Openhardware Workshop – ATmega8 Demoboard

Chemnitzer Linux-Tage 2008
Andreas Heik <andreas.heik@linux-tage.de>

im März 2008

Zusammenfassung

Ob Fahrradcomputer oder (LEGO)-Roboter, Mikrocontroller findet man in vielen Geräten. Die kleinen Bausteine sind aber oft auch Teil beliebter Bastelprojekte. Im Workshop werden Werkzeuge zur Programmierung von Atmel AVRs vorgestellt. Die Lücke zur Hardware schließt eine kleine Demo-Platine mit USB-Anschluss, Infrarotempfänger und Display.

Inhaltsverzeichnis

1 Allgemeine Hinweise

1.1 Funktion der Schaltung

Die Schaltung wurde mehrfach aufgebaut und getestet. Trotzdem können wir nicht sicherstellen, dass die Schaltung unter allen Bedingungen funktioniert. Insbesondere die Spezifikation der USB-Datenpegel von 3.3V und die Betriebsspannung von 5V können vereinzelt zu Problemen führen. Für Schäden, z.B. an Hardware können wir keine Haftung übernehmen!

1.2 Hinweise zum Löten

- Lötkolben werden bis zu 400° heiß, Verbrennungs- und Brandgefahr!
- Flußmitteldämpfe nicht einatmen!
- Lötzinn enthält Blei, während der Arbeiten nicht Essen oder Trinken, nach den Arbeiten Hände waschen!
- Augen schützen, Spritzgefahr durch flüssiges Zinn (z.B. beim Auslöten)!

2 Bestückung der Platine

2.1 Bauteilliste

C1	10μ	Q1	Quarz 12 MHz (HC18)
C2	$4\mu7$	R1	68R
C3	100nF	R2	68R
C4	10μ	R3	1.5k
C5	22pF	R4	100R
C6	22pF	R5	220R
D1	3.6V	R6	10k
D2	3.6V	R7	22R
IC1	Atmel MEGA8-P	R8	4k7
	IC Sockel (28 polig)	SV1	Stiftleiste (2x5 polig)
L1	10μH	SV2	Buchsenleiste (1x4 polig)
LCD1	Peaktech 162B (16x2)	T1	BC547 (TO92)
	Stiftleiste (1x16 polig)	TSOP1738	IR Empfangsmodul TSOP1738
	Buchsenleiste (1x16 polig)	X1	USB-Buchse, Serie B
LED1	LED 3mm (rot)		

2.2 Bestückungsplan

Der Bestückungsplan befindet sich im Anhang als zusätzliches Dokument.

2.3 Reihenfolge der Bestückung

Bestücken Sie zu Anfang die 2 Drahtbrücken (im Bestückungsplan rot markiert) zwischen $Pin1$, $Pin19$ des Mikrocontrollers und dem ISP-Sockel.
Im 2. Schritt wird der Stiftsockel für den ISP und die IC–Fassung für den Mikrocontroller eingelötet. An der IC–Fassung ist es ausreichend, die benutzen Pins zu verlöten.
Schritt 3 sieht die Montage von $C3$, $C5$, $C6$ und dem Quarz vor.
Nachdem im Schritt 4 der Elko $C1$, der Widerstand $R6$ und die USB-Buchse montiert wurden, kann eine erste Funktionsprüfung erfolgen. Stecken Sie dazu den Mikrocontroller in die Fassung und schließen Sie die Schaltung über die USB-Buchse an ein Labornetzteil an. Bei 5V sollte ein Strom von nicht mehr als 15mA fließen. Über den ISP-Sockel kann der Mikrocontroller getestet werden. Die Programmiersoftware muss die Signatur des Controllers erkennen (*avrdude* im Terminal-Modus. Nachdem die Fuse-Bits des Mikrocontroller für die externe Taktquelle programmiert wurden, sollte an den Pins $XTAL1$ und $XTAL2$ eine Frequenz von 12MHz messbar sein.
Vor weiteren Lötarbeiten sollte der Mikrocontroller wieder aus der Fassung entfernt werden. Im 5. Schritt werden die Widerstände $R1$, $R2$ und $R3$ für die USB–Anbindung montiert.
Wer sich für den Infrarotempfänger entscheidet montiert im folgenden die Bauteile $R4$, $R5$, $C2$, die $LED1$ und den $TSOP1738$. Für die LCD–Modul Ansteuerung müssen die Bauteile $R7$, $R8$, $L1$, $C4$ und $T1$ eingelötet werden. Das LCD–Modul wird über Stift- und Buchsenleisten steckbar montiert wobei die Stiftleiste am LCD–Modul angelötet wird.

