AVR329: USB Firmware Architecture

8-bit AVR® Microcontrollers

Features

- **Low Speed (1.5Mbit/s) and Full Speed (12Mbit/s) data rates**
- **Control, Bulk, Isochronuous and Interrupt transfer types**
- **Standard or custom USB device classes with AVR USB software library**
- **Up to 6 data endpoints**
- **Single or double buffering**

1. Introduction

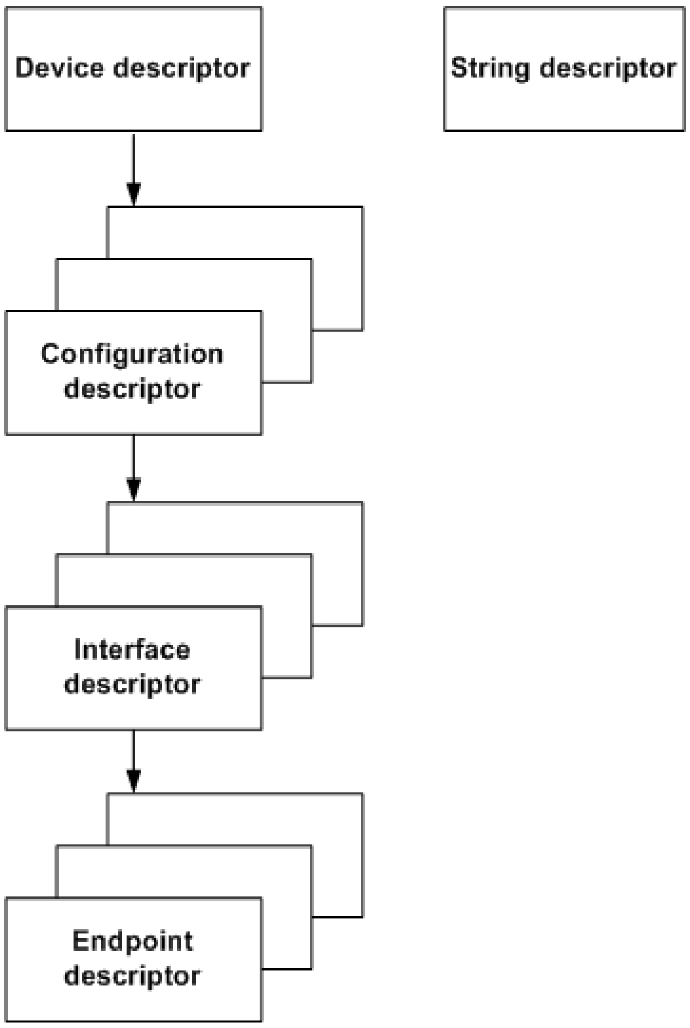
The aim of this document is to describe the USB firmware and give an overview of the architecture. The main files are described in order to give the user the easiest way to customize the firmware and build his own application. A minimun knowledge of chapter 9 of USB specification 2.0 (www.usb.org) is required to understand the content of this document.

7603A–USB–02/06

2. USB Descriptors

During the enumeration process, the host asks the device several descriptor values to identify it and load the correct drivers. Each USB device should have at the least the descriptors shown in the figure below to be recognized by the host:

Figure 2-1. USB Descriptors



2.1 Device Descriptor

The USB device can have only one device descriptor. This descriptor displays the entire device. It gives information about the USB version, the maximum packet size of the endpoint 0, the vendor ID, the product ID, the product version, the number of the possible configurations the device can have, etc.

