AVR105: Power Efficient High Endurance Parameter Storage in Flash Memory

8-bit AVR® Microcontroller

Application Note

Features

- **Fast Storage of Parameters**
- **High Endurance Flash Storage – 350K Write Cycles**
- **Power Efficient Parameter Storage**
- **Arbitrary Size of Parameters**
- **Semi-redundant Parameter Storage**
- **Optional Verification of Written Parameters**
- **Optional Recovery on Power Failure**

Introduction

Embedded systems are relying on parameters that can be preserved across RESET or power loss. In some systems this static information is used to initialize the system to a correct state at start-up, in other systems it is used to log system history or accumulated data. EEPROM memory can be used for this, but cannot match the speed of Flash memory when multiple bytes need to be stored at the same time.

The reason that Flash memory is more efficient for larger parameter sets is that page programming can be used, which decreases the programming time. The programming time per byte is thereby lower for Flash than for EEPROM when storing multi-byte parameter sets. As a direct result of a faster storage method, the power consumption can be reduced since more time can be spent in sleep mode.

This application note describes how to implement a high endurance parameter storage method in Flash memory using the self-programming feature of the AVR. By utilizing an entire Flash page and an access method similar to the one used for circular buffers, each single memory location in the Flash page is not write-accessed as often as if only one memory location was used. This approach increases the endurance of the storage and ensures that the storage area is not “worn out”. The storage endurance is proportional to the ratio between the size of the parameter set and size of the page allocated for the storage “buffer”.

Rev. 2546A–AVR–09/03

Theory of Operation

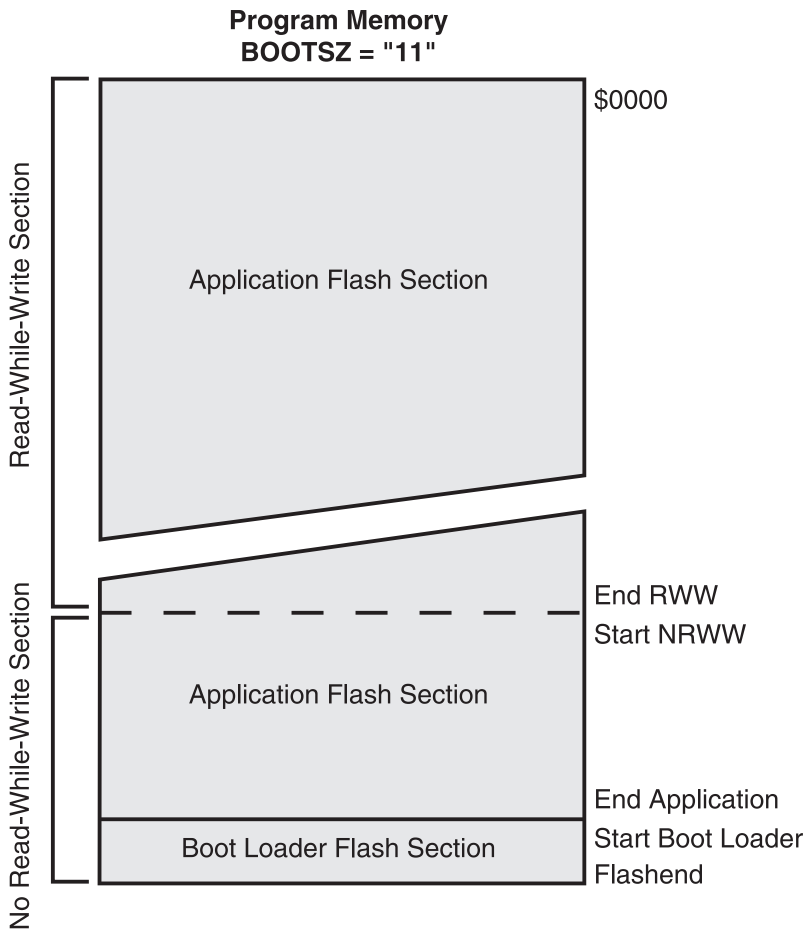
The megaAVR® series has a feature called “self-programming”. This feature makes it possible for the AVR to reprogram the internal Flash program memory. The program memory can be used in all AVRs to store constants, and now also parameters due to the possibility to change the contents of the Flash at run-time.

