

CANopen Bootloader for the *ELMB* ATmega128 microcontroller

Henk Boterenbrood
NIKHEF, Amsterdam
10 Mar 2004

Version 1.1

ABSTRACT

*The **ELMBbl** Bootloader firmware resides in the boot sector of the ELMB's ATmega128 microcontroller flash memory. It enables (re)programming of the application flash memory by means of CANopen messages received by the ELMB's CAN-bus interface.*

Contents

1	INTRODUCTION.....	2
2	DESCRIPTION.....	2
3	PROGRAMMING INSTRUCTIONS.....	5
4	OBJECT DICTIONARY.....	7
5	EMERGENCY OBJECTS.....	8
	REFERENCES.....	9

Version History		
Version	Date	Comments
1.1	10 Mar 2003	Firmware version <i>ELMBbl</i> v1.3: - CAN Node-ID may be taken from ATmega128 EEPROM. Firmware version <i>ELMBbl</i> v1.2: - addition of data bytes readout by standard SDO.
1.0	9 Aug 2003	First version describing <i>ELMBbl</i> v1.1.

Table 1. Document change record.

1 Introduction

The ELMB128 module (or ELMB for short) is equipped with an 8-bit ATMEL ATmega128 microcontroller [1]. The upper 8 Kbyte section of the ATmega128's 128 Kbyte flash memory is reserved for a so-called *Bootloader*, a separate application that takes care of the *In-Application-Programming*., a means to upgrade or reprogram application code in the ELMB while the ELMB remains installed in its application environment.

At the time of production of the ELMB a Bootloader is installed, called **ELMBbl**, enabling (re)programming of the ELMB microcontroller via the CAN-bus using the *CANopen* protocol [2] [3]. This document describes this Bootloader.

The ATmega128 fuses have been set such that the Bootloader is the active application after a power-up, thus ensuring that it is always possible to reprogram the application flash memory. In addition, the ATmega128 fuses are set such that the Bootloader cannot reprogram the boot-loader section (i.e. itself), thus guaranteeing the Bootloader program code can never be overwritten when using the CAN-bus for reprogramming (to replace the Bootloader itself a Programmer must be connected to the ELMB).

A PC host loader program (described elsewhere) to support ELMB upgrading or reprogramming can be obtained from the ELMB website.

2 Description

At power-up it is the Bootloader that becomes active first; it reports its presence by sending the following *CANopen* Emergency message:

Bootloader → Host

COB-ID	Byte 0-1	Byte 2	Byte 3-7
080h + <i>NodeID</i>	Emergency Error Code (00h 50h)	Error Register (Object 1001h) (80h)	Manufacturer specific error field (FEh 01h 28h ZZh 00h) (ZZh = <i>MCUCSR</i>)

(*MCUCSR* = MCU Control and Status Register; for details see section 5 and the ATmega128 datasheet [1]).

Having the Bootloader activated at power-up guarantees it is possible to download new application software to the ELMB under all circumstances, even when the current application programmed in the ELMB is faulty or corrupt (for example, when only part of the code was downloaded when the host loader program crashed).

After a power-up, if the Bootloader detects a user application is present in the application flash memory, it automatically jumps to the application after about 4 s.

The Bootloader does not execute an automatic jump to the application if it receives a valid *CANopen* SDO message accessing *Object Dictionary* index 1F50h, within those 4 s.

The Bootloader also does *not* jump to the application if it was started other than by power-up, i.e. when the user application explicitly made a jump to the Bootloader (as supported by the default ELMB user application software **ELMBio**, see [4]).

The reset vector of the user application is stored in flash bytes 0 to 3 with address 2 and 3 containing the word address of the start location of the application which must be smaller than F000h, due to the 8 kByte section of the flash memory reserved for the Bootloader. Hence the Bootloader detects the presence of an application by checking flash byte address 3 for a value smaller than F0h.

The Bootloader jumps immediately to the user application (if detected), if it receives an *CANopen* NMT *Reset-Node* message. Similarly, if a (re)programming procedure is finished, the reset message will start the newly programmed user application.

