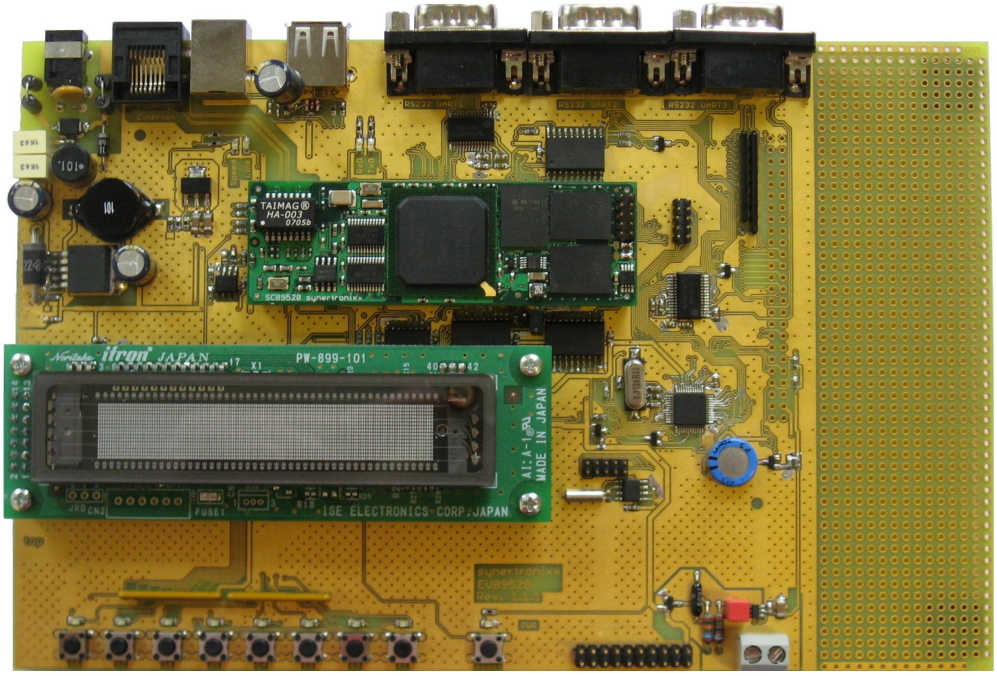
EVB9520
Getting Started



SCB9520 Getting Started

Inhaltsverzeichnis

1 Getting Started

1.1 Einrichtung des Evaluation Board

Für die Inbetriebnahme des Evaluation Boards schließen Sie es an eine Spannungsversorgung (10V - 30V) an. Für die Kommunikation und Steuerung des Boards empfiehlt sich die Verbindung mit einem Nullmodemkabel. Eine Verbindung über ein Netzwerk mit einem Netzwerkkabel (CAT 5) ist ebenfalls möglich. Die serielle Verbindung erfolgt über einen COM-Port am PC und der Debug-Schnittstelle auf dem Board. Der Zugriff erfolgt über ein Terminal-Programm.

Soll das Programm *DeviLANControl* verwendet werden, ist ein PC, ausgestattet mit MS Windows, bereitzustellen.

Die einzelnen Schritte im Detail:

Für die Inbetriebnahme sind folgende Schritte zu erledigen:

1. Das Evaluation Board mit dem Nullmodemkabel an den PC anschließen.
2. Ein Terminal-Programm auf dem PC starten. Die Einstellungen des Terminals sind wie erfolgt: 115,2kBaud, 8 Datenbits, 1 Stopbit und keine Parität.
3. Das EVB9520 (ohne SCB9520) ist an die Spannungsversorgung anzuschließen. Die PWR-Diode am Reset-Taster muss leuchten. Der Zustand der Acht LEDs ist ohne eingesteckten Socket Computer nicht definiert.
4. Spannungsversorgung abschalten.
5. Wenn die oberen Punkte alle erfolgreich abgeschlossen wurden, kann fortgefahren werden. Den Socket Computer 9520 auf den DIL64-Sockel einsetzen. Der Übertrager muss auf der Seite des RJ45-Anschlusses sein.
6. Wird nun die Spannungsversorgung eingeschaltet, blinken die RxD und TxD LEDs in nicht gleichmäßigen Abständen auf. Nun müssen die 8 LEDs ununterbrochen leuchten.

Durch das rc-Skript werden die Anwendungen *udp_config* und *evb9520* automatisch und im Hintergrund gestartet.

Auf die Einstellungen der seriellen Schnittstelle und der Terminal-Programme unter Linux und MS Windows wird in den folgenden Kapiteln eingegangen.

1.2 Zugriff über die serielle Schnittstelle (Linux)

Über die serielle Schnittstelle kann das Evaluation Board ebenfalls gesteuert werden. Hierfür wird ein Terminal-Programm benötigt. Nach dem Hochfahren des SCB9520 erscheint die Aufforderung zur Eingabe von Benutzername und Passwort (siehe Bild 10).

Die standardmäßig eingestellten Login-Daten lauten:

username : root

password : pass

Zu beachten ist, dass unter Linux die Groß- und Kleinschreibung unterschieden wird und bei der Eingabe des Passwortes keinerlei Zeichen angezeigt werden. Unter Linux gibt es mehrere Terminalprogramme die häufig ihre Verwendung finden. Speziell für Einsteiger als geeignet gelten *minicom* und *kermit*.

Abbildung 1: Der Login-Vorgang

In Screenshot 2 ist die Ausgabe nach der erfolgreichen Anmeldung zu sehen. Mit den Befehlen *ls* (list segments), *pwd* (print working directory) und *cd* (change directory) ist es möglich, sich auf dem System umzusehen.

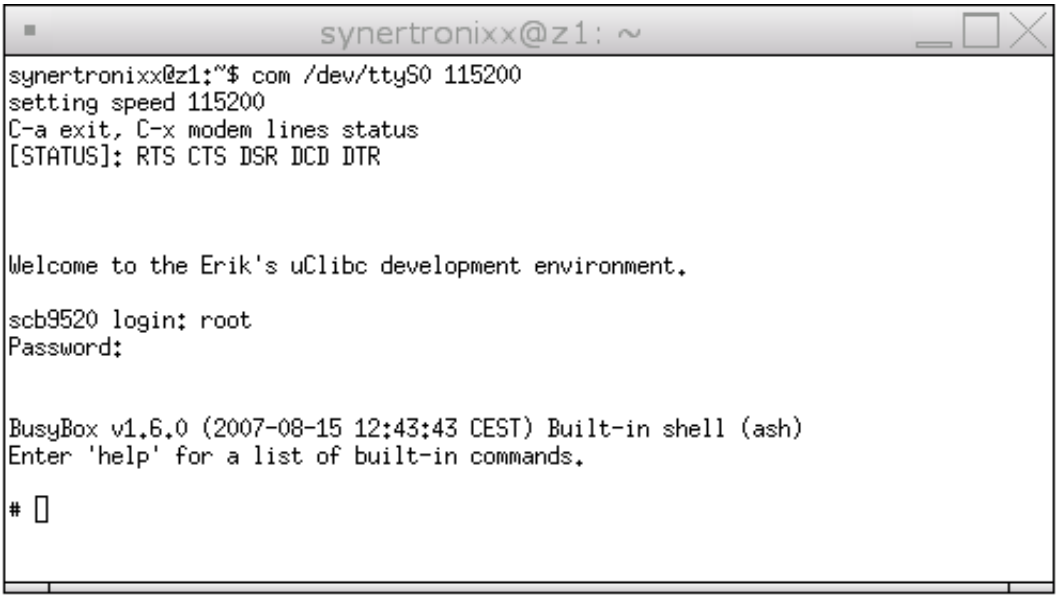


Abbildung 2: Nach dem erfolgreichen Einloggen.

Nun ist das Netzwerk einzurichten. In Abbildung 3 ist die Datei /etc/network/interfaces zu sehen. Sie ist mit dem Editor *nano* geöffnet worden, außer *nano* steht noch der Editor *vi* zur Verfügung.

Die Einträge können variieren, speziell die Einträge modulname, serial, tcpport und udpport werden erst nach der erstmaligen Verwendung des Programms *udp_config* erzeugt. Auch wenn sie nicht vorhanden sind, ist das System trotzdem bereit für den

Einsatz in einem Netzwerk. Die Voreinstellung der IP-Adresse des SCB9520 ist 192.168.1.199. Die Adresse hängt von den Gegebenheiten des lokalen Netzwerks ab und ob eine Netzwerkverbindung benötigt wird.

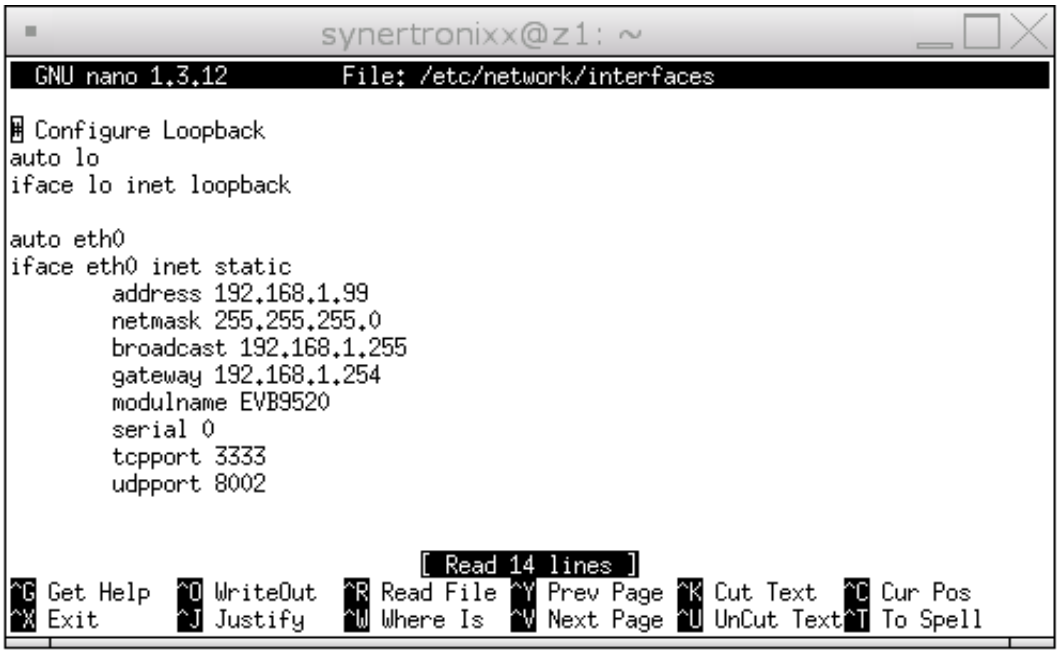


Abbildung 3: Das Ändern der IP mit dem Editor nano.

Um die Änderung der Netzwerkeinstellungen abzuschließen, sind folgenden Befehle zu verwenden:

```
ifdown eth0
ifup eth0
```

Der erste Befehl beendet den Netzwerkdienst und der zweite startet ihn mit den neuen Einstellungen. Die Ausgabe der Befehle ist in Bild 4 zu sehen.

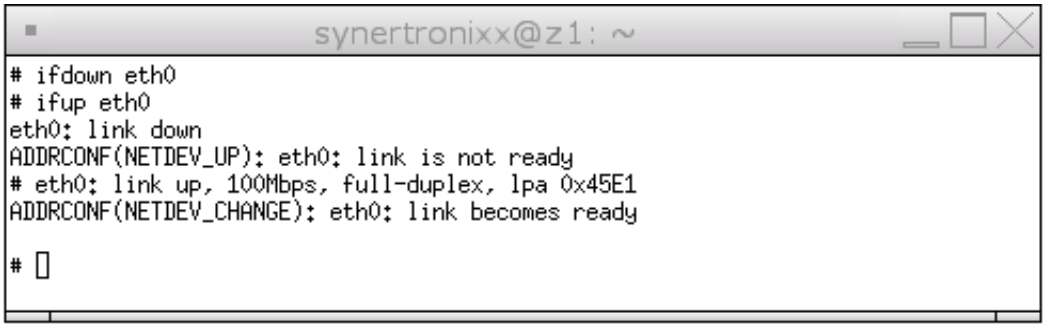


Abbildung 4: Das Initialisieren der neuen IP-Adresse.

Zum Testen der Einstellungen ist der Befehl *ping* geeignet. Ein möglicher Aufruf ist in Abbildung 5 zu sehen.

Eine gute Testabfolge ist, erst sich selbst und danach einen anderen Rechner im gleichen Netzwerk mit einem Ping anzusprechen. Der Befehl *ping* bekommt die Parameter *c* und *4*, damit er nur vier Ping-Versuche versendet, ansonsten würde *ping* ununterbrochen senden. Als letzten Parameter bekommt *ping* die IP-Adresse, die es anpingen soll.

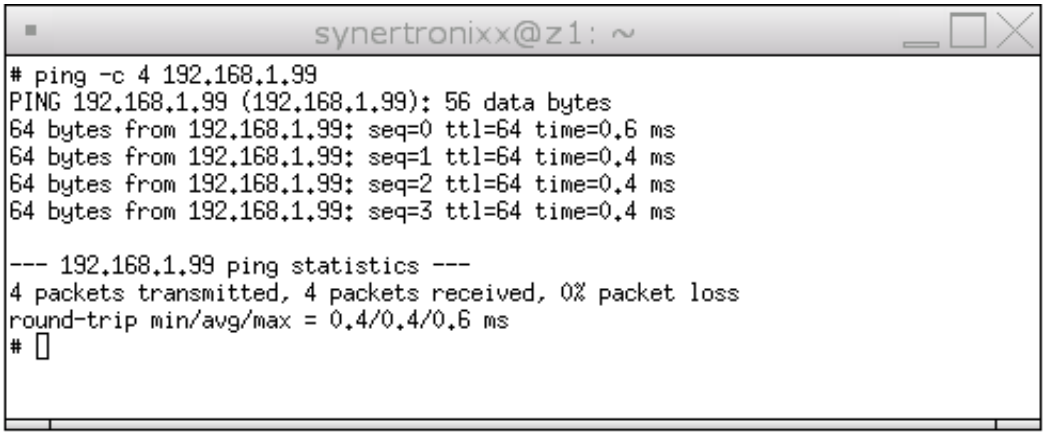


Abbildung 5: Überprüfung der Netzwerk-Einstellungen mit ping.

1.3 Zugriff über die serielle Schnittstelle (MS Windows)

Die Kommunikation zwischen Evaluation Board und PC kann auch mit dem Betriebssystem MS Windows realisiert werden.

Als erstes ist ein Terminal zu starten, hier findet das Hyperterminal seine Verwendung, andere Terminal-Programme können ebenfalls verwendet werden. Nun ist das gewünschte Icon zu wählen, der Verbindung einen Namen zu geben und mit dem OK-Button die Eingaben zu bestätigen. In Bild 6 ist ein Beispiel zu sehen.

Abbildung 6: Anlegen einer neuen Verbindung.

Das nächste Fenster ist in Screenshot 7 zu sehen. Hier ist der COM-Port zu wählen, an dem das Board angeschlossen ist. Diese Eingaben sind ebenfalls mit OK zu bestätigen.

Abbildung 7: Die Auswahl des COM-Ports.

Im Bild 8 ist der Konfigurationsdialog für die serielle Schnittstelle zu sehen. Die Einstellungen sind mit den folgenden Werten zu belegen (siehe Tabelle 1).

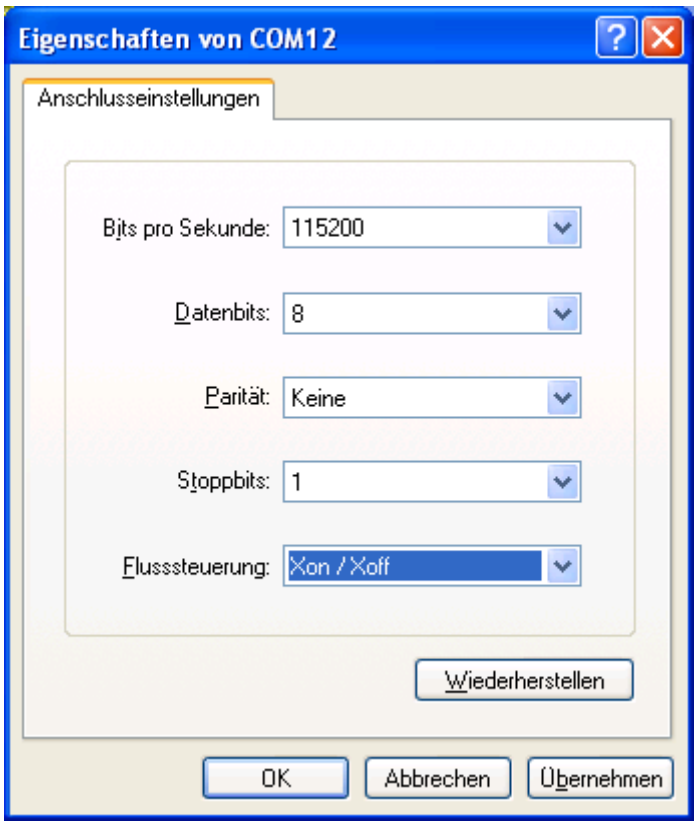


Abbildung 8: Die Konfiguration der seriellen Schnittstelle.

