

[aldenhardt 13](#)

- [Dashboard](#)
- [Inbox 0](#)
- [Account Settings](#)
- [Log Out](#)

### [Advanced Search](#)

Search...

 

- [Explore GitHub](#)
- [Gist](#)
- [Blog](#)
- [Help](#)

## [alexforencich](#) / [xboot](#)

- [Watch](#)
- [Fork](#)
- - [4](#)
  - [1](#)
- [Source](#)
- [Commits](#)
- [Network](#)
- [Pull Requests \(0\)](#)
- [Issues \(0\)](#)
- [Wiki \(2\)](#)
- [Graphs](#)
- *Branch:* master
- [Home](#)
- [Pages](#)
- [Wiki History](#)
- [Git Access](#)

# 1. Introduction

- [New Page](#)
- [Edit Page](#)

- [Page History](#)

# 1. Introduction

XBoot is an extensible, modular bootloader for the ATMEL XMEGA processor series. It is compatible with the AVR109 (butterfly) bootloader protocol with a few XMEGA specific extensions for access to the user and production signature rows. One of its main features is support for multiple serialbusses. Many bootloaders only support RS232 for programming from a PC, but XBoot's modularity allows it to support the exact same set of commands over any hardware serial port. Currently, I2C support has been incorporated. This allows for easy in-system reconfiguration of XBoot equipped chips with little additional time investment. Also, XBoot includes support for I2C address autonegotiation for when multiple, identically configured processors sit on the same I2C bus. Autonegotiation requires one extra shared open-drain connection, but many systems will have IRQ lines in them anyway and those can usually be repurposed at boot time.

Thanks for using XBoot!

Alex Forencich

## 1.1 Compatibility List

Currently, XBoot should work on any XMega processor. In the future, we intend to expand XBoot to cover other AVR processors also. The following list of processors are currently supported. An asterisk denotes the MCU has been tested and confirmed XBoot compatible.

```
atxmega16a4
atxmega32a4 *
atxmega64a1
atxmega64a3
atxmega64a4
atxmega128a1
atxmega128a3
atxmega128a4
atxmega192a1
atxmega192a3
atxmega256a1
atxmega256a3b
atxmega256a3 *
atxmega64a3 *
atxmega128a1 *
atxmega32a4 *
```

# 2. Using XBoot

## 2.1 Configure

Before building XBoot, please configure it so it will interface properly with your system. This will involve editing some parameters in the makefile and some parameters in xboot.h. The main parameters that need to be set in the makefile are the target chip (MCU) and the frequency (F\_CPU). All you need to do is make sure the only line that's not commented out is the one for your chip and the proper frequency. For the simplest bootloader configuration, you may only choose 2000000 and 32000000 for the clock speed, corresponding to the two internal RC oscillator frequencies. For the rest of the configuration, see the section 3, "Configuring XBoot".

## 2.2 Build XBoot and Program to Chip

To build XBoot, open up the Makefile and make sure the MCU line for the target processor is the only one uncommented. Then type "make". This will compile the whole package and generate xboot.hex, which can be downloaded with any programming cable capable of programming XMEGA chips. If you want to save some time and just program the boot section, type "make xboot-boot.hex" and then write the new file xboot-boot.hex to the boot section. The makefile includes built-in support for the Atmel JTAGICE mkII programmer over USB via avrdude, so if you have one connected you can type "make program" and it will take care of everything. If you don't have one of these but still want to use avrdude, modify the avrdude parameters in the makefile.

## 2.3 Write Main Application Program

To write a program to a device with XBoot installed, use a command like this:

```
avrdude -p atxmega64a3 -P /dev/ttyUSB0 -c avr109 -b 19200 -U flash:w:main.hex
```

Or for windows:

```
avrdude -p atxmega64a3 -P com1 -c avr109 -b 19200 -U flash:w:main.hex
```

Also, feel free to re-use XBoot's makefile for your own code. Like XBoot, it is reconfigurable and can be used to compile most projects. It also has these programming configuration for XBoot built in, all you need to do is switch a couple of comments around.

*NOTE:* At this time, avrdude (currently 5.10) does NOT support programming the XMEGA flash boot section (see <https://savannah.nongnu.org/bugs/?28744>). If you want to use avrdude, you will need to compile it from source with one of the patches listed on the bug report.

## 2.4 Notes for Main Application

Here are a few tips for your main application that will make using XBoot a much more pleasant experience.

### 2.4.1 Program UART Bits Properly

If you select the USE\_UART option, XBoot will program the UART. If your program uses the same UART as XBoot, you should not assume any of the UART registers are blank! Manually force all bits to the desired

state. In particular, the 2x bit is often left out of UART configuration routines, but is set by XBoot.