On reception of a reset message, if the Bootloader does *not* detect an application it remains the active application and transmits the following *CANopen* Emergency message, signalling that the Bootloader did not find a valid user application in flash to jump to:

Bootloader → Host

COB-ID	Byte 0-1	Byte 2	Byte 3-7
080h + <i>NodeID</i>	Emergency Error Code (00h 60h)	Error Register (Object 1001h) (80h)	Manufacturer specific error field (FEh AAh AAh 00h 00h)

When the Bootloader jumps to the user application, the ATmega128 MCUCSR register contents (containing bits signalling the cause(s) of a reset) are preserved, for the user application to inspect.

The Bootloader provides read and write instructions for both flash and EEPROM memory, as well as fuses, lock bits and signature bytes.

The Bootloader receives programming instructions through *Object Dictionary* index 1F50h, by means of *CANopen* SDO messages (for details see [3]). The individual instructions are listed in the next section.

The CAN node identifier *NodeID*, as shown in the messages above, is set using the ELMB's onboard DIP-switches. The node identifier can be set between 1 and 63 (has to be unique on the CAN-bus the board is connected to), using 6 of the 8 switches, and a CAN-bus baudrate of 50, 125, 250 or 500 kbit/s, using the 2 remaining switches. See Figure 1 below for details.

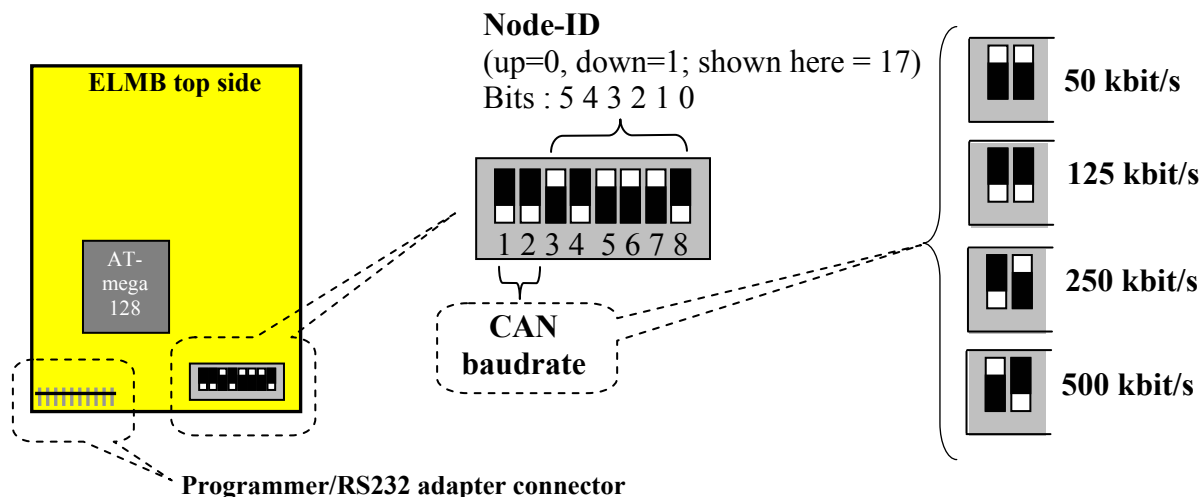


Figure 1. Location and function of ELMB DIP-switches and programming connector.

With Bootloader **ELMBbl** version 1.3, the possibility of a remotely configurable node identifier has been added, by allowing an alternative node identifier, stored in ATmega128 EEPROM, address 107h. A user application may implement the configurable node identifier feature, but *must* store and retrieve its configured node identifier (which must be in the range between 0 and 127) in ATmega128 EEPROM, address 107h.

When **ELMBbl** at startup detects a valid node identifier in this EEPROM location it will use it, instead of the node identifier set by the DIP-switches. The baudrate setting is not affected.

3 Programming Instructions

The programming instructions the **ELMBbl** Bootloader understands, have been made compatible with the Serial Programming Instruction Set of the ATmega128 [1] to a large extent¹. Some of the instructions in this Instruction Set however do not apply to the ATmega128 Bootloader mechanism of Self-Programming. Also, for efficiency reasons a number of (custom) instructions have been added. This section provides an overview of the available programming instructions.