Tabelle 1: Die Konfiguration des COM-Ports.

Bits pro Sekunde	115200
Datenbits	8
Parität	keine
Stoppbits	1
Flusssteuerung	Xon/Xoff

In Abbildung 9 ist der Screenshot des Hyperterminals nach dem Bootvorgang zu sehen. Die Login-Daten sind wie in Kapitel 1.2 beschrieben:

```
username : root
password : pass
```

Die weitere Konfiguration der Netzwerkadresse erfolgt, wie bereits in Kapitel 1.2 auf Seite 5 beschrieben.

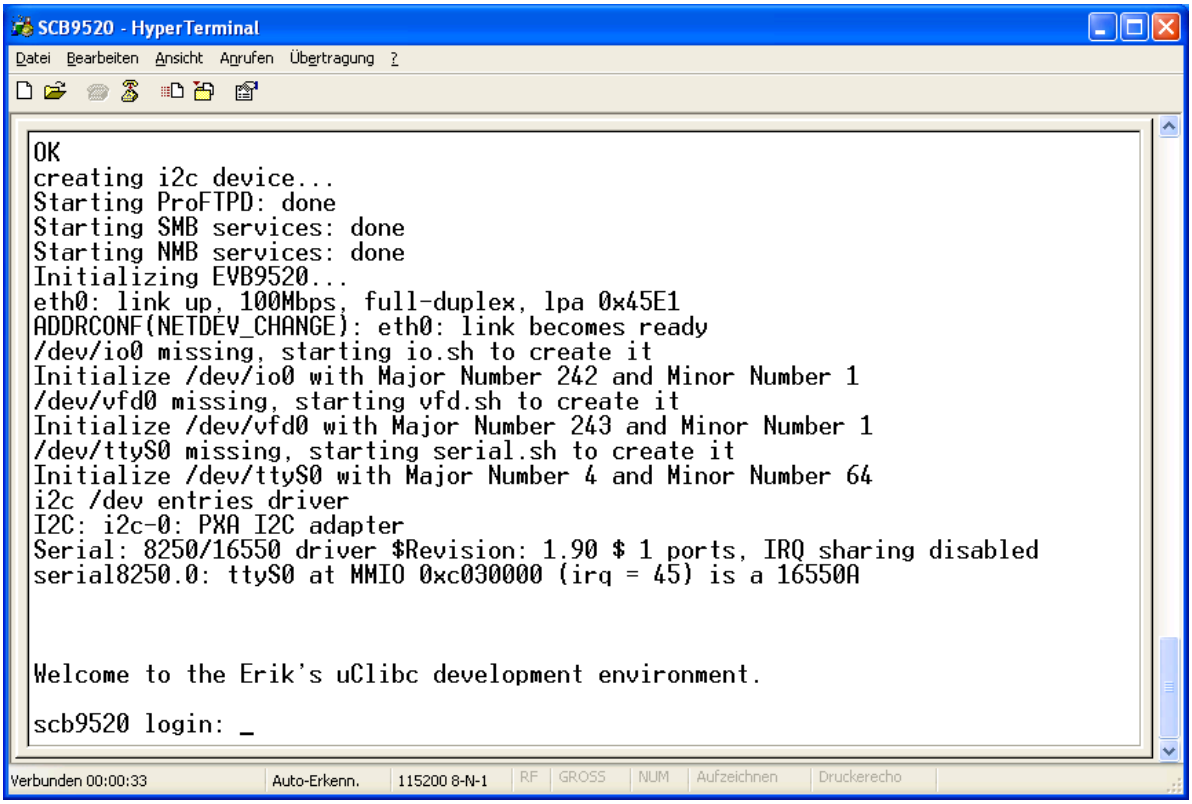


Abbildung 9: Die Ausgabe auf dem Hyperterminal.

1.4 Kompilieren für das EVB9520

1.4.1 Kompilieren des Beispielprogramm Hello World

Das Kompilieren erfolgt auf einem Linux-PC mit Hilfe des Cross-Compilers von der mitgelieferten CD.

Zum Kompilieren muss zuerst der Cross-Compiler von der mitgelieferten CD auf einem Linux-PC entpackt und eingerichtet werden. Unten steht die Befehlsfolge zum Einhängen (Mounten) der CD und zum Entpacken des Compilers. Einige Distributionen beherrschen das Automounten und hängen die CDs und USB-Sticks selbstständig ein und aus. Ist dies der Fall, wird die erste und letzte Zeile weggelassen.

Wichtig, der Compiler wird auf dem PC benutzt und nicht auf dem Evaluation Board!

```
mount /mnt/cdrom
cd /
tar xjf /mnt/cdrom/home/software/arm-linux-gcc-4.1.2.tar.bz2
umount -l /mnt/cdrom
```

Der tar-Befehl entpackt das Archiv in den Ordner /usr/local/arm, existiert dieser Pfad nicht, so wird er automatisch erstellt. Um den Compiler nutzen zu können, muss die PATH-Variable mit folgendem Befehl gesetzt werden.

```
export PATH=$PATH:/usr/local/arm/arm-linux-gcc-4.1.2/bin
```

Die Variable PATH enthält alle Pfade zu Ordnern, in denen das Betriebssystem nach ausführbaren Dateien sucht. Diese Dateien können Shell-Skripte oder kompilierte Binärdateien sein. Um diesen Eintrag nicht nach jedem Bootvorgang wiederholen zu müssen, kann der export-Befehl in die Datei /etc/profile eingetragen werden.

Als Beispiel erfolgt hier die Kompilierung eines "Hello World"-Programms. Der Quelltext ist in Listing 1 zu sehen.

```
#include <stdio.h>

int main (void)
{
        printf("Hello World!\n");

        return 0;
}
```

Listing 1: Das Programm "Hello World!"

Zum Kompilieren wird der Compiler wie folgt aufgerufen.

```
arm-linux-gcc -o hello -Wall hello.c
```

Nach dem erfolgreichen Kompilieren des Quelltextes, kann das Programm mit ftp oder scp auf das Evaluation Board übertragen werden. Die Datei ist nach /usr/local/bin zu kopieren und kann im Anschluss daran, entweder über ein Terminal, SSH oder telnet auf dem Board ausgeführt werden.

Die Datei muss mit dem Befehl *chmod* ausführbar gemacht werden und kann mit dem Aufruf *hello* ausgeführt werden. In Bild 1 ist der Vorgang zu sehen.

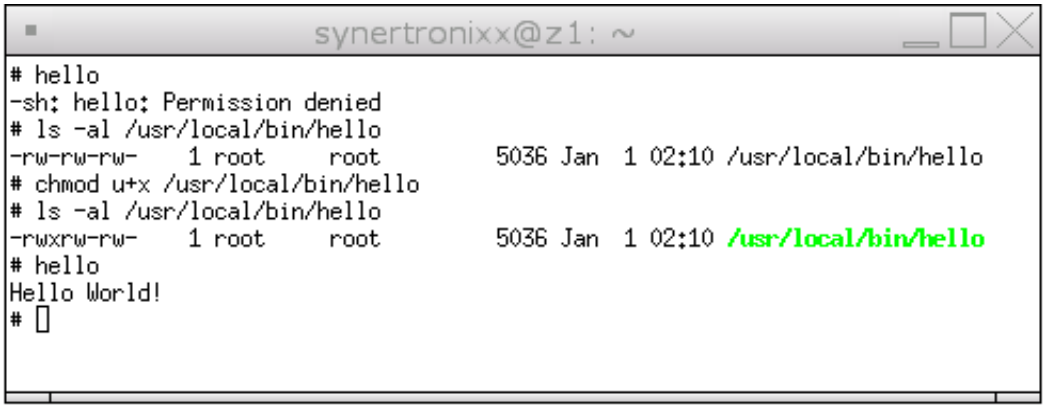


Abbildung 10: Der chmod-Befehl und das Ausführen des "Hello World"-Programms.

1.4.2 Kompilieren des Beispielprogramm evb9520

Die beigelegte CD enthält ein Beispielprogramm, welches mit Quelltext ausgeliefert wird. Dieses Programm kann als Basis für eigene Programme verwendet werden. Zum Kompilieren des Quellcodes wird das Makefile auf der Konsole mit folgendem Befehl ausgeführt:

```
make -f evb9520.mak rebuild
```

Auf der mitgelieferten CD befindet sich eine Projektdatei evb9520.vpj, diese Datei ist mit dem kostenpflichtigen und nicht mitgelieferten Editor slickedit erzeugt worden. Ist dieser Editor vorhanden, kann auf diese Projektdatei zurückgegriffen werden.

Das Kopieren der Datei ist in Kapitel 1.4.1 bereits beschrieben worden. Ausgeführt wird das Programm mit folgendem Aufruf:

```
evb9520
```

2 Die Bedienung der Applikation

2.1 Die Steuerung über die Taster und die Debug-Schnittstelle ttySA0

Die Anwendung evb9520 wird dem Befehl

```
evb9520
```

gestartet. Die Applikation befindet sich im Ordner /usr/local/bin.

Nach dem Starten der Applikation leuchten alle LEDs. Ist ein Taster gedrückt, erlischt die dazugehörige LED und leuchtet erst wieder nachdem der Taster los gelassen worden ist. Die ersten beiden Taster dienen der Menüführung. Der Taster S8 inkrementiert die Variable menu und der Taster S7 dekrementiert sie.

Folgende Menüpunkte stehen zur Verfügung:

7. Anzeige der aktuellen Zeit
8. Anzeige der Spannung am ADS1110
9. Ausgabe der Zeichen auf dem Display, die an das Device ttySA2 gesendet werden
10. Ausgabe der Zeichen auf dem Display, die an das Device ttyS0 gesendet werden

Eine Ausgabe auf dem Bildschirm erfolgt nicht. *printf*-Ausgaben sind sehr aufwendig und würden das Laufzeit-Verhalten zu stark beeinflussen. Eine Steuerung über *ssh* ist ebenfalls möglich. Es erfolgt zuerst die Anmeldung an das Evaluation Board und die restliche Steuerung ist gleich der Steuerung über die serielle Schnittstelle.

Der Befehl für den ssh-Zugriff lautet:

```
ssh root@192.168.1.199
```

2.2 Die Ansteuerung mit DeviLANControl über eine TCP-Verbindung

Als erstes ist das Evaluation Board mit Spannung zu versorgen. Die Spannung muss zwischen 10 und 30V liegen. Danach ist das Board mit dem PC über ein Nullmodemkabel an der seriellen Schnittstelle, welche sich neben den USB-Schnittstellen befindet, anzuschließen. Um sich auf dem Board einzuloggen ist nun eine serielles Terminal zu starten. Ein Login-Prompt wie in Abbildung 11 ist jetzt zu sehen.

Abbildung 11: Der Login-Prompt

Einige Funktionen des Evaluation Board können von einem entfernten Rechner aus benutzt werden. Hierfür wird das Programm *udp_config* benötigt. Das Programm ist mit folgendem Aufruf auf der Konsole zu starten:

```
udp_config
```

Die Ausgabe ist in Screenshot 12 zu sehen. Das Programm udp_config macht keine Ausgaben.

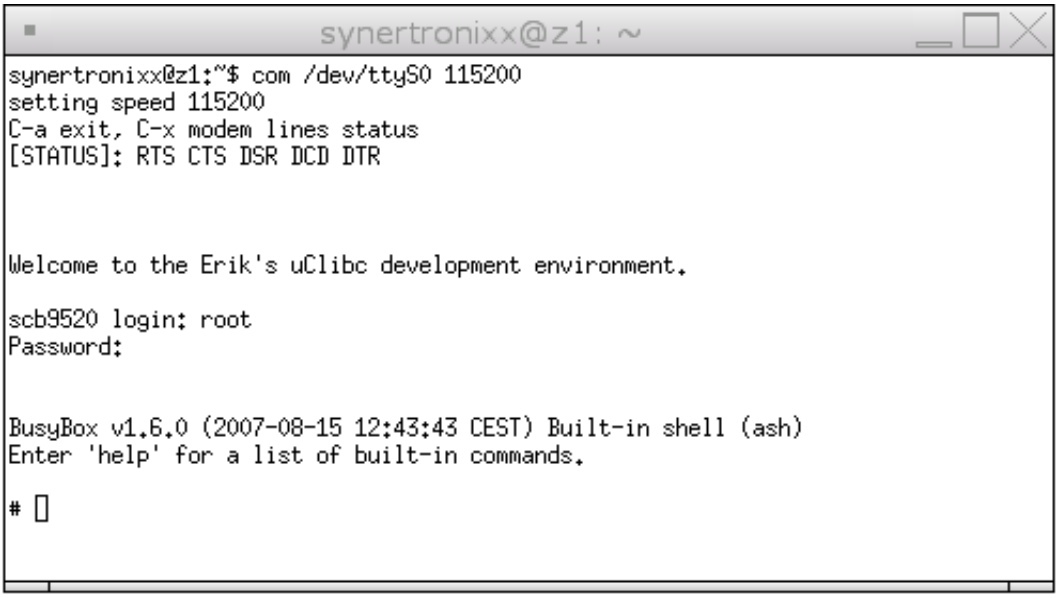


Abbildung 12: Das Programm udp_config

Nun ist auf dem Windows-PC das Programm DeviLANControl zu starten, siehe Screenshot in Bild 13. Zu sehen ist hier der Tab „Verbindungsmanager“. In diesem Reiter sind die aktiven synertronixx-Module im Netzwerk zu sehen, die über UDP eine Verbindung erzeugen wollen. Zum Suchen des Boards muss die Einstellung „UDP-Meldung senden an“ auf „IP2002/SCB9328 (Port: 80)“ umgestellt werden. Ist das Board nicht zu sehen, einfach erneut auf „suchen“ klicken.

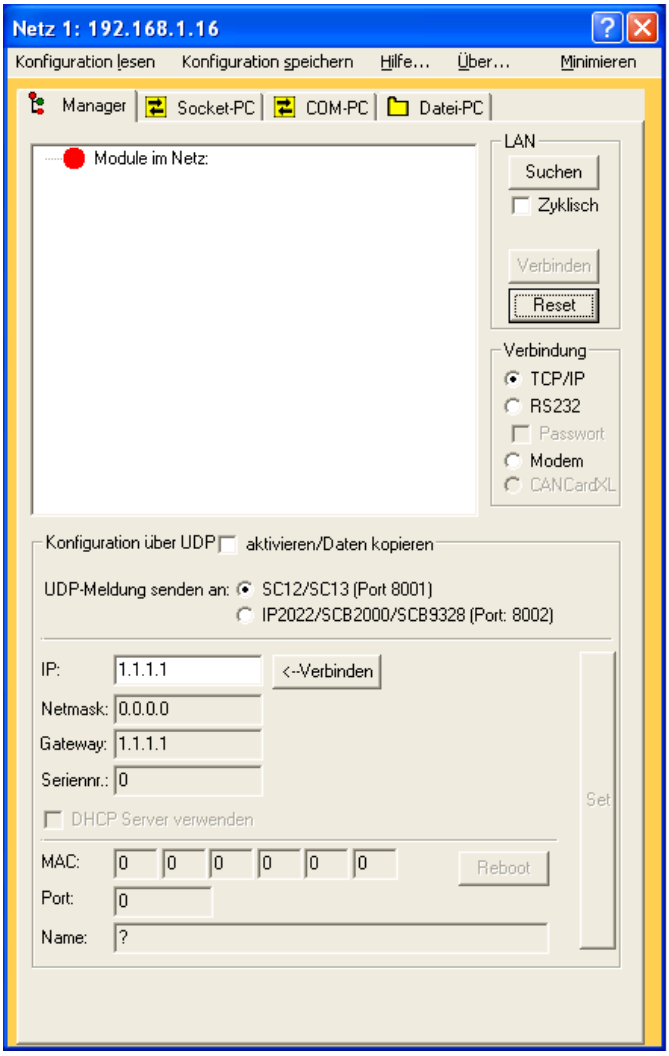


Abbildung 13: Das Programm DeviLANControl.

Die Ausgaben des Aufrufs „evb9520“ sind in Abbildung 14. Die Anwendung gibt auf der Konsole lediglich beim Start und bei eventuell auftretenden Fehlern eine Ausgabe auf der Konsole aus. Ausgaben gibt es soweit keine, da printf eine sehr rechenintensive Funktion ist und somit das Laufzeit-Verhalten stark beeinträchtigt.

