AVR1612: PDI programming driver

Features

- PDI driver for devices with PDI programming interface as target MCU's
- PDI instruction set and timing
- Programming application example for ATAVRXPLAIN/AT90USB1287
 Chip erase
 - Write and read application flash
 - Write and read EEPROM
 - Read and write Fuses
 - Read and write lock-bits
 - Read target device id

1 Introduction

The Program and Debug Interface (PDI) is an $Atmel^{\ensuremath{\mathbb{R}}}$ proprietary interface for external programming and on-chip debugging of the device.

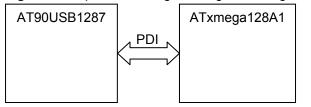
The PDI supports high-speed programming of all Non-Volatile Memory (NVM) spaces; Flash, EEPOM, Fuses, Lock-bits and the User Signature Row. This is done by accessing the NVM Controller trough the PDI interface, and executing NVM Controller commands.

The PDI is a 2-pin interface using the Reset pin for the clock input (PDI_CLK), and the dedicated pin for data input and output (PDI_DATA).

This application note describes how to implement PDI programming. It is based on the Atmel Xplain evaluation board and clarifies the protocol and timing of the PDI programming. Please also refer to application note AVR1907: Xplain Hardware User's Guide for how to start with the kit.

The Figure 1-1 shows the Xplain PDI programming block diagram. The firmware of the application is for the Atmel AT90USB1287 microcontroller; however it can be easily ported to other Atmel's chip that features USART.

Figure 1-1. Xplain PDI Programming block diagram.





8-bit **AVR**[®] Microcontrollers

Application Note

Rev. 8282A-AVR-11/10





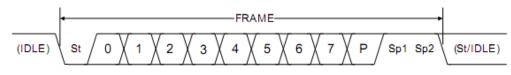
2 PDI Target Implementation

2.1 PDI Frame Format

The PDI physical layer uses a standard UART frame format. A serial frame is defined to be one character of eight data bits with start and stop bits and an even parity bit.

In case of write collisions or parity errors a break character (all data, parity and stop bits set low) can be sent. Please refer to the datasheet for more information about the break character. The Figure 2-1 describes the PDI serial frame format.

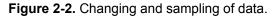
Figure 2-1. PDI serial frame format.

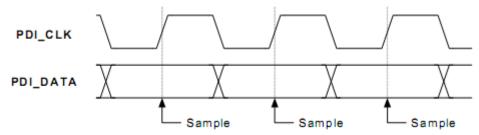


2.2 Serial transmission and reception

The PDI physical layer is either in Transmit (TX) or Receive (RX) mode of operation. By default it is in RX mode, waiting for a start bit.

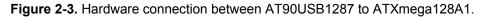
The programmer and the PDI operate synchronously on the PDI_CLK provided by the programmer. The dependency between the clock edges and data sampling or data change is fixed. As illustrated in Figure 2-2, output data (either from the programmer or from the PDI) is always set up (changed) on the falling edge of PDI_CLK, while data is always sampled on the rising edge of PDI_CLK.

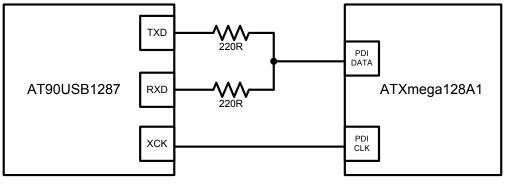




2.3 Hardware Connection with AT90USB1287's USART

The Atmel Xplain kit uses the Atmel AT90USB1287's USART port to implement the PDI port's hardware. Figure 2-3 shows the PDI hardware connection between the AT90USB1287 to the Atmel ATxmega128A1.





2.4 PDI instruction set

The PDI has a small instruction set that is used for all access to the PDI itself and to the internal interfaces. All instructions are byte instructions. Most of the instructions require a number of byte operands following the instruction. The instructions allow for an external programmer to access the PDI Controller, the NVM Controller and the NVM memories. Figure 2-4 shows the PDI instructions set, while Table 2-1 shows gives more details about each instruction.





