

AVR1308: Using the XMEGA TWI

Features

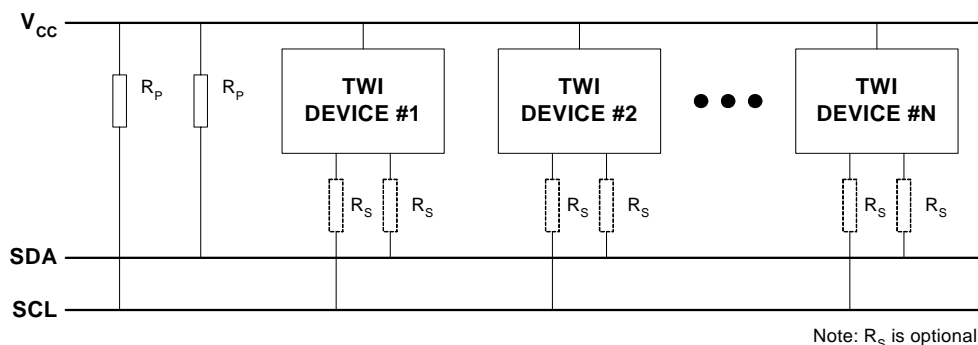
- Introduction to TWI and the XMEGA™ TWI module
- Setup and use of the XMEGA TWI module
- Implementation of module drivers
 - Master
 - Slave
- Code examples for master and slave

1 Introduction

This application note describes how to set up and use the TWI module in the XMEGA. C code drivers and examples are included for both master and slave applications.

The TWI (Two Wire Interface) is compatible with the Philips® Inter-IC, or I2C bus. TWI is used in communication between control devices like microcontrollers, and peripheral devices like LCD drivers, I/O expanders, memories and much more.

Figure 1-1. TWI bus topology.



Note: R_s is optional



8-bit **AVR**[®]
Microcontrollers

Application Note

Rev. 8054A-AVR-02/08



2 The TWI bus

The TWI bus consists of two active lines, SDA (Serial DATA) and SCL (Serial CLOCK), in addition to ground. The two active lines are bidirectional open collector lines with pull-up resistors.

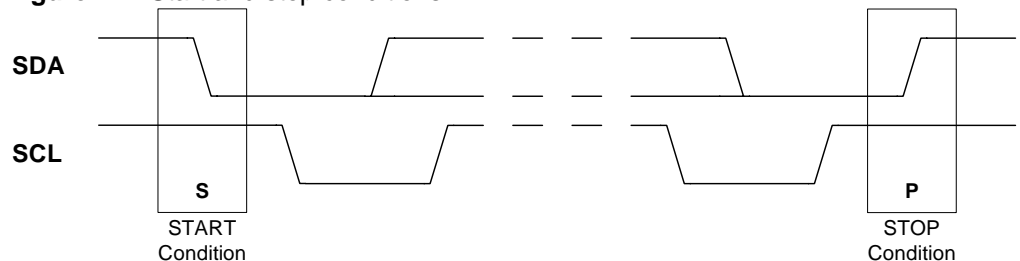
The devices connected to the bus have unique addresses, and can be receivers or transmitters depending on operation. A DTMF tone generator can be example of a receiver only, while a memory device obviously can be both receiver and transmitter.

2.1 Start and stop conditions

On an idle bus, both SDA and SCL are high. A device can initiate a transaction by first pulling SDA low, then SCL. This is called a START condition (S). The transaction is completed by first releasing SCL, then SDA. This is called a STOP condition (P). As the SDA beyond that is only allowed to toggle while SCL is low, the START and STOP conditions are unique and secure ways to indicate the start or end of the transaction. The device that initiates a transaction by the START condition becomes MASTER, and all other connected devices are at this point considered SLAVES until a STOP condition is issued.

Instead of sending a STOP to end the transaction, the MASTER can send a new START condition. This is called a REPEATED START, and leaves no possibilities for other masters to start a transaction as they could after a STOP.

