

Programme für den Atmel ATMEGA8 in Eclipse schreiben und auf den AVR flashen unter Linux

Florian Soulier

10. Mai 2012

Inhaltsverzeichnis

1	Einleitung	2
2	Hardware/Software	2
3	Einrichten	3
4	Das erste Projekt	5
4.1	Projekt erstellen und Code einfügen	5
4.2	Flashen des AVR	8

1 Einleitung

In diesem Tutorial soll erklärt werden, wie man unter Linux Programme für einen AVR schreibt, kompiliert und auf den AVR flashed. Das flashen erfolgt über die RS232-Schnittstelle (auch wenn man das Board über ein USB Kabel verbunden hat, wird die RS232-Schnittstelle verwendet).

Ich habe mich auf der Suche nach einer Antwort auf diese Frage durch viele Foren gequält, aber nie konnte eine akzeptable Lösung gefunden werden. Die häufigsten Antworten waren:

- Der AVR kann mit dem Werkbootloader nicht mit freien Tools wie AvrDude oder RP6Launcher beschrieben werden. Man muss einen extra Bootloader (für Linux) installieren.
- Man braucht Zusatzhardware wie einen ISP Anschluss (der kostet nochmal Geld) und kann darüber flashen.
- Man muss sich eine virtuelle Maschine mit Windows aufsetzen und MyAVRProgTool verwenden.

Möglichkeit 1 ist nicht ganz falsch. Man benötigt einen anderen Bootloader, da die freien Programme nicht die entsprechenden Parameter übergeben. Mit dem Betriebssystem hat dies nichts zu tun. Möglichkeit 2 ist durchführbar, allerdings mit Arbeit und Geldaufwand verbunden. Außerdem ist es ein sehr mühsamer Weg für einen absoluten Anfänger, erst einen neuen Bootloader zu flashen. Möglichkeit 3 klappt natürlich, ist aber als Antwort auf die Frage schlichtweg falsch.

Allerdings geht es auch *viel* einfacher, dank eines genialen Skriptes von *Devkid*. Im Folgenden zeige ich auf, wie man auch ohne die oben genannten Punkte und ganz ohne Geldaufwand Programme schreiben und flashen kann.

2 Hardware/Software

Ich verwende in dem Tutorial den MyAVR Light Bausatz ¹ wie er auch im Praktikum verwendet wird.

Meine verwendete Software:

- Eclipse Version: 3.7.2
- Ubuntu 12.04 LTS
- Skript zum Flashen des AVR ²

¹<http://shop.myavr.de/index.php?sp=article.sp.php&artID=200017>

²<http://www.devid.net/programmierung/mikrocontroller-a-bastelzeug/10-programmierung-auf-dem-mysmartcontrol-ueber-den-bootloader-unter-linux>

3 Einrichten

Der Schnelligkeit halber klicken wir nicht wie in Windows alles zusammen, sondern arbeiten mit dem Terminal (*STRG+ALT+T*) oder über das Menü aufrufbar).