Using the Flash memory for parameter storage is however not quite as simple as for example interfacing the EEPROM. The self-programming feature is intended to be used for firmware updating, but its flexibility allows it to be used for Flash parameter updating as well. This section describes basic information about Flash memory that is required to use the AVR’s internal program memory for parameter storage.

Read-While-Write Flash

The Flash memory can be reprogrammed using the SPM instruction. The SPM instruction can only be executed from the Boot section of the memory. Executing the SPM instruction from the application section will have no effect. The Application section is located from address 0x0000 and up to the start of the Boot section (see Figure 1). Four different Boot section sizes can be selected; the Boot section size is determined by the fuse setting. The boot sizes that can be selected are depending on the AVR used.

Figure 1. Memory Map of the ATmega128 Application and Boot Sections



The Boot section is always located in the part of the Flash referred to as the No-Read-While-Write (NRWW) section. The Application always includes the part of the Flash called the Read-While-Write (RWW) section (see Figure 1), but can, depending on the selected size of the Boot section, also include some of the NRWW part as well. When the maximum Boot section size is selected the Boot section is taking up the entire NRWW part and the Application section is therefore purely RWW memory. The NRWW and the RWW parts are fixed and are not affected by the size of the Boot section. Refer to the datasheet of a device with self-programming for more details about this.

A difference between the NRWW and the RWW part is that while erasing or writing pages in the RWW section, the AVR can continue to execute program code located in the NRWW part of the memory. This is not possible when erasing or writing pages in the NRWW part: the AVR core is halted while modifying pages in the NRWW part of the Flash. Roughly speaking, code located in the boot section can execute while the application section is being reprogrammed, assuming that the page being modified in the application section is located in the RWW memory.

Consequently, the part of the code updating the Flash parameters must be located in the Boot section, and the Flash parameters must be located in the RWW part of the application section if the AVR is to continue operating while updating the parameters. This is desired for instance if interrupts must not be blocked while writing the parameters.

Erasing and Programming a Flash Cell

Flash memory consists of independent cells each representing a single bit. The Flash cells are base on floating gate transistor technology: an electrical charge “trapped” on the transistor gate is determining the logic level read of the Flash cell. Slightly simplified, the way that the Flash works can be described as follows: When “erasing” a cell, a charge is placed on the gate and the cell is read as logic one. “Programming” the Flash is equivalent to discharging the gate, bringing the logic value to zero. It is only possible to program (discharge) a cell that has been erased (charged).

The Flash is arranged in pages. Erasing and programming the Flash is, when using Self Programming, done in pages. A Flash erase is performed on an entire page and a Flash write *can* be performed on an entire page.

Note that bits can be programmed individually. Since only the bits being programmed are discharged, the remaining unprogrammed bits are still charged. Any unprogrammed bit can be programmed at a later stage. Therefore, programming a byte that is already programmed, without erasing it in between, will result in a bit-wise AND between the old value and the new value.

Writing the value 0x01 to a Flash byte requires that the byte (8 Flash cells) is first erased, which makes the byte take the value 0xFF. To write (program) the value, the 7 most significant bits (Flash cells) are discharged. If the Flash is not erased in advance, it may not be possible to program it to the intended value: Assume that the value of the byte was 0xFE and that it was programmed with 0x01; the result would be 0x00, since the LSB could not be changed from zero to one.

Flash Endurance

The fact that a parameter(1) can be stored several times within a single Flash page makes the Flash suitable for parameter storage; the Flash cells have a guaranteed endurance of 10,000 erase/write cycles. That is, each cell can be erased and programmed 10,000 times. Therefore, if a parameter can be written 10 times in one page by using different locations each time, the endurance of the storage (page) as a whole can be seen as 100,000 write cycles. This is because each cell is only erased and written 10,000 times by moving the parameter around within the page.

