AVR1010: Minimizing the power consumption of Atmel AVR XMEGA devices

Features

- **Sleep modes**
- **Clock prescaling and source switching**
- **Power Reduction Registers**
- **RTC clock source**
- **State of digital I/O**
- **Watchdog**
- **Brown Out Detector**
- **JTAG interface**

1 Introduction

The Atmel® AVR® XMEGA® devices are capable of achieving extremely low power consumption, which is required by both portable electronics and other battery-powered applications.

To reach the lowest possible power figures there are a few points to pay attention to – it is not only the sleep mode that defines the power consumption, but also the state of the I/O pins, the number of enabled peripheral modules and so on.

This application note describes what must be done to achieve the lowest possible power consumption for XMEGA devices. Example code is also supplied, which compiles with both GCC and IAR Embedded Workbench®.

Figure 1-1. The XMEGA devices can achieve powers that are barely measurable.

8-bit AVR® Microcontrollers

Application Note

Preliminary

Rev. 8267B-AVR-12/10

2 Reducing the power consumption to a minimum

Though many factors affect the power consumption, some will naturally affect more than others. Listed below are the most important factors, with recommendations and considerations.

2.1 Operating voltage

The power consumption is proportional to the square of the device's supply voltage, which should therefore be kept as low as possible.

A reduction in supply voltage can lower the limit for the maximum system clock frequency, thus increasing the time required in ACTIVE mode to execute a given amount of code.

Minimize power consumption by using as low supply voltage as possible.

2.2 Active mode operation

In ACTIVE mode, i.e. when sleep modes are not used, the power consumption is proportional to the system clock frequency. This means that if sleep modes are not used, the device should be run at the lowest possible system clock frequency to minimize the power consumption.

Minimize power consumption by keeping the clock frequency as low as possible if sleep modes are not used.

2.3 Sleep modes

In most applications it is desirable to minimize the power consumption, but not to reduce the system clock frequency – mainly to ensure fast processing and quick response of the system/product. In such applications one can use the "sleep modes" of the Atmel® AVR® XMEGA® to put the device in a low power state when there is nothing to process. The main principle is then to run as fast as possible, and sleep as much as possible. Running as fast as possible reduces the effect of static power consumption (i.e. independent of clock frequency), e.g. due to non-volatile memory being enabled in ACTIVE mode.

The power consumption and operation of peripherals in sleep depends on which sleep mode is used. Table 2-1 shows the characteristics of the different sleep modes available for XMEGA devices. An application may switch between sleep modes during operation, depending on which mode is the most suitable at the time.

Table 2-1: Available sleep modes for XMEGA.

	Active clock domain			Oscillators		Wake-up sources			
Sleep modes	**CPU clock**	**Peripheral clock**	**RTC clock**	**System clock source**	**RTC clock source**	**Asynchronous Port Interrupts**	**TWI Address match interrupts**	**Real Time Clock Interrupts**	**All interrupts**
Idle		X	X	X	X	X	X	X	X
Power-down						X	X		
Power-save			X		X	X	X	X	
Standby				X		X	X		
Extended Standby			X	X	X	X	X	X	

The three most commonly used modes are IDLE, POWER-SAVE and POWER-DOWN:

- In IDLE, most peripherals are still operating – only the Atmel® AVR® CPU core and non-volatile memories (Flash and EEPROM) are stopped. The DMA controller and Event system are still active in this mode, allowing for e.g. AD conversions and transfers via the USART to continue, even though the CPU is not operating. The device can be woken up by all interrupts.
- In POWER-SAVE, the low frequency Real Time Clock (RTC) timer is still running while the CPU and most other peripherals are stopped. The RTC is commonly used to wake the device up at timed intervals. Because the system clock source is stopped in this sleep mode, wake-up takes a bit longer than for IDLE since the system clock must stabilize before operation.
- POWER-DOWN is the deepest sleep mode. In this mode most of the device peripherals are stopped. The device is unable to wake itself up from this mode since both the peripheral clock and RTC are disabled. This mode therefore relies on external input, e.g. asynchronous pin interrupts or TWI, to wake the device up. There are exceptions to this rule: XMEGAs with the battery backup module and 32-bit RTC. The RTC in these devices will run regardless of the sleep-mode.