## 2.4.1 Catch the "Enter Bootloader" command

When AVRDUDE starts programming the chip, the first character sent out is the "Escape" character, 0x1B. If your program transmits ASCII, or only transmits Binary during certain program states, you can monitor the UART for the escape character and cause a software reset to enter the bootloader, as shown in the following snippet:

```
if (rx_byte == 0x1B) {
    CCPWrite( &RST.CTRL, RST_SWRST_bm );
}
```

In many cases, this allows you to use the AVRDUDE program command without having to reset the AVR.

## = 3. Configuring XBoot =

XBoot is designed to be reconfigured to suit specific needs. Out of the box, everything is turned on. Turning off features and reassigning pins is easy, open up xboot.h and change the #defines.

Recommended configuration:

```
// bootloader entrance
#define USE_ENTER_DELAY
#define USE_ENTER_UART

// bootloader communication
#define USE_LED
#define USE_UART

// bootloader features
#define ENABLE_BLOCK_SUPPORT
#define ENABLE_FLASH_BYTE_SUPPORT
#define ENABLE_EEPROM_BYTE_SUPPORT
#define ENABLE_LOCK_BITS
#define ENABLE_FUSE_BITS
```

This configuration will make the bootloader work similarly to an Arduino. It will blink its light a few times, polling for a character. If none is received, it starts the application. If one shows up, it enters the bootloader and processes it.

## 3.1 Bootloader clock options

### 3.1.1 USE\_DPLL

This will turn on the DPLL for the selected oscillator, improving its accuracy. Recommended for high serial baud rates.

### 3.1.2 USE\_32MHZ\_RC

This will switch to the 32MHz RC oscillator on start. In the default configuration of xboot.h, this will be defined automatically when `F_CPU` is set to 32000000.

## 3.2 AVR 1008 fixes

If you're using a device affected by AVR1008, then you may need to enable these for the bootloader to successfully program the chip. Affected chips are the ATXMEGA256A3 rev A, ATXMEGA256A3B rev B, ATXMEGA256A3 rev B, and possibly the ATXMEGA192A3.

### 3.2.1 USE\_AVR1008\_EEPROM

This enables the AVR1008 fix for the EEPROM controller

## 3.3 Bootloader entrance options

### 3.3.1 USE\_ENTER\_DELAY

If this is defined, XBoot will run a loop, specified with the `ENTER_BLINK_*` variables, and check for an entry condition. If none is found, it jumps into the main code. (BTW, they're called `ENTER_BLINK_*` because they assume `USE_LED` is defined. If it isn't, it will still work, but the variable names don't make a whole lot of sense...)

Options

- `ENTER_BLINK_COUNT` defines the number of times to blink the LED, e.g. 3
- `ENTER_BLINK_WAIT` defines the number of loops to make between blinks, e.g. 30000

### 3.3.2 USE\_ENTER\_PIN

If this is defined, XBoot will check the state of a pin, specified with the `ENTRY_PORT` and `ENTRY_PIN_*` variables, when it starts (and possibly throughout the startup delay loop) to determine if it should start or just jump into the main program.

Options

- `ENTER_PORT` defines the port that the in is in, e.g. `PORTC`
- `ENTER_PIN` defines the pin in the port, an integer from 0 to 7
- `ENTER_PIN_CTRL` defines the `PINnCTRL` register for the pin, e.g. `ENTER_PORT.PIN0CTRL`
- `ENTER_PIN_STATE` defines the "asserted" state of the pin, 0 or 1
- `ENTER_PIN_PUEN` enables a pull-up resistor on the pin if nonzero

### 3.3.3 USE\_ENTER\_UART

If this is defined, XBoot will poll for received characters over the UART. If one is received, it will enter the bootloader code. `USE_UART` must be defined.

### 3.3.4 USE\_ENTER\_I2C

If this is defined, XBoot will poll for received characters over the I2C interface. If one is received, it will enter the bootloader code. `USE_I2C` must be defined.

## 3.4 Bootloader exit options

### 3.4.1 LOCK\_SPM\_ON\_EXIT

If this is defined, SPM instructions will be locked on bootloader exit.

## 3.5 Bootloader communication

### 3.5.1 USE\_LED

If this is defined, XBoot will use an LED for feedback, specified by the `LED_*` variables.

Options

- `LED_PORT` defines the port, e.g. `PORTA`
- `LED_PIN` defines the pin, e.g. `0`
- `LED_INV` inverts the LED state if nonzero

### 3.5.2 USE\_UART

If this is defined, XBoot will configure and use a UART for communication.

Options

- `UART_BAUD_RATE` defines the baud rate of the UART, e.g. `19200`
- `UART_PORT_NAME` defines the port that the UART is connected to, e.g. `D`
- `UART_NUMBER` defines number of the UART device on the port, e.g. `1` for `USARTD1`
- `UART_TX_PIN` defines the UART TX pin bit mask, e.g. `PIN7_bm`
- `UART_BSEL_VALUE` defines the value of `BSEL`, e.g. `12`
- `UART_BSCALE_VALUE` defines the value of `BSCALE`, e.g. `0` [these two parameters depend on the selected baud rate and processor frequency. There is a calculation included in the header file for automatically generating BAUD rates, but it is preferred to use a predefined set of parameters that is known good at the specified frequency.]