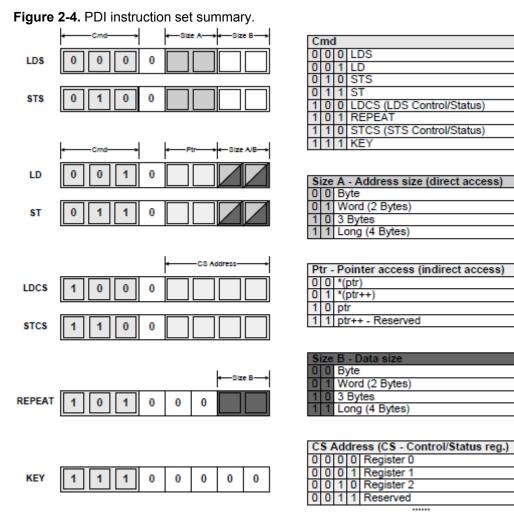


Table 2-1. PDI Instruction Set

PDI Command	Command Description	
LDS (0x00)	Load data from PDIBUS Data Space using direct addressing	
STS (0x40)	Store data to PDIBUS Data Space using direct addressing	
LD (0x20)	Load data from PDIBUS Data Space using indirect addressing	
ST (0x60)	Store data to PDIBUS Data Space using indirect addressing	
LDCS (0x80)	Load data from PDI Control and Status Register Space	
STCS (0xC0)	Store data to PDI Control and Status Register Space	
KEY (0xE0)	Set Activation Key	
REPEAT (0xA0)	Set Instruction Repeat Counter	

1 1 1 1 Reserved

Note: For more information, please refer to "PDI instruction set summary" in the "Program and Debug Interface" section of XMEGA A MANUAL

4

AVR1612

PDI instruction usage example: Write a 32-bit address into PDI Controller's pointer Pseudo Directive: ST ptr 0x12345678 PDIBUS hex code: 0x6B 0x12 0x34 0x56 0x78 "0x6B" stands for "ST ptr" command. NOTE Write a value to a address with *(ptr++) instruction through the PDI Controller Pseudo Directive: ST *(ptr++) 0xFF PDIBUS hex code: 0x67 0xFF NOTE "0x67" stands for "ST *(ptr++)" command. Write the repeat number into PDI Controller (REPEAT command uses BIG_ENDIAN data) Pseudo Directive: REPEAT 0x1234 PDIBUS hex code: 0xA1 0x34 0x12 NOTE "0xA1" stands for "REPEAT" command and 2 bytes data length. Load data from PDIBUS Data Space using indirect addressing **Pseudo Directive:** LD *(ptr++) PDIBUS hex code: 0x27 Then read the PDIBUS for the loading data. NOTE "0x27" stands for "LD *(ptr++)" command. Please refer to "29.5.7 Instruction Set Summary" section of the XMEGA A MANUAL for more information.

2.5 NVM Commands

The NVM commands that can be used for accessing the NVM memories from external programming are listed in Table 2-2. This is a super-set of the commands available for self-programming. Each command has to be loaded to the target NVM CMD register using the PDI commands.

For external programming, the Trigger for Action Triggered Commands is to set the CMDEX bit in the NVM CTRLA register (CMDEX). The Read Triggered Commands are triggered by a direct or indirect Load instruction (LDS or LD) from the PDI (PDI Read). The Write Triggered Commands is triggered by a direct or indirect Store instruction (STS or ST) from the PDI (PDI Write).

CMD[6:0]	Commands/Operation	Trigger
0x00	No Operation	-
0x40	Chip Erase ⁽¹⁾	CMDEX
0x43	Read NVM	PDI Read
Flash Page Buffer		
0x23	Load Flash Page Buffer	PDI Write
0x26	Erase Flash Page Buffer	CMDEX

Table 2-2. NVM commands available for external programming.





