AT90CAN128
AT90CAN64
AT90CAN32

"GCC" CAN Boot Loader

1. Doc. Control

Boot Loader Revision	Purpose of Modifications	Compiler Version	Date
Rev. 1.0.0	First release Boot Loader Version: 03	**WinAVR** 20060125	07/Apr/2006
Rev. 1.1.0	Addding API Start Application command Boot Loader Version: 04	**WinAVR** 20070122	19/Apr/2007

2. Features

- **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**
- **In-Application Programming**
  - **Up to 255 nodes**
  - **16 Re-locatable Reseved Identifiers**
- **Application Programming Interface**
  - **Write Flash API (application section)**

3. Description

This document describes the "**GCC**" **CAN AT90CAN128/64/32 boot loader** functionality as well as its protocol to efficiently perform operations on the on chip Flash & EEPROM memories.

This boot loader 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 boot loader can manage a communication with an host through the CAN network. It can also access and perform requested operations on the on-chip Flash & EEPROM memories.

In-application programming feature is available to manage up to 255 CAN nodes.

A special entry (Flash API) is available for users.

4. Boot Loader Environment

The “**GCC**” **CAN AT90CAN128/64/32 boot loader** is loaded in the “Boot Loader Flash Section” of the on-chip Flash memory. The boot loader size is less than 8K bytes, so the physical “Boot Loader Flash Section” only is full. The application program size must be lower or equal the “Application Flash Section” plus 8K bytes (c.f. Table 4-1 ”Device Memory Mapping (byte addressing)” on page 2).

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

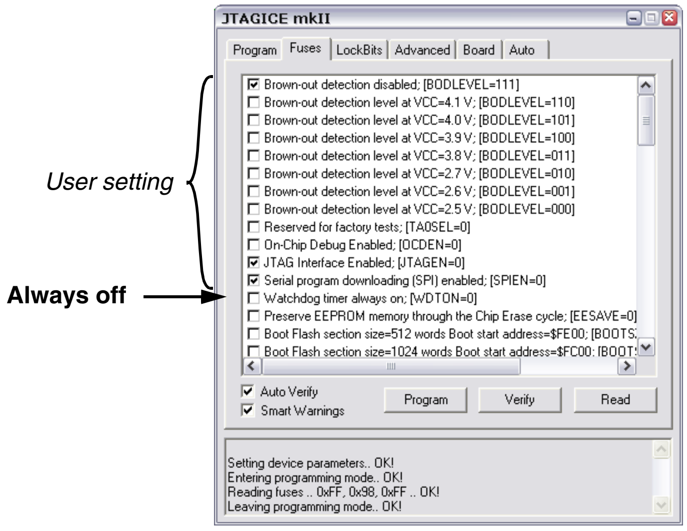
Memory		AT90CAN128	AT90CAN64	AT90CAN32
FLASH	Size	128 K bytes	64 K bytes	32 K bytes
	Add. Range	0x00000 - 0x1FFFF	0x00000 - 0x0FFFF	0x00000 - 0x07FFF
“Application Flash Section”	Size	120 K bytes	56 K bytes	24 K bytes
	Add. Range	0x00000 - 0x1DFFF	0x00000 - 0xDFFF	0x00000 - 0x05FFF
“Boot Loader Flash Section”	Size	8 K bytes		
	Add. Range	0x1E000 - 0x1FFFF	0x0E000 - 0x0FFFF	0x06000 - 0x07FFF
“Boot Loader Reset Addresses” (1)	Small (1st) Boot	0x1FC00	0x0FC00	0x07C00
	Second Boot	0x1F800	0x0F800	0x07800
	Third Boot	0x1F000	0x0F000	0x07000
	Large (4th) Boot (2)	0x1E000	0x0E000	0x06000
EEPROM	Size	4 K bytes	2 K bytes	1 K bytes
	Add. Range	0x0000 - 0x0FFF	0x0000 - 0x07FF	0x0000 - 0x03FF

Note: 1. The “Boot Loader Reset Address” depends on the fuse bits “BOOTSZ”.
Refer to the datasheet for more details on Flash memories (Flash, EEPROM, ...) behaviors.
2. “GCC CAN Boot Loader Reset” Address.

4.1 Device Fuse Setting

Please, refer to the device Data Sheet for further explanation.

Figure 4-1. Device Fuses Setting - Part 1

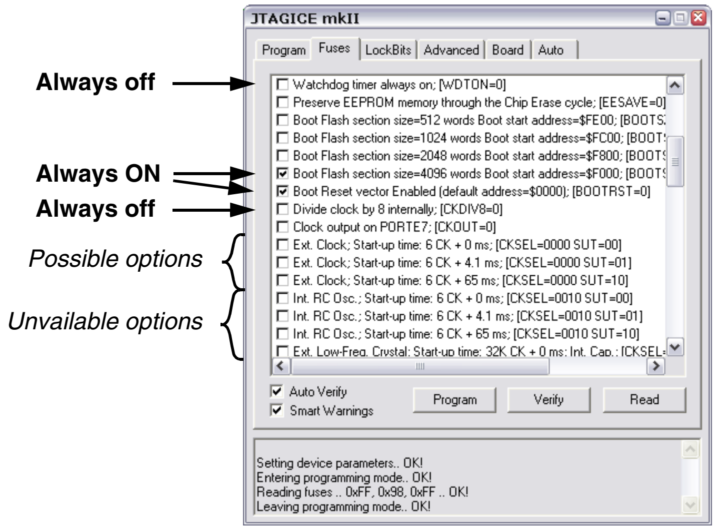


- [✓] Brown-out detection disabled; [BODLEVEL=111]
- [ ] Brown-out detection level at VCC=4.1 V; [BODLEVEL=110]
- [ ] Brown-out detection level at VCC=4.0 V; [BODLEVEL=101]
- [ ] Brown-out detection level at VCC=3.9 V; [BODLEVEL=100]
- [ ] Brown-out detection level at VCC=3.8 V; [BODLEVEL=011]
- [ ] Brown-out detection level at VCC=2.7 V; [BODLEVEL=010]
- [ ] Brown-out detection level at VCC=2.6 V; [BODLEVEL=001]
- [ ] Brown-out detection level at VCC=2.5 V; [BODLEVEL=000]
- [ ] Reserved for factory tests; [TA0SEL=0]
- [ ] On-Chip Debug Enabled; [OCDEN=0]
- [✓] JTAG Interface Enabled; [JTAGEN=0]
- [✓] Serial program downloading (SPI) enabled; [SPIEN=0]
- [ ] Watchdog timer always on; [WDTON=0]
- [ ] Preserve EEPROM memory through the Chip Erase cycle; [EESAVE=0]
- [ ] Boot Flash section size=512 words Boot start address=$FE00; [BOOTS
- [ ] Boot Flash section size=1024 words Boot start address=$FC00; [BOOT

- [✓] Auto Verify
- [✓] Smart Warnings

```
Setting device parameters.. OK!
Entering programming mode.. OK!
Reading fuses .. 0xFF, 0x98, 0xFF .. OK!
Leaving programming mode.. OK!
```

