# Direkte Programmierung mit Programmer

Mikrocontroller (u.a. auch die auf vielen Baugruppen verwendeten Atmel-Prozessoren) müssen vor der Benutzung mit einem Programmcode (Firmware) versehen werden. Es gibt verschiedene Methoden, eine Firmware erstmals in einen Mikrocontroller hineinzubringen, nachfolgend ist der Weg über die ISP-Schnittstelle beschrieben. Benötigt wird hierzu ein Gerät, welches diese Schnittstelle bedienen kann. Am unkompliziertesten für Atmel-Prozessoren hat sich der AVRISP MKII erwiesen.

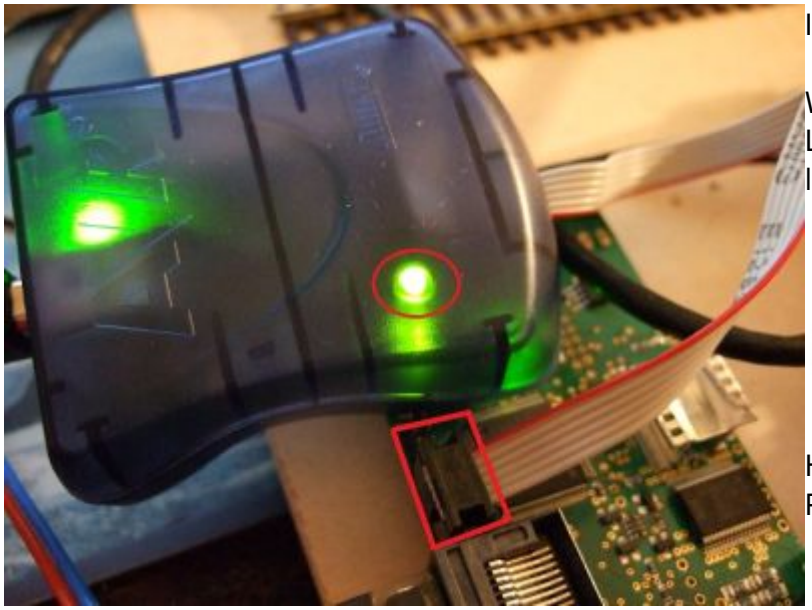
Da ich beim Programmieren der LightControl ein Problem damit hatte, den Pfostenstecker des Programmers in die PDI-Buchse der LightControl zu stecken - da war ein Wannenstecker im Weg - habe ich mir eine kleine „Verlängerung“ gelötet, die aus einem 6-poligen Pfostenstecker und einer daran angelöteten 6-poligen Pfostenleiste besteht. Auf dem Bild kann man die Verlängerung sehen.

Die Methode der direkten Programmierung kann man selbstverständlich auch bei der Aktualsierung der Baugruppe verwenden, die meisten Baugruppen unterstützen aber die Aktualisierung über die BiDiB-Tools.

## Anschluß Programmiergerät

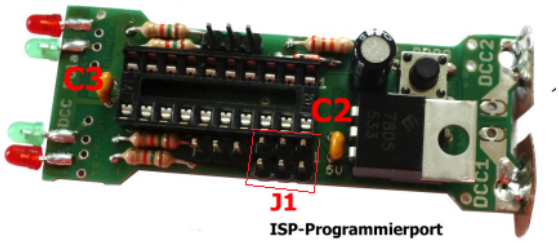
Um eine Firmware in einen AVR-Mikroprozessor zu laden, der bereits auf einer Platine verbaut ist, benutzt man die ISP-Schnittstelle. Das ist eine 6-polige Schnittstelle, die meistens in Form einer 2×3-poligen Stiftleiste auf der Platine vorhanden ist. Diese ISP-Schnittstelle verbindet man mit einem Programmiergerät. Das Programmiergerät hat außerdem einen USB-Anschluß, den man mit dem PC verbindet. Als letztes sollte man die Platine mit dem zu programmierenden Mikroprozessor mit Spannung versorgen.

Hier zwei Beispiele für den Anschluss des ISP-Steckers:



ISP am GBMBoost

Wichtig ist, das die, hier rot eingekreiste LED, grün leuchtet. Wenn sie gelb leuchtet, steckt der Stecker falsch herum.




Hier ist die ISP-Schnittstelle auf dem DCC-Pola rot markiert.

# Programmiersoftware

Nun braucht man noch Software, um die Firmware vom PC in den Mikroprozessor zu laden.

Dazu stehen mittlerweile einige gute Werkzeuge zur Verfügung. Es gibt umfangreiche Entwicklungswerkzeuge mit grafischer Benutzeroberfläche, aber auch schlanke Werkzeuge ohne unnötigen Ballast, die für verschiedene Betriebsysteme wie Linux, Windows und MacOS X geeignet sind.

## Atmel Studio 6

Die neueste grafische Entwicklungsumgebung von Atmel ist das Atmel Studio 6. Sie baut auf Microsofts Visual Studio auf und ist nur für Windows-Systeme geeignet.

Eine ausführliche Beschreibung zur Einrichtung findet sich hier im Kapitel Entwicklungsumgebung BiDiBone Programmierung sowie eine Beschreibung für das Aufspielen der Firmware auf eine BiDiBOne-Baugruppe.

## AVR Studio 4

Atmels ursprüngliche und sehr erfolgreiche Entwicklungswerkzeug ist AVR Studio 4.

Eine ausführliche Beschreibung findet sich auf den Seiten von Mikrocontroller.net

- AVR Studio 4
- Download

Hier fehlt noch Text.