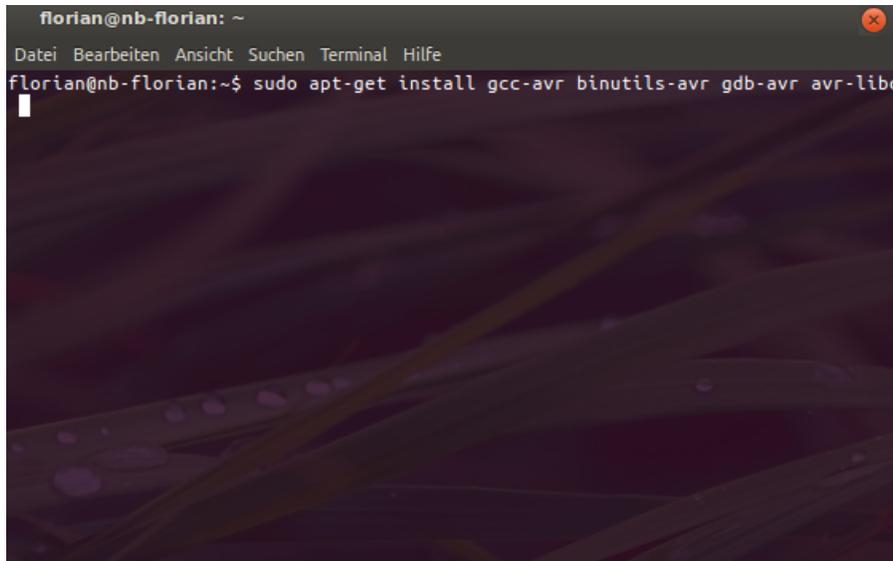


Abbildung 1: Pakete über das Terminal installieren

Pakete für den AVR installieren

```
$ sudo apt-get install gcc-avr binutils-avr gdb-avr avr-libc
```

Eclipse und die C-Development-Tools installieren

```
$ sudo apt-get install eclipse eclipse-cdt
```

Den eigenen Linux-Benutzer zusätzlich in die dialout-Gruppe hinzufügen

```
$ sudo usermod -aG dialout <benutzername>
```

Eclipse einrichten

Wir starten nun Eclipse. Nun klicken wir auf „Help/Install New Software“ und fügen eine neue Softwarequelle ein:

```
http://avr-eclipse.sourceforge.net/updatesite
```

Jetzt machen wir einen Haken bei AVR Eclipse Plugin und installieren es (vgl. Abbildung 2).

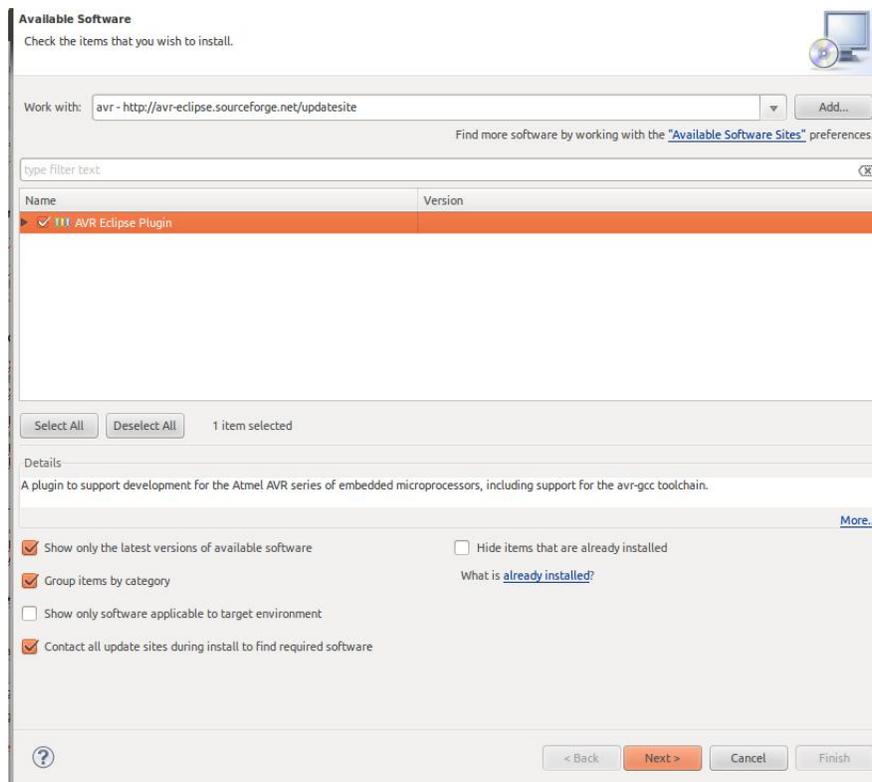


Abbildung 2: Eclipse Updatequelle einrichten

Anschließend wählen wir *Project/Properties* aus und öffnen dort die Reiter *C/C++ Build/Settings* und machen Haken bei

Generate HEX file for Flash memory

und

Generate Hex file for EEPROM

(Vergleiche dazu Abbildung 3).