The XMEGA family also supports two additional sleep-modes which are useful when a short wake-up time is needed:

- STANDBY, which is POWER-DOWN with system clock source still running.
- EXTENDED STANDBY, which is POWER-SAVE with system clock source still running.

These two sleep-modes do not give as low power consumption as POWER-DOWN and POWER-SAVE respectively, but are useful if fast response is vital to the application.

Note that asynchronous port interrupts and TWI address matches can wake the device up from all sleep modes. Refer to the device manual for further information about these sleep modes and operation of them.

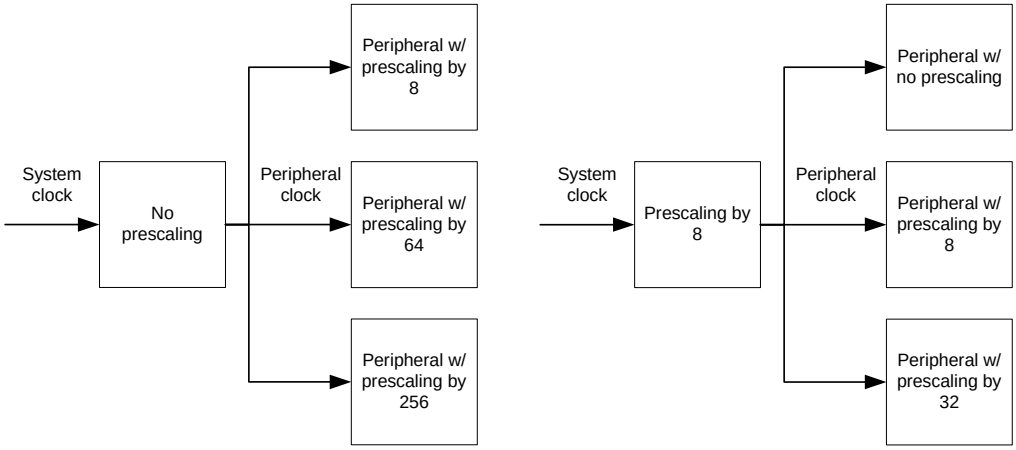
Minimize power consumption by using the deepest allowable sleep modes at any time, and running as fast as possible to minimize time spent in ACTIVE mode.

2.4 Clock Prescaling

Although it is recommended to run the CPU as fast as possible to minimize the time spent in ACTIVE mode, there are situations where it is better to reduce the clock rate. These situations commonly involve waiting in ACTIVE or IDLE mode for something that takes a fixed amount of time, e.g. serial communication. In these cases, one should avoid generating higher CPU and peripheral clock frequencies than are needed for the active peripherals. This may be achieved by using clock prescaling, which can be changed without causing glitches in the clock signal.

If prescaling is done internally in several peripherals, power can be conserved by prescaling with the largest common factor as early as possible in the clock distribution chain. This principle is illustrated in Figure 2-1.

Figure 2-1: Peripherals without and with common prescaling by largest factor.



Note that since the prescaling also affects the CPU clock, it might not always be desirable to perform this common prescaling in ACTIVE mode because computations will take longer.

Minimize power consumption by actively using prescaling, especially when waiting in ACTIVE or IDLE mode.

2.5 Clock Source Switching

One should avoid generating higher system clock rates than are actually needed. In the ideal case, prescaling is unnecessary. This can be achieved by switching between clock sources.

As an example, it is preferable to generate a 16MHz system clock by use of the PLL with the 2MHz RC oscillator as reference, rather than the 32MHz RC oscillator with prescaling to 16MHz. External clock sources may also be a good choice, especially if they are already available in the system and therefore come with no extra “cost”.

The wake-up delay for the device depends on which clock source is used for the system clock. One way to reduce this delay is to switch between clock sources so that the device goes to sleep and wakes up with a fast-responding clock source.

Minimize power consumption by switching clock sources rather than relying on prescaling alone for reducing clock rates.

2.6 Wake-Up Delays

When the device wakes up from sleep modes deeper than IDLE (with the exception of the two STANDBY-modes), the system clock source must stabilize before the CPU starts to operate. This introduces a short delay which depends on the selected clock source. If an internal RC oscillator or external clock is used, the start-up delay is 6 cycles. This is in addition to the RC oscillator start-up time. If the XTAL oscillator is used, the start-up delay is configurable. If frequency stability is wanted, it is recommended with start-up delays of 1,000 cycles for ceramic resonators and 16,000cycles for quartz crystals respectively. This is in addition to the oscillator start-up time, which will depend on the resonator and load capacitances.

