# Praktikumsbericht

Ferit Ilci

03. September 2013

# Inhaltsverzeichnis

# 1 Das Praktikum

Im Folgenden wird ein vernetztes System von Ladesäulen für Elektrofahrzeuge beschrieben. Die Komponenten des Systems sind die Ladesäulen der Firma Mennekes in ihrer Vollständigkeit und ein ARM-Prozessor mit einem WLAN-Modul. Die Funktion stellt eine zentrale Verwaltung der Ladesäulen dar. Dazu läuft auf dem ARM-Prozessor ein, in C geschriebener, Server. Das System soll Testzwecken, der bereits in Ulm eingesetzten Vernetzung von Ladesäulen, dienen. Dazu gibt es ein Modell, welches aus einer 8-Kanal Relaiskarte K8056 von Velleman, einem Server und Client (geschrieben in Python) besteht.

# Teil I

# MSR-Traffic ParkGard® Magnetfeldsensor

# 2 Kurzbeschreibung

Der Magnetfeldsensor von MSR-Traffic dient zum Erfassen von Magnetfeldänderungen, die z.B. durch Fahrzeuge verursacht werden. Daher eignet es sich zum Bestimmen vom Belegungsstatus eines Parkplatzes oder zum Zählen von Fahrzeugen. Der Einsatzort bestimmt sich durch das Vorhandensein einer nahe gelegenen Netzspannung.

Die Bodenmontage bietet die höchste Detektionsraten, da der Unterboden von Fahrzeugen die stärksten Magnetfeldänderungen bewirkt. Platzierung mittig zur Fahrbahn und bis zu 10 cm unter der Fahrbahnoberfläche. Auch Wandmontagen sind möglich. Dazu bitte die Montagehinweise des Herstellers beachten.

Die Auswertung erfolgt mittels Controller, RS 485 Modbus-Signal des Magnetfeldsensors oder Relaiskontakt (angesteuert über den Open Collector Ausgang).

Nach der Montage ist das Einlernen, durch den am Gehäuse auffindbaren Knopf, welches auf den Einlernen-Eingang des Sensors auf 24 V anlegt, erforderlich. Dazu ist zu beachten, dass sich keine Gegenstände, die das Magnetfeld beeinflussen, in der Umgebung befinden.

Über den ModBus Poll und einem RS 485 Interface kann man eine Empfindlichkeitseinstellung vornehmen.

Anmerkung: Wenn in diesem Dokument die Rede vom "ParkGard® Magnetfeldsensor" ist, ist nur der Sensor gemeint. "MSR-Traffic ParkGard® Magnetfeldsensor" bezeichnet, den von MSR-Traffic angebotene, Aufbau.

# 3 Angebot

| Menge | Artikel Nr. | Beschreibung | Einzelpreis | Preis |
|---|---|---|---|---|
| 1 | PM-03-1 / D | Magnetfeldsensor zur Detektion von Fahrzeugen auf Einzelparkplätzen<br>Kunststoffrohr für Bodenverlegung IP68 35x95 mm (DxL) mit 5 m Kabel 24 V DC, RS-485<br>Ausgang Open Collector 30 V max. 0,05 A ohmsche Last | 239,00 € | 239,00 € |
| 1 | PM-REL / D | Relaismodul für Magnetfeldsensoren PM<br>Einngang: 24 V DC, Ausgang 250 V 6 A / Schalter H-0-A für Tragschienenmontage | 23,00 € | 23,00 € |
| 1 | PM-Netz / D | Netzteil für Magnetfeldsensoren PM<br>für max. 2 Sensoren, 1 Relaismodul und 1 LED-Platzleuchte | 25,00 € | 25,00 € |
| 1 | PM-Case / D | Aufputzgehäuse für Magnetfeldsensoren PM geeignet zum Einbau von Netzteil und/oder Relaismodul inkl. Einbaukosten<br>Kunststoffgehäuse für Wandmontage 215x200x105 mm (LxBxT) | 69,00 € | 69,00 € |
| Summe | | | | 356,00 € |
| Mehrwertsteuer 19 % auf 356,00 € | | | | 67,64 € |
| **Betrag** | | | | **423,64 €** |

Abbildung 3.1:

# Teil II

# Vernetzung von Ladesäulen für Elektrofahrzeugen

# 4

# Open Charge Point Protocol

# 5 Modellierung

## 5.1. Modellüberblick


Abbildung 5.1: Modellüberblick

Abbildung 5.1 zeigt das einfache Blockschaltbild des Modells. Für das Modell wird, wie bereits in Kapitel 1 erwähnt, eine 8-Kanal Relaiskarte K8056 von Velleman verwendet, welches als eine Vernetzung von 8 Ladesäuslen simulieren soll. Die Relaiskarte ist mit dessen RS-232 Schnittstelle (siehe Unterabschnitt 5.2.1) mit dem Server verbunden. Der Server hat die Aufgabe, die Befehle von Clients über Ethernet zu empfangen, interpretieren und die entsprechenden Relais ein- oder auszuschalten. Der Server und Client ist in Python geschrieben.