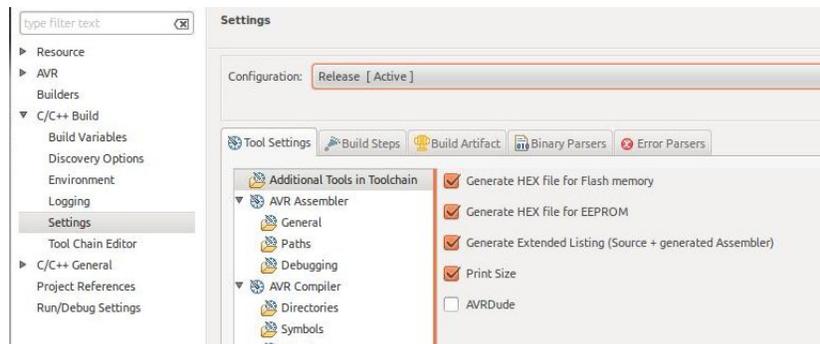


Abbildung 3: AVR-GCC (Compiler) einrichten

4 Das erste Projekt

Im Folgenden werden wir beispielhaft ein neues Projekt erstellen, ein Programm schreiben und eine *HEX-File* erzeugen. Anschließend werden wir den AVR mit dem Programm flashen.

4.1 Projekt erstellen und Code einfügen

Wir starten Eclipse und wählen *File/New/C-Project* aus. Nun erstellen wir ein neues leeres Projekt, indem wir unter *AVR Cross Target Application New empty Project* und die *AVR-GCC Toolchain* auswählen. Der Schritt ist in Abbildung 4 gezeigt.

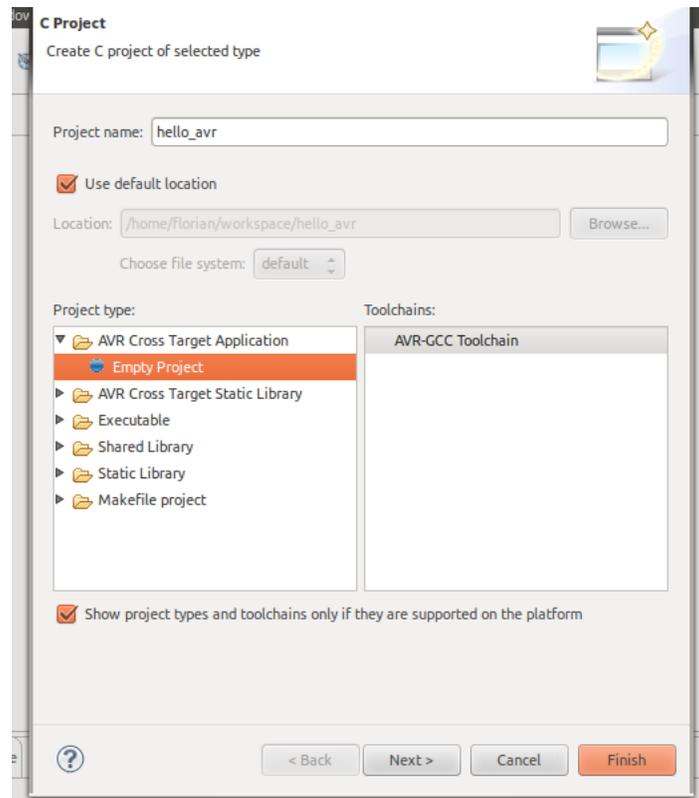


Abbildung 4: Ein C Projekt für den AVR erstellen

Nun ist das Projekt erstellt und wir können uns um den Quelltext kümmern. Dafür müssen wir aber erst eine Source-File erstellen. Dazu gehen wir wie in Abbildung 5 vor.