Figure 2-1. Start and stop conditions.



2.2 Address

After the START condition, a 7 bit ADDRESS (A) followed by a READ/WRITE (R/\overline{W}) bit is sent. The MASTER transmits the SLAVE address of the device it is accessing. The R/\overline{W} bit is transmitted as the last bit and specifies the direction of the transaction. A SLAVE recognizing its ADDRESS will respond by pulling the data line low the next SCL cycle (ACKNOWLEDGE), while all other slaves should keep the TWI lines released, and wait for the next START and ADDRESS.

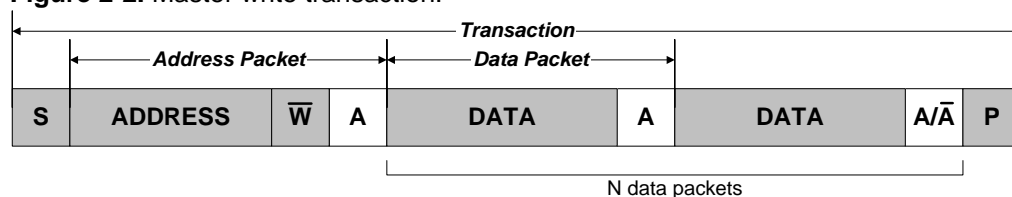
2.3 Data transfer

If the R / \overline{W} bit is low, it indicates a MASTER WRITE transaction, and the MASTER will transmit its data after the slave has acknowledged its address. Figure 2-2 shows a typical MASTER WRITE transaction. Note that there can be an arbitrary number of DATA packets within one transaction.

Table 2-1. Notations used in following protocol diagrams.

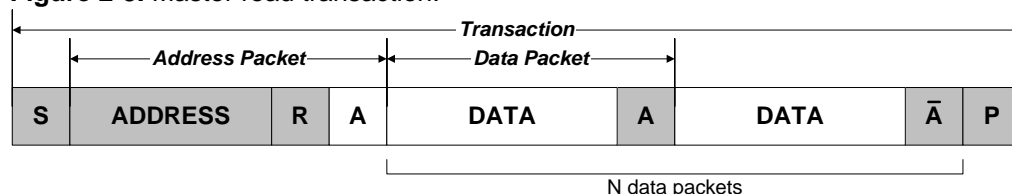
Notation	Description
S	START condition
Sr	REPEATED START condition
R	R/W bit high, indicating master read transaction
\overline{W}	R/W bit low, indicating master write transaction
A	Acknowledge (ACK)
\overline{A}	Not acknowledge (NACK)
P	STOP condition
	Gray background indicating data direction from master to slave
	White background indicating data direction from slave to master
	Diagonals indicating data direction set by last R / \overline{W} bit

Figure 2-2. Master write transaction.



A high R / \overline{W} bit indicates a MASTER READ. The master continues to generate the clock, while the SLAVE outputs its data on SDA, one bit at a time. When a bit is available for the MASTER to read, the SLAVE releases SCL so the master can provide the clock signal. Figure 2-3 shows a typical MASTER READ transaction.

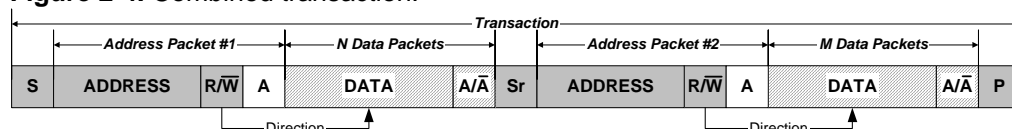
Figure 2-3. Master read transaction.





A transaction can also be combined as illustrated in Figure 2-4. Instead of sending a STOP to end the first part of the transaction, the MASTER sends a REPEATED START and ADDRESS including the R/\bar{W} bit. This allows the MASTER to change the direction of the transaction.

Figure 2-4. Combined transaction.