The table hereunder shows the format of this descriptor:

Table 2-1. Device descriptor

Field	Description
bLength	Descriptor size
bDescriptorType	Device descriptor
bcdUSB	USB version
bDeviceClass	Code class (If 0, the class will be specified by each interface, if 0xFF, it is specified by the vendor)
bDeviceSubClass	Sub class code (assigned by USB org)
bDeviceProtocol	Code protocol (assigned by USB org)
bMaxPacketSize	The maximal packet size in bytes of the endpoint 0. It has to be 8 (Low Speed), 16, 32 or 64 (Full Speep)
idVendor	Vendor identification (assigned by USB org)
idProduct	Product identification (assigned by the manufacturer)
bcdDevice	Device version (assigned by the manufacturer)
iManufacturer	Index into a string array of the manufacturer descriptor
iProduct	Index into a string array of the product descriptor
iSerialNumber	Index into a string array of the serial number descriptor
bNumConfiguration	Number of configurations

2.2 Configuration Descriptor

The USB device can have more than one configuration descriptor, however the majority of devices use a single configuration. This descriptor specifies the power-supply mode (self_powered or bus-powered), the maximum power that can be consumed by the device, the interfaces belonging to the device, the total size of all the data descriptors , etc.

For example one device can have two configurations, one when it is powered by the bus and the other when it is self-powered. We can imagine also configurations which use different transfer modes.

The table hereunder shows the format of this descriptor:

Table 2-2. Configuration descriptor

Field	Description
bLength	Descriptor size
bDescriptor	Configuration descriptor
wTotalLength	Total descriptors size
bNuminterface	Number of interfaces
bConfigurationValue	Number of the configuration
bmAttributes	self-powered or bus-powered, remote wake up
bMaxpower	2mA by unit

2.3 Interface Descriptor

A single device can have more than one interface. The main information given by this descriptor is the number of endpoints used by this interface and the USB class and subclass.

The table hereunder shows the format of this descriptor:

Table 2-3. Interface descriptor

Field	Description
bLength	Descriptor size
bDescriptorType	Interface descriptor
bInterfaceNumber	interface number
bAltenativeSetting	Used to select the replacing interface
bNumEndpoint	Number of endpoints (excluding endpoint 0)
bInterfaceClass	Class code (assigned by USB org)
bInterfaceSubClass	Subclass code (assigned by USB org) 0 No subclass 1 Boot interface subclass
iInterface	Index into a string array to describe the used interface

2.4 Endpoint Descriptor

This descriptor is used to describe the endpoint parameters such as: the direction (IN or OUT), the transfer type supported (Interrupt, Bulk, Isochronuous), the size of the endpoint, the interval of data transfer in case of interrupt transfer mode, etc.

The table hereunder shows the format of this descriptor:

