



AVR456: Firmware User's Guide for SB201-1 & SB201-2

Features

- Provides all needed safety measures for a Lithium-Ion battery
 - Over-voltage protection
 - Under-voltage protection
 - Protection against excessive discharge and charge currents
 - Temperature checks
- High accuracy voltage and current measurements
- Communication
 - Half-duplex 1-wire UART
 - Command set based on Smart Battery Specification
- AES based authentication

1 Introduction

This document describes an example implementation of a smart battery for the Atmel® SB201-1 and SB201-2 reference designs. The implementation demonstrates how to use the ATmega8HVA/16HVA AVR® to gain optimal safety and accuracy for a Lithium-Ion rechargeable battery-pack. All code is freely available to allow for easy evaluation and further development.

The SB201-1 targets single cell smart batteries, while the SB201-2 targets dual cell smart batteries. The two products are based on the same firmware source code, compiled differently for each target-hardware. SB201 hardware for evaluation of the ATmega8HVA/16HVA device and the SB201 firmware are available as ATAVRSB201 – this kit includes both the SB201-1 and the SB201-2 hardware.

Figure 1--1-1. SB201-2 reference design – hardware.



8-bit AVR®
Microcontrollers

Application Note

Rev. 8139A-AVR-12/08





2 Overview

The main purpose of a smart battery implementation is to protect the battery from overcharging, over-discharging and use outside its temperature limits.

It also estimates its remaining capacity, and communicates this to a host/application.

Often smart batteries offer a method for authenticating the battery. The authentication makes it possible to ensure that only a compatible battery is used for an application. This allows the manufacturer of the application to guarantee that the application will work as intended with regard to battery life-time and safety.

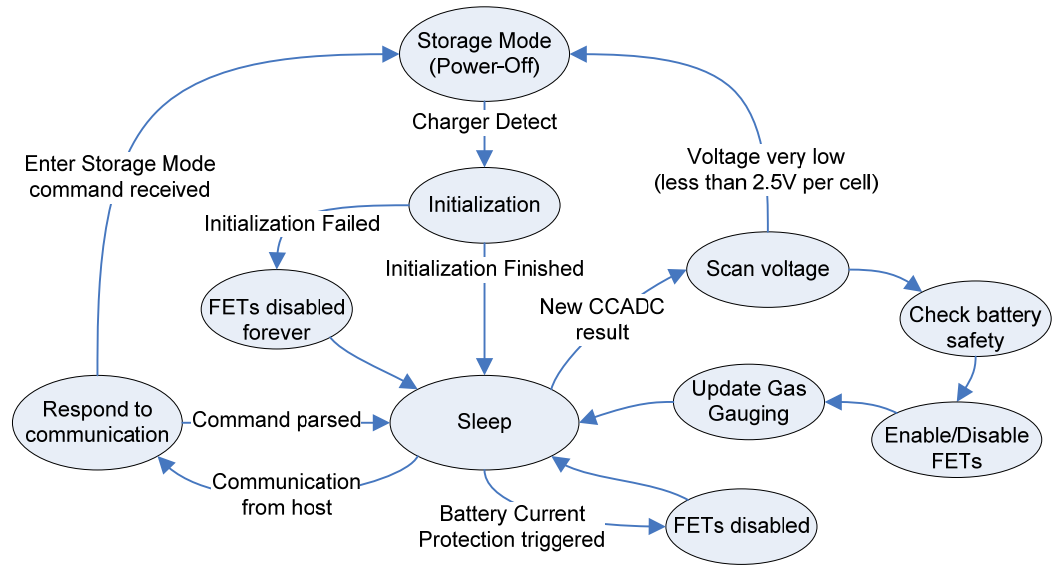
ATmega8HVA/16HVA has dedicated autonomous hardware to protect against short-circuit and excessive currents in both directions. Other less critical safety conditions, such as cell voltage level and operation temperatures, are handled by the CPU. To ensure good performance specialized analog hardware modules, the coulomb-counter ADC and the voltage ADC (CCADC and VADC), are used to gather accurate information about the operation conditions of the battery. The CPU/firmware collects and processes this information and responds accordingly. The dedicated and specialized hardware modules are described in the datasheet, and this application note demonstrate how they can be used to their best.

The CCADC can be set to trigger an interrupt at fixed intervals and operates in power-save mode. For this reason it is used as a time-base in this implementation. The CCADC accumulates (average) the current for one second and every time a conversion is completed the main loop will run one loop and the ATmega8HVA/16HVA will go back to sleep. How deep it can sleep depends on if all modules started in the main loop are finished and if there is ongoing communication.

For communication a half-duplex 1-wire UART is used. All transmissions have to be initiated by the host/application but the battery can answer.

SB201-1 is meant for 1-cell applications and SB201-2 is meant for 2-cell applications. The code size for SB201-2 is a bit bigger because it includes cell balancing. Having the same code for both versions is natural because the differences are minimal from the codes point of view. The #define "BATTPARAM_CELLS_IN_SERIES" in the header file determines if the code should target the SB201-1 or the SB201-2.

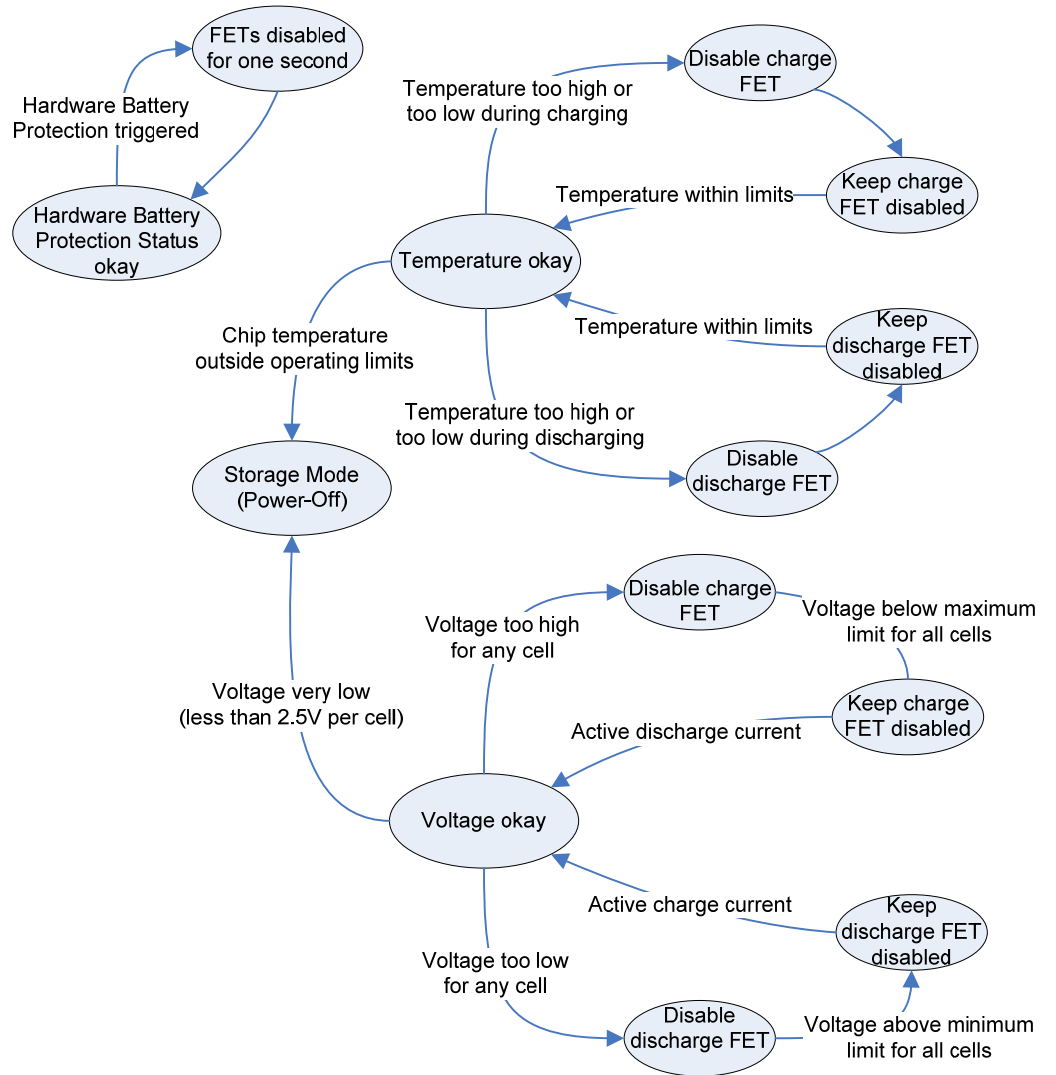
Figure 2-1 Overview of how the SB201 firmware operates



3 Battery safety

Making sure the battery is not a safety risk for the user is the most important task for ATmega8HVA/16HVA. A lithium-Ion battery can be very hot and even explode if overcharged or too much current is drawn from it, so good protection is needed for a commercial battery.