2.4 Clock stretching

The clock is always controlled by the MASTER, but can be held low any time by any device on the bus. In this way, a SLAVE device can hold back a transaction, e.g. if it needs more time to process data. Due to the bus topology, the master cannot continue clocking if SCL is held low. A slave pulling the SCL line low after the MASTER has released it is said to perform “clock stretching”.

2.5 Arbitration

As the TWI bus is a multi master bus, it's possible that two devices initiate a transfer at the exact same time. Arbitration is carried out through the next stages of the transaction, and the first device attempting to transmit a logical ‘1’ while another device transmits ‘0’ will lose arbitration. This can due to the physical characteristics of the bus easily be detected. If one device pulls a line low, the others cannot transmit high. When a device has lost arbitration, it must stop transmitting and wait until the next STOP condition before trying to take control of the bus again.

3 The XMEGA TWI module

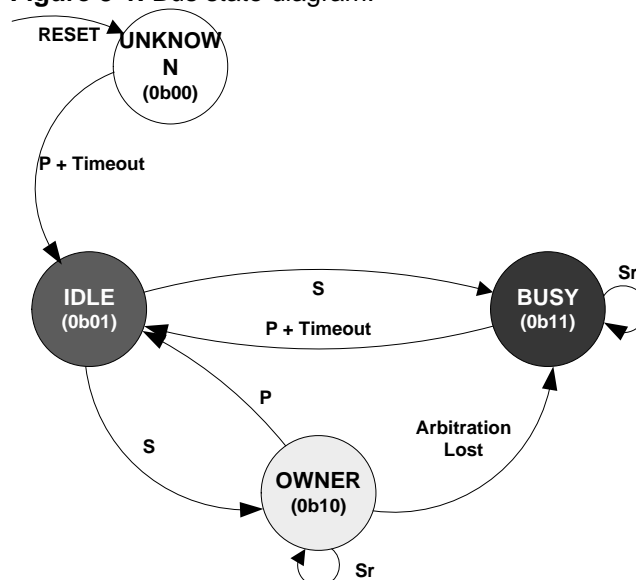
The XMEGA TWI module is separated into a master and a slave module, and the two modules can be enabled separately.

The master and slave have one common configuration register. TWI Control Register (TWIx.CTRL) holds one bit selecting external driver interface, two and four wire mode. Other than this, all control and status bits are found in separate registers for the two modules.

3.1 Bus state logic

In addition to the master and slave modules, there is a bus state logic monitoring activity on the bus. Information from this logic is used by hardware to determine the bus state (“unknown”, “idle”, “owner”, or “busy”), detecting START/Repeated START (S/Sr) and STOP (P) conditions, detecting bus collision, and to identify bus errors. The bus state logic is designed to operate in all sleep modes including power down mode.

Figure 3-1. Bus state diagram.



The bus state machine is active when the master is enabled. Important to consider, is how the initial state of the TWI bus is set. After reset or enabling the TWI module, the state is unknown, but will be changed to idle after a stop condition or a defined time out period. As I2C (unlike SMBus™) does not specify a time out period, the application can force the state to idle. Care must be taken in a multi master system. However, the error detection capabilities (arbitration) will bring the TWI to a known state e.g. after interference with an ongoing transaction.

3.2 Master

The TWI master module consists of the baud rate generator, status and control logic with supporting registers listed in the Table 3-1.

Table 3-1. Master module registers.

Register name	Symbolic name
TWI Master Control Register A	TWIX.MASTER.CTRLA
TWI Master Control Register B	TWIX.MASTER.CTRLB
TWI Master Control Register C	TWIX.MASTER.CTRLC
TWI Master Status Register	TWIX.MASTER.STATUS
TWI Master Baud Rate Register	TWIX.MASTER.BAUD
TWI Master Transmit Address Register	TWIX.MASTER.ADDR
TWI Master Data Register	TWIX.MASTER.DATA

3.2.1 Interrupts

Interrupt-controlled software is recommended for a TWI system, but in a single master system with relaxed timing requirements, polling can be acceptable. The TWI master driver included in this application note is interrupt-based.