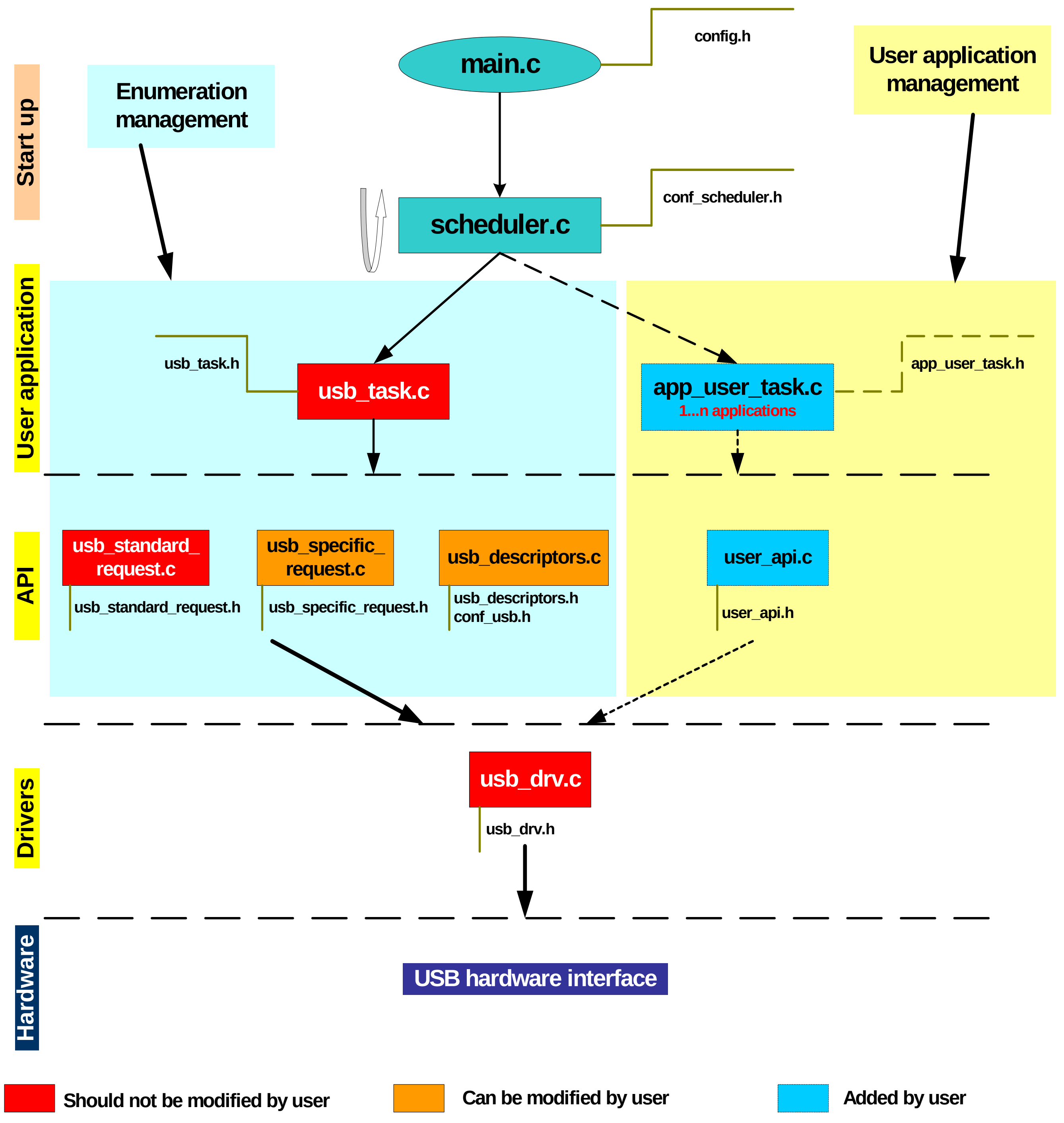
Table 2-4. Endpoint descriptor

Field	**Description**
bLength	Descriptor size
bDescriptorType	Endpoint descriptor
bEndpointAdress	Endpoint adress Bits[0..3] Number of the endpoint Bits[4..6] reserved, set to 0 Bit 7: Direction: 0 = OUT, 1 = IN
bmAttributes	Bits[0..1] Transfer type: 00=Control, 01=Isochronous, 10=Bulk, 11=Interrupt Bits [2..7] reserved, except for Isochronous transfer Only control and interrupt modes are allowed in Low Speed
wMaxPacketSize	The maximum data size that this endpoint support
bInterval	It is the time interval to request the data transfer of the endpoint. The value is given in number of frames (ms). Ignored by the bulk and control transfers. Set a value between 1and 16 (ms) for isochronous Set a value between 1 and 255 (ms) for the interrupt transfer in Full Speed Set a value between 10 and 255 (ms) for the interrupt transfer in Low Speed

3. Firmware Architecture

As shown in the figure below, the architecture of the USB firmware is designed to avoid the customer any hardware interfacing (Drivers layer should not be modified by the user). The user has to interface with the API (*usb_standard_request.c* file should not be modified) and User application layers to build his own application.

Figure 3-1. USB Firmware Architecture



Hereunder the description of the main files:

main.c, config.h, scheduler.c, conf_scheduler.h, usb_task.c, usb_task.h, usb_standard_request.c, usb_standard_request.h, usb_specific_request.c, usb_specific_request.h, usb_drv.c, usb_drv.h, app_user_task.c, user_api.c

4. Start up

The Start up contains two files which are: *main.c* and *scheduler.c.* These files manage the launch and the progress of the tasks*.*

4.1 main.c

This file is the main entry point for the application. It also performs the launch of the scheduler.

