Implementing the USB Enumeration Process on the AT8xC5131/32/22 and AT8xC51SND1

USB Microcontrollers

Application Note

The Universal Serial Bus (USB) has seen enormous success in PC systems and is replacing the older parallel and serial ports. For a standard serial port, the communication is directly performed by the application running on the computer. In order to be Plug-and-Play and Hot-Plug, the USB bus introduces a process that uniquely identifies a device to the Host computer in order for it to learn the capabilities of the device and to load the appropriate driver. This identification process is called the Enumeration process and uses a standard set of commands described in the Chapter 9 of the USB specification, "USB Device Framework".

This application note describes a way to implement the enumeration process on AT8xC5131/32/22 and AT8xC51SND1 products. The C-source code is available from the Atmel Web site.

Rev. 4290A–USB–10/03

USB Specification

This is an extract from the USB specification version 2.0:

"**9.1.2 Bus Enumeration**

When a USB device is attached to or removed from the USB, the host uses a process known as bus enumeration to identify and manage the device state changes necessary. When a USB device is attached to a powered port, the following actions are taken:

1. The hub to which the USB device is now attached informs the host of the event via a reply on its status change pipe (refer to Section 11.12.3 for more information). At this point, the USB device is in the Powered state and the port to which it is attached is disabled.
2. The host determines the exact nature of the change by querying the hub.
3. Now that the host knows the port to which the new device has been attached, the host then waits for at least 100 ms to allow completion of an insertion process and for power at the device to become stable. The host then issues a port enable and reset command to that port. Refer to Section 7.1.7.5 for sequence of events and timings of connection through device reset.
4. The hub performs the required reset processing for that port (see Section 11.5.1.5). When the reset signal is released, the port has been enabled. The USB device is now in the Default state and can draw no more than 100 mA from V_{BUS}. All of its registers and states have been reset and it answers to the default address.
5. The host assigns a unique address to the USB device, moving the device to the Address state.
6. Before the USB device receives a unique address, its Default Control Pipe is still accessible via the default address. The host reads the device descriptor to determine what actual maximum data payload size this USB device's default pipe can use.
7. The host reads the configuration information from the device by reading each configuration zero to *n*-1, where *n* is the number of configurations. This process may take several milliseconds to complete.
8. Based on the configuration information and how the USB device will be used, the host assigns a configuration value to the device. The device is now in the Configured state and all of the endpoints in this configuration have taken on their described characteristics. The USB device may now draw the amount of V_{BUS} power described in its descriptor for the selected configuration. From the device's point of view, it is now ready for use.

When the USB device is removed, the hub again sends a notification to the host. Detaching a device disables the port to which it had been attached. Upon receiving the detach notification, the host will update its local topological information."

USB Enumeration Process

The Enumeration process is used by the Host when a device is attached to the USB bus. This process allows the Host to identify and to manage the device.

- Device identification:

The Host sends standard device requests on the default control endpoint in order to identify the device and then to load the appropriate driver. The device answers to each request with the corresponding descriptor tables. The descriptor tables contain all the information relating to the device: characteristics of the device and number and characteristics of each configuration, interface and endpoint.

The 4 standard descriptor types are:

- The Device descriptor
- The Configuration descriptor
- The Interface descriptor
- The Endpoint descriptor

Other descriptor types can be added corresponding to a specific USB class.

- Device management:
  - The host manages the device address, the status and the configurations using standard requests on the default control endpoint."

Implementation

Descriptor Tables

The descriptors tables contain all information about the device required by the Host to load the appropriate drivers. The descriptor table types are described in the usb_enumeration.h file.

