AVR914: CAN & UART based Bootloader for AT90CAN32, AT90CAN64, & AT90CAN128

8-bit AVR Microcontrollers

Application Note

1. Features

- **UART Protocol**
  - **UART used as Physical Layer**
  - **Based on the Intel Hex-type records**
  - **Auto-baud**
- **CAN Protocol**
  - **CAN used as Physical Layer**
  - **7 re-programmable ISP CAN identifiers**
  - **Auto-bitrate**
- **In-System Programming**
  - **Read/Write Flash and EEPROM memories**
  - **Read Device ID**
  - **Full chip Erase**
  - **Read/Write configuration bytes**
  - **Security setting from ISP command**
  - **Remote application start command**

2. Description

This document describes the UART & CAN bootloader functionality as well as the serial protocols to efficiently perform operations on the on chip Flash & EEPROM memories.

This bootloader implements the "In-System Programming" (ISP). The ISP allows the user to program or re-program the microcontroller on-chip Flash & EEPROM memories without removing the device from the system and without the need of a pre-programmed application.

The CAN & UART bootloader can manage a communication with an host through a serial network or serial line. It can also access and perform requested operations on the on-chip Flash & EEPROM memories.

3. Doc Control

Bootloader Revision	Purpose of Modifications	Compiler Version	Date
Rev. 1.0.0	First release	-	16/06/2003
Rev. 1.0.1 (7592A)	Updated for FLIP 2.4.4 Updated for AT90CAN128/64/32 Some bugs corrections	IAR Embedded Workbench for Atmel AVR 4.11A	19/10/2005
Rev. 1.0.1 (7592B)	Correction on CAN ID_PROG_START answer	IAR Embedded Workbench for Atmel AVR 4.11A	05/01/2006

7592B–AVR–01/06

4. Bootloader Environment

The **CAN & UART AT90CAN128/64/32 bootloader** is loaded in the "Bootloader Flash Section" of the on-chip Flash memory. The bootloader size is close to 8K bytes, so the physical "Bootloader Flash Section" is fully used. This section is reserved to the bootloader and the application program size must be lower or equal the "Application Flash Section" (c.f. Table 4-1 "Device Memory Mapping (byte addressing)" on page 2).

Table 4-1. Device Memory Mapping (byte addressing)

Memory		AT90CAN128	AT90CAN64	AT90CAN32
FLASH	Size	128 K bytes	64 K bytes	32 K bytes
	Add. Range	0x00000 - 0x1FFFF	0x00000 - 0x0FFFF	0x00000 - 0x07FFF
"Application Flash Section"	Size	120 K bytes	56 K bytes	24 K bytes
	Add. Range	0x00000 - 0x1DFFF	0x00000 - 0xDFFF	0x00000 - 0x05FFF
"Bootloader Flash Section"	Size	8 K bytes		
	Add. Range	0x1E000 - 0x1FFFF	0x0E000 - 0x0FFFF	0x06000 - 0x07FFF
"Boot Reset Address"		0x1E000	0x0E000	0x06000
EEPROM	Size	4 K bytes	2 K bytes	1 K bytes
	Add. Range	0x0000 - 0x0FFF	0x0000 - 0x07FF	0x0000 - 0x03FF

Note: The bootloader start address section depends on the fuse bits "BOOTSZ".
Refer to the datasheet for more details on Flash memories (Flash, EEPROM, ...) behaviors.

4.1 Physical Environment

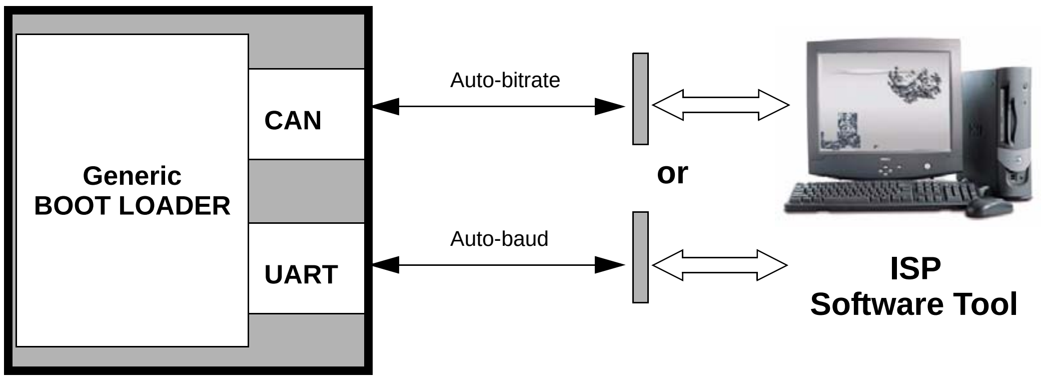
Bootloader deals with the host (or PC) through:

– A CAN interface

or

– An UART interface

Figure 4-1. Physical Environment



4.2 Bootloader Description

4.2.1 Overview

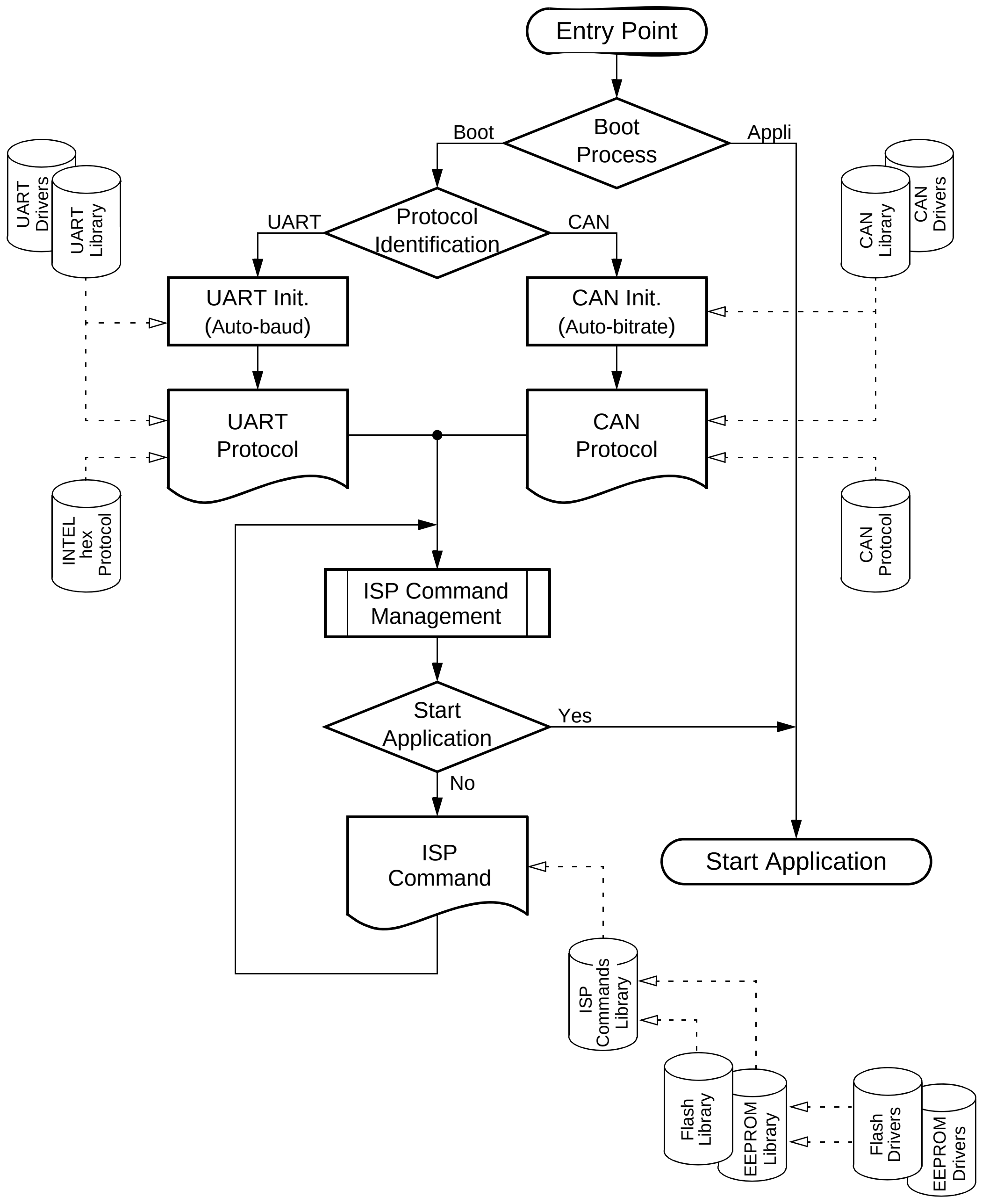


Figure 4-2. Bootloader Diagram

4.2.2 Entry Point

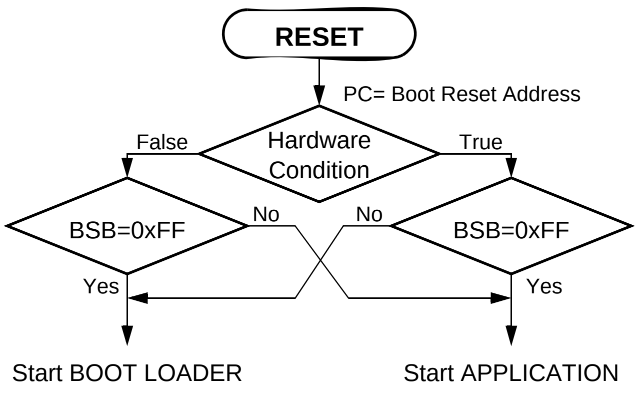
Only **one** "*Entry Point*" is available, it is the entry point to the bootloader. The "BOOTRST" fuse of the device have to be set. After Reset, the "Program Counter" of the device is set to "Boot Reset Address" (c.f. Table 4-1 "Device Memory Mapping (byte addressing)" on page 2). This "*Entry Point*" initializes the "*boot process*" of the bootloader.