Figure 4-2. Device Fuses Setting - Part 2

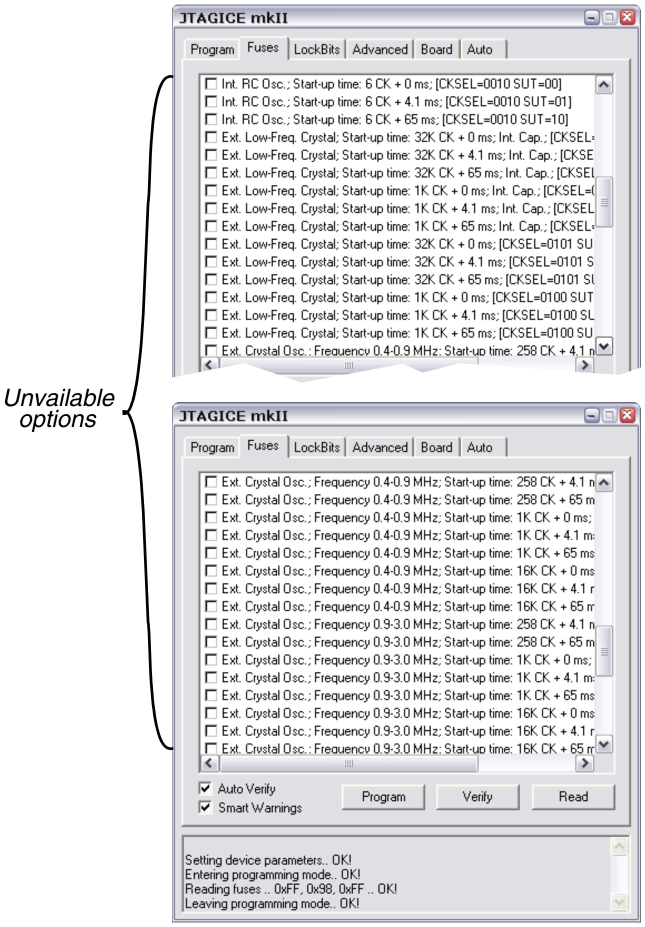


- ☐ Watchdog timer always on; [WDTON=0]
- ☐ Preserve EEPROM memory through the Chip Erase cycle; [EESAVE=0]
- ☐ Boot Flash section size=512 words Boot start address=$FE00; [BOOTS
- ☐ Boot Flash section size=1024 words Boot start address=$FC00; [BOOT
- ☐ Boot Flash section size=2048 words Boot start address=$F800; [BOOT
- ☑ Boot Flash section size=4096 words Boot start address=$F000; [BOOT
- ☑ Boot Reset vector Enabled (default address=$0000); [BOOTRST=0]
- ☐ Divide clock by 8 internally; [CKDIV8=0]
- ☐ Clock output on PORTE7; [CKOUT=0]
- ☐ Ext. Clock; Start-up time: 6 CK + 0 ms; [CKSEL=0000 SUT=00]
- ☐ Ext. Clock; Start-up time: 6 CK + 4.1 ms; [CKSEL=0000 SUT=01]
- ☐ Ext. Clock; Start-up time: 6 CK + 65 ms; [CKSEL=0000 SUT=10]
- ☐ Int. RC Osc.; Start-up time: 6 CK + 0 ms; [CKSEL=0010 SUT=00]
- ☐ Int. RC Osc.; Start-up time: 6 CK + 4.1 ms; [CKSEL=0010 SUT=01]
- ☐ Int. RC Osc.; Start-up time: 6 CK + 65 ms; [CKSEL=0010 SUT=10]
- ☐ Ext. Low-Freq. Crystal; Start-up time: 32K CK + 0 ms; Int. Cap.; [CKSEL

☑ Auto Verify
☑ Smart Warnings

```
Setting device parameters.. OK!
Entering programming mode.. OK!
Reading fuses .. 0xFF, 0x98, 0xFF .. OK!
Leaving programming mode.. OK!
```

Figure 4-3. Device Fuses Setting - Part 3



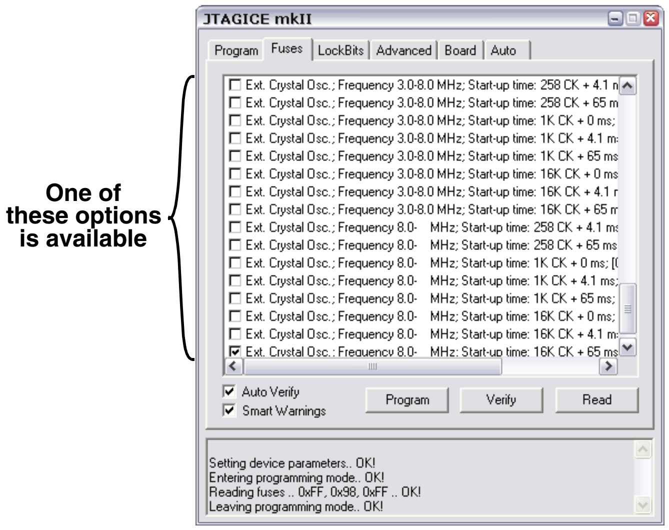
- ☐ Int. RC Osc.; Start-up time: 6 CK + 0 ms; [CKSEL=0010 SUT=00]
- ☐ Int. RC Osc.; Start-up time: 6 CK + 4.1 ms; [CKSEL=0010 SUT=01]
- ☐ Int. RC Osc.; Start-up time: 6 CK + 65 ms; [CKSEL=0010 SUT=10]
- ☐ Ext. Low-Freq. Crystal; Start-up time: 32K CK + 0 ms; Int. Cap.; [CKSEL
- ☐ Ext. Low-Freq. Crystal; Start-up time: 32K CK + 4.1 ms; Int. Cap.; [CKSE
- ☐ Ext. Low-Freq. Crystal; Start-up time: 32K CK + 65 ms; Int. Cap.; [CKSEL
- ☐ Ext. Low-Freq. Crystal; Start-up time: 1K CK + 0 ms; Int. Cap.; [CKSEL=
- ☐ Ext. Low-Freq. Crystal; Start-up time: 1K CK + 4.1 ms; Int. Cap.; [CKSEL
- ☐ Ext. Low-Freq. Crystal; Start-up time: 1K CK + 65 ms; Int. Cap.; [CKSEL
- ☐ Ext. Low-Freq. Crystal; Start-up time: 32K CK + 0 ms; [CKSEL=0101 SU
- ☐ Ext. Low-Freq. Crystal; Start-up time: 32K CK + 4.1 ms; [CKSEL=0101 S
- ☐ Ext. Low-Freq. Crystal; Start-up time: 32K CK + 65 ms; [CKSEL=0101 S
- ☐ Ext. Low-Freq. Crystal; Start-up time: 1K CK + 0 ms; [CKSEL=0100 SUT
- ☐ Ext. Low-Freq. Crystal; Start-up time: 1K CK + 4.1 ms; [CKSEL=0100 SU
- ☐ Ext. Low-Freq. Crystal; Start-up time: 1K CK + 65 ms; [CKSEL=0100 SU
- ☐ Ext. Crystal Osc.; Frequency 0.4-0.9 MHz; Start-up time: 258 CK + 4.1

