USBprog
User's Manual

Bernhard Walle*

2nd January 2010

*bernhard@bwalle.de

Contents

1 Overview

1.1 The USBprog Hardware

If you read this document, you have probably already an USBprog device. In the past, every microcontroller has its own programming hardware that is mostly relatively expensive. If you work with multiple microcontroller environments, you end up with plenty of programmers on your desk that are not only expensive but also waste space.

On the other side, most self-built programming hardware (for example several ISP programmers for the Atmel AVR controllers) was for the parallel port. However, modern PCs have no parallel port. While you can extend a PC with a parallel port PCI card, you're lost on notebooks. Building and programming USB hardware is not really difficult but still more work than for the parallel port.

The most important thing is the firmware. Once programmed with a so-called *bootloader* the firmware can be exchanged. While it's necessary to have an ISP programming device[1] once to program the USBprog, a normal PC or Mac with the *USBprog software* is sufficient to change the firmware. So it takes only a few seconds to make a JTAG device from an Atmel MTK II clone, for example.

Warning: This document only describes the USBprog hardware in version 3.0. If you still have an USBprog 2.0 device, please refer to the online documentation! If you are unsure, look at picture 2.1 on page 6.

1.2 The USBprog Software

As already mentioned in the section above, you need a special PC software to exchange the firmware of the USBprog. This software is available as both GUI and command line version that can be also used in scripts and Makefiles. For following systems are supported:

- Microsoft Windows 2000 and XP
- Linux (tested on openSUSE and Ubuntu)
- MacOS X (tested on Leopard)

[1] That can be a second, already initialised USBprog or another ISP programmer.

If you use another Unix system such as OpenSolaris or *BSD, chances are high that the software runs out of the box as long as a port of *libusb* is available.

Note: It's known that the software doesn't run in Windows Vista and Windows 7 has not been tested. If you use one of that operating systems, we recommend using a virtual machine (for example VirtualBox which is free for personal use) with a Linux installation in it.

1.3 About this Document

This documentation should be both: A guide to get started for people that just bought the hardware and want to install the software and use the hardware and also a reference documentation for both the software and the most common firmwares.

If you have any suggestions how to improve the software and the documentation or if you just find problems, please don't hesitate to contact the author via e-mail at bernhard@bwalle.de. However, I probably cannot help with hardware or firmware problems. In that case, please contact the developer of USBprog which is Benedikt Sauter (sauter@embedded-projects.net).

1.4 Terms and Definitions

At first we introduce the term *firmware* and *bootloader* as we use it in the rest of the document.

Firmware The main advantage of the USBprog device is that the firmware can be exchanged. The firmware is the program which is loaded into the flash memory of the USBprog device and which is used to perform a specific task: Programming AVR microcontrollers, emulating a JTAG debugger and so on.

Bootloader While the main firmware is exchanged with the computer program, the flash contains also a part (at the highest possible location) that can load the other firmware part. This part is—in the ideal world—only loaded once into the USBprog flash and then never gets overwritten.

1.5 Getting Support

If you have problems with the document here, there are several places where you can get support:

1. There's a web-based forum at http://forum.embedded-projects.net which is quite well-visited. Also the guy who has developed USBprog, Benedikt Sauter, reads that forum regularly.
2. Additionally, there's also a mailing list at Berlios (usbprog-pub@lists.berlios.de). You can subscribe, unsubscribe or read the archive at https://lists.berlios.de/pipermail/usbprog-pub/.

3. For general questions about microcontroller programming, http://www.mikrocontroller.net is always worth looking at. There are also plenty of USBprog users out there.
4. Especially for problems with the USBprog software, you can also send me an email directly at bernhard@bwalle.de.

2 Getting Started with the Hardware

2.1 Connectors, Jumpers and LEDs

At first we have to introduce the jumpers, connectors and LEDs which we talk about in the next sections. All figures in that document are meant to be drawn in the same perspective (USB connector on the right) as figure 2.1.