In Bild 15 ist das DeviLANControl zu sehen. Es ist zu sehen, dass ein Board gefunden worden ist. Die Verbindung wird mit dem Button "Verbinden" aufgenommen. Die Einstellung unter "Verbindung" muss auf TCP/IP stehen.

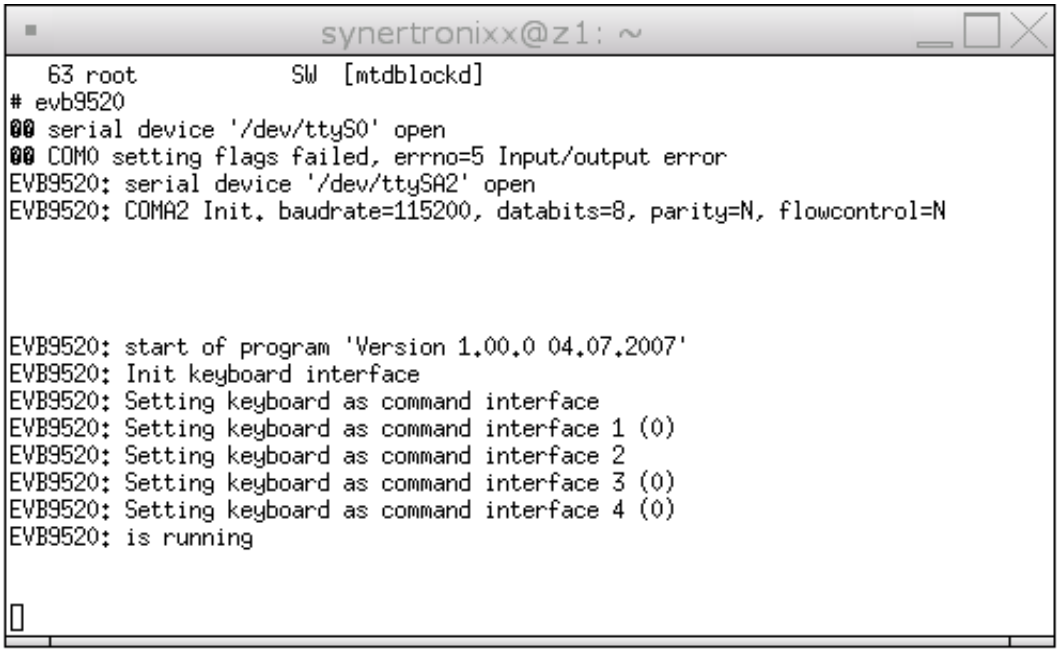


Abbildung 14: evb9520-Aufruf auf der Konsole.

Jeder gefundene Kommunikationspartner kann aufgeklappt werden. Hier sind dann zusätzliche Informationen, wie die IP-Adresse, der verwendete Port, die Netzmaske und die MAC-Adresse zu sehen.

Eine kurze Erklärung zu udp_config:

Um laufende SCB9520 bzw. EVB9520 in einem lokalen Netzwerk (LAN) zu identifizieren bzw. zu konfigurieren, steht das Programm udp_config zur Verfügung. Der UDP-Datenkanal (Port 8002) dient zur Identifikation und Konfiguration des Moduls. Zur Kommunikation ist ein Kommandointerpreter implementiert. Er reagiert auf UDP-Broadcast-Meldungen und antwortet mit den aktuellen TCP/IP-Kommunikationseinstellungen. Über den Kanal können zusätzlich die elementaren Netzwerkeinstellungen wie IP-Adresse, Netzmaske oder Gateway vorgenommen werden. Das Programm modifiziert dazu die Datei /etc/network/interfaces, die die aktuellen Netzwerkeinstellungen enthält. Die alte Datei interfaces wird umbenannt in interfaces.pre_udpsocket.

Für die Konfiguration kann das PC-Programm DeviLANControl verwendet werden. Für eine Änderung der Netzwerkeinstellungen ist wie folgt vorzugehen:

- Das gewünschte Modul im Menübaum markieren.
- Unter Konfiguration über UDP aktivieren (die Moduldaten werden damit auch in die Eingabefelder kopiert).
- Auswählen von UPD-Meldung senden an (Port 8002).
- Eintragen bzw. Ändern der gewünschten Einstellungen in den Eingabefeldern.
- Ändern der Einstellungen durch Drücken des Buttons "Set".
- Die Einstellungen werden erst nach einem Neustart des Moduls wirksam. Zur

Kontrolle sollten die Netzwerkeinstellungen überprüft werden.

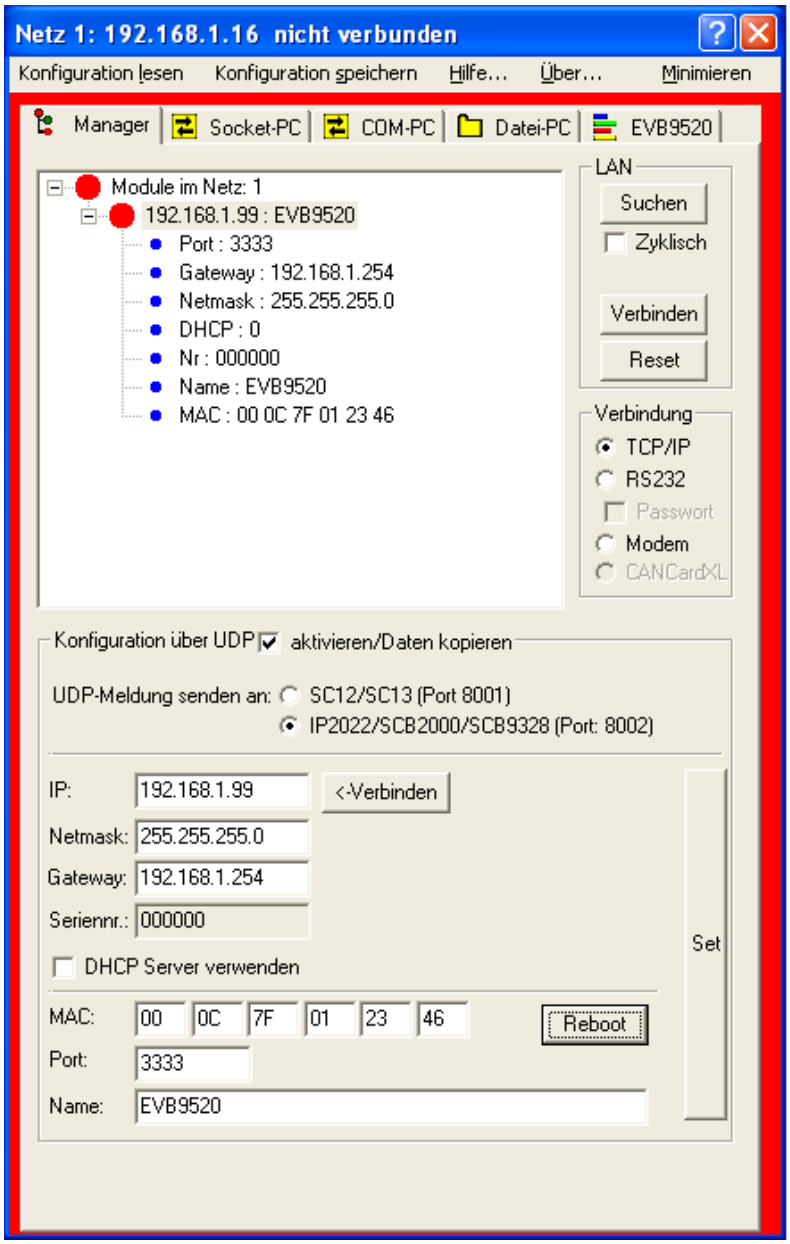


Abbildung 15: DeviLANControl nach dem Informationsaustausch mit udp_config.

Nachdem die Verbindung erfolgreich mit dem Evaluation Board aufgenommen worden ist, wird automatisch die Zeichenkette "modul_typ 9520" an das Board gesendet. Empfängt das DeviLANControl diesen String, erzeugt es einen neuen Reiter mit dem Namen "EVB9520" (siehe Bild 16).

Der neu erzeugte Tab enthält einige Steuerelemente. Mit den Radio-Buttons unter "Port A" kann der Zustand der LEDs ausgelesen und gesetzt werden. Unter "Port B" ist die aktuelle Taster-Stellung zu sehen. Unter "ADC" ist der eingelesene Analogwert in mV zu sehen.

Zusätzlich zu der Spannung erfolgt auch die Anzeige der Baudrate und der Verstärkung. Mit den beiden Listboxen kann sowohl die Baudrate, als auch die Verstärkung verändert werden. Der Abschnitt unterhalb von ADC zeigt die Zeit an, sie kann ebenfalls eingestellt werden. Die Zeichenkette zum Verändern der Zeit muss den gleichen Aufbau haben, wie die, die im DeviLANControl angezeigt wird.

Im Bereich "Mode" wird das Menü im VFD eingestellt, der aktuelle Mode für die Zeit hält auch die im DeviLANControl angezeigte Zeit aktuell. Ist der Mode auf ADC, so wird die am ADS1110 anliegende Spannung im DeviLANControl regelmäßig aktualisiert.

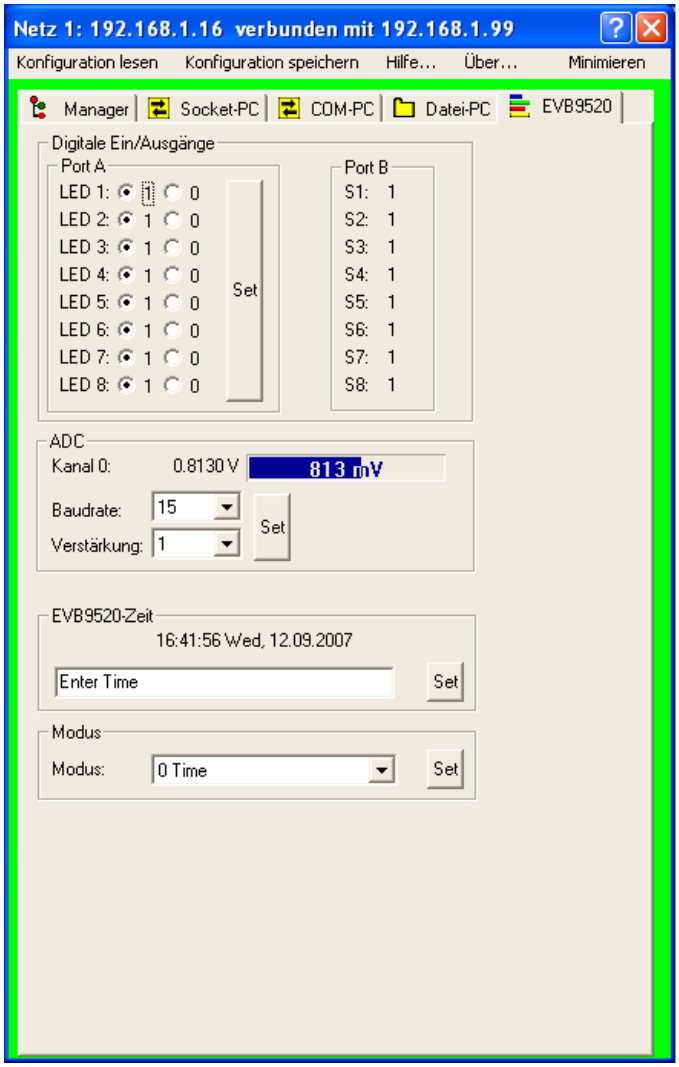


Abbildung 16: Das neue Register "EVB9520".

Wird im Reiter "Socket-PC" der Befehl data_renew, wie in Abbildung 17 zu sehen ist, verwandt, so aktualisieren sich die angezeigte Zeit, der Zustand der LEDs, die Tasterstellung und der Analogwert. Die Werte sind dann im Tab EVB9520 abzulesen.

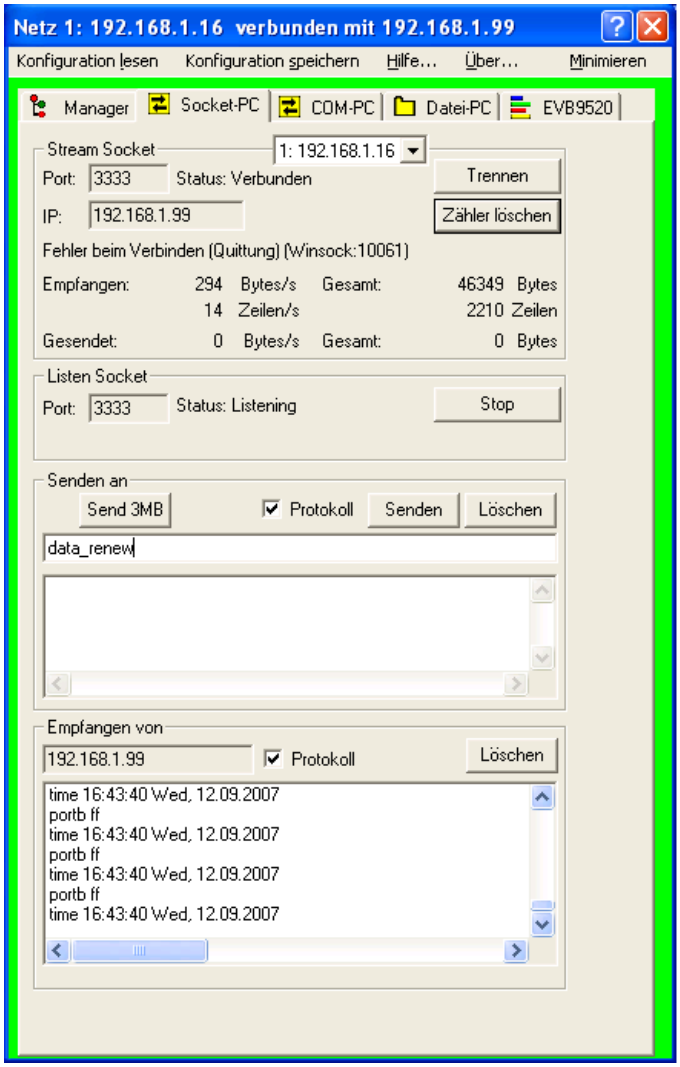


Abbildung 17: Der Befehl data_renew.

3 Die Hardware

Das Evaluation Board EVB9520 dient zur Veranschaulichung der Fähigkeiten des SCB9520.

Das Board zeichnet sich durch folgende Eigenschaften aus:

- 10 / 100 MBit Fast-Ethernet Interface
- USB Device Interface (USB Low- und Full-Speed Support)
- USB Host-Interface
- 16-Bit A/D-Wandler
- 3 serielle RS232-Schnittstellen
- I²C-Interface (max. 2 MHz)
- Je 8 digitale Ein- und Ausgänge (Taster / LED)
- Goldcap gepufferte Echtzeituhr
- VFD-Display
- Versorgungsspannung 8 - 30V DC
- 1 Steckplatz zur Aufnahme des SCB9520

Die Abbildung 19 zeigt die Positionen der wesentlichen Schnittstellen auf.

Aktuelle Informationen sowie den Schaltplan des EVB9520 finden Sie auf der Internetseite der Firma synertronixx.