## 5.2. 8-Kanal Relaiskarte K8056

Der K8056 ist ein im Handel verfügbarer Bausatz. In Abbildung 5.2 ist der K8056 zusammengebaut zu sehen und Abbildung der dazugehörige Schaltplan.

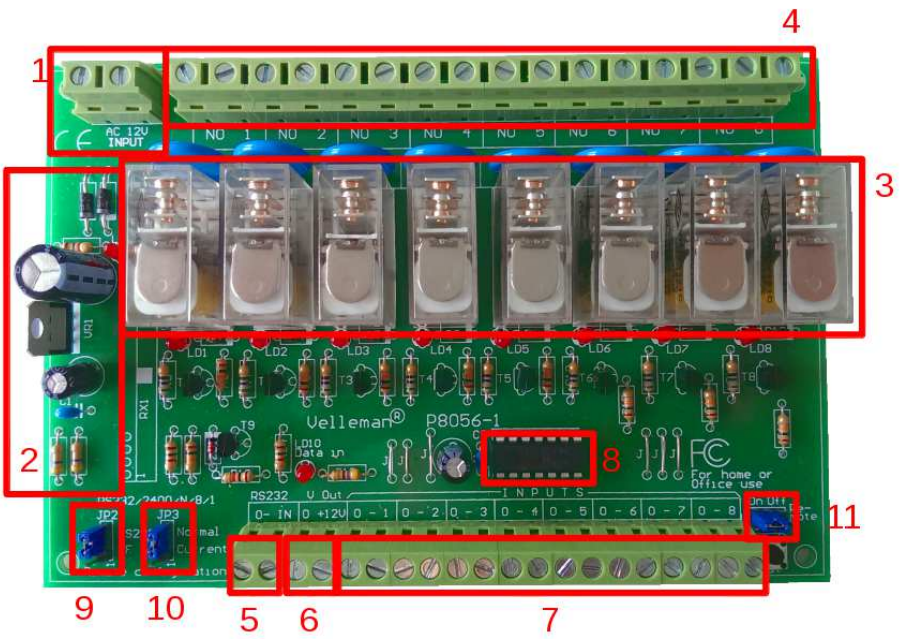


Abbildung 5.2: 8-Kanal Relaiskarte K8056 von Velleman

1. 12 V AC Spannungsversorgung
2. Gleichrichter
3. Relais 1-8
4. NO Anschlüsse der jeweiligen Relais
5. RS-232 Schnittstelle
6. 12 V DC Spannungsquelle
7. Open-Collector Ausgänge der jeweiligen Relais
8. PIC16 F630 Prozessor zur externen Ansteuerung der Relais
9. Jumper 2 zum schalten der Ansteuerungsmöglichkeit zwischen einer Fernbedienung oder die RS-232 Schnittstelle
10. Jumper 3 zum schalten zwischen hochohmiger oder niederohmiger Impedanz
11. Jumper 1 zum ein- oder ausschalten der Fernbedienungsfunktion

