AVR106: C functions for reading and writing to Flash memory

8-bit AVR® Microcontrollers

Application Note

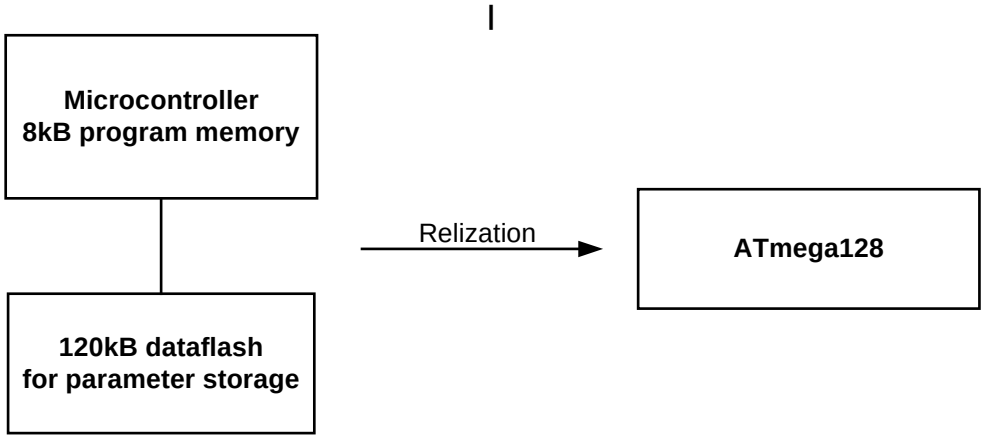
Features

- **C functions for accessing Flash memory**
  - **Byte read**
  - **Page read**
  - **Byte write**
  - **Page write**
- **Optional recovery on power failure**
- **Functions can be used with any device having Self programming Program memory**
- **Example project for using the complete Application Flash section for parameter storage.**

Introduction

Recent AVR®s have a feature called Self programming Program memory. This feature makes it possible for an AVR to reprogram the Flash memory during program run and is suitable for applications that need to self-update firmware or store parameters in Flash. This application note provides C functions for accessing the Flash memory.

Figure 1. Example of an application using the complete Application Flash section for parameter storage.



Rev. 2575B-AVR-08/06

Theory of operation

This section contains some basic theory around using the Self programming Program memory feature in AVR. For a better understanding of all features concerning Self programming, please refer to the device datasheet or application note “AVR109 Self Programming Flash”.

Using SPM

The Flash memory may be programmed using the Store Program Memory (SPM) instruction. On devices containing the Self Programming feature the program memory is divided into two main sections: Application Flash Section and Boot Flash Section.

On devices with boot block, the SPM instruction has the ability to write to the entire Flash memory, but can only be executed from the Boot section. Executing SPM from the Application section will have no effect. On the smaller devices that don’ have a boot block, the SPM instruction can be executed from the entire memory.

During Flash write to the Boot section the CPU is always halted. However, most devices may execute code (read) from the Boot section while writing to the Application section. It is important that the code executed while writing to the Application section do not attempt to read from the Application section. If this happens the entire program execution may be corrupted.

The size and location of these two memory sections are depending upon device and fuse settings. Some devices have the ability to execute the SPM instruction from the entire Flash memory space.

Write procedure

The Flash memory is written in a page-by-page fashion. The write is carried out by storing data for an entire page into a temporary page buffer prior to writing the Flash. Which Flash address to write to is decided by the content of the Z-register and RAMPZ-register. A Flash page has to be erased before it can be programmed with the data stored in the temporary buffer. The functions contained in this application note use the following procedure when writing a Flash page:

- Fill temporary page buffer
- Erase Flash page
- Write Flash page

As one can see of this sequence there is a possibility for loss of data if a reset or power failure should occur immediately after a page erase. Loss of data can be avoided by taking necessary precautions in software, involving buffering in non-volatile memory. The write functions contained in this application note provide optional buffering when writing. These functions are further described in the firmware section. For devices having the read-while-write feature, allowing the boot loader code to be executed while writing, the write functions will not return until the write has completed.

