AVR32113: Configuration and Use of the Memory Management Unit

32-bit AVR® Microcontrollers

Application Note

Features

- **Translation lookaside buffers (TLB)**
- **Protected memory spaces**
- **Variable page size**
- **Uses exceptions for fast and easy management of TLB entries**

1 Introduction

Utilizing a memory management unit (MMU) in an application gives the benefit of virtual memory, where different memory pages can point to different physical memory. Virtual memory allows multiple processes to run with address protection and flexible memory mapping. All memory pages are configured individually with respect to size, access privileges and mapping.

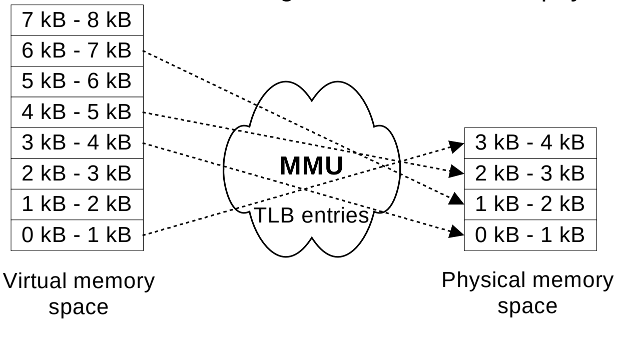
The AVR®32 MMU hardware has the assignment of converting the virtual addresses requested by the CPU to physical addresses. The MMU also raises exceptions if invalid addresses are accessed or if the running process doesn't have access to the memory page.

The MMU is also used to protect processes from each other, typically by operating systems. A process trying to access the memory segment of another process will be denied by the MMU in the AVR32 device and the CPU is informed by exceptions.

To speed up the process of converting from virtual to physical, the AVR32 MMU uses translation buffers, TLB. Depending on the device, it can have separate TLB for instructions and data or a unified TLB for both instructions and data.

The AVR32 MMU is fully configurable from software, giving it great amount of flexibility. There are special registers to ease the implementation of the memory management, making it easy to implement and with high performance.

Figure 1-1. MMU translating virtual addresses to physical addresses



Rev. 32047A-AVR32-09/06

2 Background

The AVR32 memory management unit (MMU) provides a highly flexible and configurable memory management solution. The MMU is fully configurable by the user, which allows advanced use for operating systems or simpler use for native applications.

The AVR32 has a powerful MMU, which allows efficient implementation of virtual memory and large memory spaces. The highly flexible MMU in the AVR32 has the features to implement basic or more advanced operating system memory mapping.

MMU is the hardware component that manages a systems virtual memory. The MMU includes a small amount of memory that manages the connection between virtual and physical addresses. This table is called the translation lookaside buffer, TLB. When a request for data is sent to the CPU, the MMU translates the requested address into the physical address. The TLB is used to speed up performance by enabling caching on often-used memory locations.

The MMU provides page size from 1 kB to 1 MB with a huge range of individual settings for each page. This application note addresses the basic functionality for the MMU provided with the AVR32 architecture.

2.1 MMU registers

For more details about the registers and bit-fields see the *AVR32 Architecture Manual*.

2.1.1 TLBEHI register

TLBEHI register

31	10	9	8	0
VPN		V	I	ASID

The translation buffer entry register high part (TLBEHI) is used for storing the virtual part of the page entry into the TLB with the possibility to connect the page to an a application space identifier (ASID).

- VPN – Virtual page number for the page entry.
- V – Valid page entry tells if an entry is valid.
- I – Instruction page entry, ignored on devices without ITLB (see section 4.1)
- ASID – Application space identifier can be used by the operating system when the MMU is in private memory mode.

2.1.2 TLBELO register

TLBELO register

31	10	9	8	7	4	2	1	0
PFN	C	G	B	AP	SZ	D	W	

The translation buffer entry low part register (TLBELO) is used for storing the physical part of the page entry into the TLB, along with the page configuration.

- PFN – Physical frame number, the address the virtual frame number is mapped.
- C – Cachable bit, enables caching if supported by device.

- G – Global bit, enables global address search.
- B – Bufferable bit, enables buffering if supported by device.
- AP – Access permissions bits controls read, write and execute access to an entry.
- SZ – Page size bits sets the page size.
- D – Dirty bit is set when the page is written to.
- W – Write through bit enables write through cache if supported by device.