> *Beachte:* Der Mikrocontroller und die IC–Fassung sind mit einer Kerbe versehen. Elkos sind polrichtig einzubauen, am Gehäuse ist meist der $\ominus$ Pol markiert. Die Kathode der Z-Dioden ist dunkel eingefärbt, die LED ist an der Kathode abgeflacht.

3 Schaltungsaufbau

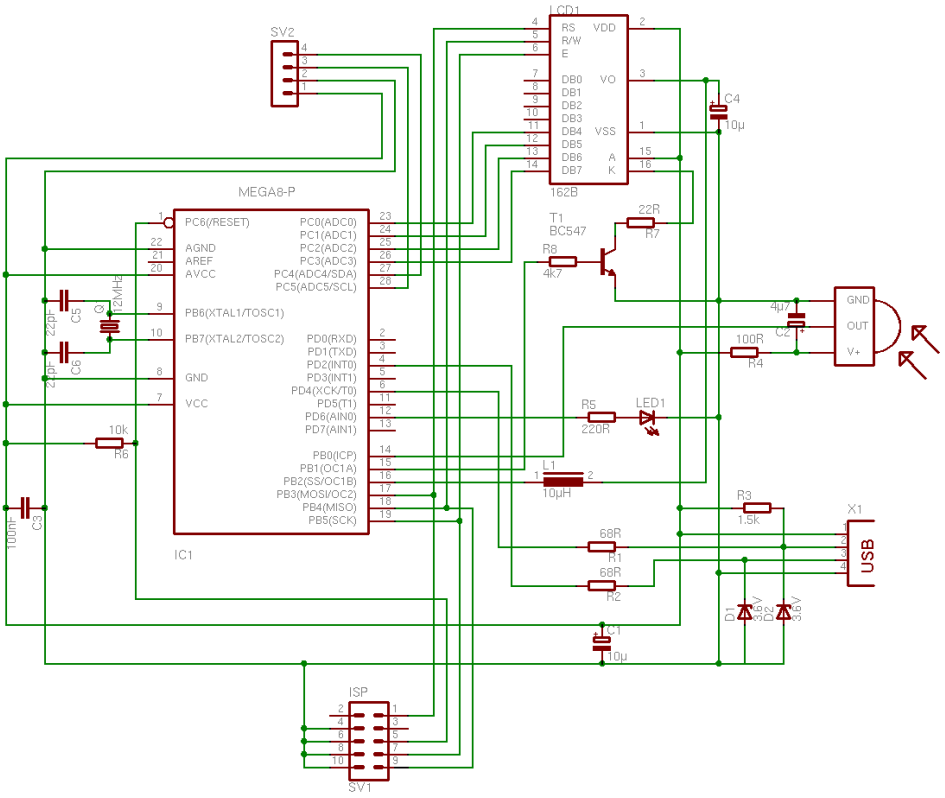

Abbildung 1: ATmega8 Demoboard

3.1 Mikrocontroller

Der AVR Mikrocontroller ATmega8 bietet für das Demoboard ausreichend Speicher (8kB Flash Programmspeicher, 1kB SRAM, 512B EEPROM) für kleine Anwendungen und I/O–Pins für die Programmierung und Anbindung peripherer Einheiten. Hierbei handelt es sich um einen 8–Bit RISC–Prozessor in Havard–Architektur mit getrennten Speicherbereichen für das Anwendungsprogramm und Datenstrukturen. Die Taktung erfolgt über einen externen Oszillator. Die Frequenz wurde auf 12MHz eingestellt, um die Kompatibilität mit dem Software USB–Stack zu gewährleisten.

