufactureDate (0x1B)	Read Word	Calculated at compile time	Per sbdat110, section 5.1.26
SerialNumber (0x1C)	Read Word	User-defined	Per sbdat110, section 5.1.27
Reserved (0x1D-0x1F)	Read Word	N/A	N/A
ManufacturerName (0x20)	Read String/Block	User-defined	Per sbdat110, section 5.1.28
DeviceName (0x21)	Read String/Block	User-defined	Per sbdat110, section 5.1.29
DeviceChemistry (0x22)	Read String/Block	"LION"	Per sbdat110, section 5.1.30
ManufacturerData (0x23)	Read String/Block	User-defined	Per sbdat110, section 5.1.31

Command (ID)	Data Direction & Size	Default Value	Data Source
Reserved (0x24-0x2E)	N/A	N/A	N/A
OptionalMfgFunction5 (0x2F)	R/W Block	N/A	Bootloader
Reserved (0x30-0x3B)	N/A	N/A	N/A
OptionalMfgFunction4 (0x3C)	R/W Word	N/A	Calibration state
OptionalMfgFunction3 (0x3D)	R/W Word	N/A	N/A
OptionalMfgFunction2 (0x3E)	R/W Word	N/A	N/A
OptionalMfgFunction1 (0x3F)	R/W Word	N/A	N/A

Notes: 1. Typical capacity in mAh for the battery pack. Defined in `pack.h`.

Local copies of all variables required by the SMBus command set are maintained in SRAM. In some cases, these values are treated as read-only; in other cases, the Battery pack, the Host, the Charger or any of these may modify the variables.

Some variables affect other commands and variables. For instance, the `CapacityMode` flag will affect all calculations involving mA vs. mW. For more information, see the Smart Battery Data Specification, section 5.1.4. The `AtRate()` function is part of a two-stage procedure for determining time remaining for either charging or discharging at a given rate. See the Smart Battery Data Specification, section 5.1.6 for more information on the `AtRate()` command. The `SpecificationInfo()` command contains bits that define scaling parameters for the pack voltage and current to allow very high capacity and high voltage packs. See the Smart Battery Data Specification, section 5.1.25 for more information on the `AtRate()` command.

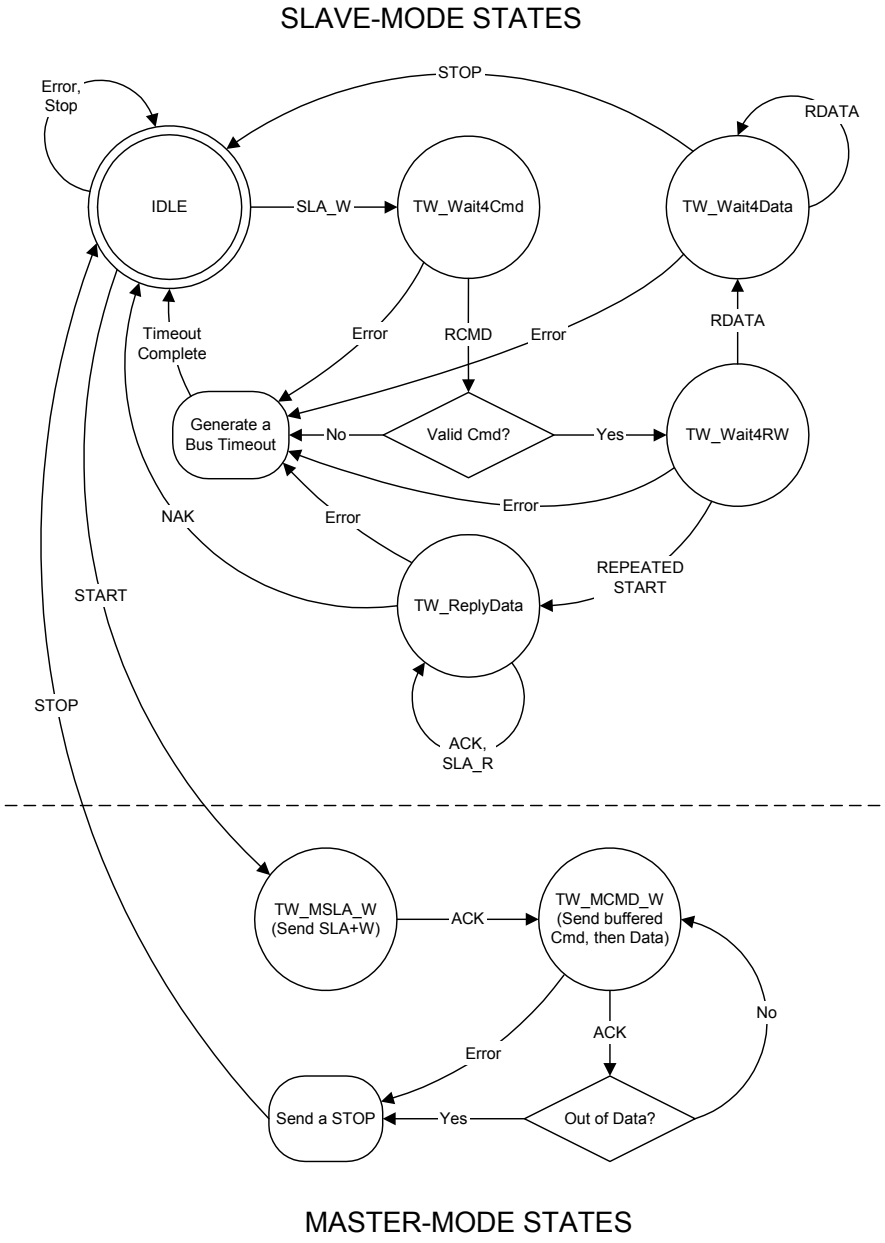
5.9.1 SMBus Slave Mode

Slave Mode SMBus communications are almost entirely interrupt-driven in the AVR. The ATmega406 TWI module is able to receive a Slave Address transmission from a Master device even if the system clock to the TWI peripheral is not enabled. Upon completion of the byte, an interrupt will be generated if enabled and if the received address matches that programmed in the TWAR register. This will wake the ATmega406 from all but the Power-Off Sleep Mode.

