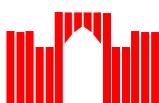
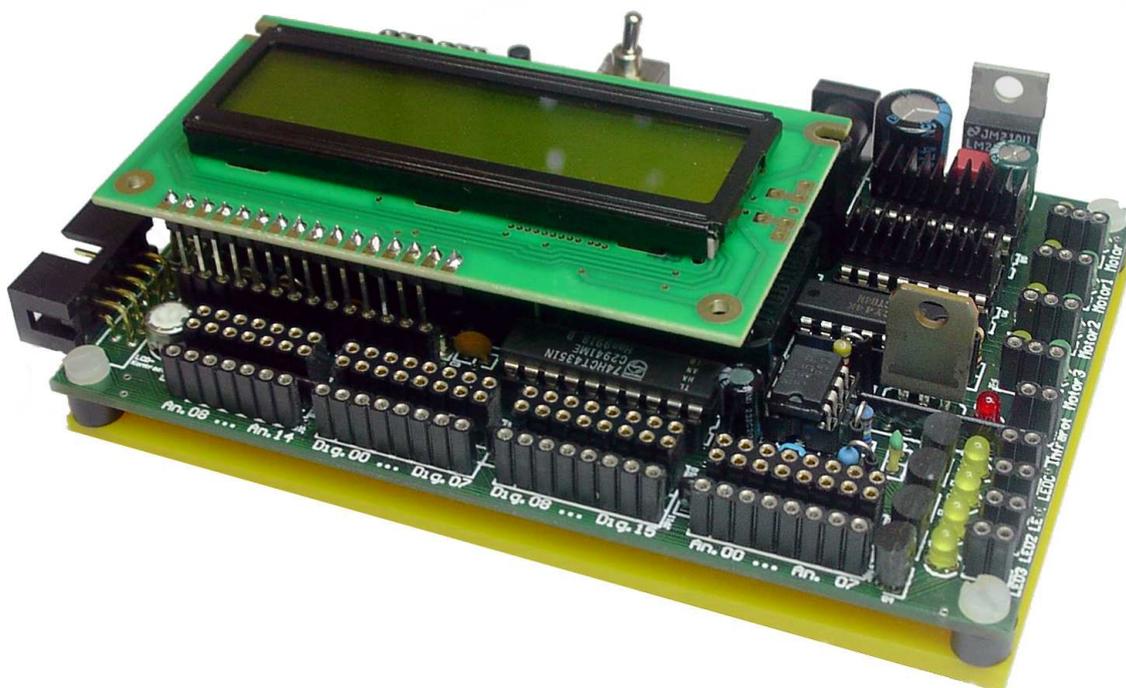

Nutzerhandbuch AKSEN-Board

Version 1.04 zur AKSEN-Bibliothek 0.965



Inhaltsverzeichnis

Einleitung	v
1 Erste Schritte	1
1.1 Inbetriebnahme	1
1.2 Installation der Software	2
1.2.1 Installation unter Linux	2
1.2.2 Installation unter Windows	3
1.2.3 Migrationshinweise	6
1.3 Schreiben, Compilieren und Flashen von Programmen	6
1.4 Flashen der AKSEN-Bibliothek	8
2 Aufbau und Funktionen	11
2.1 Stromversorgung	11
2.2 Anschlüsse	11
2.2.1 Digitale Ein- und Ausgänge	11
2.2.2 Analoge Eingänge	12
2.2.3 LED-Ausgänge und IR-Ausgang	12
2.2.4 Motor-Ports	15
2.2.5 Encoder- und Servo-Ports	15
2.2.6 CAN-Bus	15
2.2.7 Serielle Schnittstelle	16
3 Tutorials	17
3.1 Das erste Programm	17
3.2 Anschlüsse	18
3.2.1 Digitale Eingänge	18
3.2.2 Digitale Ausgänge	18
3.2.3 Analoge Eingänge	19
3.2.4 LED-Ausgänge	20
3.2.5 IR-Ausgang	21
3.2.6 Motor-Ports	24
3.2.7 Servo-Ports	29
3.2.8 Encoder-Ports	31
3.2.9 CAN-Bus	32
3.3 LCD-Display	34

Inhaltsverzeichnis

3.4	Serielle Schnittstelle	35
3.5	Multitasking	36
3.5.1	Multitasking und Infrarot-Detektion	41
3.5.2	Semaphore	44
3.6	Weitere Komponenten	45
3.6.1	DIP-Schalter	45
3.6.2	Zeitsteuerung	45
3.6.3	Versionsabfrage	46
3.7	Verfügbare Speicher	46
3.7.1	Lokale Variablen	46
3.7.2	Globale Variablen	46
3.7.3	Programmgröße	47
3.8	Testprogramm	48
4	Anhang	49
4.1	FAQ	49
4.2	Funktionen der AKSEN-Bibliothek	51
4.2.1	Analoge und Digitale Anschlüsse	51
4.2.2	Infrarot	51
4.2.3	LCD-Display	53
4.2.4	Encoder, Servos und Motoren	54
4.2.5	Multitasking	55
4.2.6	Zeitsteuerung	56
4.2.7	Diverses	56
4.2.8	Erweiterungen	57
4.3	Bauanleitung für Sensoren	58
4.4	Erweiterungen des AKSEN-Boards	59
4.4.1	RCUBE Architektur	59
4.4.2	Bluetooth-Verbindung	60
4.5	CD-Inhalt	61
4.6	Bibliotheksversionen	63
	Literaturverzeichnis	65
4.7	Schaltplan	66

Listings

3.1	helloworld.c	17
3.2	digitalin.c	18
3.3	digitalout.c	18
3.4	analogin.c	19
3.5	led.c	20
3.6	mod_ir_an.c	21
3.7	irsenden.c	22
3.8	irempfang.c	23
3.9	motor.c	24
3.10	stepper.c	27
3.11	servo.c	29
3.12	encoder.c	31
3.13	cantest.c	32
3.14	lcd.c	34
3.15	seriell.c	35
3.16	prozess2.c	37
3.17	mt_ir.c.c	41
3.18	Auszug 1 aus bench_sim.c	44
3.19	Auszug 2 aus bench_sim.c	45

Listings

Einleitung

Herzlichen Glückwunsch zum Erwerb Ihres AKSEN-Boards.

Das AKSEN-Board ist mit vielen Eingängen für Sensoren (digital und analog), digitalen Ausgängen, einem Mikrocontroller der 8051-Familie und einem CAN-Bus-Interface (optional) den gestiegenen Anforderungen der Robotik und der Embedded Systems sowohl in der Lehre als auch im professionellen Einsatz bestens gewachsen.