Figure 3-1 Overview of the battery safety system



3.1 Battery protection hardware module

The hardware battery protection on ATmega8HVA/16HVA disables both FETs when too much current flows in or out of the battery. It has three limits for discharging and two for charging. The limits can be set to different currents and reaction times. This allows a very fast short-circuit detection, but allows for lower charge/discharge spikes without cutting supply to the host/application. Table 3-1 shows an example of how the three discharge limits can be set up.

Table 3-1 Example timings for battery protection.

Current exceeds	Duration for discharge FET to be disabled
2A	20ms
3A	2ms
6A	125us

In this implementation the lowest discharge current limit (called discharge high-current protection) is disabled to allow faster discharges during testing. The parameters are stored in EEPROM and used to configure the module at startup.

Multiple charge over/high-current protections in a row indicate a malfunctioning charger. In that case, the charge FET is disabled and will not be enabled again until a discharge current higher than standby is detected. That way the battery is protected even if disconnected from the charger and connected again.

3.2 Firmware based battery protection

The firmware based battery monitoring of the operating conditions is protecting the battery from over-voltage, under-voltage, and hazardous temperatures.

3.2.1 Current

Since the battery lifetime is decreased by high charge and discharge currents, some batteries might also want to limit currents to levels below 2A, which is the lowest value for the hardware battery protection. It might also be desired to use more sophisticated current limitation schemes/algorithms than simple thresholds. This is left up to the end user to implement.

3.2.2 Temperature

Very high or low operating temperatures reduce the lifetime of the battery, and moreover, high temperature can also be a sign that the battery is damaged. Therefore it is common practice to disable charging/discharging when the temperature is outside a certain temperature range. This is a good idea both for general safety and battery lifetime. This implementation checks the temperatures every four seconds (to minimize active time and power consumption). This is considered sufficient for most batteries, as rapid thermal changes are not expected.

By default, cell temperature monitoring is disabled, because no thermistor is soldered on SB201 at assembly. Thermistors are however provided with the SB201 kit. See 11.3 for more information. Instead the internal ATmega8HVA/16HVA temperature reference is assumed to be the same as the cell temperature.

If the temperature is too high or too low, either the charge or discharge FET will be disabled depending on in which direction the current is flowing. It will not be enabled again until the temperature is within the allowed limits.

If the internal temperature of the ATmega8HVA/16HVA is outside the operating limits (-20°C to 80°C), the ATmega8HVA/16HVA will turn itself off and the FETs will automatically be disabled.

All the temperature limits for the battery are stored in EEPROM.

3.2.3 Voltage

By checking the voltage the battery can protect itself from overcharging and over-discharging. Both of these conditions reduce battery lifetime and is a safety risk. Therefore, when a charge current is flowing the cells are continuously checked for over-voltage and similarly while discharging, they are checked for under-voltage.

When the voltage is too high, the charge FET will be disabled. It will not be enabled again until a discharge current higher than standby is flowing. The protection works the same for the discharge FET; after an under-voltage condition the discharge FET will not be enabled again until a charge current is detected.





When the discharge FET is disabled, its body diode is conducting in the charge direction. However, the diode causes a voltage drop that reduces the voltage difference between the charger and battery stack, which again causes a reduced charging current. If the battery voltage is high (~3.55V and up), virtually no charge current will flow when the charger is charging with standard 4.2V.

This condition can occur if the cell voltage drops below the low-voltage limit due to high current drawn from the battery (e.g. 3C). In this case the remaining capacity of the battery is still fairly high. When the discharge FET is disabled, by hitting the low-voltage limit, the battery cell will recover and the voltage will increase. The voltage can increase from 2.7V to 3.6V, which is high enough to limit the charging.

To handle this scenario the discharge FET is enabled if the battery voltage is above a given threshold or if a charging current is detected (a few mA is enough).

If the voltage drops very low (e.g. 0.2V below minimum operating voltage for the cell), the ATmega8HVA/16HVA turns itself off to limit further discharging of the battery. To avoid shutting down on short discharge spikes, e.g. due to high outrush current when inserting the battery in an application, the voltage has to be low for two seconds before power off is performed.

The voltage is also used for:

- Checking if cell balancing should be activated on SB201-2.
- Check when the battery is fully charged/discharged. Useful for the gas gauging modules.
- Disabling Deep Under-Voltage Recovery mode. When the voltage is high enough, DUVR mode is disabled and the firmware can control the FETs.

All limits for the voltage protection are stored in EEPROM.

4 Battery parameters/settings

Most battery parameters are stored in EEPROM for easy reconfiguration. Those values are in “physical” units, i.e. mA, minutes etc. Internally SB201 uses other units to reduce computation complexity to a minimum; therefore at startup some parameters in the EEPROM are converted into other units and stored in SRAM.

Changes to some settings that require that the code is recompiled affect more than just a value in EEPROM; most obvious are the number of cells and availability of external temperature sensors.

All parameters and settings are located and commented in the `battery_pack_parameters.h/c` files. Battery characteristics data is stored in the firmware module that uses them, e.g. `cc_gas_gauging` and `voltage_based_SoC`.

5 Clocks & Calibration

The ATmega8HVA/16HVA's internal fast RC oscillator is used as the system clock. It runs at a nominal frequency of 8MHz, but the system clock prescaler should be fused to 1/8 for SB201, resulting in a system clock with a frequency of 1MHz.

The fast RC oscillator varies over temperature and this may cause the communication to fail because of variations in baud rate. However, this is avoidable by calibrating the fast RC at runtime with the use of the slow RC oscillator, which frequency can be determined as a linear function of temperature. This makes it possible to calibrate the

fast RC oscillator sufficiently accurately to have reliable UART communication. The calibration is run every time the chip temperature has changed by 2 degrees °C.

When the slow RC period is known (calculated), it is known how many fast RC ticks that should occur during that period for it to be running at 1MHz. When calibrating the number of fast RC ticks during 8 slow RC periods is counted and the calibration register is adjusted until the number is as correct as the adjustment allows for. The calibration register is only adjusted one step at a time to avoid changing the frequency too much.

5.1 Real time counter

There is no need for an accurate real time counter (RTC) on SB201. Real time is only used for knowing approximately when a number of minutes have passed and is not used for any calculations.

The RTC is clocked by the slow RC oscillator through the CCADC conversion time. Since the actual slow RC period can be calculated as described earlier, it is possible to improve the accuracy of the RTC if needed.

6 Initialization flow (after reset)

During initialization all peripheral and firmware modules are configured, calibrated, and prepared for operation. Upon the very first startup (i.e. power-up after programming) CCADC offset adjustment is also performed. The initialization sequence is listed below.