When the SMBus itself is inactive, as determined by if both clock (SCL/SMBCLK) and data (SDA/SMBDATA) lines being low for more than 2 seconds, the TWI Bus Connect/Disconnect Interrupt will alert the smart battery that it may switch to Power-Save mode (please refer to section 5.10 for operation modes). Likewise, when bus activity resumes, the same interrupt will immediately awaken the ATmega406 to restore operation.

Once the system is active, upon receiving a match to the pack's address the TWI interrupt service routine will track the protocol state by means of a state machine. The diagram of Figure 5-7 defines the behavior.

Figure 5-7. Flowchart for TWI Interrupt Service Routine



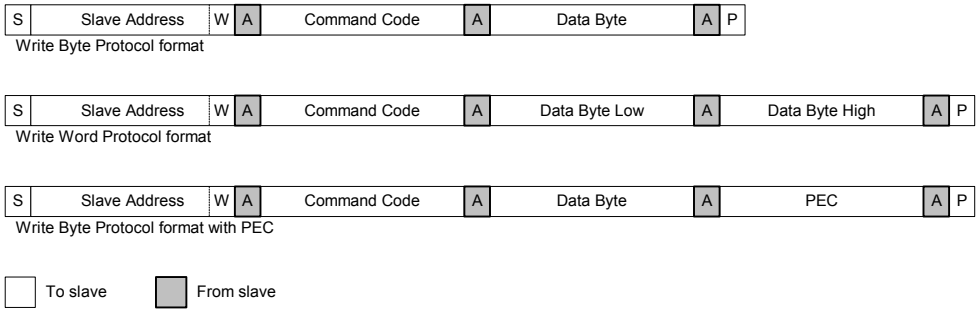
When the AVR receives the Command byte, a table look-up is used to check that the command is valid for SMBus Slave mode. If it is not, a bus timeout error is generated to inform the sender of the fault. If the command is valid, the bus is re-enabled by clearing the TWI Interrupt flag (TWINT) and the state machine advances.

5.9.1.1 SMBus Slave Write

If the next action on the bus, after the acknowledge of the Command byte, is that a byte is received, this indicates a WRITE sequence so the state machine remains in Slave Receive mode and collects the remaining data (see Figure 5-7). When all bytes

have been received, a flag is set asking for the foreground code to process the received command.

Figure 5-8. SMBus Slave Write command examples

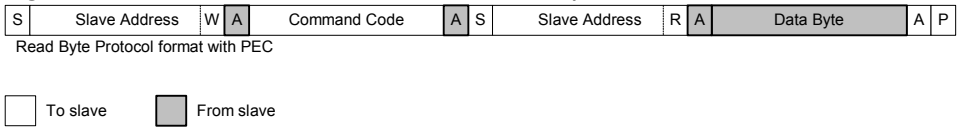


In the foreground code, a PEC validation is performed if the byte count indicates that PEC has been sent. Once it is determined that the data is valid, the command byte is used as an index into a table of pointers to functions, with each function corresponding to one and only one command. In most cases the function will write the received data to the SRAM-based SMBus variables. As needed, other actions will also be taken. Once the action is completed, the bus is re-enabled and is then ready for further transactions. If modifications are made to the software, it is important that the bus is re-enabled in less than 25ms or the Host may assume that the pack is generating a bus timeout error.

5.9.1.2 SMBus Slave Read

Alternately, if the next bus action after receiving a Command byte is a Repeated Start, this indicates that a READ is being requested (see Figure 5-9). The command value is stored and a flag is set for requesting the foreground code, in the main loop that is, to generate the data for the response. The command value is used by the foreground code as an offset into a table of pointers to functions. The referenced function will assemble the required data, calculating PEC if needed, and trigger the sequence for transmitting the data back to the bus master.

Figure 5-9. SMBus Slave Read command example



5.9.1.3 SMBus time-out error generation

Error checking is performed at appropriate points in the flow. For instance, if an invalid command is received, this can be detected immediately. When errors are detected, the ATmega406 returns to the IDLE state of the TWI state machine and does not clear the TWINT flag in TWCR, thereby leaving the SCL line stuck low. A 25ms timer is also started; when this timer expires, foreground code releases the SCL line by clearing TWINT and resetting the TWI peripheral to an Idle condition, and the SMBus is free to resume activity. See Section 5.8.3 for more details.