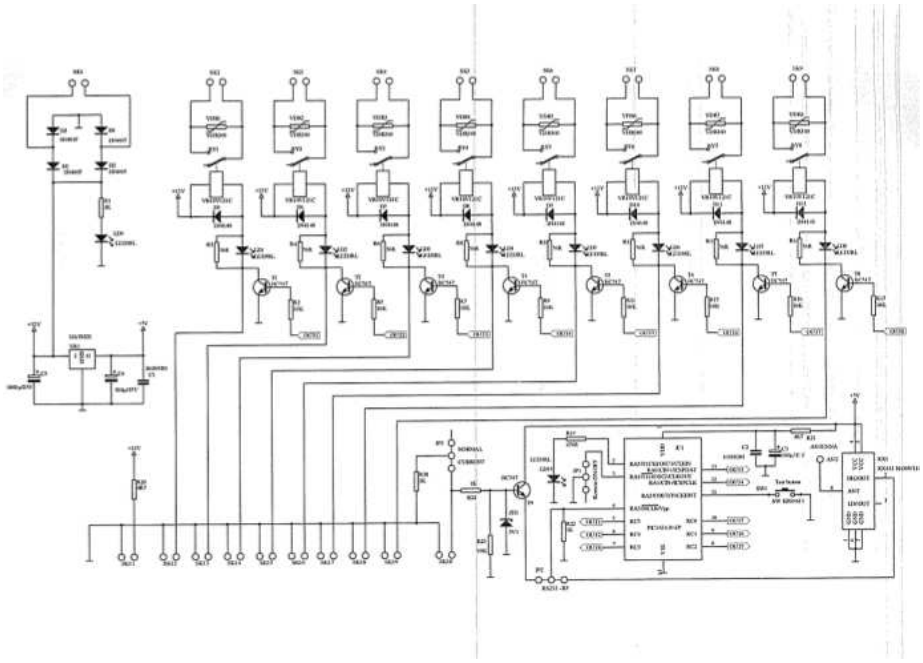

Abbildung 5.3: Schaltplan von der 8-Kanal Relaiskarte K8056

Die Relais lassen sich auf drei verschiedene Wege ansteuern. Die Ansteuerung über die Fernbedienung wird in diesem Projekt nicht gebraucht und daher hier nicht behandelt. Die zweite Möglichkeit ist die Ansteuerung über die Open-Collector Ausgänge. Um einen Relais über den Open-Collector anzusteuern, schließt man die jeweiligen Kontakte des Open-Collector Ausgangs kurz. Ist somit dieser Schaltkreis erst einmal geschlossen, so hat die dritte Ansteuerungsmöglichkeit (sowohl auch die erste, über die Fernbedienung), über den PIC-Prozessor, keine Auswirkungen mehr. Dieses verhalten ist aus dem Schaltplan in Abbildung 5.3 nachvollziehbar. Der PIC-Prozessor nimmt über die RS-232 Schnittstelle die Befehle auf und schaltet die entsprechenden Relais oder speichert die zugewiesene Adresse. Das Gerät kann die Adresse zwischen 1 und 255 besitzen. Standardmäßig ist sie auf 1. Da in diesem Projekt nur eine Karte verwendet wird, verbleibt diese Adresse. Die jeweiligen LEDs zeigen welcher Relais gerade geschalten ist. Dabei bedeutet eine leuchtende LED, dass das Relais mit den NO-Kontakten geschlossen ist.

### 5.2.1. Die RS-232 Schnittstelle

Die RS-232 Schnittstelle arbeitet mit folgenden Parametern:

| | |
|---|---|
| Baudrate: | 2400 |
| Paritätsbits: | Keine |
| Databits: | 8 |
| Stopbits: | 1 |

Abbildung 5.4 zeigt das Nachrichtenprotokoll für die Relaiskarte K8056. Dabei ist jeder Block ein Byte groß. Der Hersteller empfiehlt, die Nachrichten mindes-

tens zweimal loszuschicken. Die Schnittstelle am K8056 ist mit nur zwei Kontakten unidirektional. Eine Antwort vom K8056 ist also nicht zu erwarten.


Abbildung 5.4: Übertragungsprotokoll von der Relaiskarte K8056

**SB: Startbyte** signalisiert der Relaiskarte den Anfang einer Nachricht. Dieses Byte enthält immer 0x0D und wird anhand von dem Wert vom K8056 erkannt.

**CRDADR: Cardadress** beinhaltet die Kartenadresse. Beim Eingang einer Nachricht wird vom PIC-Prozessor erst die Adresse geprüft, bevor es die Operation ausführt. Entspricht die Adresse nicht der eigenen, so wird die Nachricht von der Karte wieder verworfen. Werte zwischen 0x01 und 0xFF stellen gültige Adressen dar.

**CMD: Command** ist der eigentliche Befehl an die Relaiskarte. Im Folgenden ist der Überblick über alle Befehle und derren Beschreibung.

**0x45 ('E'):** Nothalt für alle Karten, ungedacht derren Adresse. Der Hinweis von oben, dass die Relais, die durch den Open-Collector Ausgang aktiviert wurden, nicht durch den PIC-Prozessor ausgeschaltet werden können, ist zu beachten. D.h. diese werden durch diesen Befehl nicht ausgeschaltet.