1. Set up the watchdog and check if too many watchdog resets have occurred. If it has it indicates a major problem and the battery probably should be disabled, but nothing is done in this implementation.
2. Disable unused peripherals and pins to reduce the power-consumption.
3. Check that the battery parameters and signature row is correct (CRC16). If they are not, the part can be measuring wrong values and have wrong limits for battery protection. So, the battery probably should be disabled if the checks fail, but it is not done in this implementation.
4. Calibrate the bandgap. If it fails, the readings from the ADC modules cannot be trusted and therefore the battery will be disabled.
5. Init the hardware battery protection module
6. Set up the VADC module with gain coefficients and offsets from the signature row.
7. Values to convert between internal units from the CCADC to mA/mAh are calculated from the shunt resistor value stored in EEPROM.
8. Some often used limits from the battery parameters are converted to internal units to avoid doing that calculation every second.
9. The VADC runs one scan of all available inputs to have values for the first main loop and also to check if DUVR mode can be disabled.
10. If not a serious failure occurred earlier in the initialization process, the CCADC and coulomb counter gas gauging modules are configured. The remaining capacity for the CC based gas gauging is set by using the State of Charge (SoC) from the voltage based gas gauging. It is however marked as inaccurate and updated as soon as the voltage based gas gauging has an accurate estimation.
11. If it is the very first startup, it will measure the CCADC offset. This process takes up to 30 seconds.



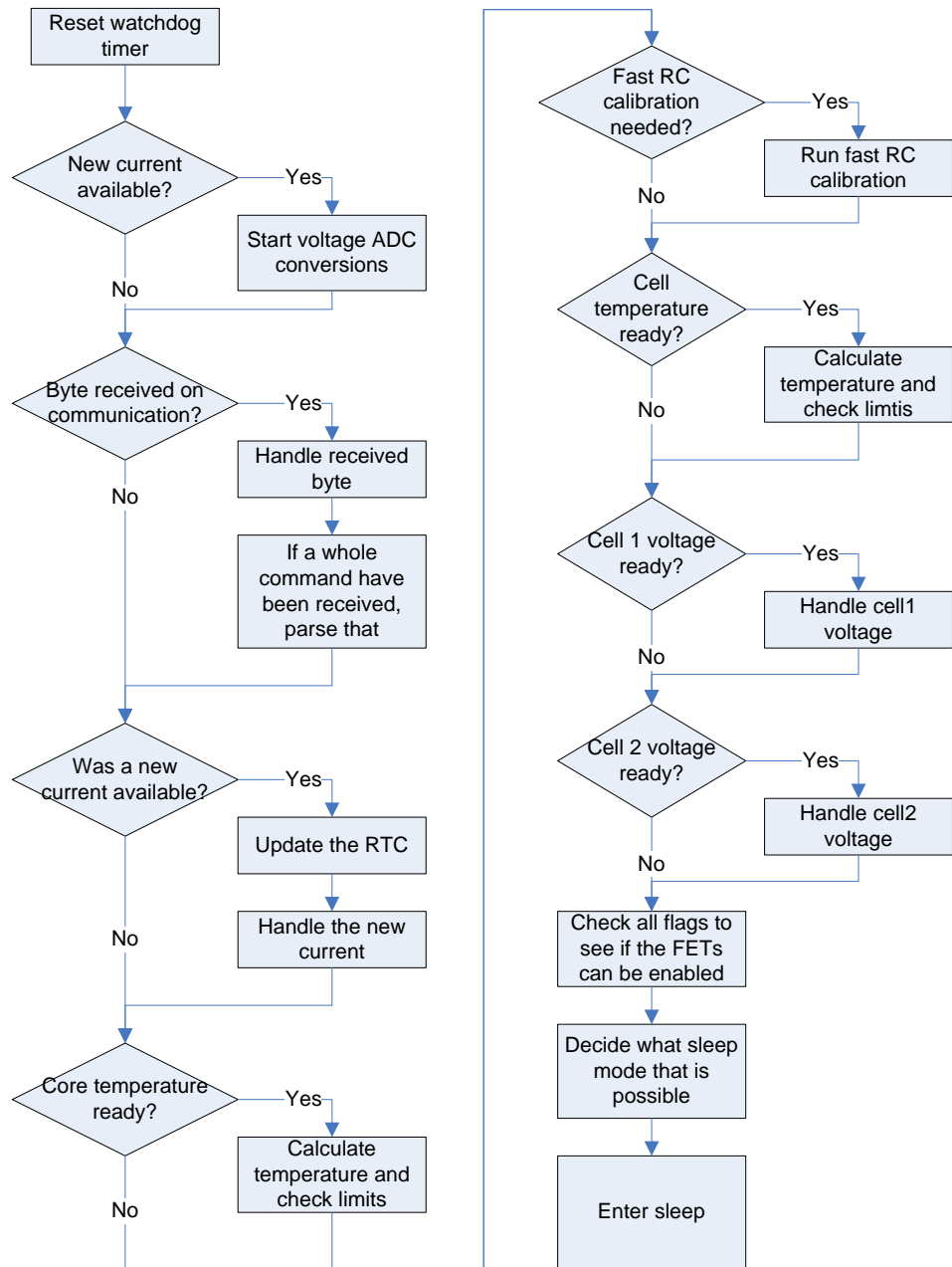


12. Finally the communication is initialized and global interrupts enabled before entering the main loop.

7 Main loop

The purpose of the main loop is to gather and react to data from the interrupts. It runs through all available data and checks that all are within the limits. If errors are discovered during charging or discharging, the corresponding FET is disabled and an error flag is set. If no error flags are set at the end of the main loop, the FETs will be enabled again. Since the main loop is run once per second, the FETs will be disabled for a minimum of 1 second when the firmware detects an error.

Figure 7-1 Simplified flowchart for the main loop



The hardware battery protection module prevents the FETs from being enabled within one second after Battery Protection event (caused by too high current) has been triggered. Since the current protection is handled by hardware the main loop doesn't have to check for that before enabling the FETs.

8 Interrupts

The implementation is highly interrupt based: All processing is based on interrupts triggers and interrupts (and flags) are an essential part of the implementation. The interrupts used in the implementation are listed in Table 8-1:





Table 8-1: Interrupts used in SB201 smart battery application.

Interrupt source/ function name	Description
Voltage Regulator Monitor interrupt VREGMON_ISR()	Stores all interrupt settings for the various modules and enter power-save until the voltage regulator is okay again or CREG is empty. If the voltage regulator is okay, restore the interrupts and continue operation. See the ATATmega8HVA/16HVA datasheet for description about how the voltage regulator monitor works.
Watchdog Timeout interrupt WDT_Timeout_ISR()	Only used by the voltage regulator monitor interrupt to wake the device up to check if the voltage regulator is okay.
Timer 0 Output Compare Match interrupt RCCAL_InputCapture_ISR()	Used to capture the number of fast RC clocks during one slow RC period. See Clocks & Calibration.
Battery Protection interrupt BATTPROT_BatteryProtection_ISR()	Triggered when the hardware battery protection module disabled the FETs due to excessive currents or short-circuit. It sets a flag, which is used to alert the host. If it was an overcharge it increases the number of times that has happened to be able to detect a malfunctioning charger.
CCADC Accumulating Conversion Complete interrupt Ccadc_Acc_ISR()	If the CCADC is set to negative polarity, the result is negated; otherwise it is just stored as it is. The main loop will then process the result. If polarity switching compensation (see 10.2) is activated, the compensation is done here.
Voltage ADC Conversion Complete interrupt VADC_ISR()	Flags that a channel or group of channels is complete. The main loop will then process the result. If there are more channels to scan in the ongoing sweep, sets the mux and starts the VADC again. Otherwise turns the VADC off.
External Interrupt Request 0 External_SW_UART_ISR()	Triggered when a start bit is detected on the 1-wire UART. Sets the communication timer to one and a half bit period to time the read of the first bit in the message. Overrides the sleep setting and sets it to idle. This is to prevent the device from entering power-save when the communication timer is used.
Timer 1 Output Compare Match interrupt Timer_SW_UART_ISR()	Is triggered when a new bit should be read or sent on the UART. If the transmission is finished, it turns the interrupt off. Otherwise it is kept running.