Addressing

The Flash memory in AVR is divided into 16-bit words. This means that each Flash address location can store two bytes of data. For an ATmega128 it is possible to address up to 65k words or 128k bytes of Flash data. In some cases the Flash memory is referred to by using word addressing and in other cases by using byte addressing, which can be confusing. All functions contained in this application note use byte addressing. The relation between byte address and word address is as follows:

- Byte address = word address • 2

A Flash page is addressed by using the byte address for the first byte in the page. The relation between page number (ranging 0, 1, 2…) and byte address for the page is as follows:

- Byte address = page number • page size (in bytes)

Example on byte addressing:

A Flash page in an ATmega128 is 256 bytes long.

Byte address 0x200 (512) will point to:

- Flash byte 0x200 (512), equal to byte 0 on page 2
- Flash page 2

When addressing a page in ATmega128 the lower byte of the address is always zero. When addressing a word the LSB of the address is always zero.

Implementation

The firmware is made for the IAR compiler. The functions may be ported to other compilers, but this may require some work since several intrinsic functions from the IAR compiler are used. Implementation is done by including the file **Self_programming.h** in the main C file and adding the file **Self_programming.c** to the project. When using Self-programming it is essential that the functions for writing are located inside the Boot section of the Flash memory. This can be controlled by the usage of memory segment definitions in the compiler linker file (***.xcl**). All other necessary configurations concerning the firmware are done inside the file **Self_programming.h**

Page size

The constant **PAGESIZE** must be defined to be equal to the Flash page size (in bytes) of the device being used.

Enabling Flash recovery

Defining the constant **__FLASH_RECOVER** enables the Flash recovery option for avoiding data loss in case of power failure. When Flash recovery is enabled, one Flash page will serve as a recovery buffer. The value of **__FLASH_RECOVER** will determine the address to the Flash page used for this purpose. This address must be a byte address pointing to the beginning of a Flash page and the write functions will not be able to write to this page. Flash recovery is carried out by calling the function **RecoverFLASH()** at program startup.

Defining Flash memory for writing

The memory range in which the functions are allowed to write is defined by the constants **ADR_LIMIT_LOW** and **ADR_LIMIT_HIGH**. The write functions can write to addresses higher or equal to **ADR_LIMIT_LOW** and lower than **ADR_LIMIT_HIGH**.

Placing entire code inside Boot section

It is necessary to redefine a range of segments defined inside the default ***.xcl** file in order to place the entire application code in the Boot section of Flash. The location and size of the Boot section varies with the device being used and fuse settings. Programming the BOOTRST fuse will move the reset vector to the beginning of the Boot section. It is also possible to move all the interrupt vectors to the Boot section. Refer to the interrupt section in the device datasheet for instructions on how to do this. The segment definitions that have to be redefined in order to place the entire program code into the Boot section is as follows:

TINY_F, NEAR_F, SWITCH, DIFUNCT, CODE, FAR_F, HUGE_F, INITTAB, TINY_ID, NEAR_ID and CHECKSUM.

The file **lnkm128s.xcl** provided with this application note will place the entire code into the 8kB Flash section of an Atmega128. This file can easily be modified to be used with other devices and provides instructions on how to do this.

Placing selected functions inside Boot section

Alternatively it is possible to place only selected functions into defined segments of the Flash memory. In fact it is only the functions for writing that need to be located inside the Boot section. This can be done by defining a new Flash segment equivalent to the Boot memory space and use the @ operator to place the desired functions into this segment. The @ operator does not apply to functions called inside the function it is used on.