All of the instructions are issued by writing to *Object Dictionary* index 1F50h, subindex 1. Object 1F50h, subindex 2 is used to efficiently write 4 bytes (2 program instruction words) at a time to the (temporary) Page Buffer in the ATmega128. Subsequent write accesses to 1F50h, subindex 2, store instruction words to consecutive addresses in the Page Buffer. It takes 64 such write accesses to fill a page. A *Write Flash Page* instruction must follow these 64 write instructions to actually write the page buffer content to the flash memory. The *Page Address* has been set in a previous *Set Page Address* instruction.

See Table 2 below for a more detailed overview of the available instructions.

Note the following points:

- Setting the ATmega128 fuse bits using the Bootloader is not possible.
- “*Load Flash...*” instructions must have been preceded by a “*Set Page Address*” instruction.
- The “*Load Flash Double Word*” instruction is the preferred (because most efficient) way of downloading new code to the ELMB; the other “*Load Flash...*” instructions are present mostly for reasons of compatibility.
- All instructions are issued by writing to the *Object Dictionary* (OD), even when the actual 'programming' instruction is a *read* operation (such as reading a byte from flash or EEPROM); in the *CANopen* standard an SDO reply to an OD write access does not contain data apart from the SDO protocol; for efficiency reasons however, the last data byte of the SDO reply message is used to return the data byte that is the result of a Bootloader read operation !
- However, 'reading' return values can also be done conform the *CANopen* standard, i.e. by reading *OD* entries: read index 1F50h, sub 1 to read the last byte returned by a programming instruction, read index 1F50h, sub 2 to read the last 4 bytes returned. See Table 3. (this feature has been added in **ELMBbl version 1.2** and up).

¹ This has been done to remain compatible with an earlier version of the ELMB with an ATmega103 microcontroller where the Bootloader function was handled by a second onboard microcontroller, which used the Serial Programming interface and instruction set to program the ATmega103.

Instruction	Instruction Format				Comments
	Byte 0	Byte 1	Byte 2	Byte 3	
OD index 1F50, subindex 1 (by <i>SDO Expedited Transfer</i> write)					
Programming Enable	AC	53	00	00	For compatibility reasons only
Page Erase	AC	80	00	00	Erase page set by <i>Set Page Address</i>
Read Flash Low Byte	20	a_{MSB}	a_{LSB}	*	a = 16-bit word address
Read Flash High Byte	28	a_{MSB}	a_{LSB}	*	a = 16-bit word address
Load Flash Low Byte ¹	40	00	a	d	a = 7-bit page buffer word addr d = data byte
Load Flash High Byte ¹	48	00	a	d	a = 7-bit page buffer word addr d = data byte Low byte must be loaded before high byte within the same address
Write Flash Page	4C	a_{MSB}	a_{LSB}	00	a = 16-bit page word address (least significant 7 bits are don't care)
Read EEPROM Byte	A0	a_{MSB}	a_{LSB}	*	a = 12-bit byte address
Write EEPROM Byte	C0	a_{MSB}	a_{LSB}	d	a = 12-bit byte address d = data byte
Read Lock Bits	58	00	00	*	
Write Lock Bits	AC	FF	00	l	l = lock bits (only the Bootlock bits can be programmed: bits 2-5; other bits are don't care)
Read Signature Byte	30	00	i	*	i = 0, 1, 2; returns 1E, 97, 02 resp.
Read Fuse Bits	50	00	00	*	
Read Fuse Extended Bits	50	08	00	*	
Read Fuse High Bits	58	08	00	*	
Set Page Address	D1	a_{MSB}	a_{LSB}	00	a = 16-bit page word address (least significant 7 bits must be zero)
Load Flash Word ¹	D2	a	d_{lo}	d_{hi}	a = 7-bit page buffer word addr d = data bytes
OD index 1F50, subindex 2 (by <i>SDO Expedited Transfer</i> write)					
Load Flash Double Word	d_0	d_1	d_2	d_3	d_x = data bytes

Table 2. ELMBbl Bootloader Programming Instructions.