**0x44 ('D'):** Display-Adress. Dieser Befehl zeigt über die Relais die Adresse der Karte, in binärer Darstellung, an. Über die LEDs ist der jeweilige Stellenwert zu erkennen. Dabei bedeutet eine leuchtende LED eine 1 und anders herum eine 0. LD1, also die vom ersten Relais von links aus der Abbildung 5.2, repräsentiert das MSB und LD8 das LSB.

**0x53 ('S'):** Relais einstellen. Dieser Befehl setzt das gewünschte Relais. Wie das entsprechende Relais gewählt wird, ist im Byteblock "ADR/RL" beschrieben.

**0x43 ('C'):** Relais löschen. Setzt das entsprechende Relais zurück, also in den ausgeschalteten Zustand. Wie das entsprechende Relais gewählt wird, ist im Byteblock "ADR/RL" beschrieben.

**0x54 ('T'):** Relais toggeln. Schaltet das entsprechende Relais um. Wie das entsprechende Relais gewählt wird, ist im Byteblock "ADR/RL" beschrieben.

**0x41 ('A'):** Adress Change. Wechselt die Adresse der Karte. Nach dem Ausführen diesen Befehls, zeigt die Karte die neue Adresse, wie im Befehl 0x44 ('D'), an.

**0x46 ('F'):** Dieser Befehl weist allen Karten die Adresse 1 zu. Nach dem Ausführen diesen Befehls, zeigt die Karte die neue Adresse, wie im Befehl 0x44 ('D'), an. Also schaltet es das rechteste Relais in Abbildung 5.2 ein und alle anderen aus.

**0x42 ('B'):** Byte senden. Dieser Befehl erlaubt es den Status aller Relais mit einem Befehl zu steuern. Wie die entsprechenden Relais gewählt wird, ist im Byteblock "ADR/RL" beschrieben.

**ADR/RL: Adress or Relay** entspricht der Spezifikation des Befehls aus dem Byteblock "CMD". Für die Befehle 0x53 ('S'), 0x43 ('C') und 0x54 ('T') wird hier eine Zahl in Form von ASCII erwartet. Diese Zahl entspricht der Relaisnummer, links beginnend mit 1. Also werden hier Werte zwischen 0x31 und 0x38 erlaubt. Für die beiden Befehle 0x53 ('S') und 0x43 ('C') ist auch der Wert 0x39 (also die 9 in ASCII-Format) erlaubt, welches erlaubt, die Operation auf alle Relais auszuführen. Wenn der Befehl 0x41 ('A') ausgeführt werden soll, muss an dieser Stelle die neue, gültige Adresse stehen. Bei dem Befehl 0x42 ('B') interpretiert der PIC-Prozessor dieses Byte als Bitmuster und schaltet die Relais an den Stellen der logischen Nullen aus und die der logischen Einsen ein. Dabei entspricht das linke Relais aus Abbildung 5.2 wieder dem MSB und umgekehrt. Für alle andere Befehle wird dieser Byteblock ignoriert, muss aber vorhanden sein.

**CHCKSUM: Checksum** ist die Prüfsumme der Nachricht. Sie wird aus dem Zweierkomplement, der Summe der vier vorherigen Bytes + 1 berechnet. Die realisierung ist aus dem Code im Anhang zu entnehmen.

## 5.3. Der Server

## 5.4. Der Client

# 6 Implementierungsvorbereitung

Die Implementierung erfolgt mittels einer ARM-Toolchain und einer Abatron BDI2000. Der BDI200 ist ein Debug-Interface für GDB. In diesem Dolkument ist die Implementierung unter Linux Debian und/oder Ubuntu beschrieben. Für die Vorbereitung müssen folgende Schritte angewandt werden:

1. Einrichtung der Toolchain
2. Konfiguration des BDI2000
3. TFTP-Server und Zugriff einrichten

## 6.1. Einrichten der ARM-Toolchain

Die Toolchain beinhaltet unter Linux im einfachsten Fall einen Compiler für ARM, das Paket binutils, Libraries und den GDB für ARM.