Definition of Boot segment in *.xcl file for an ATmega128 with 8kB Boot size:

1. Make a new define for Boot size.

```
-D_..X_BOOTSEC_SIZE=2000  /* 4096 words  */
```

2. Define a new segment for the entire Boot section based on the definition in step 1.

```
-Z(FARCODE)BOOT_SEGMENT=(_..X_FLASH_END-_..X_BOOTSEC_SIZE+1)-
_..X_FLASH_END
```

Placing a C function into the defined segment:

```
void ExampleFunction() @ BOOT_SEGMENT {
     ------
}
```

The C-code above will place the function ExampleFunction() into the defined memory segment “BOOT_SEGMENT”.

Firmware description

The firmware consists of five C functions and one example project for IAR v 2.28a / 3.10c using an ATmega128. The example project is configured to have the entire program code located in the Boot section of Flash and can be used as a starting point for the application sketched in Figure 1.

Description of C functions

Table 1. C functions for accessing Flash memory.

Function	Arguments	Return
ReadFlashByte( )	MyAddressType flashAdr	unsigned char
ReadFlashPage()	MyAddressType flashStartAdr, unsigned char *dataPage	unsigned char
WriteFlashByte( )	MyAddressType flashAddr, unsigned char data	unsigned char
WriteFlashPage()	MyAddressType flashStartAdr, unsigned char *dataPage	unsigned char
RecoverFlash()	Void	unsigned char

The datatype **MyAddressType** is defined in **Self_programming.h**. The size of this datatype is depending upon the device that is being used. It will be defined as an **long int** when using devices with more than 64kB of Flash memory, and as a **int** (16 bit) using devices with 64kB or less of Flash memory. The datatypes are actually used as __**flash** or __**farflash** pointers (consequently 16 and 24 bit). The reason why a new datatype is defined is that integer types allow a much more flexible usage than pointer types.

ReadFlashByte() returns one byte located on Flash address given by the input argument.

ReadFlashPage() reads one Flash page from address ucFlashStartAdr and stores data in array pucDataPage[]. The number of bytes stored is depending upon the Flash page size. The function returns FALSE if the input address is not a Flash page address, else TRUE.

WriteFlashByte() writes byte ucData to Flash address ucFlashAddr. The function returns FALSE if the input address is not a valid Flash byte address for writing, else TRUE.

WriteFlashPage() writes data from array pucDataPage[] to Flash page address ucFlashStartAdr. The number of bytes written is depending upon the Flash page size. The function returns FALSE if the input address is not a valid Flash page address for writing, else TRUE.

RecoverFlash() reads the status variable in EEPROM and restores Flash page if necessary. The function must be called at program startup if the Flash recovery option is enabled. The function Returns TRUE if Flash recovery has taken place, else FALSE.

Flash recovery

When the Flash recovery option is enabled a page write will involve pre-storing of data into a dedicated recovery page in Flash, before the actual write to a given Flash page takes place. The address for the page to be written to is stored in EEPROM together with a status byte indicating that the Flash recovery page contains data. This status byte will be cleared when the actual write to a given Flash page is completed successfully. The variables in EEPROM and the Flash recovery buffer are used by the Flash recovery function **RecoverFlash()** to recover data when necessary. The writing of one byte to EEPROM takes about the same time as writing an entire page to Flash. Thus, when enabling the Flash recovery option the total write time will increase considerably. EEPROM is used instead of Flash because reserving a few bytes in Flash will exclude flexible usage of the entire Flash page containing these bytes.

Flowcharts

Figure 2. Flowchart for function ReadFlashByte().

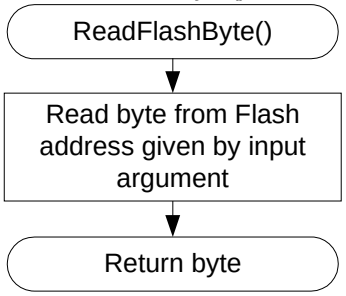


Figure 3. Flowchart for function ReadFlashPage().