## Eclipse C/C++

Auch die Integrierte Entwicklungsumgebung: Eclipse C/C++ kann man in Zusammenarbeit mit avrdude (siehe unten AVRDUDE) zur Programmierung der Mikroprozessoren verwenden.

Auf der Eclipse-Site stehen neben dem Grundpaket verschiedene Paket-Lösungen bereit. Für uns ist Eclipse C/C++ relevant, das wir von einem nahen Server herunterladen. Eclipse ist eine Java-Anwendung und wird in ein passendes Verzeichnis extrahiert, z.B. /library/eclipse4.3.2_cpp_64Bit oder C:\Programme\eclipse4.3.2_cpp_64Bit. Gestartet wird das Programm mit den üblichen Mitteln des verwendeten Betriebssystems (eclipse.exe, eclipse.app, ...).

Eclipse ist u.A. für die Betriebsysteme Linux, Windows und MacOS X geeignet und benötigt Java als Unterbau. Für unseren Fall reicht eine so genannte „Standard Edition Runtime Environment (JRE)“ ab Version 7 also JRE 7, wie sie auf vielen Rechnern vorhanden sein dürfte.

Ab Eclipse 4.3 (Kepler) benötigen wir nur noch ein Pug-In für den Zugriff auf unsere Hardware, z.B. AVR Eclipse Plugin. Darin sind alle benötigten Werkzeuge, wie AVRDUDE und die AVR-Toolchain enthalten.

> Für Windows-Systeme muss man die GNU-Werkzeuge separat laden, z.B. mit WinAVR. Zur Ansteuerung eines Programmers ist zusätzlich die libusb notwendig (mit Filter bei gleichzeitiger Verwendung von Atmel AVR/Studio).

Eine ausführliche Beschreibung für die weitere Einrichtung der IDE Eclipse findet sich hier im Kapitel Entwicklungsumgebung BiDiBone Programmierung bzw. Eclipse C/C++.

Weitere interessante Links:

- AVR-GCC-Tutorial (mikrocontroller.net)
- AVR Eclipse (mikrocontroller.net)
- AVR eclipse Plugin
- libusb für AVR ISP mk II unter Windows

## AVRDUDE

Es ist jedoch auch möglich avrdude, dass es für Linux und Windows gibt, als schlankes Tool alleine zu nutzen.

Meine Beispiele unten beziehen sich auf Linux. Für Windows läßt man einfach „sudo“ am Anfang der Zeile weg und ersetzt den Namen des USB-Ports (Option -P) durch den entsprechenden COM-Port.

Für Linux-Benutzer muß ich wahrscheinlich nichts weiter erklären. Die Windows-Benutzer müssen nun ein cmd.exe starten und dort den Befehl (ohne sudo) eingeben. Die Firmwaredateien werden im aktuellen Verzeichnis erwartet, ansonsten muß man den vollständigen Pfad zu den Firmwaredateien angeben. Tipp für Windowsanwender: Wenn man auf einen Ordner mit der rechten Maus klickt und dabei die Shifttaste gedrückt hält, erscheint im Kontextmenu die Auswahl 'Eingabeaufforderung im aktuellen Verzeichnis öffnen', das spart die lästige Pfadeingabe.

Achtung: Auch wenn hier auf der Wikiseite der Befehl in mehreren Zeilen dargestellt wird, muß er immer in einer Zeile geschrieben werden!

### GBM16T

```
sudo avrdude -v -P usb -c avrisp2 -p x128a1 -B10 -e -U flash:w:gbm16t.hex -U eeprom:w:gbm16t.eep -U fuse0:w:0xff:m -U fuse1:w:0xaa:m -U fuse2:w:0xba:m -U fuse4:w:0xff:m -U fuse5:w:0xeb:m
```

### GBMBoost

```
sudo avrdude -v -P usb -c avrisp2 -p x128a1 -B10 -e -U flash:w:gbmboost_master.hex -U eeprom:w:gbmboost_master.eep -U fuse0:w:0xff:m -U fuse1:w:0xaa:m -U fuse2:w:0xba:m -U fuse4:w:0xff:m -U fuse5:w:0xeb:m
```

### LightControl

```
sudo avrdude -v -P usb -c avrisp2 -p x128a1 -B10 -e -U flash:w:LightControl.hex -U eeprom:w:LightControl.eep -U fuse0:w:0xff:m -U fuse1:w:0xaa:m -U fuse2:w:0xba:m -U fuse4:w:0xff:m -U fuse5:w:0xeb:m
```

### MoBaLiSt

```
sudo avrdude -v -P usb -c avrisp2 -p m32 -B10 -e -U flash:w:mobalist.hex -U
eeprom:w:mobalist.eep
```

### OpenDCC Z1 mit Xpressnet

#### udev-Regel

```
KERNEL==„ttyUSB*“, ATTRS{product}==„USB-IF OpenDCC V1.2“,
SYMLINK+=„opendcc_z1“

avrdude -v -c avr911 -p m644p -P /dev/opendcc_z1 -U flash:w:OpenDCC_XP.hex -U
eeprom:w:OpenDCC_XP.eep -b 19200
```

### OpenDCC-Dekoder Version 1

```
sudo avrdude -v -P usb -c dragon_isp -p t2313 -e -B10 -U lfuse:w:0xee:m -U
hfuse:w:0xd9:m -U efuse:w:0xff:m -U flash:w:OpenDecoder.hex -U
eeprom:w:OpenDecoder.eep
```

From:
https://forum.opendcc.de/wiki/ - **BiDiB Wiki**

Permanent link:
**https://forum.opendcc.de/wiki/doku.php?id=programmer**

Last update: **2016/07/05 10:52**