9 FET control

If an error occurs that requires the current flow to be cut, the FET will get disabled immediately and an error flag will be set. The FET will not get enabled again until all errors have disappeared, i.e. all error flags are cleared.

The flags are divided into different variables in the code, but only the flag name is used here.

Table 9-1: Interrupts used in SB201 smart battery application.

Flag name	Clearing conditions	Description
CriticalConditonDetected	Will never get cleared.	Set if the bandgap or VADC can't be calibrated/initialized and will keep both FETs disabled all the time.

Flag name	Clearing conditions	Description
ChecksumFailure	Will never get cleared.	Set if either the signature or EEPROM battery parameters CRC is wrong. Is not done in this implementation, but the battery should be disabled if the checksums are wrong.
InDUVR	Cleared when DUVR mode is disabled.	Set at startup and if the code detects that DUVR mode have been enabled again. Will prevent both FETs from being enabled, but DUVR has control over the charge FET so charging is possible.
SystemIsInStandby	Cleared if the current is higher than the active current threshold in either direction.	Set if the current is less than the active current threshold. By default that is 10mA
ReoccurringChargeProtection	Cleared when an active discharge current is detected.	Set if a too high charge current is detected too many times in a row, which indicates a malfunctioning charger. Will disable the charge FET when set and prevent it from being enabled again until cleared.
ChargingProhibited	Cleared when an active discharge current is detected.	Set if too high voltage is detected. Will prevent the charge FET from being enabled while set.
DischargingProhibited	Cleared when an active charge current is detected.	Set if too low voltage is detected. Will prevent the discharge FET from being enabled while set.
VoltageTooHigh	Cleared if the last voltage was below maximum for all cells.	Set if last voltage was too high for any cell. Disables the charge FET.
VoltageTooLow	Cleared if the last voltage was above minimum for all cells.	Set if last voltage was too low for any cell. Disabled the discharge FET.
CellTemperatureTooHigh	Cleared if the last temperature reading is below maximum limit.	Set if any cell (or internal temperature reference of the ATmega8HVA/16HVA if no thermistor is connected) temperature is too high. Will disable either the charge or discharge FET when set depending on which direction the current is flowing. Will prevent both FETs from being enabled while set.
CellTemperatureTooLow	Cleared if the last temperature reading is above minimum limit.	Set if any cell (or internal temperature reference of the ATmega8HVA/16HVA if no thermistor is connected) temperature is too low. Will disable either the charge or discharge FET when set depending on which direction the current is flowing. Will prevent both FETs from being enabled while set.

10 CCADC

The result from the CCADC is used to check the current level, compute the average current over one minute (approximated to save memory) and to compute the accumulate charge that has flown in and out of the battery. Only the accumulated result from the CCADC is used when checking limits and accumulating charge (i.e. instant current is not used).

The CCADC is set to 1 second conversion time and to trigger an interrupt when conversion is complete. That way it can also be used to update the RTC. The actual timing will depend on the temperature, but as described in Clocks & Calibration the RTC does not need to be very accurate.

The actual conversion time is although important when accumulating current for counting the capacity that has flown through the battery. The CCADC is clocked by





the slow RC oscillator and its actual period can be calculated from the internal temperature reference of the ATmega8HVA/16HVA and the slow RC temperature predication value that is stored in the signature row. More about this can be read in the datasheet. The cc_gas_gauging module is responsible for accumulating the capacity and it compensates for the actual slow RC period.

If polarity switching is enabled, the CCADC complete conversion interrupt is responsible for storing the result so that it is always positive for a charging current and negative for a discharging current.

When a new current result is ready from the CCADC it is passed to the battery_current_monitoring module by the main loop, where it is added to the one minute average current. The average is calculated as in Equation 10-1. The scaling is used to improve accuracy and is removed when the average is returned. Note that an exponential filter is used to estimate a 1 minute moving average current. This is done to save memory, as a 1 minute moving average is very SRAM consuming.

Battery_current_monitoring also gives easy access to the current and offset calibrated current for other modules that need it.

Equation 10-1 Method for calculating average current.

$$\text{New average} = \frac{124 * \text{Old average} + 4 * \text{current} * 2^n}{128}$$

$n = \text{AVERAGE_CURRENT_SCALING}$ (default value: 4)

10.1 CCADC Offset

The CCADC have a certain offset. When accumulating charge, it can be cancelled out by switching polarity at a fixed interval. But since it is possible to measure the offset it's nice to use that to correct the measurement when it's used as momentary current. It helps most on low current when checking if the battery is in standby mode or not.

The offset is measured the first time the battery is started and takes 20-30 seconds, during which both FETs will be disabled. It discards the first sample after every switch and compares the average of the other samples between positive and negative CCADC polarity. Half the difference is the offset and that can then be added or removed (depending on polarity setting) to later measurements to get a more correct value.

Switching polarity too often is not a good idea because the first accumulating conversion after a polarity switch is not correct, more info about that in the next chapter.

10.2 Challenges with polarity switching

The ACC conversion from the CCADC is calculated from the average of 256 instantaneous conversions by hardware (when having it set to 1 second conversion time). But every time the polarity is switched, the first two to three instantaneous current conversions are wrong. The first one is completely off, the second quite wrong and the third is almost always correct and if not, only slightly wrong.

Figure 10-1 Example ICC samples before and after polarity switch.



This is caused by settling time in the CCADC, which requires 2-3 ICC conversions to settle. In the example in Figure 10-1 the error for 1 second conversion time and constant current of -185 would be:

$$error = \frac{254 * 185 + (-140 + 101) - (256 * 185)}{256 * 185} = -0,86\%$$

To remove the problem as much as possible, the two or three samples after the switch should be removed from the ACC result for that second and replaced with an average of the ICCs before and after them.

To do that the part have to wake up three or four extra times after a polarity switch to read the ICC results and that adds active time and therefore power consumption. It also adds complexity and code space.

SB201 takes a simpler approach, which still removes most of the error. The sum of the two samples after the switch is small compared to one ICC sample, so they are already considered "removed". And instead of adding the average of the ICC readings around them, only the ICC sample from just before the polarity switch is used. As that sample is available when switching the polarity, practically no runtime is added.

With the same assumptions as in last error calculation, the error will now be:

$$error = \frac{(254 * 185 + (-140 + 101) + 2 * 182) - 256 * 185}{256 * 185} = 0,095\%$$

Note that in Figure 10-1 the results are not polarity compensated, so the result from before the switch is negated before using it in the error calculation.

Since the ACC result have the same range for different conversion times, the result is downscaled by the hardware. Therefore the ICC result also has to be downscaled before it is added to the ACC result.

For different values of samples and downscale from Table 10-1, the ACC result is:

$$ACC = \left(\sum_{n=1}^{samples} ICC[n] \right) \gg \text{downscale}$$

Table 10-1 ICC samples for different ACC conversion times

Conversion time	Samples	Downscale
1sec	256	3





Conversion time	Samples	Downscale
500ms	128	2
250ms	64	1
125ms	32	0

So to replace the two “missing” ICC samples, the ICC sample from just before the switch has to be divided by 4 (doubled and then scaled down 3 bits) before adding it to the ACC result.

10.3 Shunt calibration

The result from the CCADC is the voltage drop over the external shunt resistor, which on the SB201 have a value of 10mOhm \pm 1%. The shunt resistance is used with the result from CCADC to calculate the current. As a calibration option the shunt value can be changed runtime – by sending a new shunt resistance value via the communication interface.

A multimeter can be used to measure the exact current flowing in/out of the batteries. This information can then be used to calculate what the shunt resistance really is. This is described in more details in application note AVR491.

The shunt resistance can be set between 4000 and 16000 microOhm using the ShuntCalibration command (0x2A). Other values are possible, but require changes to the code to avoid computational overflows.

11 VADC

ATATmega8HVA/16HVA has a 5 channel VADC for measuring cell voltage, chip temperature and cell/battery temperature. The chip temperature uses an internal diode. External thermistors are required to measure the battery cell temperatures.