3.2 Stromversorgung via USB

Der USB–Port versorgt die Schaltung mit 5V. Maximal 500mA Stromaufnahme sind für den Betrieb des AVR einschließlich Peripherie völlig ausreichend. Die Stromaufnahme des Mikrocontroller liegt im Betriebszustand bei

etwa 15mA. Für die Betriebsspannung wird ein Bereich von 2.7V bis 5.5V angegeben.

3.3 LED–Signalisierung

Die LED am Port $PD6$ kann zur Anzeige von Zuständen des Mikrocontroller genutzt werden. Voraussetzung dafür ist die Programmierung dieses Pins als Ausgang. Im Projekt USBtiny–ir wird die LED zur Signalisierung eingehender Infrarotsignale genutzt.

3.4 ISP–Anschluss

ISP steht für In–System–Programming. D.h. der Mikrocontroller kann im unter Spannung stehendem System programmiert werden. Programmieren bedeutet hierbei das Anwendungsprogramm in den Flash–Speicher zu schreiben. Das Anwendungsprogramme selbst wird mit einem Cross–Compiler erzeugt.
Das Demoboard verfügt über einen 10 poligen ISP–Stecksockel. Die Pinbelegung folgt der Spezifikation von Atmel (auch Kanda-Standard) und kann mit üblichen Programmiergeräten benutzt werden (z.B. STK500). Außerdem sind viele Schaltungsbeispiele für einfache Programmiergeräte verfügbar. Zum Programmiergerät wird noch die Programmiersoftware benötigt. Auf dem Gebiet der Open–Source–Anwendungen wird häufig auf *avrdude* verwiesen, da diese Anwendung mit vielen Programmiergeräte genutzt werden kann.

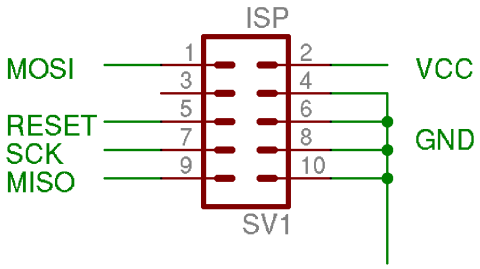

Abbildung 2: Anschlussbelegung des ISP-Sockels (Bestückungsseite)

3.5 USB–Dateninterface

USB–Geräte nutzen 2 Datenleitungen $D+$ und $D-$ mit Differenzsignal für die Kommunikation. High-Pegel an $D-$ signalisiert ein Lowspeed Gerät (wird durch $R3$ realisiert).
Softwareimplementation des USB-Stack für AVR Mikrocontroller gibt es in verschiedenen Ausprägungen. USBtiny steht unter der GPL und wird deshalb in den Projekten bevorzugt. Die Implementationen nutzen i.A. 2 I/O–

Pins zur Kommunikation sowie einen Interrupt–Pin. Die Einbindung der USB–Routinen in die Anwendung erfolgt durch die `usb_poll()`–Funktion in der main–loop. Die Behandlung von USB–Requests wird in der `usb_setup()`–Funktion realisiert.
Übrigens gibt es auch interessante Implementation von USB–Human Interface Devices.

> *Beachte:* Die USB–Spezifikation definiert 3.3V für die Datenpegel. Vereinzelte Probleme an USB–Hostcontrollern durch den Betrieb des Mikrocontrollers mit 5V sollen durch die Zenerdioden $D1$ und $D2$ gelöst werden. In den meisten Fällen sind die Dioden nicht erforderlich und sollten deshalb auch nur bei Bedarf bestückt werden.

3.6 LCD–Display

Zum Einsatz kommt ein Punktmatrix–LCD–Modul mit dem weit verbreiteten HD 44780 Controller bzw. Kompatiblen (KS0060). Der Betrieb erfolgt im 4–Bit Modus. D.h. neben 4 Datenleitungen sind noch 3 Steuerleitungen erforderlich. Für die Initialisierung und Ansteuerung des Displays sind verschiedene Bibliotheksfunktionen verfügbar. Die Pinbelegung sowie die Abfolge zur Initialisierung sind im Datenblatt zum Displaycontroller beschrieben.