Note: 1. The term “parameter” in this context covers both single parameters and sets of parameters. “Parameter” can thus refer to a set of data of arbitrary size.

Write Time

When writing to EEPROM one byte is written at a time. This is not the case when using Flash since page programming is utilized. It is therefore an advantage to use Flash storage when the size of the parameter is larger than a single byte. The larger the parameter, the less time is used per stored byte.

Writing a parameter of type *long* (4 bytes) takes approximately 4 ms (excluding the erase time, which is of the same duration). It thus takes 1 ms per byte when writing a long parameter to the Flash. In comparison it takes 32 ms to write the same amount of data to the EEPROM(1). For the EEPROM the erase is included, but the comparison is fair, since the erase of the Flash is only done when all locations in the Flash pages are used up: Consider using Flash parameter storage with an ATmega128. The page size for this device is 256 bytes. It is possible to store a long variable 256/4 = 64 times in one page. The duration of the erase should therefore be averaged over 64 writes to determine the average erase-write time for the storing of the long variable. The erase time of approximately 4 ms divided by 64 is … not much.

The conclusion is therefore that a larger parameter set is more efficient in Flash storage in terms of write time – thus more time can be spent in sleep modes, saving valuable power.

Note: 1. Using ATmega128 as example, the EEPROM programming time is device dependent – please refer to the datasheet of the used device for these details.

Flash Write Access Constraints

The Flash and the EEPROM share modules for erasing and programming memory. The resulting constraint is that the EEPROM and Flash cannot be written (or erased) at the same time. This means that before using the self-programming feature to modify the program memory, it should be tested whether an EEPROM write cycle is ongoing. If so, the self-programming must wait for the EEPROM write-access to finish. This also applies the other way around; EEPROM write access must wait for Flash write-access to finish.

Flash Data Retention Time

The duration of time that a programmable memory can preserve the correct values is referred to as the *data retention time* (or just data retention). The reason why the memory does not have infinite data retention is that the memory cells are leaking. This means that the charge stored in the cells is weakened over time and at some instance the charge no longer represents the logic level that it should have. Both the erased and the programmed cells are leaking towards an unpredictable state. If the data retention time is exceeded, the contents of the Flash memory become unreliable.

Decreased Data Retention Due to Erase and Write Access

When erasing and writing the Flash cells they are physically worn. Read access does not affect data retention. As the number of erase/write cycles increases, the leakage from the cells increases. The consequence is that the charge is weakened faster and that the data therefore will become invalid sooner – in other words, the data retention time is decreased. If the number of E/W cycles performed on a cell exceeds the guaranteed 10K E/W cycles, the data retention will decrease below the expected 20 years.

In relation to parameter storage in Flash it is important to know that the Flash pages are independent of each other; if one Flash page is used for parameter storage this page may have decreased data retention, while the rest of the Flash, used for program code, is not affected by the decreased data retention of the parameter page.

Power Loss Considerations

One must assume that any embedded systems can be exposed to power failures. This is one of the reasons for using non-volatile memory types for parameters; it will then be possible to recover the parameters after power cycling.

There are many different methods to verify that parameters have been written correctly. The preferred choice depends on the time and code space available to do the verification. A safe method that uses very little memory space and time, is keeping a write-complete flag in a static part of the memory. This flag can be used when recovering from a power failure to determine if the last write was correctly completed. If not, appropriate actions can be taken.

The considerations related to avoid Flash corruption resembles to those that applies for EEPROM corruption (refer to the AVR datasheet regarding voiding EEPROM corruption). To avoid Flash corruption due to power loss it is therefore recommended to always enable the Brown-out Detection feature when using the self-programming feature.

Using Self-programming

Details regarding the procedure used to perform self-programming can be studied in the device datasheets and in the application note ”AVR109: Self Programming”.