5.9.2 SMBus Master Mode

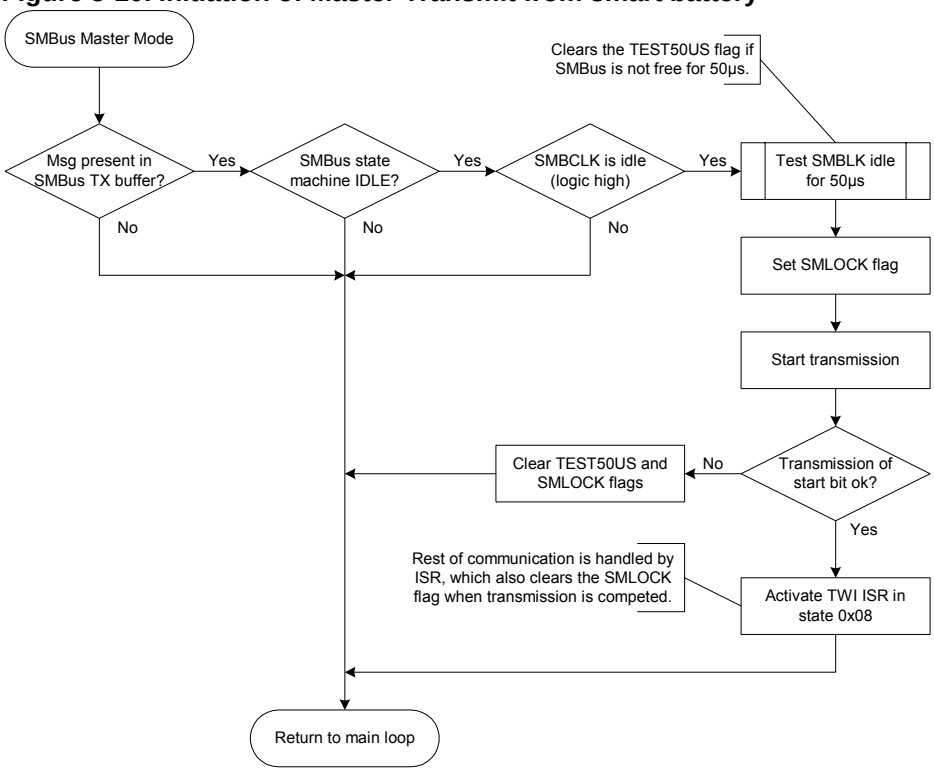
Master mode is initiated by the foreground code. When a message is available for transmission via Master mode, typically caused by a timer event expiring, it is placed in the TWI Master Transmit Buffer. However, transmission cannot begin immediately

due to the SMBus requirement that a bus master is required to check for bus idle time of 50µs.

To meet this requirement, the ATmega406 pin-change detection mechanism is used to monitor for bus idle conditions. This reference implementation assumes that PA6 is connected to the SMBCLK (SCL) signal. Alternatively, in a user specific implementation, any I/O signal with pin-change capability can be used instead of PA6. Note that the SCL line may not be read directly as the SCL and SDA lines are not shared with general-purpose I/O ports.

A multi-stage method is employed to ensure bus availability (refer to Figure 5-10). First, the presence of a message in the Transmit Buffer results in execution of the foreground code that manages Master mode. Next, if the Slave state machine is not in the IDLE state, no attempt is made to take control of the bus. Next, the SCL line is checked by reading PA6 to be sure it is not currently at a logical zero condition. At this point the bus appears to be free, so as the last stage a flag (TEST50US) is asserted to indicate that the 50µs bus-free test has now begun. During this time the SCL line is monitored by means of enabling the Pin-Change Interrupt for PA6. The foreground SMBus Master mode management code is then exited. With a clock speed of 1MHz, 50µs corresponds to 50 instructions at most. Therefore, when the code is re-entered it is guaranteed that at least 50µs has passed and handling the timing of the 50µs by the use of a timer is thus not required.

Figure 5-10. Initiation of Master Transmit from smart battery



When the Pin-Change Interrupt is enabled, any bus activity will trigger the interrupt. The corresponding ISR will clear the TEST50US flag, indicating that the bus is not free. Thus, when the foreground code is re-entered, if this flag is not asserted but there is a message in the buffer, it is understood that the test has failed and must be started again; thus the flag will again be asserted and the routine will be exited.

When the foreground code is eventually called again from the main loop and the TEST50US flag is still asserted, conditions are good for beginning a transmission. One last check is performed to ensure that the SMBCLK line is not low and the TWI ISR State Machine is in the IDLE state. Finally, a START bit is transmitted to take control of the bus. An additional lock flag (SMLOCK) is also asserted to indicate that Master mode has been entered by the TWI Hardware, so that the foreground code will not attempt to repeatedly initiate a transmission for the same message.

If the ATmega406 is successful in taking ownership of the bus, at the completion of the transmission of the Start bit the TWI ISR will be activated with a Status code of 0x08. As a result, the state machine will now vector into the states related to Master Mode transmission, and the ISR will handle all further aspects of the transmission. Alternatively, if the TWI module is unsuccessful in taking over the bus, the TWI ISR will still be entered but with status codes that are indicative of an error having occurred. In this latter case, the SMLOCK and TEST50US flags will be cleared to force another bus takeover attempt in the future.

5.9.3 Handling SMBus Errors

When an error is detected in SMBus communications while operating as a Slave device, there are two ways to signal this to the Master: (1) withhold the ACK, or (2) generate a Bus Timeout. The ATmega406 TWI peripheral only activates TWINT after it has already provided either an ACK or a NAK automatically; therefore, the ACK/NAK response cannot be based on a validity check of the data that was just transferred. As a result, the only viable response mechanism for the AVR is to force the Bus Timeout error. The first possible point in time where an error could be detected is dependent on whether the ATmega406 is receiving data, or is expected to send data back to the Master.

In an SMBus Slave Write transaction, such as ‘Write Word’, the ISR handles all facets of the transaction without intervention from the foreground code; therefore, there is little opportunity to check for errors until all of the data for this command has been transferred to the ATmega406. The Slave Write transactions are also ‘blind’ in the sense that all received data bytes, including the final byte, are to be ACK’d. Although this would normally simplify the Slave’s receive routine by not requiring it to know how many bytes should be received (and thus be in a position to NAK the final byte rather than simply ACKing everything), the lack of such knowledge results in the ISR being unable to generate a Bus Timeout error. This is because the Master will send a STOP after the final byte has been ACK’d, and as a result the ATmega406 has lost control over the SCL line since it is now in the “not addressed Slave” mode.