In addition, there is a 13 cycle minimum delay before an Interrupt Service Routine (ISR) starts executing after wake-up. This is due to, e.g. the program counter being pushed on stack and the jump to the ISR.

During the start-up delay the power consumption is close to the power consumption in IDLE, and thus represents “inefficient” power. If possible, it is therefore recommended to wake up as seldom as possible and rather “do more” every time the device wakes up.

To minimize the wake-up delay and conserve power, use an RC oscillator or external clock source, and wake up as seldom as possible.

2.7 Power Reduction Registers

Most peripherals and internal modules can be individually stopped to avoid that these draw power in ACTIVE mode and in IDLE sleep. This is done by setting their respective bits in the Power Reduction Registers (PRR), which causes them to be disconnected from the peripheral clock domain. It is required to disable modules and peripherals via their respective control registers before setting their PRR bit, for the Power Reduction to be effective. Some modules must be reinitialized after clearing their PRR bit. Please refer to the sections about the individual PRR-bits in the datasheet manual for more information.

In POWER-SAVE and POWER-DOWN the modules are stopped regardless of the PRRs, since the peripheral clock domain is disabled.

To minimize power consumption, use the PRRs to disable peripherals and modules that are not used.

2.8 RTC Clock Source

One of the reasons for using IDLE, POWER-SAVE and EXTENDED STANDBY is that the RTC and its clock are active in these sleep modes. The RTC is commonly used to wake the device up at timed intervals.

For most Atmel® AVR® XMEGA-families, three different oscillators can be used to clock the RTC: An external 32kHz crystal, the internal 32kHz RC oscillator and the internal 32kHz Ultra Low Power (ULP) oscillator. In all cases, a prescaled 1kHz clock signal is available and should be used for reduced power consumption. For the external 32kHz crystal oscillator, a special low power mode is also available (X32KLPM).

It is recommended to use an external 32kHz crystal with X32KLPM enabled. This gives lower power consumption than the ULP, yet greater accuracy than the internal RC oscillator. This oscillator may also be used as the system clock source, if such a low frequency is acceptable.

Table 2-2. Examples of accuracy and current consumption for RTC w/ clock sources.

Oscillator	Accuracy[3]	Current consumption[1,3]
ULP	+/- 30%	1µA
RC32k	+/- 1%	30µA
TOSC (32 kHz XTAL)[2]	+/- 0,001% (10 ppm)	0.6µA

Notes:
1. At 3V operating voltage.
2. Depends on e.g. quality of crystal.
3. Please refer to the datasheet for exact values and conditions, the values stated here are meant as guideline only.

For Atmel® AVR® XMEGA-families with the battery backup module and 32-bit RTC, only the 32kHz crystal oscillator may be used as clock source. In these devices, the RTC is left running regardless of sleep.

Minimize power consumption by clocking the RTC at 1kHz with an external crystal in low power mode.

2.9 State of Digital I/O Pins

All digital I/O pins are by default floating to avoid hardware conflicts. However, since the pins have digital input buffers it is important to ensure that the level on an I/O pin is well-defined to avoid sporadic internal switching and leakage. The leakage caused by floating I/O is relatively small and is mainly observable in sleep, but can be minimized by ensuring that the state of the pins is either high or low.

If a pin is connected to an analog source, the digital input buffer on that pin should be disabled even if it is not configured as an input. This is done by use of the PINnCTRL registers for the individual ports. Please refer to the device manual for information on how to do this configuration.

To minimize power consumption, enable pull-up or -down on all unused pins, and disable the digital input buffer on pins that are connected to analog sources.

2.10 Virtual Port Registers

To minimize the time spent in ACTIVE mode, virtual port registers can be used. This allows for single-cycle access with I/O memory specific instructions like IN, OUT and bit manipulation for the registers DIR, IN, OUT and INTFLAGS for up to four I/O ports.

Use the Virtual Port registers for I/O port access to minimize power consumption.