Kontrastreglung

Die Kontrasteinstellung des LCD–Modul erfolgt über eine veränderliche Spannung. Im einfachsten Fall kann das durch einen einstellbaren Widerstand realisiert werden.
Der eingesetzte Mikrocontroller verfügt über einen PWM–Modus (Pulsweitenmodulation). Damit lässt sich am Ausgang $OC1B$ ein Rechtecksignal mit variabler Pulsbreite erzeugen. Ein LC–Glied ($L1$ und $C4$) glättet das Signal zu einer Gleichspannung. Der PWM–Ausgang $OC1B$ läuft im invertierten Modus, da mit einer kleiner Spannung ein höherer Kontrast erzielt wird.

Hintergrundbeleuchtung

Die Ansteuerung der Hintergrundbeleuchtung des LCD–Modul erfolgt analog der Kontrastreglung mittels Pulsweitenmodulation. Bedingt durch die Stromaufnahme der Hintergrundbeleuchtung bis ca. 100mA wird der Ausgang des Mikrocontrollers über einen Transistor $T1$ getrieben. Der Widerstand $R7$ begrenzt den maximal fließenden Strom.

3.7 IR–Empfänger

Der IR–Empfänger $TSOP1738$ ist für Infrarotfernbedienungen mit 38kHz Modulationsfrequenz ausgelegt. Das demoduliertes IR–Signal wird an den

Input Capture Pin $ICP1$ des Mikrocontrollers geleitet. Der Input Capture Pin kann in Kombination mit einem Timer benutzt werden um zwischen jedem Flankenwechsel einen Zeitstempel zu erzeugen.
Die Differenzen der Zeitstempel beschreiben die Dauer zwischen zwei Flankenwechseln des IR–Signales und werden in einer Datenstruktur gespeichert. Die Datenstruktur (Differenzen der Zeitstempel) wird auf Anforderung an den Host übermittelt.
Die Dekodierung des Signals erfolgt auf dem Host (z.B. LIRC).

4 AVR Projekte für das Demoboard

Das Demoboard wurde so ausgelegt, dass neben eigenen Anwendungen die Projekte *USBtiny–ir* oder *LCD2USB* mit wenigen Anpassung betrieben werden können. Die angepassten Anwendung mit Quellcode sind im Downloadbereich verfügbar.

> *Beachte:* Beide Projekte sind für einen Takt von 12 MHz (externes Quarz) ausgelegt. Diese Einstellung muss einmalig durch programmieren der Fuse–Bits erfolgen.

4.1 USBtiny–ir

Das Projekt *USBtiny–ir* bildet einen Empfänger für Infrarot–Fernbedienungen mit USB–Anschluss. Empfangene IR–Signale werden über die LED angezeigt. Aus Sicht des USB–Protokolls ist die Anwendung zu *IgorPlug-USB* kompatibel und kann mit folgenden Softwareprojekten genutzt werden:

- LIRC
- VDR-Plugin Usbremote

4.2 LCD2USB

Das Projekt *LCD2USB* ermöglicht die Ansteuerung eines Punkmatrix LCD–Moduls über USB. Der Betrieb dieser Hardwarekombination ist mit folgenden Softwareprojekten möglich:

- LCD4Linux
- lcdproc (ab 0.5.2)

Das Projekt *LCD4Linux* ist für die Darstellung von Systeminformationen des Host konzipiert. Das Projekt *LCDproc* basiert auf einem Client–Server–Framework, welches die Darstellung von Informationen nur durch die verfügbaren Clients begrenzt.

5 AVR Anwendungsentwicklung

Nachfolgendes Kapitel gibt einen Einstieg in die Programmierung von AVR Mikrocontroller am Beispiel des ATmega8 Demoboards.

5.1 Programmierwerkzeuge

Für die Entwicklung von Anwendungen für AVR Mikrocontroller sind ein Cross–Compiler sowie die entsprechenden Bibliotheken mit controllerspezifischen Funktionen erforderlich.

- avr-binutils
- avr-gcc
- avr-libc

Alle Werkzeuge (ausgenommen avr-libc) stammen von den gleichnamigen GNU-Tools ab und werden beim Übersetzen mit dem Flag *–target=avr* erzeugt. Da die Pakete untereinander Abhängigkeiten aufweisen, ist die Übersetzung und Installation in der aufgeführten Reihenfolge erforderlich.

