chip45boot2 Infosheet

Flexible bootloader for AVR microcontrollers with auto-baud, hexfile support and extra functions.

BASIC SPECIFICATIONS

Name	Supported Devices	Memory Usage
chip45boot2 (current release is v2.90)	ATmega8, ATmega88, ATmega88P, ATmega8535, ATmega16, ATmega168, ATmega168P, ATmega32, ATmega324P, ATmega328P, ATmega3290P, ATmega64, ATmega640, ATmega644, ATmega644P ATmega128, ATmega1280, ATmega1281, ATmega2560, ATmega2561 AT90CAN32, AT90CAN64, AT90CAN128	1k word boot block size (2048 bytes)

CHIP45BOOT2 FEATURES

- direct support of Intel Hex files
- automatic baud rate setting to 19200 baud, independent of CPU clock settings (!!!)
- baud rate adjustable during operation (9600 - 230400 baud)
- simple command line interface with any terminal program
- programming of flash and eeprom memory
- byte wise read and write access to SRAM memory and MCU registers
- byte wise read and write access to EEPROM memory
- 1k boot block (2048 bytes)
- distributed as precompiled hex files for simple usage

NEW DEVICES

If your favorate AVR controller type is not included in the above list of supported devices, send an email to info@chip45.com and the chip45 team will add the controller to the list and provide a new precompiled hexfile for free!

QUICKSTART

If you are an experienced user of Atmel AVR controllers and you are familiar with bootloader related fuse bit settings, you may start like this:

- set fusebits to 1k boot block, activate boot reset vector and disable 1/8 prescaler
- download the correct chip45boot2 hexfile to your target
- connect your target to a PC and terminal program
- set PC serial port to 19200 baud, 8N1, XON/XOFF
- hold shift-U keys pressed while powering on or resetting your target
- see the welcome message “c45b2” plus version number plus prompt on the next line
- now the bootloader is ready to accept the below described commands
- please read the note on “fragmented hexfiles” below!

USAGE

If the chip45boot2 bootloader did not come preloaded on your device, you have to program it once to your MCU with an In-System-Programming adapter (ISP-adapter). After this, further programming of the MCU can be done with the bootloader and an ISP adapter is longer needed. Suitable ISP adapters are for example the CrispAVR-USB by chip45 (www.chip45.com/CrispAVR-USB) or the AVRISP-mkII bei Atmel. Beside those there exist many third party programmers, which can be used.

PROGRAMMING OF THE BOOTLOADER

Use your favorate ISP adapter and PC software to download the chip45boot2 hexfile to the MCU. Make sure that it matches either the MCU type as well as the UART desired for communication! Beside the hexfile, also the fusebits of the AVR MCU have to be set properly. The chip45boot2 ZIP files contains several screenshots showing working fusebit settings for the respective MCUs. Some general remarks on fusebit setting are here:

Fusebits	Function
BOOTSZ	These two fusebits select the size of the flash boot block area! Chose a 2kbyte boot block, i.e. “Boot Flash size = 1024 words”, since Atmel counts flash addresses in 16 bit words (2 bytes). This also selects the boot block start address.
BOOTRST	This fusebit activates the bootloader, i.e. after a reset the MCU starts code execution not at bottom fo flash (0x0000), but at the above set boot block address. This fusebit must be set, to have the possibility to enter the bootloader after every reset.
WDTON	This fusebit enables the software watchdog timer. If your application requires a watchdog to run and you must set this fusebit, you cannot use the bootloader without code modification. The bootloader currently does not contain any code to trigger the watchdog, hence the watchdog will force a reset after start of the bootloader and timeout of the watchdog timer.
CKDIV8	This fusebit divides the clock source by eight before clocking the MCU core. Since the chip45boot2 automatically adjust its baudrate prescaler settings to a received 19200 baud character sequence, it has problems with certain too low clock settings. If using the internal 8MHz RC oscillator, the CKDIV8 bit must be disabled, since the resulting 1MHz clock is not suitable for 19200 baud communication. chip45boot2 works properly from about 4MHz up. This is not the case for baudrate-suitable clock frequencies, e.g. an external 1.8432MHz crystal works fine and thus an external 14.7456MHz crystal also works fine, even with CKDIV8 enabled, since this results in 1.8432MHz MCU clock.
SUT_CKSEL	These fusebits in combination select the clock source for the MCU core and the startup time after a reset. Most of the newer AVR controllers come with the internal 8MHz RC oscillator selected as clock source and have CKDIV8 enabled and older AVR controllers come with the internal 1MHz RC oscillator selected. chip45boot2 works proper with the internal RC oscillator, provided the frequency is at least 4MHz (preferably 8MHz). Alternatively external clock sources can be used, e.g. odd frequencies like 14.7456MHz to achieve error free baud rate prescaler values. Keep in mind, that “external clock” selects an external clock signal, not an external crystal! If you use a crystal, make sure not to select “external clock”, since it will make further ISP access impossible as long as you do not connect some external clock signal to XTAL1 of the MCU! Which is a proper solution to reanimate misconfigured MCUs.