4.2 config.h

This file contains the system configuration definition, such as the clock definition. For example to define a 16Mhz clock frequency you have to declare:

```
#define FOSC            16000       // Oscillator frequency(KHz)
```

4.3 scheduler.c

This file should not be modified by the user. It manages the routine to call the tasks. Each task will be called following the order specified in *conf_scheduler.h*.

4.4 conf_scheduler.h

The tasks called by the scheduler have to be defined in this file. The initialization tasks will be called just once, the others will be called continuously. To add his own tasks, the user has to declare them in this file.

Example:

```
#define Scheduler_task_1_init   usb_task_init
#define Scheduler_task_1        usb_task
#define Scheduler_task_2_init   mouse_task_init
#define Scheduler_task_2        mouse_task
```

Using the declaration of the example above, the scheduler will start by calling the *usb_task_init* then *mouse_task_init* once, and after it will continuously call *usb_task* followed by *mouse_task.*

5. User application

The User application layer manages the end-user interface. The user should add the several application tasks. The USB task is provided by Atmel and the user should not modify it.

5.1 usb_task.c

This file performs all USB requests (Standard and Specific). It manages the USB enumeration process and all asynchronuous events (Suspend, Resume, Reset, Wake up...). This file should not be modified by the user.

5.2 app_user_task.c

The user can add more than one task, it depends of his application.These tasks will manage the higher level of the user application (the interface with the end user).

For example, for a mouse application, the user has to add the button and the sensor management as a *app_user_task.c*.

6. API

The API layer permits the interfacing between the user application layer and the drivers layer. It hides the drivers peculiarity to the user.

6.1 usb_standard_request.c

This file manages all the USB standard requests. These requests are the same for any USB class enumeration. The user should not modify this file.

6.2 usb_standard_request.h

This file defines the values of the standard requests parameters.

6.3 usb_specific_request.c

This file manages the specific requests. These requests depend of the application, the user has to add the related functions to manage them.

6.4 usb_specific_request.h

This file defines the values for the parameters of the specific requests. If the application has specific requests, the user has to add the values/parameters of these requests in this file.

6.5 usb_descriptor.c

In this file we filled out all parameters/values of the enumeration descriptor structures defined in *usb_descriptor.h.* This file is important for the user to make his own application. Any descriptor values added in *usb_descriptor.h* have to be filled out in this file.

6.6 usb_descriptor.h

This file contains all parameters/values and the structures of the descriptors. Before adding any descriptor to the *usb_descriptor.c,* the user has to add the related stucture type and value definition in this file.

6.7 conf_usb.h

This file defines the definition of the endpoints numbers used by the application

For example, if you want to use the endpoint 1 as mouse application endpoint, you should define it in *conf_usb.h* file as below:

```
#define EP_MOUSE_IN            1  //! Number of the mouse interrupt IN
                                      endpoint
```

6.8 user_api.c

This file will be added by the user. It will make the interfacing between the driver level and the user application.

For example, this file could manage the tansfer of data between the PC and the device.

7. Drivers

The driver layer permits to hide all the hardware complexity to the user.

7.1 usb_drv.c

This file presents the interface between the user and the USB hardware, it contains all the USB drivers routines. The user should not modify this file.

7.2 usb_drv.h

This file contains the USB low level driver definition.

8. FAQ

8.1 How to change the VID & the PID?

The VID (Vendor ID) and the PID (Product ID) allow the identification of the device by the host. Each manufacturer should have its own VID, which will be the same for all products (it is assigned by USB org). Each product should has its own PID (it is assigned by the manufacturer).