- ☐ Ext. Crystal Osc.; Frequency 0.4-0.9 MHz; Start-up time: 258 CK + 4.1 n
- ☐ Ext. Crystal Osc.; Frequency 0.4-0.9 MHz; Start-up time: 258 CK + 65 m
- ☐ Ext. Crystal Osc.; Frequency 0.4-0.9 MHz; Start-up time: 1K CK + 0 ms;
- ☐ Ext. Crystal Osc.; Frequency 0.4-0.9 MHz; Start-up time: 1K CK + 4.1 m
- ☐ Ext. Crystal Osc.; Frequency 0.4-0.9 MHz; Start-up time: 1K CK + 65 ms
- ☐ Ext. Crystal Osc.; Frequency 0.4-0.9 MHz; Start-up time: 16K CK + 0 ms
- ☐ Ext. Crystal Osc.; Frequency 0.4-0.9 MHz; Start-up time: 16K CK + 4.1 r
- ☐ Ext. Crystal Osc.; Frequency 0.4-0.9 MHz; Start-up time: 16K CK + 65 m
- ☐ Ext. Crystal Osc.; Frequency 0.9-3.0 MHz; Start-up time: 258 CK + 4.1 n
- ☐ Ext. Crystal Osc.; Frequency 0.9-3.0 MHz; Start-up time: 258 CK + 65 m
- ☐ Ext. Crystal Osc.; Frequency 0.9-3.0 MHz; Start-up time: 1K CK + 0 ms;
- ☐ Ext. Crystal Osc.; Frequency 0.9-3.0 MHz; Start-up time: 1K CK + 4.1 m
- ☐ Ext. Crystal Osc.; Frequency 0.9-3.0 MHz; Start-up time: 1K CK + 65 ms
- ☐ Ext. Crystal Osc.; Frequency 0.9-3.0 MHz; Start-up time: 16K CK + 0 ms
- ☐ Ext. Crystal Osc.; Frequency 0.9-3.0 MHz; Start-up time: 16K CK + 4.1 r
- ☐ Ext. Crystal Osc.; Frequency 0.9-3.0 MHz; Start-up time: 16K CK + 65 m

☑ Auto Verify
☑ Smart Warnings

```
Setting device parameters.. OK!
Entering programming mode.. OK!
Reading fuses .. 0xFF, 0x98, 0xFF .. OK!
Leaving programming mode.. OK!
```

Figure 4-4. Device Fuses Setting - Part 4



JTAGICE mkII

Program | Fuses | LockBits | Advanced | Board | Auto

```
☐ Ext. Crystal Osc.; Frequency 3.0-8.0 MHz; Start-up time: 258 CK + 4.1 n
☐ Ext. Crystal Osc.; Frequency 3.0-8.0 MHz; Start-up time: 258 CK + 65 m
☐ Ext. Crystal Osc.; Frequency 3.0-8.0 MHz; Start-up time: 1K CK + 0 ms;
☐ Ext. Crystal Osc.; Frequency 3.0-8.0 MHz; Start-up time: 1K CK + 4.1 m
☐ Ext. Crystal Osc.; Frequency 3.0-8.0 MHz; Start-up time: 1K CK + 65 ms
☐ Ext. Crystal Osc.; Frequency 3.0-8.0 MHz; Start-up time: 16K CK + 0 ms
☐ Ext. Crystal Osc.; Frequency 3.0-8.0 MHz; Start-up time: 16K CK + 4.1 r
☐ Ext. Crystal Osc.; Frequency 3.0-8.0 MHz; Start-up time: 16K CK + 65 m
☐ Ext. Crystal Osc.; Frequency 8.0-   MHz; Start-up time: 258 CK + 4.1 ms
☐ Ext. Crystal Osc.; Frequency 8.0-   MHz; Start-up time: 258 CK + 65 ms
☐ Ext. Crystal Osc.; Frequency 8.0-   MHz; Start-up time: 1K CK + 0 ms; [(
☐ Ext. Crystal Osc.; Frequency 8.0-   MHz; Start-up time: 1K CK + 4.1 ms;
☐ Ext. Crystal Osc.; Frequency 8.0-   MHz; Start-up time: 1K CK + 65 ms;
☐ Ext. Crystal Osc.; Frequency 8.0-   MHz; Start-up time: 16K CK + 0 ms;
☐ Ext. Crystal Osc.; Frequency 8.0-   MHz; Start-up time: 16K CK + 4.1 m
☑ Ext. Crystal Osc.; Frequency 8.0-   MHz; Start-up time: 16K CK + 65 ms
```

☑ Auto Verify
☑ Smart Warnings

Program | Verify | Read

```
Setting device parameters.. OK!
Entering programming mode.. OK!
Reading fuses .. 0xFF, 0x98, 0xFF .. OK!
Leaving programming mode.. OK!
```

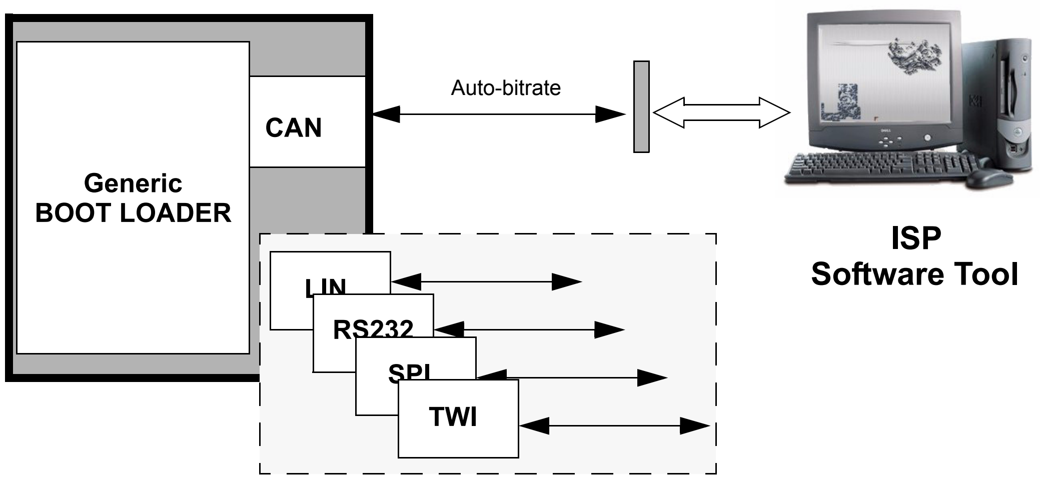
Mandatory fuse setting:

- **Fuse High Byte:**
  - **BOOTRST** programmed,
  - **BOOTSZ [1:0]** programmed for **4096** words,
  - **WDTON** unprogrammed.
- **Fuse Low Byte:**
  - **CKSEL [3:0]** programmed to select a clock with an high accuracy to match with CAN requirement (the internal RC oscillator doesn't match).