CMD[6:0]	Commands/Operation	Trigger
Flash	·	
0x2B	Erase Flash Page	PDI Write
0x2E	Flash Page Write	PDI Write
0x2F	Erase & Write Flash Page	PDI Write
0x78	Flash CRC	CMDEX
Application Sec	tion	
0x20	Erase Application Section	PDI Write
0x22	Erase Application Section Page	PDI Write
0x24	Write Application Section Page	PDI Write
0x25	Erase & Write Application Section Page	PDI Write
0x38	Application Section CRC	CMDEX
Boot Loader Se	ction	
0x68	Erase Boot Section	PDI Write
0x2A	Erase Boot Loader Section Page	PDI Write
0x2C	Write Boot Loader Section Page	PDI Write
0x2D	Erase & Write Boot Loader Section Page	PDI Write
0x39	Boot Loader Section CRC	NVMAA
Calibration and	User Signature sections	
0x03	Read User Signature Row	PDI Read
0x18	Erase User Signature Row	PDI Write
0x1A	Write User Signature Row	PDI Write
0x02	Read Calibration Row	PDI Read
Fuses and Lock	Bits	
0x07	Read Fuse	PDI Read
0x4C	Write Fuse	PDI Write
0x08	Write Lock Bits	CMDEX
EEPROM Page I	Buffer	
0x33	Load EEPROM Page Buffer	PDI Write
0x36	Erase EEPROM Page Buffer	CMDEX
EEPROM		
0x30	Erase EEPROM	CMDEX
0x32	Erase EEPROM Page	PDI Write
0x34	Write EEPROM Page	PDI Write
0x35	Erase & Write EEPROM Page	PDI Write
0x06	Read EEPROM	PDI Read

Notes: For more information, please refer the "External Programming" section of XMEGA A MANUAL

1. If the EESAVE fuse is programmed the EEPROM is preserved during chip erase.

6

3 The Programming Interface

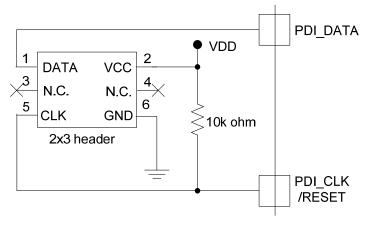
The PDI low level driver layer handles the basic interface to the USART driver. The physical layer uses a bi-directional half-duplex synchronous serial receiver and transmitter (as a USART in USART mode). The physical layer includes start-of-frame detection, frame error detection, parity generation, parity error detection, and collision detection. The PDI is accessed through two pins. Table 3-1 shows the PDI interface checklist.

 Table 3-1. PDI port interface checklist.

Signal name	Recommended pin connection	Description
	A reset pull-up should be 10k or weaker, or be removed altogether. Any reset decoupling capacitors should be removed if PDI programming and debugging is used.	
	Other external reset sources driving this line should be disconnected.	
PDI_CLK	(Any load on the clock line may give a delay on the clock edge that causes data bit to sampled/generated too late and result in communication failure.)	PDI_CLK: clock input/Reset pin (internally pulled-up)
PDI_DATA	Connect to programming header/test point only	PDI_DATA: PDI data input/output (internally pulled-down)

In addition to these two pins, VCC and GND must also be connected between the External Programmer/debugger and the device. Figure 3-1 shows a typical connection.

Figure 3-1. PDI connection.



3.1 Overview of the PDI programming process

The process of the PDI programming is shown below:

- 1. Enable the PDI module.
- 2. Enter external programming mode.
- 3. Program or read the memory (Flash/EEPROM /Fuses/Lock-bits) with PDI.
- 4. Exit the PDI programming mode.





3.2 Enable the PDI module

The PDI Physical must be enabled before it can be used. This is done by first forcing the PDI_DATA line high for a period longer than the equivalent external reset minimum pulse width (refer to device data sheet for reset characteristics).

The first PDI_CLK cycle must start no later than 100 μ S after the RESET functionality of the Reset pin was disabled. If this does not occur in time the RESET functionality of the Reset pin is automatically enabled again and the enabling procedure must start over again.

After this sequence, the PDI is enabled and ready to receive instructions. The enable sequence is shown in Figure 3-2.

Figure 3-2. Sequence for enabling the PDI.

	Disable RESET function on Reset (PDI_CLK) pin Activate PDI
PDI_DATA	within 100us
PDI_CLK	

3.3 Enter external programming mode

Even after an external programmer has established communication with the PDI module, the internal interfaces are not accessible by default. To get access to the NVM Controller and the NVM memories for programming, a unique key must be signalized by using the KEY instruction. The internal interface is accessed as one linear address space using a dedicated bus (PDIBUS) between the PDI and the internal interfaces.