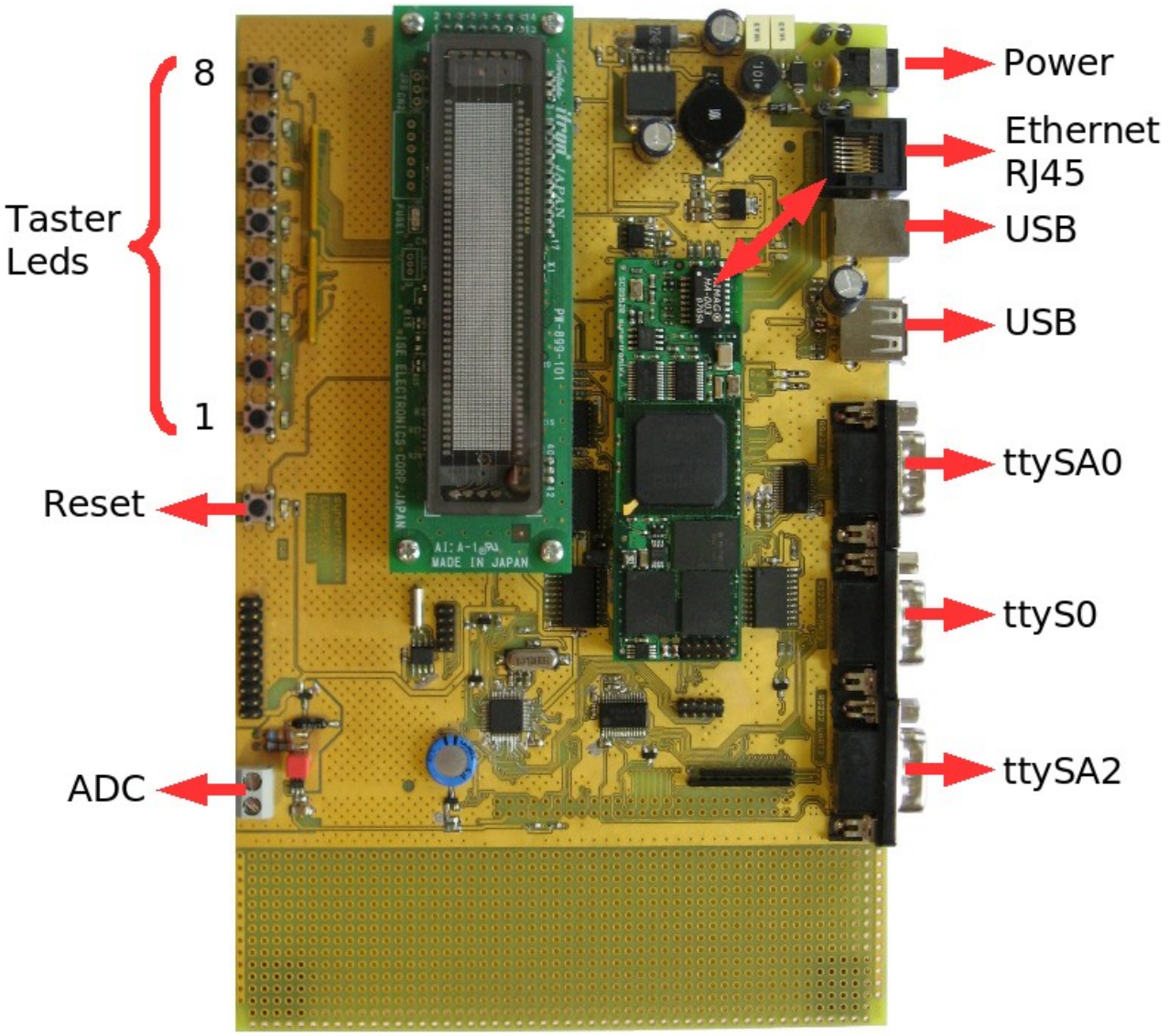


Abbildung 18: Der Aufbau des Evaluation Boards

3.1 Der Aufbau

Die Echtzeituhr und der ADC sind beide am I²C-Bus angeschlossen. Das Ansprechen der jeweiligen Bausteine erfolgt über ihre Adresse, welche aus dem Datenblatt zu entnehmen ist. Die Taster und LEDs sind am gleichen Datenbus (D0 - D7) angeschlossen.

Der Zugriff auf das VFD, die Taster und LEDs wird über den Adress-Decoder gesteuert (Abbildung 18). Wenn das negierte CS3 low ist und A16 und A17 ebenfalls low sind, wird über den Adress-Decoder das ebenfalls invertierte CS_541 am Decoder-Ausgang low.

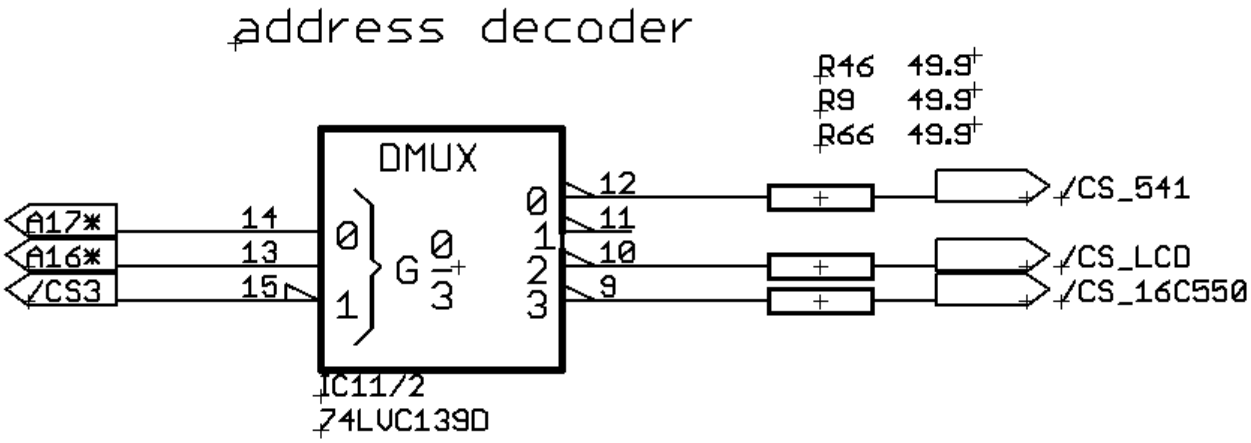


Abbildung 19: Der Adress-Dekoder

Das /CS_541 läuft wiederum in einen weiteren Adress-Decoder (siehe Bild 20). An diesem sind auch /RD und das /CS3 angeschlossen. Sind /WR, /CS_541 low und /CS3 low, ist /CS541_WR* low und die LEDs werden gesetzt.

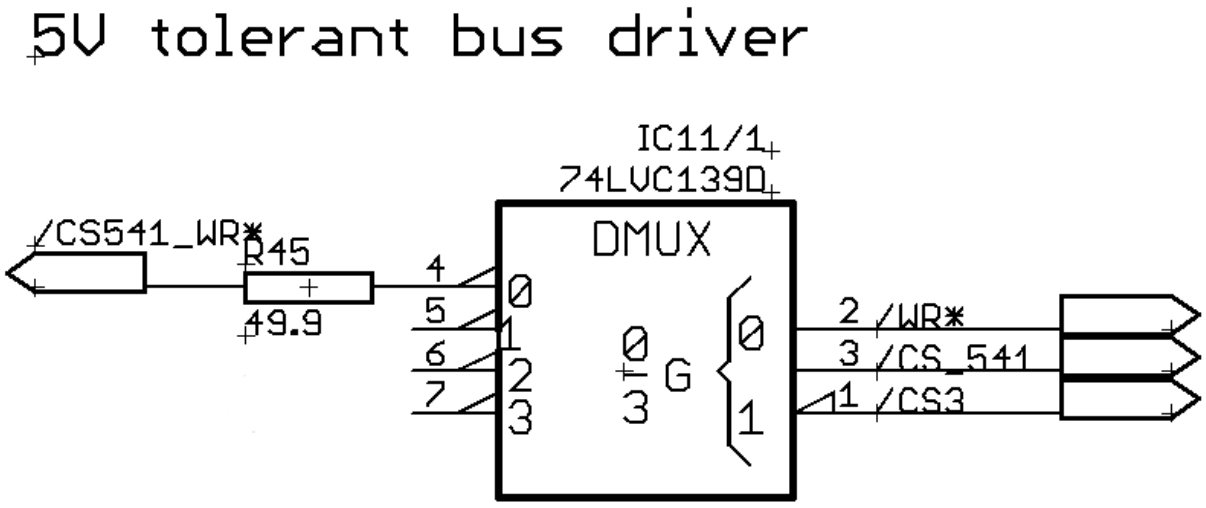


Abbildung 20: Die Unterscheidung zwischen Einlesen und Ausgeben.

Um die Taster auslesen zu können, muss /CS_541 low sein und zusätzlich muss /WR low sein (näheres zur Beschaltung siehe Abbildung 21). Der Zustand von /CS_541 kann softwareseitig, also innerhalb des Treibers beeinflusst werden. Das /RD-Signal wechselt zyklisch und gegensätzlich zum /WR-Signal seinen Zustand. Dieser Wechsel ist innerhalb des PXA270 realisiert.

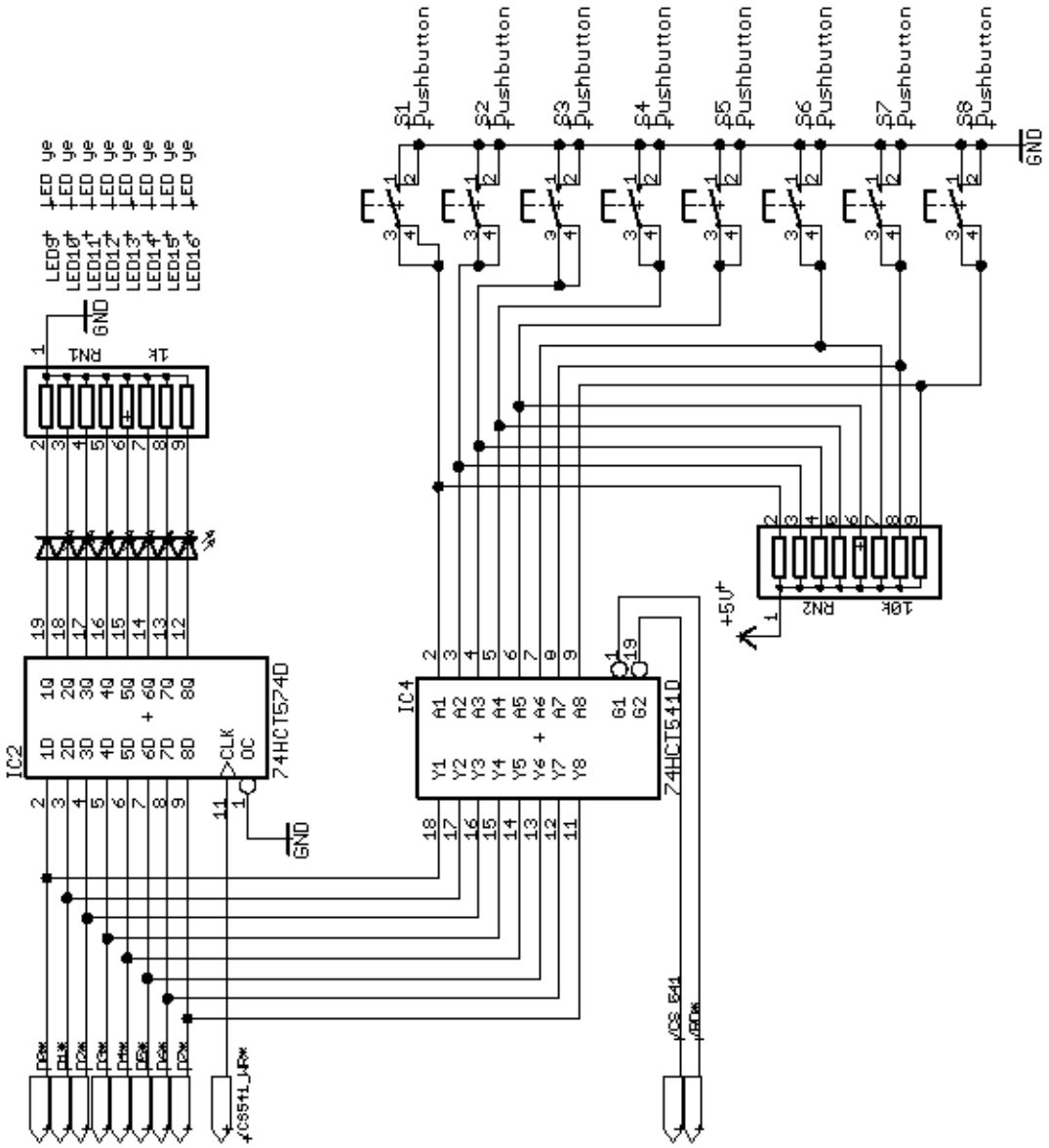


Abbildung 21: Die Beschaltung der Taster und LEDs.

3.2 Die Echtzeituhr

Der M41T00 hat einen niedrigen Stromverbrauch von 0,3 mA. Für weitere Daten ist das Datenblatt heranzuziehen. Die Echtzeituhr kann, von einem Goldcap versorgt, ungefähr eine Woche ohne Batterie- oder Netzteil-Betrieb die korrekte Uhrzeit liefern. Es hat je

einen BCD-codierten Zähler für das Jahrtausend, das Jahr, den Monat, den Tag, die Stunde, die Minute und die Sekunde.

Die Beschaltung des M41T00 ist in Bild 22 zu sehen.

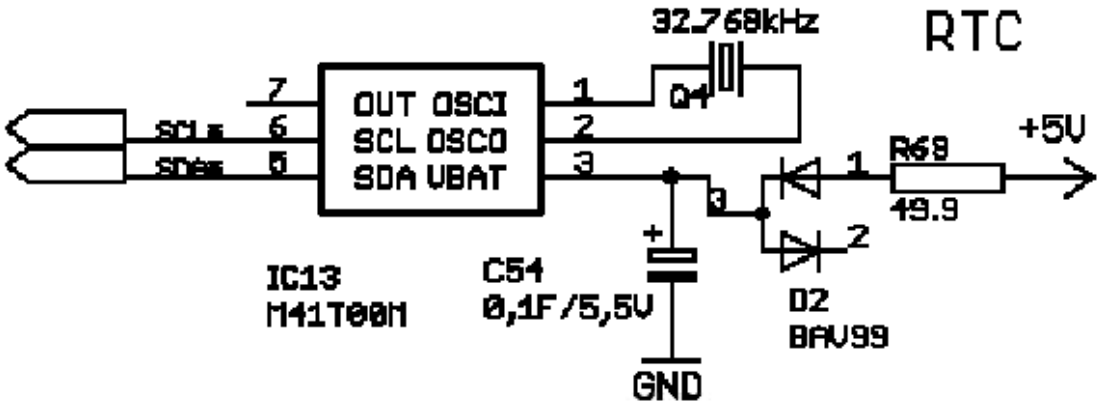


Abbildung 22: Die Beschaltung des M41T00

Der Zugriff auf die Echtzeituhr erfolgt über den linux-internen, generischen I²C-Bus-Treiber.

Beim Auslesen oder Schreiben des M41T00 wird ein Array von acht Bytes gelesen oder geschrieben.

Adresse	Daten								Funktion	Reichweite
	D7	D6	D5	D4	D3	D2	D1	D0		
0	ST	10 Sek			Sekunde				Sekunde	00-59
1	X	10 Min			Minute				Minute	00-59
2	CEB	CB	10 Std		Std				Jahr/Std	0-1/00-23
3	X	X	X	X	X	Tag			Tag	1-7
4	X	X	10 Datum		Datum				Datum	1-31
5	X	X	X	X	Monat				Monat	1-12
6	10 Jahr				Jahr				Jahr	0-99
7	OUT	FT	S	Kalibrierung					Kontroll	

Tabelle 2: Die Register des m41t00

In Tabelle 2 ist der Inhalt der Register des M41T00 zu sehen. Die mit X markierten Bits enthalten keine nutzbaren Informationen. Mit dem Setzen des ST-Bit auf Eins, stoppt der Oszillator. Das Setzen des Bits auf Null lässt ihn innerhalb einer Sekunde starten. Ist das CEB-Bit gesetzt, toggelt das CB-Bit bei jedem Jahrtausend-Wechsel. Die Bits S, FT und OUT dienen zum Messen der Abweichung des Oszillators und werden nicht ausgewertet. Die Zeit-Informationen sind BCD-codiert und mit folgenden Formeln in das jeweilige

Zahlensystem umzurechnen:

$$X_{10} = \left(\frac{X_{16}}{16} \right) \cdot 10 + X_{16} \, mod \, 10 \tag{1}$$

$$X_{16} = \left(\frac{X_{10}}{10} \right) \cdot 16 + X_{10} \, mod \, 16 \tag{2}$$

3.3 Der AD-Wandler

Der AD-Wandler ADS1110 hat eine Stromaufnahme von 0,24 mA, eine 16 Bit-Auflösung und eine einstellbare Abtastrate. Zusätzlich bietet er vier verschiedene Verstärkungen. Die Eigenschaften sind dem Datenblatt entnommen worden.