TWI master interrupts are separated in two main occurrences, master read and master write. The read interrupt flag is set whenever a master read operation is



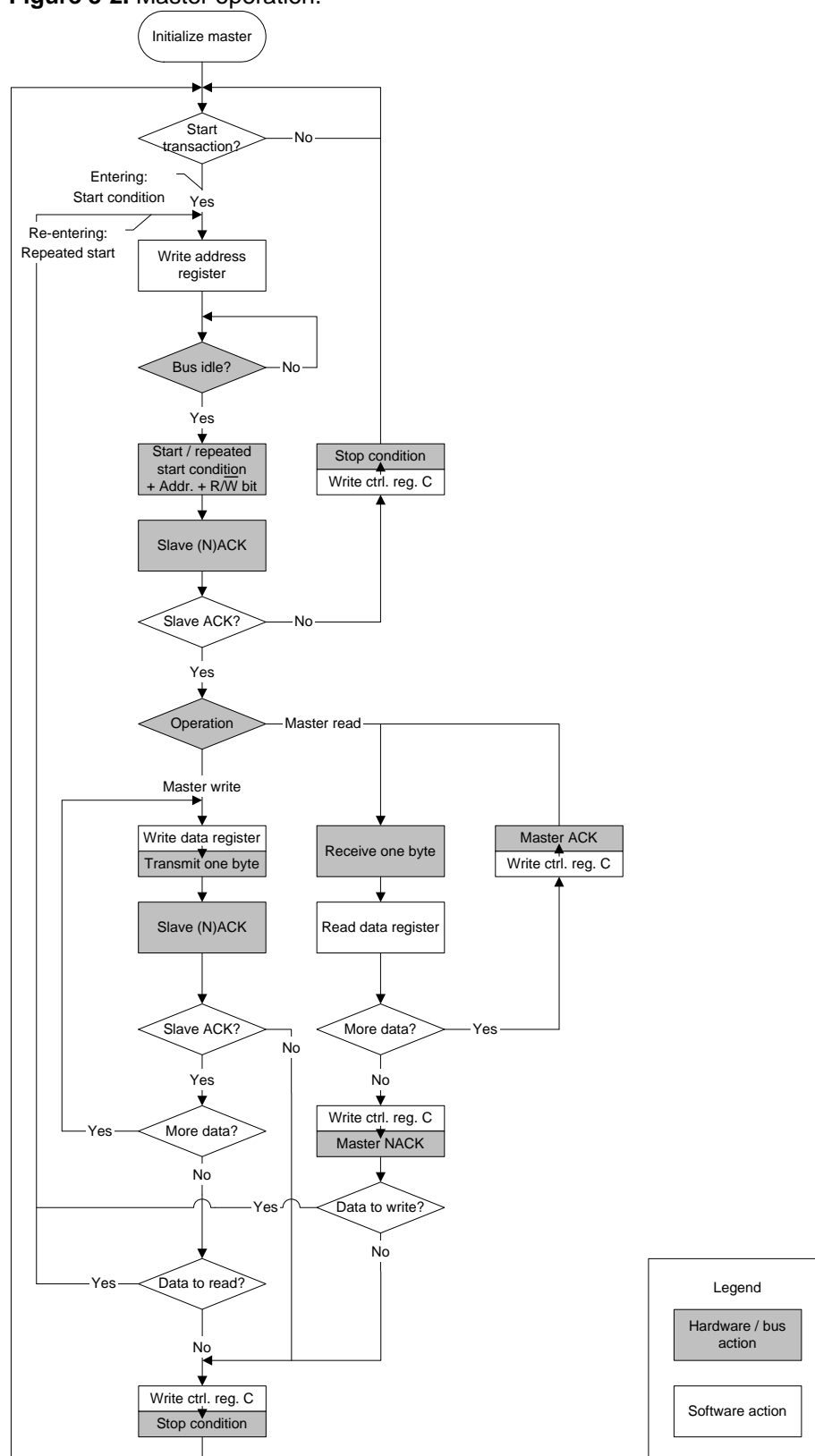
successfully completed, not losing arbitration or bus errors detected, and will if enabled trigger a master read interrupt. The write interrupt flag is set on completion of a master write operation. In addition to the arbitration lost or bus error flag, it is also set to signal these errors, also during a master read operation. If enabled, a master write interrupt is triggered.