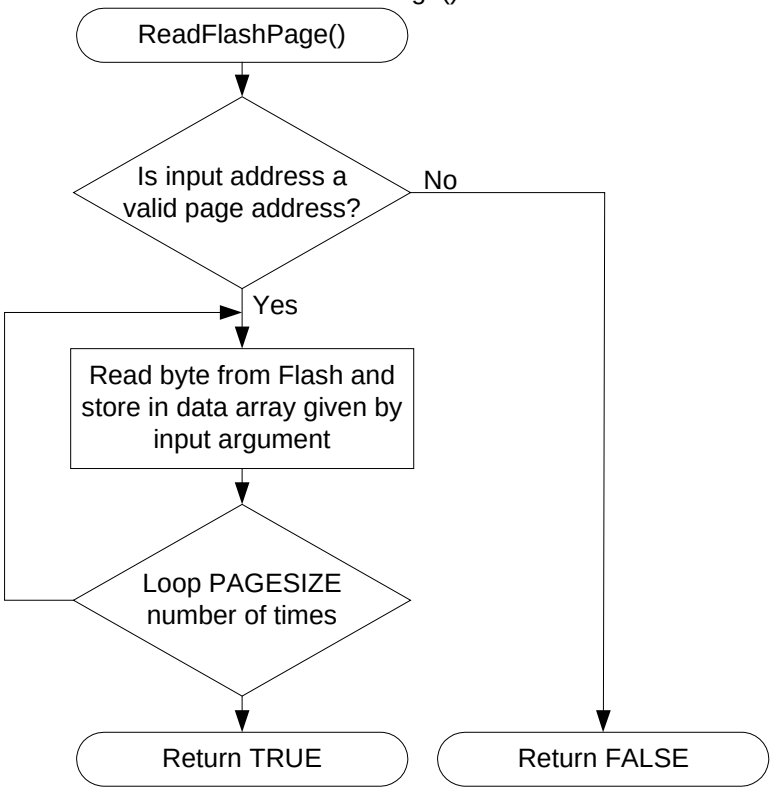


Figure 4. Flowchart for function WriteFlashByte().