It is important to notice that a write-cycle to Flash is not stopped by an External RESET or a Brown-out RESET. Only the Power-on RESET will stop the ongoing Flash write cycle. The consequence is that the write cycle will continue as long as possible even if the supply voltage drops below minimum recommended operation voltage. This does not induce risk of data corruption – this feature increases the likelihood that the Flash page write/erase is completed.

Implementation

The implementation of the Flash parameter storage example is done using the IAR EWAVR 2.27b compiler. The implementation can be ported to other compilers, but this may require some work since several intrinsic functions from the IAR compilers have been utilized in the code example.

The aim for the code example was to make a driver that can store a multi-byte parameter quickly and also to obtain a high endurance for this storage method. The endurance of the storage depends on the size of the parameter and the size of the Flash page used. In the code example a 7-byte structure parameter is stored within a Flash page on the ATmega128. The page size on this device is 256 bytes. One page can hold up to 35 copies of the parameter, in addition to the write-completed flags. The guaranteed endurance of the Flash storage is thereby:

Endurance = Cell endurance * Copies = 10,000 * 35 = 350,000 writes.

Relying on typical data on Flash endurance rather than the minimum guaranteed endurance, one could expect up to ten times the minimum endurance – several million updates of the parameter.

Furthermore, it has been a focus to minimize the overall power consumption of an application using the Flash storage method. This is achieved by placing the parameters in the RWW part of the Flash memory. It is thereby possible to continue code execution from the NRWW part of the Flash while the parameters are updated, saving time. The code is made so that interrupts will continue to operate and interrupt driven code in the RWW section is therefore still executable.

Optional

To get an even more reliable storage method it is possible to “enable” the use of write-completion flags. If such a flag is used, the writing of the parameter and the parameter-location flag (which is also the write-completion flag) will be done in two separate write operations. It is in this way possible to determine that the parameter is stored correctly if the write-completion flag is programmed. However, the storing of the parameter and the write-completion flag will take twice the amount of time compared to storing both in one operation.

To increase the robustness of the storage method, the parameter is temporarily stored in EEPROM while the parameter page is erased. This is done to ensure that a power-failure between the erase operation and the write operation will not result in loss of the parameter. If a power failure occurs while erasing the page, the parameter will thus be recovered from EEPROM and then programmed into the parameter page.

These features are controlled by the FlashStorageDriver.c file and are by default enabled.

Requirements

The example is targeting the ATmega128 and is therefore adapted to the memory configuration of this device. If the example is to be used with other devices the linker command file and the device dependent information in the FlashStorageDriver.c file needs to be changed.

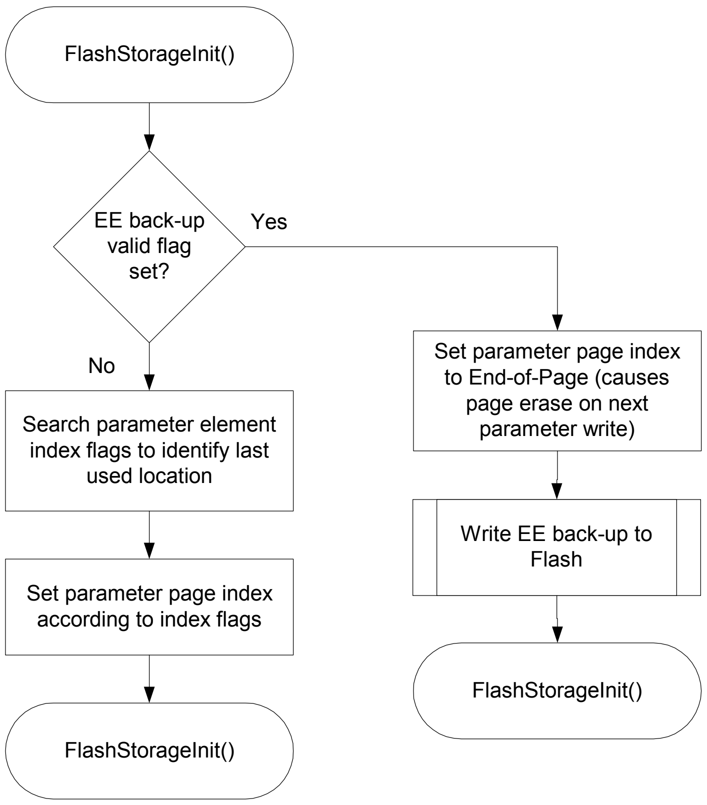
Firmware Description

The driver consists of three functions. These are described by the flowchart (Figures 2, 3, and 4) and the following sections.

