Working With Bootloaders & Build Your Own Bootloader – 1

T.K. HAREENDRAN

AVR tutorial

Share this:

Yes, now we are in a plan to working with bootloaders and create custom bootloaders. Good, but for enforcing this you will probably need to understand/refresh some useful information!

At first, note that, fundamentally a bootloader (BL) is just a normal AVR application that is positioned in a special region of flash memory. The BL can not easily update its own code, but the BL can reprogram the application region while the BL continues to run (applications very rarely update the BL).

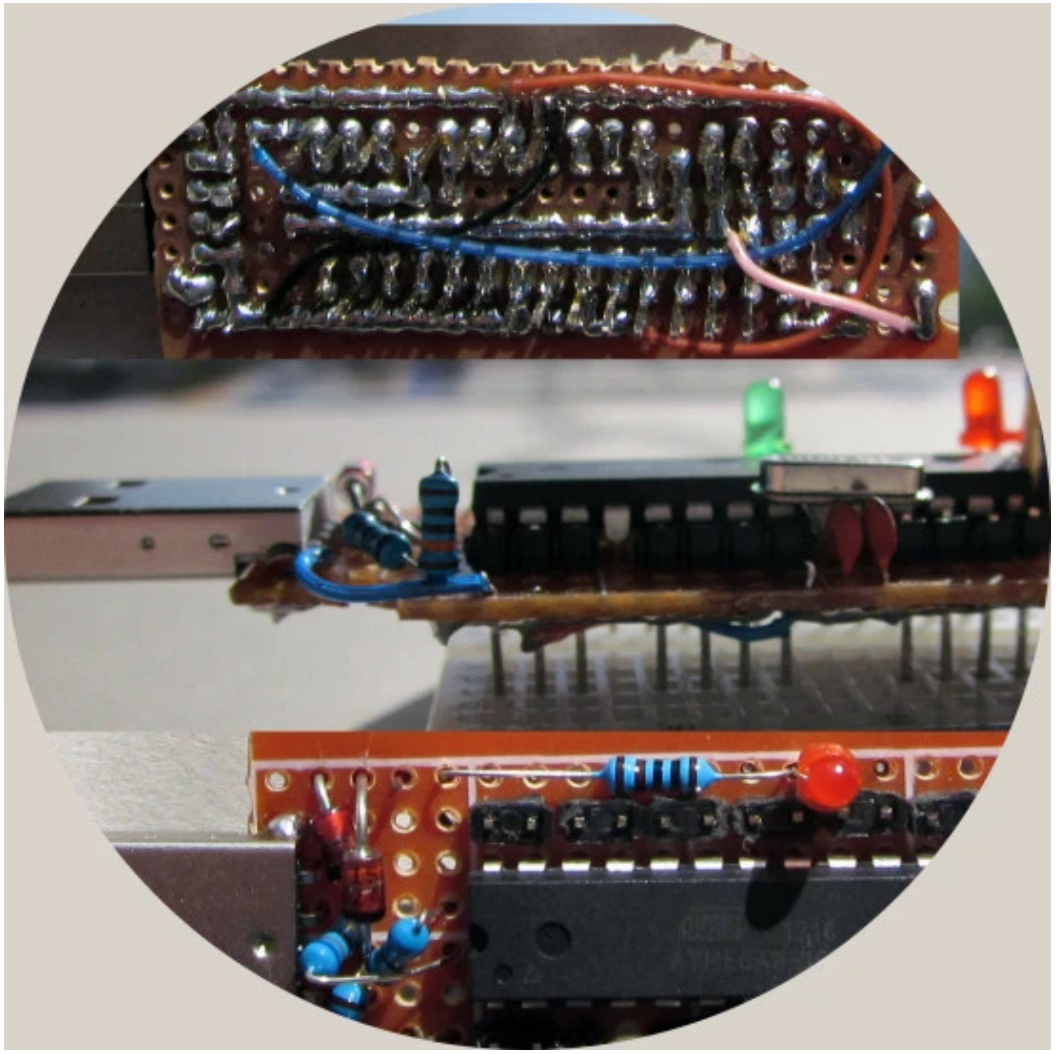
- A bootloader (BL) is best written as a stand-alone program that has no dependencies on the application
- You will need an external programmer to write the BL to the MCU. If you have a working external programmer that can write the application and update fuses, it will okay for the BL
- Bootloaders work best as small, reliable, and infrequently changed programs. It is almost always better to set the BOOTRST fuse to run the BL after reset. When the BL runs after reset, you can easily reprogram the application even when the application is totally broken or blank (if the application is run on reset instead, you may be forced to use your external programmer to recover from the mishap)