Die vor Ihnen liegende Anleitung soll Ihnen den Umgang mit dem Board nahebringen. Wir möchten Sie bitten, diese Anleitung gut durchzulesen, da Sie viele Ihrer Fragen beantworten wird. Für weitere Informationen möchten wir Sie auf die offizielle Homepage des AKSEN-Projekts aufmerksam machen. Sie finden Sie unter

<http://ots.fh-brandenburg.de/aksen>

Das AKSEN-Board ist nun auch im Internet erhältlich, den Shop finden Sie unter

<http://www.aksen-roboter.de>

Sollten trotzdem Fragen offenbleiben, so wenden Sie sich bitte an Ihren Händler oder schreiben Sie uns eine eMail unter aksen-support@fh-brandenburg.de. Wir wünschen Ihnen viel Spaß mit dem AKSEN-Board und hoffen, dass Sie lange Freude daran haben werden.

Übersicht über die Dokumentation

Um mit dem AKSEN-Board arbeiten zu können, muss auch der Host-Rechner entsprechend eingerichtet werden. Die Installation des Compilers sdcc und des AKSEN-Flashers beschreibt das Kapitel 1. Ebenfalls in diesem Kapitel wird der grundlegende Umgang mit dem AKSEN-Board beschrieben.

Das 2. Kapitel beschreibt die Funktionalität der Ports des AKSEN-Boards und gibt Hinweise zum Bau von Sensorik und Aktorik.

In den Tutorials (Kapitel 3) wird der Einsatz der AKSEN-Bibliothek beschrieben. Für die Programmierung des AKSEN-Boards werden ausführlich dokumentierte Code-Beispiele gegeben. Im Anhang befindet sich der Schaltplan, eine Liste der Funktionen der AKSEN-Bibliothek und weiterführende Hinweise. In einem eigenen Dokument, der Aufbauanleitung, wird der Bau eines AKSEN-Boards Schritt für Schritt beschrieben.

Inhalt der Verkaufspackung

Die vor Ihnen liegende Verkaufspackung enthält das AKSEN-Board (komplett oder als Baupack), ein Flasher-Kabel, ein Stromanschlusskabel und eine CD mit der Entwicklungsumgebung (Compiler und Flasher), dem Handbuch, der Bauanleitung und Beispielprogrammen

Schemazeichnung des AKSEN-Boards

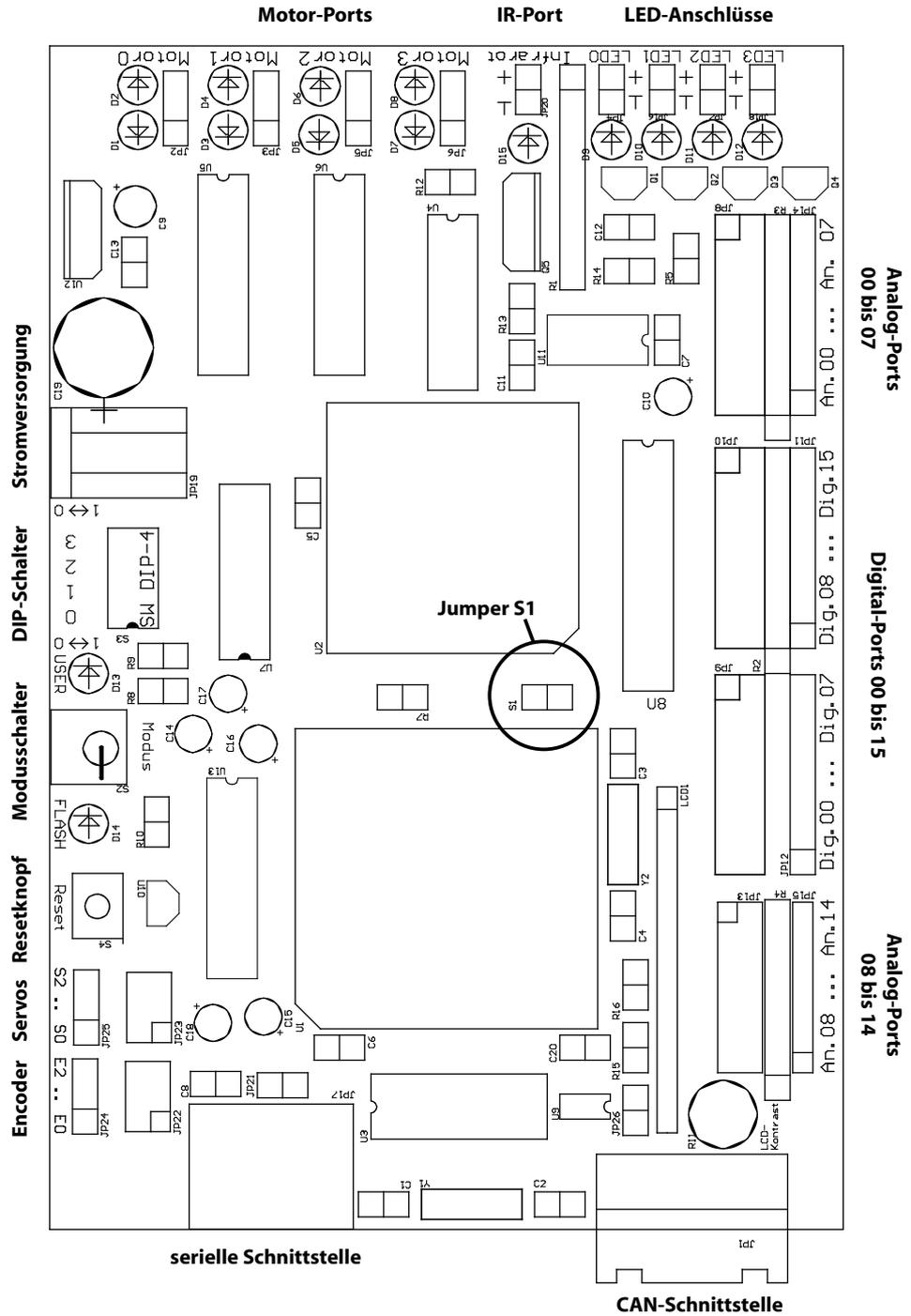


Abbildung 0.1: Schemazeichnung des AKSEN-Boards

1 Erste Schritte

1.1 Inbetriebnahme

Das AKSEN-Board ist praktisch sofort nach Entnahme aus der Verpackung betriebsbereit. Um das Board korrekt einzuschalten, gehen Sie wie folgt vor:

- Prüfen Sie, ob der Modus-Schalter (links neben den Dip-Schaltern) in Richtung User (grüne LED) zeigt. Mehr über diesen Schalter erfahren Sie weiter hinten in diesem Kapitel.
- Versorgen Sie das Board mit Strom. Dabei ist unbedingt auf die richtige Polung zu achten (Masse innen).



Abbildung 1.1: Polung der Stromanschlussbuchse

Die Spannung sollte im Bereich von 5 bis 8 Volt liegen. Der Stromanschluss erfolgt über die Buchse rechts neben den Dip-Schaltern. Lesen Sie dazu bitte auch den Abschnitt über die Stromversorgung in Kapitel 2.