* : the data byte resulting from the read operation is returned as byte 7 in the SDO reply message.

¹ Preferably use the *Load Flash Double Word* instruction !

Instruction	Reply Data Bytes				Comments
	Byte 0	Byte 1	Byte 2	Byte 3	
OD index 1F50, subindex 1 (by <i>SDO Expedited Transfer</i> read)					
Read byte return value	r	00	00	00	r = data byte is result of last 'read' instruction
OD index 1F50, subindex 2 (by <i>SDO Expedited Transfer</i> read)					
Read byte return value	r_0	r_1	r_2	r_3	r_x = data bytes are results from last 4 'read' instructions (oldest first)

Table 3. ELMBbl Bootloader Programming 'read' Instruction return values.

4 Object Dictionary

The CANopen Object Dictionary (OD) of the **ELMBbl v1.2** (and up) Bootloader application is listed in Table 4 below.

Communication Profile Area						
Index (hex)	Sub Index	Description	Data/Object	Attr	Default	Comment
1000	-	Device type	U32	RO	00000000h	Meaning: no CANopen profile supported
1001	-	Error register	U8	RO	0	<i>Not supported</i>
1002	-	Manufacturer status reg	U32	RO	0	¹ (see footnote)
100A	-	Manufacturer software version	VisStr	RO	"BL11"	ELMBbl application version 1.1
1F50		Download Program Data	Array			
	0	Number of entries	U8	RO	2	
	1	Instructions 1	U32	RW		See section 3 for details
	2	Instructions 2	U32	RW		See section 3 for details

Table 4. ELMBbl v1.2 (and up) CANopen Object Dictionary.

¹ Manufacturer Status Register bits:

04000000: a compile option was used: only DIP-switch 1 is used to set the baudrate (125 or 250 kBaud); other 7 switches are used for setting the Node-ID between 1 and 127; when this option is not set a 6-bit Node-ID is used and 2 bits are used for baudrate selection.

5 Emergency Objects

Emergency messages are triggered by the occurrence of an internal (fatal) error situation. An emergency CAN-message has the following general syntax:

ELMB → Host

COB-ID	Byte 0-1	Byte 2	Byte 3-7
080h + <i>NodeID</i>	Emergency Error Code	Error Register (Object 1001h)	Manufacturer specific error field

The following Emergency messages may be generated by the **ELMBbl** Bootloader application (note that byte 2 containing the Error Register is not included in the table; here it is equal to 80h in all cases):

Error Description	Emergency Error Code (byte 0-1)	Manufacturer-Specific Error Field (byte 3-7)
Bootloader is now in control	5000	Byte 3: FE Byte 4: 01 Byte 5: 28 Byte 6: microcontroller MCUCSR register contents ¹ Byte 7: 00
Bootloader cannot jump to application: invalid	6000	Byte 3: FE Byte 4: AA Byte 5: AA Byte 6: 00 Byte 7: 00

¹ ATmega128 *MCUCSR* register bits: **01h**: Power-On Reset, **02h**: External Reset, **04h**: Brown-Out Reset, **08h**: Watchdog Reset, **10h**: JTAG Reset, **80h**: JTAG Interface Disable

References

- [1] **8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash, ATmega128, ATmega128L**,
ATMEL product datasheet.
<http://www.atmel.com/products/AVR/>

- [2] CAN-in-Automation e.V.,
CANopen, Application Layer and Communication Profile,
CiA DS-301, Version 4.0, 16 June 1999,
<http://www.can-cia.de>

- [3] H.Boterenbrood,
CANopen, high-level protocol for CAN-bus,
Version 3.0, NIKHEF, Amsterdam, 20 March 2000,
<http://www.nikhef.nl/pub/departments/ct/po/doc/CANopen30.pdf>

- [4] H.Boterenbrood,
CANopen Application Software for the ELMB128,
Version 2.1, NIKHEF, Amsterdam, Mar 2004,
<http://www.nikhef.nl/pub/departments/ct/po/doc/ELMB21.pdf>