Note that even if master read and master write interrupts are individually enabled, they share the same interrupt vector.

3.2.2 Master operation

An overview of master operation is illustrated in Figure 3-2. It indicates the flow in a master write and/or read transaction. For further details, please refer to the code included in this application note.

Figure 3-2. Master operation.



3.3 Slave

The TWI slave module consists of status and control logic with supporting registers listed in Table 3-2. As only the master can generate the clock signal, the slave does not include a baud rate generator.

Table 3-2. Slave module registers.

Register name	Symbolic name
TWI Slave Control Register A	TWIX.SLAVE.CTRLA
TWI Slave Control Register B	TWIX.SLAVE.CTRLB
TWI Slave Status Register	TWIX.SLAVE.STATUS
TWI Slave Transmit Address Register	TWIX.SLAVE.ADDR
TWI Slave Data Register	TWIX.SLAVE.DATA

3.3.1 Interrupts

As for the master, interrupt-controlled software is also recommended for the slave module. There are two main sources for interrupts, slave address recognition and slave data reception or transmission.

Slave address or stop interrupt is triggered when the address recognition logic detects a valid address. In addition, this interrupt is also triggered on a transmit collision or stop condition. The stop condition triggering is individually enabled through a separate control bit. Slave data interrupt is triggered when a slave byte transmit or receive is successfully completed, without any bus error or transmit collision.

