AVR-Bootloader

user manual

(software version 1.09)

03.07.2006

Forschungs- und Transferzentrum Leipzig e.V.
Wächterstraße 13
D-04107 Leipzig
Tel.: +49 (0)341-3076 1136
Fax: +49 (0)341-3076 1220
E-Mail: info@easytoweb.net
Internet: http://www.easytoweb.net/

In this manual are descriptions for copyrighted products that are not explicitly indicated as such. The absence of the trademark (™) and copyright (©) symbols does not imply that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this manual.

The information in this document has been carefully checked and is believed to be entirely reliable. However, FTZ Leipzig assumes no responsibility for any inaccuracies. FTZ Leipzig neither gives any guarantee nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. FTZ Leipzig reserves the right to alter the information contained herein without prior notification and accepts no responsibility for any damages which might result.

Additionally, FTZ Leipzig offers no guarantee nor accepts any liability for damages arising from the improper usage or improper installation of the hardware or software. FTZ Leipzig further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.

© Copyright 2002 Forschungs- und Transferzentrum Leipzig e.V. an der HTWK Leipzig (FH).
Rights - including those of translation, reprint, broadcast, photomechanical or similar reproduction and storage or processing in computer systems, in whole or in part - are reserved. No reproduction may occur without the express written consent from FTZ Leipzig

Forschungs- und Transferzentrum Leipzig
Udo Jakobza / Thomas Minner
Wächterstr. 13
D-04107 Leipzig
Germany
phone +49(0)341-3076-1136
fax +49(0)341-3076-1220
e-mail: info@easyToWeb.net
http://www.easyToWeb.net

Index of contents

Index of figures

Introduction

This manual to the *AVR-Bootloader* contains four basic sections. Chapter *Quick start* offers a short overview of basic things for the use of the *AVR-Bootloader*. Chapter *Example projects and general information* gives an overview of the range of example applications contained in the install file. Chapter *Project configuration* explains the adaptation of the source code to a new project. Chapter *PC software* gives a detailed description for handling the PC software used to configure and program a new project.

1 Quick start

If you want to use the given examples of a project on easyToWeb with the default values, refer to following flow chart.

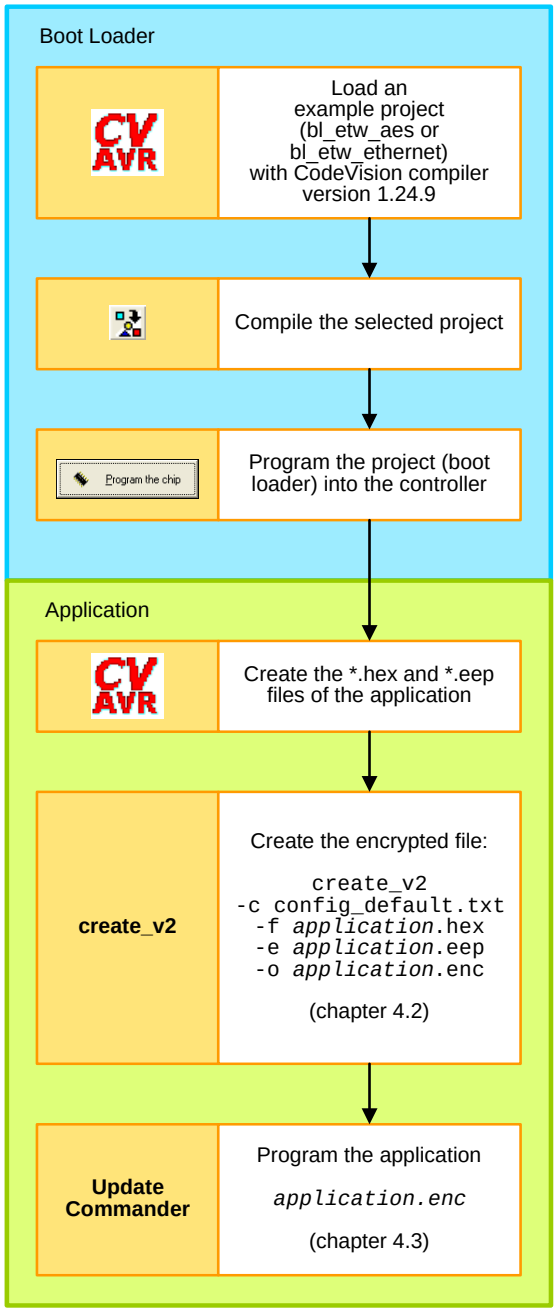

Figure 1.1 Usage of the project examples

Use the following flow chart, to adapt the boot loader to a new project.