- Mit Einstecken des Stromsteckers ist das Board in Betrieb. Das LCD-Display zeigt eine kleine Begrüßungsbotschaft an und die LEDs auf der rechten Board-Seite blinken für ein paar Sekunden.

Wenn keine Anzeige erscheint bzw. die LEDs nicht blinken, prüfen Sie folgende Dinge:

- Ist die Stromversorgung korrekt (Polung und Spannung)? Der Stromverbrauch sollte bei maximal 170 mA liegen.
- Wenn die Lämpchen leuchten, aber das Display nichts anzeigt, prüfen Sie, ob der Kontrast richtig eingestellt ist. Dazu drehen Sie den weissen Trimmer unterhalb des LCD-Displays auf der linken Seite des Board langsam nach links und rechts. Stellen Sie den richtigen Kontrast ein. Achten Sie auch darauf, dass alle Anschlussbeinchen des LCD-Displays korrekt in ihren Buchsen stecken.
- Wenn das Display schwarze Blöcke oder sonstige ungewöhnliche Zeichen anzeigt, drücken Sie den RESET-Knopf. Er befindet sich links neben dem Modusschalter.

1.2 Installation der Software

Im wesentlichen werden zwei Programme benötigt, um das AKSEN-Board zu programmieren. Das ist zum einen der Compiler, der den Quelltext in für das Board verständlichen Maschinencode übersetzt und zum anderen der AKSEN-Flasher, der diesen Maschinencode auf das Board überträgt.

Wenn Sie eine frühere Version der AKSEN-CD installiert haben, lesen Sie bitte zuvor die Migrationshinweise im Abschnitt 1.2.3.

1.2.1 Installation unter Linux

Der Compiler `sdcc` (Small Device C-Compiler) ist ein Open-Source-Projekt. Es ist unter

`http://sdcc.sourceforge.net`

zu finden.

Für Benutzer von Linux-Distributionen mit rpm-Paketmanagement (SuSE, Red Hat, Mandrake) gibt es `sdcc` auch in Paketform. Debian-Nutzer können das Paket `sdcc` verwenden. Zum gegenwärtigen Zeitpunkt liegt das Debian-Paket nur für `unstable/testing`-Installationen vor.

Um `sdcc` manuell zu installieren, müssen zunächst die gepackten binaries von `http://www.sourceforge.net/projects/sdcc` heruntergeladen werden. Die Links befinden sich weiter unten auf der Projektseite. Für Rechner mit x86-Architektur muss die Datei `sdcc-linux-x86`, für PowerPCs die Datei `sdcc-linux-ppc` ausgewählt werden. Die Sourcen des Compilers finden sich in der Datei `sdcc`.

Für die aktuelle Version 2.4 sind für die weitere Installation folgende Schritte notwendig (bezogen auf die x86-Version). Sie müssen sich spätestens beim Kopieren der Dateien als `root` einloggen, da normale User auf `/usr/local` normalerweise keine Schreibrechte haben.

- Erstellen Sie ein temporäres Verzeichnis und entpacken Sie die heruntergeladene Datei:

```
bzip2 -d <pfadzurdatei>/sdcc-2.4.0-linux-x86.tar.bz2
```

bzw.

```
tar -xvf <pfadzurdatei>/sdcc-2.4.0-linux-x86.tar.gz
```

- Es wurde ein Verzeichnis mit Namen `sdcc` erstellt. Wechseln Sie in dieses Verzeichnis und kopieren Sie den Inhalt nach `/usr/local`:

```
cp -r * /usr/local
```

Ein anderer Installationsort erfordert momentan eine Neuerstellung des Compilers.

- Testen Sie die Installation:

```
/usr/local/bin/sdcc -v
```

Die Versionsnummer des Compilers sollte angezeigt werden.

Oder: Compiler selber kompilieren

Alternativ kann der Compiler auch problemlos selbst kompiliert werden, dazu die Sourcen entpacken und kompilieren:

```
tar -xvf <pfadzurdatei>/sdcc-2.4.0.tar.gz
cd sdcc
./configure && make && make install
```

Der Vorteil des selbstkompilierten sdcc ist die Möglichkeit, ihn mit `make deinstall` zu deinstallieren.

Unter Debian stable muß kompiliert werden, da die Binärpakete die V2.3 der glibc6 verlangen. Um den sdcc auf mehreren x86 Debian-stable-Rechnern zu installieren, kann das auf der CD beiliegende `sdcc-2.4.0-bin-debian-stable.tar.gz` benutzt werden (Auspacken, `cd sdcc`, `make install`)

Wenn für die Programmierung des AKSEN-Boards die AKSEN-Bibliothek genutzt werden soll, dann müssen die entsprechenden Header-Dateien eingebunden werden. Diese befinden sich auf der AKSEN-CD im Verzeichnis `aksen-lib`.

- Kopieren Sie das Verzeichnis `aksen-lib` von der AKSEN-CD nach `/usr/local`
- Um zu testen, ob die Installation funktioniert, wechseln Sie in das Verzeichnis `beispiele` und starten Sie `make` - damit sollten alle Beispiele übersetzt werden. Ergebnis sind *.ihx-Dateien in den Unterverzeichnissen, z.B. `helloworld.ihx` in `helloworld`.

```
boersch@boersch ~
$ cd /usr/local/aksen-lib/beispiele/

boersch@boersch /usr/local/aksen-lib/beispiele
$ make
```

Mehr zur Benutzung der Bibliothek finden Sie im Kapitel 3 sowie weiter hinten in diesem Kapitel.

Der Flasher `aksen` befindet sich auf der AKSEN-CD unter `Flasher/Linux` und kann in ein beliebiges Verzeichnis entpackt werden. Für eine einfache Benutzung sollte dieses Verzeichnis in der `PATH`-Umgebungsvariable enthalten sein.

1.2.2 Installation unter Windows

Der Compiler `sdcc` kann als normales Windows-Programm installiert werden, für mehr Komfort bei der Übersetzung wird vorgeschlagen eine Cygwin-Umgebung zu benutzen. Die Installation besteht aus den folgenden Schritten:

1 Erste Schritte

1. Installation Cygwin
2. Installation sdcc
3. Installation AKSEN-Bibliothek
4. Anpassen das Makefiles

Alle benötigten Dateien sind auf der AKSEN-CD enthalten. Die Pfadangaben beziehen sich im folgenden auf die AKSEN-CD.

Installation Cygwin

Cygwin¹ ist eine Linux-ähnliche Umgebung für Windows, damit werden alle GNU-Programme (z.B. make) unter Windows lauffähig.