2.1.3 PTBR register

PTBR register

31	0
PTBR	

The page table base address register (PTBR) can be used by software to hold the memory address in where the page table is stored.

2.1.4 TLBEAR register

TLBEAR register

31	0
TLBEAR	

The translation buffer exception address register (TLBEAR) contains the most resent virtual address of a MMU related exception.

2.1.5 MMUCR register

MMUCR register

31 26 20 14 8 5 4 3 2 1 0

IRP	ILA	DRP	DLA	--	S	N	I	M	E

The memory management unit control register (MMUCR) controls the use of the MMU.

- IRP – instruction replacement pointer, which instruction TLB entry to access.
- ILA – instruction lockdown amount, number of instruction TLBs to lock.
- DRP – data replacement pointer, which data TLB entry to access.
- DLA – data lockdown amount, number of data TLBs to lock.
- S – Segmentation bit enables segmentation.
- N – Not found bit, i.e. page miss when using the tlbs instruction.
- I – Invalidate bit invalidates all entries in the TLB.
- M – Mode bit selects between shared and private mode.
- E – Enable bit enables paging.

2.1.6 TLBARLO/TLBARHI registers

TLBARLO / TLBARHI register

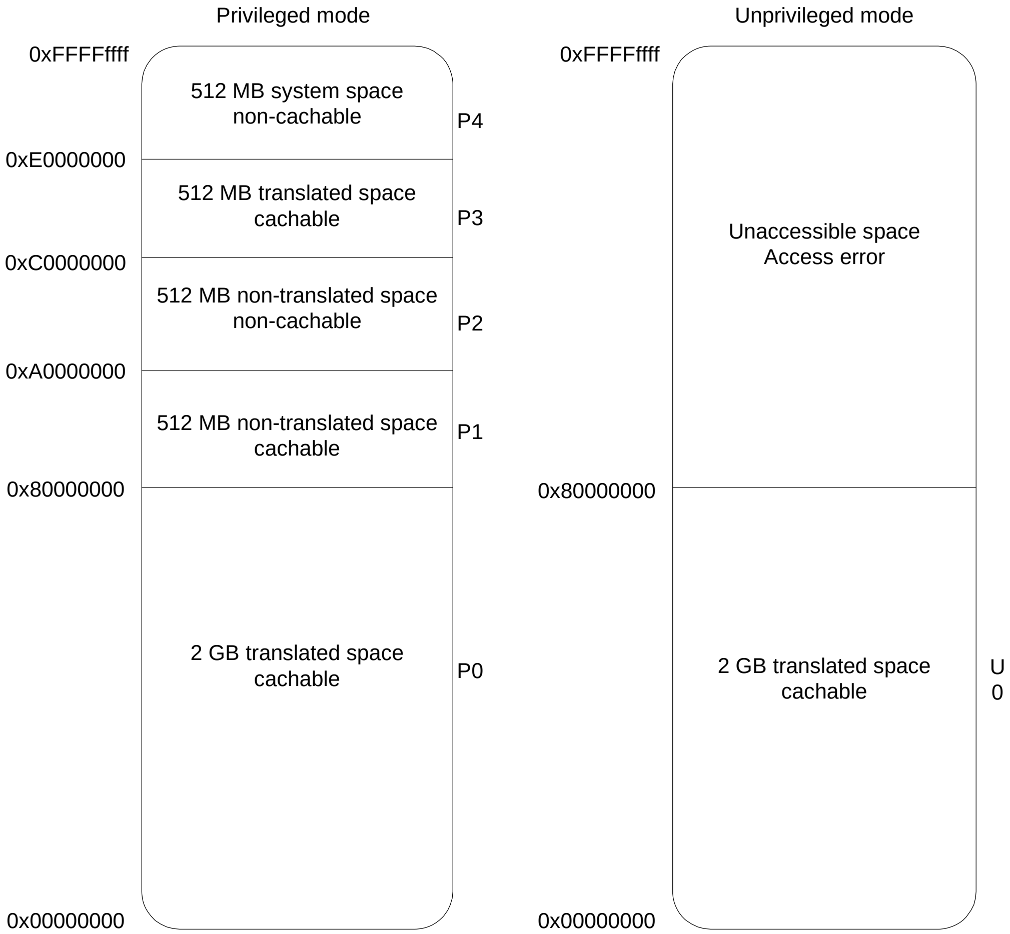
31	0
TLBARLO / TLBARHI	

The translation buffer accessed high and low register contains which TLB entries that have been accessed since the last reset of this register. The two 32 bit registers give a total of 64 bits that represents the TLB entries. Instruction and data TLB is separated by the I-bit in the TLBEHI register (se section 2.1.1).