Device Descriptor

The device descriptor table contains the unique identification of the device (Vendor ID, Product ID and Release Number) and general information about the device. The device descriptor is sent by the device when the Host sends a GET_DESCRIPTOR request with a DEVICE Descriptor type.

```
/*_____ U S B   D E V I C E   D E S C R I P T O R ___________________________*/

struct usb_st_device_descriptor
{
  Uchar  bLength;               /* Size of this descriptor in bytes */
  Uchar  bDescriptorType;       /* DEVICE descriptor type */
  Uint16 bscUSB;                /* Binay Coded Decimal Spec. release */
  Uchar  bDeviceClass;          /* Class code assigned by the USB */
  Uchar  bDeviceSubClass;       /* Sub-class code assigned by the USB */
  Uchar  bDeviceProtocol;       /* Protocol code assigned by the USB */
  Uchar  bMaxPacketSize0;       /* Max packet size for EP0 */
  Uint16 idVendor;              /* Vendor ID */
  Uint16 idProduct;             /* Product ID assigned by the manufacturer */
  Uint16 bcdDevice;             /* Device release number */
  Uchar  iManufacturer;         /* Index of manu. string descriptor */
  Uchar  iProduct;              /* Index of prod. string descriptor */
  Uchar  iSerialNumber;         /* Index of S.N.  string descriptor */
  Uchar  bNumConfigurations;    /* Number of possible configurations */
};
```

Configuration Descriptor

The following descriptors describe the interfaces and the endpoints used by the configurations.

```
/*_____ U S B   C O N F I G U R A T I O N   D E S C R I P T O R _____________*/

struct usb_st_configuration_descriptor
{
  Uchar  bLength;               /* size of this descriptor in bytes */
  Uchar  bDescriptorType;       /* CONFIGURATION descriptor type */
  Uint16 wTotalLength;          /* total length of data returned */
  Uchar  bNumInterfaces;        /* number of interfaces for this conf. */
  Uchar  bConfigurationValue;   /* value for SetConfiguration resquest */
  Uchar  iConfiguration;        /* index of string descriptor */
  Uchar  bmAttibutes;           /* Configuration characteristics */
  Uchar  MaxPower;              /* maximum power consumption */
};
```

```
/*_____ U S B   I N T E R F A C E   D E S C R I P T O R _____________________*/

struct usb_st_interface_descriptor
{
  Uchar bLength;                /* size of this descriptor in bytes */
  Uchar bDescriptorType;        /* INTERFACE descriptor type */
  Uchar bInterfaceNumber;       /* Number of interface */
  Uchar bAlternateSetting;      /* value to select alternate setting */
  Uchar bNumEndpoints;          /* Number of EP except EP 0 */
  Uchar bInterfaceClass;        /* Class code assigned by the USB */
  Uchar bInterfaceSubClass;     /* Sub-class code assigned by the USB */
  Uchar bInterfaceProtocol;     /* Protocol code assigned by the USB */
  Uchar iInterface;             /* Index of string descriptor */
};


/*_____ U S B   E N D P O I N T   D E S C R I P T O R _______________________*/

struct usb_st_endpoint_descriptor
{
  Uchar  bLength;               /* Size of this descriptor in bytes */
  Uchar  bDescriptorType;       /* ENDPOINT descriptor type */
  Uchar  bEndpointAddress;      /* Address of the endpoint */
  Uchar  bmAttributes;          /* Endpoint's attributes */
  Uint16 wMaxPacketSize;        /* Maximum packet size for this EP */
  Uchar  bInterval;             /* Interval for polling EP in ms */
};
```