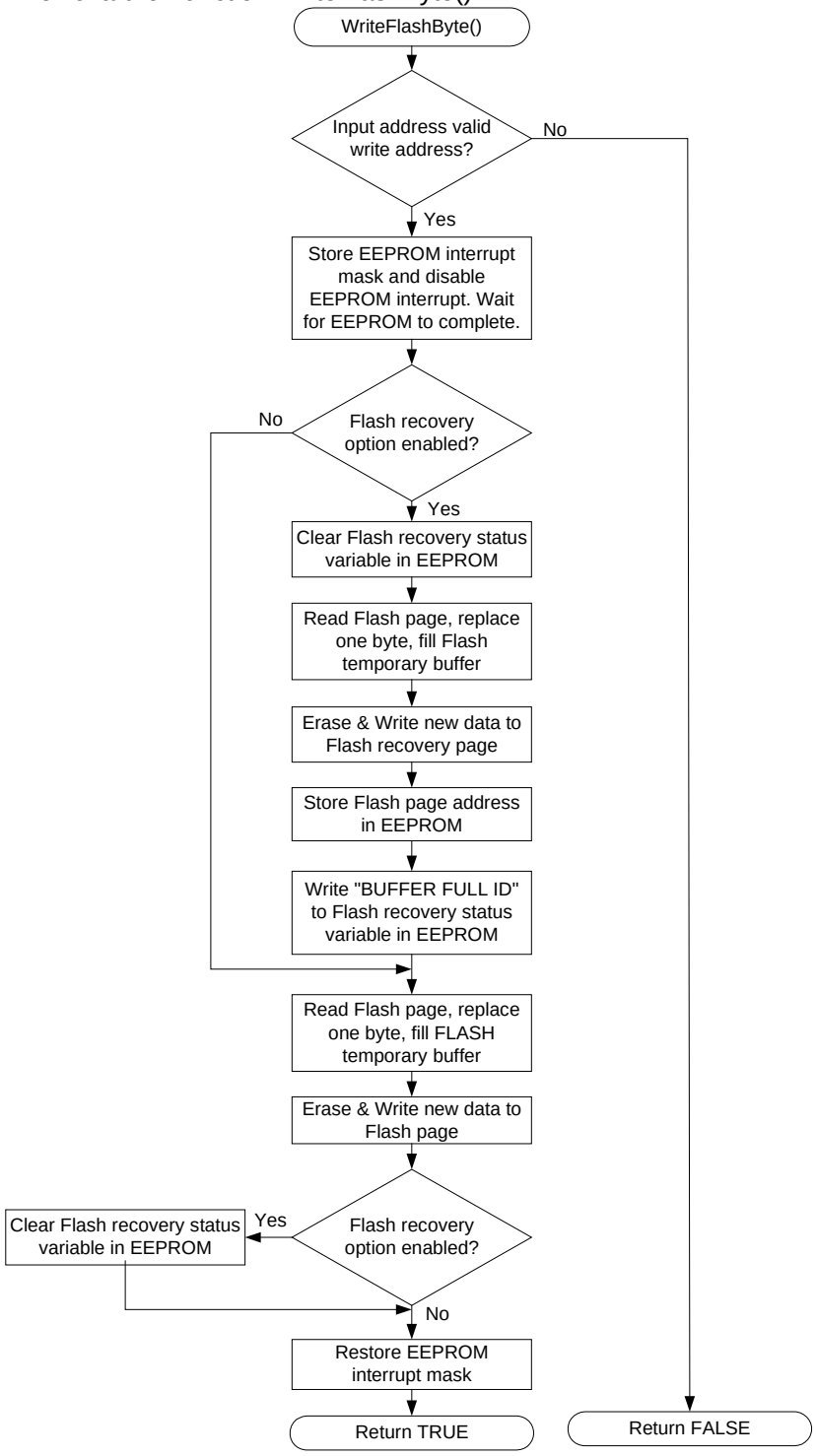


Figure 5. Flowchart for function WriteFlashPage().