2.2 Virtual memory space

The AVR32 architecture uses a 32-bit virtual memory space. The mapping and translation from virtual to physical addresses is done by the MMU.

Figure 2-1. Virtual memory space



1. The AVR32 use two distinct modes, privileged mode and unprivileged mode. The difference between these modes is covered in the *AVR32 Architecture Manual*. This application note will assume the privileged mode is used (i.e. the CPU in supervisor mode).

2.3 MMU configuration

The AVR32 can use both segmentation and page translation. As these two can be configured independently of each other, there are four different modes of operation as described in Table 2-1.

The segmentation is enabled by the S-bit in MMUCR, and page translation is enabled by the E-bit in MMUCR. Segmentation is enabled by default, and page translation is disabled after a reset.

Table 2-1. Modes of operation

Segment translation	Page translation	Description
Off	Off	Virtual and physical addresses are equal.
On	Off	The P0, P4 and U0 have no translation while P1, P2 and P3 are mapped to the physical location 0x00000000 to 0x1F000000.
Off	On	All addresses are mapped according to TLB entries [1].
On	On	P1 and P2 are mapped directly to the physical address range 0x00000000 to 0x1F000000, while P4 is mapped directly to the corresponding physical address. U0, P0 and P3 are mapped according to TLB entries [1].

Notes: 1. See section 4 for descriptions concerning the TLB

3 Enabling the MMU

As the MMU is optional in the AVR32 architecture, only segmentation is enabled by default. Page translation is enabled with the E-bit in the MMUCR register.

3.1 MMUCR – MMU control register

The MMUCR controls the operation of the MMU, making it possible to enable the MMU, enable segmentation and select mode as specified in Table 2-1.

The control register is also used for replacing and locking entries in the TLB. Four bit-fields are used for this purpose:

- IRP – Instruction replacement pointer
- ILA – Instruction lockdown pointer
- DRP – Data replacement pointer
- DLA – Data lockdown pointer

Each of these four bit-fields is up to 6 bits, allowing up to 64 valid pages in both the instruction TLB and data TLB.

The IRP and DRP fields give the index of the page entry to write or read from the TLB.

The ILA and DLA fields specify the number of entries to be locked down, counting from the first entry in the respective TLB. Thus, to lock down 10 entries these 10 entries have to be organized at the 10 first entries in the TLB.

3.2 Page table

The AVR32 MMU page table needs to be implemented in software along with an exception handler swapping entries in and out of the TLB (see section 4). This is usually handled by the operating system.

It is recommended that the page table is stored in the format given by the TLBELO register, making it possible to pass the entries directly into the MMU.

For more information see the *AVR32 Architecture Manual*.

3.3 Page entries

The AVR32 MMU can be configured to support pages from 1 kB to 1 MB.

Depending on the device specification, each page can be configured to be global, cacheable or bufferable, and the cache can be set to be write-through or write-back.

The entries have individual access permissions for read, write and execute for both privileged CPU mode and unprivileged CPU mode. This allows protecting memory segments from illegal access.

The dirty bit is set by hardware in the entry if it is written to. An exception will rise when the dirty bit is set, making it possible for the software to handle the dirty page.

For more information see the *AVR32 Architecture Manual*.

4 Translation lookaside buffer

In order to speed up the translation process, the AVR32 uses a special cache that contains buffered entries from the page table. This cache is called the translation lookaside buffer (TLB). One can use one unified TLB or two separate TLB, one for instructions and one for data. A single TLB can contain up to 64 different entries and each entry can be individually locked in the TLB to further increase performance.

4.1 TLB types

The TLB entries can be divided into instruction TLB (ITLB) and data TLB (DTLB) or unified TLB (UTLB). This is indicated in each TLB entry by the I-bit in the TLBEHI register. The TLB entries are indexed by using the irp- and drp-field in the MMUCR register.

For devices without ITLB entries the I-bit in TLBEHI is ignored by hardware and all TLB entries are indexed by the drp-field in the MMUCR register.

