chip45boot2 Infosheet

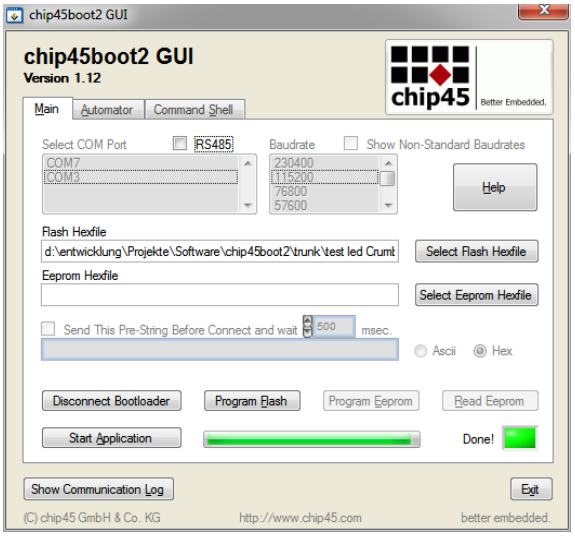
Flexible bootloader for AVR microcontrollers with auto-baud, hexfile support and extra functions.

BASIC SPECIFICATIONS

Name	Devices with pre-compiled hex files.	Memory Usage
chip45boot2 (current release is v2.9Q)	ATmega8, ATmega88(P), ATmega8515, ATmega8535 ATmega16(A), ATmega162, ATmega164(A/P), ATmega165, ATmega168(P) ATmega32, ATmega324P, ATmega325(0), ATmega328(P), ATmega3290P ATmega32U4 ATmega64(0), ATmega645(0), ATmega644(A/P) ATmega128, ATmega1280/1, ATmega1284(P) ATmega2560, ATmega2561 AT90CAN32, AT90CAN64, AT90CAN128 AT90PWM216, AT90PWM316 ATmega128RFA1	1k word boot block size (2048 bytes)
	ATxmega64a1, 128A1(U) ATxmega64A3, 128A3(U), 192A3, 256A3, 256A3B (including workaround for older mask revision D) ATxmega16A4, 16A4U, 32A4, 32A4U, 128A4U ATxmega64D3, ATxmega128D3	8k fixed size bootblock for xmegas

CHIP45BOOT2 FEATURES

- direct support of Intel Hex files
- automatic baud rate setting to host baudrate, independent of CPU clock settings (!!!)
- RS485 support!
- 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) for ATmegas/AT90CANs, 8k fixed size bootblock for Xmegas
- distributed as precompiled hex files for simple usage
- comfortable PC and Mac OS X software available for free



PRECOMPILED HEX FILES

Pre-compiled hex files are available for a bunch of AVR MCUs in the bootloader/build folder. If you need another one, see further below for porting to new targets.

WHAT ABOUT ALL THE NEW A-DEVICES?

The A-devices, like ATmega128A instead of ATmega128 are hexfile compatible with the previous non-A devices. So you can use the hexfiles for a non-A on the same A device. The same applies for a PA-device, just use the P hexfile on the same PA-device.
This also aplies vice-versa! The latest AVR GNU Toolchain is lacking support for some of the older non-A devices. No problem, in that case just use the A hexfile on the non-A device.

AND WHAT ABOUT THE NEW ATXMEGA...U's?

The new versions of the Xmegas named ATxmega...U (e.g. ATxmega128A3U) provide an onchip USB controller, which is not used by the bootloader at all. You can use the non-U precompiled hexfiles also on the U-devices (and vice-versa). Some users reported problems with this procedure (which we were not able to reproduce, so it's likely that the reason is something different).

chip45boot2 Infosheet

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
- **with the chip45boot2 GUI:**
  - start the GUI, select COM port and baud rate and press “Connect to Bootloader”
  - reset your target (not necessary, if your target supports auto-reset-feature)
- **with a terminal program:**
  - set PC serial port to 19200 baud, 8N1, XON/XOFF (we suggest to start with 19200 baud, even though higher baudrates are possible, depending of target MCU clock)
  - 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 ATAVRISP2 by Atmel 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. Most important for the bootloader to work properly are the BOOTSZ and BOOTRST fuses. 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 to select “external crystal” and not “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 the host's baud to determine its correct baud rate prescaler. After this, the bootloader is active and ready to communcate at host's baud rate 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 two most simple ways to activate the bootloader in practice is:

1) with a standard terminal program
- start your terminal program (19200 baud, 8N1, XON/XOFF handshake)
- hold SHIFT-U pressed
- reset the target MCU
- release SHIFT-U
- press some other character than SHIFT-U (e.g. carriage return or space)
- see the welcome message

2) with the chip45boot2 GUI PC application
- start the chip45boot2 GUI and select COM port and baud rate
- reset the target MCU (not necessary if target has auto-reset-feature)
- press the button “Connect to Bootloader”

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	Set the baudrate to a new value. **Note:** This command has been removed in V2.9A, since bootloader can connect at any supported baudrate. A change during operation is not necessary.
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.
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.

Command	Description
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

If you use the chip45boot2 GUI or the Mac OS X BootFeedX, you don't have to take care of hexfile fragmentation, both application handle this correctly.

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.

WATCHDOG

The bootloader tries to disable the watchdog after startup. If the watchdog was enabled in software before (even before the last reset), it will be disabled and should not interrupt the bootloader. If you jump to application, the bootloader will not restore the previous watchdog state. Your application has to take care of enabling the watchdog on start.
Note: If the watchdog is enabled by fusebits, it's not possible to disable it in software. If you need a permanantly running watchdog, you would have to buy the source code and add some code to trigger the watchdog.

RS485 SPECIALITIES

The precompiled hexfiles also contain an RS485 version of each target and each USART of the supported AVR controllers. Since RS485 is (in most applications) a half-duplex communication, an additional IO pin must be used to switch the direction of the RS485 transceiver between transmit and receive. The precompiled hexfiles always use the corresponding XCKn pin to the USARTn (e.g. USART1 uses XCK1 for direction control). If you need to use a different IO pin for direction control, you have to buy the source code of the chip45boot2 bootloader at our online shop. The IO pin can then easily be changed in a header file.

The RS485 communication should work with most common USB RS485 converters, e.g. http://www.ftdichip.com/Products/EvaluationKits/USB-RS485-PCB.htm. The converters might vary in dead time settings between send and receive, so a particular converter might need some finetuning in the delay() functions in the source code. Let us know, if you run into problems here, we will try to assist.

If you use the chip45boot2 GUI with a USB RS485 converter (i.e. over a virtual COM port), make sure you have the RS485 checkbox set before connecting, otherwise connecting to the bootloader will not work. Make sure you have installed the latest version of the GUI on your PC.

The non-RS485 bootloader versions wait for the 'U' characters for connecting and after baud rate adjustment and checking some 'U' characters, the welcome message and prompt is sent directly. This leads into trouble on half-duplex systems, hence the RS485 bootloader versions wait for the 'U' characters and do the baud rate adjustmetn and checking, but wait for one other character than a 'U' and receiving this other character, the RS485 direction is

changed and the welcome message is sent.

Since the RS485 communication requires direction changing and some deatime to make sure, that handshake characters like XON and XOFF will not get lost, the download of a hexfile is slower than with a full-duplex UART communication. It's possible to set the baudrate as high as 500000 baud, but anything higher than 62500 baud has not much effect anymore.

MIT License