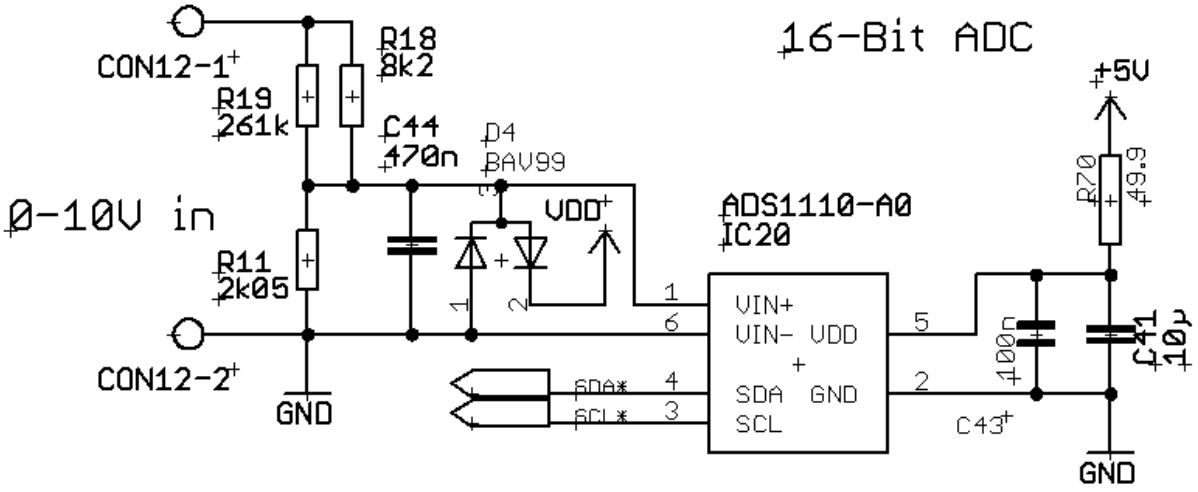


Abbildung 23: Der ADC

Der ADS1110 wird an den I²C-Bus angeschlossen und kann ebenfalls über den generischen Linux-Treiber angesteuert werden. Desweiteren gibt es den ADS1110 in acht verschiedenen Ausführungen, wo er verschiedene Adressen bietet. Dies hat den Vorteil, dass Überschneidungen der Adressen auf dem Bus umgangen werden. So können mehrere AD-Wandler am gleichen Bus betrieben werden.

In Abbildung 23 ist der Anschluss zu sehen. Der ADS1110 kann mit einer Abtastrate 15, 30, 60 oder 240Hz betrieben werden. Die Verstärkung ist frei wählbar von 1, 2, 4 oder 8 und dient dazu, kleinere Spannungen mit höherer Auflösung zu messen. Falls eine größere Verstärkung als Eins gewählt ist, bleibt das Ergebnis in den Datenregistern korrekt, da die Formel zur Berechnung die Verstärkung berücksichtigt (siehe Formel 4). Die Ausgabe des Bausteins ist eine skalare Größe, welche proportional zur angelegten Spannung ist.

Es gibt drei Register mit jeweils acht Byte, die ersten beiden beinhalten die Daten und das letzte die Konfiguration des Bausteins.

Der ADS1110 kann beliebig oft ausgelesen werden. Das heißt, wird der Wert in den Datenregistern zwischen zwei Auslesezyklen nicht aktualisiert, liest man zwei- oder mehrmals den selben Wert aus. Ein Register bzw. ein Bit im Steuerregister, welches eine vollendete Analog-Digital-Wandlung anzeigt, gibt es somit nicht.

Bit	7	6	5	4	3	2	1	0
Name	ST	0	0	SC	DR1	DR0	PGA1	PGA0

Tabelle 3: Das Konfigurationsregister des ADS1110

In der Tabelle 3 ist das Konfigurationsregister zu sehen. Das siebte Bit ist für den kontinuierlichen Betrieb irrelevant, das Bit Vier ist standardmäßig auf Null gesetzt und steuert, ob kontinuierlich oder einmalig die Differenzspannung am Eingang eingelesen wird. Die Bits Drei und Zwei setzen die Abtastrate fest (siehe Tab.4) und die Bits Eins bis Null die Verstärkung (siehe Tabelle 5).

DR1	DR0	Datenrate	maxcode (max. Auflösung)	mincode (min. Auflösung)
0	0	240SPS	2047	2048
0	1	60SPS	8191	8192
1	0	30SPS	16383	16384
1	1	15SPS	32767	32768

Tabelle 4: Die Bits zur Steuerung der Abtastrate

PGA1	PGA0	gain (Verstärkung)
0	0	1
0	1	2
1	0	4
1	1	8

Tabelle 5: Die Bits zur Festlegung der Verstärkung

Die Differenzspannung ist mit folgender Formel 3 zu berechnen. Es empfiehlt sich, noch den Faktor 1000 in die Formel einzubinden. So bleibt die Berechnung im Integer-Bereich und eine rechenintensive Float-Berechnung kann umgangen werden.

$$\Delta U = \frac{daten \cdot 2{,}048}{gain \cdot (-mincode)} \qquad (3)$$

$$\Delta U = \frac{daten \cdot 10}{gain \cdot (-mincode)} \quad (4)$$

Der Spannungsteiler am Eingang des ADS1110 hat eine Verhältnis von Eins zu 4,878 (siehe Formel 7). Um die korrekte Spannung zu ermitteln, muss die Spannung mit dem Faktor 4,878 multipliziert werden. Der Faktor 2,048 multipliziert mit 4,878 ist 9,99 und wird auf 10 aufgerundet und in die Formel 4 eingesetzt.

$$\Delta U = U_0 \cdot \left(\frac{2050\,\Omega}{2050\,\Omega \cdot \left(\frac{8\text{k}2\,\Omega \cdot 261\,k\,\Omega}{8\text{k}2\,\Omega + 261\,k\,\Omega} \right)} \right) \quad (5)$$

$$\Delta U = U_0 \frac{2{,}05}{10} \quad (6)$$

$$\Delta U = \frac{1}{4{,}878} \quad (7)$$

3.4 Das Display

Das auf dem Evaluation Board verwendete Display ist ein Vacuum Fluorescent Display (VFD) von der Firma Noritake-itron. Allgemein zeichnen sich VFDs durch eine flache Bauweise und einen weiten Blickwinkel aus. Das Display wird über die 8 Bit breite parallele Schnittstelle angesteuert. Die Spannungsversorgung beträgt 5V und das Display hat 11 verschiedene Monospace- und auch Proportional-Schriften.

Die Ansteuerung des VFDs ist in Bild 24 zu sehen.

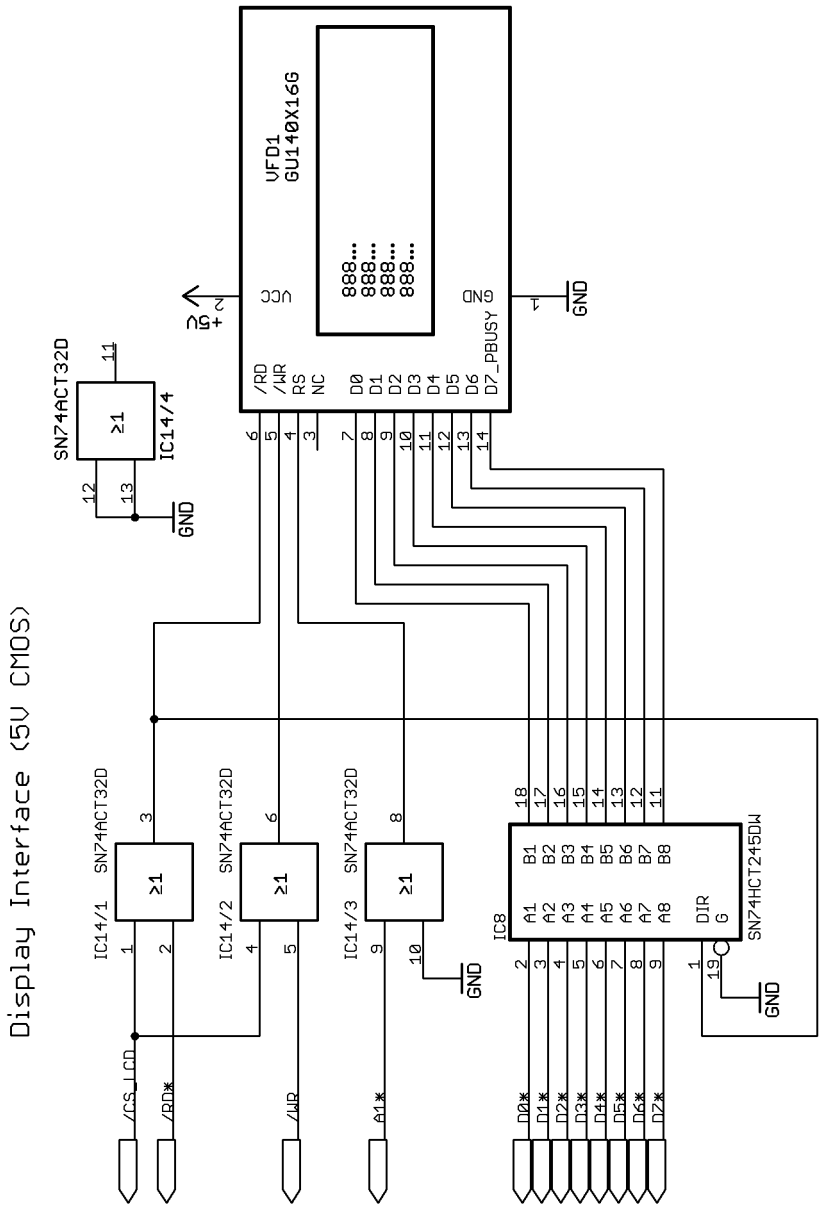


Abbildung 24: Das VFD-Display

Auf das Display können Daten geschrieben werden, wenn /CS_LCD und /WR low sind. /CS_LCD ist low, wenn am Adress-Decoder /CS3 und A16 low und A17 high ist.

Der Pin RS des Displays findet keine Verwendung. Bei anderen Displays dient das RS-Bit zur Unterscheidung zwischen Daten- und Kommando-Wort. Diese Unterscheidung ist hier anders gelöst. Alle Daten werden Ziffer für Ziffer übermittelt und liegen zwischen 20H und FFH. Die Kommando-Sequenzen fangen jeweils mit einer Zahl an, die kleiner als 20H ist.

3.5 Die Schnittstellen

Als Schnittstellen finden die RS232-, die Ethernet-Schnittstelle und der I²C-Bus Verwendung.

Hierfür werden die bereits im Linux-Kernel enthaltenen Treiber verwendet. Für die externe RS232-Schnittstelle (/dev/ttyS0) muss das Kernel-Modul 8520 geladen sein. Dieses erfolgt nicht automatisch, sondern muss explizit in einem rc-Skript realisiert sein.

Die Module *i2c_core*, *i2c_pxa* und *i2c_dev* werden für Kommunikation über den I²C-Bus benötigt. Linux erkennt den Bus und lädt automatisch das *i2c_core*-Modul. Die anderen beiden werden ebenfalls im rc-Skript beim Boot-Vorgang geladen.

4 Die Software

Die Software stellt für den Kunden einen Einstieg in die Programmierung von Linux im Bereich eingebetteter System dar. Hierfür findet die *toolchain buildroot* ihren Einsatz. Sie ist zum cross-kompilieren für ARM-Prozessoren kompiliert.

Das Evaluation Board wird bereits mit der Software ausgeliefert, verändert der Kunde diese, muss er die veränderten Programme nach /root/bin kopieren. Das Programm *scp* dient zum Kopieren von Dateien und Daten über das SSH-Protokoll. Die Module sind in Ordner /usr/local/bin zu kopieren. Das Kopieren kann mit dem *scp* vorgenommen werden, da ein SSH-Server (openSSH) auf dem Board aktiv ist. Der Name scp steht für Secure Copy und ist ein Programm zum verschlüsselten Kopieren von Daten innerhalb eines Netzwerks. Das SSH-Protokoll ermöglicht zudem das verschlüsselte Anmelden auf entfernte Rechner. Auf den meisten Linux-System findet OpenSSH, eine freie Implementation des SSH-Protokolls, Verwendung.

Um sich auf dem seriellen Terminal anzumelden, kann ein einfaches Terminal-Programm verwendet werden. In Abbildung 25 ist das Moduldiagramm über den Entwicklungsaufbau zu sehen.

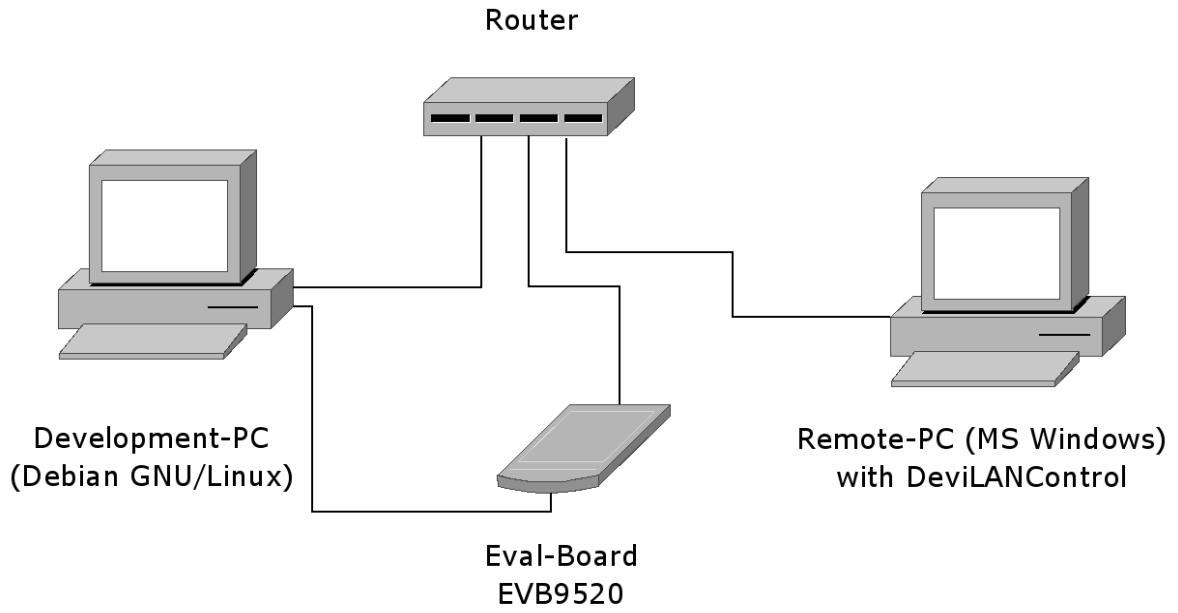


Abbildung 25: Die Verbindung zwischen Board, Entwicklungs-PC und Remote-PC.

4.1 Der Kernel

Verwendung findet der Linux-Kernel 2.6.31 und *Busybox*. *Busybox* ist eine Sammlung von Programmen und enthält die meisten auf Linux-Systemen üblichen Werkzeuge wie z.B. *ls*

(list), *cd* (change directory) und *mkdir* (make directory). Busybox-Programme sind sehr klein und eignen sich daher für den Gebrauch auf Embedded-Systemen, da hier oft sehr wenig Platz vorhanden ist. Einen Überblick über die Zusammenhänge von den Treiber-Einstiegspunkten und Modulen ist in Abbildung zu sehen. Das Bild 26 gibt nur einen Überblick. In den folgenden Kapiteln wird näher darauf eingegangen.

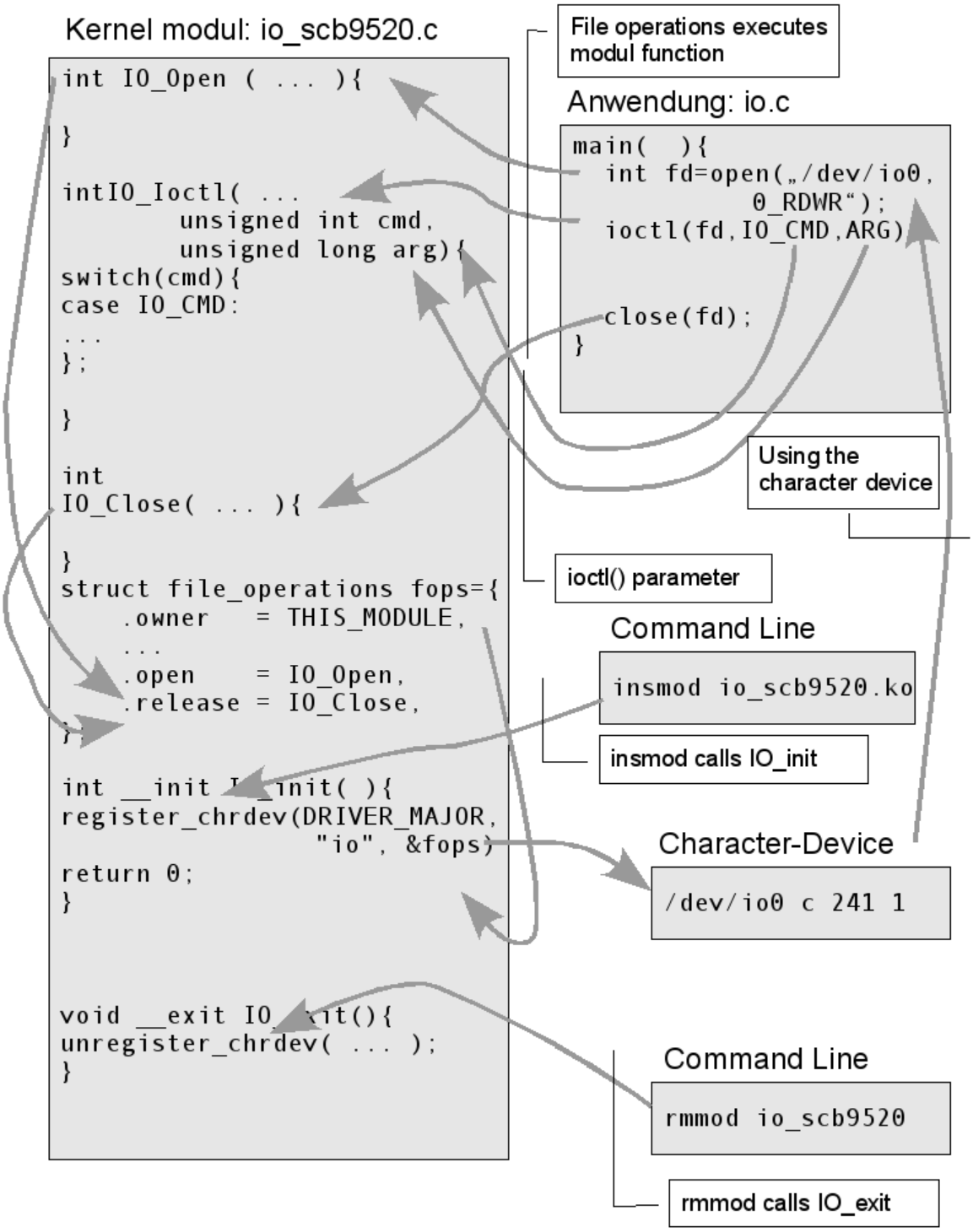


Abbildung 26: Die Verbindung zwischen Kernel-Modul und Applikation.