4.2.3 Boot Process

The "*boot process*" of the bootloader allows to start the application or the bootloader itself. This depends on two variables:

- The "**Hardware Condition**".
  The Hardware Condition is defined by a device input PIN and its activation level (Ex: INT0/PIND.0, active low). This is set in "config.h" file.
- The "**Boot Status Byte**".
  The Boot Status Byte "**BSB**" belongs to the "Bootloader Configuration Memory" (c.f. Section 5.4.4.1 "Boot Status Byte - "BSB"" on page 9). Its default value is 0xFF. An ISP command allows to change its value.

Figure 4-3. Boot Process Diagram



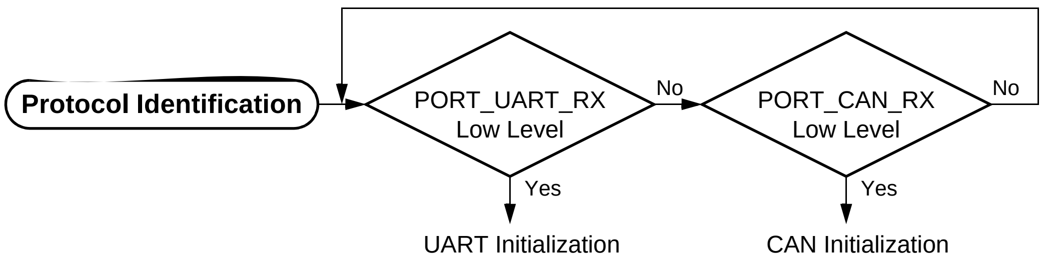
4.2.4 Protocol Identification

The "*Protocol Identification*" of the bootloader select what protocol to use, CAN or UART protocol. A polling of the physical lines is done to detect an activity on the media. These lines are:

- **PORT_CAN_RX**: The polling is be done on RXCAN/PIND.6.
- **PORT_UART_RX**: Depends on the definition set in "config.h" file.
  - If "**USE_UART1**" is defined, the polling is be done on RxD0/PINE.0.
  - If "**USE_UART2**" is defined, the polling is be done on RxD1/PIND.2.

The first low level on one of these lines starts the initialization of the corresponding peripheral.

Figure 4-4. Protocol Identification Diagram



4.2.5 CAN Initialization

The CAN, used to communicate with the host, has the following configuration:

– **Standard:** CAN format 2.0A (11-bit identifier).
– **Frame**: Data frame.
– **Bitrate**: Depends on Extra Byte - “**EB**” (see “Extra Byte - “EB”” on page 10):
- “**EB**” = 0xFFH: Use the software auto-bitrate.
- “**EB**” **!=** 0xFFH: Use CANBT[1..3] bytes to set the CAN bitrate (see “CANBT[1..3] - “BTC[1..3]“,” on page 10).

The initialization process must be performed after each device Reset. The host initiates the communication by sending a data frame to select a node. In case of auto-bitrate, this will help the bootloader to find the CAN bitrate. The CAN standard says that a frame having an acknowledge error is re-sent automatically. This feature and the capability of the CAN peripheral to be set in “LISTEN” mode are used by the auto-bitrate. Once the synchronization frame is received without any error, a recessive level is applied on the acknowledge slot by releasing the “LISTEN” mode.

The software auto-bitrate supports a wide range of baud rates according with the system clock (CKIO) set on the device (c.f. “**FOSC**“ definition in “config.h “ file). This functionality is not guaranteed on a CAN network with several CAN nodes.

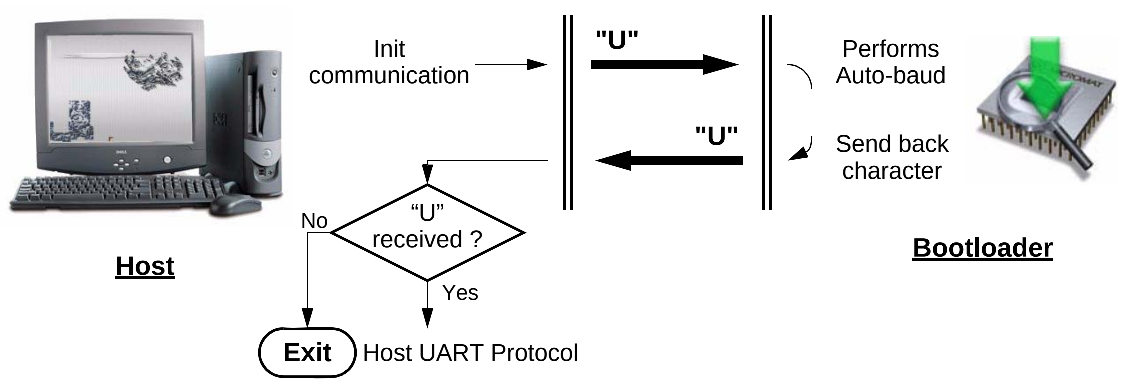
4.2.6 UART Initialization

The defined UART, used to communicate with the host, has the following configuration:

– **Character**: 8-bit of data.
– **Parity**: none.
– **Stop**: 1 bit.
– **Flow Control**: none.
– **Baud rate**: an auto-baud is performed to find the baud rate chosen by the host.

The initialization process must be performed after each device Reset. The host initiates the communication by sending a “**U**” character (0x55) as synchronization character to help the bootloader to find the baud rate (auto-baud). Only **one** synchronization character is sent and at the end of this character the bootloader must have its UART initialization done.
The bootloader supports a wide range of baud rates according with the system clock (CKIO) set on the device (c.f. “**FOSC**“ definition in “config.h “ file).

Figure 4-5. UART Synchronization



4.2.7 CAN or UART Protocols Overview

The “*CAN or UART Protocols*” are higher level protocols over serial line.

They are described in specific paragraphs in this document (See "CAN Protocol & ISP Commands" on page 12. & see "UART Protocol & ISP Commands" on page 19).

4.2.8 ISP Commands Overview

Each of "*CAN or UART Protocols*" decodes "*ISP commands*". The set of "*ISP commands*" obviously is independent of both protocols.

It is described in a specific paragraph in this document (See "CAN Protocol & ISP Commands" on page 12. & see "UART Protocol & ISP Commands" on page 19).

4.2.9 Output From Bootloader

The output from the bootloader is performs after receiving the ISP command: "*Start Application*" ((See "CAN Protocol & ISP Commands" on page 12. & see "UART Protocol & ISP Commands" on page 19).

5. Memory Space Definition

The bootloader supports up to five separate memory spaces. Each of them receives a code number (value to report in the corresponding protocol field) because low level access protocols (drivers) can be different.

The access to memory spaces is a byte access (i.e. the given addresses are byte addresses).

Table 5-1. Memory Space Code Numbers

Space [1]	**Code Number**	**Access**
Flash Memory (default)	**0**	Read & Write
EEPROM Data Memory	**1**	Read & Write
-	**2**	-
Bootloader Information	**3**	Read only
Bootloader Configuration	**4**	Read & Write
Device registers [2]	**5**	Read only
Signature	**6**	Read only

Note: 1. Sometimes, the discriminating is not physical (ex: "Signature" is a sub-set of the code of the bootloader Flash Section" as well as "Bootloader Information").
2. Not yet implemented.

5.1 Flash Memory Space

The Flash memory space managed by the bootloader is a sub-set of the device Flash. It is the "*Application Flash Section*".

Table 5-2. Flash Memory Space (Code Number **0**)

Flash Memory Space	**AT90CAN128**	**AT90CAN64**	**AT90CAN32**
Size	120 K bytes	56 K bytes	24 K bytes
Address Range	0x00000 - 0x1DFFF	0x00000 - 0xDFFF	0x00000 - 0x05FFF
Number of page(s)[1]	2	1	1

Note: 1. Page parameter is different in the bootloader and in the device itself.