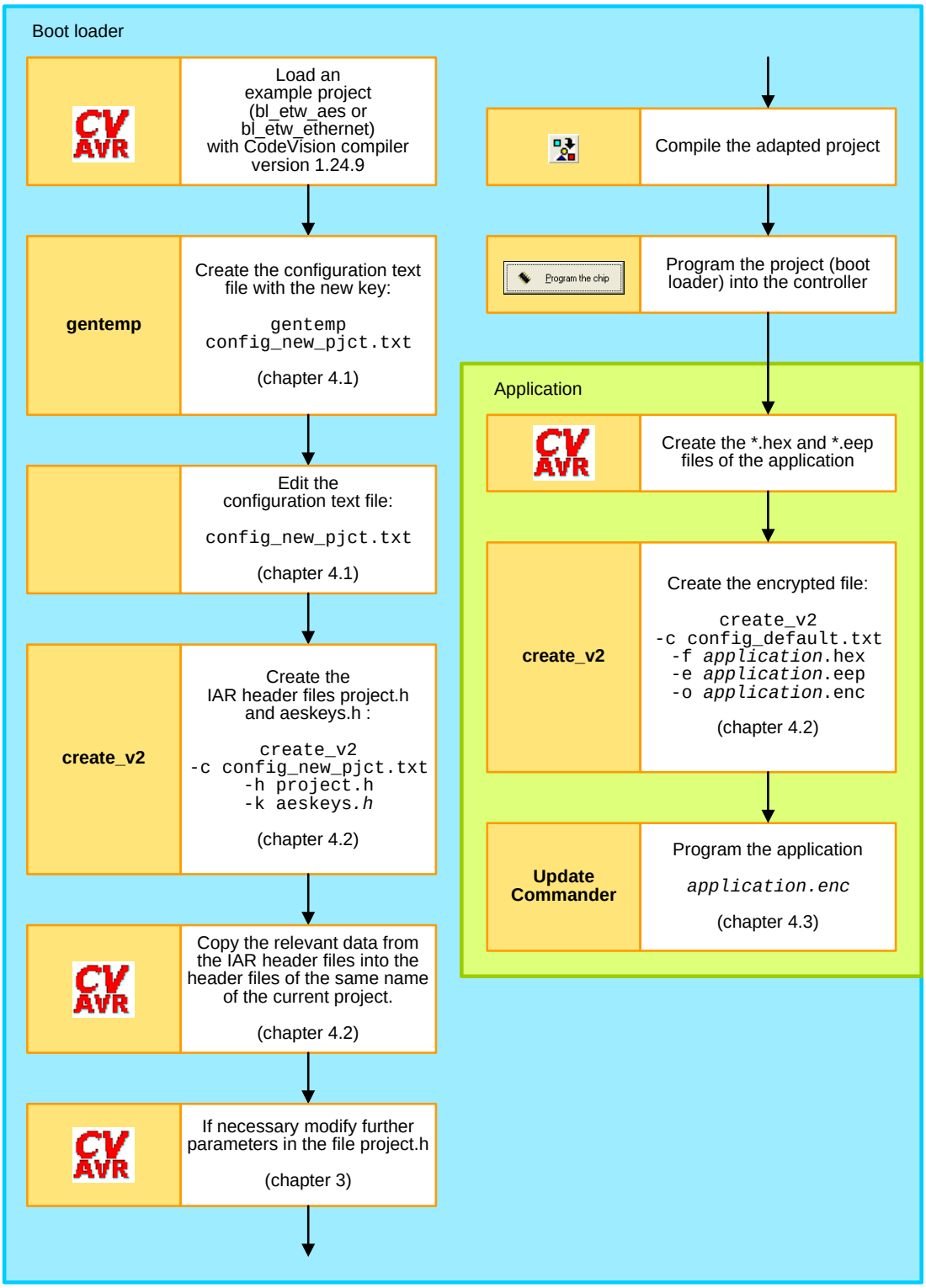

Figure 1.2 Adaptation and use on own hardware

2 Example projects and general information

The install file contains four boot loader example projects. The following overview shows the functions offered by these projects and suitable applications.

Project name	BL_ETWEVA.PRJ	BL_ETWCOM.PRJ	BL_ETWETH.PRJ	BL_ATMEGA2561V.PRJ
Optimized for hardware	easyToWeb	easyToWeb	easyToWeb	STK500
AES encryption	activated	activated	activated	activated
CRC test	activated	activated	activated	activated
Interface	serial interface	serial interface	ethernet interface	serial interface
Code size (compiled with CVAVR 1.24.9)	1928 Byte	3710 Byte	7374 Byte	3818 Byte
Required development environment	CodeVisionAVR Evaluation Version 1.24.6 and higher	CodeVisionAVR Evaluation Version 1.24.6 and higher	CodeVisionAVR Evaluation Version 1.24.6 and higher	CodeVisionAVR Evaluation Version **1.24.9** and higher

Figure 2.1 Example projects

The *AVR-Bootloader* was tested on different controllers. The figure shows the modules which can be integrated into the several controller types.

Interface	UART		Ethernet		
Encryption	none	AES	none	AES	Maximum boot sector size
ATmega8	1476 Byte	---	---	---	2048 Byte
ATmega16	1850 Byte	---	---	---	2048 Byte
ATmega32	1852 Byte	3612 Byte	---	---	4096 Byte
ATmega128	1928 Byte	3710 Byte	5432 Byte	7374 Byte	8192 Byte
ATmega2561V	2046 Byte	3818 Byte	untested	untested	8192 Byte

Tested with CodeVisionAVR 1.24.9

Figure 2.2 Tested controllers

Note: You can only create and program the ATmega 2561V bootloader without mistakes by using the CodeVision version 1.24.9.

In principle the *AVR-Bootloader* can be adapted to applications using following controllers. The source code is configured for these controllers.

Maximum boot sector size	Controllers
2 KByte	ATmega8(L), ATmega88(V), ATmega8515(L), ATmega8535(L)
	ATmega16(L), ATmega168(V), ATmega162(V), ATmega165(V), ATmega169(V)
4 KByte	ATmega32(L), ATmega325(V), ATmega3250(V), ATmega329(V), ATmega3290(V)
8 KByte	ATmega64(L), ATmega640(V) ATmega644(V), ATmega645(V), ATmega6450(V), ATmega649(V), ATmega6490(V)
	ATmega128(L), ATmega1280(V) ATmega1281(V)
	ATmega2560(V), ATmega2561(V)

Figure 2.3 Controllers with boot loader support

3 Project configuration

In order to use the Bootloader in a new application the compiler macros of the project configuration file *project.h* have to be adapted to the project. This chapter describes the necessary steps for the porting. All macros coloured in red should be parameterized with the PC software *create_v2* (→ chapter PC application – *create_v2*).

3.1 Main macros

These macros are used for the basic configuration of the boot loader. Note that the use of the AES communication protocol for the data communication requires the CRC16_ENABLE macro to be activated. The possibility to debug the program is supported only by the boot loader version working with the UART interface. This is because of the limited boot sector capacity.

Function	Data communication via the UART interface	Data communication via network
Activate the boot loader module	**BOOTLOADER**	**BOOTLOADER**
Activate the communication protocol	**BL_AES_PROTOCOL**	**BL_AES_PROTOCOL**
Activate the CRC functions	**CRC16_ENABLE**	**CRC16_ENABLE**
Activate the application test	**BL_CRC_CHECK**	**BL_CRC_CHECK**
Activate a encrypted data communication	**AES_ENABLE**	**AES_ENABLE**
Activate the interface	**BL_UART_ENABLE**	**BL_UDP_ENABLE**
Activate the debug output	**DEBUG**	

Figure 3.1 Main macros

3.2 Project macros

With the help of these macros information about the project version number and the attached equipment are available to the user over the debug terminal or in the PC software *Update Commander*. The value of the device signature is equal to the second byte of the AES_SIGNATURE.