Die Module sollen die Funktionalität der Taster und LEDs, sowie die des VFDs bereitstellen. Die Adresse des Speicherbereichs PXA_CS3_PHYS ist in der Datei asm/arch/scb9520.h definiert. Die Module sind so entwickelt, dass später ein Zugriff auf diese Funktionen erfolgen kann. Hierbei ist darauf zu achten, sich an die Linux-Konventionen zu halten. Der Zugriff erfolgt später mit den Funktionen *open*, *close*, *write*, *read* und *ioctl*.

4.2 Das I/O-Modul

Die Entwicklung von Kernel-Modulen, um einen Treiber zu realisieren, erfolgt immer nach einem gleichen Schema. Die fünf wichtigsten Systemaufrufe sind *open*, *close*, *read*, *write* und *ioctl*. Der Name ist frei wählbar. Aufgrund besserer Lesbarkeit, wurde für den IO-Treiber (Taster und LEDs) die Vorsilbe ”IO” gewählt. Danach folgt der Name des Systemaufrufs.

- IOOpen
- IOClose
- IORead
- IOWrite
- IOIoctl

Damit später vom Userspace auf die Funktionen zugegriffen werden kann, gibt es die Struktur *struct file_operations*. Diese Funktion beinhaltet folgende Elemente.

```
struct file_operations {
        struct module *owner;
        ssize_t (*read) (struct file *, char __user *, size_t,
                                        loff_t *);
        ssize_t (*write) (struct file *, const char __user *,
                                        size_t, loff_t *);
        int (*ioctl) (struct inode *, struct file *,
                        unsigned int, unsigned long);
        int (*open) (struct inode *, struct file *);
        int (*release) (struct inode *, struct file *);
};
```

Listing 2: Einige Elemente aus der Struktur struct file_operations.

Die Struktur enthält noch mehr Elemente, diese finden aber keine Verwendung und werden deshalb ausgelassen.

Die Zuweisung der selbstgeschriebenen *IOOpen*()-Funktion zum Funktionszeiger erfolgt über die Funktionsadressen, siehe Listing 3. Mit den anderen Funktionen wird gleichermaßen vorgegangen.

Die Zuweisung des Makros THIS_MODULE zu dem Feld *owner*, erzeugt einen Entladeschutz für das Modul. Versucht ein Prozess oder ein Anwender das Modul zu

entladen und es ist noch in Gebrauch, sendet der Kernel eine Fehlermeldung und das Modul bleibt geladen.

```
static struct file_operations fops = {
        .owner   = THIS_MODULE,
        .ioctl   = IOIoctl,
        .open    = IOOpen,
        .write   = IOWrite,
        .read    = IORead,
        .release = IOClose,
};
```

Listing 3: Die Zuweisung der Funktionen zu den Elementen Struktur struct file_operations.

Linux folgt der Unix-Philosophie und somit ist alles eine Datei. Auch in der Treiber-Entwicklung wird so die Verbindung zwischen dem Kernel (Treiber) und dem Userspace hergestellt. Um auf den Treiber zuzugreifen, wird mit dem Befehl

```
mknod /dev/io0 c 242 1
```

ein Device erstellt. Der Parameter c erzeugt ein Character-Device mit Major-Number 242 und der Minor-Number 1. Ein Character-Device ist ein zeichenorientiertes Device, welches die Daten immer in der selben Reihenfolge ausgibt. Die Major-Number stellt die Beziehung zwischen dem Treiber und der sich im Dateisystem befindenden Datei her. Diese Nummer wird einmalig vergeben, ist aber vom Typ her char und somit auf 256 begrenzt, was einen Flaschenhals darstellt. An diesem Problem wird bereits gearbeitet. Statt der Major-Nummern sind Gerätenummern eingeführt worden. Gerätenummern sind 32 Bit lang und können somit bis zu 4 Milliarden Geräte ansprechen. Aktuell existieren beide Systeme parallel. Es wurde sich in diesem Projekt für das ältere und ausgereiftere System, den Major-Nummern, entschieden.

```
module_init( IOinit );
   module_exit( IOexit );
static int __init IOinit(void)
{
        CS3_mem = ioremap(PXA_CS3_PHYS, 0x2);
        if( register_chrdev(DRIVER_MAJOR, "io", &fops) != 0){
                printk("register_chrdev failed\n");
                return -EIO;
        }
        io_infos.amount_leds=8;
        io_infos.amount_switches=8;
        return 0;
}
static void __exit IOexit(void)
{
        unregister_chrdev(DRIVER_MAJOR,"io");
        iounmap(CS3_mem);
}
module_init( IOinit );
module_exit( IOexit );
```

Listing 4: Die Funktionen zum Laden und Entladen der Module.

Im Listing 4 sind die beiden Funktionen zu sehen, über die das Modul geladen oder entladen werden kann. Der Aufruf insmod *io_scb9520.ko* ruft die Funktion *IOinit()* auf, *register_chrdev* meldet den Treiber beim Kernel mit der Major-Nummer, einem frei wählbaren Namen und dem Zeiger auf die Variable des *struct file_operations* an. Die Funktion *ioremap()* ordnet einer reale Adresse eine virtuelle zu, der Memory Mapped Unit kurz MMU. Heutige Betriebssysteme arbeiten mit einem virtuellen Adressraum (MMU), da dieser eine Möglichkeit darstellt, dem Betriebssystem mehr virtuellen Speicher anzubieten, als physikalisch vorhanden. Der Befehl zum Entladen des Moduls heißt *rmmod* und ruft die Funktion *IOexit* auf. Hier wiederum wird die Funktion *unregister_chrdev* aufgerufen. Zum Schluss wird der virtuelle Adressraum mit der Funktion *iounmap* wieder freigegeben. Die Makros *module_init* und *module_exit* dienen dazu, dass die Lade- und die Entladefunktion des Moduls einen frei wählbaren Namen haben dürfen. Ohne diese Makros müsste *IOinit init_module* und *IOexit cleanup_module* heißen.

```
static int
IOOpen ( struct inode *io_device, struct file *instance )
{
        return 0;
}
```

Listing 5: Die IOOpen-Funktion.

In Listing 5 ist der Öffnungsdialog zu sehen. Die Parameter der Funktion sind die Strukturen struct inode **io_device* und *struct file *instance*. Der erste Parameter enthält alle Elemente, die die Gerätedatei definiert, also Zugriffsrechte und Besitzer. Der Instanz-Parameter ermöglicht die Feststellung, ob in lesendem, schreibendem oder lesendem und schreibendem Modus ein Zugriff auf die Datei erfolgen kann. Bei Bedarf kann die minor-Nummer über die Funktion *iminor()* ermittelt werden. Die Funktion *open* erlaubt auch die Zugriffsbeschränkung durchzuführen, z.B. das nur ein Prozess auf das Device zugreifen darf. Dieses ist hier nicht erforderlich und somit enthält die Funktion keine weiteren Anweisungen.

```
static int
IOClose( struct inode *io_device, struct file *instance )
{
        return 0;
}
```

Listing 6: Die IOClose-Funktion.

In der in Listing 6 zu sehenden Funktion IOClose wird das Device geschlossen.

```
static int
IOWrite( struct file *file, const char __user *user,\
                            size_t cnt, loff_t *offset )
{
        int ret_val = 0;
        struct IOInOut io_in_out;
        if(copy_from_user( &io_in_out, user, cnt)!=0){
                printk("Too much for 8 LEDs...\n");
                ret_val = -1;
        }
        writeb( io_in_out.leds, CS3_mem );
        return ret_val;
}
```

Listing 7: Die IOWrite-Funktion.

Im oberen Listing 7 ist der Schreibzugriff auf die LEDs realisiert. Mit der Funktion *copy_from_user()* wird ein Speicherbereich aus dem Userspace in den Kernelspace kopiert. Die Funktion bekommt als Parameter die Adresse der Variablen in denen zu speichern ist, die Adresse von der kopiert wird und die Größe des zu kopierenden Bereichs übergeben. Die Funktion liefert die Anzahl der noch zu kopierenden Bytes. Ist dieser Wert Null, so konnte alles kopiert werden. Die Funktion *writeb()* schreibt ein Byte an eine bestimmte virtuelle Adresse. Der erste Parameter ist das zu kopierende Byte und der zweite die virtuelle Adresse.

```
static int
IORead( struct file *file, char __user *user, size_t cnt,\
   loff_t *offset )
{
        int ret_val = 0;
        struct IOInOut io_in_out;
        io_in_out.switches=readw(CS3_mem);
        if(copy_to_user(user, &io_in_out, cnt)!=0){
                printk("Too much for 8 switches...\n");
                ret_val = -1;
        }
        return ret_val;
}
```

Listing 8: Die IORead-Funktion.

Die *IORead*-Funktion ist in Listing 8 zu sehen. Hier wird mit der *readb()*-Funktion ein Byte von einer virtuellen Adresse ausgelesen. Mit *copy_to_user()* wird der Inhalt eines Speicherbereichs vom Kernelspace in den Userpsace kopiert. Übergeben wird dieser Funktion die Adresse aus dem Userspace, die Adresse im Kernelspace und die Größe der zu kopierenden Daten. Auch hier ist der Rückgabewert die Anzahl der Bytes, welche noch nicht kopiert werden konnten.

```
static int
IOIoctl( struct inode *io_device, struct file *instance,\
                           unsigned int cmd, unsigned long arg )
{
        int ret_val;
        switch (cmd){
                case IO_GET_INFOS:
                        copy_to_user( (void*)arg, &io_infos, \
                                       sizeof(io_infos) );
                        ret_val = 1;
                        break;
                default:
                        printk("IO ioctl command not known\n");
                        ret_val = -1;
        }
        return ret_val;
}
```

Listing 9: Die IOIoctl-Funktion.

Unter Linux findet die IO-Control-Schnittstelle (*ioctl*) häufig ihre Verwendung. Diese Funktion arbeitet meist mit der switch-case-Anweisung und ermöglicht das einfache Ausführen von Befehlen. Hierfür wird dem ioctl-Kommando beim Aufruf aus der Applikation

ein Kommando übergeben, welches in der switch-case-Anweisung ausgewertet wird. Die Kommandos sind Zahlen, welche über #defines in der io_scb9520.h definiert sind. Mit den Funktionen *copy_to_user* und *copy_from_user* können Informationen zwischen User- und Kernelspace kopiert werden. Das Listing 9 zeigt ein Beispiel hierfür auf.

4.2.1 Das VFD-Modul

Es folgen die Funktionen aus dem vfd_scb9520-Modul. Dieses Modul dient zur Ansteuerung des Displays der Firma Noritake-itron. Dieses Modul hat ebenfalls eine init- und eine *exit*-Funktion, in denen das Device mit *register_chrdev* angemeldet und mit *unregister_chrdev* am Kernel abgemeldet wird. Zusätzlich, wird die Funktion *VFDFirstInit()* aufgerufen. Diese Funktion sendet die Kommando-Worte 1BH, 40H und 0CH mit der bereits bekannten *writeb()*-Funktion an das VFD. Die Befehlssequenz von 1BH und 40H bewirkt das Rücksetzen des VFD auf die Standard-Einstellung und der Befehl 0CH löscht die Anzeige. In der Entlade-Funktion erfolgt das Ausschalten des Displays. Hierfür wird die Folge 1FH, 28H, 61H, 40H und 00H an das Display geschickt.

```
static void
WaitPBusy (void)
{
        while(readb(CS3_mem+2)&0x80);
        udelay(10);
}
```

Listing 10: Die Funktion WaitPBusy() wartet auf die erfolgreiche Umsetzung des Befehls.

Beim Beschreiben des Displays ist darauf zu achten, dass der vorherige Befehl bereits abgearbeitet ist. Dies zu Realisieren ist entweder mit der *udelay()*-Funktion möglich oder man wertet das PBusy-Bit (D7-Bit) aus. Dieses Bit bleibt solange high, bis der Befehl abgearbeitet ist. Das Warten mit *delay* oder *sleep* stellt einen unnötigen Ressourcenverbrauch dar und sollte vermieden werden. In der Praxis hat sich herausgestellt, dass das Warten auf PBusy nur beim Schreiben von Text auf das Display funktioniert. Das Senden von Befehlen in schneller Abfolge erzeugte Darstellungsfehler, die aufzeigten, dass Befehlswörter länger brauchen und somit eine längere Wartezeit erfordern. Als Lösung musste hier doch auf die *delay*-Funktion zurückgegriffen werden.

4.2.2 Der I²C-Bus

Der Linux-Kernel hat einen eigenen I²C-Bus-Treiber, welcher den einfachen Zugriff auf den Bus mit Hilfe der Befehle *open*, *close*, *read*, *write* und *ioctl* ermöglicht.

Die I²C-Bus-Spezifikationen sind dem Datenblatt der Firma Philips entnommen.

4.3 Die Applikation

4.3.1 Die Applikation-Beschreibung

Die Applikation soll dem Kunden als Demonstration dienen und den Einstieg in die Software-Entwicklung unter Linux erleichtern. Die Entwicklung der Applikation erfolgte mit Modulen und Headern. Das Projekt gliedert sich in die nachfolgenden C- und Header-Dateien auf.

- evb9520.c
- ads.c/h
- clock.c/h
- io.c/h
- serial_comA2.c/h
- serial_com0.c/h
- server.c/h
- terminal.c/h
- vfd.c/h

Mit Hilfe der Taster soll das auf dem Evaluation Board laufende Programm gesteuert werden. Gefordert ist ein Steuermenü, welches über die Taster gesteuert wird. Die LEDs sollen in Abhängigkeit zu den gedrückten Tastern ihren Zustand wechseln. Leuchten sie, sollen sie erlöschen und umgekehrt. Die Taster steuern ein Menü in dem die Uhrzeit ausgelesen oder der Analogwert am ADS1110 auf dem VFD ausgegeben wird. Zusätzlich sind zwei Menü-Punkte zu realisieren, die die Tastatureingaben auf den beiden vorhandenen seriellen Schnittstellen /dev/ttySA2 und /dev/ttyS0 auf das VFD ausgeben.