PDI Control and Status Register Space can be accessed with STCS (Store) and LDCS (Load) instruction. Please refer to "29.7 Register Description - PDI Control and Status Register" section and "29.5.7 Instruction Set Summary" section of the XMEGA A MANUAL for more information.

The key that must be sent using the KEY instruction is 64 bits long. The key that will enable NVM Programming is:

0x1289AB45CDD888FF

The sequence of entering external programming is as following:

- 1. Load the PDI RESET register with the Reset Signature (0x59).
- 2. Load the correct NVM key in the PDI.
- 3. Poll NVMEN in the PDI Status Register (PDI STATUS) until NVMEN is set.

When the NVMEN bit in the PDI STATUS register is set the NVM interface is active from the PDI.

3.4 Memory Programming

This section describes how to program the Non Volatile Memory (NVM) in XMEGA[®] with external programming. The NVM consist of the Flash Program Memory, User Signature and Calibration rows, Fuses and Lock Bits, and EEPROM data memory.

For external programming the device is accessed through the PDI and PDI Controller, using PDI physical connection. Through the PDI, the external programmer access all

8 AVR1612

NVM memories and NVM Controller using the PDI Bus. Doing this all data and program memory spaces are mapped into the linear PDI memory space.

Figure 3-3 shows the PDI memory space and the base address for each memory space in the Atmel ATxmega128A1.

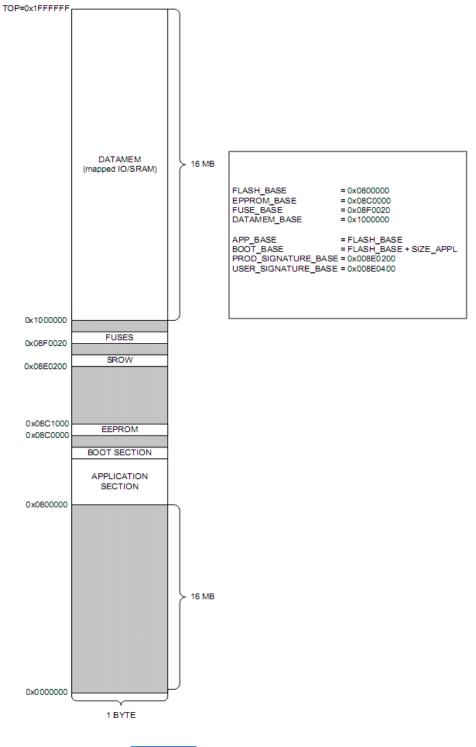


Figure 3-3. Memory map for PDI accessing the data and program memories.





3.4.1 Chip Erase

The Chip Erase command is used to erase the Flash Program Memory, EEPROM and Lock-Bits. The User Signature Row, Calibration Row and Fuses are not affected.

The Chip Erase sequence is as below:

- 1. Load the NVM CMD register with Chip Erase command (refer to Table 2-2).
- 2. Set the CMDEX bit in NVM CTRLA register.

NOTE The Chip Erase command disables the PDI controller and the NVM. Poll the NVMEN bit until this is set, indict the PDI controller is enabled.

3.4.2 Program Flash and EEPROM Page

Flash and EEPROM page programming is done by first filling the associated page buffer, and then writing the entire page buffer to a selected page in Flash or EEPROM.

The size of the page buffers depend on the Flash and EEPROM size in each device, and details on page size and page number is described in each device data sheet.

3.4.2.1 Flash and EEPROM Programming Sequence

Before programming a Flash or EEPROM page with the data in the Flash or EEPROM page buffer, the Flash or EEPROM page must be erased. Programming an un-erased Flash or EEPROM Page will corrupt the content in the Flash or EEPROM Page.

- 1. Erase Flash or EEPROM Page Buffer.
- 2. Load the Flash or EEPROM Page Buffer.
- 3. Perform a Page Erase and Write.

3.4.2.2 Erase Page Buffer

The Erase Flash Page Buffer and Erase EEPROM Page Buffer commands are used to erase the Flash and EEPROM page buffers.