4.2 Physical Environment

A generic boot loader deals with the host (or PC) through a CAN interface. The generic boot loader is a service able to be connected to other interfaces (LIN, RS232, SPI, TWI, ...).

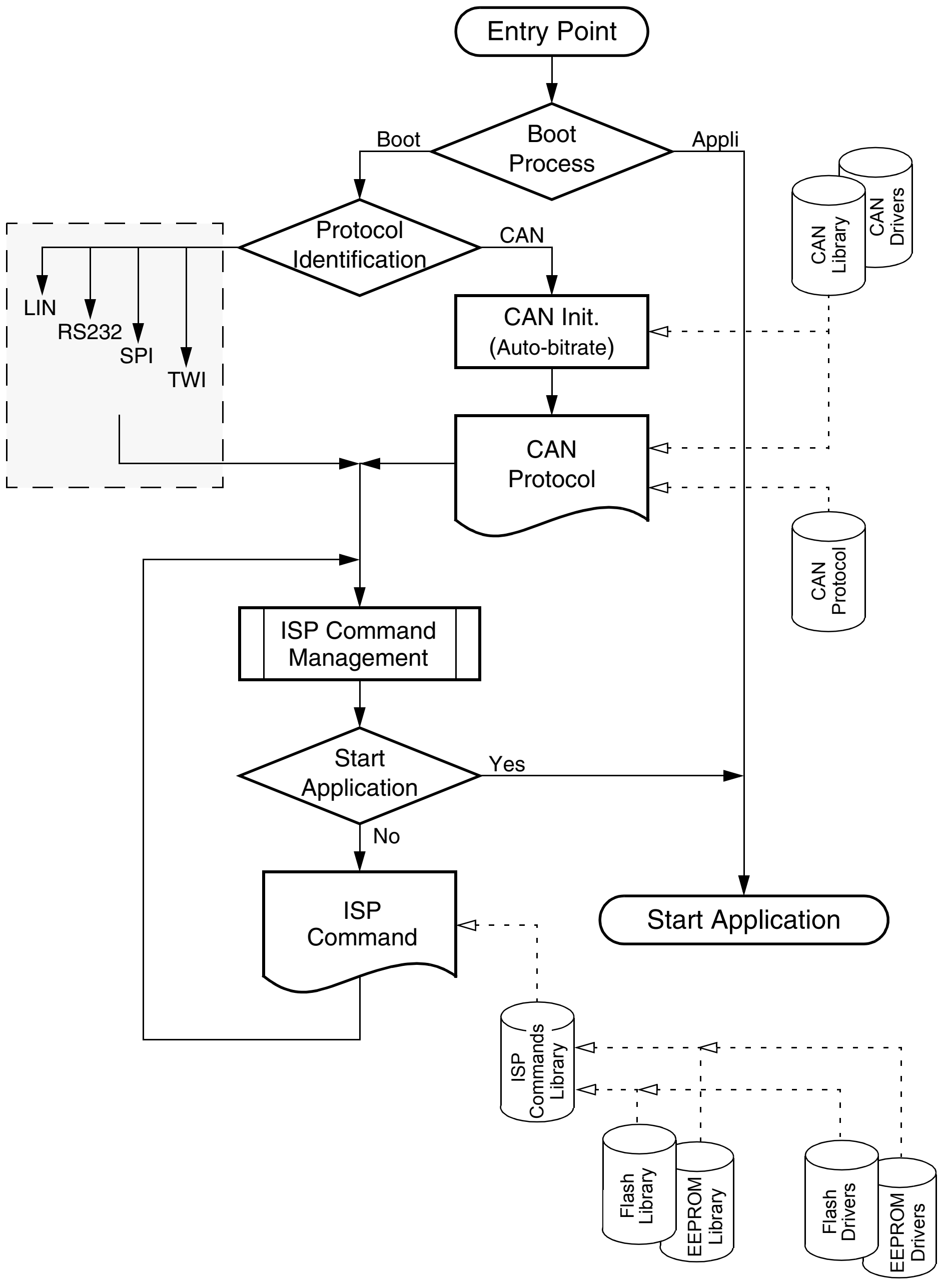
Figure 4-5. Physical Environment



4.3 Boot Loader Description

4.3.1 Overview

Figure 4-6. Boot Loader Diagram



4.3.2 Entry Point

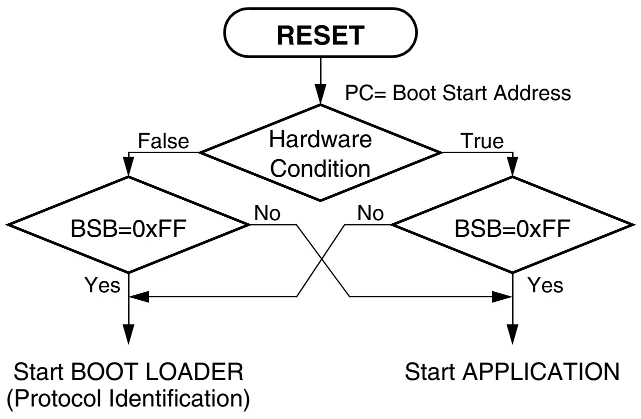
Only **one** "*Entry Point*" is available, it is the entry point to the boot loader. The "BOOTRST" fuse of the device have to be set. After Reset, the "Program Counter" of the device is set to "Boot Reset Address" (c.f. Table 4-1 "Device Memory Mapping (byte addressing)" on page 2). This "*Entry Point*"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

4.3.3 Boot Process

The “*boot process*” of the boot loader allows to start the application or the boot loader 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 “Boot Loader Configuration Memory” (c.f. Section 5.4.4.1 ”Boot Status Byte - “BSB”” on page 11). Its default value is 0xFF. An ISP command allows to change its value.

Figure 4-7. Boot Process Diagram



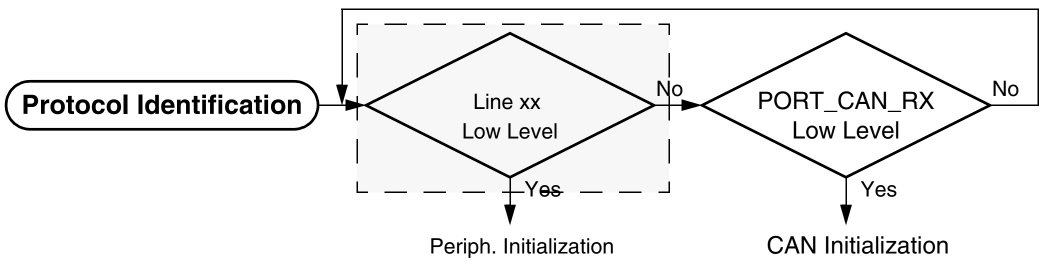
4.3.4 Protocol Identification

The “*Protocol Identification*“ of the boot loader select what protocol to use, CAN or other 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.
- *(user defined interface)*.

A low level on **PORT_CAN_RX** line starts the initialization of the CAN peripheral.