In additions, strings and class specific descriptor tables are also defined:

```
/*_____ U S B   M A N U F A C T U R E R   D E S C R I P T O R _______________*/

struct usb_st_manufacturer
{
  Uchar  bLength;               /* size of this descriptor in bytes */
  Uchar  bDescriptorType;       /* STRING descriptor type */
  Uint16 wstring[USB_MN_LENGTH];/* unicode characters */
};


/*_____ U S B   P R O D U C T   D E S C R I P T O R _________________________*/

struct usb_st_product
{
  Uchar  bLength;               /* size of this descriptor in bytes */
  Uchar  bDescriptorType;       /* STRING descriptor type */
  Uint16 wstring[USB_PN_LENGTH];/* unicode characters */
};
```

```
/*_____ U S B   S E R I A L   N U M B E R   D E S C R I P T O R ____________*/

struct usb_st_serial_number
{
  Uchar  bLength;               /* size of this descriptor in bytes */
  Uchar  bDescriptorType;       /* STRING descriptor type */
  Uint16 wstring[USB_SN_LENGTH];/* unicode characters */
};


/*_____ U S B   L A N G U A G E   D E S C R I P T O R ______________________*/

struct usb_st_language_descriptor
{
  Uchar  bLength;               /* size of this descriptor in bytes */
  Uchar  bDescriptorType;       /* STRING descriptor type */
  Uint16 wlangid;               /* language id */
};

/*_____ U S B   H I D   D E S C R I P T O R _________________________________*/

struct usb_st_hid_descriptor
{
  Uchar  bLength;               /* Size of this descriptor in bytes */
  Uchar  bDescriptorType;       /* HID descriptor type */
  Uint16 bscHID;                /* Binay Coded Decimal Spec. release */
  Uchar  bCountryCode;          /* Hardware target country */
  Uchar  bNumDescriptors;       /* Num. of HID class descriptors to follow */
  Uchar  bRDescriptorType;      /* Report descriptor type */
  Uint16 wDescriptorLength;     /* Total length of Report descriptor */
};
```

The descriptor tables must be sent in the following order:

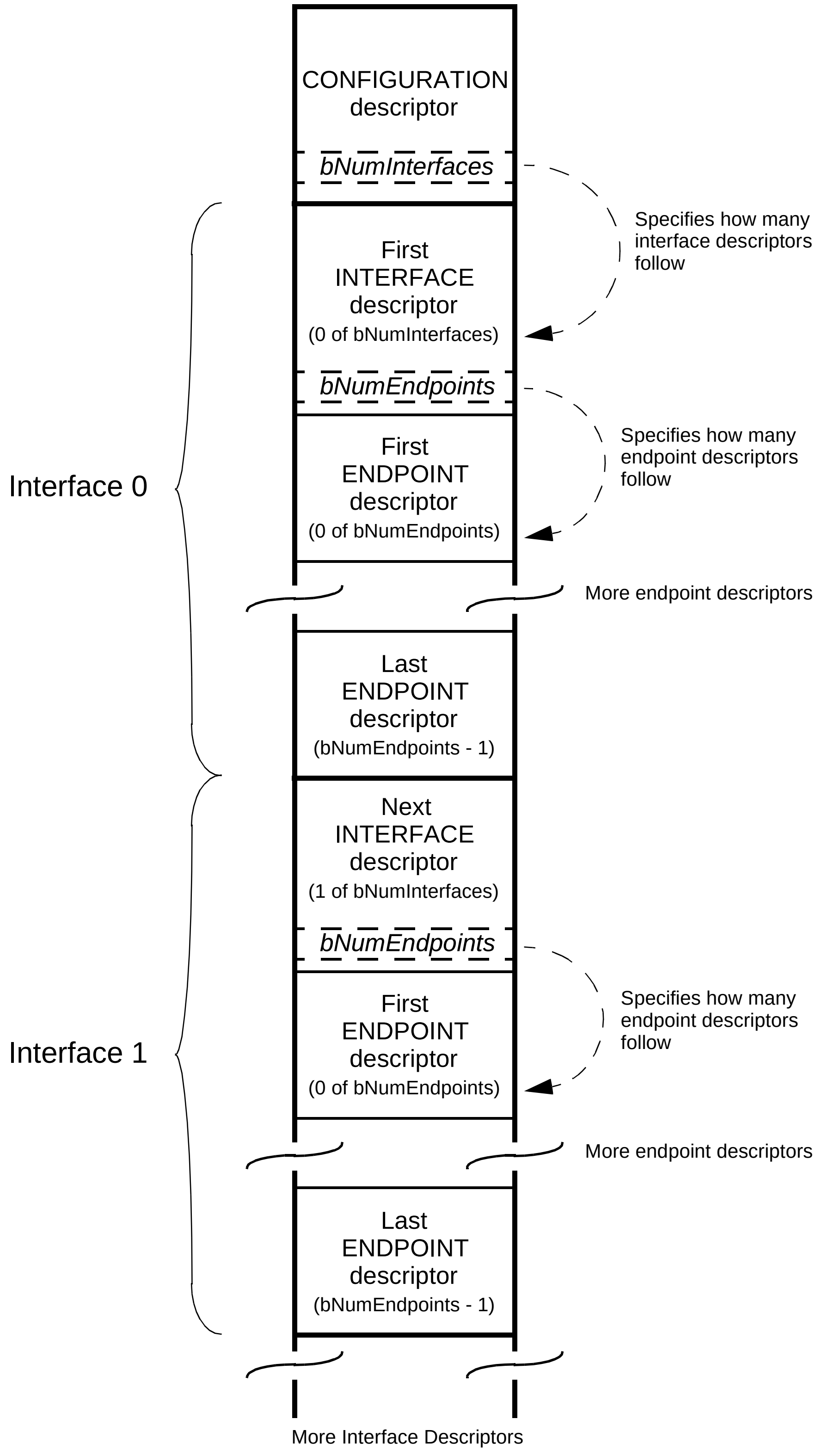
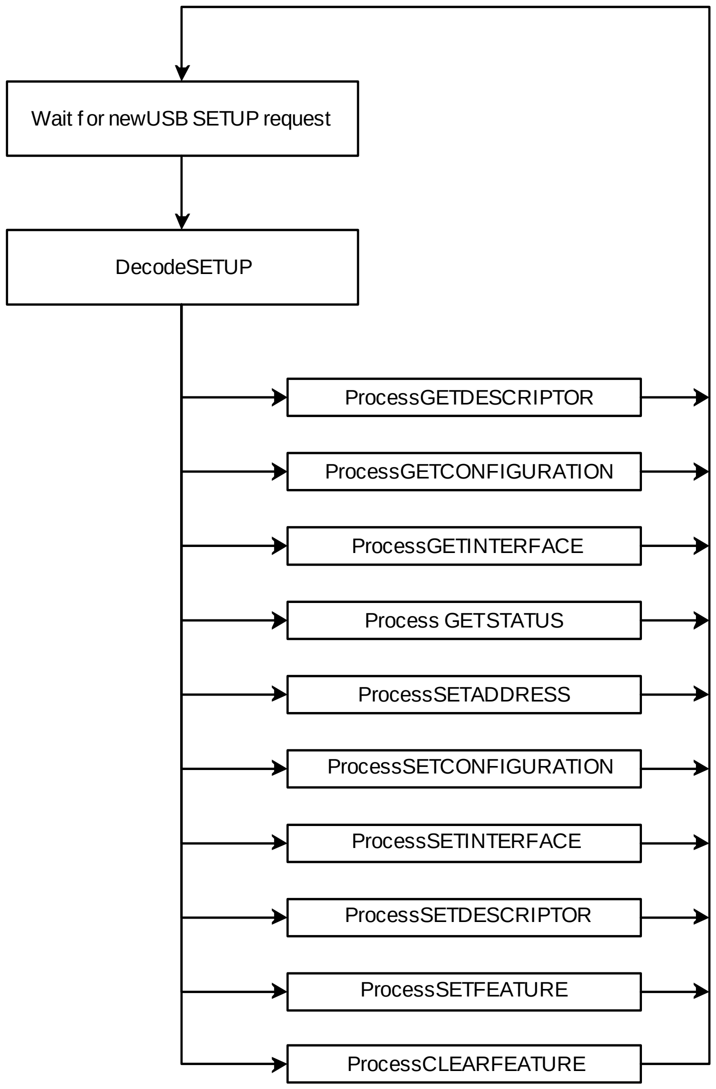

CONFIGURATION descriptor
bNumInterfaces
Specifies how many interface descriptors follow
First INTERFACE descriptor (0 of bNumInterfaces)
bNumEndpoints
Specifies how many endpoint descriptors follow
First ENDPOINT descriptor (0 of bNumEndpoints)
Interface 0
More endpoint descriptors
Last ENDPOINT descriptor (bNumEndpoints - 1)
Next INTERFACE descriptor (1 of bNumInterfaces)
bNumEndpoints
Specifies how many endpoint descriptors follow
First ENDPOINT descriptor (0 of bNumEndpoints)
Interface 1
More endpoint descriptors
Last ENDPOINT descriptor (bNumEndpoints - 1)
More Interface Descriptors

Figure 1. Configuration Descriptors

Example of Software Implementation

The following diagram describes the way to implement the enumeration process. Each new SETUP packet has to be decoded in order to launch the right process.