To resolve this issue, on Write-type transactions the command byte must be checked to determine how many additional data bytes are expected. Thus, after the specified number of bytes has been received, the complete received command can be passed to the foreground code for error checking while still holding SCL and therefore stalling the bus. The foreground code is thereafter responsible for forcing TWINT to be cleared after its check is completed. If the received command is error-free, it will clear TWINT immediately. If there are one or more errors, the TWINT, and thereby also the SCL line, will not be cleared until after the Timeout period has elapsed, and the received command will be discarded.

Since the command byte must be checked itself, it can also be easily validated from within the ISR. If any error conditions are discovered, the ISR resets its state back to IDLE and leaves SCL asserted low by not clearing TWINT, and signals the foreground code to cause a bus timeout error. After generating the timeout the foreground code clears TWINT, thereby restarting the TWI module.

The two-wire interface is designed to inherently handle a number of physical-layer errors as well. Since the bus is open-drain with pull-up resistors, bus contention is resolved at the time of the first bit-level difference between two bus masters transmitting simultaneously. The ACK/NAK bit after each byte serves as a presence indicator for the addressed device. However, the protocol does not guarantee uncorrupted delivery, as there is no provision for parity or other detection mechanism on a byte-for-byte basis. The inclusion of PEC in the specification is intended to provide an indication of whether error-free delivery has been achieved.

Please refer to Table 5-3 for details about error handling.

Table 5-3. Error handling in SMBus slave mode

Moment of Occurrence	Error	Responsive action
After receipt of Command Byte	Command value out of range	This error can only be detected after the ACK for the (erroneous) Command Byte has already been sent. The next transaction on the bus could be either a data write to the Slave (which could therefore be NAK'd), or a Repeated Start condition as a precursor to initiating a Read from the Slave.
	Command not valid in slave mode	This error can only be detected after the ACK for the (erroneous) Command Byte has already been sent. The next transaction on the bus could be either a data write to the Slave (which could therefore be NAK'd), or a Repeated Start condition as a precursor to initiating a Read from the Slave.
After receipt of the first data byte during Slave Write transactions	Command only allowed in Slave Read mode	The TWINT flag is not cleared, a timeout error is generated, and the packet is discarded.
	Attempted to write bits that are read-only or 'reserved'	In this case, the Write-type command itself was valid, but the action requested for that command type was not valid. This error is identified by the foreground command interpreter code. If the Slave discovers such an error, the TWINT flag is not cleared, a timeout error is generated, and the packet is discarded.
After receipt of the complete packet	PEC error	The ISR software determines in advance precisely how much data is expected so that it can halt the bus prior to the Master issuing a STOP, when dealing with a Write-type command. When all data bytes have been received, the received message (including PEC) is passed to the foreground code for interpretation and error checking. If the Slave's received PEC from the Master does not match its internally-generated value a transmission error has occurred, so the TWINT flag is not cleared, a timeout error is generated, and the packet is discarded.
	Value out of range	The TWINT flag is not cleared, a timeout error is generated, and the packet is discarded.

5.9.4 Packet Error Checking Implementation

The SMBus implementation in this reference design supports the use of Packet Error Checking (PEC). When a Master device communicates with the ATmega406 as a Slave, its desire to use PEC is indicated by whether it provides an ACK or a NAK after the last data byte is transferred during a Slave Read command. If an ACK is issued, it indicates that the Master still needs more data; this must be assumed to be a request for the PEC byte. Please refer to the SMBus specification for more details.

To accommodate the request for a PEC byte, the firmware will always generate a PEC value, but when handling a 'Read' command the firmware allows the Master to determine whether it is sent or not, as described above. Whenever any Master (whether the Host, a Charger or other device) requests PEC information on a Read command, a flag, UsePEC, is asserted indicating that the AVR should use PEC on its transmissions from then on (for both Slave Transmit functions as well as its own SMBus Master mode commands). Likewise, if any Master performs a Slave Read command and does not request PEC, then the flag is de-asserted.

For Slave Read command types, PEC is generated using all bytes from the complete transaction. This includes the original Slave Address + W, the Command, the Slave Address + R, and all of the reply data. Depending on whether the slave address is assigned by the Host system or is fixed, it may be possible to pre-calculate a partial CRC value based on the Slave address and the command, rather than generating it on-the-fly each time.

For Slave Write command types, it is not known for certain whether PEC will be included in the transmission. Therefore the TWI ISR code must accommodate the presence or absence of PEC, regardless of the state of the `UsePEC` flag. Upon completion of the receive operation, it is determined whether the PEC value is included as part of the received packet. As discussed in the previous section, the number of expected bytes is determined in advance according to the specific command, knowing that PEC may or may not be present in addition to this. After all expected bytes are received, the ISR leaves TWINT asserted and notifies the foreground code of the presence of a complete packet. The foreground code then analyzes the packet for errors, and if any are found a timeout error is generated and the packet is discarded. Otherwise, the command action is carried out and TWINT is cleared, freeing the bus to allow the Master to send a STOP and return the ISR state machine to the IDLE state.

In cases where errors are detected by the foreground code, it is necessary to be able to force the ISR back to the IDLE state. Therefore, the ISR's state variable has scope beyond the ISR itself.

5.10 In System Programming (ISP) over SMBus

This feature allows upgrades to a smart battery firmware, or rather the ATmega406 firmware, while in its target application. There are many possible ways of implementing the ISP functionality; the one provided offers basic ISP through the SMBus and does not require any additional hardware connections on the battery pack. Alternatively, advanced ISP with support for encrypted firmware upgrades could be added. Please refer to the Application Notes AVR230 and AVR231 for respectively DES and AES encryption for ISP firmware upgrades, and AVR109 for a general treatment of the bootloader concept. Please study the sections in the datasheet regarding self-programming and lock-bits to ensure correct protection of the firmware.