Figure 4-8. Protocol Identification Diagram



4.3.5 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 12):
- "**EB**" = 0xFFH: Use the software auto-bitrate.
- "**EB**" **!=** 0xFFH: Use Bit-Timing Control [1..3] bytes to set the CAN bitrate (see "Bit-Timing Control [1..3] - "BTC[1..3]"" on page 12).

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 boot loader 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.

4.3.6 CAN Protocol Overview

The "*CAN Protocol*" is an higher level protocol over serial line (CAN Bus).

It is described in specific paragraphs in this document (See "CAN Protocol & ISP Commands" on page 15.).

4.3.7 ISP Commands Overview

The "*CAN Protocol*" decodes" *ISP commands*". The set of "I*SP commands*" obviously is independent of any protocol.

It is described in a specific paragraph in this document (See "CAN Protocol & ISP Commands" on page 15.).

4.3.8 Output From Boot Loader

The output from the boot loader is performs after receiving the ISP command: "*Start Application*" (See "CAN Protocol & ISP Commands" on page 15.).

5. Memory Space Definition

The boot loader supports up to five (**5**) 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	**0**	Read & Write
EEPROM Data Memory	**1**	Read & Write
Signature	**2**	Read only
Boot Loader Information	**3**	Read only
Boot Loader Configuration	**4**	Read & Write
Device registers [2]	**5**	Read only

Note: 1. Sometimes, the discriminating is not physical (ex: “Signature” is a sub-set of the code of the boot loader Flash Section” as well as “Boot Loader Information”).

2. New feature.

5.1 Flash Memory Space

The Flash memory space managed by the boot loader 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 boot loader 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 8). Specific ISP commands allows to select the different pages.

The boot loader 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 11).

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 11).

5.1.3 Limits

The ISP commands on the Flash memory space has no effect on the boot loader (no effect on "*Boot Loader 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 8.

5.2 EEPROM Data Memory

The EEPROM data memory space managed by the boot loader 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 boot loader 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 11).

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 11).

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 9.

5.3 Boot Loader Information

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

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

Signature Space		AT90CAN128	AT90CAN64	AT90CAN32
Bootloader Version	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 ”Boot Loader Information Space (Code Number 3)” on page 9.

5.3.4 Boot Loader Information Byte Description

5.3.4.1 Boot Version

Boot Version: 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 Boot Loader Configuration

The Boot loader configuration space managed by the boot loader is included in the “*Boot Loader Flash Section*”.

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

Signature Space			AT90CAN128	AT90CAN64	AT90CAN32
Boot Status Byte	**“BSB”**	Add.: 0x00	(default value=0xFF)		
Software Security Byte	**“SSB”**	Add.: 0x01	(default value=0xFF)		
Extra Byte	**“EB”**	Add.: 0x02	(default value=0xFF) (1)		
Bit-Timing Control 1	**“BTC1”**	Add.: 0x03	(default value=0xFF) (2)		
Bit-Timing Control 2	**“BTC2”**	Add.: 0x04	(default value=0xFF) (2)		
Bit-Timing Control 3	**“BTC3”**	Add.: 0x05	(default value=0xFF) (2)		
Node Number	**“NNB”**	Add.: 0x06	(default value=0xFF) (3)		
CAN Re-locatable ID Segment	**“CRIS”**	Add.: 0x07	(default value=0x00)		
Start Address Low	**“SA_L”**	Add.: 0x08	(default value=0x00)		
Start Address High	**“SA_H”**	Add.: 0x09	(default value=0x00)		
Number of page(s)			-- No paging --		

Note: 1. See “Extra Byte - “EB”” on page 12. for validity.
2. See “Bit-Timing Control [1..3] - “BTC[1..3]“” on page 12. for validity.
3. See “(CAN) Node Number - “NNB”” on page 12. 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 11).

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 "Boot Loader Configuration Space (Code Number 4)" on page 10.

5.4.4 Boot Loader Configuration Byte Description

*5.4.4.1 Boot Status Byte - "**BSB**"*

The Boot Status Byte of the boot loader is used in the "*boot process*" (Section 4.3.3 "Boot Process" on page 6) to control the starting of the application or the boot loader. If no Hardware Condition is set, the default value (0xFF) of the Boot Status Byte will force the boot loader to start. Else (Boot Status Byte != 0xFF & no Hardware Condition) the application will start.

*5.4.4.2 Software Security Byte - "**SSB**"*

The boot loader 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 boot loader 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 boot loader 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	-

ISP Command	NO_SECURITY	WR_SECURITY	RD_WR_SECURITY
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 Bit-Timing Control[1..3] bytes (“**BTC1**”, “**BTC2**” & “**BTC3**”) of Boot loader configuration space to set the CAN bit timing registers of the CAN peripheral (no auto-bitrate).

5.4.4.4 *Bit-Timing Control [1..3] - “**BTC[1..3]**“*

When “**EB**” **!=** 0xFFH, Bit-Timing Control[1..3] bytes (“**BTC1**”, “**BTC2**” & “**BTC3**”) of Boot loader configuration space are used to set the CAN Bit-Timing Registers of the CAN peripheral - no auto-bitrate.
A way to setup these bytes is described in Section 5.6.4.1 ”CANBT[1..3] Registers” on page 14.

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

The node number is a physical address of a CAN node in a CAN cluster for Boot Loader. This “**NNB**” is used by the Host to open the communication with this CAN node.

See “CAN Protocol & ISP Commands” on page 15.

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

All the CAN ISP identifiers belongs to the an segment. “**CRIS**”<<4 defines the base value of this segment.
See “CAN Protocol & ISP Commands” on page 15.

5.4.4.7 *Start (application) Address High & Low- "**SA_H**" & "**SA_L**"*

See "CAN Protocol & ISP Commands" on page 15.