Function	Data communication via the UART interface	Data communication via network
Define the project name	**DEBUG_PROJECT_NAME** “AVR-BOOTLODER“	**DEBUG_PROJECT_NAME** **“AVR-BOOTLODER“**
Indicate the project version as character string	**DEBUG_PROJECT_VERSION** “x.xx“	**DEBUG_PROJECT_VERSION** “x.xx“
Indicate the controller typ as character string	**DEBUG_MCU_TYP_STRING** “ATmegaxxx“	**DEBUG_MCU_TYP_STRING** “ATmegaxxx“
Indicate the high bytes of the project version number	**PROJECT_VERSION_ NUMBER_HIGH** <0xXX>	**PROJECT_VERSION_ NUMBER_HIGH** <0xXX>
Indicate the low bytes of the project version number	**PROJECT_VERSION_ NUMBER_LOW** <0xXX>	**PROJECT_VERSION_ NUMBER_LOW** <0xXX>
Define the device signature	**DEVICE_SIGNATURE** <0xXX>	**DEVICE_SIGNATURE** <0xXX>

Figure 3.2 Project macros

3.3 Hardware configuration

You can configure the hardware by using these macros. The macro BL_TIMER_COUNT_DOWN defines the time between system start and start of the application. In this time window it is possible for the PC software to initialize a programming process.

Function	Data communication via the UART interface	Data communication via network
Activate the status LEDs	**BL_STATUS_LED**	**BL_STATUS_LED**
Status LEDs are high activ	**BL_STATUS_LED_HIGH_ACTIV**	**BL_STATUS_LED_HIGH_ACTIV**
Status LEDs are low activ	**BL_STATUS_LED_LOW_ACTIV**	**BL_STATUS_LED_LOW_ACTIV**
Select the status LED port	**BL_STATUS_LED_PORT** <PORTx>	**BL_STATUS_LED_PORT** <PORTx>
Select the status LED DDR register	**BL_STATUS_LED_DDR** <DDRx>	**BL_STATUS_LED_DDR** <DDRx>
Define the init LEDs	**BL_STATUS_INIT_LED** <0...7>	**BL_STATUS_INIT_LED** <0...7>
Define the update LEDs	**BL_STATUS_UPDATE_LED** <0...7>	**BL_STATUS_UPDATE_LED** <0...7>
Configure the 16 bit timer prescaler	**BL_TIMER_PRESCALER** <0,1,8,64,256,1024>	**BL_TIMER_PRESCALER** <0,1,8,64,256,1024>
Define the timeout in ms	**BL_TIMER_WAIT_PERIOD_MS** <X>	**BL_TIMER_WAIT_PERIOD_MS** <X>

Figure 3.3 Hardware configuration

3.4 Header file path

These macros contain the relative paths to the header files which have to be included in the respective boot loader configuration.

Module	Data communication via the UART interface	Data communication via network
Bootloader	**BOOTLOADER_H_FILEPATH** "..\include\bootloder.h"	**BOOTLOADER_H_FILEPATH** "..\include\bootloder.h"
Hardware	**HARDWARE_H_FILEPATH** "..\include\hardware.h"	**HARDWARE_H_FILEPATH** "..\include\hardware.h"
CRC	**CRC_H_FILEPATH** "..\include\crc.h"	**CRC_H_FILEPATH** "..\include\crc.h"
AES	**AES_H_FILEPATH** "..\include\aes.h"	**AES_H_FILEPATH** "..\include\aes.h"
	AES_KEYS_H_FILEPATH "..\include\aeskeys.h"	**AES_KEYS_H_FILEPATH** "..\include\aeskeys.h"
SPM	**SPM_H_FILEPATH** "..\include\spm.h"	**SPM_H_FILEPATH** "..\include\spm.h"
Interface	**LOADER_UART_H_FILEPATH** "..\include\loader_uart.h"	**UDP_H_FILEPATH** "..\include\udp_bl.h"
		DEVICE_H_FILEPATH "..\include\device_bl.h"
		ETHERNET_H_FILEPATH "..\include\ethernet_bl.h"
		BUFFER_H_FILEPATH "..\include\buffer_bl.h"
		IP_H_FILEPATH "..\include\ip_bl.h"
		ARP_RARP_H_FILEPATH "..\include\arp_rarp_bl.h"
		CS8900_H_FILEPATH "..\include\cs8900.h"
Debug	**BL_DEBUG_H_FILEPATH** "..\include\bl_debug.h"	
	AES_DEBUG_H_FILEPATH "..\include\aes_debug.h"	
	DEBUG_H_FILEPATH "..\include\debug.h"	
	DEBUG_DRIVER_H_FILEPATH "..\include\debug_driver.h"	
	UTIL_H_FILEPATH "..\include\util.h"	

Figure 3.4 Header file path

3.5 AES

Using these macros the encryption can be parameterized. The parameters key length, block signature, data buffer size and initial vector have to be identical to the values which are saved in the configuration text file (→ chapter PC software).

Function	Data communication via the UART interface	Data communication via network
Define the key length 1-128 Bit, 2-192 Bit, 3-256 Bit	**AES_KEY_COUNT** <X>	**AES_KEY_COUNT** <X>
Define the block signature	**AES_SIGNATURE** <0xXXXXXXXX>	**AES_SIGNATURE** <0xXXXXXXXX>
Define the data block buffer size	**AES_BUFFER_SIZE** <XXX>	**AES_BUFFER_SIZE** <XXX>
Define the initial vector	**AES_INITIALVECTOR_3** <0xXXXXXXXX>	**AES_INITIALVECTOR_3** <0xXXXXXXXX>
	AES_INITIALVECTOR_2 <0xXXXXXXXX>	**AES_INITIALVECTOR_2** <0xXXXXXXXX>
	AES_INITIALVECTOR_1 <0xXXXXXXXX>	**AES_INITIALVECTOR_1** <0xXXXXXXXX>
	AES_INITIALVECTOR_0 <0xXXXXXXXX>	**AES_INITIALVECTOR_0** <0xXXXXXXXX>

Figure 3.5 AES macros

3.6 CRC

These macros are used for the configuration of the CRC test. The CRC polynomial and the memory location of the lookup table need to be defined. The CRC check is used to test the received data blocks and the entire application.

Function	Data communication via the UART interface	Data communication via network
Define the test polynomial	**CRC16_POLYNOM** <0xXXXX>	**CRC16_POLYNOM** <0xXXXX>
Define memory location of the look up table	**CRC16_LOOKUP_RAM**	**CRC16_LOOKUP_RAM**

Figure 3.6 CRC macros

3.7 Interface

These macros allow the configuration of the data communication.