4.2 TLB structure

An entry in the TLB is divided into two parts. One part is describing the virtual section and the other part is describing the physical section. The fields of these two parts are extensively covered in the *AVR32 Architecture Manual*. Though the table entries organization may differ from this document to suit specific implementation needs.

4.3 TLB instructions

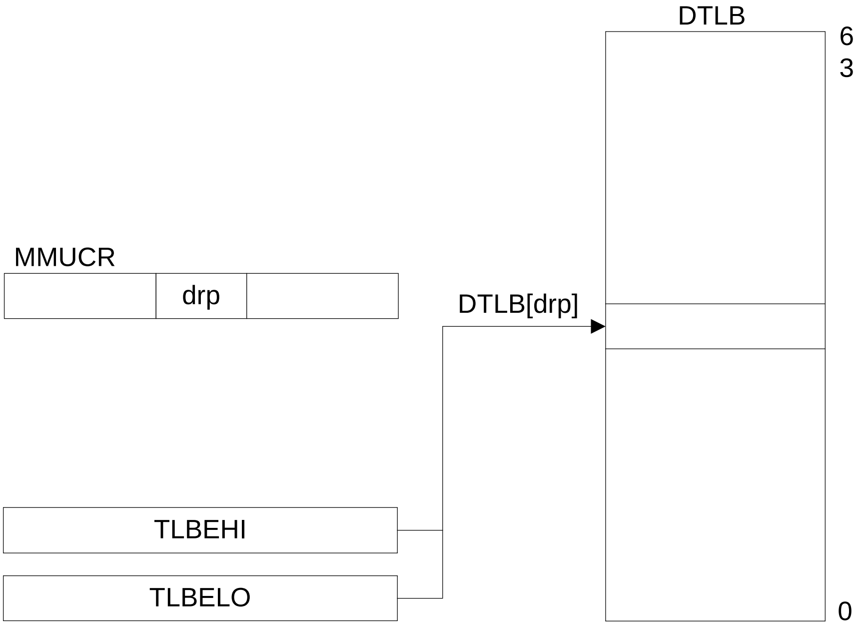
Associated with the TLB are three assembly instructions:

- tlbw – write a new entry to the TLB.
- tlbr – read an entry from the TLB.
- tlbs – search the TLB for an entry.

4.3.1 tlbw instruction

A *tlbw* instruction writes the contents of TLBEHI and TLBELO into the TLB. The *drp* or *irp* fields in the MMUCR register specify the index in the TLB the contents are written to. If an instruction is to be written, the *irp* field is used, for data the *drp* field is used. Before the *tlbw* instruction can be executed, the correct values must be set in the MMUCR registers, as well as TBLEHI and TBLELO. Figure 4-1 shows how a *tlbw* instruction can be executed.

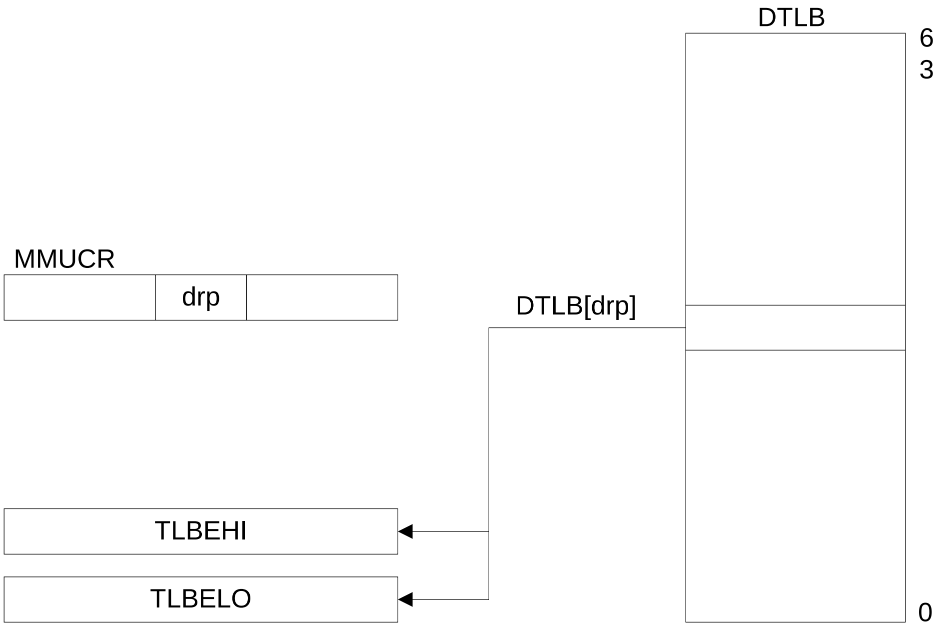
Figure 4-1. Data flow when *tlbw* instruction executed