5.1.1 Reading or Programming

The "*ISP Read*" or "*ISP Program*" commands only access to Flash memory space in byte addressing mode into a page of 64K bytes (c.f. Table 5-2 "Flash Memory Space (Code Number 0)" on page 6). Specific ISP commands allows to select the different pages.

The bootloader will return a "*Device protection*" error if the Software Security Byte "**SSB**" is set while read or write command occurs (c.f. Section 5.4.4.2 "Software Security Byte - "SSB"" on page 9).

5.1.2 Erasing

The "*ISP Erase*" command is a full erase (all bytes=0xFF) of the Flash memory space. This operation is available whatever the Software Security Byte "**SSB**" setting. A the end of the operation, the Software Security Byte "**SSB**" is reset to level 0 of security (Section 5.4.4.2 "Software Security Byte - "SSB"" on page 9).

5.1.3 Limits

The ISP commands on the Flash memory space has no effect on the bootloader (no effect on "*Bootloader Flash Section*").

The sizes of the Flash memory space (code number 0) for ISP commands are given in Table 5-2 "Flash Memory Space (Code Number 0)" on page 6.

5.2 EEPROM Data Memory

The EEPROM data memory space managed by the bootloader is the device EEPROM.

Table 5-3. EEPROM Data Memory Space (Code Number **1**)

EEPROM Data Memory Space	AT90CAN128	AT90CAN64	AT90CAN32
Size	4 K bytes	2 K bytes	1 K bytes
Address Range	0x0000 - 0x0FFF	0x0000 - 0x07FF	0x0000 - 0x03FF
Number of page(s)	-- No paging --		

5.2.1 Reading or Programming

The EEPROM data memory space is used as non-volatile data memory. The "*ISP Read*" or "*ISP Program*" commands access byte by byte to this space (no paging).
The bootloader will return a "*Device protection*" error if the Software Security Byte "**SSB**" is set while read or write command occurs (c.f. Section 5.4.4.2 "Software Security Byte - "SSB"" on page 9).

5.2.2 Erasing

The "*ISP Erase*" command is a full erase (all bytes=0xFF) of the EEPROM Data Memory space. This operation is available whatever only if the Software Security Byte "**SSB**" is reset (Section 5.4.4.2 "Software Security Byte - "SSB"" on page 9).

5.2.3 Limits

The sizes of the EEPROM Data Memory space (code number 1) for ISP commands are given in Table 5-3 "EEPROM Data Memory Space (Code Number 1)" on page 7.

5.3 Bootloader Information

The Boot loader information space managed by the bootloader is included the code of the bootloader. It is in the "*Bootloader Flash Section*".

Table 5-4. Bootloader Information Space (Code Number **3**)

Signature Space		AT90CAN128	AT90CAN64	AT90CAN32
Bootloader Revision	Address: 0x00 (Read only)	≥ 0x01		
Boot ID1	Address: 0x01 (Read only)	0xD1		
Boot ID2	Address: 0x02 (Read only)	0xD2		
Number of page(s)		-- No paging --		

5.3.1 Reading or Programming

The "*ISP Read*" command accesses byte by byte to this space (no paging).
No access protection is provided on this read only space.

5.3.2 Erasing

Not applicable for this read only space.

5.3.3 Limits

Details on the Boot loader information space (code number 3) for ISP commands are given in Table 5-4 "Bootloader Information Space (Code Number 3)" on page 8.

5.3.4 Bootloader Information Byte Description

5.3.4.1 Boot Revision

Boot Revision: Read only address =0x00, value ≥ 0x01.

5.3.4.2 Boot ID1 & ID2

Boot ID1 & ID2: Read only addresses = 0x01 & 0x02, value = 0xD1 & 0xD2.

5.4 Bootloader Configuration

The Boot loader configuration space managed by the bootloader is included in the "*Bootloader Flash Section*".

Table 5-5. Bootloader Configuration Space (Code Number **4**)

Signature Space			AT90CAN128	AT90CAN64	AT90CAN32
Boot Status Byte	"**BSB**"	Add.: 0x00	(default value=0xFF)		
Software Security Byte	"**SSB**"	Add.: 0x05	(default value=0xFF)		
Extra Byte	"**EB**"	Add.: 0x06	(default value=0xFF) (1)		
CANBT1	"**BTC1**"	Add.: 0x1C	(default value=0xFF) (2)		
CANBT2	"**BTC2**"	Add.: 0x1D	(default value=0xFF) (2)		
CANBT3	"**BTC3**"	Add.: 0x1E	(default value=0xFF) (2)		
Node Number	"**NNB**"	Add.: 0x1F	(default value=0xFF) (3)		
CAN Re-locatable ID Segment	"**CRIS**"	Add.: 0x20	(default value=0xFF)		
Number of page(s)			-- No paging --		

Note: 1. See "Extra Byte - "EB"" on page 10. for validity.

2. See “CANBT[1..3] - “BTC[1..3]“,” on page 10. for validity.
3. See “(CAN) Node Number - “NNB”” on page 10. for validity.

5.4.1 Reading or Programming

The “*ISP Read*” command accesses byte by byte to this space (no paging).

Access protection is only provided on the Software Security Byte (c.f. Section 5.4.4.2 ”Software Security Byte - “SSB”” on page 9).

5.4.2 Erasing

The “*ISP Erase*” command is **not available** for this space.

5.4.3 Limits

Details on the Boot loader configuration space (code number 6) for ISP commands are given in Table 5-5 ”Bootloader Configuration Space (Code Number 4)” on page 8.

5.4.4 Bootloader Configuration Byte Description

*5.4.4.1 Boot Status Byte - “**BSB**”*

The Boot Status Byte of the bootloader is used in the “*boot process*” (Section 4.2.3 ”Boot Process” on page 4) to control the starting of the application or the bootloader. If no Hardware Condition is set, the default value (0xFF) of the Boot Status Byte will force the bootloader to start. Else (Boot Status Byte != 0xFF & no Hardware Condition) the application will start.

*5.4.4.2 Software Security Byte - “**SSB**”*

The bootloader has the Software Security Byte “**SSB**” to protect itself and the application from user access or ISP access. It protects the Flash and EEPROM memory spaces and itself.

The “I*SP Program*” command on Software Security Byte “**SSB**” can only write an higher priority level. There are three levels of security:

Table 5-6. Security levels

Level	Security	“SSB”	Comment
0	**NO_SECURITY**	0xFF	- This is the default level. - Only level 1 or level 2 can be written over level 0.
1	**WR_SECURITY**	0xFE	- In level 1, it is impossible to write in the Flash and EEPROM memory spaces. - The bootloader returns an error message. - Only level 2 can be written over level 0.
2	**RD_WR_SECURITY**	≤ 0xFC	- All read and write accesses to/from the Flash and EEPROM memory spaces are not allowed. - The bootloader returns an error message. - Only an “*ISP Erase*” command on the Flash memory space resets (level 0) the Software Security Byte.

The table below gives the authorized actions regarding the SSB level.

Table 5-7. Allowed actions regarding the Software Security Byte “**SSB**”

ISP Command	NO_SECURITY	WR_SECURITY	RD_WR_SECURITY
Erase **Flash** memory space	Allow	Allow	Allow
Erase **EEPROM** memory space	Allow	-	-
Write **Flash** memory space	Allow	-	-
Write **EEPROM** memory space	Allow	-	-
Read **Flash** memory space	Allow	Allow	-
Read **EEPROM** memory space	Allow	Allow	-
Write byte(s) in **Boot loader configuration** (except for “**SSB**”)	Allow	-	-
Read byte(s) in **Boot loader configuration**	Allow	Allow	Allow
Write “**SSB**”	Allow	only a higher level	-
Read **Boot loader information**	Allow	Allow	Allow
Read **Signature**	Allow	Allow	Allow
Blank check (any memory)	Allow	Allow	Allow
Changing of memory space	Allow	Allow	Allow

5.4.4.3 *Extra Byte - “**EB**”*

The Extra Byte is used to switch the CAN Initialization to auto-bitrate or to fixed CAN bit timing.

– “**EB**” = 0xFFH: Use the software auto-bitrate.

– “**EB**” **!=** 0xFFH: Use CANBT[1..3] bytes of Boot loader configuration space to set the CAN bit timing registers of the CAN peripheral (no auto-bitrate).

Note: Not yet exploited. This will be done in a future bootloader version.

5.4.4.4 *CANBT[1..3] - “**BTC[1..3]**“,*

When “**EB**” **!=** 0xFFH, CANBT[1..3] bytes of Boot loader configuration space are used to set the CAN bit timing registers of the CAN peripheral.(no auto-bitrate).
An other way to write these byte is described in Section 5.5.4.1 ”CANBT[1..3] Registers.” on page 11.

Note: Not yet exploited. This will be done in a future bootloader version.