5.2 Programmablauf

Ein typischer Programmablauf einer AVR Anwendung startet mit der Initialisierung von Variablen, Ein-, Ausgängen und Peripheriebausteinen. Die eigentliche Anwendung wird meist in einer Endlosschleife und/oder als Interruptroutine realisiert.

5.3 Programmierbeispiel

Das klassische *Hello World* ist als Einstieg in die Programmierung von I/O-Pins ungeeignet. Für die Anzeige von Programmzuständen bietet sich aber die Signalisierung mittels der LED auf dem Demoboard an.

5.3.1 Quellcode

```
#include <avr/io.h>
#define F_CPU 12000000UL      // Quarz 12MHz
#include <util/delay.h>

// _delay_ms
// maximal Delay = 262.14 ms/F_CPU in MHz

void long_delay(uint16_t ms)
{
    for (; ms>0; ms--) _delay_ms(1);
```

```
}

int main( void )
{
    DDRD = (1 << PD6);              // PD6 an PORTD als Ausgang

    while( 1 ) {                    // Endlosschleife

        PORTD |= (1 << PD6);        // setzt PD6 an PortD auf 1
        long_delay(1000);           // warten ...

        PORTD &= ~(1 << PD6);       // loescht PD6 an PortD
        long_delay(1000);           // warten ...
    }

    return 0;
}
```

Der Quelltext demonstriert, wie I/O–Pin *PD*6 als Ausgang definiert und wechselweise auf H- und L-pegel gelegt wird. Zu beachten ist, dass die delay-Routinen aus einer Folge leerer Taktzyklen (nop) bestehen. Während dieser delays "warte" der Mikrocontroller.

5.3.2 Kompilierung

Der Übersetzungsprozess lässt sich mittels *Makefile* gut automatisieren. Aus jedem zum Projekt gehörenden C-File erzeugt der Compiler ein Objektfile. Anschließend werden die Objekte mit den Bibliotheken zu einem Binärfile gelinkt. Das für den Mikrocontroller erforderliche *ihex*-Format erfolgt durch Konvertierung aus dem Binärfile.

```
$ make
avr-gcc -Os -g -Wall -I. -mmcu=atmega8 -c -o blink.o blink.c
avr-gcc -g -mmcu=atmega8 -o blink.elf blink.o
avr-objcopy -j .text -j .data -O ihex blink.elf blink.hex
```

Ggf. kann eine Überprüfung des vom Programm benötigten Speicherbedarfs mit den Ressourcen des Mikrocontrollers erfolgen.

5.3.3 Programmieren des Flash-Speichers

Voraussetzung für die Übertragung der AVR Anwendung in den Flash-Speicher des Mikrocontrollers ist ein Programmiergerät und eine passende Programmiersoftware.

Das Programmiergerät verbindet den ISP-Sockel des Demoboards mit einer Schnittstelle am PC. Hinweise zu einfachen Programmiergeräten finden Sie im folgenden Kapitel.

Die Programmiersoftware läuft auf dem PC und kommuniziert über das Programmiergerät mit dem Mikrocontroller. Üblicherweise wird auf diesem Weg der Flash–Speicher des Mikrocontrollers beschrieben ("gebrannt"). Aber

es lassen sich z.B. auch die Fuse–Bits ändern oder der Inhalt des EEPROMs lesen bzw. schreiben.

Im Rahmen der GNU-Tools steht mit *avrdude* eine leistungsfähige Programmiersoftware zur Verfügung. *avrdude* lässt sich vollständig per Kommandozeile steuern und eignet sich deshalb hervorragend für die Einbindung in *Makefiles*. Ein interaktiver Modus kann mit der Option *-t* für den Terminal-Mode aktiviert werden.

```
$ make flash
avrdude -p atmega8 -c bsd -U flash:w:blink.hex
```

5.4 Bootloader

Als Alternative zum Programmieren des Flash-Speichers mittels Programmiergerät kann auch ein Bootloader zum Einsatz kommen. Das Verfahren wird auch als Read-While-Write Self-Programming bezeichnet.

Der Bootloader ist dabei eine AVR Anwendung, welche in der Bootloader Section (*No Read-While-Write Section*) des Flash-Speichers abgelegt ist. Über eine Bedingung (z.B. PIN auf Low-Pegel) wird das Verhalten des Bootloaders definiert.