The software structure used is essentially two separate code images. One image contains the smart battery application itself and resides in the lower portion of the Flash memory area, called the Application Section. A second code image resides in the Boot Section of the ATmega406 Flash memory.

The boot loader functionality is restricted to a simplistic SMBus protocol handler, capable only of managing ISP over SMBus. In order to force the compiler to create code that begins at Flash memory address 0x4800 (word address) rather than at 0x0000, a special linker (`.XCL`) file is used to define a different memory range for program space. No special compiler settings are required beyond the normal settings for the ATmega406 device, except for the use of this customized linker file. Since the boot loader section only has up to 4Kbytes of code space, care has been taken to ensure that only necessary functionality is included. The present code implementation uses less than 1800 bytes of code space, so there is still space to expand the functionality while using less than 4Kbytes if desired.

To start code execution of a Boot Loader located in the Boot Section after RESET, rather than at the beginning of the Application Section, the BOOTRST fuse should be programmed (Please refer to the ATmega406 datasheet for details on fuse settings).

The Boot Loader code initially performs a simple check of the Application Section of the flash memory to determine if it contains a valid program image. In the `low_level_init()` function of the bootloader code, if the Reset vector at location 0x0000 contains 0xFFFF, it can be safely assumed that the Application area does not contain a valid image, and control will stay with the boot loader. Otherwise, a jump to address 0x0000 is performed, effectively starting the main application code.

Since the SMBus imposes a startup time limit of 500ms, the number of clock cycles available for the Boot Loader to perform extensive validity checks, like a complete CRC of the application section, is limited. It is possible to implement a more complete validation of the Application code and to also perform a very thorough verification in the primary application, and run this after the SMBus interface is brought into operation. This is left up to the user to implement.

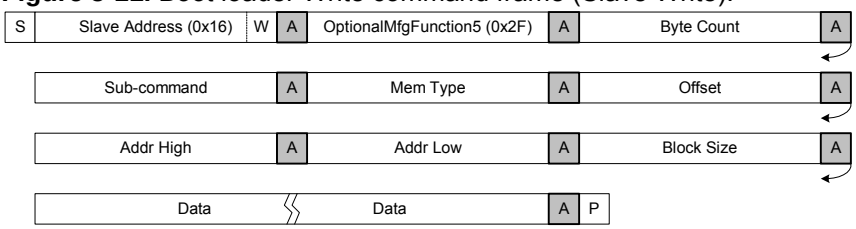
If the Application Section of flash memory does not contain a valid image, then execution continues in the Boot Loader's code. The entry point for the bootloader when being accessed from the application area is the entry point to the bootloader's `main()` function. Note that if any changes are made to the bootloader code or if it is recompiled, this address should be verified and the `smbus.c` code should reflect the correct address in its handler for `OptionalMfgFunction5`. Since this could be entered from either the boot loader or from the application area, all critical variables for the boot loader are initialized inside of `main()` to ensure that they are properly initialized regardless of how `main()` was entered. It also ensures that all interrupts are disabled to prevent inadvertent execution of any application-area code. Next, it initializes the TWI port to prepare it for SMBus communications. Finally, control passes to the main loop of the loader.

The main loop waits for the only SMBus command that is valid for the boot loader, namely, `OptionalMfgFunction5`. Upon receiving this command, it interprets the contents per the protocol defined below and performs the requested action.

Each SMBus Slave Write command modifies a Status flag based on success or failure of the command. It is possible, but not necessary, to query this Status flag after each Slave Write operation, even those that only update a portion of the SRAM data buffer. For operations that take a relatively long time, such as erasing flash or EEPROM, the Status flag will also indicate if the device is busy.

All communications are expected to use the standard smart battery address. Likewise, only command identifier 0x2F, `OptionalMfgFunction5`, is used for ISP over SMBus. All Slave Write operations use the Write Block mode and must all conform to the protocol defined below. Slave Read operations use only the Read Block protocol and will only return the Status value. It is left to the designer to implement a Memory Read command if this is desired, but this may expose the internal memory contents to copying.

Figure 5-11. Boot loader Write command frame (Slave Write).



Note that both EEPROM and Flash memory addresses are given as byte addresses. For the Slave Write commands, the meaning of the fields varies by the specific command and the memory type.

Five primary commands are implemented within the `OptionalMfgFunction5` command: Write, Erase, Patch, Insert and Verify. Provision is also included for a Read command, but this should be omitted in final products as it poses a security hole. Alternatively, one could consider adding an encryption layer.

There are also two secondary commands, Exit and Activate. Activate is used to switch to bootloader mode while running in the main application, and is ignored if received while already in the bootloader. Exit is used when all bootloader tasks have been completed and it is desired to start executing the main application code.

Two additional optional commands may be implemented by the user, the 'w' and 'v' commands (distinguished by the use of lower-case letters). These commands indicate that decryption should be performed on the data block (either an entire page for Flash, or the specified block size for EEPROM) prior to the write or verify operation. In this way, encrypted data may be transferred to the internal SRAM buffer but will be decrypted before use, ensuring data security.