- Starten Sie die Datei `setup.exe` aus dem Verzeichnis `\Windows\cygwin`
- Choose a Download Source: Wählen Sie Install from Local Directory
- Select root install directory: Geben Sie ein Verzeichnis² an, in das Cygwin installiert wird, z.B. `C:\cygwin`
- Select Local Package Directory: Woher bekommt Setup die benötigten Pakete; wählen Sie hier das Verzeichnis `\Windows\cygwin` der CD (ist standardmäßig gewählt)
- Select Packages: Zusätzlich zu den Standardpaketen wird das Paket `make` gewählt. Klicken Sie hierzu auf die Kategorie `Devel`, dann das Wort `Skip` in der Zeile des `make`-Paketes, bis das Paket ausgewählt ist (Abb. 1.2)



Abbildung 1.2: Paket-Selektor im Cygwin-Setup

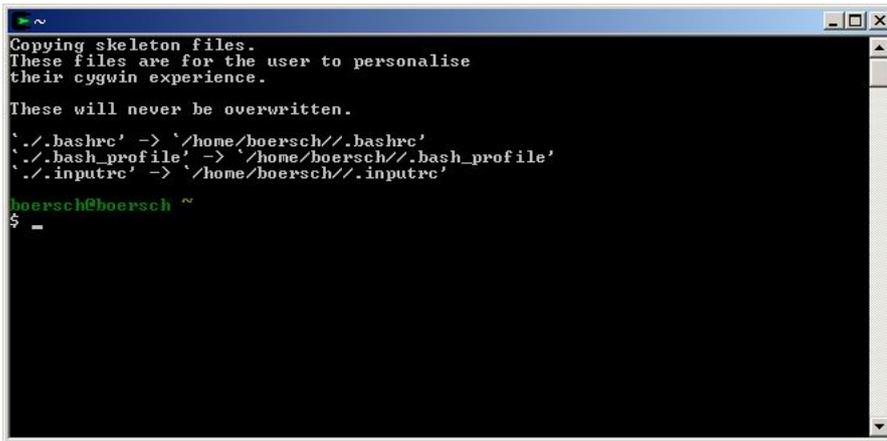
- Create Icons: Legen Sie ein Desktop-Icon an

¹<http://www.cygwin.com/>

²Jedes andere Verzeichnis ist möglich, wir gehen aber in dieser Beschreibung von diesem Pfad aus

- Damit ist Cygwin fertig installiert, das Icon startet eine normale Shell (Bash, Abb. 1.3), das Homverzeichnis befindet sich von Windows aus gesehen unter `C:\cygwin\home\Nutzername`.

Hinweis: Mit dem Cygwin-Installationsprogramm `setup.exe` können jederzeit Pakete aus dem Internet nachgeladen, in die Cygwin-Umgebung installiert oder geupdatet werden. Wenn Sie Cygwin komplett deinstallieren wollen, löschen Sie die angelegten Verzeichnisse und Schlüssel wie in der FAQ³ beschrieben.



```
~
Copying skeleton files.
These files are for the user to personalise
their cygwin experience.

These will never be overwritten.

'./bashrc' -> '/home/boersch//.bashrc'
'./bash_profile' -> '/home/boersch//.bash_profile'
'./inputrc' -> '/home/boersch//.inputrc'

boersch@boersch ~
$ -
```

Abbildung 1.3: Cygwin-Bash, frisch installiert

Installation sdcc

- Die Installationsdatei des Compilers `sdcc` befindet sich als `sdcc-2.4.0-setup.exe` auf der AKSEN-CD im Verzeichnis `Windows\sdcc`. Bei der Installation trägt sich der Compiler in die Pfad-Variable des Systems ein, so dass er direkt in der Cygwin-Shell getestet werden kann (ev. Shell neu starten):

```
$ sdcc -v
SDCC : mcs51/gbz80/z80/avr/ds390/pic16/pic14/TININative/xa51/ds400/hc08 2.4.0
```

Installation AKSEN-Bibliothek

- Jetzt müssen die Header-Dateien von der AKSEN-CD kopiert werden. Sie befinden sich unter `\aksen-lib`. Kopieren Sie das komplette Verzeichnis `\aksen-lib` in den Verzeichnis-Baum des Cygwin nach `C:\cygwin\usr\local\`
- Die letzte fehlende Komponente ist der Flasher. Kopieren Sie ihn in ein beliebiges Verzeichnis auf der Festplatte. Es macht sich gut, einen Desktop-Link auf den Flasher einzurichten, dann ist er jederzeit einfach aufrufbar. Der Flasher befindet sich auf der AKSEN-CD im Verzeichnis `\Windows\Flasher`

³<http://cygwin.com/faq.html>, How do I uninstall all of Cygwin?

1 Erste Schritte

- Um zu testen, ob die Installation funktioniert, wechseln Sie in der Shell in das Verzeichnis `/usr/local/aksen-lib/beispiele` und starten Sie `make` - damit sollten alle Beispiele übersetzt werden. Ergebnis sind *.ihx-Dateien in den Unterverzeichnissen, z.B. `helloworld.ihx` in `helloworld`.

Hinweis: Wenn Sie für Cygwin oder SDCC andere Installationsverzeichnisse gewählt haben, müssen die Pfade `SDCC`, `SDCC_CYG`, `AKSEN`, `AKSEN_CYG` im Konfigurationsfile `/usr/local/aksen-lib/beispiele/config.mak` angepasst werden.

1.2.3 Migrationshinweise

Von CD Version 1.03 nach Version 1.04

Zwischen diesen beiden CD-Versionen hat sich nur die AKSEN-Bibliothek von Version 0.956 auf 0.965 verändert. Die neue Bibliothek ist abwärtskompatibel bis auf die Funktion `servo_D0 ()`, die durch die neue Funktionalität einer Servobank zur Ansteuerung von zusätzlichen 8 Servos ersetzt wurde. Installieren Sie die neue Bibliothek wie üblich unter 1.4 beschrieben.

Von CD Version 1.00 nach Version 1.03

Die wesentlichen Änderungen betreffen den Compiler `sdcc` und die AKSEN-Bibliothek.

Compiler `sdcc`: Im Februar 2004 fand der Release-Wechsel des `sdcc` von 2.3.0 auf 2.4.0 statt. Vor Installation des `sdcc` 2.4.0 sollte die vorherige Installation des `sdcc` 2.3.0 deaktiviert werden:

- unter Windows: Umbenennen oder Löschen des Installationsverzeichnisses vom `sdcc` (meist `C:\SDCC`)
- unter Linux: `sdcc`-Paket deinstallieren oder Verzeichnis `/usr/local/sdcc` löschen

Entfernen der alten AKSEN-Bibliothek:

- Umbenennen oder Löschen von `/usr/local/aksen-lib`

Anpassen vorhandener Projekte: Die Bibliothek ist abwärtskompatibel - vorhandene Projekte sollten sich ohne Probleme mit dem `sdcc` 2.4.0 und der neuen Bibliothek kompilieren lassen, allerdings müssen dazu die `config.mak`-Dateien entsprechend der neuen `config.mak` aus dem Bibliotheksverzeichnis angepasst werden. In der Regel genügt hier einfaches Kopieren.

Die Änderungen sind notwendig, da sich die Verzeichnisstruktur des `sdcc` geändert hat (früher komplett unter `/usr/local/sdcc`, nun die Binaries in `/usr/local/bin` und der Rest in `/usr/local/share/sdcc`) und sich einige Compiler-Optionen verändert haben.