2.11 General Purpose I/O Registers

Another way to minimize the time spent in ACTIVE mode is to use the GPIO registers for storage of variables. This is also due to the possibility of single-cycle access with I/O memory specific instructions.

Note that the GPIO registers are defined as volatile, so temporary variables should in some cases be used when manipulating variables stored in these registers: Otherwise, the performance gain will be lost.

Use the General Purpose I/O registers for variable storage to minimize power consumption.

2.12 Watchdog

The Watchdog is basically a timer with a separate clock source. It will, if enabled, contribute to the power consumption in sleep. The watchdog can only be clocked by the internal 32kHz Ultra Low Power (ULP) oscillator, prescaled to 1kHz.

To minimize the power consumption, disable the Watchdog.

2.13 Brown Out Detector

The purpose of the Brown Out Detector (BOD) is to ensure that the device is not operating at a too low voltage. It is highly recommended to use the internal BOD to ensure that the device always operates within specification.

However, during sleep the device is “not operating”, or rather, it is not executing code. For this reason, the BOD can be configured separately for ACTIVE/IDLE and sleep modes. This allows for the BOD to be enabled only in ACTIVE and IDLE mode. All configuration of the BOD is done with the device fuses.

To further reduce power consumption, the BOD may be run in sampled mode. The sample rate is approximately 1kHz, as it is clocked from the prescaled ULP oscillator. The BOD cannot detect voltage dips between samples in this mode, so it should only be used in applications with slowly varying operating voltages, such as battery-powered ones.

Disable the BOD - or better, disable it while in sleep - to reduce power consumption. Use sampled mode if only slow changes in operating voltage are likely.

2.14 JTAG interface and On-Chip Debugging

The JTAG interface is used for programming and debugging, but has no function during operation of an end-product. It is clocked and active during sleep if the On-chip Debugging (OCD) feature is enabled. The OCD and JTAG interface should therefore be disabled if it is not needed.

The OCD can be disabled in fuses, while the JTAG interface can be disabled both in fuses and in software. Disabling the JTAG in software ensures that the device can be reprogrammed because the JTAG interface is re-enabled upon RESET.

Alternatively, the PDI interface can be used for programming and debugging. In this case the JTAG interface may not be required at all, and may be disabled by fuses. The PDI interface also works in all sleep modes.

To minimize power consumption, disable the OCD and the JTAG interface.

2.15 Flash and EEPROM Power Reduction Modes

With the Atmel® AVR® XMEGA® NVM (Non-Volatile Memory) controller, it is possible to enable power reduction modes for the EEPROM and Flash. In these modes, the EEPROM and the currently unused section of Flash (i.e., application or boot section)

are powered down in ACTIVE mode, just as they are in any sleep mode. These power reduction modes will not affect power consumption in sleep.

If the CPU attempts to access a non-volatile memory with power reduction mode on, the CPU is halted for a time interval corresponding to wake-up from IDLE sleep while the memory is re-activated.

NB: There is an errata regarding Flash power reduction mode and sleep. For the affected devices, the workaround is to disable the Flash power reduction mode before entering sleep, then enabling it again on wake-up. Power consumption in sleep is not affected by this.

Enable power reduction mode for EEPROM and Flash to reduce power consumption in ACTIVE mode.

2.16 Writing to EEPROM

If more than one byte is to be written to EEPROM, one should make use of the EEPROM page buffer rather than doing byte-wise writes. This is because it takes just as long to write one byte as it takes to write an entire page to EEPROM. If, e.g., two bytes are to be written, byte-wise writing will take twice as long as necessary. Since the current consumption also increases during EEPROM writing, this gives a “double penalty”.

To minimize power consumption, use page-wise writing to EEPROM rather than byte-wise.

3 Code Examples

Six code examples are supplied with this application note. The main code files for these are:

```
xmega_power_consumption.c
xmega_sleep_example.c
xmega_rtc32_power_consumption.c
xmega_rtc32_sleep_example.c
xplain_power_consumption.c
xplain_sleep_example.c
```

These are respectively meant for three different setups:

- Generic XMEGA, w/ I/O pins left floating
- XMEGA w/ battery backup system, 32-bit RTC and only 32kHz crystal connected (e.g. A3B-family)
- Xplain evaluation board (ATxmega128A1)

The differences between these setups are the RTC driver and clock source, plus some tweaks which are specific for the Xplain evaluation board.