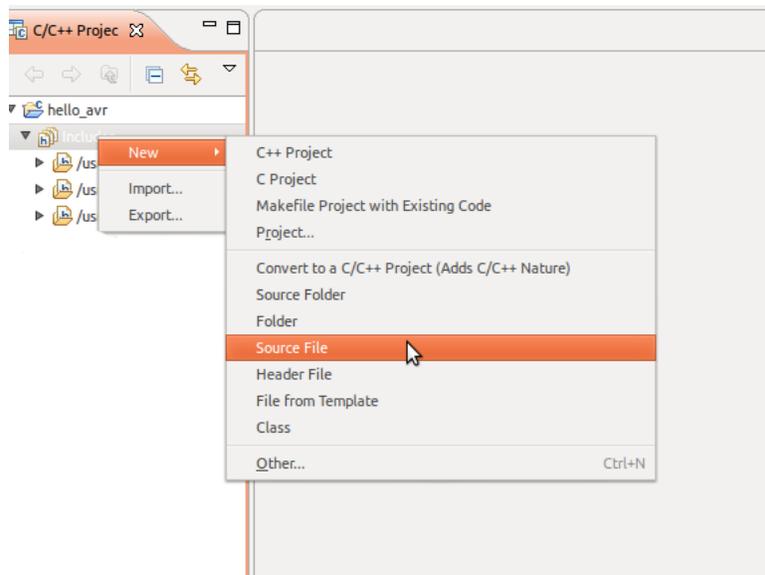


Abbildung 5: Eine C-Quelltextdatei erzeugen

Als Source-Folder geben wir natürlich den Projektordner an. Die Source-File darf man beliebig benennen, es macht aber natürlich Sinn, die Datei wie das Projekt zu benennen. Das macht die Sache übersichtlicher und für einen anderen Programmierer leichter zu durchschauen.

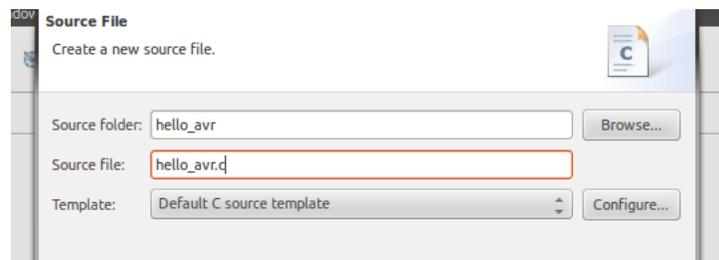


Abbildung 6: Speicherort und Dateinamen festlegen

Jetzt ist es endlich so weit. Die eigentliche Programmierarbeit beginnt. Um den Ablauf einmal ganz zu durchlaufen, geben wir uns mit einem simplen Programm zu Frieden, das die rote LED dauerhaft leuchten lässt. Der anspruchsvolle Quellcode umfasst folgende Zeilen:

```
#include <avr/io.h>

int main(void)
{
    DDRB=(1 << PB1);
    while(1)
    {
        PORTB=0x02;
    }
    return 0;
}
```

Kompilieren

Nach dem schnellem Kopieren des Quelltextes wählen wir unter *Project/Build Configurations/Set Active/Release* einmalig den Release-Mode aus. Das ist wichtig, denn sonst wird uns keine *HEX-File* erstellt.

Jetzt können wir auf *Project/Build All* oder mit *STRG+B* die *Hex-Datei* erstellen lassen. Wenn wir jetzt in den Release-Ordner hineinschauen (liegt im Projektordner), dann finden wir die entsprechende *HEX-Datei*.

4.2 Flashen des AVR

Hier geht es um die eigentliche Problemstellung, welche Anlass für mich gewesen ist, überhaupt ein Tutorial zu schreiben. Die Frage die in etlichen Foren kursiert: „Wieso kann ich meinen AVR nur mit MyAvrProgTool beschreiben?“ Das Problem ist, dass die Übergabeparameter für den AVR nicht öffentlich zugänglich gemacht wurden und das Programmierool sich nicht unter Linux emulieren lässt. Es gibt aber eine Lösung für das Problem und zwar findet man diese im Blog von Devkid: <http://www.devkid.net/programmierung/mikrocontroller-a-bastelzeug/10-programmierung-auf-dem-mysmartcontrol-ueber-den-bootloader-unter-linux>. Devkid hat ein Python-Skript geschrieben, um den AVR unter Linux zu flashen. Im Folgenden gebe ich eine Erklärung ab, wie man damit den AVR flashed.

Vorbereitungen

Vor dem Flashvorgang müssen wir allerdings noch eine Vorbereitung treffen: Damit der μ C nachher auch die LED leuchten lassen kann, müssen wir mit einem Kabel aushelfen. Dazu verbinden wir PORTB PB1 mit LED-Rot. Dabei kann nicht so viel passieren, wenn du sicher gehen willst, achte darauf, dass der μ C nicht am USB Port deines PCs hängt. Das Ergebnis sollte etwa so wie in Abbildung 7 aussehen.