4.3.2 tlbr instruction

A *tlbr* instruction reads an entry from the TLB and put the data TLBEHI and TLBELO. The *drp* or *irp* fields in the MMUCR register specify the index to be read in the TLB. If an instruction is to be read, the *irp* field is used, for data the *drp* field is used. Before the *tlbw* instruction can be executed, the correct values must be set in the MMUCR registers, as well as TBLEHI and TBLELO. Figure 4-2 shows how a *tlbr* instruction can be executed.

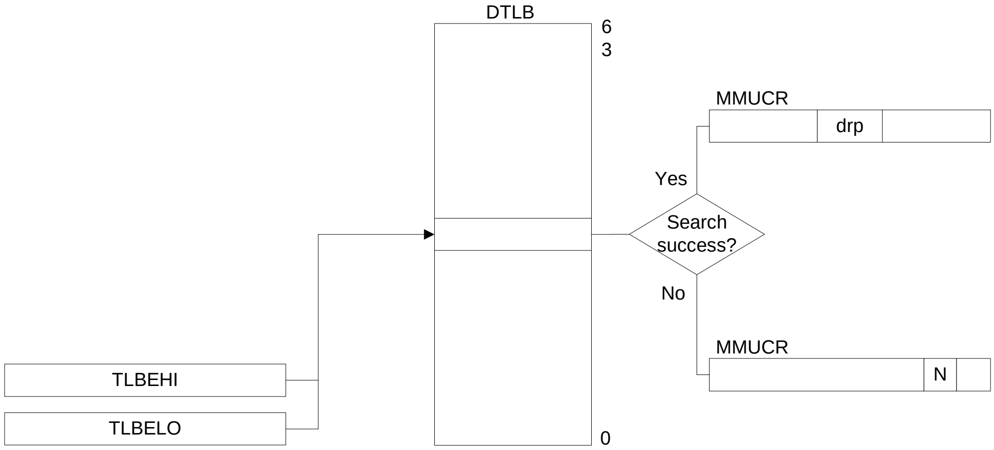
Figure 4-2. Data flow when *tlbr* instruction executed



4.3.3 tlbs instruction

A *tlbs* instruction searches the TLB for an entry matching the contents in TLBEHI and TLBELO. If the search is successful, the *drp* field in the MMUCR is updated with the address to the entry in TLB. If the search was unsuccessful the *not found* bit (N) in MMUCR is set. Figure 4-3 shows how a *tlbs* instruction can be executed.

Figure 4-3. Data flow when *tlbs* instruction executed



5 MMU application design considerations

5.1 Program counter (PC) and valid memory spaces

When switching mode on the MMU the program counter (PC) must be in a place where it can continue executing, if not the MMU will raise an exception. The TLB entries must be entered before a mode change.

5.2 MMU exceptions

Handling exceptions related to the MMU is mandatory when using the MMU, since unhandled MMU exceptions will lead to an unresolved state and the processor will need to be reset.

The exception handler also has to reside inside a valid page entry

5.3 TLB page flags

There are several flags possible to set for each page; they affect the page in different ways. The most important D-flag and V-flag must be set correct to avoid unexpected exceptions.

The dirty bit, D flag, must be set to dirty to allow the CPU to write to a memory address without rising an DTLB modified exception. This flag can also be used by the operating system to detect dirty pages and flush these to disk.

The valid bit, V flag, must be set if the page is valid. Entering an entry in the TLB without this bit set will result in a page miss exception when the memory area is accessed.

6 Implementations

6.1 Driver files

The driver consists of two files “mmu.c” and “mmu.h”. Where “mmu.h” declares all functions and “mmu.c” contains the source code.