After the fusebits are set properly and the hexfile is programmed, you can work with the bootloader.

ACTIVATING THE BOOTLOADER

After a reset the bootloader waits for approximately 2 seconds to detect a transmission at its RXD pin. If so, it will measure the timing of the rising and falling edges of four consecutive characters 'U' at 19200 baud to determine its correct baud rate prescaler. After this, the bootloader is active and ready to communcate at 19200 baud with a terminal program (or another PC software) and will print a short welcome message: “c45b2” plus version number followed by a carriage return and a promt character '>'.

The most simple way to activate the bootloader in practice is:

- start your terminal program (19200 baud, 8N1, XON/XOFF handshake)
- hold SHIFT-U pressed
- reset the target MCU
- see the welcome message
- release SHIFT-U

WORKING WITH THE BOOTLOADER

The bootloader understands some simple commands, which are listed in the following table. When a command has been executed properly, it will be echoed (depending on the command, additional return values are appended) and a '+' is appended. In case of any error, a '-' is appended.
The bootloader uses a '\n' (new line, 0x0d) as line ending, hence it works properly with Windows and Unix line endings. Make sure to set the line endings on your terminal program to 'CR+LF' for outgoing messages.

Command	Description
bNN Return value: bNN+	Set the baudrate to a new value. NN is a two digit decimal number, indicating a multiple of 9600 baud. Useful values are b01 (9600), b02 (19200), b04 (38400), b06 (57600), b12 (115200), b24 (230400) Example: b12 Set baudrate to 115200 baud (=12x9600)
mwAAAADD return value: mwAAAADD+ DD	Write one byte to target address in SRAM memory (mw means memory write) AAAA is a four digit hex number representing the address DD is a two digit hex number containing the 8 bit data word to be written to the above address Example: mw002a80 Write byte 0x80 to memory address 0x002a Note: The lower DD in the return value is not the data word of the command, but is the data word read back from the address. So you see instantly, if your write operation was successful.

Command	Description
mrAAAA return value: mrAAAA+ DD	Read one byte from target address in SRAM memory (mr means memory read) AAAA is a four digit hex number representing the address DD is a two digit hex number containing the 8 bit data word read from the above address Example: mr002a Read byte from memory address 0x002a
ewAAAADD return value: ewAAAADD+ DD	Write one byte to target address in EEPROM memory (ew means eeprom write) AAAA is a four digit hex number representing the address DD is a two digit hex number containing the 8 bit data word to be written to the above address Example: ew004253 Write byte 0x53 to eeprom address 0x0042 Note: The lower DD in the return value is not the data word of the command, but is the data word read back from the address. So you see instantly, if your write operation was successful.
erAAAA return value: erAAAADD+ DD	Read one byte from target address in EEPROM memory (er means eeprom read) AAAA is a four digit hex number representing the address DD is a two digit hex number containing the 8 bit data word read from the above address Example: er0042 Read byte from eeprom address 0x0042
pf return value: pf+	Programm flash memory After entering this command, a standard intel hexfile is read from the respective UART. Make sure, that you have XON/XOFF handshake activated, since a short break Is necessary when a flash page write occurs. For every successfully read and parsed line of the hexfile, a '.' is printed on the terminal. When a flash page has been sccessfully written, an addition '*' is printed. In case a checksum error is detected in a record line, a '-' is printed and command is terminated.
pe return value: pe+	Programm EEPROM memory After entering this command, a standard intel hexfile is read from the respective UART. Make sure, that you have XON/XOFF handshake activated, since a short break is necessary when a flash page write occurs. For every successfully read and parsed line of the hexfile, a '.' is printed on the terminal. Since the EEPROM is written byte wise, an addition '*' is printed for every line. In case a checksum error is detected in a record line, a '-' is printed and command is terminated.
g return value: g+	Go to application This command jumps to the application start at flash address 0x0000. All altered IO registers will be reset to it's initial state, so that your application may assume the datasheet reset defaults for all IO registers. Alternatively a reset can be performed and after the 2-3 second timeout (when no characters will be received at the UART), the bootloader will automatically jump to the application.