- 1. Load the NVM CMD register with the Erase Flash/EEPROM Page Buffer command.
- Set the CMDEX bit in the NVM CTRLA register. This requires the timed CCP sequence during self-programming.

The BUSY flag in the NVM STATUS register will be set until the operation is completed.

3.4.2.3 Load Page Buffer

The Load Flash Page Buffer and Load EEPROM Page Buffer commands are used to load one byte of data into the Flash and EEPROM page buffers.

- 1. Load the NVM CMD register with the Load Flash/EEPROM Page Buffer command.
- 2. Write the selected memory address by doing a PDI Write operation.

Since the Flash page buffer is word accessing and the PDI uses byte addressing, the PDI must write the Flash Page Buffer in correct order.

3.4.2.4 Erase & Write Page	
	The Erase & Write Application Section Page, Erase & Write Boot Loader Section Page, and Erase & Write EEPROM Page is used to erase one page and then write a loaded Flash/EEPROM page buffer into that page in the selected memory space, in one atomic operation.
	 Load the NVM CMD register with Erase & Write Application Section/Boot Loader Section/User Signature Row/EEPROM Page command. Write the selected page by doing a PDI Write. The page is written by addressing any byte location within the page.
	The BUSY flag in the NVM STATUS register will be set until the operation is finished.
3.4.3 Read NVM	
	The Read NVM command is used to read the Flash, EEPROM, Fuses, and Signature and Calibration row sections.
	 Load the NVM CMD register with the Read NVM command. Read the selected memory address by performing a PDI Read operation.
NOTE	The address is PDI memory space address which described in each device data sheet.
3.4.4 Write Fuse/Lock Bit	
	The Write Fuse and Write Lock Bit command is used to write the fuses and the lock bits to a more secure setting.
	1 Load the NV/M CMD register with the Write Even / Loak Bit command

- 1. Load the NVM CMD register with the Write Fuse/ Lock Bit command.
- 2. Write the selected fuse or Lock Bits by doing a PDI Write operation.

The BUSY flag in the NVM STATUS register will be set until the command is finished. For lock bit write the LOCK BIT write command can also be used.

3.5 Exit the PDI programming

If there is no activity on the PDI_CLK line for approximately 100 μ s, the PDI automatically disabled. Then set the PDI_CLK to High and set the PDI_DATA to Low.

4 Example of programming sequence

After enabling the PDI module and entering external programming mode, the device is ready for programming and reading the memory. For more information about the PDI initialization, please refer to Section 2.

- Read the memory (include Flash, EEPROM, User Signature, Fuse bits)
 - Use STS instruction to write the "Read NVM command (0x43)" to the NVM controller' CMD register. The CMD register's address is 0x01CA (The NVM Controller's base address is 0x01C0 and the CMD register's offset address is 0x0A). Please refer to "4.21 Register Summary - NVM Controller" section and "31. Peripheral Module Address Map" section of the XMEGA A MANUAL for more information.
 - 2. Set the memory address with "ST ptr" command which is described in Section 3.4. The address is mapped into the PDI memory space.
 - 3. Set the data length into the repeat counter with "REPEAT" command.
 - 4. Send the "LD *(ptr++)" command to the PDI controller.
 - 5. Poll to read the PDIBUS until the data delivery completion.