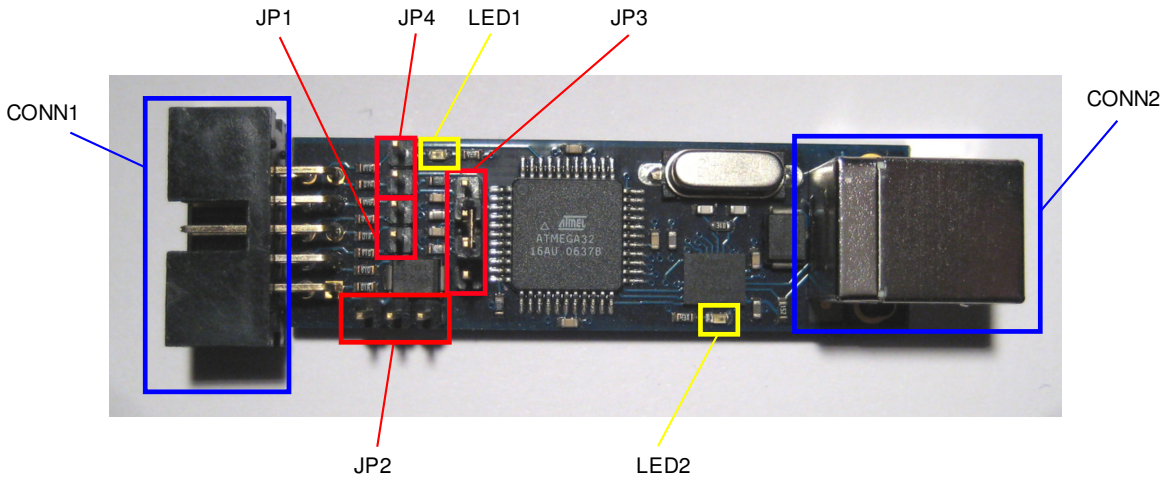

Figure 2.1: The USBprog device (version 3.0) with all connectors, jumpers and LEDs

2.1.1 Connectors

CONN1 is the output interface which is used to do something useful with the USBprog beside from blinking LEDs. For example if you use the *avrispmk2* firmware, this is an ISP interface which is used to program microcontrollers.

CONN2 is obviously the USB connector which is used to connect your USBprog to the computer. You have to use a *Type B* cable which is the normal cable to connect USB devices—apart from micro or mini USB.

2.1.2 Jumpers

JP1 (the lower two of the four pins) is the jumper that must be connected if the bootloader of the USBprog should be updated. See also section 2.2 on page 8 how to flash the bootloader.

JP2 controls how the power supply of the circuit that should be programmed (connected to CONN1). Figure 2.2 shows the possible connections:

The default is no power supply for the programmed circuit. In that case you must ensure that the device is supplied with power by other means. This is the safest possibility.

If you connect the two leftmost pins, that means that the 5 V VCC is directly from the USB port. This setting is dangerous because an error (short-circuit) in your circuit can damage the computer.

An alternative is the setting of the two leftmost pins: In that case, the 5 V VCC is not directly from the USB port but with a Schottky diode. This is safer than the direct connection.

JP3 has two functions: At first it can be used as 5 V serial interface for debugging. You cannot directly connect this jumper to the computer but you need some level converter in between[1]. This functionality is only needed by firmware developers.

More important is another function which is used by the bootloader at startup. To put the device in update mode, disconnect the device, then connect pins 2 and 3 as shown in figure 2.3 and connect the device again.

JP4 is application specific, i. e. used directly by the firmware that is flashed into the USBprog. Currently, JP4 is not used by any of the public firmwares.

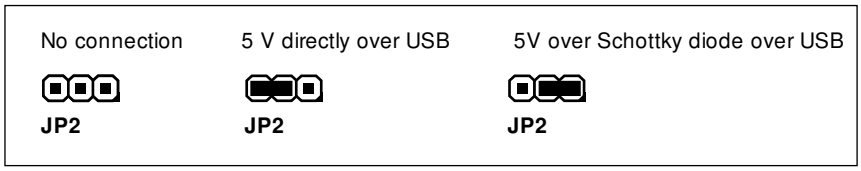

Figure 2.2: Possible connections of JP2

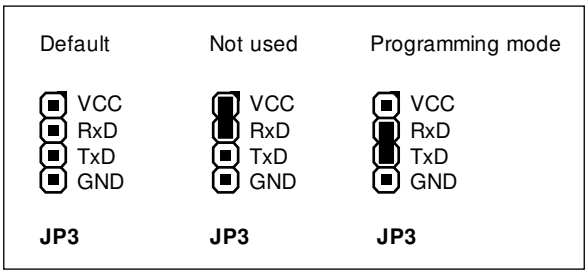

Figure 2.3: Possible connections of JP3

[1] If you want to build such a level converter yourself, look for example at http://www.nslu2-linux.org/wiki/HowTo/AddASerialPort. If you want to buy such a cable in Germany, the Embedded Project Shop also has one. Look for "FTDI-Kabel TTL-232R USB zu TTL serielles Kabel (5.0 V)".

2.1.3 LEDs

LED1 (red) is used by the firmware. There are two common scenarios:

If the USBprog is in *update mode,* the LED blinks slowly. In the wide-spread Atmel MTK II clone firmware, this LED shines while the programming of the microcontroller is ongoing.

LED2 (green) is just the Power LED.

2.2 Initialise the Hardware

This section describes how to install the bootloader. If you bought a version of USBprog which already contains the bootloader, you can safely skip that step. If you don't know if you have a version with bootloader, please read section 2.2.1.