5.4.4.5 *(CAN) Node Number - “**NNB**”*

See “CAN Protocol & ISP Commands” on page 12.

Note: Not yet exploited. This will be done in a future bootloader version.

5.4.4.6 *CAN Re-locatable ID Segment - “**CRIS**”*

See “CAN Protocol & ISP Commands” on page 12.

5.5 Device Registers

The device registers space managed by the bootloader is the 64 I/O registers and the 160 Ext. I/O registers of the device. They are accessed by the equivalent assembler instruction:

LDS Rxx, REG_ADD

where **REG_ADD** is in the address range 0x20 (**PINA**) up to 0xFA (**CANMSG**).

5.5.1 Reading or Programming

The "*ISP Read*" command accesses byte by byte to this space (no paging).

No access protection is provided on this read only space.

5.5.2 Erasing

Not applicable for this read only space.

5.5.3 Limits

This space is not bit addressing and an unimplemented register returns 0xFF.

5.5.4 Device Registers Description

c.f. appropriate data sheet for information.

5.5.4.1 CANBT[1..3] Registers.

If they are read before to disable the auto-bitrate (when "**EB**" **=** 0xFFH), in the same time they they are copied into "**BTC1**", "**BTC2**" & "**BTC3**" of the Boot loader configuration space (see "CANBT[1..3] - "BTC[1..3]"," on page 10).

Note: Not yet exploited. This will be done in a future bootloader version.

5.6 Signature

The Signature space managed by the bootloader is included the code of the bootloader. It is in the "*Bootloader Flash Section*".

Table 5-8. Signature Space (Code Number **6**)

Signature Space		AT90CAN128	AT90CAN64	AT90CAN32
Manufacturer Code	Address: 0x30 (Read only)	0x1E		
Family Code	Address: 0x31 (Read only)	0x81		
Product Name	Address: 0x60 (Read only)	0x97	0x96	0x95
Product Revision	Address: 0x61 (Read only)	≥ 0x00	≥ 0x00	≥ 0x00
Number of page(s)		-- No paging --		

5.6.1 Reading or Programming

The "*ISP Read*" command accesses byte by byte to this space (no paging).

No access protection is provided on this read only space.

5.6.2 Erasing

Not applicable for read only space.

5.6.3 Limits

Details on the Signature space (code number 6) for ISP commands are given in Table 5-8 "Signature Space (Code Number 6)" on page 11.

6. CAN Protocol & ISP Commands

This section describes the higher level protocol over the CAN network communication and the coding of the associated ISP commands.

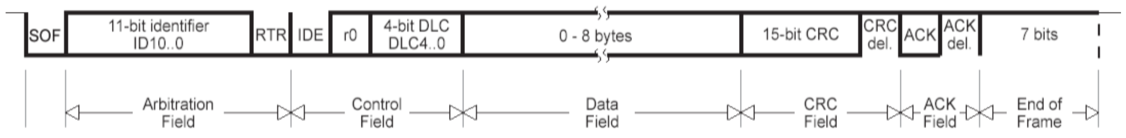
6.1 CAN Frame Description

The CAN protocol only supports the CAN standard frame (c.f. ISO 11898 for high speed and ISO 11519-2 for low speed) also known as CAN 2.0 A with 11-bit identifier.

A message in the CAN standard frame format begins with the "Start Of Frame (SOF)", this is followed by the "Arbitration field" which consist of the identifier and the "Remote Transmission Request (RTR)" bit used to distinguish between the data frame and the data request frame called remote frame. The following "Control field" contains the "IDentifier Extension (IDE)" bit and the "Data Length Code (DLC)" used to indicate the number of following data bytes in the "Data field". In a remote frame, the DLC contains the number of requested data bytes. The "Data field" that follows can hold up to 8 data bytes. The frame integrity is guaranteed by the following "Cyclic Redundant Check (CRC)" sum. The "ACKnowledge (ACK) field" compromises the ACK slot and the ACK delimiter. The bit in the ACK slot is sent as a recessive bit and is overwritten as a dominant bit by the receivers which have at this time received the data correctly.

The ISP CAN protocol only uses CAN standard data frame.

Figure 6-1. CAN Standard Data Frame



To describe the ISP CAN protocol, a symbolic name is used for Identifier, but default values are given within the following presentation.

Table 6-1. Template for ISP CAN command

Identifier 11 bits	**Length** 4 bits	**Data[0]** 1 byte	**...**	**Data[n-1]** 1 byte	**Description**
SYMBOLIC_NAME (“**CRIS**”<<4) + **X**	n (≤8)	Value or meaning			Command description

Because in a point-to-point connection, the transmit CAN message is repeated until a hardware acknowledge is done by the receiver.

The bootloader can acknowledge an incoming CAN frame only if a configuration is found.

This functionality is not guaranteed on a network with several CAN nodes.

6.2 CAN ISP Command Data Stream Protocol

6.2.1 CAN ISP Command Description

Several CAN message identifiers are defined to manage this protocol.

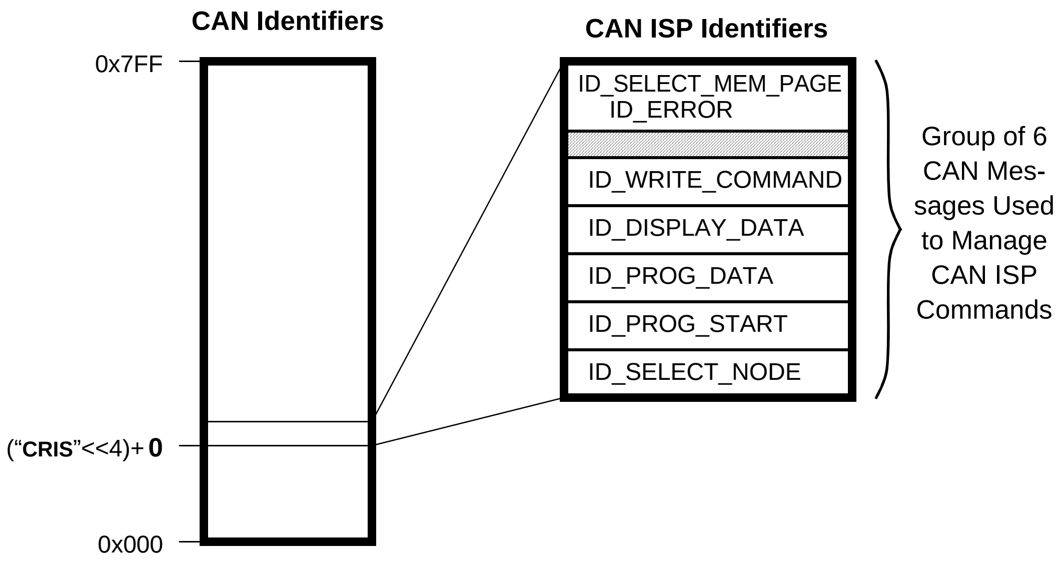
Table 6-2. Defined CAN Message Identifiers for CAN ISP Protocol

Identifier	ISP Command Detail	Value
ID_SELECT_NODE	Open/Close a communication with a node	(“**CRIS**” << 4) + **0**
ID_PROG_START	Start Memory space programming	(“**CRIS**” << 4) + **1**
ID_PROG_DATA	Data for Memory space programming	(“**CRIS**” << 4) + **2**
ID_DISPLAY_DATA	Read data from Memory space	(“**CRIS**” << 4) + **3**
ID_START_APPLI	Start application	(“**CRIS**” << 4) + **4**
ID_SELECT_MEM_PAGE	Selection of Memory space or page	(“**CRIS**” << 4) + **6**
ID_ERROR	Error message from bootloader only	

It is possible to allocate a new value for CAN ISP identifiers by writing the “**CRIS**” byte with the base value for the group of identifier.

The maximum “**CRIS**” value is 0x7F and its the default value is 0x00.