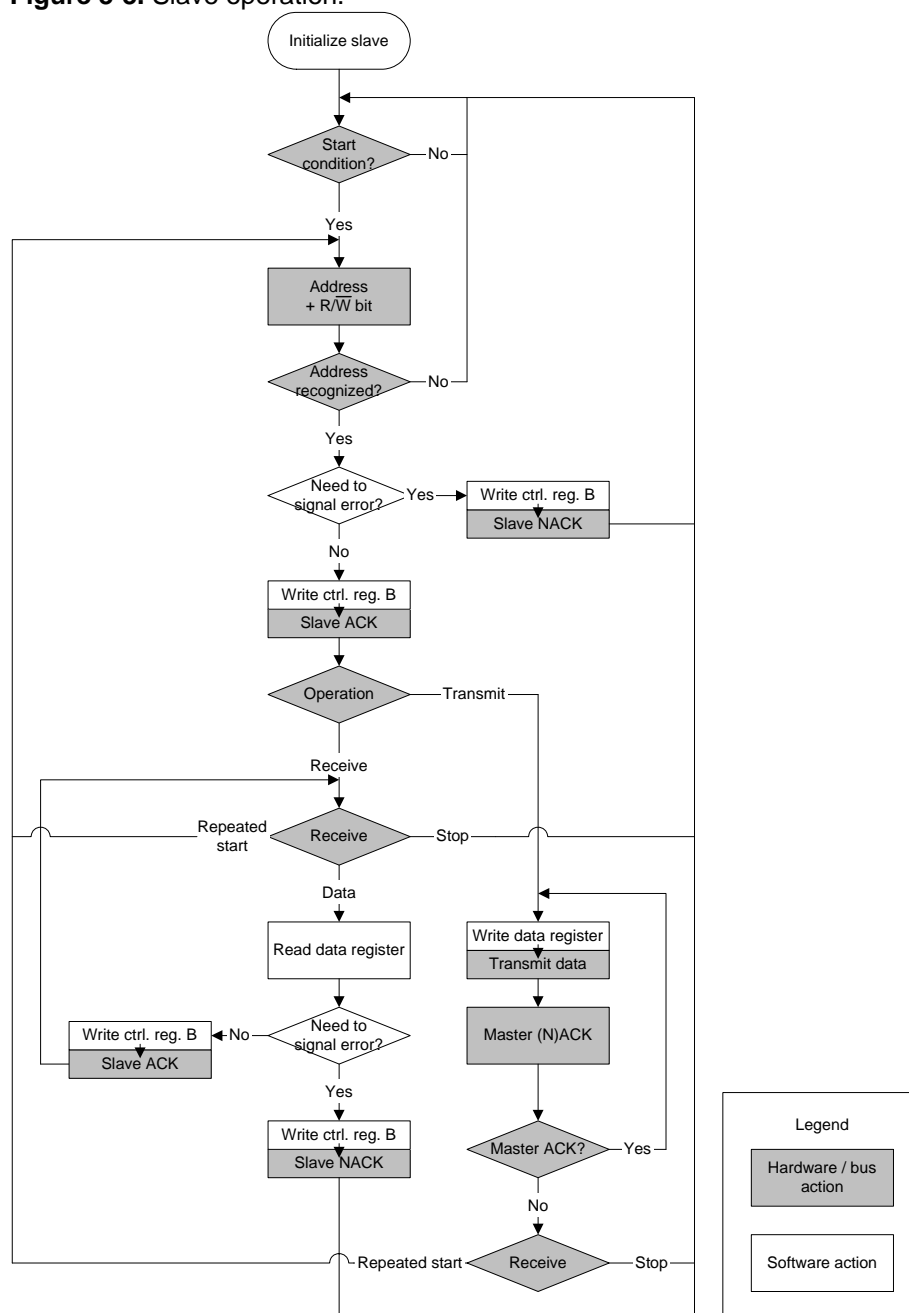
The logic detecting bus errors is shared between the master and slave modules, and detection depends on the master module being enabled and the peripheral clock being at least 4 x the SCL frequency. Without bus error detection, the SLAVE can operate at any peripheral frequency.

Note that even if the slave address and data interrupts are individually enabled, they share the same interrupt vector.

3.3.2 Slave operation

An overview of slave operation is illustrated in Figure 3-3. It illustrates the flow in a slave read and/or write transaction. For further details, please refer to the code included in this application note.

Figure 3-3. Slave operation.



4 Driver Implementation

This application note includes a source code package with basic interrupt-driven drivers for the TWI master and slave implemented in C. An example using a TWI master module to communicate with a TWI slave module using the drivers is included. It is written for the IAR Embedded Workbench® compiler.

Note that this TWI driver is not intended for use with high-performance code. It is designed as a library to get started with the TWI. For timing and code space critical



application development, you should access the TWI registers directly. Please refer to the driver source code and device datasheet for more details.

4.1 Files

The source code package consists of the following files:

- *twi_example.c* – example code using the TWI drivers
- *twi_master_driver.c* – master driver source file
- *twi_master_driver.h* – master driver header file
- *twi_slave_driver.c* – slave driver source file
- *twi_slave_driver.h* – slave driver header file

For a complete overview of the available driver interface functions and their use, please refer to the source code documentation.

4.2 Doxygen Documentation

All source code is prepared for automatic documentation generation using Doxygen. Doxygen is a tool for generating documentation from source code by analyzing the source code and using special keywords. For more details about Doxygen please visit <http://www.doxygen.org>. Precompiled Doxygen documentation is also supplied with the source code accompanying this application note, available from the *readme.html* file in the source code folder.



Headquarters

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

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel 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

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2008 Atmel Corporation. All rights reserved. Atmel®, logo and combinations thereof, AVR® and others, are the registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.