2.2.1 Check if the Bootloader is Installed

To check if the device already contains a bootloader, just put the device into programming mode manually. For this step, you don't require any specific PC software. You just need a working USB port.

1. Disconnect your USBprog from the computer if it's connected.
2. Close the jumper `JP1` as described in section 2.1.2 on page 6.
3. Connect the USBprog to a USB port.

If `LED1` (see figure 2.1 on page 6) now blinks, everything is fine. If not, there's no working bootloader and you need to flash it.

2.2.2 Flashing the Bootloader

When buying a USBprog at the Embedded Projects Webshop you can choose between a slightly cheaper version without an already flashed bootloader and a slightly more expensive version that already has flashed the bootloader.

Flashing the bootloader is the only step where you need a second programming device. It's possible to use an already initialised USBprog with the "AVRISP mk2 Clone" firmware flashed. Any other programmer for Atmel AVR microcontrollers will do it, too. You need the standardised *ISP* interface.

We need not only a programmer but also a programming software. While there are plenty of programmers for Atmel AVR microcontrollers, we use AVRdude because it supports quite a lot of hardware devices and it's available on basically any platform that is around there. On Windows, we suggest to use WinAVR which is quite easy to install.

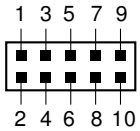

Figure 2.4: Pin numbering used in the ISP pin assignment

Setting the Jumpers

To program the microcontroller that is on the USBprog board, you have to connect jumper JP1 as described in section 2.1.2 on page 6.

There are two possibilities how the programmer gets its power:

1. The programmer has its own power supply. That is the case for every USB-based programmer since USB can supply the device with up to 500 mA.
2. The programmer takes the power from the circuit that should be programmed. That is the case for most "self-made" parallel port programmers since the parallel port is not able to supply devices with power.

If you have a device of the second category, you have to set the jumper JP2 as described in section 2.1.2 on page 6 and shown in figure 2.3 on page 7. We suggest the 3rd method with the Schottky diode.

Wiring

After the jumpers are right, connect the USBprog with a free USB port of your computer which is necessary to supply the USBprog with power. The green power LED (LED1, see section 2.1.3 on the previous page) indicates that everything is okay. If necessary connect your programmer with power and with your computer.

For the connection between your programmer and your computer you need a 10-pole ribbon cable. The pin assignment is the standardised ISP interface as shown in table 2.1. The numbering scheme is shown in figure 2.4.

Pin	Description	Pin	Description
1	MOSI (data out)	2	VCC (+5 V)
3	–	4	GND (signal ground)
5	RESET	6	"
7	SCK (clock)	8	"
9	MISO (instruction in)	10	"

Table 2.1: ISP Pin Assignment

Flashing the Firmware

Now it's time to flash the firmware with AVRdude. At first you have to find out how your programmer is named in AVRdude which is listed in [1] where the `-c` option is described. This symbolic name is spelled as `PROGRAMMER` in the commands below.

At first you have to download the firmware file at http://svn.berlios.de/svnroot/repos/usbprog/trunk/usbprog_base/firmware/usbprog_base.hex. Save the file as `usbprog_base.hex`.

Now flash the firmware with the command

```
% avrdude -p m32 -c PROGRAMMER -U flash:w:usbprog_base.hex
```

The second step requires to set the fuse bits with following interactive AVRdude session:

```
% avrdude -p m32 -c PROGRAMMER -t
avrdude> write lfuse 0 0xe0
avrdude> write hfuse 0 0xd8
avrdude> quit
```

After everything was successful, disconnect the USBprog, remove the jumper `JP1` and connect the USBprog again. The red LED blinks now. This is a sign that you're ready to upload a firmware which is described in section 2.3—just the next section!

2.3 Upload the First Firmware

Well, in theory that's not the correct order because you need to install the software first which is described in section 3.1 on page 12. But maybe you're more a software guy and the software installation already succeeded, so you're ready to execute that step before looking into detail of the software. If not, read section 3.1 on page 12 and come back.

Note: If you're using Windows, we strongly suggest that you read the installation section first because the driver installation is more complicated. On Unix, no driver installation is necessary since USB devices can driven by userspace programs there.

We describe the command line interface here as "first step" for a simple reason: It's exactly the same on every platforms and a command can be reproduced by the user more easily than finding GUI elements.

So open a shell on Unix (Linux/MacOS) or open the *USBprog Command Line* in the "Start" menu on Windows. You should see a prompt that looks like

```
(usbprog) _
```

Note: On Linux, we suggest to run that first test as `root` user. So switch to `root` using the `su` command on Fedora/openSUSE or execute the command using `sudo` on Ubuntu. The installation section will show how to configure your system to avoid that in future.

Before you connect the USBprog, you should manually switch to update mode by inserting jumper `JP3` like shown in figure 2.3 on page 7 as "Programming mode". After you connected the USBprog

to the USB port of your computer, wait a few seconds to make sure that the operating system detected the device[2].