Function	Data communication via the UART interface	Data communication via network
Set the UART baud rate	**BL_UART_BAUD_RATE** <X>	
Define the using UARTs (only necessary if several available)	**BL_UART0..4**	
Activate the Ethernet module		**ETHERNET_ENABLE**
Activate the ARP module		**ARP_ENABLE**
Activate the IP module		**IP_ENABLE**
Activate the UDP module		**UDP_ENABLE**
Activate the DS2430 module		**DS2430_ENABLE**
Activate the Ethernet controller		**DEVICE_CS8900**
Activate the control of the Ethernet controller via memory bus		**CS8900_MEMORY**
Specify the control of the memory bus		**BL_BOARD_ETW**
Define the default IP address		**IP_DEFAULT** <{XXX,XXX,XXX,XXX}>
Define the default MAC address		**MAC_DEFAULT_1** <0xXX>
		MAC_DEFAULT_2 <0xXX>
		MAC_DEFAULT_3 <0xXX>
		MAC_DEFAULT_4 <0xXX>
		MAC_DEFAULT_5 <0xXX>
		MAC_DEFAULT_6 <0xXX>
Define the size of the data division of the UDP TX block		**UDP_TX_DATA_SIZE** <XXX>
Define the size of the data division of the UDP RX block		**UDP_RX_DATA_SIZE** <XXX>
Define the UDP port number		**UDP_PORT_NUMBER** <XXXXX>

Figure 3.7 Interface macros

3.8 Debug

With these macros different debug modules can be activated. The boot loader version with network interface does not support these debug modules because of capacity reasons.

Function	Data communication via the UART interface	Data communication via network
Activate the debug outputs of the communication protocol	**BL_DEBUG**	
Activate the debug outputs of the AES encryption	**AES_DEBUG**	
Activate the debug outputs of the CRC test	**CRC16_DEBUG**	
Define the debug interface	**DEBUG_UPLINK_UART0** or **DEBUG_UPLINK_UART1**	
Set the baud rate of the debug interface	**DEBUG_UART_BAUD_RATE** <X>	

Figure 3.8 Debug macros

4 PC software

This chapter gives a detailed description of functionality and use of the PC programs necessary for controller programming.

The tasks to be achieved relating to encrypted programming. Three different programs ensure. The program *gentemp* generates a configuration text file for the encryption. To create the encrypted and block-structured file use the program *create_v2*. The file is transmitted to the controller using the program *Update Commander*.

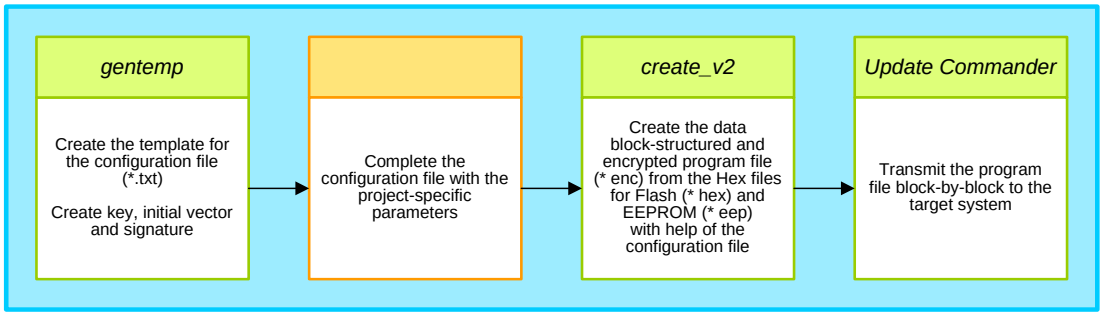

Figure 4.1 Configurations steps up to programming

The previous figure shows the sequence using the programs. Following subchapters give a detailed description of the program tasks.

4.1 PC application – *gentemp*

With the help of the program *gentemp* a template for a configuration text file is created. This file is needed afterwards to create the encrypted file.

In order to create a template enter following command into the DOS prompt:

```
gentemp      config_project_name.txt
```

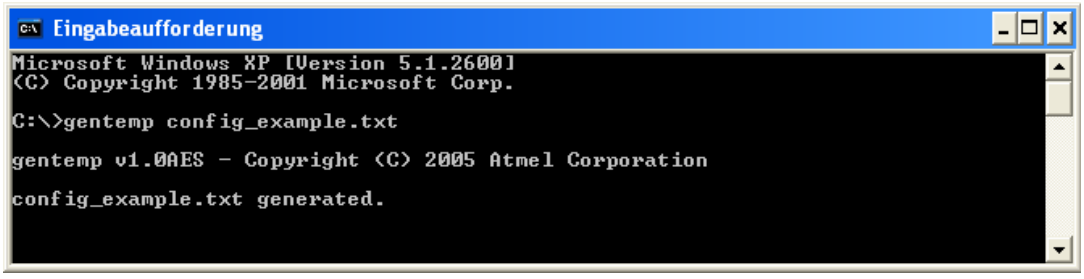

Figure 4.2 Execution of *gentemp*

When executing this command from the location of the program *gentemp*, the template represented in the following figure 4.3 will be created.

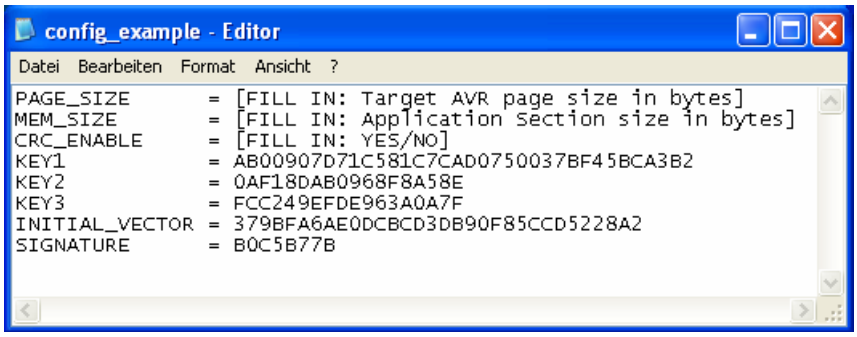

Figure 4.3 Config_example.txt

Now this template needs to be configured with the parameters of the appropriate project. The values for the memory capacity of the application sector and the page size can be taken from the figure 4.4. The key, the initial vector and the block signature which are contained in the template were generated by the program. They values are different with each execution of the program. These generated values can also be changed of course. However, a modification of these values can affect the security of the encryption negatively. Depending on desired key length the lines KEY3 (key length 192 bits) or additionally KEY2 (key length 128 bits) can be removed from the file. If no encryption is desired the lines KEY1-KEY3 and INITIAL_VECTOR must be removed from the template. If you want to achieve a CRC test before the application starts, the macro CRC_ENABLE must be set.