FlashStorageInit

The initialization of the Flash storage is an investigation of the state of the Flash storage. If the parameter is stored in the temporary EEPROM storage (determined from the EEPROM Back-up Valid flag), the parameter is recovered from there and copied into the Flash storage. Otherwise the last used position in the Flash buffer is identified from the index flags.

Figure 2. Flash Storage Initialization Process



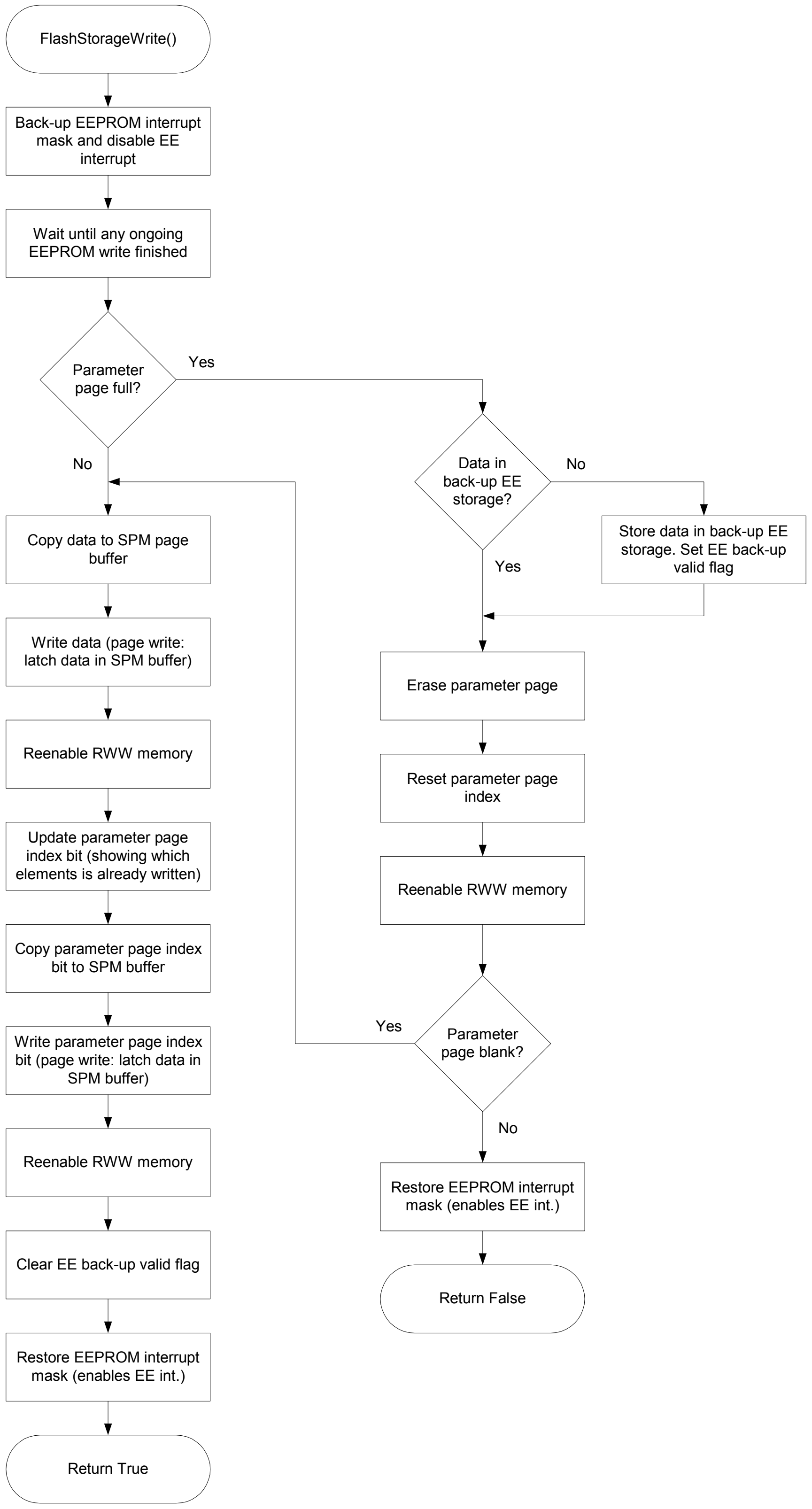
FlashStorageInit()
EE back-up valid flag set?
Yes
No
Search parameter element index flags to identify last used location
Set parameter page index according to index flags
FlashStorageInit()
Set parameter page index to End-of-Page (causes page erase on next parameter write)
Write EE back-up to Flash
FlashStorageInit()