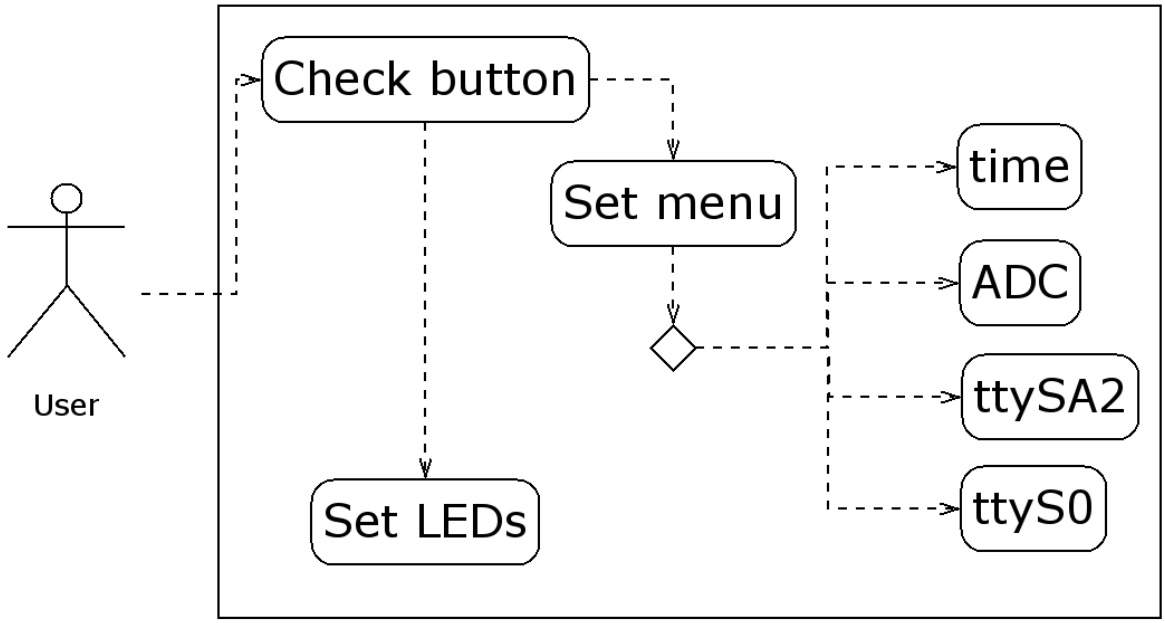


Abbildung 27: Das Anwendungsfall-Diagramm für die Steuerung über die Taster.

Desweiteren ist eine TCP-Kommunikation mit einem anderen Rechner zu realisieren. Hierfür steht bereits ein Windows-Programm zur Verfügung. Das Programm versendet bestimmte Befehle an das Board, welche vom Test-Programm zu analysieren sind. Wird ein bekannter Befehl erkannt sendet das Evaluation Board seine Kennung, setzt die LEDs, gibt den aktuellen Zustand der LEDs und Taster aus, sendet die aktuelle Zeit oder den aktuellen Analogwert am ADC.

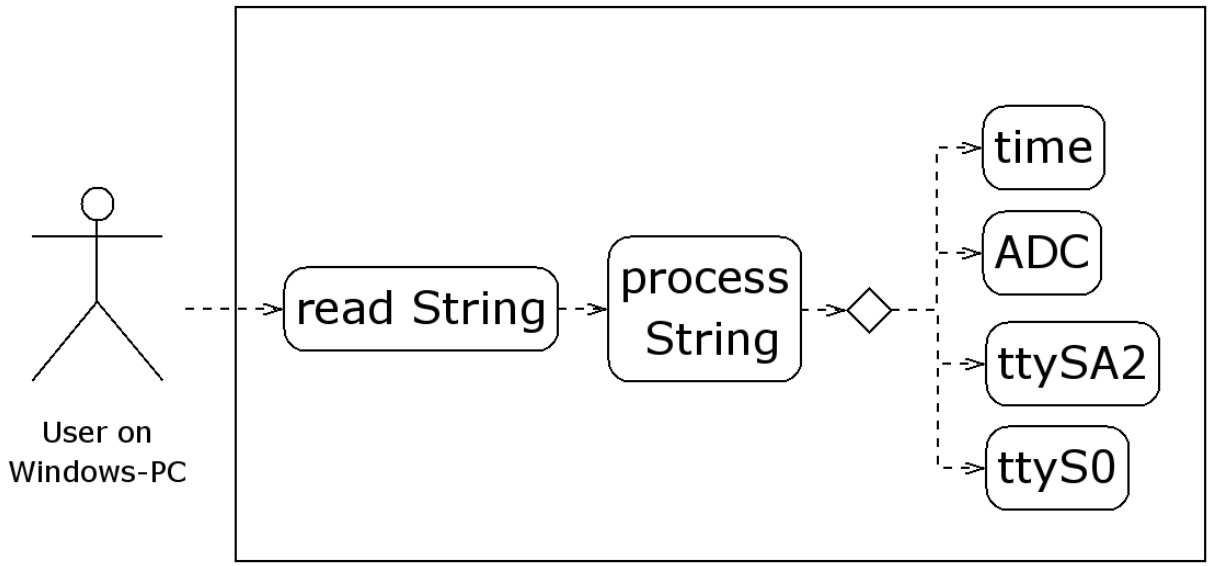


Abbildung 28: Das Anwendungsfall-Diagramm für den Zugriff über TCP.

Die Datenmodellierung

Die main-Routine beginnt mit dem Öffnen der Dateien vfd0, i2c0 ,io0 ,ttyS0 und ttySA2. Diese Dateien befinden sich im /dev-Verzeichnis, in der sich alle Device-Dateien befinden. Wenn eine Datei sich nicht öffnen lässt, erfolgt ein Programmabbruch. Die seriellen Schnittstellen sind nicht blockierend eingestellt.

In Bild 29 ist das Datenfluss-Diagramm für die main-Funktion zu sehen. In der main-Funktion läuft eine do-while-Schleife so lange, bis die Variable running gleich Null ist. In dieser Schleife erfolgt das Auslesen der Taster. Außerdem setzt die Funktion *IOSetLeds()* die LEDs. Erfolgt eine Eingabe, wird die *SetMenu*-Funktion aufgerufen. Parallel dazu wird mit der *select*-Anweisung der TCP-Port überwacht. Erfolgt ein Dateneingang über das Netzwerk, so liest die Funktion *ServerGetMsg* sie ein und übergibt sie der Funktion *DataHandler()*.

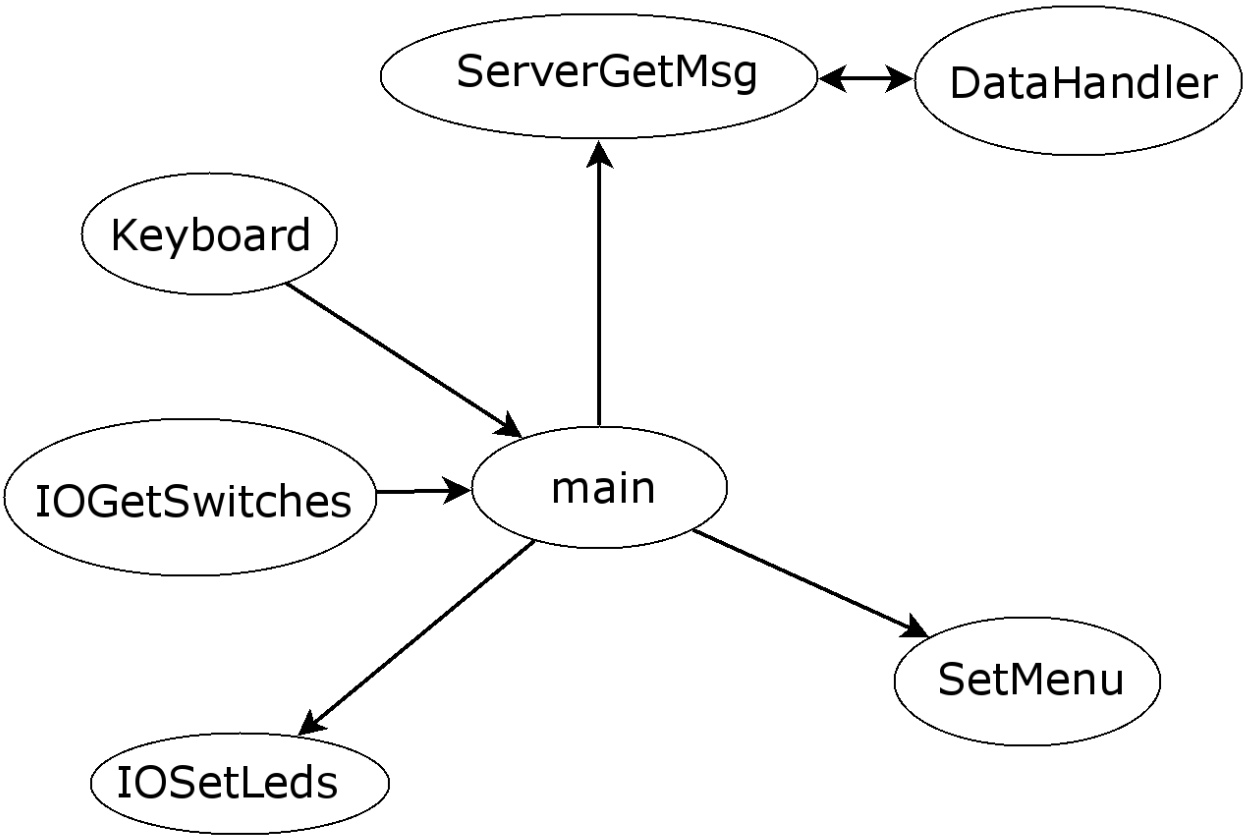


Abbildung 29: Das Datenfluss-Diagramm für die main-Funktion.

In Abbildung 30 ist die *SetMenu*-Funktion feiner aufgelöst. Diese Funktion erzeugt ein Menü in Abhängigkeit zur menu-Variable.

- Menü 0 : Ausgabe der Zeit.
- Menü 1 : Ausgabe des Analogwertes am ADC.
- Menü 2 : Ausgabe der Eingaben des ttySA2-Devices.
- Menü 3 : Ausgabe der Eingaben des ttyS0-Devices.
- Menü 4-9 : Frei, zur Verwendung durch den Kunden.

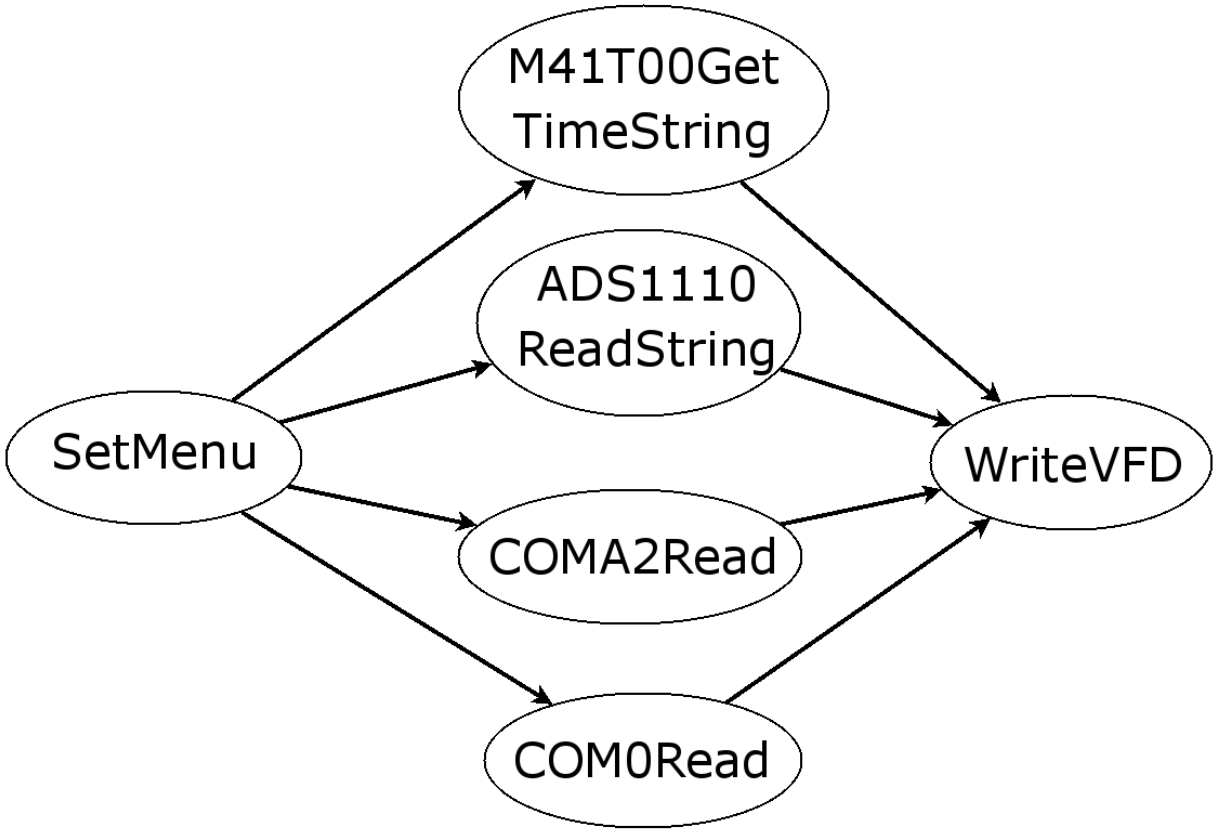


Abbildung 30: Das Datenfluss-Diagramm für die SetMenu-Funktion.

In Bild 31 ist die Funktion *DataHandler* näher erklärt. Die *SeverGetMsg*-Funktion liest den TCP-Stream bis zum ersten Zeilenende-Zeichen aus und ruft dann die *DataHandler*-Funktion auf. Diese Funktion wertet mit Hilfe der *strstr()*-Funktion die Zeichenkette aus und führt in Abhängigkeit der gefundenen Teil-Strings verschiedene Funktionen aus.

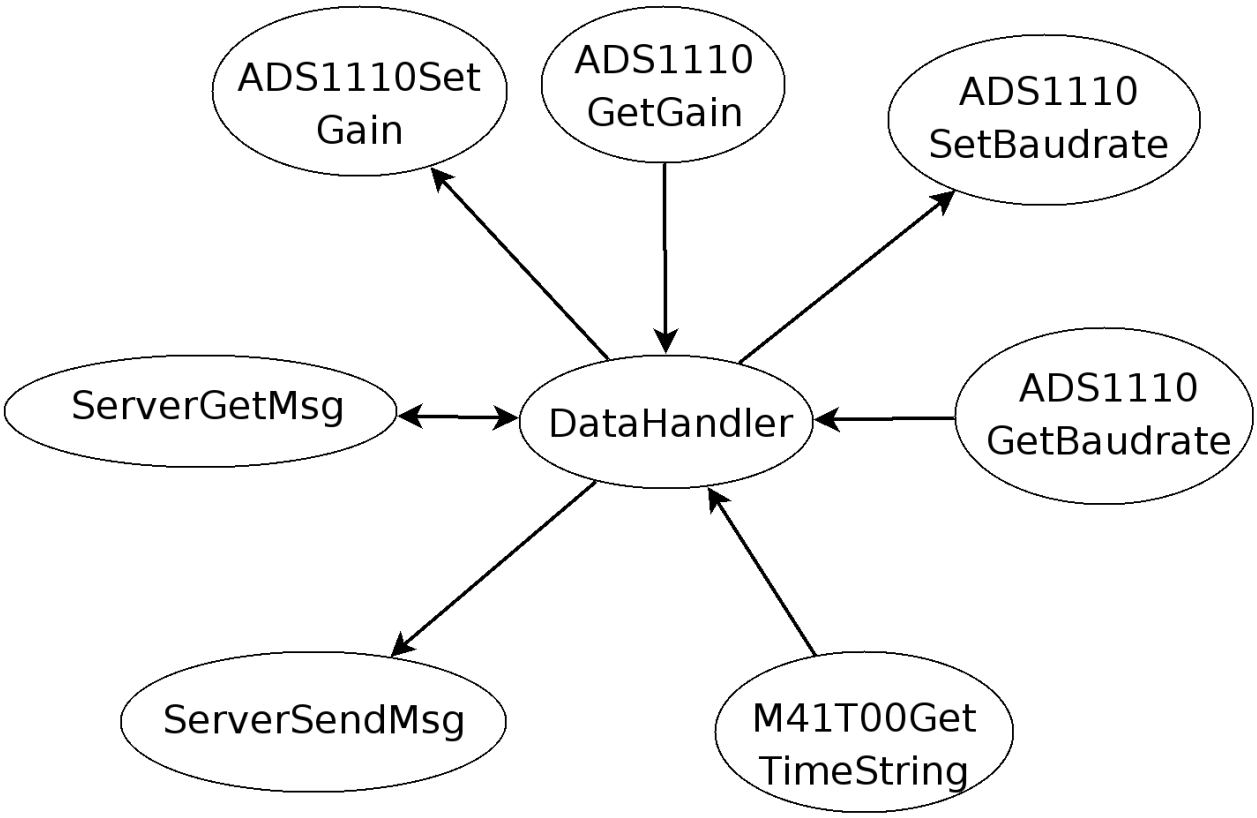


Abbildung 31: Das Datenfluss-Diagramm für die DataHandler-Funktion.