1.3 Schreiben, Compilieren und Flashen von Programmen

Programme für das AKSEN-Board werden in der Sprache C geschrieben. Die Quelltexte werden mit dem oben erwähnten `SDCC`-Compiler in Maschinen-Code umgewandelt und mit dem Flasher auf das AKSEN-Board gebracht. Die Quelltexte können mit jedem beliebigen Editor

1.3 Schreiben, Compilieren und Flashen von Programmen

geschrieben werden. Unter Windows bieten sich hier beispielsweise Textpad oder CONText an. Unter Linux ist KDevelop eine gute Wahl. In beiden Betriebssystemen wird der Compiler über die Kommandozeile aufgerufen. Dazu wird sowohl unter Linux und Windows/Cygwin der make-Mechanismus genutzt. Der Flasher arbeitet unter Linux auch kommandozeilenbasiert, unter Windows steht ein fensterbasiertes Programm zur Verfügung.

Im allgemeinen verläuft die Programmentwicklung so, dass zunächst ein Projektverzeichnis eingerichtet wird und zum Compilieren notwendige Dateien angepasst werden. Danach kann nach Belieben Quellcode erzeugt werden. Widerstehen Sie der Versuchung viele Projekte in ein Verzeichnis zusammenzufassen. Dadurch leidet die Übersichtlichkeit enorm.

Das erwähnte Verzeichnis `/usr/local/aksen-lib/beispiele` kann als Grundlage für ein eigenes Projektverzeichnis mit beliebigem Ziel kopiert und auch umbenannt werden. Es sind also folgende Schritte notwendig um ein AKSEN-Programm zu erstellen:

1. Kopieren Sie sich das Beispiel-Verzeichnis:

```
$ cp -r /usr/local/aksen-lib/beispiele $HOME/meine_Projekte
```

2. Nutzen Sie ein vorhandenes Unterverzeichnis, z.B. `first_project`, als Projektverzeichnis für Ihr erstes Projekt. Die Makefiles der einfachen Beispiele sind identisch und übersetzen die einzelne C-Datei eines Verzeichnisses in die IHX-Datei. Sie können also die vorhandene C-Datei editieren oder durch eine andere ersetzen.

```
boersch@boersch ~/meine_Projekte/first_project
$ cd
```

```
boersch@boersch ~
$ cd meine_Projekte/first_project/
```

```
boersch@boersch ~/meine_Projekte/first_project
$ ls
first.c  makefile
```

- ▷ Verzeichnis des alten sdcc umbenennen
- ▷ SDCC installieren
- ▷ Verzeichnis der alten aksen-lib umbenennen
- ▷ Neue aksen-lib kopieren
- ▷ make im Verzeichnis beispiele
- ▷ config.mak im Projektverzeichnis anpassen
- ▷ make im Projektverzeichnis
- ▷ Bibliothek flashen

Tabelle 1.1: Kurzanleitung Migrationsschritte von CD1.00 zu CD1.03 für Fortgeschrittene

1 Erste Schritte

3. Führen Sie `make` in dem Verzeichnis aus.
4. Sie sollten bei erfolgreichem Erstell-Vorgang (auf Fehlermeldungen achten) eine Datei `first.ihx` in Ihrem Projekt-Verzeichnis vorfinden. In dieser Datei befindet sich nun der für den Download auf das AKSEN-Board vorbereitete Maschinencode.
5. Machen Sie Ihr AKSEN-Board bereit zum Flashen. Dazu stellen Sie bei eingeschaltetem Board den Modus-Schalter auf Flash in Richtung der roten LED. Diese sollte dann aufleuchten. Drücken Sie RESET. Das LCD-Display sollte „Nutzerprogramm flashen“ anzeigen. Verbinden Sie AKSEN-Board und Computer mit dem Flasher-Kabel.
6.
 - **unter Linux:** Rufen Sie den Flasher mit dem Befehl `aksen /dev/ttyS0 test.ihx` auf. Ersetzen Sie gegebenenfalls `/dev/ttyS0` durch die korrekte serielle Schnittstelle, sowie `test.ihx` durch den Dateinamen Ihrer `.ihx`-Datei. Der Flashvorgang unter Linux ist wesentlich langsamer als unter Windows. Grund dafür sind zusätzliche Sicherheitsmechanismen.
 - **unter Windows:** Um die `ihx`-Datei auf das AKSEN-Board zu übertragen, starten Sie den Flasher. Wählen Sie die `ihx`-Datei aus (sowie, wenn nötig die serielle Schnittstelle) und klicken Sie auf Start. Wenn die Datei komplett übertragen ist, wird auf dem LCD-Display des AKSEN-Boards eine Erfolgsmeldung ausgegeben.
7. Wenn das Display des AKSEN-Boards die Meldung „Programmierung beendet“ anzeigt, stellen Sie den Modus-Schalter wieder auf User (grüne LED) und drücken RESET. Ihr Programm sollte jetzt auf dem AKSEN-Board laufen.

Sollte das Flashen unter Linux, gerade bei neuer Hardware, nicht einwandfrei funktionieren, dann hilft ein Kernelupdate auf Version 2.6.0 und höher. Der Flashvorgang wird dadurch auch wesentlich beschleunigt.

1.4 Flashen der AKSEN-Bibliothek

Die AKSEN-Bibliothek wird, wie jedes Programm, erweitert und verbessert. Deshalb gibt es die Möglichkeit den Bibliothekskern neu auf das AKSEN-Board aufzubringen. Da dieser Vorgang sich deutlich von einem normalen Flashvorgang abheben soll, wurde ein Sicherheitsmechanismus entworfen, der es nötig macht einen Jumper zu setzen.

Um die AKSEN-Bibliothek neu zu flashen, gehen Sie wie folgt vor:

1. Entfernen Sie das Display. Es ist mit einer Schraube an der rechten oberen Ecke am Board befestigt. Lösen Sie diese Schraube und ziehen Sie das Display dann vorsichtig ab.
2. Suchen Sie den Jumper S1. Er befindet sich zwischen den beiden großen Schaltkreisen U1 und U2. Schliessen Sie den Jumper.
3. Setzen Sie nun das Display wieder auf, schrauben Sie es aber noch nicht fest.