Immediately after a CCADC ACC result is ready, a new VADC scan is configured and started. This is to ensure that the data are ready when the main loop can process it (since the V-ADC conversion takes long time compared with the processing of the data). Ideally seen the V-ADC should only run while the CPU is busy processing information. Alternatively energy is wasted in Noise Reduction mode/Idle mode sleep or even active mode, just waiting for the conversion to complete. In reality Noise Reduction sleep is used whenever it is required to wait for a conversion to complete.

Configuring the VADC scan means to set which channels should be sampled. It depends on how many cells and thermistors that are used. Cell voltage is sampling every second, but temperatures are only sampled every fourth second as they do not change that fast. The ADC input channels are scanned in the same order as they are processed in the main loop. This is to ensure that the CPU can process data while converting the next sample.

When one channel is finished the VADC Conversion Complete interrupt routine sets a data ready flag to notify the main loop. This allows the main loop to process e.g. the result from cell 1 voltage while the VADC is sampling cell 2 voltage. The interrupt routine advances to the next input channel in scan (reconfigure the VADC mux) or, if the scan is completed, turns off the ADC sample.

11.1 Calibration of V-ADC

To ensure optimum accuracy when using the V-ADC, all results must be corrected for offset and gain error. The offset and gain compensation can be read from the HVA signature row. Please refer to the datasheet for more details.

Equation 11-1. Offset and gain compensation of V-ADC result.

$$ADC_{compensated} = (ADC_{result} - ADC_{offset}) \cdot ADC_{gain} \quad ; \quad (\text{downscaled by } 2^{14})$$

11.2 Chip temperature

To calculate the chip temperature, the VADC result is multiplied with the VPTAT value from the signature row and scaled down as described in the datasheet. Further, it is multiplied by 10 to get results in 0.1°K, which required according to the SBS specification.

All temperatures are stored in 0.1°K.

Note that the V-ADC result for the internal temperature reference only requires gain compensation. Offset compensation is not required.

11.3 Cell temperature

Since no thermistors (NTC) are soldered on SB201 during assembly, cell temperature reading (and processing) is disabled by default. Instead the chip temperature is assumed to be the same as the cell temperature when evaluating the system temperature.

To use one or two thermistors, connect them as described in application note AVR455 and change two defines in `battery_pack_parameters.h` (search for `CELLTEMPERATURE_INPUTS` and read the comments just above it).

To convert the VADC result to mV the result has to be offset calibrated, multiplied with a gain coefficient and then scaled down. The downscaling is chosen to maximize accuracy so the VADC module by default returns 20*voltage in mV.

Equation 11-2. How to calculate voltage on VADC according to the datasheet.

$$ADC_{mV} = \frac{1}{10} \cdot \frac{(ADC_{result} - ADC_{offset}) \cdot ADC_{gain}}{16384}$$

Equation 11-3. How SB201 calculates the scaled voltage from VADC result.

$$ADC_{scaled_mV} = \frac{(ADC_{result} - ADC_{offset}) \cdot ADC_{gain}}{8192}$$

The thermistors that come with SB201 are Mitsubishi RH16-3h103f and the NTC firmware module in file `ntc_rh163h103f.c` converts the scaled voltage from the VADC module to Kelvin or Celsius. To use another NTC other data used by the firmware module have to be used.





11.4 Cell voltage

To get the real voltage from the VADC result an offset have to be added, the result multiplied with a gain coefficient and then scaled down, as written in the datasheet. This conversion between the V-ADC result and the actual voltage is only calculated once per conversion; later use of the cell voltage does thus not spend time converting the V-ADC result to Volts.

12 Gas gauging

Gas gauging is estimating how much capacity and runtime there is left in the battery. ATmega8HVA/16HVA can achieve high accuracy gas gauging by means of very accurate VADC and CCADC. The great storage capability in Flash, SRAM as well as EEPROM allows the use of accurate battery models.

SB201 uses a combination of voltage and coulomb counter based gas gauging. The voltage based is used to update the remaining capacity, which the coulomb counter then will increase/decrease during use.

12.1 Gas gauging strategy on the SB201

At startup, the voltage based gas gauging estimates the State of Charge (SoC) using an estimate for the internal resistance of the battery. The Remaining Capacity is calculated from that SoC and from the latest Full Charge Capacity stored in EEPROM. This provides a reasonable initial estimate of the SoC and Remaining Capacity, though it is not as accurate as the estimate will be when the Open-Circuit Voltage (OCV) is measured, or when the battery has been charged to full capacity ones. For this reason, a flag is used to indicate that the SoC/Remaining Capacity is "inaccurate", and that a better estimation should be performed as soon as possible.

The SoC from the voltage based gas gauging module is considered accurate when less than a standby current has been maintained for 30 minutes (threshold for standby current is specified in firmware, and are stored in EEPROM). The first time this happens after startup, the remaining capacity is updated and is no longer flagged as inaccurate.

Remaining capacity is updated every second by the coulomb counter and it is used to calculate all the information SB201 can provide to the host/application.

Because the end-of-discharge voltage is reached earlier when high currents are drawn from the battery, the calculation of the Runtime to Empty takes the current level into account. The calculation uses characterization data for the specific battery cell type to determine how the available capacity changes at different currents levels compared to when using only a very small current. Ideally the characteristic data should be updated as the battery gets older, as the battery characteristics change slightly over time, but this is not offered in this application.

12.2 Voltage based gas gauging

Voltage based gas gauging works quite well to determine the SoC from the open circuit voltage (OCV, i.e. at no or low current). But as soon as "more" current is flowing, the battery's internal resistance affects reduce the terminal voltage. The resistance changes with temperature, SoC and battery age, which makes it challenging to accurately estimate the SoC, from the OCV only.

Two other problems with voltage based gas gauging are:

- It only gives the state of charge in percents and no indication of the capacity of the battery which makes it hard to give an accurate runtime of the battery.
- When the current stops, the voltage does not immediately return to the OCV. How long it will take depends on the battery, but generally around 30 minutes.

However, the combining of OCV based estimation and coulomb counting based estimation provides a good overall estimation that is not very sensible to temperature, aging etc.

12.3 Coulomb counter gas gauging

A coulomb counting approach keeps track of how much current that is both charged into the battery and drawn from it. It is very accurate even at high currents. The disadvantage of the coulomb counting based gas gauging is that it can measure changes only: This means that it does not offer information about the absolute charge level, and therefore an initial charge state is also required.

When doing coulomb counter gas gauging, the current charge/remaining capacity has to be estimated using a voltage based approach at startup and also possibly after long time in standby. If the initial voltage based SoC estimate is wrong, the coulomb counter based estimate will also be wrong. It can however calibrate itself when reaching a full discharge and full charge.

By using a coulomb counter the full charge capacity can be updated as the battery gets older with the following method:

It is known when the battery is fully charged and fully discharged, so how much charge that has been used between them is the full charge capacity. A full charge on a Lithium-Ion cell is when the charge current has dropped low enough, often down to the range of 10-15mA. Full discharge is a bit trickier, as the End-of-Discharge (EoD) voltage will be reached before the battery is actually empty when discharging with high currents because of the internal resistance. So the EoD has to be reached at a low enough current for it to be a full discharge.

The fact that the battery will typically never get fully discharged because the host/application shuts down represent a challenge when trying to estimate the Remaining Charge and Runtime-to-Empty.

13 Battery characterization data

SB201 need a table for SoC versus OCV, a table for how much less capacity that is available at higher currents and optionally, but recommended, the full charge capacity.

SB200 can be used together with the PC demonstration software to log all data that is available from SB201 and can therefore be used as a platform for characterizing batteries.

13.1.1 Full charge capacity

The full charge capacity of the battery is often different from the design capacity. The battery will update the Full Charge Capacity if it determines that it is not correct.