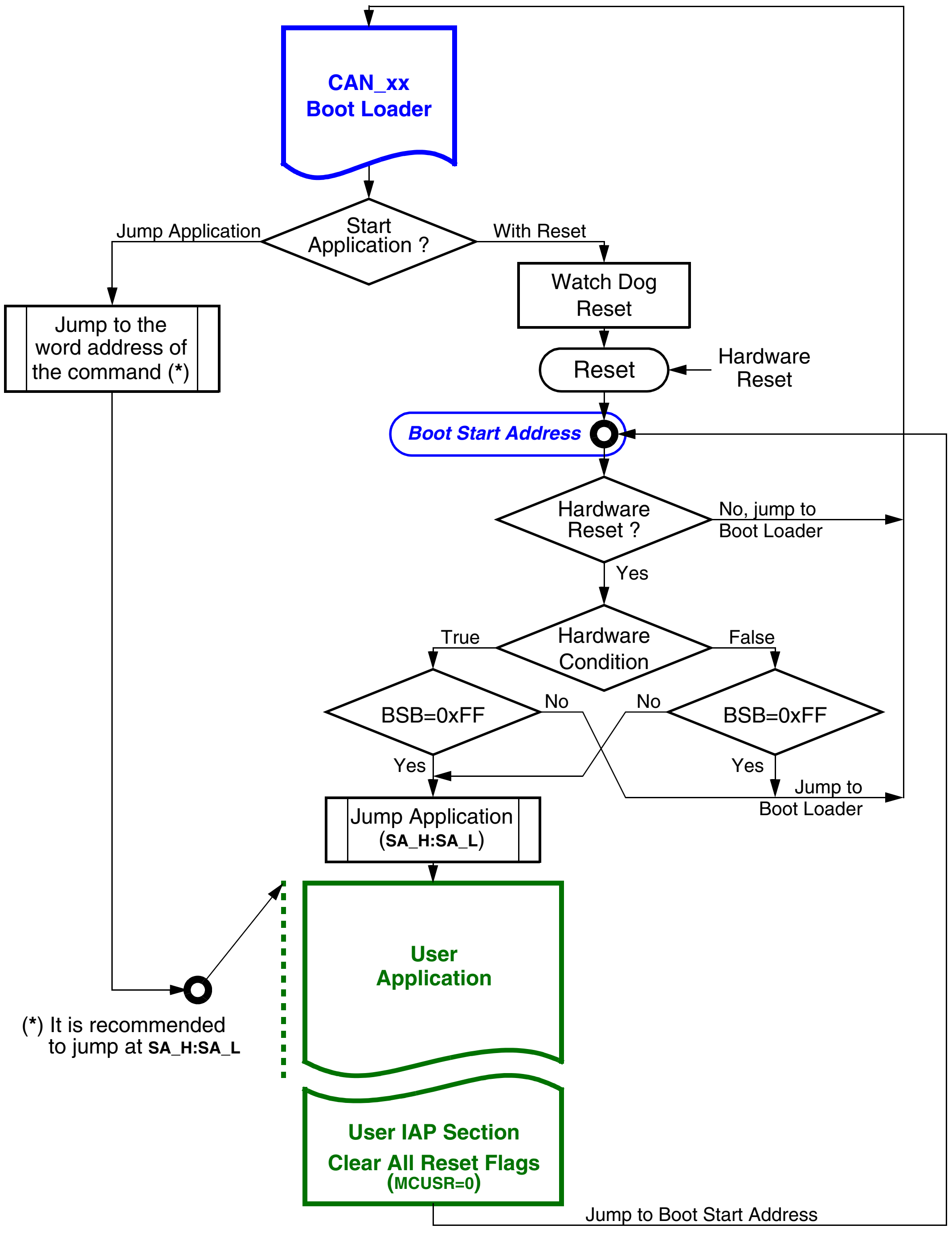


Figure 5-1. Start Application & Reset Diagram

5.5 Signature

The Signature space managed by the boot loader is included the code of the boot loader. It is in the "*Boot Loader Flash Section*".

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

Signature Space		AT90CAN128	AT90CAN64	AT90CAN32
Manufacturer Code	Address: 0x00 (Read only)	0x1E		
Family Code	Address: 0x01 (Read only)	0x81		
Product Name	Address: 0x02 (Read only)	0x97	0x96	0x95
Product Revision	Address: 0x03 (Read only)	≥ 0x00	≥ 0x00	≥ 0x00
Number of page(s)		-- No paging --		

5.5.1 Reading or Programming

The "I*SP 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 read only space.

5.5.3 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 14.

5.6 Device Registers

The device registers space managed by the boot loader 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**).

5.6.1 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 this read only space.

5.6.3 Limits

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

5.6.4 Device Registers Description

c.f. appropriate data sheet for information.

5.6.4.1 CANBT[1..3] Registers

The CANBT[1..3] Registers are at the addresses 0xE2 to 0xE4.

They can be read before disabling the auto-bitrate ("**EB**" **!=** 0xFFH) and re- copied into "**BTC1**", "**BTC2**" & "**BTC3**" of the Boot loader configuration space (see "Bit-Timing Control [1..3] - "BTC[1..3]"" on page 12). Then, the Boot loader will always start with <u>this</u> Bit-Timing (while "**EB**" **!=** 0xFFH !!!). Is very useful in case of IAP.

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.

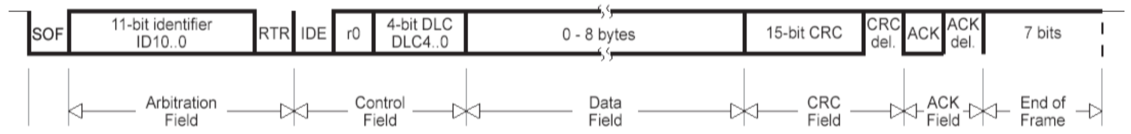
6.1 CAN Frame Description

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 boot loader 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.

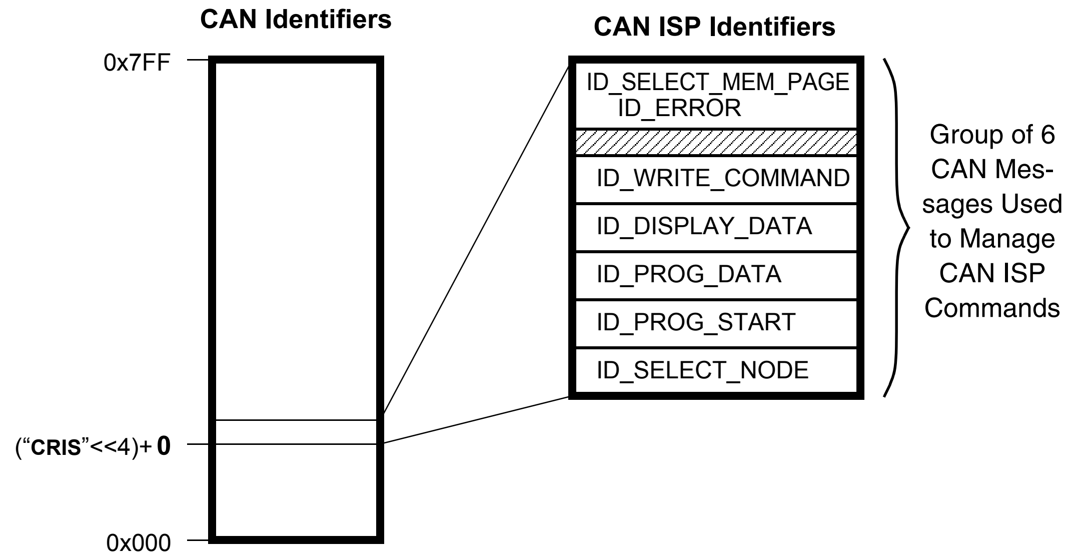
Table 6-2. Defined CAN Message Identifiers for CAN ISP Protocol

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 boot loader 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. If "**CRIS**" > 0x7F then it will be automatically set to 0x00 (default value).