How does a BL know when to start the application? This is a good question, and you can find number of answers for this query! If your device has an input mechanism (for example a button switch) you can just signal your intent with a simple trigger. The BL would normally run the application immediately, and only remain resident itself when a trigger event (like a button push) is initiated during reset. Another solution is for the BL to check an external communication channel (for example USB) for a particular symbol during startup. As in the first solution, here the BL would normally run the application immediately, and only remain resident itself after checking the external communication channel and receiving a certain response.

If these two methods are not practical for your application, you can try the EEPROM solution (assuming your AVR has an EEPROM). This means the storing of a byte value which indicates when the bootloader should continue running and when it should start the application. We will learn more about this procedure through the forthcoming parts of this tutorial.

Quick Reference

- A BL is a piece of software that is located in a special part of the microcontroller's flash memory. It is able to communicate with the outside world through the pins of the microcontroller, and is capable of changing the major part (application section) of the flash content
- A BL makes it possible to connect the microcontroller to the PC application directly(without any external programmer), read the new software from the PC and put it into the main part of the flash
- As BL can be written in many different ways, it can be adapted to specific application needs, even with a decryption algorithm to prevent reverse-engineering, firmware hacking, etc



USB bootloader for Atmel AVR controllers

AVRUSBBoot is a popular bootloader (thanks Thomas Fischl) for the Atmel AVR controllers, uses a firmware-only USB driver to transfer binary data from the PC to the flash of the microcontroller. Once the AVR is flashed with the BL, no other ISP programmer is needed. Thereafter, the microcontroller can be reprogrammed over USB interface.

In principle, AVRUSBBoot can be used with all circuits which are supported by the AVR USB driver. To switch between the BL and the application, an aditional jumper/switch is necessary. The firmware has to be adapted to your hardware. All necessary changes has to be done in the following two files:

- **bootloaderconfig.h:**
  Define the condition when the BL should be started, e.g. if a special pin is put to ground with a jumper, and the initialisation of the hardware.
- **usbconfig.h:**
  Define the used data line pins. You have to adapt USB_CFG_IOPORT, USB_CFG_DMINUS_BIT and USB_CFG_DPLUS_BIT to your hardware. The rest should be left unchanged.

(you need avr-gcc, avr-binutils and avr-libc, to compile the firmware)

The bootloader firmware has to be written to the controller with an ISP programmer. Once the bootloader is flashed, you don't need a programmer and you can download the binary data over USB. However, always remember to set the fuses for the external clock source when flashing the BL for the first time. A C++ tool for downloading hex files is available.

Note that, to compile the program, you need libusb. When the device is connected and the BL is started (set the jumper before connecting the device), hex files can be written to the flash of the microcontroller.

Useful Download Links:

• avrusbboot: **http://www.fischl.de/avrusbboot/avrusbboot.2006-06-25.tar.gz**
• firmware-only avr usb driver: **http://www.obdev.at/products/avrusb/**
• libusb: **http://libusb.sourceforge.net/**
• libusb-win32: **http://libusb-win32.sourceforge.net/**

AVRUSBBoot a small bootloader enough to fit into the 2 kB boot block of the ATMega8. This boot loader is good for prototyping as there is no need to connect a programmer. Just set a jumper and the programmer is built into the target. More details available here: **http://www.fischl.de/avrusbboot/**

→ Part 20: **Working With Bootloaders & Build Your Own Bootloader – 2**