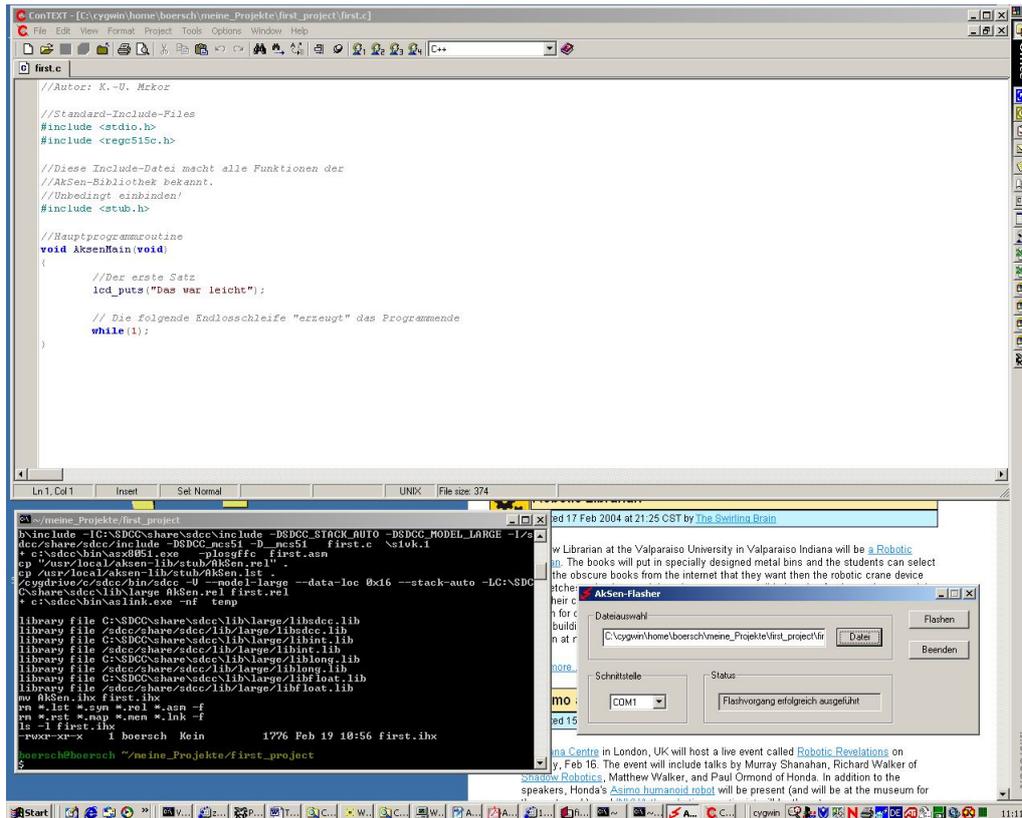


Abbildung 1.4: Typischer Desktop: CONText-Editor, Shell, Flasher

4. Versorgen Sie das Board mit Strom und schalten Sie den Modus-Schalter auf Flash (rote LED). Drücken Sie Reset.
5. Jetzt können Sie die Bibliothek flashen. Dazu gehen Sie genau wie beim Flashen eines Programms vor, nur dass Sie als Datei die Bibliotheksdatei auswählen.
6. Wenn der Flashvorgang erfolgreich beendet wurde, schalten Sie die Stromversorgung ab. Ziehen Sie das Display ab.
7. Öffnen Sie jetzt den Jumper S1 wieder.
8. Setzen Sie das Display auf und schrauben es fest. Achten Sie darauf, die Schraube nicht zu überdrehen. Ausserdem müssen alle Stifte der Anschlussleiste des Displays in der Fassung auf dem Board stecken.

Die aktuelle Version der Bibliothek befindet sich im Verzeichnis `aksen-lib\lib` und trägt den Namen `vector.ihx`.

Zum Einspielen der neuen Bibliothek in das System löschen oder retten Sie das alte Verzeichnis `/usr/local/aksen-lib/` und kopieren das neue Verzeichnis `aksen-lib` dorthin. Ihre Programme müssen dann neu kompiliert werden, um die neuen Header zu nutzen und gegen

1 Erste Schritte

die neue Bibliothek gelinkt zu werden. Die aktuellste Bibliotheksversion (wie auch die aktuelle AKSEN-CD) finden Sie immer auf der AKSEN-Webseite.

2 Aufbau und Funktionen

2.1 Stromversorgung

Das Board wird mit einer Spannung im Bereich von 5 bis 8 Volt betrieben. Die Stromversorgung erfolgt über die Anschlussbuchse rechts neben den DIP-Schaltern. Es ist auf die richtige Polung zu achten. Der Masse-Pol ist innen, die Spannung aussen angeschlossen.

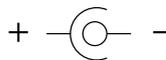


Abbildung 2.1: Polung der Stromanschlussschaltung

Als Spannungsversorgung empfehlen wir:

- NiCd- oder NiMH-Zellen (6 Stück). Diese Zellen sind recht leicht, haben aber nur eine kleine Kapazität.
- 7.2V-Bleiigel-Akkus. Bleiakkus sind schwer und meist klobig, dafür ist ihre Kapazität wesentlich größer.
- LiPoly-Zellen. Diese Zellen wiegen knapp 50g und können dabei bis zu 2Ah Kapazität haben. Die Ladezeit beträgt bei dieser Kapazität 2 Stunden. Leider sind diese Akkus sehr teuer und auch etwas empfindlich bei Tiefentladung.
- stationär. Es können Labornetzteile, PC-Steckkarten oder Steckernetzteile mit der richtigen Spannung verwendet werden.

Die Spannungsversorgung auf dem AKSEN-Board ist zweigeteilt. Die Versorgung für Servos, Motoren und LEDs ist nicht strombegrenzt und nur von der Leitungsbreite auf dem Board abhängig. Die Spannung ist mit der an der Stromversorgungsbuchse identisch. Alle weiteren Komponenten und Ausgänge werden über einen Spannungsregler auf einen Maximalstrom von 1 Ampere begrenzt. Wird der Strom komplett ausgenutzt, dann muss auf ausreichende Kühlung des Spannungsreglers U12 (rechts neben dem Stromanschluss und dem dicken Kondensator) geachtet werden. Ein Kühlkörper ist dann auf jeden Fall notwendig. Die Spannung in diesem Stromkreis beträgt konstant 5V.

2.2 Anschlüsse

2.2.1 Digitale Ein- und Ausgänge

Diese Ports eignen sich zum Anschluss von digitalen Sensoren, wie z.B. Tastern.

2 Aufbau und Funktionen

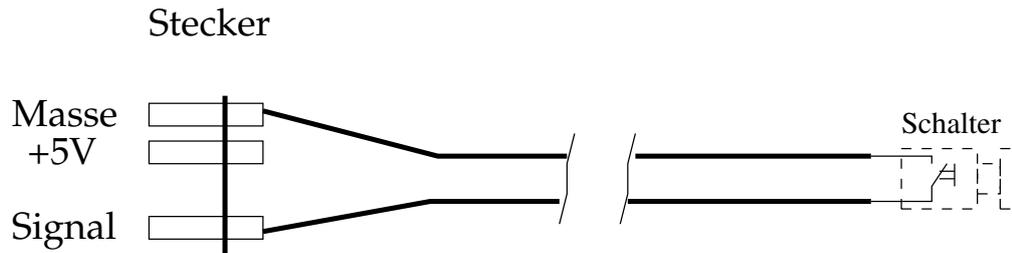


Abbildung 2.2: Anschlussbeispiel für einen Taster

Die digitalen I/O-Ports befinden sich unterhalb des LCD-Displays. Sie sind mit Dig. 00 ... Dig. 07 bzw. Dig. 08 ... Dig. 15 gekennzeichnet. Verwechseln Sie auf keinen Fall den Signal- und den 5V-Anschluss, dies kann zu irreparablen Schäden am Board führen. Die Signalpins sind nicht darauf ausgelegt, hohe Ströme abzugeben. Sie reichen jedoch aus, um eine CMOS-Last zu treiben. Diese Pins sind direkt an die I/O-Ports des Mikrocontrollers angeschlossen. Der Spannungs-Pin (5V) wird direkt von der Board-Spannung gespeist.