For the generic setup, the ULP is used as clock source for the RTC. A 32kHz crystal is used in the other setups. Note that the latter is mandatory for operation of the 32-bit RTC.

3.1 Power Consumption

All the `power_consumption.c` examples simply step through different sleep modes at timed intervals. This is meant to allow for simple verification of the power

consumption for the different Atmel® AVR® XMEGA® devices. The code steps through the following modes, staying for 8 seconds in each:

- ACTIVE
- IDLE
- POWER-SAVE
- POWER-DOWN

Note that the device will stay in POWER-DOWN, since an external interrupt is required to wake the device up from this sleep mode.

3.2 Sleep Example

All the `sleep_example.c` examples are meant as “code skeletons”. By default, the device is woken up at 5 second intervals, held in ACTIVE for half a second, then put to sleep in POWER-SAVE or POWER-DOWN mode. The user may build on this example for his/her application.

4 Sleep manager

4.1 Purpose

For this application note, a sleep manager has been implemented. Centralized control of sleep is necessary in applications that consist of several firmware modules that require different sleep modes depending on their status or activity level. Otherwise, one would risk that the firmware modules disrupted each other’s operation.

4.2 Operation

The sleep manager resides in `sleepmgr.c`, `sleepmgr.h` and `config_sleepmgr.h`, and can be accessed via the following four functions:

```
void SLEEPMGR_Init( void );
void SLEEPMGR_Lock( SLEEPMGR_mode_t mode );
void SLEEPMGR_Unlock( SLEEPMGR_mode_t mode );
void SLEEPMGR_Sleep( void );
```

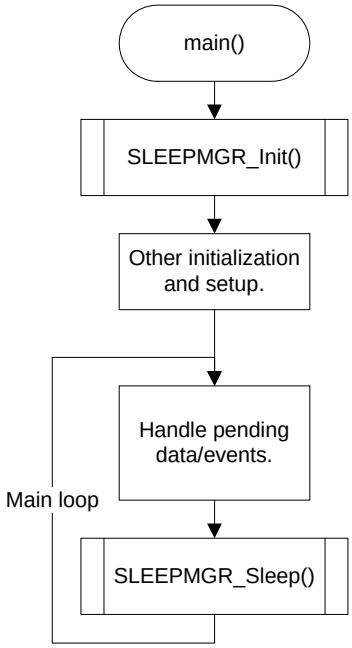
Prior to use, the sleep manager must be initialized to ensure correct operation.

It then keeps track of which sleep mode is allowable by the use of *locks*. This is simply a list of values, with a one-to-one correspondence with a list of sleep modes that are defined at compile-time. The firmware modules may individually lock and unlock the sleep modes (increase/decrease the locks’ values) to prevent the sleep manager from putting the device in “too deep” sleep, i.e., modes that would interfere with the modules’ operation.

When the sleep manager is requested to put the device to sleep, it searches through the locks for the first non-zero value. The device is then put to sleep in the mode which this lock corresponds to. Consequently, the order of the locks must be such that the deeper sleep modes come later. If no locks have been set after initialization, the sleep manager will default to the deepest sleep mode. Interrupts are disabled to prevent the locks from changing during this search. Any pending interrupts will cause the device to instantly wake up again.

Note that the main loop of the application should be responsible for requesting the sleep manager to put the device to sleep, as shown in Figure 4-1.

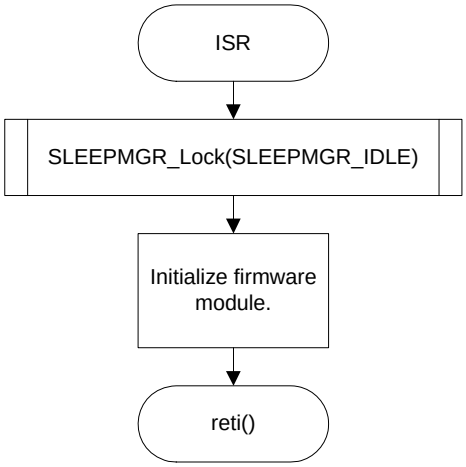
Figure 4-1: Suggested use of the sleep manager in main().