The value of the VID and the PID are defined in *usb_descriptor.h.* To change them you have to change the values below:

// USB Device descriptor

```
#define VENDOR_ID            0x03EB // Atmel vendor ID = 03EBh
#define PRODUCT_ID           0x201C
```

8.2 How can I change the string descriptors value?

The value of the string descriptors are defined in *usb_descriptor.h.* For example to change the product name value, you have to change the following values:

The length of the string value:

```
#define USB_PN_LENGTH         18
```

The String value:

```
#define USB_PRODUCT_NAME \
{ Usb_unicode('A') \
 ,Usb_unicode('V') \
 ,Usb_unicode('R') \
 ,Usb_unicode(' ') \
 ,Usb_unicode('U') \
 ,Usb_unicode('S') \
 ,Usb_unicode('B') \
 ,Usb_unicode(' ') \
 ,Usb_unicode('M') \
 ,Usb_unicode('O') \
 ,Usb_unicode('U') \
 ,Usb_unicode('S') \
 ,Usb_unicode('E') \
 ,Usb_unicode(' ') \
 ,Usb_unicode('D') \
 ,Usb_unicode('E') \
 ,Usb_unicode('M') \
 ,Usb_unicode('O') \
}
```

8.3 How can I configure my device in self-powered or bus-powered mode?

The parameter to configure the device in self-powered or bus-powered mode is defined in *usb_descriptor.h* file. Hereunder the definition of each mode

bus-power mode:

// USB Configuration descriptor

```
#define CONF_ATTRIBUTES    USB_CONFIG_BUSPOWERED
```

Self-power mode:

// USB Configuration descriptor

```
#define CONF_ATTRIBUTES    USB_CONFIG_SELFPOWERED
```

8.4 How can I add a new descriptor?

To add a new descriptor to your application, you have to follow the steps below:

1. Define the values of the descriptor parameters and its stucture type in *usb_descriptors.h* file.

For example the HID descriptor and its structure should be defined in *usb_descriptors.h* file as shown below:

/* ____HID descriptor____*/

```
#define HID                     0x21
#define REPORT                  0x22
#define SET_REPORT      0x02
#define HID_DESCRIPTOR          0x21
#define HID_BDC                 0x1001
#define HID_COUNTRY_CODE        0x00
#define HID_CLASS_DESC_NB       0x01
#define HID_DESCRIPTOR_TYPE     0x22
```

/*_____U S B   H I D   D E S C R I P T O R__________________________________*/

```
typedef struct {
  U8  bLength;              /* Size of this descriptor in bytes */
  U8  bDescriptorType;      /* HID descriptor type */
  U16 bscHID;               /* Binay Coded Decimal Spec. release */
  U8  bCountryCode;         /* Hardware target country */
  U8  bNumDescriptors;     /* Number of HID class descriptors to follow */
  U8  bRDescriptorType;     /* Report descriptor type */
  U16 wDescriptorLength;    /* Total length of Report descriptor */
} S_usb_hid_descriptor;
```

2. Add the new descriptor to the *s_usb_user_configuration_descriptor* structure in *usb_descriptors.h* file:

```
typedef struct
{
   S_usb_configuration_descriptor cfg_mouse;
   S_usb_interface_descriptor     ifc_mouse;
   S_usb_hid_descriptor           hid_mouse;
   S_usb_endpoint_descriptor      ep1_mouse;
} S_usb_user_configuration_descriptor;
```