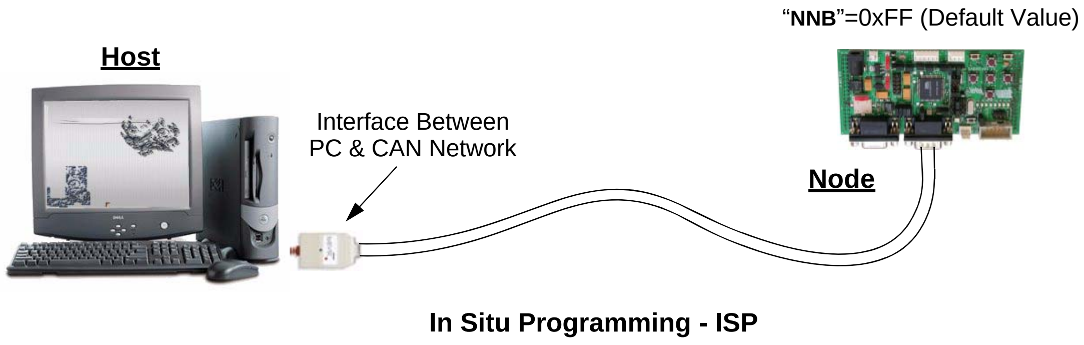
Figure 6-2. Remapping of CAN Message Identifiers for CAN ISP Protocol



6.2.2 Communication Initialization

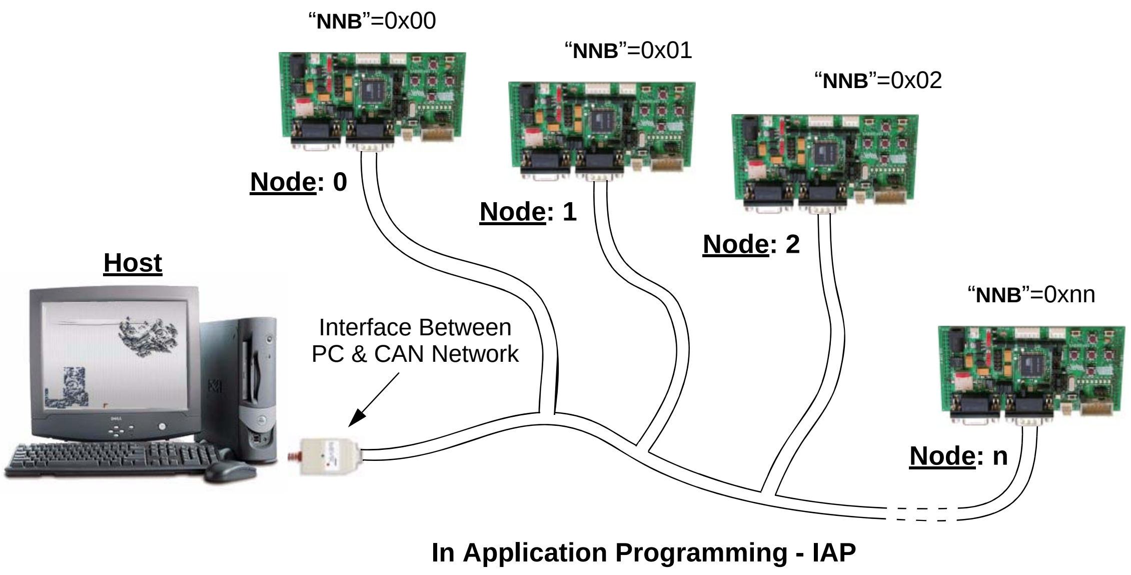
The communication with a device (CAN node) must be opened prior to initiate any ISP communication. To open communication with the device, the Host sends a “Connecting” CAN message (“*ID_SELECT_NODE*”) with the node number “**NNB**” passed as parameter. If the node number passed is 0xFF then the CAN bootloader accepts the communication (Figure 6-3). Otherwise the node number passed in parameter must be equal to the local "**NNB**" (Figure 6-4).

Figure 6-3. CAN Bootloader First Connection



"NNB"=0xFF (Default Value)
Host
Interface Between PC & CAN Network
Node
In Situ Programming - ISP

Figure 6-4. CAN Bootloader Network Connection



Before opening a new communication with another device, the current device communication must be closed with its connecting CAN message ("*ID_SELECT_NODE*").

6.3 CAN ISP Commands

6.3.1 CAN Node Select

A CAN node must be first **opened** at the beginning and then **closed** at the end of the session.

6.3.1.1 CAN Node Select Requests from Host

Table 6-3. CAN Node Select Requests from Host

Identifier	L	Data[0]	Description
ID_SELECT_NODE ((“**CRIS**”<<4)+ **0**)	1	Node Number (“**NNB**”)	**Open** or **close** communication with a specific node

6.3.1.2 CAN Node Select Answers from Bootloader

Table 6-4. CAN Node Select Answers from Bootloader

Identifier	L	Data[0]	Data[1]	Description
ID_SELECT_NODE ((“**CRIS**”<<4)+ **0**)	2	“*Bootloader Revision*”	0x00	Communication **closed**
			0x01	Communication **opened**

6.3.2 Changing Memory / Page

To change of memory space and/or of page, there is only one command, the switch is made by “*Data[0]”* of the CAN frame.

6.3.2.1 Changing Memory / Page Requests from Host

Table 6-5. Changing Memory / Page Requests from Host

Identifier	L	Data[0]	Data[1]	Data[2]	Description
ID_SELECT_MEM_PAGE ((“**CRIS**”<<4)+ **6**)	3	0x00	Memory space	Page	No action
		0x01			Select Memory space
		0x02			Select Page
		0x03			Select Memory space & Page

6.3.2.2 Changing Memory / Page Answers from Bootloader

Table 6-6. Changing Memory / Page Answers from Bootloader

Identifier	L	Data[0]	Description
ID_SELECT_MEM_PAGE ((“**CRIS**”<<4)+ **6**)	1	0x00	Selection OK (even if “*Data[0]”*=0 in the request frame)

6.3.3 Reading / Blank Checking Memory

These operations can be executed only with a device previously open in communication. This command is available on the memory space and on the page previously defined.

To start the reading or blank checking operation, the Host sends a CAN message ("*ID_DISPLAY_DATA*") with the operation required in Data[0], the start address and end address are passed as parameters.

6.3.3.1 *Reading / Blank Checking Memory Requests from Host*

Table 6-7. Reading / Blank Checking Memory Requests from Host

Identifier	**L**	**Data[0]**	**Data[1]**	**Data[2]**	**Data[3]**	**Data[4]**	**Description**
ID_DISPLAY_DATA (("**CRIS**"<<4)+ **3**)	5	0x00	Start Address (MSB, LSB)		End Address (MSB, LSB)		Display data of selected Memory space / Page
		0x80					Blank check on selected Memory space / Page

6.3.3.2 *Reading / Blank Checking Memory Answers from Bootloader*

Table 6-8. Reading / Blank Checking Memory Answers from Bootloader

Identifier	**L**	**Data[0]**	**Data[1]**	**...**	**Data[7]**	**Description**
ID_DISPLAY_DATA (("**CRIS**"<<4)+ **3**)	up to 8	Up to 8 Data Bytes				Data Read
	0	-	-	-	-	Blank check OK
	2	First not blank address		-	-	Error on Blank check
ID_ERROR (("**CRIS**"<<4)+ **6**)	1	0x00	-	-	-	Error Software Security Set ("*Display data*" only)

6.3.4 Programming / Erasing Memory

These operations can be executed only with a device previously open in communication. They need two steps:

- The first step is to indicate address range for program or erase command.
- The second step is to transmit the data for programming only.

To start the programming operation, the Host sends a "start programming" CAN message (ID_PROG_START) with the operation required in "*Data[0]*", the start address and the end address are passed as parameters.

6.3.4.1 *Programming / Erasing Memory Requests from Host*

Table 6-9. Unit. Programming / Erasing Memory Requests from Host

Identifier	**L**	**Data[0]**	**Data[1]**	**Data[2]**	**Data[3]**	**Data[4]**	**Data[5..7]**	**Description**
ID_PROG_START (("**CRIS**"<<4)+ **1**)	5	0x00	Start Address (MSB, LSB)		End Address (MSB, LSB)		-	Init. prog. the selected Memory space / Page
	3	0x80	0xFF	0xFF	-	-	-	Erase the selected Memory space / Page
ID_PROG_DATA (("**CRIS**"<<4)+ **2**)	n	data[0..(n-1)] (n≤8)						Data to program

6.3.4.2 *Programming / Erasing Memory Answers from Bootloader*

Table 6-10. Programming / Erasing Memory Answers from Bootloader

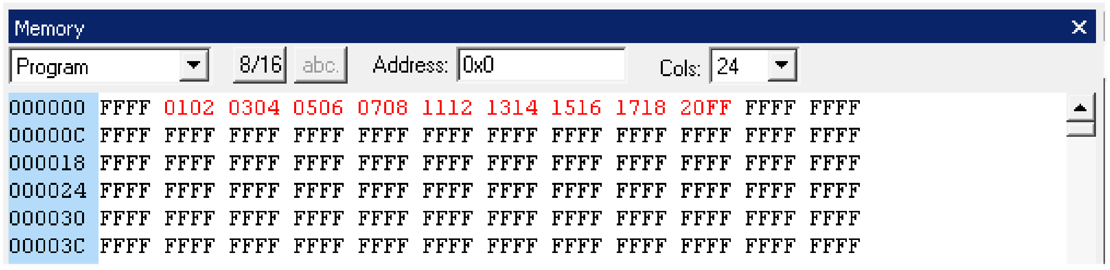
Identifier	L	Data[0]	Description
ID_PROG_START ((“**CRIS**”<<4)+ **1**)	0	-	Init. prog. command OK
	1	0x00	Erase done
ID_PROG_DATA ((“**CRIS**”<<4)+ **2**)	1	0x00	Command OK and end of transfer
		0x02	Command OK but new (other) data expected
ID_ERROR ((“**CRIS**”<<4)+ **6**)	1	0x00	Error - Software Security Set (“*Init. program*” only)