Figure 2. Enumeration Process



Get Device Descriptor Process Example

Definition of the Device Descriptor to Send

```
code struct USB_device_descriptor_st default_device_descriptor =
{ 0x12, DEVICE, 0x1001, 0x0FF, 0xFF, 0x00, 32, VENDOR_ID,
PRODUCT_ID, RELEASE_NUMBER, 0, 0, 0, 1 };
```

VENDOR_ID, PRODUCT_ID, RELEASE_NUMBER can be easily modified by the developer. These unique IDs are defined by the final application manufacturer. Refer to the http:\\www.usb.org web site in order to acquire a referenced vendor ID.

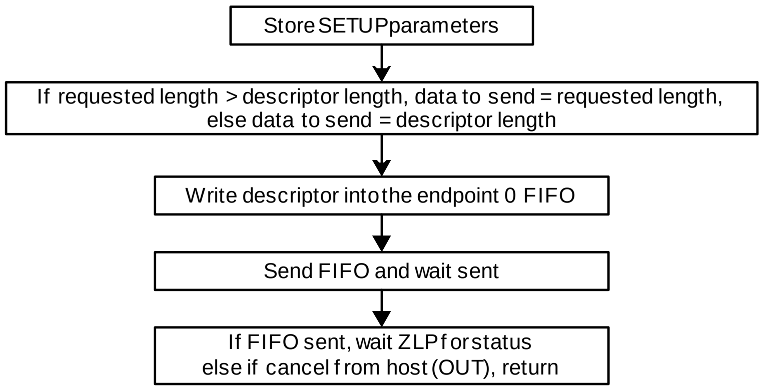
Get Device Descriptor Process

The algorithm below describes the Get Device Descriptor process. The first step is to store the Setup parameters. The main parameter in this example is the length of the data requested by the Host. It indicates the maximum number of bytes the device has to send. If this number is higher than the descriptor length to send, the device should send the complete descriptor. If this number is lower than the descriptor length to send, the device should send the exact number of bytes requested.

The device stores in the default control endpoint FIFO the bytes to send to the host and set the TXRDY bit. Once the data have been sent, the device clear the TXCMPL bit and wait to receive the Status from the Host, a OUT Zero Length Packet (ZLP).

If the ZLP occurs before the data have been sent, this means that the Host has cancelled the control transaction. The following packet will be a new SETUP request.

Figure 1. Device Descriptor Process


Store SETUP parameters
If requested length > descriptor length, data to send = requested length, else data to send = descriptor length
Write descriptor into the endpoint 0 FIFO
Send FIFO and wait sent
If FIFO sent, wait ZLP for status else if cancel from host (OUT), return

This is the C-code corresponding to the algorithm.

```
void usb_get_descriptor(void)
{
Uint16      length;
Uchar       i;
Uchar       descriptor_type;
Uchar code* pbuffer;

ACC             = Usb_read_byte();
descriptor_type = Usb_read_byte();
ACC             = Usb_read_byte();
ACC             = Usb_read_byte();
length          = Usb_read_byte();
length         |= Usb_read_byte() << 8;
Usb_clear_rx_setup();
Usb_set_DIR();
if (length >= DEVICE_DESCRIPTOR_LENGTH) length = DEVICE_DESCRIPTOR_LENGTH;
pbuffer=(descriptor_type==DEVICE)?
  &my_device_descriptor->bLength:&my_configuration_descriptor.cfg.bLength;
for (i=0;i<length;i++,pbuffer++) Usb_write_byte(*pbuffer);
Usb_set_tx_ready();
while (!Usb_tx_complete() && !Usb_rx_complete());
if (Usb_rx_complete())
  {
  Usb_clear_DIR() ;
  Usb_clear_rx() ;
  return ;
  }
else Usb_clear_tx_complete();
usb_wait_receive();
Usb_clear_DIR() ;
}
```

Clear Feature Process Example

In this example, only the endpoint 1 feature is supported. Every other Clear Feature is Stalled.