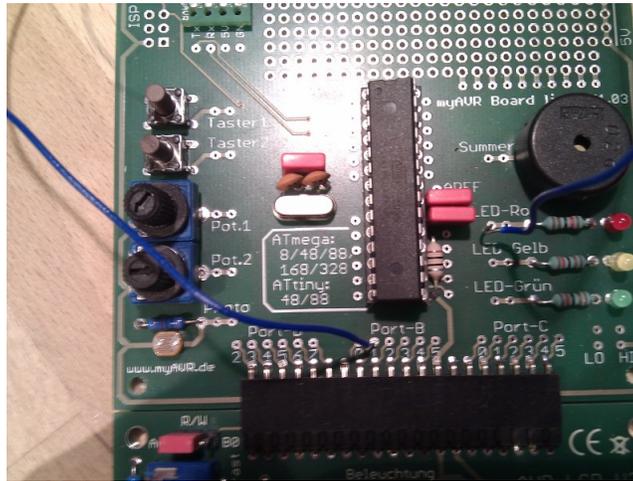


Abbildung 7: LED mit PB1 verbinden

Flashen

Nun widmen wir uns endlich dem Flashen. Wir brauchen dazu das Shell-Skript *flash.sh* und das Python-Skript *myavr_smartcontrol_bootloader.py*. Diese zwei Dateien kopieren wir in den Release-Ordner des Projektes (falls kein Release-Ordner vorhanden ist, führe die Schritte in Kapitel 4.1 durch).

Das Python-Skript kann Binaries auf den AVR übertragen. Nun erstellen wir mit Eclipse aber keine *Binaries* sondern *HEX-Files*. Um das gewünschte Ergebnis zu erreichen, müssen wir die *HEX-File* zu erst in eine *Binary* umwandeln. Das geht mit Hilfe des Terminals. Öffne das Terminal und navigiere mit *cd* in den Release-Ordner deines Projektes. Nun führe folgenden Befehl aus:

```
$ avr-objcopy -I ihex -O binary <name der hex> code.bin
```

<name der hex> muss natürlich durch die Hex-Datei ersetzt werden (Zum Beispiel: *hello_avr.hex*).

Damit haben wir eine Binary mit dem Namen *code.bin* erzeugt. Anschließend möchten wir mit Hilfe des gerade kopierten Python-Skripts die *code.bin* auf den AVR flashen. Das geht folgendermaßen:

```
$ python myavr_smartcontrol_bootloader.py code.bin
```

Wenn du alles richtig gemacht hast, sollte jetzt deine rote LED leuchten und in deinem Terminal etwas in der Art stehen:

```
florian@nb-florian:~/workspace/hello_avr/Release$ ./flash.sh hello_avr.hex
Connecting to chip, press the RESET button!
.....

Connected successfully, hardware info:
Hardware:      mySmartControl
Hardware version: V1.02
Software version: V1.01
Processor:     ATmega8
Processor clock: 3686400
Block size:    192
Max. size:     6144
Baudrate:      38400

Writing program...
.
Finished writing!
florian@nb-florian:~/workspace/hello_avr/Release$
```

Abbildung 8: Ergebnis des Flashs

Zeitsparen

Nun sind das viele Arbeitsschritte. Erst mit Eclipse die hex kompilieren, dann mit `avr-objcopy` die binary erstellen und dann mit dem Python-Skript den AVR flashen. Man kann sich das Leben auch etwas vereinfachen, sodass wir nur so viele Arbeitsschritte haben, wie die Windows-Flasher.

Im Ordner liegt noch ein Shell-Skript namens `flash.sh`. Man kann die Schritte ab der Überschrift *Flashen* zusammenfassen, in dem man im Release Ordner folgenden Befehl in dem Terminal ausführt:

```
$ ./flash.sh <dateiname.hex>
```

Du musst dann nur noch in Eclipse die *HEX-File* erstellen und anschließend im Terminal das Shell-Skript ausführen.