The full charge capacity as well as the design capacity should be the amount of capacity that is available at a very low current, i.e. about 10mA.

This parameter is stored in EEPROM in the `battery_pack_parameters` module.

13.1.2 State of charge vs. voltage

The voltage based gas gauging uses a table of the OCV at different SoCs from 0% to 100%. SB201 does a linear interpolation between the two closest points in the table, so the SoCs don't have to be at a fixed interval. This allows having fewer values between 70% and 30% where the OCV is almost linear for most batteries, and more values near the end of the discharge where the voltage changes a lot more. Note that SB201 requires having values at exact 100% and 0%.

The table is stored in the `voltage_based_SoC` module.

13.1.3 Remaining capacity compensation

Because of the internal battery resistance and the fact that it increases towards the end of a discharge, it is not possible to get as much capacity out of a battery at higher currents. Therefore SB201 needs to know how much less capacity that is available at different currents to know how much longer the battery can sustain the current load.

SB201 supports an arbitrary number of currents in the remaining capacity compensation table and will interpolate between the two closest. If the current is higher than the highest value in the table, it will predict the compensation value by using the two highest values.

The internal resistance of the battery varies with temperature, but that is yet not supported by this implementation.

14 Cell balancing (only for SB201-2)

When using two cell battery packs it is important that they are balanced with regard to cell voltage to get maximum capacity from the battery pack. To understand why, understanding of the Lithium-Ion charge cycle is required:

The charger will charge with either the maximum charge current or the maximum charge voltage. In the beginning of the charge cycle, when the battery is almost empty, the charge current will reach the maximum limit. Hence, when the battery starts building a certain charge level, the charge voltage will increase and eventually reach the voltage limit (typically 4.2V per Li-Ion cell), and thus be the limiting factor for the charger. An intelligent charger will not allow the charge voltage of the battery pack to increase above this level and as a result the charge current will decrease. Those two stages are called constant current charge and constant voltage charge.

The problem is that when using two cells, the charger will have the voltage limit at 8.4V. If one cell is more charged than the other, i.e. in misbalance, it will reach the maximum charge voltage earlier than the other cell. In this case the most charged cell could have reached 4.2V, while the least charged is at 4.1V. In this situation the voltage of the battery stack is 8.3V, and the charger will not notice that one of the battery cells have reached its maximum charge voltage. The charger will continue the constant current charge and the voltage will continue to increase until the battery pack voltage is 8.4V. This means that the most charged cell can have reached 4.25V, which is the absolute maximum voltage for security reasons, and SB201 will disable the charge FET.

When this happens neither cell is fully charged as a Lithium-Ion cell should be charged at maximum charge voltage for quite some time. If the problem arises there is a high risk that the cell misbalance will continue to get worse and worse. As a result, if the cells are highly misbalanced, the charger will not be able to operate as intended, and the battery pack will not be charged to its maximum capacity.

The solution used by SB201 is to discharge the most charged cell until the voltage difference between the cells is low enough. The threshold, called `BATTPARAM_MISBALANCE_VOLTAGE_THRESHOLD`, is stored in EEPROM in the `battery_pack_parameters` module.

Since misbalancing is primarily an issue while charging a battery pack, cell balancing is only active while charging the battery. During discharging, activation of cell balancing would have limited effect. Be aware that if two highly misbalanced cells are operating together, it may take several charge-discharge cycles to balance the cells.

15 Power management

To save battery power, the ATmega8HVA/16HVA should sleep as much as possible in power-save mode. However, it should not enter Power-Save if a peripheral module, which does not operate in Power-Save mode, is being used. In this implementation that means the VADC and the timers.

The VADC might still be running at the end of the main loop if the main loop is executed faster than one VADC scan. In that case the ATmega8HVA/16HVA will enter ADC Noise-Reduction mode and wake up by the VADC interrupt when the conversion is finished.

The timers are used by the fast RC calibration and the communication. When they are used the part can only enter Idle mode. As the calibration is initiated from the firmware, it is always known when the calibration is running; It is however more difficult to determine if the communication is active, as this is caused by an external event.

A transmission is started by a start bit that triggers the external interrupt request. The interrupt routine then starts the timer to know when to sample the bits in the message. The challenge is that a start bit can be received after the check if communication is active. Solution is to set sleep mode to idle in the external interrupt routing.

The ATmega8HVA/16HVA will enter power-off if one of the following conditions occurs:

- The chip temperature is outside the ATmega8HVA/16HVA. See 3.2.2
- The battery voltage is very low. See 3.2.3

16 Communication

SB201 have a 1-wire half-duplex UART to communicate with the host/application. The host has to initiate the communication and SB201 will communicate only if asked. On some errors SB201 stops to send, therefore the host has to have some sort of timeout to detect this.

The command set is the SBS specification 1.1 from <http://www.sbs-forum.org>. The specification is not followed completely, some commands are not fully supported and some reserved commands are used. See chapter 17 for the list of commands.





16.1 1-wire UART

The 1-wire UART uses the external interrupt request 0 and timer 1. The external interrupt can wake the device from power-save and is used to detect a start bit. After that the timer compare interrupt is used to time when to read or send the next bit. Because of that it is very important that all other interrupts are as short as possible (preferably less than a half bit period) to ensure proper communication.

Erroneous timing of the UART timer interrupt can cause both receive and transmit errors, and therefore it is important to have some sort of error checking. The standard protocol used by SB201 uses a CRC8 to verify messages.

The UART is configured to run at 4800 baud, i.e. bits per second, and with 1 stop bit and no parity.

16.2 Protocol

The SBS specification uses the SMBus protocol, but since SB201 uses software UART, some changes have been made to improve reliability. Addressing has also been removed.

SB201 supports the four different types of messages used by SBS:

- Read word
- Write word
- Read block
- Write block

A word command always has two bytes of data and a block command always 32 bytes, which is different from the SBS block commands that has variable length.

The host sends a command where the MSB determines if it's a write or read. 0 means write and 1 means read. This allows for 128 different commands that both support read and write. From the command itself it is not possible to determine if it is a word or block command; both sides have to agree what all commands are.

SB201 immediately sends back the command; this allows the host to detect if SB201 received the command correctly. What happens after that depends on the type of message used by that command.

A Packet Error-checking Code (PEC) is appended to all data to be able to detect errors. The PEC is a CRC8 with polynomial 0x8D (CRC8-CCITT) calculated from the command and all data bytes.

16.2.1 Read commands

On read commands, SB201 starts to send data after it has sent back the command. The PEC is also appended. However, if the command wasn't supported, nothing is sent back (except for the command byte).

16.2.2 Write command

On write commands, the host can start to send data to SB201 after it has received the replied command byte. After SB201 has received the whole message, it will execute the command if it was understood and the PEC was correct. It will send an ACK back to the host after it has finished executing it. If the command was supported but the PEC wrong, a NACK is sent back. On all other errors nothing is sent back.

ACK is 0xFF and NACK is 0x00.