3. In the File *usb_descriptors.c,* add the size of the new descriptor to the *wTotalLength* parameter of the configuration descriptor and add the descriptor value (see the example below):

```
code S_usb_user_configuration_descriptor usb_conf_desc = {
 { sizeof(S_usb_configuration_descriptor)
 , CONFIGURATION_DESCRIPTOR
 , Usb_write_word_enum_struc(sizeof(S_usb_configuration_descriptor)\
     +sizeof(S_usb_interface_descriptor)   \
     +sizeof(S_usb_hid_descriptor)         \
     +sizeof(S_usb_endpoint_descriptor)    \
                           )
 , NB_INTERFACE
 , CONF_NB
 , CONF_INDEX
 , CONF_ATTRIBUTES
 , MAX_POWER
 }
 ,
 { sizeof(S_usb_interface_descriptor)
 , INTERFACE_DESCRIPTOR
 , INTERFACE_NB_MOUSE
 , ALTERNATE_MOUSE
 , NB_ENDPOINT_MOUSE
 , INTERFACE_CLASS_MOUSE
 , INTERFACE_SUB_CLASS_MOUSE
 , INTERFACE_PROTOCOL_MOUSE
 , INTERFACE_INDEX_MOUSE
 }
 ,
 { sizeof(S_usb_hid_descriptor)
 , HID_DESCRIPTOR
 , HID_BDC
 , HID_COUNTRY_CODE
 , HID_CLASS_DESC_NB
 , HID_DESCRIPTOR_TYPE
 , Usb_write_word_enum_struc(sizeof(S_usb_hid_report_descriptor_mouse))
 }
 ,
 { sizeof(S_usb_endpoint_descriptor)
 , ENDPOINT_DESCRIPTOR
 , ENDPOINT_NB_1
 , EP_ATTRIBUTES_1
 , Usb_write_word_enum_struc(EP_SIZE_1)
 , EP_INTERVAL_1
 }
};
```

4. Do not forget to add all functions related to manage this new descriptor.

8.5 How can I add a new endpoint?

to configure a new endpoint, you have to follow the steps below:

1. As explained in the USB descriptors section, an endpoint belongs to an interface. The first thing to do to add a new endpoint, is to increment by one to the ***NB_ENDPOINT*** parameter value. This value is defined in *usb_descriptor.h* file and it belong to the interface descriptor prameters.

// USB Interface descriptor

```
#define INTERFACE_NB xx
#define ALTERNATE xx
#define NB_ENDPOINT xx//This parameter=the endpoints number of the
interface
#define INTERFACE_CLASS xx
#define INTERFACE_SUB_CLASS xx
#define INTERFACE_PROTOCOL xx
#define INTERFACE_INDEX xx
```

2. The next step is to define the endpoint descriptor values . These values have to be defined in *usb_descriptors.h*.

```
// USB Endpoint 1 descriptor FS
#define ENDPOINT_NB_1       (EP_MOUSE_IN | 0x80)
#define EP_ATTRIBUTES_1     0x03            // BULK = 0x02, INTERUPT = 0x03
#define EP_IN_LENGTH_1      8
#define EP_SIZE_1           EP_IN_LENGTH_1
#define EP_INTERVAL_1       0x02  // Interrupt polling interval from host
```

EP_MOUSE_IN is defined in *conf_USB.h* to specify the endpoint numeber used by the application.

3. Add the new endpoint descriptor to the configuration descriptor (proceed as explained in the previous FAQ point).
4. Add the hardware initialization call in *usb_specific_request.c*

```
void usb_user_endpoint_init(U8 conf_nb)
{
  usb_configure_endpoint(EP_MOUSE_IN,      \
                         TYPE_INTERRUPT,     \
                         DIRECTION_IN,  \
                         SIZE_8,       \
                         ONE_BANK,     \
                         NYET_ENABLED);
}
```

9. Coding Style

The coding style explained hereunder are important to understand the firmware:

- Defined contants use caps letters.

```
#define FOSC 8000
```

- Macros Functions use the first letter as cap

```
#define Is_usb_sof() ((UDINT &   MSK_SOFI)    ? TRUE: FALSE)
```

- The user application can execute its own specific instructions upon each usb events thanks to hooks defined as following in *usb_conf.h.*

```
#define Usb_sof_action()         sof_action();
Note: The hook function should perform only short time requirement
operations !
```

- Usb_unicode() macro function should be used everywhere (String descriptors...) an unicode char is exchanged on the USB protocol.

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