4.3.2 Die Ein- und Ausgabefunktion

Der Zugriff und die Abfrage auf die Taster und die LEDs erfolgt über das Device /dev/io0, welches beim Bootvorgang von dem rc-skript S99scb9520 erstellt wird. Die IOs können gelesen und gesetzt werden. Der Zugriff erfolgt über die Funktionen *open*, *close*, *write*, *read*. Über *ioctl* sind zusätzlich noch Informationen abrufbar. Der ioctl-Aufruf kopiert eine Struktur *IOInOut* von dem Kernel- in den Userspace. Diese Struktur enthält die Anzahl der LEDs und Taster und soll dem Kunden als einfaches Beispiel für die Benutzung von ioctl-Kommandos dienen.

Es ist möglich, die aktuell gedrückten Taster auszulesen, die LEDs zu setzen und die betätigten Taster an den LEDs auszugeben. Desweiteren ist die Funktionen *ToggleBit()* realisiert worden, sie prüft den aktuellen Zustand einer LED und kehrt ihn um.

4.3.3 Die Echtzeituhr

Die Echtzeituhr ist über den I²C-Bus in das Evaluation Board integriert. Die Verwendung des I²C-Busses erfordert immer, dass einer der beiden Kommunikationspartner Master und der andere Slave ist. Bei jeder Verwendung des Busses ist der Socket Computer der Master und der angesprochene Baustein der Slave. Als erstes wird das i2c0-Device mit

der open-Funktion geöffnet. Jeder Schreib- oder Lese-Zyklus beginnt mit dem Aufruf

```
ioctl(fd_clock, I2C_SLAVE_FORCE, 0x68)
```

fd_clock ist der Filedescriptor auf das I²C-Device, I2C_SLAVE_FORCE ist ein im generischen Treiber realisierter Befehl der den angesprochenen Baustein als Slave setzt. Als letztes wird die Adresse 68H, die im M41T00 einprogrammiert ist, als Parameter mitgegeben.

Danach folgt entweder das Auslesen oder das Schreiben der Gerätedatei. Beim Lese- und Schreibvorgang sollten laut Datenblatt immer acht Bytes gelesen oder geschrieben werden. Die Formeln sind in den Funktionen *DecToBcd*() und *BcdToDec*() ausgelagert. Die Bits, die umzurechnen sind, werden maskiert und dann an eine der beiden Funktionen als Parameter übergeben.

Die Ermittlung des Wochentags erfolgt mit dem unten aufgeführten Feld.

```
char dow_field[7][4]={"Mon\0","Tue\0","Wed\0","Thu\0","Fri\0",
                                                "Sat\0","Sun\0"};
```

Bevor der M41T00 ausgelesen werden kann, muss er bereits einmal beschrieben worden sein.

4.3.4 Der AD-Wandler

Der AD-Wandler ADS1110 misst die differentielle Spannung an den Eingangs-Pins VIN- und VIN+ . Der Zugriff erfolgt ebenfalls über den I²C-Bus. Der Baustein hat aber die Adresse 4AH. Zuerst ist das Device mit *open()* zu öffnen. Der Filedescriptor für den ADC heißt fd_ads1110 und das Device liegt in /dev und heißt i2c0.

```
ioctl(fd_ads1110, I2C_SLAVE_FORCE, 0x4A)
```

Auch hier muss vor dem Lesen und Schreiben das Device als Slave mit dem obigen Befehl gesetzt werden. Laut Datenblatt sind immer drei Bytes auszulesen und nur ein Byte zu schreiben.

Beim Lesen steht in den ersten beiden Bytes der Wert für die differentielle Spannung und das letzte Byte enthält die Konfiguration. Die Konfiguration besteht im Wesentlichen aus der Verstärkung und der Abtastrate. Die Einstellung auf kontinuierlichen Modus oder Einzelmessung ist einmalig in der Startsequenz vorzunehmen.

```
int config_reg[4][2]={{2048,1}, {8192,2}, {16384,4},{32768,8}};
```

Die Ermittlung der Verstärkung und der Baudrate erfolgt mit dem angegebenen Befehl. Im config_reg-Feld steht jeweils in der ersten Zeile die minimale Auflösung mincode.

Die beiden Bits für die Verstärkung sind im dritten ausgelesenen Byte an Position Null und Eins zu finden. Die Baudrate steht im dritten und vierten Bit. Zum Lesen der Verstärkung sind die beiden Bits zu maskieren und mit Hilfe des config_reg-Arrays auszugeben. Für die Baudrate sind ebenfalls die beiden Bits zu maskieren und zusätzlich um zwei Positionen nach links zu verschieben. Die Baudrate steht in der zweiten Zeile des Feldes.

```
gain     = config_reg[buf[2]&0x03][0];
baudrate = config_reg[(buf[2]&0x0C>>2)][1];
```

Für das Setzen der Baudrate muss zuerst die Verstärkung gelesen und dann wiederum mit der gewünschten Baudrate verodert werden. Vorher müssen die zu setzenden Bits mit

einer switch-case-Anweisung ermittelt und die Bits um zwei Positionen nach rechts verschoben werden. Diese Vorgehensweise ist bei dem Setzen der Verstärkung analog durchzuführen.

4.3.5 Das VFD-Softwaremodul

Zum Benutzten des Displays muss das Kernel-Modul vfd_scb9520 geladen sein.

Auch hier erfolgt der Zugriff auf das Display über die Funktionen *open*, *close*, *read*, *write* und *ioctl*. Für den Befehl *icotl* sind mehrere Funktionen realisiert worden.

Befehl	Funktion
VFD_GET_INFO	gibt den Hersteller und den Namen des Displays aus
VFD_OFF	schaltet das VFD aus
VFD_SET_CURSOR_LINE1_POS1	setzt den Cursor in Zeile 1 an Position 1
VFD_SET_CURSOR_LINE2_POS1	setzt den Cursor in Zeile 2 an Position 1
VFD_CLEAR_DISPLAY	löscht das Display
VFD_GO_XY	setzt den Cursor an die Position X Y
VFD_CLEAR_LINE_1	löscht Zeile 1
VFD_CLEAR_LINE_2	löscht Zeile 2

Tabelle 6: Mögliche ioctl-Befehle für das Display.

Anzumerken bleibt, dass die meisten Befehle als dritten Parameter NULL bekommen. Der Aufruf zum Positionieren des Cursors bekommt eine Adresse auf die Struktur vom Typ *struct vfd_point*. Mit dieser Adresse und dem bereits bekannten Befehl *copy_from_user* kopiert sich das Kernel-Modul die Koordinaten in das Modul. Der Befehl VFD_GET_INFO hat als Parameter eine Adresse auf eine Struktur vom Typ *struct vfd_infos vfd_info*. In dieser Struktur stehen Informationen über das Noritake-itron-VFD. Diese Struktur enthält drei Strings, der erste enthält den Namen des Displays und des Evaluation Board, der zweite enthält den Namen und die mögliche Auflösung und der letzte String stellt eine Kurzinformation dar und enthält nur den Namen ”GU140X16G-7002 VFD”.

4.3.6 Kommunikation über TCP-Sockets

Während das Programm läuft, ist immer ein Socket offen, um eine TCP-Verbindung mit z.B. dem Programm *DeviLANControl* aufnehmen zu können. Im Anschluss daran wird die Funktion *ServerConnect()* gestartet. Diese Funktion öffnet den Server-Socket und ermittelt den Port aus der Datei interfaces, welche im Ordner /etc/network zu finden ist. Ist hier kein Port eingetragen, erfolgt die Zuweisung des Standard-Ports 3333, der auch in die Datei interfaces eingetragen wird. Der Server-Socket wird mit der Option SO_REUSEADDR

eingestellt, welche bewirkt, dass der verwendete Port sofort wieder verwendbar ist. Normalerweise ist ein Port bis zu zwei Minuten nach Verbindungsabbau gesperrt, dies ist ein Sicherheitsmerkmal, welches hier nicht notwendig ist. Das Programm reagiert nur auf bestimmte Zeichenketten und sendet daraufhin auch keine Daten.

Bei TCP-Verbindung gehen anders als bei UDP normalerweise keine Daten verloren. Wie die Daten ankommen, als ein ganzer Frame oder in kleinen Frames, ist nicht vorherzusehen. Deshalb liest die Funktion *ServerGetMsg* solange, bis das Zeilenendezeichen 0x0D im String erkannt wird. Ist der String komplett übertragen worden, so wird die Funktion *DataHandler* aufgerufen. Diese Funktion durchsucht den String nach Schlüsselworten, wie unter anderem ”get_config” oder ”porta”. Hat es z.B. get_config gefunden, wird zuerst der Modul-Typ als String dem entfernten Rechner zugesandt. Das Programm DeviLANControl legt daraufhin einen Tabulator an, der passende Steuerelemente beinhaltet. Danach wird der Begriff ”porta” und eine zweistellige Hexadezimal-Ziffer gesucht. Diese Ziffer wird anschließend an den LEDs mit der bereits bekannten Funktion *IOSetLeds()* auf dem Board und im DeviLANControl-Programm ausgegeben. Mit der Stringsuche nach dem Begriff ”portb” erfolgt dieselbe Ausführung nur für die Taster.

Es folgt die Suche nach dem Begriff ”ADC 0”. Ist er gefunden, wird der aktuelle Analogwert in mV an das *DeviLAN*-Programm geschickt. Nach ”adc_baudrate” und ”adc_gain” wird ebenfalls gesucht, um bei Erfolg die Baudrate und die Verstärkung zu versenden.

Als letztes erfolgt die Suche nach dem Begriff ”time” und es wird bei Erfolg die aktuelle Uhrzeit des M41T00 verschickt.

Es gibt weitere Schlüsselworte, auf die die Funktion *DataHandler()* reagiert. Diese sind in Tabelle 7 nachzulesen.

Tabelle 7: Die Befehlsreferenz des DeviLANControl.

Keyword	Parameter	Beschreibung
get_config	keine	sendet den Modulnamen, den Zustand der LEDs und Taster, die Uhrzeit, die Verstärkung und die Baudrate des ADC, sowie den Analogwert am AD-Wandler
data_renew	keine	sendet aktuelle Werte des Tasters, des ADC und der Uhr
porta	leds	setzt die LEDs auf den Wert der Variable leds
adc_baudrate	ads1110_sps	setzt die Baudrate
adc_gain	ads1110_gain	setzt die Verstärkung
time	Uhrzeit (s.u.)	sendet die aktuelle Uhrzeit
mode	menu	setzt den Menü-Modus

Der Wert für die LEDs muss zwischen 0 und 255 liegen und als Hexadezimalzahl übergeben werden. Die Baudrate kann die Werte 15, 30, 60 und 240 annehmen und die Verstärkung kann die Wert 1, 2, 4 und 8 haben. Die menu-Variable hat einen Wertebereich von 0-3. Der String time bekommt folgende Sequenz angehängt: 23:12:59 Tue, 31.12.2099.

Die Schlüsselworte get_config und data_renew sind ohne Parameter zu versenden.

```
get_config
data_renew
adc_baudrate 240
adc_gain 2
time 23:12:59 Tue, 31.12.2099
mode 1
```

Falls das Programm DeviLANControl die Verbindung abbricht, kann sie jederzeit wieder aufgenommen werden. Die Anwendung wartet mit dem *select*-Befehl nicht-blockierend auf eine neue Verbindung. Die anderen Funktionen des Programms laufen unabhängig von der TCP-Kommunikation.

4.4 Das rc-Skript

Das rc-Skript S99scb9520 wird beim Bootvorgang ausgeführt. Es befindet sich in /etc/init.d. Alle Skripte, die ausführbar sind und sich in dem init.d-Ordner befinden, werden beim Bootvorgang ausgeführt. Bei jedem Linux-System wird am Anfang des Hochfahrens die Datei /etc/inittab ausgewertet, hier wird das Skript rcS ausgeführt. Dieses führt mit Hilfe

einer for-Schleife alle ausführbaren Skripte und Programme im Ordner init.d aus. Der Name S99scb9520 rührt daher, dass die besagte Schleife in Abhängigkeit der Zahl nach dem S, die Reihenfolge der Ausführung bestimmt. Damit sicher alle nötigen Dienste gestartet sind, wird das SCB-Skript als letztes ausgeführt.

Das Skript enthält eine case-Anweisung und beim Booten wird es mit dem Paramter ”start” ausgeführt. Dies hat zur Folge, dass die beiden Module für das VFD und Ein- und Ausgabe gestartet werden. Das Modul 8250 ist für die externe serielle Schnittstelle. Desweiteren erfolgt die Ausführung folgender Skripte: *io.sh*, *vfd.sh* und *serial.sh*. Diese Skripte erzeugen die Device-Dateien für die I/Os, das Display und die serielle Schnittstelle. Sie werden nur ausgeführt, falls diese Dateien noch nicht existieren. Die Fehlermeldungen werden mit einer Ausgabeumlenkung unterdrückt. Auf der Konsole gibt es, wie bei C, drei Standardausgaben. Die Standardeingabe stdin liegt auf 0, stdout liegt auf 1 und die Fehlerausgabe stderr liegt auf 2. Mit der spitzen Klammer können die jeweiligen Ausgaben in Verbindung mit ihrer Nummer umgelenkt werden. Das Device null hat hier eine Lösch-Funktion. Die Programme *evb9520* und *udp_config* werden beide mit dem Parameter ”&” aufgerufen. Dies bewirkt, dass das Programm in den Hintergrund gesetzt wird und das Terminal weiterhin benutzbar ist. Mit den Befehlen *top* oder *ps* ist es möglich, sich eine Liste aller aktiven Programme anzusehen. Die von *top* ausgegebene Tabelle zeigt die Process Identifier-Nummer (PID) an. Mit dem Befehl *kill* und der PID als Parameter kann so ein Programm beendet werden.

```
#!/bin/sh
if [ "$VERBOSE" != no ]
then
  echo -n "Initializing EVB9520... "
  echo ""
fi
case "$1" in
  start|"starting all modules and apps")
  if [ `ls -l /dev/io0 2>/dev/null|wc -l` -eq 0 ]
  then
    echo /dev/io0 missing, starting io.sh to create it
    /root/io.sh
  else
    echo /dev/io0 already exists, so nothing has to be done
  fi
  if [ `ls -l /dev/vfd0 2>/dev/null|wc -l` -eq 0 ]
  then
    echo /dev/vfd0 missing, starting vfd.sh to create it
    /root/vfd.sh
  else
    echo /dev/vfd0 already exists, so nothing has to be done
  fi
  if [ `ls -l /dev/ttyS0 2>/dev/null|wc -l` -eq 0 ]
  then
    echo /dev/ttyS0 missing, starting serial.sh to create it
    /root/serial.sh
  else
    echo /dev/ttyS0 already exists, so nothing has to be done
  fi
  modprobe i2c-dev
  modprobe i2c-pxa
  modprobe 8250
```

```
  insmod /root/modules/io_scb9520.ko 2>/dev/null
  insmod /root/modules/vfd_scb9520.ko 2>/dev/null
  /usr/local/bin/udp_configure &
  /usr/local/bin/evb9520    &
  ;;
  stop)
  echo "stopping all modules"
  killall evb9520    2>/dev/null
  killall udp_configure 2>/dev/null
  rmmod i2c-dev 2>/dev/null
  rmmod i2c-pxa 2>/dev/null
  rmmod /root/modules/io_scb9520.ko 2>/dev/null
  rmmod /root/modules/vfd_scb9520.ko 2>/dev/null
  echo "removing all files"
  rm /dev/io0 2>/dev/null
  rm /dev/vfd0 2>/dev/null
  ;;
  *)
  echo "Usage:S99scb9520 {start|stop}" >&2
  exit 1
  ;;
esac
```

Listing 11: Das rc-Skript

Im obigen Listing 11 wird geprüft, ob die Dateien /dev/io0, /dev/vfd0 und /dev/ttyS0 vorhanden sind. Sind es sie nicht, so werden sie erstellt, ansonsten erfolgt eine Meldung das nichts zu tun ist.

Mit dem Parameter stop erfolgt die Entladung der Module io_scb9520 und vfd_scb9520 und die Character-Dateien werden gelöscht. Auch hier erfolgt das Löschen der Fehlerausgaben. Alle Fehlermeldungen von *rmmod* würden lediglich bedeuten, dass das Modul schon entladen ist und somit machen sie hier keinen Sinn. Der Befehl< *killall* beendet Prozesse in Abhängigkeit des Namens.

Die Shellskripte, die die Device-Dateien erstellen, sehen alle sehr ähnlich aus. Sie enthalten den unten aufgeführten Befehl.

```
mknod /dev/vfd0 c $IO_MAJOR $IO_MINOR
```

Mit dem Befehl *mknod* werden Gerätedateien erzeugt. Der Parameter -c steht für ein Character-Device und darauf folgen die Major bzw. Minor-Nummer.

Die Programme *udp_config* und *evb9520* werden vom rc-Skript beim Booten im Hintergrund gestarted. Zum Beenden der Programme kann der Befehl *killall*, wie oben im rc-Skript zu sehen, verwendet werden.

```
killall evb9520
killall udp_config
```