Figure 16-1 A read word command

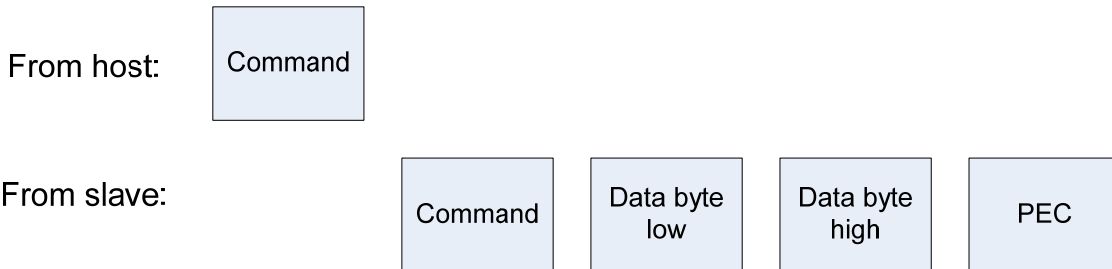


Figure 16-2 A write word command

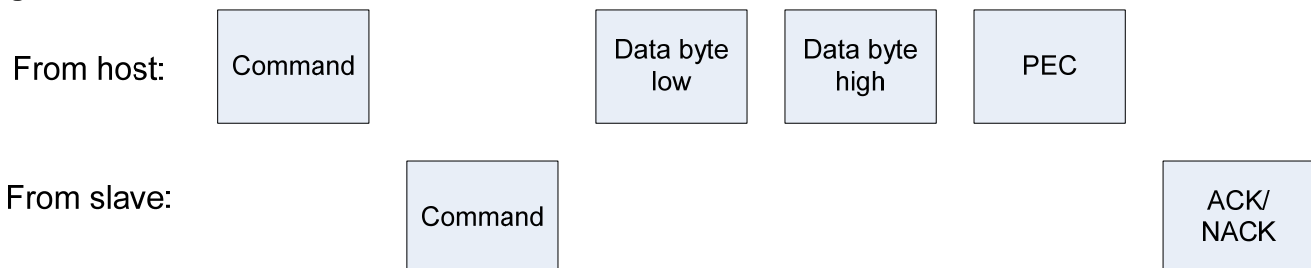


Figure 16-3 A read block command

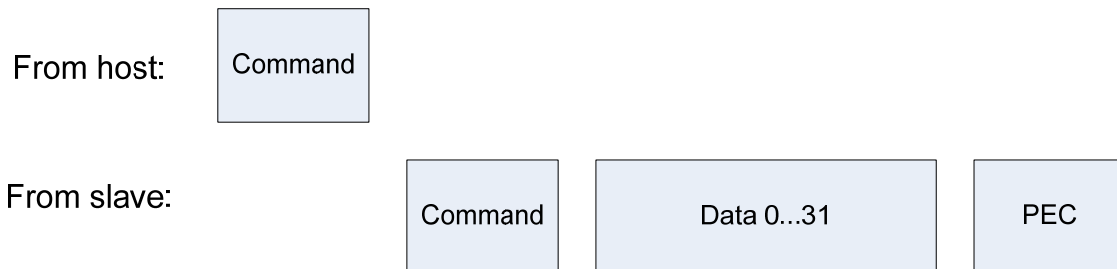
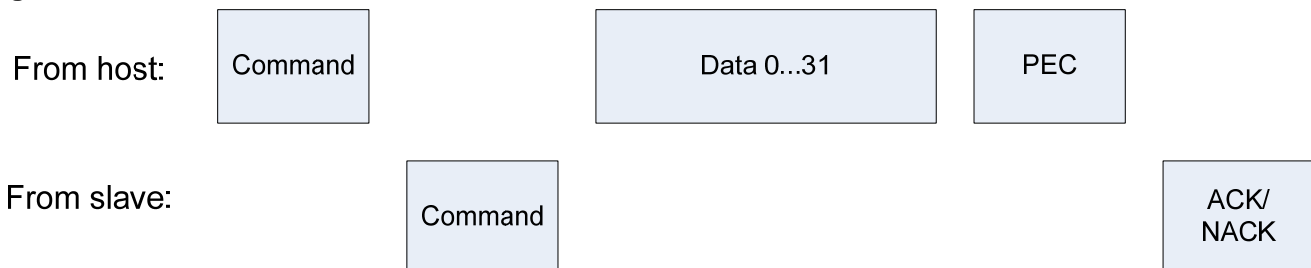


Figure 16-4 A write block command



16.2.3 Errors

At 4800 baud, there is only 208 clock cycles for SB201 between each bit. This means that to ensure error free communication all the time, all other interrupts would have to be less than 104 cycles, including the prologue of the timer interrupt. This is not





always true for the interrupt service routines used by SB201. However, since those interrupts (CCADC & VADC conversion complete) only runs once per second, and since the blocking interrupts has to occur just a few cycles before the timer interrupt, a retransmission of the “broken” communication is sufficient and adequate to solve this.

The errors the host can detect include:

- Wrong command returned by SB201
- Frame error on data sent from SB201
- PEC incorrect on data sent from SB201
- SB201 does not respond within reasonable time (min 50ms)
- SB201 sends a NACK to indicate an error occurred

On all those errors the host should send a break/arbitration lock and retransmit. A break/arbitration lock is a minimum of 18 normal bit periods of low on the communication line. This resets the communication on SB201 and makes it ready to receive a new transmission. A break can be sent as soon as the error is detected.

If the same command fails many times in a row, that command is probably not supported by SB201 or something serious is wrong. It might be that the frequency of the clock on the host or SB201 has changed too much or maybe that SB201 have shut itself down.

16.3 Authentication

SB201 contains both an AES encryption algorithm and a SHA256 based HMAC that can be used for challenge-response authentication. It allows the host to make sure the connected battery is a model it is compatible with.

Both algorithms cannot be used at the same time, the decision has to be made at compile-time with the use of the two defines `AUTH_USE_AES` and `AUTH_USE_HMAC_SHA2`. All SB201 are programmed with AES at assembly and therefore have to be recompiled to use HMAC-SHA256.

AES uses a 16-bytes (128-bit) key and challenge and response size of 16-bytes. HMAC-SHA256 uses a 32-bytes (256-bit) key and challenge and response size of 31-bytes. Using AES and HMAC-SHA2 is identical except for the difference in challenge and response size.

The battery and host both have to know the key in advance. Typically all batteries of the same model will have the same key preloaded and the host maybe a couple of keys for different battery models.

The key preprogrammed into SB201 at assembly for AES is “You cannot pass!”. The default key for HMAC-SHA256 is “This is a long key for HMAC-SHA2”. But a real application should use a key not only consisting of printable characters.

Both AES and HMAC-SHA256 are considered secure and are approved by, among others, NSA. No known attack (other than brute-force) is known for either one (as of May 2008). Both algorithms run in constant time to avoid timing based side-channel attacks. Nevertheless, the strength of the authentication also relies on other factors which may be exploited in side-channel attacks, such as how the key is stored in the memory, how the challenge response is performed and much more. If a very high security level is required the application developer is encouraged to learn good cryptology practice and apply this.

From a pure brute force perspective the authentication methods used in this application note will be quite strong: On the AES 128-bit key, a brute-force attack would take on average 2^{127} ($1.7 \cdot 10^{38}$) tries to find the right key. If someone wants to break the key in 1 year ($32.6 \cdot 10^6$ sec), and uses 1 million (10^6) computers in parallel, it will demand that each computer can do:

$$\frac{\text{Keys}}{\text{sec} \cdot \text{computers}} = \frac{1.7 \cdot 10^{38}}{32.6 \cdot 10^6 \cdot 10^6} = 5.2 \cdot 10^{24} \text{ key tries / sec}$$

Which are approximately 5200 million-million complete AES encryptions of 16 bytes using a 128-bit key per nano-second. So, even with a good portion of luck one should not expect to identify the key within a fair amount of time based on brute force attacks. Still, it is left up to the designer to ensure that the cryptographic method used is sufficiently strong for the desired purpose and that it is used at best practice.

16.3.1 Authentication procedure

The host first has to ask what model the battery is, so that it knows which key to use. Next step is to send random data to the battery. The battery then executes the AES or HMAC-SHA256 algorithm on the data with the shared key. When the host reads back the response from the battery it can verify that the battery has used the correct key and can be quite sure it is the model it said it was.

Step-by-step:

1. The host sends the authenticate command (a SBS write block command) to the battery. The first byte is always 0x01 in this implementation (it is used by the battery to know when a new challenge has arrived). After that 16-bytes of random data should be sent for AES and 31 bytes for HMAC-SHA256.
2. The battery then runs the algorithm on the received data. Note that HMAC-SHA256 also includes the status byte (as the first byte in the data) in the calculations, so when verifying the response, the host also has to do that. It also means that the message length used when padding in the SHA256 algorithm should be calculated from a challenge length of 32-bytes.
3. The host sends the authenticate command again to read back the changed data, but this time as a SBS read block command. The first byte is a status byte (described in Table 16-1) and the rest is the response. For HMAC-SHA256 the last byte of the response has to be truncated to fit in the 32-bytes block message.
4. The host then verifies the data from the battery and checks that the algorithm was run with the correct key.