6.3.4.3 *Programming Memory Examples*

Table 6-11. Programming Memory Examples

Request/ Answer	CAN Message (hexadecimal)			Description
	Identifier	L	Data[..70]	
R (>>)	000	1	FF	CAN Node Select
A (<<)	000	2	01 01	Communication opened
Default Memory space = Flash, default Page = page_0				
R (>>)	001	5	00 00 02 00 12	Init. Prog. Add 0x0002 up to 0x0012
A (<<)	001	0	00	Init. prog. command OK
R (>>)	002	8	01 02 03 04 05 06 07 08	1st Data transfer
A (<<)	002	1	02	Command OK, new data expected
R (>>)	002	8	11 12 13 14 15 16 17 18	2nd Data transfer
A (<<)	002	1	02	Command OK, new data expected
R (>>)	002	1	20	3rd Data transfer
A (<<)	002	1	00	Command OK, end of transfer

Figure 6-5. Result of the Above Programming Memory Example [(1)]



```
000000 FFFF 0102 0304 0506 0708 1112 1314 1516 1718 20FF FFFF FFFF
00000C FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
000018 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
000024 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
000030 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
00003C FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
```

Note: 1. AVR Studio Program Memory representation

6.3.5 Starting Application

This operation can be executed only with a device previously open in communication.

To start the application, the host sends a start application CAN message with the “way of” selected in “*Data[1]*”. The application can be start by a watchdog reset or by jumping to address 0x0000 in the Flash memory.

No answer is returned by the bootloader.

Table 6-12. Start application Requests from Host

Identifier	L	Data[0]	Data[1]	Data[2]	Data[3]	Description
ID_START_APPLI ((“**CRIS**”<<4)+ **4**)	2	0x03	0x00	**-**	**-**	Start application with watchdog reset
	4		0x80	0x00	0x00	Jump to address 0x0000

7. UART Protocol & ISP Commands

This section describes the higher level protocol over the UART serial line and the coding of the associated ISP commands.

7.1 UART Frame Description

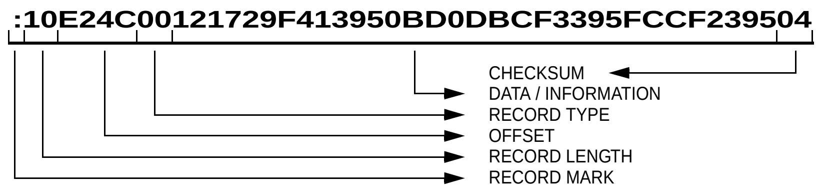
The UART Protocol is based on the Intel Extended Hex-type records.

Each record begins with a RECORD MARK field containing 03AH, the ASCII code for the colon (’ **:** ’) character. Each record has a RECORD LENGTH field which specifies the number of bytes of data or information which follows the RECORD TYPE field of the record. Note that one data byte is represented by two ASCII characters. The maximum value of the RECORD LENGTH field is 0xFF’ or 255.

Table 7-1. Intel Hex Type Frame

RECORD MARK ‘:’	RECORD LENGTH	OFFSET	RECORD TYPE	DATA / INFORMATION	CHECKSUM
1 byte	1 byte	2 bytes	1 byte	n byte(s)	1 byte

Example:



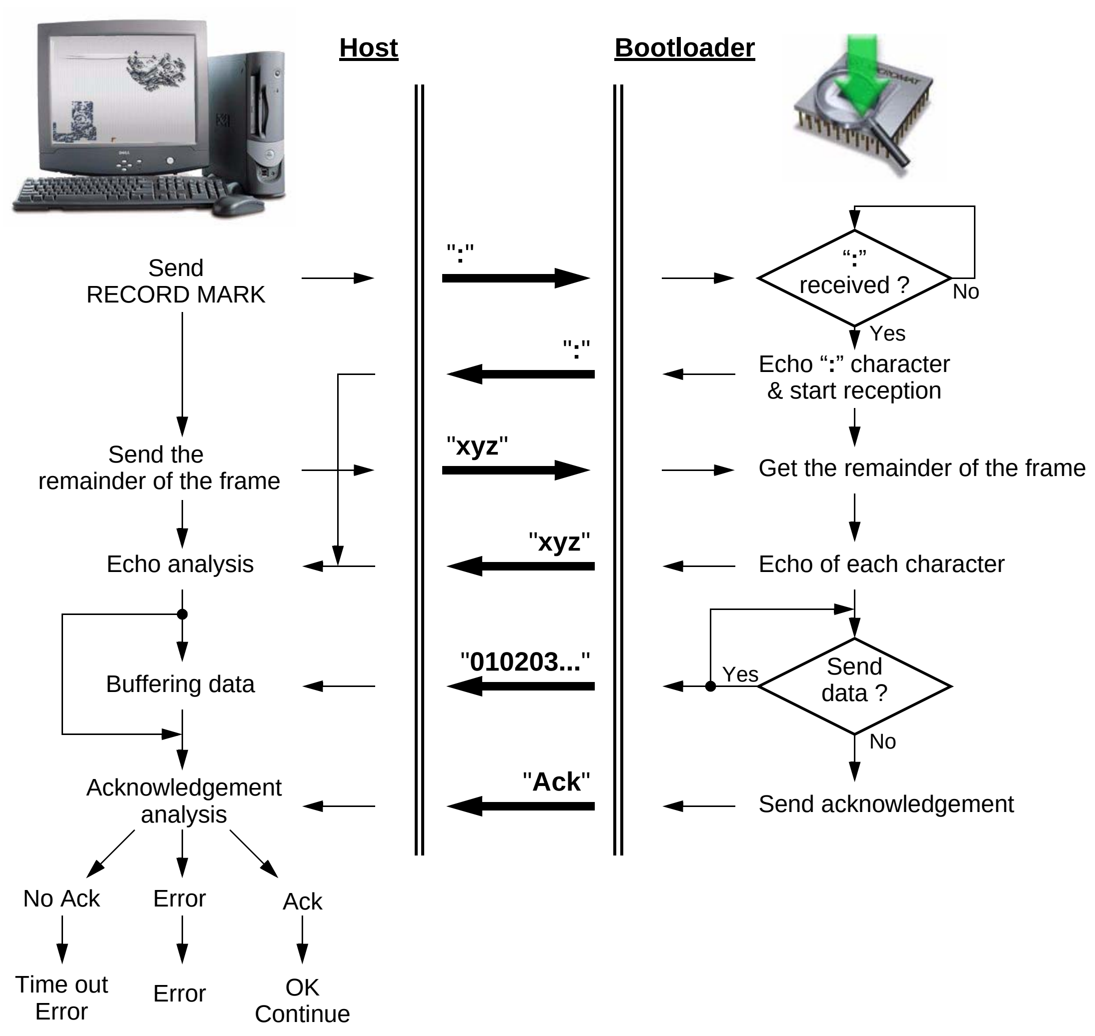
- **RECORD MARK** (1 ASCCI byte)
  This field contains 0x03, the hexadecimal encoding of the ASCII colon (’ **:** ’) character.
- **RECORD LENGTH** (1 byte once ASCII decoded)
  This field specifies the number of bytes of DATA/INFORMATION field which follows the RECORD TYPE field.
- **OFFSET** (2 bytes once ASCII decoded)
  Each record has an OFFSET field which specifies the 16-bit starting load offset of the data bytes, therefore this field is only used for Data Records. In other records where this field is not used, it should be coded as four ASCII zero characters (’**0000**’ or 0x030303030).
- **RECORD TYPE** (1 byte once ASCII decoded)
  Each record has a RECORD TYPE field which specifies the record type of this record. The RECORD TYPE field is used to interpret the remaining information within the record.
- **DATA / INFORMATION** (n byte(s) once ASCII decoded)
  Each record has a variable (RECORD LENGTH) length DATA/INFORMATION field. It consists of zero or more bytes encoded as pairs of hexadecimal digits. The meaning of data depends on the RECORD TYPE.
- **CHECKSUM** (1 byte once ASCII decoded)
  This field contains the checksum (two’s complement) on the RECORD LENGTH, OFFSET, RECORD TYPE and DATA/INFORMATION fields ASCII decoded.

7.2 UART ISP Command Data Stream Protocol

All ISP commands are sent using the same flow. Each frame sent by the host first must be echoed by the bootloader.
Each command flow may end with:

- "**X**" : If checksum error
- "**L**" : If read security is set
- "**P**": If program security is set
- "**.**": If command OK (or byte + "**.**": If read byte OK)

Figure 7-1. Command Flow Summary



7.3 UART ISP Commands

7.3.1 Changing Memory / Page

To change of memory space and/or of page, there are two commands.