### 6.1.1. Einrichtung unter Ubuntu

Unter Ubuntu ist die ARM-Toolchain relative zügig mit installiert. Zunächst muss der entsprechende Repository von Linaro in die apt-sourcelist, mit dem folgendem Befehl, aufgenommen werden:

```
sudo add-apt-repository ppa:linaro-maintainers/toolchain
```

An dieser stelle ist es wichtig die Hinweise zu beachten. Nun kann der Compiler mit `apt` heruntergeladen und installiert werden:

```
sudo apt-get install gcc-arm-linux-gnueabi
```

Ein GCC-Compiler ist nun mit dem Namen `arm-linux-gnueabi-gcc-`*`VERSION`* ausführbar. Wenn nicht kann man mit dem Befehl

```
dpkg -L gcc-arm-linux-gnueabi
```

herausfinden, wo die Datei steckt und anschließend den Pfad in die PATH-Umgebungsvariable aufgenommen werden.

Auf die gleiche Weise, wie der Compiler, lässt sich auch binutils installiert werden:

```
sudo apt-get install binutils
```

Einzelne Libraries sind je nachdem, welche benötigt werden, ebenfalls mit apt zu installieren. Für eine anständige Entwicklungsumgebung ist das Paket newlib zu empfehlen.

### 6.1.2. Einrichtung unter Debian

Für die Einrichtung unter Debian macht es durch aus Sinn, als root zu arbeiten, da alle Befehle mit root-Rechten auszuführen sind. Die Pakete sind auf dem Emdebian-Repository erhältlich. Vorraussetzung ist auch hier, dass binutils installiert ist. Als erstes wird der Keyring installiert:

```
apt-get install emdebian-archive-keyring
```

Anschließend den folgenden Eintrag in die Datei /etc/apt/sources.list übernehmen:

```
deb http://www.emdebian.org/debian wheezy main
```

Nun kann man mit apt den Compiler herunterladen und installieren:

```
apt-get install gcc-4.7-arm-linux-gnueabi
```

Das Paket xapt stellt eine gute Hilfe dar, Bibliotheken für den ARM-Prozessor zu installieren. Auch dieses Pket lässt sich über apt auf das System draufspielen:

```
apt-get install xapt
```

Die richtigten Bibliotheken für ARM-Prozessoren kann man nun folgendermaßen herunterladen:

```
xapt -a armel -m library-name
```

Hier ist eine Liste einiger, allzweck gebräuchlicher, Bibliotheken:

- libc6
- libc6-dev
- libc-bin
- libc-dev-bin
- glibc
- linux-kernel-headers
- ...

**GDB-Cross Package installieren**

Der GNU-Debugger ist über Emdebain genauso einfach zu installieren wie der Compiler. Dazu einfach mit dem folgenden Befehl (ebenfalls als root) die Pakete draufspielen:

```
apt-get install gdb-arm-linux-gnueabi
```

Wenn der Fehler

```
E:Sub-process /usr/bin/dpkg returned an error code(1)
```

auftaucht, lässt er sich mit dem Befehl

```
dpkg -i --force-overwrite /var/cache/apt
.../archives/gdb-arm-linux-gnueabi
```

umgehen.

### 6.1.3. Test der Toolchain

Das kleine Programm `file` zeigt Dateiinformationen an. D.h. wenn ein Programm, z.B. helloworld, mit dem GCC-Compiler für ARM erzeugt wurde, gibt

```
file helloworld
```

die Information über die Architektur an. Hier sollte ARM heraus zulesen sein.

Zum emulieren eines ARM-Prozessors bietet sich `qemu` an, welches sich mit dem folgenden Befehl installieren lässt.

```
apt-get install qemu
```

Für Ubuntu ist zusätzlich noch qemu-user erforderlich:

```
apt-get install qemu-user
```

```
qemu-arm helloworld
```

dürfte nun das Programm abspielen können.

## 6.2. Konfiguration des BDI2000

Die erste konfiguration von dem BDI2000 erfolgt über die RS-232 Schnittstelle. Dabei ist zu beachten, dass es mit einem Null-Modem Kabel an dem Rechner angeschlossen wird. Zur konfiguration gibt es die Software bdisetup und die Datei rm9200dk.cfg. An der entsprechenden Stelle der Datei (markiert mit "[host]") muss die IP-Adresse eines TFTP-Servers stehen, auf dem die cfg-Datei im root-Verzeichnis, abgelegt ist. Der BDI2000 sucht nämlich nach jedem Start des Geräts auf dem Server nach der Datei und aktualisiert diese.

Die notwenigen Dateien für die Konfiguration sind in dem .zip-Archiv zu finden. Nachdem dieses die .zip-Datei entpackt ist, ist es sinnvoll, in das entpackte Verzeichnis zu wechseln und von dort aus die Konfiguration vorzunehmen. Der Aufruf von "make", in dem Verzeichnis, erzeugt die ausführbare Datei bdisetup. bdisetup gibt eine kurze Übersicht über sich, wenn man es ohne Parameter ausführt.