Now the `devices` command should display a single USBprog device:

```
(usbprog) devices
 [ 0]  *  Bus 003 Device 002: 1781:c620
          usbprog: USBprog in update mode
```

Since it's the only device and since the device is already in *update mode,* the star (*) after the number shows you that this device is already selected. So you can proceed and upload the `blinkdemo` firmware. That firmware does nothing but blinking the LED. Blinking the LED you ask? It's already blinking! Well, the sequence is different, so it should be easily possible to distinguish the `blinkdemo` blinking from the update mode blinking.

Enough talking. Let's now upload the firmware:

```
(usbprog) upload blinkdemo
Opening device ...
Writing firmware ...
################################################################################
Starting device ...
Detecting new USB devices ...
```

Okay, that's it. Because you want to use the firmware now, remove the JP3 jumper so that the firmware starts after next reconnect instead of putting the USBprog in update mode.

[2] On Linux, you can watch that by reading the system log in `/var/log/syslog` or `/var/log/messages` or repeatedly executing the `dmesg` command.

3 Getting Started with the Software

3.1 Installation

3.1.1 Binary Packages for Linux

This section describes the installation of binary packages on wide-spread Linux distributions. If you have a more exotic Linux distribution, another Unix flavour or if you just want to use the latest and greatest version, proceed by reading section 3.1.4 on page 14.

In any case: After the installation has finished, the program can be started using `usbprog` in a shell for the command line version of the USBprog software and `usbprog-gui` for the graphical user interface (if installed). In the last case, USBprog should also appear in the start menu of your desktop environment[1], at least after logging out and logging in again.

Ubuntu/Debian

That's the easiest distribution because Uwe Herrman (uwe@debian.org) was so kind to provide Debian packages. Because everything that is in Debian is also in *Universe,* you also have that advantage on Ubuntu.

So: Just open a Terminal and install USBprog by entering

```
% sudo aptitude install usbprog
```

If you also want to use the graphical interface of the USBprog software, use

```
% sudo aptitude install usbprog usbprog-gui
```

The only "disadvantage" of installing the GUI is that it probably will install some dependencies like wxGtk.

[1] At least if you use KDE, GNOME or Xfce.

openSUSE and SLES

There are binary packages in the openSUSE Build Service, directly from the author. So they should be always up to date. They are in the *electronics* repository. To add the repository to your package list, open a shell and then enter

```
% sudo zypper ar -r http://repos.opensuse.org/electronics/@@DIST@@/electronics.repo
```

The term `@@DIST@@` has to be substituted version of your openSUSE or SLES as shown in table 3.1.

String	Distribution
`openSUSE_11.0`	openSUSE 11.0
`openSUSE_11.1`	openSUSE 11.1
`openSUSE_11.2`	openSUSE 11.2
`SUSE_Linux_Factory`	openSUSE Factory (the unstable development version)
`SLE_10`	SUSE Linux Enterprise (Desktop/Server) 10
`SLE_11`	SUSE Linux Enterprise (Desktop/Server) 11

Table 3.1: SUSE Distribution strings for the *electronics* Build Service repository

After that, refresh the package list with

```
% sudo zypper ref
```

You're now ready to install USBprog:

```
% sudo zypper install usbprog usbprog-gui
```

If you don't need the graphical user interface, just omit the `usbprog-gui` package. Of course, since the repository is now known by the package manager, you can also install or uninstall the USBprog packages using YaST.

Fedora, CentOS and RHEL

You may be surprised, but the binary packages for Fedora, Red Hat and CentOS are also built in the openSUSE Build Service. The only difference of that section compared with the previous section is that we describe `yum` instead of `zypper` here.

There's no command so add a repository in `yum`. Instead, you have to change to the directory `/etc/yum.repos.d` and download the repository file. In that case, you have to substitute `@@DIST@@` with values of table 3.2 on the following page.

```
% su
# cd /etc/yum.repos.d
# wget http://repos.opensuse.org/electronics/@@DIST@@/electronics.repo
```

Since the refresh is done automatically by `yum`, just install the package(s) with

String	Distribution
Fedora_11	Fedora 11
Fedora_12	Fedora 12
CentOS_5	CentOS 5
RHEL_5	Red Hat Enterprise Linux 5

Table 3.2: Distribution strings for Red Hat-like distributions in the Build Service

```
# yum install usbprog usbprog-gui
```

As with SUSE and Ubuntu, you can omit the `usbprog-gui` package if you don't plan to use the graphical user interface.

3.1.2 Windows

3.1.3 MacOS

3.1.4 Compiling USBprog from Source

Bibliography

[1] AVRDUDE Manual, http://www.nongnu.org/avrdude/user-manual/avrdude.html