### 3.5.3 USE\_I2C

If this is defined, XBoot will configure and use an I2C/TWI controller in slave mode for communication.

## Options

- `I2C_DEVICE_PORT` defines the port the I2C interface is on, e.g. `E` for `TWIE`
- `I2C_MATCH_ANY` will enable the I2C controller promiscuous mode (match any address) if nonzero
- `I2C_ADDRESS` defines the default I2C address `0x10`
- `I2C_GC_ENABLE` enables the I2C bus general call capability (address `0`) if nonzero

### 3.5.4 USE\_I2C\_ADDRESS\_NEGOTIATION

Enables I2C address autonegotiation if defined. Requires `USE_I2C`.

## Options

- `I2C_AUTONEG_DIS_PROMISC` will disable I2C promiscuous mode after completion of autonegotiation routine if nonzero
- `I2C_AUTONEG_DIS_GC` will disable I2C general call detection after completion of autonegotiation routine if nonzero
- `I2C_AUTONEG_PORT` defines the port in which the autonegotiation pin is located, e.g. `PORTA`
- `I2C_AUTONEG_PIN` defines the pin, e.g. `2`

### 3.5.5 USE\_ATTACH\_LED

Enables the autonegotiation code to turn on a light when a new I2C address is received.

## Options

- `ATTACH_LED_PORT` defines the port, e.g. `PORTA`
- `ATTACH_LED_PIN` defines the pin, e.g. `1`
- `ATTACH_LED_INV` inverts the LED state if nonzero

## 3.6 General Options

### 3.6.1 USE\_INTERRUPTS

Defining this will configure XBoot to use interrupts instead of polled I/O for serial communications. This will increase code size and won't offer much advantage at the time being, so only use if you know what you're doing.

### 3.6.2 USE\_WATCHDOG

Defining this will enable the watchdog timer during operation of the bootloader. This can reduce the overhead caused by failed programming attempts by resetting the chip if the bootloader and host get out of sync.

## Options

- `WATCHDOG_TIMEOUT` determines the watchdog timeout period; leave only one of the listed lines uncommented (see XMEGA A series datasheet for details)

## 3.7 Bootloader features

Generally, these are all enabled, but they can be disabled to save code space.

### 3.7.1 `ENABLE_BLOCK_SUPPORT`

Enables flash block access support

### 3.7.2 `ENABLE_FLASH_BYTE_SUPPORT`

Enables flash byte access support

### 3.7.3 `ENABLE_EEPROM_BYTE_SUPPORT`

Enables EEPROM byte access support

### 3.7.4 `ENABLE_LOCK_BITS`

Enables lock bit read and write support (note: cannot clear lock bits to 1, complete chip erase from external programmer needed to do that)

### 3.7.5 `ENABLE_FUSE_BITS`

Enables fuse bit read support (cannot write fuse bits outside of hardware programming)

Last edited by alexforencich, December 16, 2010

[Delete this Page](#)



Powered by the [Dedicated Servers](#) and [Cloud Computing](#) of Rackspace Hosting®

- [Blog](#)
- [About](#)
- [Contact & Support](#)
- [Training](#)
- [Job Board](#)
- [Shop](#)
- [API](#)
- [Status](#)



- © 2011 GitHub Inc. All rights reserved.
- [Terms of Service](#)
- [Privacy](#)
- [Security](#)

# Markdown Cheat Sheet

## Format Text

### Headers

```
# This is an <h1> tag
## This is an <h2> tag
##### This is an <h6> tag
```

### Text styles

```
*This text will be italic*
_This will also be italic_
**This text will be bold**
__This will also be bold__

*You **can** combine them*
```

## Lists

### Unordered

```
* Item 1
* Item 2
  * Item 2a
  * Item 2b
```

### Ordered

```
1. Item 1
2. Item 2
3. Item 3
  * Item 3a
  * Item 3b
```

## Miscellaneous

### Images

```
![GitHub Logo](/images/logo.png)
Format: ![Alt Text](url)
```

### Links

```
http://github.com - automatic!  
[GitHub](http://github.com)
```

## Blockquotes

```
As Kanye West said:  
> We're living the future so  
> the present is our past.
```

## Code Examples in Markdown

Syntax highlighting with [GFM](#)

```
```javascript  
function fancyAlert(arg) {  
  if(arg) {  
    $.facebox({div:'#foo'})  
  }  
}  
```
```

Or, indent your code 4 spaces

Here is a Python code example  
without syntax highlighting:

```
def foo:  
    if not bar:  
        return true
```

## Inline code for comments

```
I think you should use an  
`<addr>` element here instead.
```