Table 5-4. OptionalMfgFunction5 Sub-commands

Command	Command fields required	Functionality[1]
WRITE ('W' / 'w')	*Mem Type*: (F)lash / (N)onvolatile EEPROM *Offset*: Buffer offset (EEPROM only) *Addr High/Low*: Start address in Flash/EEPROM *Block Size*: Bytes to write (EEPROM only)	Write the contents of the internal SRAM buffer to the specified memory region. Operation may fail based on settings of the 'lock' fuse bits.
ERASE ('E')	*Mem Type*: (F)lash / (N)onvolatile EEPROM *Addr High/Low*: Start address in Flash/EEPROM *Block Size*: Bytes to erase (EEPROM only)	Erase the specified memory region. Operation may fail based on settings of the 'lock' fuse bits. This operation does not affect or make use of the internal SRAM buffer.
PATCH ('P')	*Mem Type*: (F)lash / (N)onvolatile EEPROM *Offset*: Buffer offset (EEPROM only) *Addr High/Low*: Start address in Flash/EEPROM *Block Size*: Bytes to read (EEPROM only)	Loads the internal SRAM buffer with the present contents of the specified memory region. The buffer may then be partially overwritten using the INSERT command and then written back to memory.
INSERT ('I')	*Offset*: Buffer offset *Block Size*: Byte count *Data*: Data bytes to write to SRAM buffer	Place the specified data into the internal SRAM buffer, starting at the specified buffer offset. Note that a complete fill of the buffer is not possible in a single INSERT operation.
VERIFY('V' / 'v')	*Mem Type*: (F)lash / (N)onvolatile EEPROM *Offset*: Buffer offset (EEPROM only) *Addr High/Low*: Start address in Flash/EEPROM *Block Size*: Bytes to verify (EEPROM only)	After loading the internal SRAM buffer, the Verify command will perform a comparison to the specified memory region. The Status flag will indicate the good/bad result of the comparison.
READ	Not implemented at this time	The Read command may be used primarily to read out the contents of the device's EEPROM memory so that it may be restored after a programming operation.
ACTIVATE ('A')	None, but data Byte Count has to be 1, i.e. equivalent to Word Write.	Transfer control from Application code to Bootloader. All other smart battery functions stop.
EXIT ('X')	None	Transfer control from Bootloader to Application code. All smart battery functions start again.

Notes: 2. Setting of lock bits may limit the possibilities to read and write to the Flash from the Boot Loader. Please refer to the datasheet for more details.

W: If Flash is specified, the address is the base of the Flash page to be written, in bytes. No other information is required as only a complete page can be written to flash. If EEPROM is specified, then the address is the starting address in EEPROM space; offset is the starting location within the on-chip SRAM buffer, and the byte count must be provided in the Block Size parameter.

w: Same as the “W” command, but the contents of the SRAM buffer must first be decrypted before being written. Also, the contents of this message are also encrypted to prevent an attacker from gaining information about the target location of the write command.

V: For both Flash and EEPROM, the address is any valid address (byte-based); it is not restricted to page boundaries. If flash is specified, the byte count is assumed to be 128 bytes as a precaution against someone easily mapping the contents of flash one byte at a time. For EEPROM, and byte size from 1 to 128 may be used.

v: Same as the “V” command, but the contents of the SRAM buffer must first be decrypted before being verified. Additionally, the contents of this message block are also encrypted to prevent an attacker from gaining information about the target location of the write command.

E: For Flash memory, an entire Flash page will be erased; the address is forced to the beginning of the specified page. Any EEPROM address and byte count is allowed, up to 128 bytes.

P: To ‘patch’ a memory area, first the Patch command is issued to copy the original memory contents to the internal SRAM buffer. The address must be at a page boundary for Flash memory. Next, the ‘I’ command is used to supply ‘repair’ data that overwrites only parts of the buffer contents. Finally, a ‘W’ is performed to save the update.

I: Since this command writes only to the internal SRAM buffer, the Address field is ignored. The Offset and Block Size values are required, as well as the data block. Due to SMBus packet size limits, only up to 24 bytes may be transferred on the initial packet. However, if the Byte Count field specifies more than 24 bytes, then subsequent ‘chained’ SMBus packets are expected to contain **only** data, thus allowing up to 32 bytes to be transferred in each subsequent packet until the Block Size that was specified in the initial packet has been fulfilled. The most that can ever be transferred is dictated by the SRAM buffer size (128 bytes) minus the starting offset. Thus, for a complete fill of the buffer using chained packets, an offset of zero must be specified. Alternately, individual Insert command packets may be issued with any amount of data as long as the offset plus the size of any packet does not exceed 128.

Note that the Write and Verify commands, whether for EEPROM or Flash memory as its target, use only the contents of the on-chip data buffer, not the data block within the ‘W’ or ‘V’ SMBus message. All data must first be written to the on-chip SRAM data buffer using the “I” command, and subsequently stored or verified from there. Note that data held in the buffer is not destroyed after a Write or Verify operation, so if the same data needs to be repeatedly written to different memory locations, this can be done by sending new ‘W’ commands without having to reload the SRAM buffer after each ‘W’ operation; simply supply a different address in the ‘W’ command each time.

Encrypted data may be used for writing or verifying EEPROM or SRAM memory as well if desired. Note that use of the Patch command cannot be supported when using encryption.