If an interrupt occurs while the sleep manager is about to put the device to sleep, it will instantly wake up again and execute the corresponding ISR. Any update of the locks must then occur in the ISR to ensure that the locks are set before the sleep manager is requested to put the device to sleep again. A general example of an ISR is shown in Figure 4-2.

Note that the sleep modes must also be unlocked when the firmware module is either done with its task or it locks a different sleep mode.

Figure 4-2: Example use of the sleep manager in an ISR.



The list of sleep modes is defined in `config_sleepmgr.h`. By default, the sleep manager supports the following modes (in the order listed):

- Idle – `SLEEPMGR_IDLE`
- Extended standby – `SLEEPMGR_ESTDBY`
- Power-save – `SLEEPMGR_SAVE`
- Standby – `SLEEPMGR_STDBY`
- Power-down – `SLEEPMGR_DOWN`

If your application does not need all sleep modes, the list should be modified to save both memory and execution time. Refer to the configuration file (`config_sleepmgr.h`) for information on how to modify the list.

5 Low Power Initialization

For convenience, a function that disables all the peripherals/modules and enables pull-up on all available I/O pins is included. This puts an Atmel® AVR® XMEGA® in the least power consuming configuration, with the exception of EEPROM and Flash power reduction modes, when nothing but VCC and GND is connected.

The function, `LOWPOWER_Init()`, supports all the different XMEGA families (A1, A3, A3B, A4) and resides in `lowpower.c`, `lowpower.h` and `lowpower_macros.h`.

Since this function is meant for XMEGAs with floating I/O pins, some custom configuration may be necessary for measurement in other setups. Refer to `lowpower_macros.h` for the actual example code.

6 Measurement Setup

The supplied code examples are meant for current measurements on either the Xplain board, or in a setup where only VCC, GND and alternatively an external crystal are connected to the device (I/O pins left floating). If the setup differs from these, it may be necessary to change the pin configurations. Special considerations are noted below.

6.1 Xplain Board

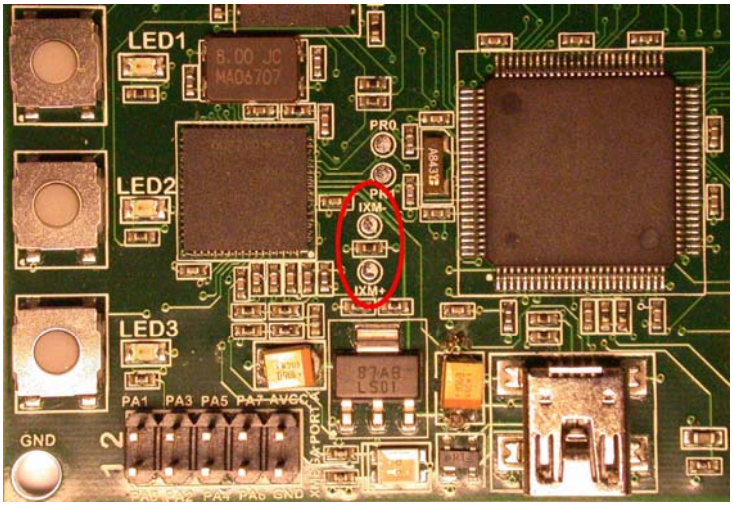
To measure on the Xplain evaluation board, the following modifications are necessary to reduce power consumption:

- Remove all jumpers from the board
- Enable pull-down on pin 3 of PORTQ (disables audio amplifier)
- Disable the input buffer on pin 1 of PORTB (potentiometer)

The two last points are done in the Xplain-specific example code. The Xplain board features a 32kHz crystal, which is used to clock the RTC to minimize power consumption.

Current measurements are done by connecting an Ampere-meter between the points named *IXM+* and *IXM-*. The shunt located between these measurement points must be removed. The location of the measurement points and shunt is shown in Figure 6-1.

Figure 6-1: Location of IXM and shunt on Xplain board.



6.2 XMEGA with 32-Bit RTC

Atmel® AVR® XMEGA® with the battery backup module and 32-bit RTC only allow for the external 32kHz crystal oscillator to be used as clock source for the RTC. This crystal must therefore be connected for wake-up at timed intervals to be possible.

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

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Milennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chou-ku, Tokyo 104-0033
JAPAN
Tel: (+81) 3523-3551
Fax: (+81) 3523-7581