Table 16-1: Read authentication status byte.

Status byte	Description
0x01	Authentication has not started yet
0x02	Authentication has started, but is not finished
0x04	Authentication is finished
0x08	An error occurred

16.3.2 Importance of random data

It is very important that the host does not send the same challenge every authentication. If it does, a battery can hardcode the response and don't have to





know the key, rendering the authentication useless for determining if a compatible battery is connected. Same if only a few different data combinations are used.

The best way to protect against this sort of attack is to send pseudo-random data concatenated with some unique data for each host/application. The unique data can for example be a serial number. Much more info about this can be found on the internet and in books about cryptology, search for "replay attack".

17 List of communication commands

17.1 SBS compliant commands

The following commands are almost implemented as defined in the Smart Battery Data specification Rev1.1 available at <http://www.sbs-forum.org>. SB201 does not support using and returning data in power (mW) instead of current (mA) so the commands that should support both only supports current.

- RemainingCapacityAlarm
- RemainingTimeAlarm
- AtRate
- AtRateToFull
- AtRateToEmpty
- AtRateOK
- Voltage
- Current
- AverageCurrent
- RelativeStateOfCharge
- AbsoluteStateOfCharge
- RunTimeToEmpty
- AverageTimeToEmpty
- AverageTimeToFull
- ChargingCurrent
- ChargingVoltage
- CycleCount
- DesignCapacity
- DesignVoltage
- SpecificationInfo
- ManufacturerDate
- SerialNumber
- ManufacturerName
- DeviceName
- DeviceChemistry
- ManufacturerData

17.2 Non SBS-compliant commands

Most of these commands exist in the Smart Battery Data specification, but they are implemented a bit differently in SB201. Some new commands are used, using either the optional SBS commands or reserved commands.

ManufacturerAccess	0x00
Read and write word. This command does nothing and will always read as 0x0000.	

BatteryMode	0x03
Read word. The bits defined and described in SBS are used, but they are not set to their default values (as SB201 doesn't support the default settings) and changing them is not possible. Some reserved bits are replaced with cell balancing information and FET status.	
Bit	Description
15	CAPACITY_MODE Set to 0 – "Report in mA or mAh"
14	CHARGER_MODE Set to 1 – "Disable broadcasts of ChargingVoltage and ChargingCurrent" as SB201 cannot initiate communication.
13	ALARM_MODE Set to 1 – "Disable AlarmWarning broadcast" as SB201 cannot initiate communication.
12	Reserved
11	BALANCING_CELL2 Set when cell 2 is being discharged to keep the cells balanced.
10	BALANCING_CELL1 Set when cell 1 is being discharged to keep the cells balanced.
9	PRIMARY_BATTERY Set to 0 – "Battery operating in its secondary role"
8	CHARGE_CONTROLLER_ENABLED Set to 0 – "Internal Charge Control Disabled"
7	CONDITION_FLAG Set to 0 – "Battery OK"
6	Reserved
5	DUVRD from FCSR Is set to the Deep under-voltage Recovery Disabled bit from the FET Control and Status Register
4	CPS from FCSR Is set to the Current Protection Status bit from the FET Control and Status Register
3	DFE from FCSR Is set to the Discharge FET Enabled bit from the FET Control and Status Register.
2	CFE from FCSR Is set to the Charge FET Enabled bit from the FET Control and Status Register
1-0	Not used





Temperature	0x08
<p>Read word.</p> <p>This should return the battery pack temperature; in SB201 it returns the chip temperature in 0.1 Kelvin.</p> <p>TemperatureNTC1/TemperatureNTC2 can be used for getting the battery temperature.</p>	

MaxError	0x0C
<p>This command is not supported.</p>	

RemainingCapacity	0x0F
<p>Read word.</p> <p>According to the specification, this should return the remaining capacity at 0.2C discharge current. SB201 will return the remaining capacity at the average current.</p>	

BatteryStatus	0x16
<p>Read word.</p> <p>This command is basically implemented as specified in SBS, except that reserved bits are used and the error codes are not supported.</p>	
Bit	Description
15	<p>OVER_CHARGED_ALARM</p> <p>Controlled by the chargingProhibited flag</p>
14	<p>TERMINATE_CHARGE_ALARM</p> <p>Controlled by the voltageTooHigh flag</p>
13	<p>VREGMON_TRIGGERED</p> <p>Reserved bit in SBS, is set if the voltage regulator monitor triggered since last time BatteryStatus was read. It is cleared when read.</p>
12	<p>OVER_TEMP_ALARM</p> <p>Controlled by the cellTemperatureTooHigh flag</p>
11	<p>TERMINATE_DISCHARGE_ALARM</p> <p>Is set when remaining capacity is below a defined limit (SBS_TERMINATE_DISCHARGE_LIMIT) and cleared when a charge is detected. Recommended to use REMAINING_CAPACITY_ALARM instead as it is configurable at runtime</p>
10	<p>BATTERY_PROTECTION_TRIGGERED</p> <p>Is set if the battery protection hardware module has disabled the FETs since last time this command was read. Will be cleared when read.</p>
9	<p>REMAINING_CAPACITY_ALARM</p> <p>Is set if the remaining capacity is lower then the value set by the RemainingCapacityAlarm command. Updated once every minute.</p>
8	<p>REMAINING_TIME_ALARM</p> <p>Is set if AverageTimeToEmpty is less than the value set by the RemainingTimeAlarm command. Updated once every minute.</p>

7	INITIALIZED Controlled by be inverse of the criticalConditionDetected flag. Since both FETs will be disabled when that flag is set, the host/application can only detect that criticalConditionDetected is set if it also is connected to a charger.
6	DISCHARGING Is set if the current the last second was a discharge.
5	FULLY_CHARGED Is set when the battery is considered fully charged and cleared as soon as a discharge current higher than standby is detected.
4	FULLY_DISCHARGED Is set when there is no capacity left in the battery at all. TERMINATE_DISCHARGE_ALARM and the remaining capacity/time alarm functions can be used to get earlier warnings. Is cleared when the RelativeStateOfCharge is above 20%
3-0	Not used

Authentication	0x24
Read and write block. Described in 16.3	

ShuntCalibration	0x2A
Read and write word. Is used to set the shunt resistor value in EEPROM. The value is set in microOhms and supports values between 4000 and 16000 uOhm; however, no check is done on the value. Reading this command can be used to be sure the write command was successfully received.	

FETDisable	0x2B
Write word. This command can be used to individually force the charge and discharge FETs to be disabled, which can be useful for demonstration and testing purposes. If bit 0 is high in the word, the charge FET will be disabled and kept disabled until this command is sent again with bit 0 low. Bit 1 works the same way for the discharge FET. Note that this command cannot force the FETs to be enabled; it can only force them to be disabled.	

StorageMode	0x2C
Write word This command will force ATmega8HVA/16HVA to enter power-off mode. To make sure a power-off really is wanted, the sent word has to be 0xFADE or this command will do nothing.	

TemperatureNTC2	0x2D
Read word. If a second NTC is connected, this will return its temperature in 0.1 Kelvin. If no NTC is connected, 0 Kelvin is returned	





TemperatureNTC1	0x2E
Read word. If the first NTC is connected, this will return its temperature in 0.1 Kelvin. If no NTC is connected, 0 Kelvin is returned	

VoltageCell2	0x3E
Read word. Returns the voltage over cell 2 in mV. If only one cell is used, it will return 0.	

VoltageCell1	0x3F
Read word. Returns the voltage over cell 1 in mV.	



Headquarters

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

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
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

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2008 Atmel Corporation. All rights reserved. Atmel®, logo and combinations thereof, AVR® and others, are the registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.