FRAGMENTED HEXFILES

Depending on the compiler you use and the arrangment of your code and variables, the memory layout of your generated hexfile, i.e. arrangement/alignment of code space, constant variables, initialized/non-initialized variables, etc., can contain a gap or a step. The following example shows such a step in the hexfile:

```
:100110000D920197E1F78C8130E0280F311DC90164
:10012000C057D14F0FB6F894DEBF0FBECDBFCF91F1
:08013000DF910895F894FFCF60
:100138005A75206D65696E657220456E74736368C3
```

Even though the addresses are used linearly, the third line contains only eight data bytes and the following lines start no longer at a 16-byte boundary, but with an “offset” of eight.
This cannot be handled by the chip45boot2 hexfile parser currently (and is not planned to be handled in the future, since it would lead to larger than 2k bootloaders for several devices) and can lead to misprogrammed bytes in the flash memory.
If you observe problems with downloading your program, please check your hexfile for such steps. If you find one, you can convert your hexfile to a hexfile without such step with the following command line tool:

```
srec_cat.exe test.hex -Intel -Output test_new.hex -Intel -Line_Length 44
```

screc_cat normally converts between different output formats, but can be used to convert a hexfile into a hexfile with fixed 44 characters per line, which means 16 data bytes per line. srec_cat comes with the WinAVR compiler toolset or can be downloaded at http://srecord.sourceforge.net/download.html.

HEX FILE LICENSE

chip45 GmbH & Co. KG grants the permission to use the precompiled chip45boot2 hexfiles for either non-commercial or commercial applications without limitations and free of charge. The precompiled chip45boot2 hexfiles come “as is” without any warranties.

SOURCE CODE LICENSE

If you want to use and modify the bootloader source code, e.g. to add customer specific functions, the source code can be purchased at www.chip45.com/chip45boot2. The source code is being distributed as a single ZIP file, which provides an AVR-Studio project file, the WinAVR source files and a batch file for automatic generation of the MCU specific hex files. The source code is well commented and can be easily modified and extended according to your special requirements.

- You may use and modify the bootloader source code for any of your products or projects.
- You may distribute unlimitted compiled binary versions of the bootloader.
- You may not distribute your modified source code to a third party.
- You may not redistribute the original source code to a third party.
- You may redistribute your license of the bootloader to a third party only with permission of chip45 GmbH & Co. KG.
- The chip45boot2 source code comes “as is”, without any warranties.

The current release V2.87 of chip45boot2 was compiled with gcc version 4.3.2 (WinAVR 20090313).

WHAT ELSE DO YOU NEED?

– You need an ISP adapter for downloading the chip45boot2 bootloader into your target MCU (see http://www.chip45.com/Programmer for suitable devices). You have to do this just once, after this an ISP adapter is no longer required for further programming, since the bootloader will do the programming over a UART connection.
– The WinAVR development environment and compiler/assembler (see http://winavr.sourceforge.net) if you want to modify the bootloader code.

REVISION HISTORY

The major version number is 2 for chip45boot2, development started with 2.00, since the old chip45boot used version 1.xx
The minor version number is the revision number from the svn repository.

2.53
- first official release of chip45boot2

2.82
- changed line ending of input hex files from '\r' (0x0d) to '\n' (0x0a)

2.83
- added the version number to the welcome message
- change line ending macro from '\n' to hex number 0x0a
- reverted line endings for character output bach to '\r', only for hex file reading the '\n' should be used
- use CR and LF macros instead of '\r' and '\n', to be independent of compiler handling
- the command is now echoed immediately after reading the command and the command handling only appends the + or some return value

2.87
- added internal RXD pin pullup activation to make it less sensitive against disturbances of a probably floating pin in the final application
  accidental high->low transitions after reset could activate the bootloader instead of the application

2.90
- renamed many variables and functions to indicate the data type and related module/header file
- fixed a bug in command line reception

DISCLAIMER