2.2.2 Analoge Eingänge

Die analogen Ports messen die am Port anliegende Spannung und stellen den Spannungswert als Zahl zwischen 0 (=0V) und 255 (=5V). Sie sind daher für analoge Sensoren, wie IR-Empfänger-Dioden, Photosensoren oder Optokoppler, bestens geeignet. Die Beschaltung ist die gleiche, wie bei den Digital-Ports.

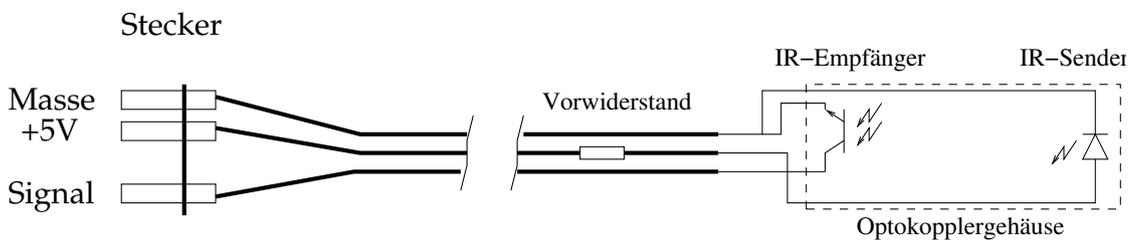


Abbildung 2.3: Anschlussbeispiel für einen Optokoppler

Die Digitalisierung der Spannungswerte übernimmt der Mikrocontroller. Der Signalpin ist über einen $47\text{k}\Omega$ Widerstand mit der Versorgungsspannung verbunden (Pull-Up-Widerstand).

2.2.3 LED-Ausgänge und IR-Ausgang

Bei der Nutzung von LEDs ist zum Einen auf die richtige Polung der LED zu achten. Zum zweiten muss ein Vorwiderstand benutzt werden.

Durch die Form des Steckers lässt sich eine LED sowohl an den LED-Ports und am IR-Port

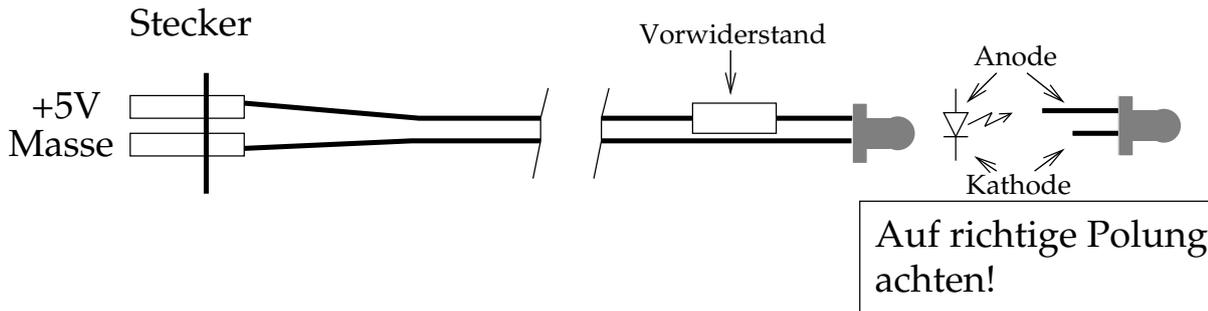


Abbildung 2.4: Aufbau und Anschluss einer LED

als auch an einem digitalen oder analogen Port anschliessen. Der Vorwiderstand R_{vor} berechnet sich aus dem Stromverbrauch der Diode I_D und der Spannung, die sie benötigt U_D , sowie der Versorgungsspannung des Boards $V_{cc} = 5V$:

$$R_{vor} = (V_{cc} - U_D) / I_D \quad (2.1)$$

Die für die Gleichung benötigten Parameter der Diode finden Sie im Datenblatt zu der Diode. In einigen Katalogen sind die Werte gleich mit angegeben.

Der IR-Ausgang gibt ein Signal mit einer Frequenz von 40KHz aus. Diese kann programmseitig noch mit einer niedrigeren Frequenz (z.B. 100Hz) moduliert werden. Mit im Handel erhältlichen Infrarot-Fernsteuerungs-Sensoren, die Signale mit dieser Frequenz detektieren können, ist es möglich, dieses Signal wieder zu empfangen. Hier hat sich zum Beispiel der IS1U60 von Sharp (erhältlich bei <http://www.rs-components.de> für ca. 4 Euro) bewährt.

Funktionsweise des Sensors: der Sensor erzeugt am Ausgang LOW, wenn eine Frequenz um 40

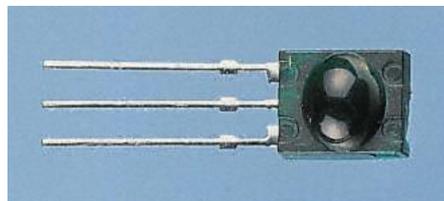


Abbildung 2.5: IR-Fernsteuerungs-Empfänger von Sharp

kHz detektiert wird, ansonsten HIGH. Allerdings fällt das HIGH-Signal nach ca. 30 ms wieder auf LOW. Es ist also sinnvoll, die 40 KHz mit einer Frequenz zu modulieren, deren Periode kleiner als 60 ms ist, da sich sonst das Taktverhältnis verschiebt.