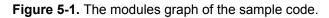


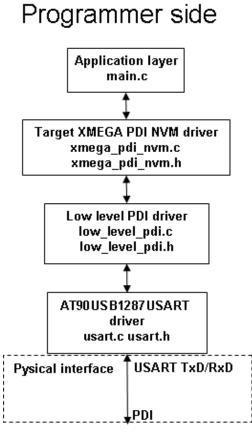
- Erase and program the flash memory
 - 1. Erase the flash page buffer.
 - i. Use the "ST ptr" command to set the address 0x00000000.
 - ii. Write "Erase Flash Page Buffer" (0x26) command to the NVM controller.
 - iii. Dummy write the flash page buffer with "ST *(ptr++)" command.
 - iv. Polling the NVM busy bit until it has been cleared.
 - 2. Load the flash page buffer.
 - i. Write the "Load Flash Page Buffer" (0x23) command to the NVM controller.
 - ii. Use the "ST ptr" command to set the start address.
 - iii. Set the data length into the repeat counter with "REPEAT" command.
 - iv. Send the "ST *(ptr++)" command to the PDI controller.
 - v. Send the writing data through the PDIBUS.
 - 3. Write "Erase & Write Application Section Page" (0x25) command to the NVM controller.
 - 4. Use the "ST ptr" command to set the start address.
 - 5. Dummy write the flash page with "ST *(ptr++)" command.
 - 6. Polling the NVM busy bit until it has been cleared.
- Erase and program the EEPROM memory
 - 1. Erase the flash page buffer.
 - i. Use the "ST ptr" command to set the address 0x00000000.
 - ii. Write "Erase EEPROM Page Buffer" (0x36) command to the NVM controller.
 - iii. Dummy write the flash page buffer with "ST *(ptr++)" command.
 - iv. Polling the NVM busy bit until it has been cleared.
 - 2. Load the flash page buffer.
 - i. Write the "Load EEPROM Page Buffer" (0x33) command to the NVM controller.
 - ii. Use the "ST ptr" command to set the start address.
 - iii. Set the data length into the repeat counter with "REPEAT" command.
 - iv. Send the "ST *(ptr++)" command to the PDI controller.
 - v. Send the writing data through the PDIBUS.
 - 3. Write "Erase & Write EEPROM Page" (0x35) command to the NVM controller.
 - 4. Use the "ST ptr" command to set the start address.
 - 5. Dummy write the flash page with "ST *(ptr++)" command.
 - 6. Polling the NVM busy bit until it has been cleared.
- Write the fuse bits
 - 1. Write the "Write Fuse" (0x4C) command to the NVM controller.
 - 2. Use the "STS ptr" command to set the fuse bits' value.
 - 3. Polling the NVM busy bit until it has been cleared.

5 The Hierarchy of the code

The code of this application can be divided into four layers. Figure 5-1 shows the hierarchy of this application code.

- 1. Application Layer: Integrate each module. User code may go into this layer.
- 2. The high level target XMEGA NVM driver, which interface the low level PDI driver.
- 3. The low level PDI driver uses the reduced instructions set for the PDI interface to communicate with the programmer USART driver for basic PDI operations.
- 4. AT90USB1287 USART driver is responsible for setting up the physical USART module based on the settings given from the PDI low level layer.





Target side





6 Quick Start Guide

NOTE

This section is intended as a step-by-step tutorial on how you get started using the Atmel Xplain kit programming the Atmel ATxmega128A1 chip.

The Atmel AT90USB1287 fuses needs to be set to clock the device to 8MHz that means the CKDIV8 fuse needs to be unchecked if checked.

- 1. Download and unzip the source code for AVR1612.
- Download and install Atmel AVR Studio[®] 4.18 (or later) from AVR Studio 4, WINAVR-20090313 (or later) from <u>http://sourceforge.net/projects/winavr/files/</u>.
- 3. Extract the AVR1612's code to a specific location (any location can be used, here used "C:\xplain_pdi_prog\").
- 4. Open either the GCC project file or the IAR[™] project file and compile the source code.
- 5. Connect the USB cable of Atmel Xplain to provide power to the Xplain board.
- 6. Connect a programmer (JTAGICE mkII or like) to the JTAG USB header on the Atmel Xplain.
- Start Atmel AVR Studio and select Tools -> Program AVR -> Connect -> then select the correct programmer and port. Program the Atmel AT90USB1287 device with compiled hex file.
- Exit the programming interface. And disconnect and reconnect the Xplain USB cable if necessary to reset the board. If every thing works the LEDs on the Atmel Xplain should now flash, indicating that the Atmel ATxmega128A1 target device is programmed with a test program stored inside the flash of the Atmel AT90USB1287.



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

© 2010 Atmel Corporation. All rights reserved. / Rev.: CORP072610

Atmel[®], Atmel logo and combinations thereof, AVR[®], AVR Studio[®], XMEGA[®] and others are registered trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.

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 THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, 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 AND 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 products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.