- **Select Memory**: To select the memory space and the page in this memory space.
- **Select New Page**: To change the page in the memory space already selected.

7.3.1.1 *Changing Memory / Page Requests from Host*

Table 7-2. Changing Memory / Page Requests from Host

ISP Command Request (R)		RECORD LENGTH	OFFSET	RECORD TYPE	DATA[0]	DATA[1]
Select New Page	":"	0x02	Start Address	0x0**2**	- [7..4] =Page - [3..0]=0x0	0x00
Select Memory	":"	0x02	0x0000	0x0**4**	Memory Space Code Number	Page

7.3.1.2 *Changing Memory / Page* Answers from Bootloader

Table 7-3. Changing Memory / Page Answers from Bootloader

Answer (A)	Character[0]	Character[1]	Character[2]
Command Done	"**.**"	"**C_R**"	"**L_F**"
Wrong Checksum	"**X**"	"**C_R**"	"**L_F**"

7.3.1.3 *Changing Memory / Page Examples*

Table 7-4. Changing Memory / Page Examples

ISP Command	R/A	Frame	Comment
Select Memory	R	>> "**:020000040100F9**"	Select EEPROM, Page 0
	A	<< "**:020000040100F9.$C_R$$L_F$**"	Command Done
Select New Page	R	>> "**:02234502100084**"	Select Page 1, Add. 0x2345
	A	<< "**:02234502100084.$C_R$$L_F$**"	Command Done
Select Memory	R	>> "**:020000040001F8**"	Select Flash, Page 1
	A	<< "**:020000040000F8X$C_R$$L_F$**"	Checksum error

Note: 1. Because the page size is 64K bytes, the physical address is 0x012345.

7.3.2 Reading / Blank Checking Memory

- The "ISP **Read Memory**" command allows to read an address range of a memory space.
- The "ISP **Blank Check Memory**" command allows to blank check an address range of a memory space.

The two commands are available on the memory space and on the page previously defined.

7.3.2.1 *Reading / Blank Checking Memory Requests from Host*

Table 7-5. Reading / Blank Checking Memory Requests from Host

ISP Command Request (R)		RECORD LENGTH	OFFSET	RECORD TYPE	DATA[0,1] 2 bytes	DATA[2,3] 2 bytes	DATA[4]
Read Memory	":"	0x05	0x0000	0x0**4**	Start Address	End Address	0x0**0**
Blank Check Memory	":"	0x05	0x0000	0x0**4**	Start Address	End Address	0x0**1**

7.3.2.2 *Reading / Blank Checking Memory Answers from Bootloader*

Table 7-6. Reading / Blank Checking Memory Answers from Bootloader

Answer (A)	Character[0..n]
Read Memory Command Done	Address = data (16-byte formatted) + "C_R" + "L_F"

Answer (A)	Character[0]	Character[1]	Character[2]	Character[3]
Blank Check Memory OK	"."	"C_R"	"L_F"	-
Blank Check Memory Error	First Failed Address		"C_R"	"L_F"
Wrong Checksum	"X"	"C_R"	"L_F"	-
Error - Software Security Set ("*ISP Read Memory*" only)	"L"	"C_R"	"L_F"	-

7.3.2.3 *Reading / Blank Checking Memory Examples*

Table 7-7. Reading / Blank Checking Memory Examples

ISP Command	R/A	Frame	Comment
Read Memory	R	>> "**:050000040003001500DF**"	Read the selected memory & page from Add. 0x0003 up to 0x0015
	A	<< "**:050000040003001500DFC_RL_F**" << "**0003=030405 .... 0F101112C_RL_F**" << "**0013=131415C_RL_F**"	Read data & command done
Blank Check Memory	R	>> "**:0500000400000100 01F5**"	Blank check the selected memory & page from Add. 0x0000 up to 0x0100
	A	<< "**:050000040000010001F5.C_RL_F**"	Blank Check Memory OK
Read Memory	R	>> "**:050000040010001100D6**"	Read the selected memory & page from Add. 0x0010 up to 0x0011
	A	<< "**:050000040010001100D6LC_RL_F**"	Security Set, read aborted
Blank Check Memory	R	>> "**:05000004600060020134**"	Blank check the selected memory & page from Add. 0x6000 up to 0x6002
	A	<< "**:050000046000600201346000C_RL_F**"	Blank check failed at Add. 0x6000
Blank Check Memory	R	>> "**:0500000400005FFF0123**"	Blank check the selected memory & page from Add. 0x0000 up to 0x5FFF
	A	<< "**:0500000400005FFF0123XC_RL_F**"	Checksum error

7.3.3 Programming / Erasing Memory

- The "ISP **Program Memory**" command allows to program an address range of a memory space. This command is available on the memory space and on the page previously defined.
- The "ISP **Erase Memory**" command allows to (full) erase a memory space. This command is available on the memory space previously defined.

7.3.3.1 *Programming / Erasing Memory Requests from Host*

Table 7-8. Programming / Erasing Memory Requests from Host

ISP Command Request (R)		RECORD LENGTH	OFFSET	RECORD TYPE	DATA field
Program Memory	“:”	n	First Address	0x0**0**	n data
Erase Memory	“:”	0x05	0x0000	0x0**4**	0x00, 0xFF, 0x00, 0x00, 0x02

7.3.3.2 *Programming / Erasing Memory Answers from Bootloader*

Table 7-9. Programming / Erasing Memory Answers from Bootloader

Answer (A)	Character[0]	Character[1]	Character[2]
Command Done	“**.**”	“C_R”	“L_F”
Wrong Checksum	“**X**”	“C_R”	“L_F”
Error - Software Security Set (“*ISP Program Memory*” only)	“**P**”	“C_R”	“L_F”

7.3.3.3 *Programming / Erasing Memory Examples*

Table 7-10. Programming / Erasing Memory Examples

ISP Command	R/A	Frame	Comment
Program Memory	R	>> ”**:020000001234B8**”	Program in the selected memory & page Add. 0x0000=0x12 & 0x0001=0x34
	A	<< ”**:020000001234B8.C_RL_F**”	Command Done
Erase Memory	R	>> ”**:0500000400FF000002F6**”	Erase selected memory
	A	<< ”**:0500000400FF000002F6.C_RL_F**”	Command Done
Program Memory	R	>> ”**:02000200567821**”	Program in the selected memory & page Add. 0x0002=0x56& 0x0003=0x78
	A	<< ”**:02000200567821PC_RL_F**”	Security Set, program aborted.
Erase Memory	R	>> ”**:0500000400FF000002F0**”	Erase the selected memory
	A	<< ”**:0500000400FF000002F0XC_RL_F**”	Checksum error

7.3.4 Starting Application

The Host sends a start application message generating a jump to address 0x0000 in the Flash memory.

No answer is returned by the bootloader.

Table 7-11. Start application Requests from Host

ISP Command Request (R)		RECORD LENGTH	OFFSET	RECORD TYPE	DATA field
Start Application	“:”	0x00	0x0000	0x0**1**	no data

8. Appendix A: #define in “config.h” file

8.1 Processor Definitions

```
// Global
#define AVR
#define AT90CAN128  1
#define AT90CAN64   2
#define AT90CAN32   3

// Hardware condition (for boot or application start)
  // INT on DVK90CAN1 board = INT0 or PD.0 - active low with pull-up
    #define PIN_HWCB      PIND_Bit0
    #define PORT_HWCB     PORTD_Bit0
    #define LEVEL_HWCB    0           // active at "0" or "1"
    #define PULLUP_HWCB   1           // pull-up "ON"="1", "OFF"="0"
/*  // Center Key on DVK90CAN1 board = PE.2 active low with pull-up
    #define PIN_HWCB      PINE_Bit2
    #define PORT_HWCB     PORTE_Bit2
    #define LEVEL_HWCB    0           // active at "0" or "1"
    #define PULLUP_HWCB   1           // pull-up "ON"="1", "OFF"="0" */
// Application
#define USE_DEVICE   AT90CAN128
#define USE_UART1
#define FOSC         8000

// Switches for Specific definitions
#ifndef USE_DEVICE
#       error You must define USE_DEVICE AT90CAN128, AT90CAN64 or AT90CAN32 first in
"config.h" file
#   elif USE_DEVICE == AT90CAN128
#       define MANUF_ID         0x1E        // ATMEL
#       define FAMILY_CODE      0x97        // 128 Kbytes of Flash
#       define PRODUCT_NAME     0x81        // AT90CAN family
#       define PRODUCT_REV      0x00        // rev 0
#       define FLASH_SIZE       0x1FFFF     // in bytes
#       define FLASH_PAGE_SIZE  0x100       // in bytes
#       define BOOT_SIZE        0x2000      // in bytes
#       define EEPROM_SIZE      0x1000      // in bytes
#   elif USE_DEVICE == AT90CAN64
#       define MANUF_ID         0x1E        // ATMEL
#       define FAMILY_CODE      0x96        // 64 Kbytes of Flash
#       define PRODUCT_NAME     0x81        // AT90CAN family
#       define PRODUCT_REV      0x00        // rev 0
#       define FLASH_SIZE       0x0FFFF     // in bytes
#       define FLASH_PAGE_SIZE  0x100       // in bytes
#       define BOOT_SIZE        0x2000      // in bytes
#       define EEPROM_SIZE      0x0800      // in bytes
#   elif USE_DEVICE == AT90CAN32
#       define MANUF_ID         0x1E        // ATMEL
#       define FAMILY_CODE      0x95        // 32 Kbytes of Flash
#       define PRODUCT_NAME     0x81        // AT90CAN family
#       define PRODUCT_REV      0x00        // rev 0
#       define FLASH_SIZE       0x07FFF     // in bytes
#       define FLASH_PAGE_SIZE  0x100       // in bytes
#       define BOOT_SIZE        0x2000      // in bytes
#       define EEPROM_SIZE      0x0400      // in bytes
#   else
#       error USE_DEVICE definition is not referenced in "config.h" file
#endif

#ifndef USE_UART1
#       ifndef USE_UART2
#               error You must define either USE_UART1 or USE_UART2 in
"config.h" file
#       endif
#endif

// Polling pins definition
#ifdef USE_UART1
#   define PIN_UART_RX     PINE_Bit0        // for UART0
#   define PORT_UART_TX    PORTE_Bit1       // for UART0
#endif
#ifdef USE_UART2
#   define PIN_UART_RX     PIND_Bit2        // for UART1
#   define PORT_UART_TX    PORTD_Bit3       // for UART1
#endif

#define PIN_CAN_RX      PIND_Bit6
#define PORT_CAN_TX     PORTD_Bit5
```