Nach dem Anlegen der Betriebsspannung bzw. nach einem Reset wird der Code des Bootloaders gestartet.

Im Normalfall (Bedingung nicht erfüllt) springt der Bootloader zur Startadresse der Application Flash Section und startet die AVR Anwendung. Im Programmiermodus (Bedingung erfüllt) initialisiert der Bootloader eine I/O-Schnittstelle (z.B. UART), liest von dieser Daten und beschreibt damit die Application Flash Section (*Read-While-Write Section*).

Die Aufteilung des Flash-Speichers in Application Flash Section und Bootloader Section erfolgt mittels der Fuse-Bits. Der Flash-Speicher eines ATmega8-Mikrocontrollers läßt sich beispielsweise in $6kbyte$ Application Flash und $2kbyte$ Bootloader Section teilen. D.h. der Bootloader-Code muß im Beispiel so übersetzt werden, daß dieser ab Adresse $0x1800$ beginnt.

Für die erstmalige Installation eines Bootloaders ist ein Programmiergerät erforderlich.

5.4.1 USBaspLoader

Das Projekt *USBaspLoader* stellt einen Bootloader mit Software USB–Stack bereit, welcher sich im Programmiermodus wie ein *USBasp*–Programmiergerät verhält.

Im Beispielcode ist die Bedingung für den Bootloader auf den I/O-Port $PC5$ aktiv Low–Pegel definiert.

6 Programmiergeräte

6.1 Brian Dean's Programmer

Beispielhaft sei hier auf die einfachste Lösung für den Parallelport verwiesen. Das Programmiergerät wird vom avrdude als bsd unterstützt.

25 pol. Parallelport	AVR
Pin 7	RESET
Pin 8	SCK (clock input)
Pin 9	MOSI (instruction in)
Pin 10	MISO (data out)
Pin 18	GND

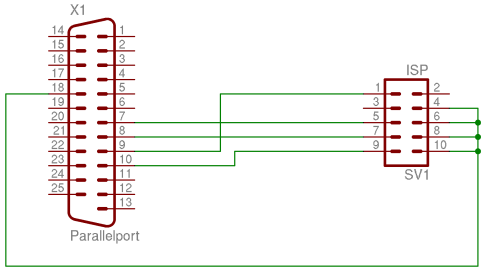

Abbildung 3: Schaltung – BSD Programmer

6.2 USBtinyISP

Natürlich lässt sich auf Basis des USB–Stacks *USBtiny* auch ein Programmiergerät mit wenigen Bauteilen entwerfen. USBtinyISP bietet eine Anregung dazu.

7 Anhang

7.1 FAQ

Wie wird das Projekt usbtiny–ir programmiert und getestet?

```
$ cd atmega8_demoboard/usbtiny-1.3_atmega8/ir/
$ make fuse          # programmieren der Fuse-Bits
$ make flash         # programmieren der Anwendung
```

```
$ lsusb
Bus 002 Device 008: ID 03eb:0002 Atmel Corp.
```

```
$ cd atmega8_demoboard/usbtiny-1.3_atmega8/read_igorplug/
$ sudo ./igorplug     # als root, Zugriff auf USB-Gerät
avr_open o.k.
Send: 1 read: 1       # Echo-Test
Send: 2 read: 2
Send: 3 read: 3
Send: 4 read: 4
RC Code: 15           # Fernbedienungscodes (RC5)
RC Code: 32800
```

Wie wird das Projekt lcd2usb programmiert und getestet?

```
$ cd atmega8_demoboard/lcd2usb/firmware/
$ make fuse           # programmieren der Fuse-Bits
$ make flash          # programmieren der Anwendung
```

```
$ lsusb
Bus 002 Device 009: ID 0403:c630 Future..lcd2usb interface

$ cd atmega8_demoboard/lcd2usb/testapp/
$ sudo ./lcd2usb      # als root, Zugriff auf USB-Gerät
--       LCD2USB test application       --
--       (c) 2006 by Till Harbaum       --
-- http://www.harbaum.org/till/lcd2usb --
Found LCD2USB device on bus 002 device 009.
Echo test successful!
Firmware version 2.8
Installed controllers: CTRL0
Keys: 0:off 1:off
```

Warum kann ich nicht auf mein usbtiny–ir zugreifen?