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



<u>**Example**</u>: "**CRIS**" = 0x28

- "**ID_SELECT_NODE**" = 0x280
- ....
- "**ID_ERROR**" = 0x286

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 boot loader 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 Boot Loader First Connection

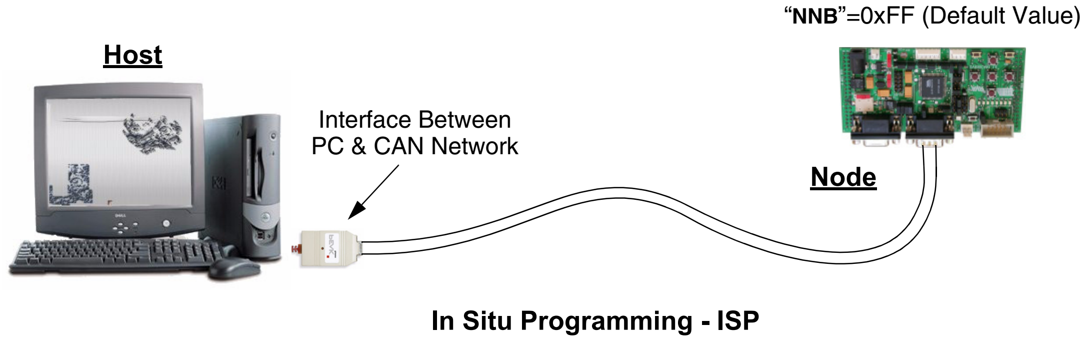
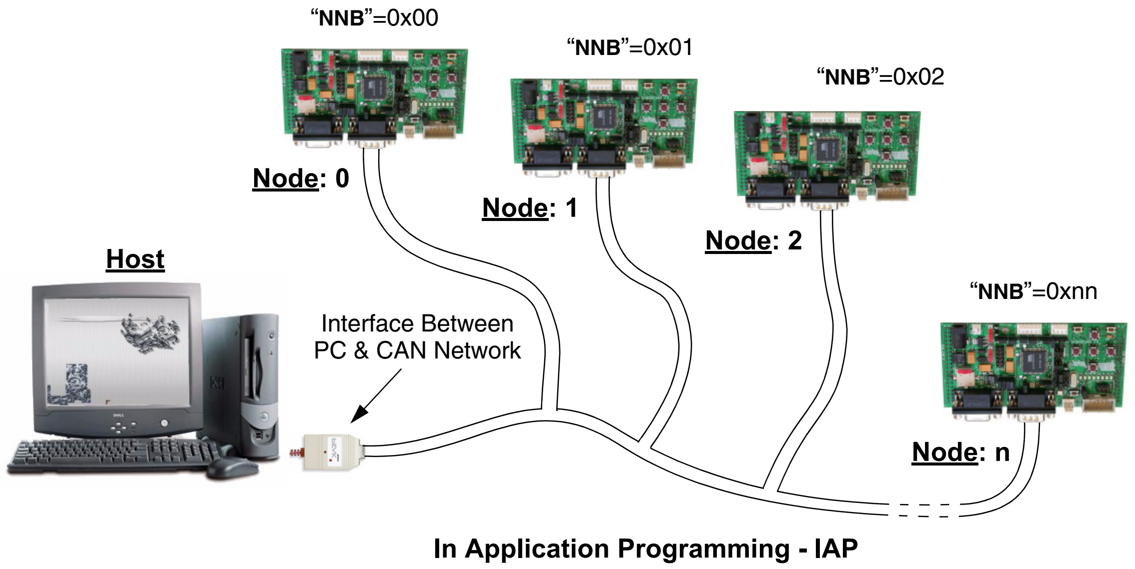

“NNB”=0xFF (Default Value)
Host
Interface Between PC & CAN Network
Node
In Situ Programming - ISP

Figure 6-4. CAN Boot Loader 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 Boot Loader

Table 6-4. CAN Node Select Answers from Boot Loader

Identifier	L	Data[0]	Data[1]	Description
ID_SELECT_NODE ((“**CRIS**”<<4)+ **0**)	2	“*Boot Loader Version*”	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 Boot Loader

Table 6-6. Changing Memory / Page Answers from Boot Loader

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 Boot Loader

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

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 Boot Loader

Table 6-10. Programming / Erasing Memory Answers from Boot Loader

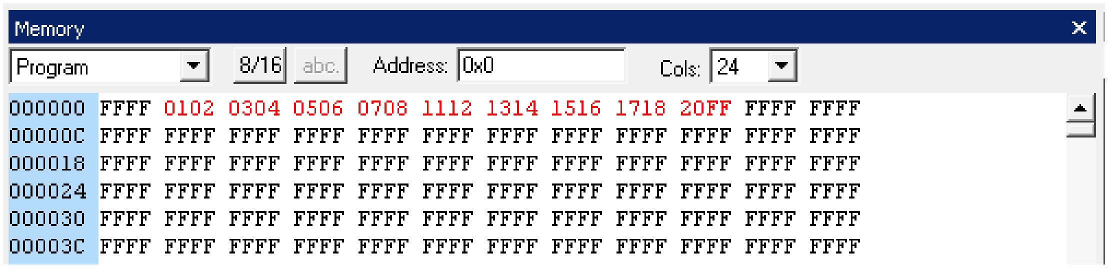
Identifier	L	Data[0]	Description
ID_PROG_START ((“**CRIS**”<<4)+ **1**)	0	-	Command OK
ID_PROG_DATA ((“**CRIS**”<<4)+ **2**)	1	0x00	Command OK and end of transfer
		0x02	Command OK but new (other) data expected
ID_ERROR ((“**CRIS**”<<4)+ **6**)	1	0x00	Error - Software Security Set (“*Init. program*” 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	Prog. Add 0x0002 up to 0x0012
A (<<)	001	0	0	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.

6.3.5.1 Starting Application Requests from Host

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 the defined word address. The jump word address can be differ from **SA_H:SA_L** (Boot Loader Configuration Space).

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	Jump W-Add. (MSB, LSB)		Start Application at W-Add. (MSB : LSB) without reset

6.3.5.2 Starting Application Answer from Boot Loader

No answer is returned by the boot loader.

7. API - Application Programming Interface

7.1 API Definition

An application programming interface (API) is a source code interface that a computer system or program library provides in order to support requests for services to be made of it by a computer program.

7.2 API Implementation

The specificity of ATMEL AVR 8-bit microprocessors is that the code providing the writing in flash needs to be located in the bootloader section. If the "***GCC***" CAN Boot Loader is flashed in a part, the bootloader section is already occupied and unavailable for user.

The solution that allows the user to write in flash is to “open” the ‘flash_api_wr_block’ routine contained in the "***GCC***" CAN Boot Loader. Then the user could call it in its own application.

7.3 Entry Point

Because the user application and the "***GCC***" CAN Boot Loader are not compiled together, an entry point is used.

Regardless the part number, the entry point is fixed to: **`FLASH_SIZE - 6 bytes`**

Table 7-1. “flash_api_wr_block” API Entry Point Versus Part Number.