When the Clear Feature addressed to the endpoint 1 is received, the firmware clears the Stall request on this endpoint and resets this endpoint in order to reset the data toggle. The endpoint status is also reset. This status is returned to the host during the Get Status transaction.

The following C-code is an example of this implentation.

```
void usb_clear_feature (void)
{
  if (bmRequestType == ZERO_TYPE)
  {
    Usb_clear_RXSETUP();
    Usb_set_STALLRQ();
    while (!(Usb_STALL_sent()));
    Usb_clear_STALLRQ();
  }
  if (bmRequestType == INTERFACE_TYPE)
  {
    Usb_clear_RXSETUP();
    Usb_set_STALLRQ();
    while (!(Usb_STALL_sent()));
    Usb_clear_STALLRQ();
  }
  if (bmRequestType == ENDPOINT_TYPE)
  {
    if (Usb_read_byte() == 0x00)
    {
      ACC = Usb_read_byte();                /* dummy read */
      switch (Usb_read_byte())              /* check wIndex */
      {
        case ENDPOINT_1:
        {
          Usb_select_ep(EP_IN);
          if(Usb_STALL_requested()) { Usb_clear_STALLRQ(); }
          if(Usb_STALL_sent()) { Usb_clear_STALLED(); }
          UEPRST = 0x02;
          UEPRST = 0x00;
          Usb_select_ep(EP_CONTROL);
          endpoint_status[EP_IN] = 0x00;
          Usb_clear_RXSETUP();
          Usb_set_TXRDY();
          while (!(Usb_tx_complete()));
          Usb_clear_TXCMPL();
          break;
        }
        case ENDPOINT_0:
        {
          Usb_clear_RXSETUP();
          Usb_set_TXRDY();
          while (!(Usb_tx_complete()));
          Usb_clear_TXCMPL();
          break;
        }
        default:
        {
          Usb_clear_RXSETUP();
          Usb_set_STALLRQ();
          while (!(Usb_STALL_sent()));
          Usb_clear_STALLRQ();
          break;
        }
      }
    }
  }
}
```

How to use the Atmel Enumeration Functions

Polling Mode

The usb_enumeration_process function manages a control transaction that occurs on the default control endpoint. This function has to be called each time a new setup has been detected.

```
if (Usb_endpoint_interrupt())
    {
      Usb_select_ep(EP_CONTROL);
      if (Usb_setup_received()) { usb_enumeration_process(); }
    }
```

Interrupt Mode

In this mode, the USB interrupt has to be enabled. The firmware should enable the Endpoint 0 interrupt generation. In the interrupt management, the firmware checks if a new Setup occurs on the default control endpoint. Before calling the usb_enumeration_process function, the firmware disables the endpoint 0 interrupt generation in order for the usb_enumeration_process function to manage the Control transaction until the end. Once the control transaction is complete, the firmware enables the endpoint 0 interrupt generation.

```
void USB_interrupt_process(void)  interrupt IRQ_USB
{
if (Usb_endpoint_interrupt())
    {
      Usb_select_ep(EP_CONTROL);
      if (Usb_setup_received())
        { Usb_disable_ep_int(EP_CONTROL);
        usb_enumeration_process();
        Usb_enable_ep_int(EP_CONTROL);}
        }
    }
}
```

Testing the Enumeration Implementation

The http:\\www.usb.org web site distributes a free software to test the chapter 9 implementation. This software tests the standard transactions described in the chapter 9 of the USB specification.

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

Disclaimer: Atmel Corporation makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in Atmel's Terms and Conditions located on the Company's web site. The Company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of Atmel are granted by the Company in connection with the sale of Atmel products, expressly or by implication. Atmel's products are not authorized for use as critical components in life support devices or systems.

© Atmel Corporation 2003. **All rights reserved.** Atmel® and combinations thereof are the registered trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be the trademarks of others.

Printed on recycled paper.