	Size of the application sector	Page size
ATmega8	6144 Byte	64 Byte
ATmega16	14336 Byte	128 Byte
ATmega32	28672 Byte	128 Byte
ATmega128	122880 Byte	256 Byte

Figure 4.4 Size for ATmega controllers in detail

4.2 PC application – *create_v2*

When developing an application with the development environment of CodeVision the compiler generates a Hex file with the flash program code and a Hex file with the EEPROM data for programming. Before programming the developed application with the help of the boot loader a data record and block-structured file has to be created from these Hex files. This task is managed by the program create_v2 with the help of the project-specific configuration text file. Depending on configuration this program also handles encryption as well as computation of the CRC test value of the data blocks.
Additionally the program integrates information into the target file to set certain boot lock bits if necessary.

Creating a program code file is done by entering following command into the DOS prompt:

```
create_v2   –c config_project_name.txt -e eeprom.eep –f flash.hex
            –o destination_file.enc –l BLB02
```

If the application does not contain EEPROM data or if no boot lock bits need to be programmed, leave out the relating parameters.

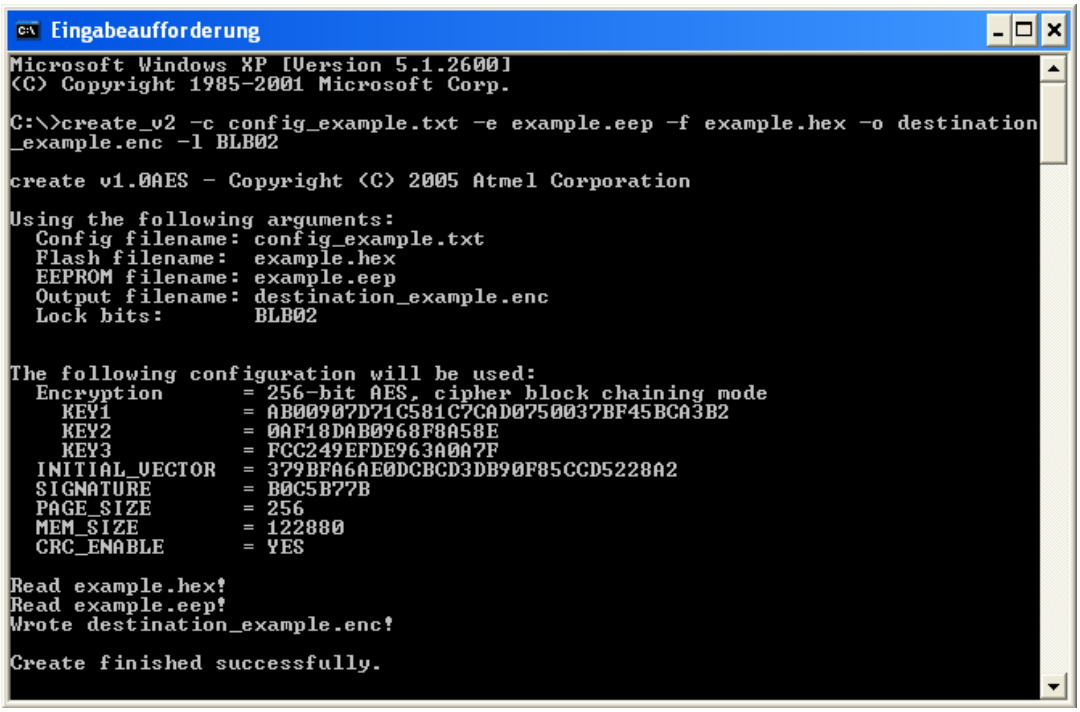

Figure 4.5 Execution of *create_v2*

After the message "Create finished successfully" appeared in the DOS prompt the data block-structured and if necessary encrypted program file has been created and can be used for programming. The created program file is a binary file containing data completely ASCII coded.
Also files containing flash program code and EEPROM data which was generated by compilers of other development environments can be used for the creation of the program code file. Note that the used files must contain data in the Intel Hex format. It is not possible for the program *create_v2* to process files of other formats. The EEPROM data file generated by the CodeVision compiler does not have the ending *.hex, though its content is structured in the Intel Hex format, too.

Beside this the program *create_v2* can create the header files *aeskey.h* and *project.h* of the controller firmware for the IAR compiler. On execution essential macros are created. The parameters of the macros can simply be copied into the projects header files of the same name.

The following operational sequence is used:

1. Create the header files:

```
create_v2   –c config_project_name.txt -h project.h –k aeskeys.h
```

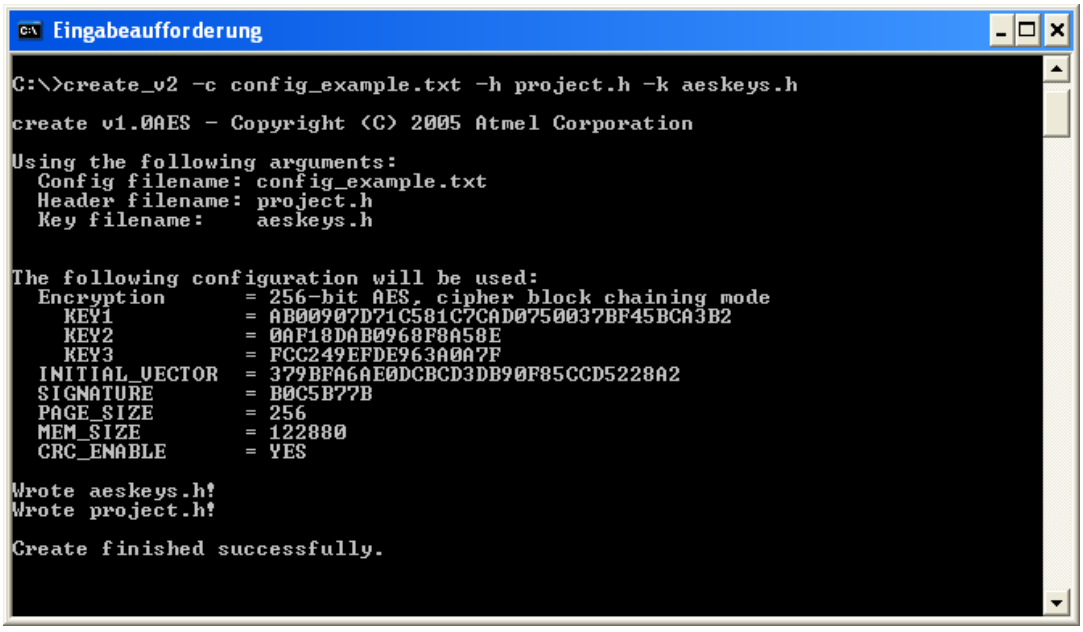

Figure 4.6 Creation of the IAR header files

2. Copy the certain parameters of the generated project.h into the file *project.h* of the current project:

generated IAR project.h	project.h
CRC_CHECK	BL_CRC_CHECK
SIGNATURE	AES_SIGNATURE
BUFFER_SIZE	AES_BUFFER_SIZE
INITIALVECTOR_3	AES_INITIALVECTOR_3
INITIALVECTOR_2	AES_INITIALVECTOR_2
INITIALVECTOR_1	AES_INITIALVECTOR_1
INITIALVECTOR_0	AES_INITIALVECTOR_0
KEY_COUNT	AES_KEY_COUNT

Figure 4.7 Correspondence table

3. Copy the generated array from the file *aeskey.h* into the file of the same name of the current project.

IAR notation	CodeVision notation

__**farflash unsigned char** kTable[32] =
{
0xab, 0x01, 0x41, 0xeb, 0x1c, 0xb0, 0x71, 0xe5,
0xd0, 0xea, 0x00, 0xbd, 0xf4, 0xb7, 0x28, 0xd9,
0x0a, 0xe3, 0x36, 0x58, 0x96, 0x1f, 0x29, 0xc7,
0xfc, 0x84, 0x27, 0x7e, 0xe9, 0xc7, 0x82, 0x3f,
};
	flash unsigned char kTable[32] =
	{
	0xab, 0x01, 0x41, 0xeb, 0x1c, 0xb0, 0x71, 0xe5,
	0xd0, 0xea, 0x00, 0xbd, 0xf4, 0xb7, 0x28, 0xd9,
	0x0a, 0xe3, 0x36, 0x58, 0x96, 0x1f, 0x29, 0xc7,
	0xfc, 0x84, 0x27, 0x7e, 0xe9, 0xc7, 0x82, 0x3f,
	};

Figure 4.8 Key declaration

Important: Note, that copying the key from the configuration text file into the array of the projects *aeskeys.h* causes errors because the notation of the keys in the files is different. (figure 4.3)

4.3 PC application – *Update Commander*

The program *Update Commander* has the purpose to transmit the program code file block-by-block to the target system. The program allows to transfer the data via RS-232, USB (USB to UART Bridge) and via Ethernet to the target system.

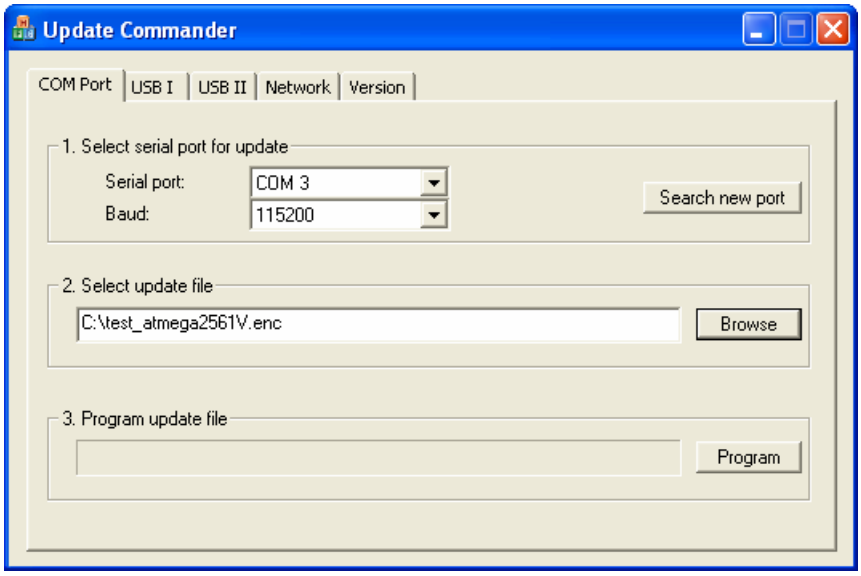

Figure 4.9 Update Commander window

➢ Classification of the modules of the *Update Commanders*:

- COM port: Module for programming of devices with a RS232 interface or a USB interface, configured as COM port
- USB I: Module for programming of devices with a USB interface based on a FTDI chip
- USB II: Module for programming of devices with a USB interface based on Silabs chip
- Network: Module for programming of devices via Ethernet, transfer of the data with UDP/IP

Subsequently, the description of the use of the modules COM port and network is given.

Programming via RS232:

At first select the PC COM port used for data communication and the baud rate. If the desired port does not appear in the selection list, press the button *search new port*. If this should not succeed examine the desired port not to be used already.

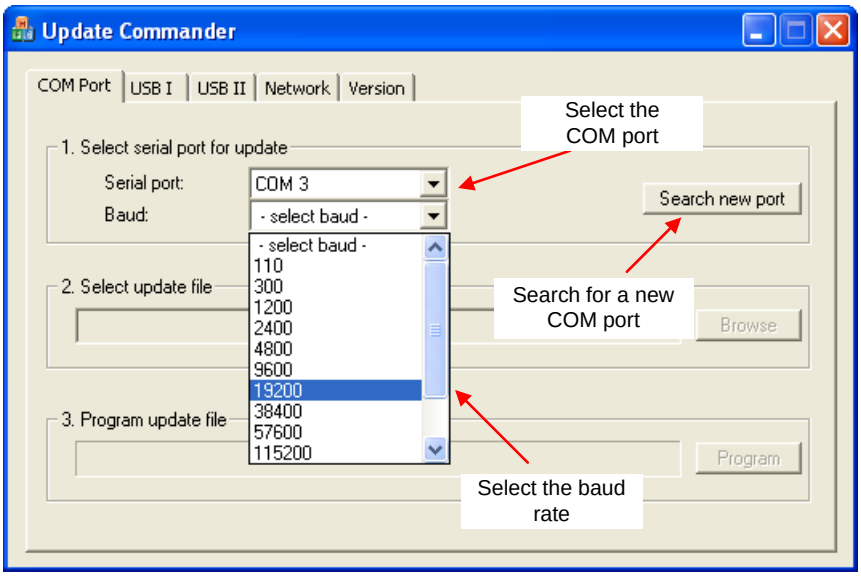

Figure 4.10 Select the COM port

The second step is to select the file to be programmed. After this selection the programming process can be executed by pressing the *program* button.