5.11 Power Modes of Operation

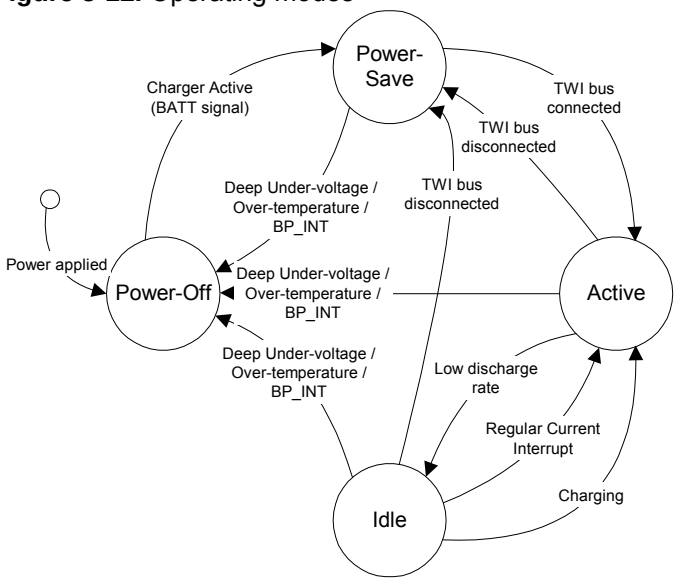
The battery pack can be in one of four power modes: Power-Off, Power-Save, Idle and Active. These mode names reflect the Sleep Modes for ATmega406. Refer to Table 5-5. Note that the implementation made is meant as a reference to how the power management can be implemented; it can be implemented differently if desired.

Table 5-5. Smart battery modes of operation

Mode of Operation	Used when:
Power-Off	Used when battery fully drained, also referred to as a Deep Under-voltage condition. Only charging through pre-charge FET is supported. Host cannot draw power from battery.
Power-Save	Used when Host is turned off, or if the battery is disconnected from Host/ Charger.
Idle	Used when discharge rate is low.
Active	Used during charging and ‘normal’ discharging rates

In all modes except Power-Off Mode, the Hardware Battery Protect circuitry is initialized and operational. See the accompanying state chart in Figure 5-12 for a graphical view of the interaction of these modes.

Figure 5-12. Operating modes



5.11.1 Power-Off Mode

In this mode, the pack has no charge remaining and is waiting for a recharge cycle. The ATmega406 is prevented from operating due to the on-chip voltage regulator being in power-off mode (please refer to the ATmega406 datasheet for details on the Power-Off Sleep Mode). In the Power-Off Sleep Mode the ATmega406 draws virtually no current from the battery. This mode is used to protect the battery: If a Li-Ion battery discharged under a certain limit it will be subject to permanent damage.

The Precharge FET is enabled automatically in hardware, allowing low-current charging to occur. The Charge and Discharge FETs are disabled via hardware. Since

the ATmega406 is not executing code, no current measurement nor time measurement takes place in this mode. The SMBus is also inactive.

This mode is entered automatically by hardware if the battery voltage drops below the ATmega406 Deep Under-Voltage detection level that result in an automatic power-off of the on-chip voltage regulator (See datasheet regarding available Deep Under-Voltage levels). Software can also force entry into this mode when the pack voltage drops below minimum cell voltage levels in order to prevent permanent cell damage. The hardware protection can thus be seen as a secondary independent battery protection circuit.

This mode is exited by a hard Reset is caused by the BATT pin going higher than 6 - 8V, which causes a Power-On-Reset with the PORF bit in MCUSR set to 1. Therefore, upon initial execution of the `main()` function after a Reset, when the PORF flag is set the operating mode will, in this implementation, be forced to be the Power-Save mode.

5.11.2 Power-Save Mode

In this mode, the battery is essentially asleep due to having been either removed from the Host or Charger, or the Host having been turned off. However, the CPU is ready to wake up immediately in response to either SMBus activity, which generates a TWI Bus Connect interrupt, or external low-level interrupts. Low-level interrupts are typically generated by a Charge Display pushbutton, used to show the charge state of the battery using a bar of LEDs.

In this mode, the Precharge FET is deliberately enabled to allow low-current charging. The Charge and Discharge FETs are disabled to prevent accidental short-circuits from occurring outside of the pack. Upon entering Power-Save Mode, the software disables the CCADC, configures the Wake-Up Timer and switches the ATmega406 to Power-Save Sleep Mode. Since the CCADC is disabled, battery drain cannot be measured and must be estimated. To aid in estimating battery drain, elapsed time is maintained through the use of the Wake-Up Timer. Thus, fixed periodic subtractions from the battery's available charge can be made to maintain a reasonably accurate charge estimate. The SMBus is not active in this mode except for the TWI Bus Connect/Disconnect interrupt, which is used to detect activity on the SMBus.

This mode is entered when either an SMBus Power-Down command is received while in either Idle or Active Mode, or when the SMBus has been disabled. The TWI Bus Connect/Disconnect interrupt is used to detect the latter condition.

This mode is exited when either the battery's available power has dropped to zero, in which case the Power-Off Mode will be entered; or the host system has activated the SMBus, typically due to either the battery being inserted into the system or the system being activated. When the host system has activated the SMBus, the battery will switch to Active Mode, from which it can change to Idle Mode if appropriate.

5.11.3 Idle Mode

This mode is used when the battery is active, but is discharged at a low rate, e.g. when maintaining a laptop computer's memory in Standby mode. The Precharge FET is disabled but the Charge and Discharge FETs are enabled. At low discharge rates, the current consumption of the ATmega406 itself could be a contributing factor in the overall power consumption and therefore the ATmega406 is operated in a reduced power configuration; hence the use of Idle Sleep Mode. Rather than continuously measuring the current consumption of the system, the current drain is only periodically sampled by using the Regular Current operating mode of the CCADC.

Despite the somewhat relaxed timing requirements, the 32kHz crystal oscillator is used as the CCADC clock source in this mode. The SMBus must be fully active and ready to respond to all requests, but other tasks such as battery voltage and temperature measurements can operate at a reduced rate in order to save power.

This mode is entered only from Active Mode, as the main difference from that mode is the accuracy of the method of measuring pack current. Specifically, when battery current drain falls below a predetermined level, the software changes to this mode in order to conserve power. Upon entry, the CCADC operating mode is reconfigured to use the Regular Current mode rather than the Accumulate Mode. Note that the result of the first four conversions is discarded by software as they do not hold reliable measurements. The CCADC Regular Discharge Current register is initialized upon entry. The CCADC Regular Charge Current register is not used in this implementation.