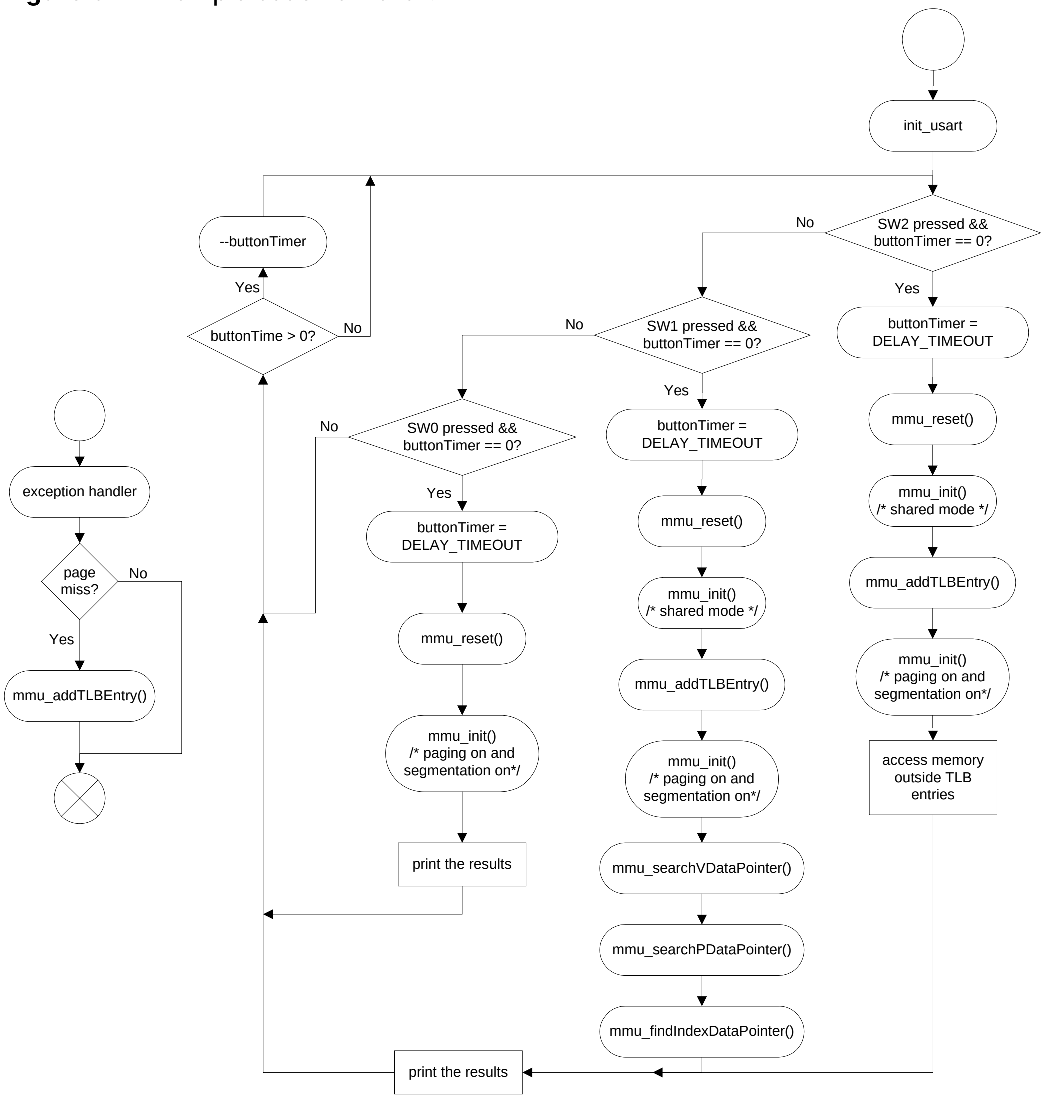
6.2 Example application

The example application, mmu_example.c, is using the MMU driver files to show how the MMU works when segmentation is on and paging off, and when both segmentation and paging is turned on.

Since the linker scripts supplied with the native GNU C compiler assumes segmentation is turned on, the example application will only demonstrate the use of the MMU when segmentation is turned on.

Figure 6-1 shows the flow of the example application. For detailed explanation of the functions please see the code documentation (see chapter 6.3).

Figure 6-1. Example code flow chart



6.3 Doxygen documentation

All source code is prepared for doxygen automatic documentation generation. Premade doxygen documentation is also supplied with the source to this application note, located in src/doxygen/index.html.

Doxygen is a tool for generating documentation from source code by analyzing the source code and using known keywords. For more details see http://www.stack.nl/~dimitri/doxygen/.

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