Als erster Schritt ist die Frimware/Logic auf den BDI2000 drauf zuspielen. Dazu genügt es den folgenden Befehl auszuführen, wenn das aktuelle Verzeichnis, das ist, welches vorhin entpackt wurde. Ansonsten muss das Verzeichnis mit dem Parameter -d mit angegeben werden:

```
./bdisetup -u -p/dev/ttyUSB0 -b57 -aGDB -tARM
```

Nach den angegebenen Parametern findet das Programm bdisetup die richtige Dateien. Hier ist zu beachten, dass der Port, der mit dem Schlüssel -p mit gegeben wird, durch den Port ersetzt werden muss, mit dem auch der BDI2000 am Rechner angeschlossen ist.

Nun folgt die übermittelung der Netzwerkdaten für den BDI2000. Diese sind die IP-Adresse der BDI2000, die Subnetzmaske, der default Gateway, IP-Adresse des Hosts und der vollständige Pfad vom TFTP-Server zur .cfg-Datei. Wenn hier, wie oben beschrieben, die Datei im root-Verzeichnis liegt, genügt es hier nur den Dateinamen anzugeben. Der vollständige Befehl sieht dann ungefähr so aus:

```
./bdisetup -c -p/dev/ttyUSB0 -b57 -i10.0.0.101 -h10.0.0.100
...-m255.255.255.0 -g10.0.0.1 -frm9200dk.cfg
```

Die Werte der Parameter sind, entsprechend den gewünschten Netzwerkeigenschaften, zu ersetzten. Ob alle Werte richtig übernommen sind, kann man mit der

Option -v für bdisetup, überprüfen. der Schlüssel -s schließt die Konfiguration ab und die rote Mode-LED auf dem BDI2000 sollte aufhören zu blinken.

```
./bdisetup -v -p/dev/ttyUSB0 -b57 -s
```

## 6.3. TFTP-Server einrichten

Der TFTP-Server ist unter Linux in zwei Schritten eingerichtet. Der erste ist die Installation des Pakets tftpd als root:

```
apt-get install tftpd
```

Der Service dabei ist unter Ubuntu xinetd und unter Debian openbsd-inetd, welche unter /etc/init.d/ zu finden sind. Auf diese beiden Services sind die Operationen start, restart und stop anzuwenden. Also z.B., um den Service zu stoppen:

```
/etc/init.d/xinetd stop
```

Für gewöhnlich richtet sich der Server bei der Installation so ein, dass er beim Start des Betriebssystems, mit startet, so dass man da nicht manuell Hand anlegen muss. Die Konfigurationsdateien sind unter /etc/ zu finden und heißen wie das Paket und zusätzlich das Postfix .conf. Beim inetd.conf reicht es folgende Zeile einzutragen:

```
ftp dgram udp wait nobody /usr/sbin/tcpd
.../usr/sbin/in.tftpd /srv/tftp
```

Dabei ist das letzte Argument das gewünschte Verzeichnis, welches vom TFTP-Server freigegeben wird. Da TFTP-Server für häufig Bootzwecke angewendet werden, ist es hier Standardmäßig /tftpboot/. Dieser kann nach beliebn geändert werden.
Für den xinetd hat die Datei folgendes Aussehen:

```
service tftp
{
   socket_type = stream
   protocol = udp
   wait = yes
   user = nobody
   server = /usr/sbin/in.tftpd
   server_args = /srv/tftp
   disable = yes
}
```

## 6.4. Zugriff über telnet

Die eigentliche Bedienung der BDI2000 läuft über das telnet-Protokoll. Dazu muss es in einem Netzwerk angeschlossen sein. Mit einem Cross-Over Kabel lässt sich das kleinste Netzwerk schon realisieren. Anschließend reicht die Ausführung von `telnet` (mit der in Abschnitt 6.2 zugewiesenen IP-Adresse der BDI2000) auf der Kommandozeile aus, um Zugriff auf den BDI2000 zu erhalten:

```
telnet 10.0.0.101
```

Von hier aus lässt sich z.B. mit dem Befehl `load` die .bin-Datei aus dem TFTP-Server laden. Dazu muss selbstverständlich solch eine Datei auf dem Server existieren. Der Befehl `run` startet das Programm auf dem Board, sofern der BDI2000 mit dem JTAG Interface an dem Board angeschlossen ist.