FlashStorageWrite

The parameter storage routine will first inspect if EEPROM access is ongoing. If this is the case the routine will disable the EEPROM interrupt and wait for the access to finalize. If the parameter page is full, the parameter is first stored in EEPROM. Next, the parameter page is erased. When the EEPROM holds the parameter, the EEPROM Back-up Valid flag is set. Once the erase is completed, the parameter is written to the parameter page and the EEPROM Back-up Valid flag is cleared. After writing a location in the parameter page, the corresponding Parameter Index Bit is cleared. This way it is possible to determine during initialization that a parameter location is valid. Finally, the EEPROM interrupt is restored to it original state.

In addition to the operations listed above, the access control of the RWW part of the memory is handled so that the memory read access is enabled when returning from the function.

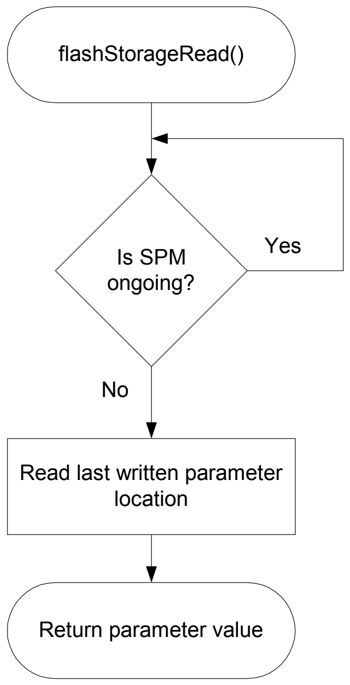
Figure 3. Flash Storage Write Process


FlashStorageWrite()
Back-up EEPROM interrupt mask and disable EE interrupt
Wait until any ongoing EEPROM write finished
Parameter page full?
Yes
No
Data in back-up EE storage?
No
Yes
Store data in back-up EE storage. Set EE back-up valid flag
Copy data to SPM page buffer
Write data (page write: latch data in SPM buffer)
Reenable RWW memory
Update parameter page index bit (showing which elements is already written)
Copy parameter page index bit to SPM buffer
Write parameter page index bit (page write: latch data in SPM buffer)
Reenable RWW memory
Clear EE back-up valid flag
Restore EEPROM interrupt mask (enables EE int.)
Return True
Erase parameter page
Reset parameter page index
Reenable RWW memory
Parameter page blank?
Yes
No
Restore EEPROM interrupt mask (enables EE int.)
Return False

FlashStorageRead

It is not possible for the application to read the Flash parameter directly, since to current location of the parameter is not known by the application. Therefore a special function is used to read the parameter. The function first verifies that the parameter can be read, that SPM is not ongoing, and then reads and returns the parameter from the parameter page.

Figure 4. Flash Storage Read Process



Literature List

1. AVR datasheets for devices with self-programming, available on the Atmel web site.
2. Application Note "AVR109 – Self programming", available on the Atmel web site.

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