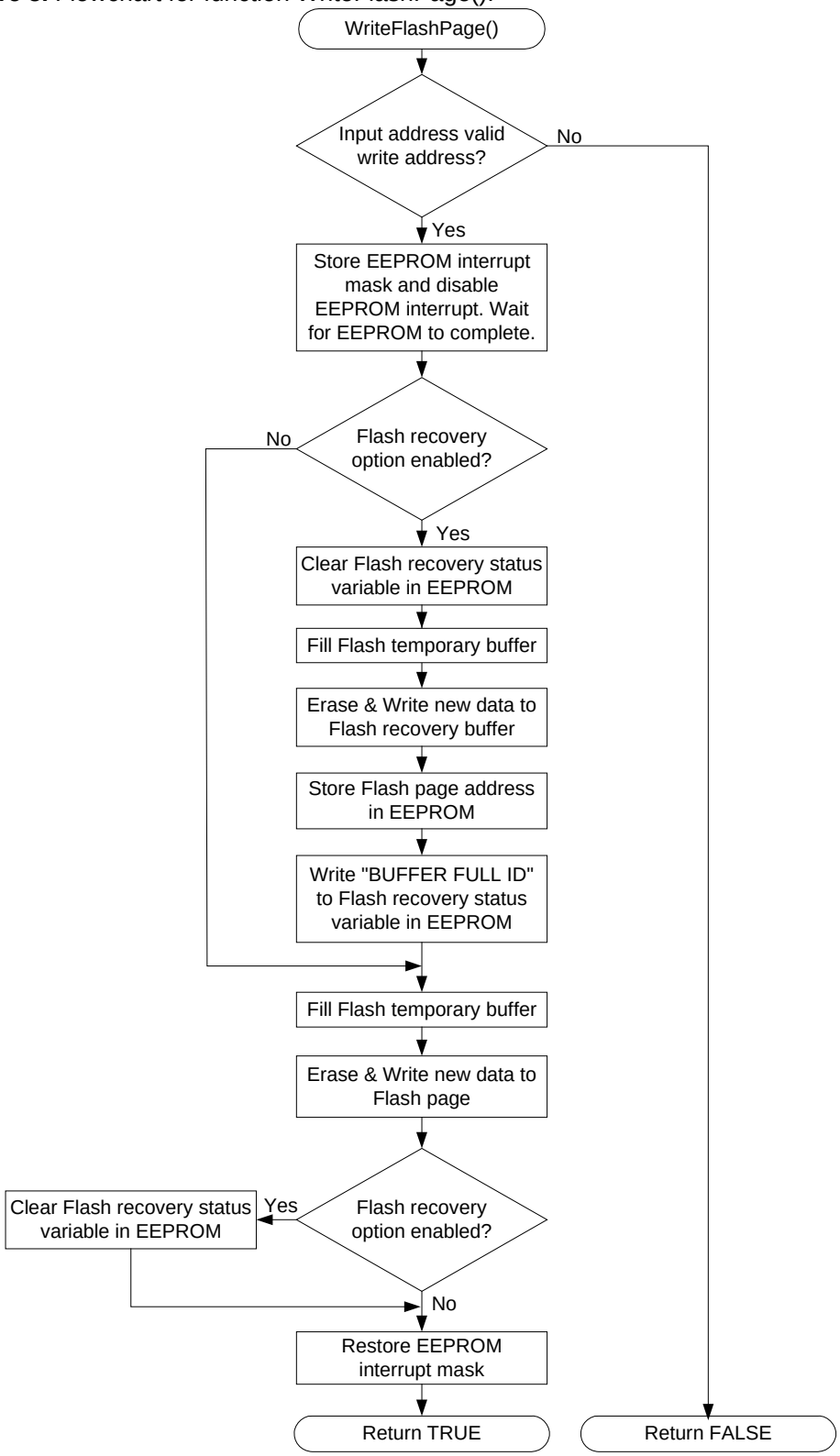
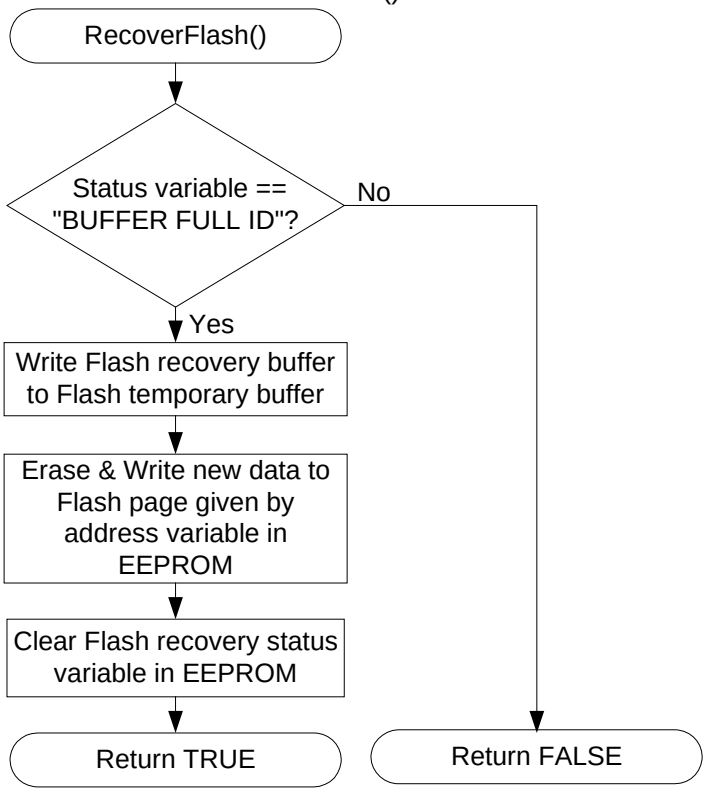


Figure 6. Flowchart for function RecoverFlash().



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

2575B-AVR-08/06