This mode will be exited if a charge current is detected, in which case Active Mode will be used. Also, if the battery current drain exceeds a selected level, the Regular Current Interrupt will fire and the software will switch back to Active Mode.

Since any interrupt will bring the ATmega406 out of Idle Sleep Mode, a mechanism must be established to identify when it is allowable for the ATmega406 to re-enter Idle Sleep Mode. In this implementation, if all active tasks in the main loop have been handled, it is ok to enter Idle Sleep Mode again.

The tasks that need to be handled include periodic VADC scans and calculations of the cell voltages, SMBus communication, CCADC current measurements and charge tracking. All of these are initiated or maintained by interrupts, and therefore sleeping while waiting for interrupts is not problematic in this application. The VADC peripheral receives clock in Active, Idle and ADC Noise Reduction Sleep Modes, while SMBus transmissions (including Slave-mode responses) requires either Active or Idle Sleep Mode. Therefore, Idle Sleep Mode is used instead of ADC Noise Reduction Sleep Mode.

5.11.4 Active Mode

In this mode, the battery is either charged or discharged at a relatively high current. When charging, the ATmega406 current consumption is unimportant. When discharging at a high rate, the ATmega406 current consumption is negligible compared that of the Host. Therefore in this mode the ATmega406 could be left running continuously without detrimental impact on battery life. However, since the software is already designed to utilize Idle Sleep Mode, it is used in the battery's Active Mode also.

In Active Mode, the Charge and Discharge FETs are enabled and the Precharge FET is disabled. The CCADC runs in Accumulate mode for maximum accuracy, using the 32kHz crystal clock oscillator. The SMBus is fully active.

Active Mode can be entered from either Idle or Power-Save Mode. When the battery is awake while being charged, Active Mode is always used. When high discharge currents exist or when first waking up from Power-Save Mode, Active Mode is selected because it measures the pack conditions with maximum accuracy. The CCADC is switched to Accumulate mode upon entry. The first four CCADC conversions after a mode switch, both for Instantaneous and Accumulate conversions, must be ignored, as they will not be accurate. This is handled in the software; however, the current gained or lost during that time is not accounted for in this implementation.

The software exits Active Mode in response to one of two conditions: either the SMBus has gone inactive, in which case it switches to Power-Save Mode; or when the current drain falls below a predetermined level, in which case it switches to Idle Mode.

6 Areas for Potential Improvement

6.1 Calibration

Depending on the accuracy of the sense resistor used, it may be necessary to include gain calibration on the current measurement. In order to avoid frequently dealing with expensive 'long' multiply operations, it is recommended to add scaling to the routines that report results and capacities, rather than correcting each and every sample. It is possible to adjust the calibration of the `Current` and `AverageCurrent` readings in approximately 0.5% steps by modifying the number of samples of the Instantaneous CCADC interrupt that are used each second. Please see the source code for details.

6.2 Power Management

The three active power modes have been designed as an example to highlight how the various peripherals and capabilities of the ATmega406 device can be exploited. More sophisticated modes can be added as desired.

It is possible to use the Interrupt mode of the watchdog timer to provide a periodic wakeup in Power-Down Sleep Mode rather than using the Wakeup Timer with the Power-Save Sleep Mode. This would result in slightly lower current consumption since the SlowRC Oscillator could then be turned off in Power-Down Sleep Mode.

Also, if better resolution is needed for the Wake Up Timer's estimated power consumption while in Shutdown mode, the `WakeUp_ISR()` routine can be enhanced.

6.3 Temperature Measurement and Utilization

Thermal effects on pack capacity are not presently being taken into account, as this effect can vary widely from one cell manufacturer to another.

Additionally, since it is not possible to know in advance what thermistor the designer will choose, the temperature calculations for thermistors is left to the designer. The supplied software provides the necessary infrastructure to measure the VADC result for each thermistor.

6.4 Hardware Battery Protection

In the present software, any fault condition resulting in triggering of `HWP_int()` will result in the pack switching to Power-Off Mode. A status code is saved to EEPROM to aid in determining the specific fault that occurred. This routine can be enhanced to allow multiple retries before shutting down the pack completely.

6.5 EEPROM

Considerably more use can be made of the on-chip EEPROM memory of the ATmega406, such as maintaining pack charge state information, and historical data such as the number of charge/discharge cycles or thermal extremes.

6.6 Encrypted firmware updates

The bootloader in this implementation is prepared for, but does not implement encrypted communication. If desired, please also check out the application notes AVR230: DES Bootloader and AVR231: AES Bootloader.

6.7 Battery Authentication

If the bootloader or application code were enhanced to include encryption, it would be possible to implement secure authentication codes via a challenge/response mechanism to ensure that only the right battery can be used with a given product.

7 Literature reference list

1. SMBus specification
   http://www.smbus.org/specs/smbus110.pdf
2. Smart Battery Data Specification
   http://www.sbs-forum.org/specs/sbdat110.pdf
3. Smart Battery Charger Specification
   http://www.sbs-forum.org/specs/sbc110.pdf
4. ATmega406 Datasheet
   http://www.atmel.com/dyn/resources/prod_documents/doc2548.pdf
5. Application note: AVR454: HW User's Guide – ATAVRSB100 - Smart Battery Development Board
   http://www.atmel.com/dyn/products/app_notes.asp?family_id=607
6. Doxygen documentation: `readme.html` and `doxygen` directory downloaded with the source code.

8 Table of Contents

Atmel Corporation

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

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

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

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

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature

2599B-AVR-09/05