Voraussetzung für den erfolgreichen Test ist die Registrierung als USB-Gerät. Dazu die Ausgaben von `lsusb` prüfen.
Ggf. kann das Gerät bereits durch den in verschiedenen Linux–Distributionen enthaltenen Kernelmodul `lirc_igorplugusb` belegt sein. In diesem Fall muss diese Kernelmodule entladen werden.

Wie wird der Bootloader USBaspLoader installiert?

Voraussetzung für die Installation des Bootloader ist ein Programmiergerät, z.B. Brian Dean's Programmer. Beim programmieren der Fuse-Bits ist zu beachten, daß das BOOTRST-Feature aktiviert ist (0).

```
$ cd atmega8_demoboard/bootloader/USBaspLoader/firmware
$ make fuse           # programmieren der Fuse-Bits
$ make flash          # programmieren des Bootloader
```

Wie wird eine Anwendung mit dem Bootloader USBaspLoader installiert?

Damit der Bootloader aktiv wird, ist die Bedingung $PC5$ auf Low–Pegel zu erfüllen. Dazu wird eine Brücke (Jumper) auf die Pins 2 und 3 von SV2 gelegt.
Nach anschließen des Demoboards an einen USB-Port ist ein USB-Gerät mit der USB–ID $16c0 : 05dc$ verfügbar.

```
$ lsusb
Bus 001 Device 029: ID 16c0:05dc
```

Die Programmierung des Flash-Speichers kann jetzt mittels *avrdude* erfolgen. Als Programmiergerätetyp wird *usbasp* verwendet.
Beachte, ggf. sind root-Rechte für den Zugriff auf das USB-Gerät *usbasp* erforderlich.

Wie gelange ich in den Terminalmodus von avrdude

```
$ avrdude -c bsd -p atmega8 -t
```

Option	Argument	Beschreibung
-c	bsd	Typ des Programmiergerätes (siehe dazu `/etc/avrdude/avrdude.conf`)
-p	atmega8	Typ des zu programmierenden Mikrocontrollers
-t		Startet im Terminal–Modus

7.2 Links

- Downloadbereich Chemnitzer Linux–Tage 2008
  [http://chemnitzer.linux-tage.de/2008/vortraege/openhardware.html]
- IgorPlug-USB
  [http://www.cesko.host.sk/IgorPlugUSB/IgorPlug-USB%20(AVR)_eng.htm]
- GNU-Tools
  [http://www.gnu.org/software/]
- AVR-Libc
  [http://www.nongnu.org/avr-libc/]
- AVRDUDE
  [http://www.bsdhome.com/avrdude/]
- USBtiny–ir
  [http://www.xs4all.nl/ dicks/avr/usbtiny/]
- LIRC
  [http://www.lirc.org/]

- VDR-Plugin Usbremote
  [http://www.vdr-wiki.de/wiki/index.php/Usbremote-plugin]
- LCD2USB
  [http://www.harbaum.org/till/lcd2usb/]
- LCD4Linux
  [http://lcd4linux.bulix.org/]
- LCDproc
  [http://lcdproc.omnipotent.net/]
- Brian Dean's Programmer
  [http://www.bsdhome.com/avrdude/]
- USBtinyISP
  [http://www.ladyada.net/make/usbtinyisp/]
- USBaspLoader
  [http://www.obdev.at/products/avrusb/usbasploader.html]

7.3 Farbcode für Widerstände

Farbe	erster Ring erste Ziffer	zweiter Ring zweite Ziffer	dritter Ring Multiplikator	vierter Ring Toleranz
schwarz	0	0	1	
braun	1	1	10	1%
rot	2	2	100	2%
orange	3	3	1 000	
gelb	4	4	10 000	
grün	5	5	100 000	
blau	6	6	1 000 000	
violett	7	7	10 000 000	
grau	8	8	100 000 000	
weiß	9	9		
gold			0,1	5 %
silber			0,01	10 %

Die Farbcodierung für Induktivitäten erfolgt analog, wobei der Wert in μH angegeben wird.