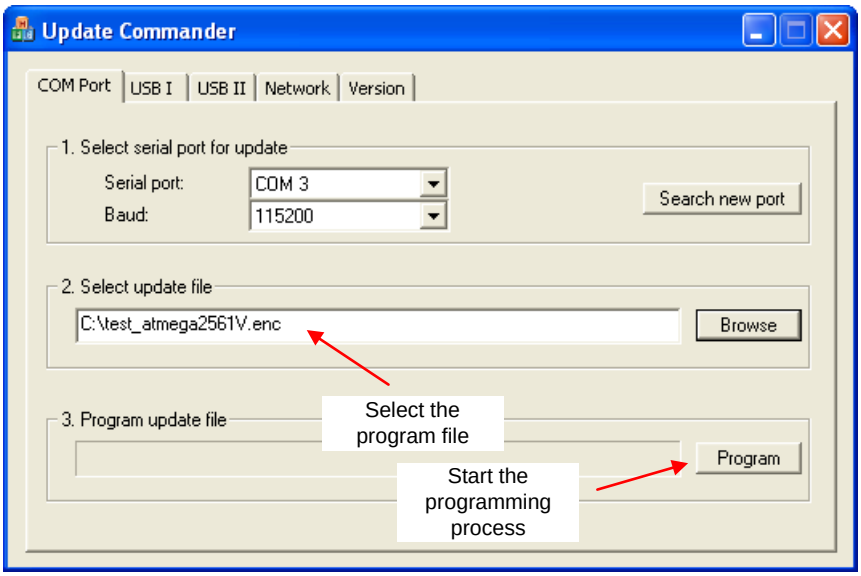

Figure 4.11 Select file – launch programming

During the programming process the progress of the programming is displayed. In the lower right corner of the program window the boot loader version number and the designation of the attached device is shown.

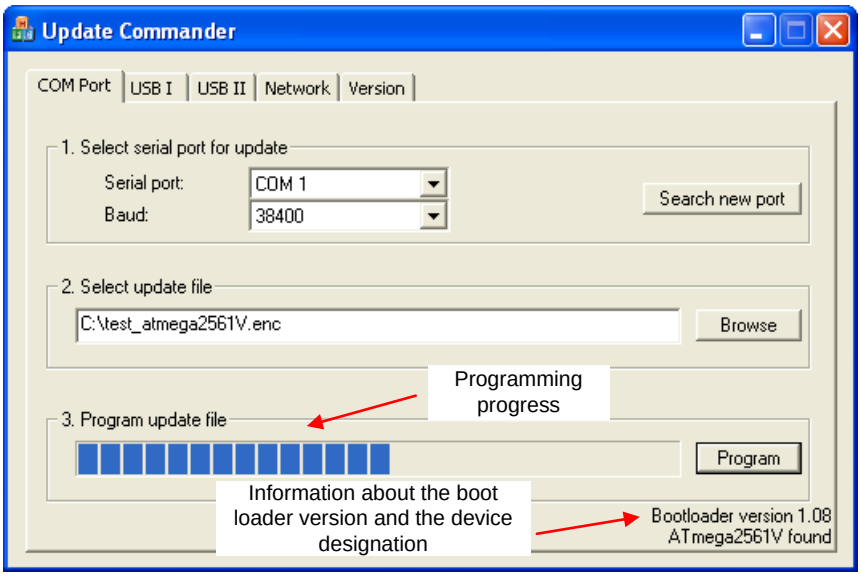

Figure 4.12 Programming state – COM

Programming via Ethernet:

When programming via network first enter the IP address of the device to be programmed into the designated field. Afterwards select the file to be programmed and press the *program* button.

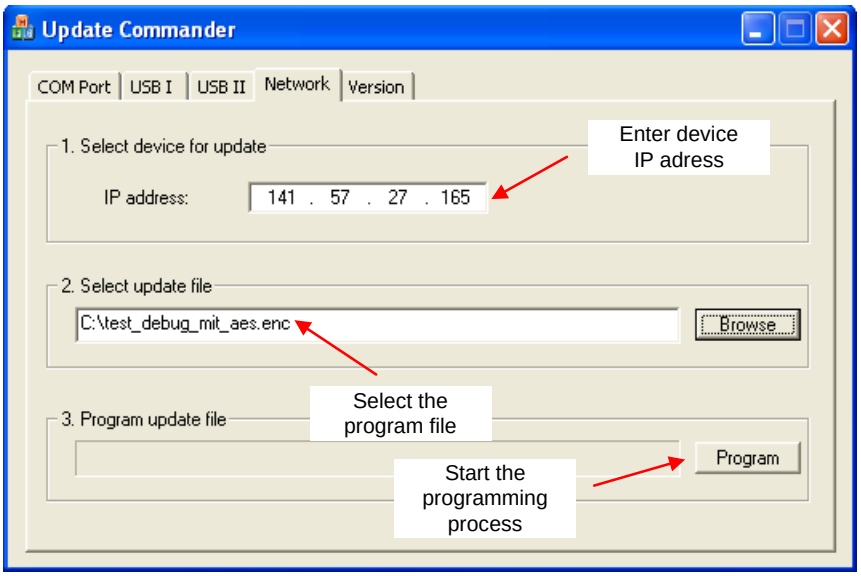

Figure 4.13 IP address choice and programming

During this procedure also here the programming progress, the boot loader version number and the designation of the attached device are shown.

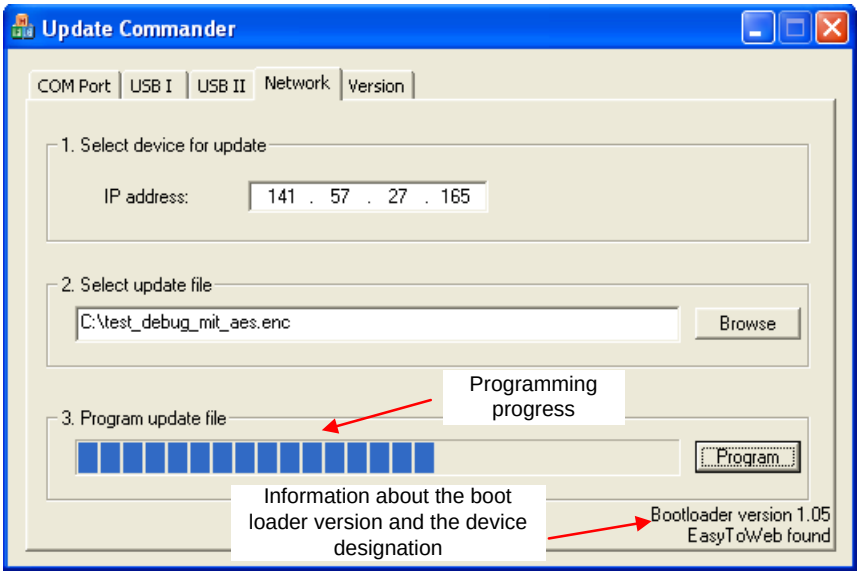

Figure 4.14 Programming state – Ethernet

- Status and error messages with programming

Any specified status and error messages apply both to programming via RS232 and to programming via Ethernet.
At following situations status and/or error messages are displayed:

Unsuccessful activation of the boot loader:

If the controller contains a valid application being currently executed and a programming process by the activation of the *program* button is started at the same time, then the status message represented in figure 4.15 will appear. Since the Bootloader is not active the controller cannot answer to the *REQUEST_SIGN*. For this reason the status message tells the user to activate the boot sector again by releasing a reset in order to continue programming. Alternatively the programming process also can be canceled by pressing the *cancel* button.

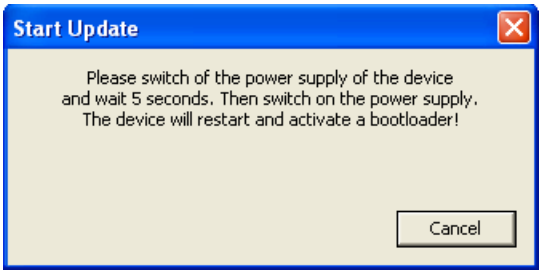

Figure 4.15 Bootloader activation

Programming success:

If programming was successfully terminated the user will receive following status message.

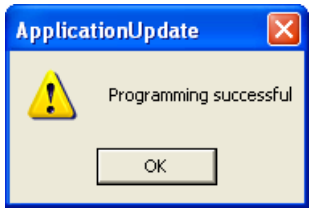

Figure 4.16 Programming successful

For the user however there is only absolute reliability for successful programming if the application starts. Errors when writing the flash are not detected within programming.
The CRC test of the flash content is passed only directly before the start of application. At this time the connection between software and boot loader is already inactive.

Programming of non valid applications:

If unintentionally anyone tries to program a device with an application not designated for it the error message represented in the figure 4.17 is displayed after transmission of the data is completed.

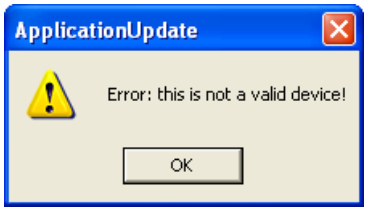

Figure 4.17 Software is not valid for this device

A non valid application is characterized either by a wrong block signature or by an unknown encryption (wrong key). Also an incorrect transmission can cause this error message. However, this case is improbable since it was already detected by the preceding CRC test.

Transmission errors:

The program *Update Commander* encounters a transfer error if the according data block analysis character is not received within a determined period of time or if a CRC error was detected.

In principle a retransmission of the concerned data block is done after detecting a transfer error. However, after detecting four sequential transfer errors the programming process is canceled. The user will receive following error message (figure 4.18).

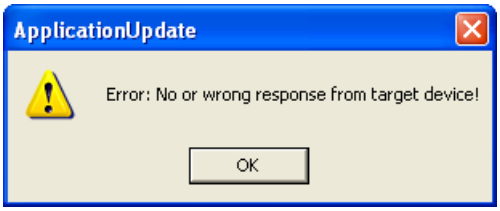

Figure 4.18 Transmission error

5 Change log

Version 1.09

==========

- 30.06.2006
- macros BL_FLASHSIZE_32BIT and BL_FLASHSIZE_16BIT renamed and moved to bootloader.h
- macros BL_INTERRUPT_VECTOR_CHANGE and BL_INTERRUPT_VECTOR_SELECT also moved to bootloader.h
- macros BL_WATCHDOG_TIMER_CHANGE and BL_WATCHDOG_TIMER_SET defined in bootloader.h
- new status LED macros:

  project.h: BL_STATUS_LED
  BL_STATUS_LED_HIGH_ACTIV
  BL_STATUS_LED_LOW_ACTIV
  BL_STATUS_LED_PORT PORTx
  BL_STATUS_LED_DDR DDRx
  BL_STATUS_INIT_LED 0..7
  BL_STATUS_UPDATE_LED 0..7
- BL_LED_PORT_ON, BL_INIT_LED_ON, BL_UPDATE_LED_ON, BL_LED_PORT_OFF moved to bootloader.h
- new timer macros in project.h: BL_TIMER_PRESCALER and BL_TIMER_WAIT_PERIOD_MS
- loader_uart.c, loader_uart.h, loader_..() functions and LOADER_.. macros renamed in bl_uart.c, bl_uart.h and so on
- code size decreased
- source code configured for the following controllers

ATMEGA8(L)	ATMEGA32(L)	ATMEGA6450(V)
ATMEGA88(V)	ATMEGA325(V)	ATMEGA649(V)
ATMEGA8515(L)	ATMEGA3250(V)	ATMEGA6490(V)
ATMEGA8535(L)	ATMEGA329(V)	ATMEGA128(L)
ATMEGA16(L)	ATMEGA3290(V)	ATMEGA1280(V)
ATMEGA162(V)	ATMEGA64(L)	ATMEGA1281(V)
ATMEGA168(V)	ATMEGA640(V)	ATMEGA2560(V)
ATMEGA165(V)	ATMEGA644(V)	ATMEGA2561(V)
ATMEGA169(V)	ATMEGA645(V)	

Version 1.08

==========

- 14.06.2006
- test project for ATmega2561V included
- baudrate selection in Update Commander added
- debug modul bugs fixed

Version 1.07b

==========

- 23.05.2006
- uart deactivation bug fixed
- aes signature comparison bug fixed

Version 1.07a

==========

- 20.03.2006
- added missing files (bl_etwcom.prj, bl_etwcom.c, bl_etwcom.txt)

Version 1.07

==========

- 15.03.2006
- added example project for evaluation version of the CodeVision development environment (_bl_etw_cvavr_eva)
- added chapter example projects and generals in the user manual

Version 1.06

==========

- 24.02.2006
- First Version