Part Number	Entry Point Word Address	Entry Point Byte Address
AT90CAN32	0x03FFD	0x07FFA
AT90CAN64	0x07FFD	0x0FFFA
AT90CAN128	0x0FFFD	0x1FFFA

7.4 Routine Prototype

```
//-----------------------------------------------------------------------------
// This function allows to write up to 65535 bytes (64K Bytes-1 byte) in
// the Flash memory.
// This function manages alignment issue (byte and page alignments).
//-----------------------------------------------------------------------------
// Warning:
//      1 -  This function isn't able to address the fully 64K bytes but we
//           cannot find in the device a source buffer up to 64K bytes !
//      2 -  The 64K page setting must be done before (i.e. setting the
//           RAMPZ register).
// Parameters:
//     - src     Source of (SRAM) buffer address.
//     - dest    Destination, start Flash address where data must be written.
//     - byte_nb Number of byte to write.
// Return:       (none)
//------
extern void flash_api_wr_block(unsigned char* src, unsigned short dest, unsigned short byte_nb);
```

8. Appendix A: #define's

8.1 Type Definition

```
// -------------- DECLARATION --------------
typedef unsigned char      Bool;
typedef unsigned char       U8 ;
typedef unsigned short      U16;
typedef unsigned long       U32;
typedef signed char         S8 ;
typedef signed short        S16;
typedef signed long         S32;
```

8.2 Processor Definition

```
// -------------- PROCESSOR DEFINITION --------------
#define MANUF_ID             0x1E        // ATMEL
#define FAMILY_CODE          0x81        // AT90CANxx family

#define XRAM_END             XRAMEND     // Defined in "iocan128/64/32.h"
#define RAM_END              RAMEND      // Defined in "iocan128/64/32.h"
#define E2_END               E2END       // Defined in "iocan128/64/32.h"
#define FLASH_END            FLASHEND    // Defined in bytes in "iocan128/64/32.h"
#define FLASH_SIZE           ((U32)(FLASH_END)) + 1 // Size in bytes
#define FLASH_PAGE_SIZE      256         // Bytes, constant for AT90CANxx devices

// For specific definitions & switches
#if   defined(__AVR_AT90CAN128__)
#   define PRODUCT_NAME        0x97      // 128 Kbytes of Flash
#   define PRODUCT_REV         0x00      // Rev 0
#   define _RAMPZ_IS_USED_     1           // RAMPZ used if Flash upper than 64K bytes

#elif defined(__AVR_AT90CAN64__)
#   define PRODUCT_NAME        0x96      // 64 Kbytes of Flash
#   define PRODUCT_REV         0x00      // Rev 0

#elif defined(__AVR_AT90CAN32__)
#   define PRODUCT_NAME        0x95      // 32 Kbytes of Flash
#   define PRODUCT_REV         0x00      // Rev 0

#else
#   error Wrong device selection in plug-in for AVR-GCC: "Project Options -> Device"
#endif

#define FOSC  8000          // 8 MHz External cristal
#define F_CPU(FOSC*1000)   // Need for AVR GCC
```

8.3 CAN Link Definition

```
//---------------- CAN DEFINITION -------------
#define CAN_BAUDRATE   CAN_AUTOBAUD

#define PIN_CAN_RX     PIND_Bit6
#define PORT_CAN_TX    PORTD_Bit5
```

8.4 Boot Loader Definition

```
//-------------- BOOTLOADER CONFIGURATION -------------
#define BOOT_LOADER_SIZE            0x2000  // Size in bytes: 8KB
#define MAX_FLASH_SIZE_TO_ERASE     ( FLASH_SIZE - ((U32)(BOOT_LOADER_SIZE)) )

#define BOOT_VERSION    0x04
#define BOOT_ID1        0xD1
#define BOOT_ID2        0xD2

#define BSB_DEFAULT     0xFF
#define SSB_DEFAULT     0xFF
#define EB_DEFAULT      0xFF
#define BTC1_DEFAULT    0xFF
#define BTC2_DEFAULT    0xFF
```

```
#define BTC3_DEFAULT         0xFF
#define NNB_DEFAULT          0xFF
#define CRIS_DEFAULT         0x00
#define SA_H_DEFAULT         0x00
#define SA_L_DEFAULT         0x00

#define BSB  ((U16) &boot_conf[0])
#define SSB  ((U16) &boot_conf[1])
#define EB   ((U16) &boot_conf[2])
#define BTC1 ((U16) &boot_conf[3])
#define BTC2 ((U16) &boot_conf[4])
#define BTC3 ((U16) &boot_conf[5])
#define NNB  ((U16) &boot_conf[6])
#define CRIS ((U16) &boot_conf[7])
#define SA_H ((U16) &boot_conf[8])
#define SA_L ((U16) &boot_conf[9])

#define BOOT_CONF_SIZE         10
#define SSB_INDEX            0x01
#define SSB_NO_SECURITY      0xFF
#define SSB_WR_PROTECTION    0xFE
#define SSB_RD_WR_PROTECTION 0xFC
```

8.5 Memory Definition

```
//-------- MEMORY DEFINITION -----------------
#define MEM_FLASH            0
#define MEM_EEPROM           1
#define MEM_SIGNATURE        2
#define MEM_BOOT_INF         3   // Boot Loader information
#define MEM_BOOT_CONF        4   // Boot Loader configuration
#define MEM_HW_REG           5
#define MEM_DEF_MAX          MEM_HW_REG
#define MEM_DEFAULT          MEM_FLASH

#define PAGE_DEFAULT         0x00
#define ADD_DEFAULT          0x0000
#define N_DEFAULT            0x0001
```

8.6 CAN Protocol Definition

```
//------ IAP data --------------------------
#define MAX_BASE_ISP_IAP_ID         0x7F0
#define MIN_BASE_ISP_IAP_ID         0x000
//------ Protocol commands -----------------
#define CAN_ID_SELECT_NODE          0x00
#define CAN_ID_PROG_START           0x01
#   define  CAN_INIT_PROG           0x00
#   define  CAN_FULL_ERASE_1        0x80
#   define  CAN_FULL_ERASE_2        0xFF
#   define  CAN_FULL_ERASE_3        0xFF
#define CAN_ID_PROG_DATA            0x02
#define CAN_ID_DISPLAY_DATA         0x03
#   define  CAN_READ_DATA           0x00
#   define  CAN_BLANK_CHECK         0x80
#define CAN_ID_START_APPLI          0x04
#   define  CAN_START_APPLI         0x03
#   define  CAN_RESET_APPLI         0x00
#   define  CAN_JUMP_APPLI          0x01
#define CAN_ID_SELECT_MEM_PAGE      0x06
#   define  CAN_NO_ACTION           0x00
#   define  CAN_SEL_MEM             0x01
#   define  CAN_SEL_PAGE            0x02
#   define  CAN_SEL_MEM_N_PAGE      0x03

#define CAN_ID_ERROR                0x06
#define COMMAND_OK                  0x00
#define OK_END_OF_DATA              0x00
#define OK_NEW_DATA                 0x02
#define LOCAL_BUFFER_SIZE           0x100
```

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	-	-	-	-	-	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	Jump W-Add.		-	-	-	-	Start Application at W-Add.
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 Boot Loader

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 Version	0x00	-	-	-	-	-	-	Communication closed
			0x01	-	-	-	-	-	-	Communication opened
ID_PROG_START ((“**CRIS**”<<4)+ **1**)	0	-	-	-	-	-	-	-	-	Command OK
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