8.2 UART Definitions

```
//-------------- UART LIB CONFIGURATION ---------------
#define UART_AUTOBAUD_EXTERNAL_DETECTION
#define UART_MINIMUM
#define BDR_GENERATOR BRG_TIMER1
#define BAUDRATE    AUTOBAUD
//#define BAUDRATE    19200
#define test_hit()  uart_test_hit()
#define _getkey()   uart_getchar()
#define putchar     uart_putchar
```

8.3 Bootloader Definitions

```
//-------------- BOOTLOADER CONFIGURATION -------------
// Uart protocol
#define PROTOCOL_DATA                  64
#define GLOBAL_BUFFER_SIZE             PROTOCOL_DATA+4
#define NB_BYTE_MAX_FOR_DISPLAY_COMMAND 64
#define HEX_SIZE_DISP_PAGE             16

#define USE_RCS_HEX_PROTOCOL
#define USE_RCS_CAN_PROTOCOL

//----------- Bootloader identification definition ----
#define BOOT_VERSION    0x01 // @00  // Ver 01: JT-18.10.05
#define BOOT_ID1        0xD1 // @01
#define BOOT_ID2        0xD2 // @02

#define MAX_OFFSET_ID   0x7F0

#define NO_SECURITY     0xFF
#define RD_WR_SECURITY  0xFC
#define BSB_DEFAULT     0xFF
#define SSB_DEFAULT     0xFF
#define EB_DEFAULT      0xFF
#define NNB_DEFAULT     0xFF
#define CRIS_DEFAULT    0xFF   // if (offset_id_copy>MAX_OFFSET_ID) offset_id_copy=0;
#define BTC1_DEFAULT    0xFF
#define BTC2_DEFAULT    0xFF
#define BTC3_DEFAULT    0xFF

#define SSB_RD_PROTECTION 0xFC
#define SSB_WR_PROTECTION 0xFE
```

8.4 Memory Definitions

```
//-------- Memory Definition -----------------
#define MEM_USER               0
#define MEM_CODE               0
#define MEM_FLASH              0
#define MEM_EEPROM             1
#define MEM_CUSTOM             2
#define MEM_BOOT               3   // Boot information
#define MEM_XAF                4   // Boot configuration
#define MEM_HW_REG             5
#define MEM_SIGNATURE          6

#define MEM_DEFAULT MEM_FLASH

#define PAGE_DEFAULT  0x00
```

9. Appendix B: CAN Protocol Summary

Table 9-1. CAN Protocol Summary - Requests from Host

ISP Command Request Identifier	L	Data [0]	Data [1]	Data [2]	Data [3]	Data [4]	Data [5]	Data [6]	Data [7]	Description
ID_SELECT_NODE ((“**CRIS**”<<4)+ **0**)	1	Node	-	-	-	-	-	-	-	Open or close communication
ID_PROG_START ((“**CRIS**”<<4)+ **1**)	5	0x00	Start Address		End Address		-	-	-	Initialization of programming
	3	0x80	0xFF	0xFF	-	-	-	-	-	Full (selected memory) erasing
ID_PROG_DATA ((“**CRIS**”<<4)+ **2**)	n	data[0..(n-1)] (n≤8)								Data to program
ID_DISPLAY_DATA ((“**CRIS**”<<4)+ **3**)	5	0x00	Start Address		End Address		-	-	-	Display (read) data
		0x80					-	-	-	Blank check
ID_START_APPLI ((“**CRIS**”<<4)+ **4**)	2	0x03	0x00	-	-	-	-	-	-	Start Application with reset
	4		0x01	0x0000		-	-	-	-	Start Application jump add. 0
ID_SELECT_MEM_PAGE ((“**CRIS**”<<4)+ **6**)	3	0x00	Memory space	Page	-	-	-	-	-	No action
		0x01			-	-	-	-	-	Select Memory space
		0x02			-	-	-	-	-	Select Page
		0x03			-	-	-	-	-	Select Memory space & Page

Table 9-2. CAN Protocol Summary - Answers from Bootloader

ISP Command Answer Identifier	L	Data [0]	Data [1]	Data [2]	Data [3]	Data [4]	Data [5]	Data [6]	Data [7]	Description
ID_SELECT_NODE ((“**CRIS**”<<4)+ **0**)	2	Boot loader revision	0x00	-	-	-	-	-	-	Communication closed
			0x01	-	-	-	-	-	-	Communication opened
ID_PROG_START ((“**CRIS**”<<4)+ **1**)	0	-	-	-	-	-	-	-	-	Initialization of programming command OK
	1	0x00	-	-	-	-	-	-	-	Erase done
ID_PROG_DATA ((“**CRIS**”<<4)+ **2**)	1	0x00	-	-	-	-	-	-	-	Cmd. OK & end of transfer
		0x02	-	-	-	-	-	-	-	Cmd. OK & new data expected
ID_DISPLAY_DATA ((“**CRIS**”<<4)+ **3**)	n	data[0..(n-1)] (n≤8)								Data Read
	0	-	-	-	-	-	-	-	-	Blank check OK
	2	1st Failed Address		-	-	-	-	-	-	Error on Blank check
ID_SELECT_MEM_PAGE or ID_ERROR ((“**CRIS**”<<4)+ **6**)	1	0x00	-	-	-	-	-	-	-	Selection OK or Error Software Security Set

10. Appendix C: UART Protocol Summary

Table 10-1. UART Protocol Summary - Requests from Host

ISP Command Request		RECORD LENGTH	OFFSET	RECORD TYPE	Data [0]	Data [1]	Data [2]	Data [3]	Data [4]	...	Data [n-1]
Program Memory	":"	n	First Address	0x0**0**	data[0..(n-1)] ($n \leq 255$)						
Start Application	":"	0x00	0x0000	0x0**1**	-	-	-	-	-	-	-
Select New Page	":"	0x02	Start Address	0x0**2**	[7..4]=Page [3..0]=0x0	0x00	-	-	-	-	-
Select Memory	":"	0x02	0x0000	0x0**4**	Memory Space	Page	-	-	-	-	-
Read Memory	":"	0x05	0x0000		Start Address		End Address		0x00	-	-
Blank Check Memory	":"	0x05	0x0000		Start Address		End Address		0x01	-	-
Erase Memory	":"	0x05	0x0000		0x00	0xFF	0x00	0x00	0x02	-	-

Table 10-2. UART Protocol Summary - Answers from Bootloader

Answer	Character[0]	Character[1]	Character[2]	Character[3]	...	Character[n]
Command done (OK)	"**.**"	"C_R"	"L_F"	-	-	-
Read Memory Command Done (OK)	Address = data (16-byte formatted max.) + "C_R" + "L_F"					
Wrong Checksum	"**X**"	"C_R"	"L_F"	-	-	-
Blank Check Memory Error	1st Failed Address		"C_R"	"L_F"	-	-
Error - Software Security Set ("*ISP Read Memory*" only)	"**L**"	"C_R"	"L_F"	-	-	-
Error - Software Security Set ("*ISP Program Memory*" only)	"**P**"	"C_R"	"L_F"	-	-	-

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

Printed on recycled paper.