Durch die Modulation ist das Signal unabhängig von den herrschenden Lichtverhältnissen und trägt in der Regel über mehrere Meter (abhängig von der Sendeleistung wurden bis 6 m erzielt). Es kann so beispielsweise als Lichtschranke oder Abstandssensor verwendet werden. Zum Empfang der doppelt modulierten IR-Strahlung sind spezielle Funktionen verfügbar, die an den digitalen Eingängen 04 bis 07 versuchen, ein Signal einer bestimmten Frequenz mit einer PLL zu detektieren. Dazu beobachtet die PLL-Routine die Flanken am entsprechenden Digital-Port:

2 Aufbau und Funktionen

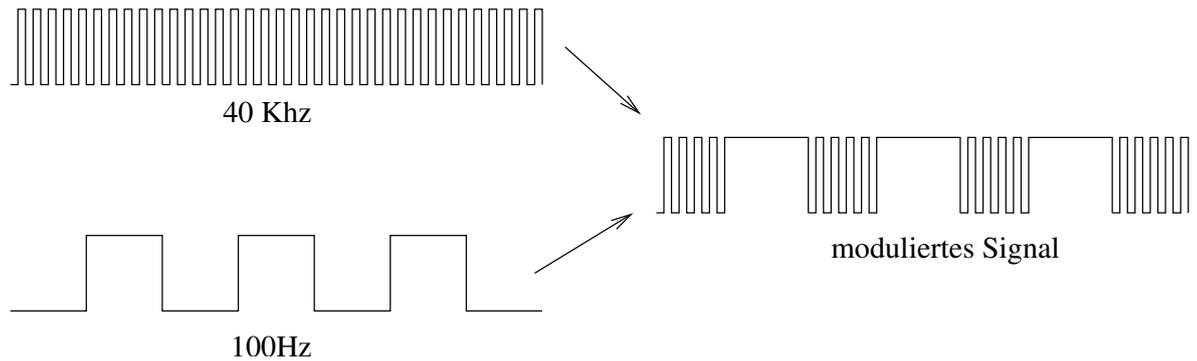


Abbildung 2.6: Modulation des 40KHz-Signales mit einem 100Hz-Signal

wenn diese in den richtigen Abständen (z.B. nach 5ms für 100Hz und nach 4ms für 125Hz) erfolgen, dann gilt eine Periode als richtig erkannt und ein Zähler wird inkrementiert. Dieser Zähler kann vom Programm abgefragt werden.

Da die Sender und Empfänger eine unsystematische Drift aufweisen, ist es möglich, dass eine Flanke vereinzelt wenige Millisekunden zu früh oder zu spät ankommt. Zur Kompensation kann daher ein maximaler Fehler in Millisekunden angegeben werden. Sollte sich eine Flanke um mehr als den maximalen Fehler verschieben, dann wird der Zähler auf Null zurückgesetzt.

Es werden die Frequenzen **100 Hz und 125 Hz** empfohlen, aber auch folgende Sende- und Empfangsfrequenzen sind möglich:

Periode [ms]	Param für ir_mod_senden	Frequenz [Hz]	max. Fehler
2	1	500,00	1
4	2	250,00	1
6	3	166,67	2
8	4	125,00	2
10	5	100,00	3
12	6	83,33	3
14	7	71,43	3
16	8	62,50	4
18	9	55,56	4
20	10	50,00	5
22	11	45,45	5
24	12	41,67	5
26	13	38,46	6
28	14	35,71	6
30	15	33,33	7
32	16	31,25	7
34	17	29,41	7
36	18	27,78	8
38	19	26,32	8
40	20	25,00	9
42	21	23,81	9
44	22	22,73	9
46	23	21,74	10
48	24	20,83	10
50	25	20,00	11

Zu beachten sei hier, dass es nicht möglich ist, zwei sich überlagernde 40kHz-Signale gleich-

zeitig zu empfangen, egal ob sie mit 100 oder 125 Hz moduliert sind. Ein weiterer Aspekt ist die CPU-Last, die durch die Sende- und Detektor-Routinen erzeugt wird. Der Sender verbraucht ca. 5%, die Empfänger jeweils 6%.

2.2.4 Motor-Ports

Die Motor-Ports sind für Anwendungen mit hohen Lasten geeignet. Pro Port kann maximal ein Ampere Strom bereitgestellt werden. Über die Funktionen der AKSEN-Bibliothek lassen sich die Drehrichtung und die Geschwindigkeit des Motors (in zehn Stufen) ändern.

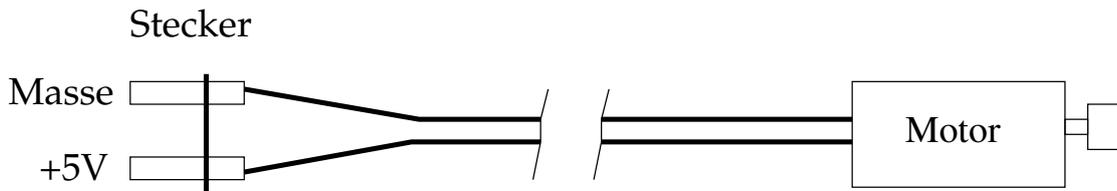


Abbildung 2.7: Anschlussbeispiel - Motor

2.2.5 Encoder- und Servo-Ports

Die Pinbelegung der Encoder- und Servo-Ports ist identisch mit den analogen und digitalen Anschlüssen. Die Signal-Pins der Servo-Ports geben ein pulswertenmoduliertes Signal ab, mit dem die Stellung eines Servos gesteuert werden kann. Die Pulsweite kann über die Servo-Befehle der AKSEN-Bibliothek eingestellt werden.

Die Encoder-Ports messen digitale Übergänge von High nach Low. Die Flanken werden gezählt und können über die AKSEN-Bibliothek eingelesen werden.

2.2.6 CAN-Bus

Mit dem CAN-Bus kann eine Kommunikationsmöglichkeit zwischen AKSEN-Boards oder anderer Hardware hergestellt werden. Die Möglichkeiten sind dabei enorm: es kann mit Geschwindigkeiten von bis zu einem Megabit gesendet werden, die Kommunikation ist paketorientiert. Das AKSEN-Board bietet darüber hinaus die Möglichkeit, Slave-Geräte über den CAN-Bus

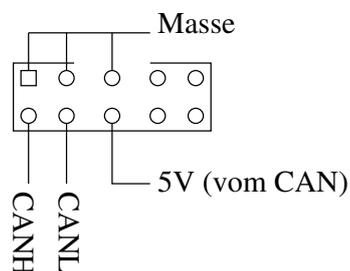


Abbildung 2.8: Pinbelegung am CAN-Bus

2 Aufbau und Funktionen

mit Strom zu versorgen, bzw. selbst als Slave zu fungieren. Dies kann über die Jumper JP26 bzw. JP21 erreicht werden. Durch Stecken von JP21 wird die Stromversorgung des Boards ($V_{CC2} = 5V$) an die der CAN-Komponenten angeschlossen. Ist darüber hinaus auch noch JP26 gesteckt, liegt diese Spannung auch an dem CAN-Stecker an und kann so zur Versorgung externer Schaltungen herangezogen werden.

Ebenfalls denkbar ist die Kombination JP21=offen und JP26=gesteckt. Diese Kombination ermöglicht die Versorgung der CAN-Komponenten des AkSen-Boards durch eine externe über den CAN-Stecker zugeführte Spannung. Dadurch könnte der Spannungsregler des AkSen-Boards entlastet werden.

Die Eingangs erwähnte Kombination JP21=gesetzt und JP26=gesetzt darf nur gewählt werden, wenn kein anderes Board seine Spannung auf dem CAN-Stecker bereitstellt! Dies läßt sich durch die Kombination JP21=gesetzt und JP26=offen (Auslieferungszustand des AKSEN-Boards) vermeiden.

2.2.7 Serielle Schnittstelle

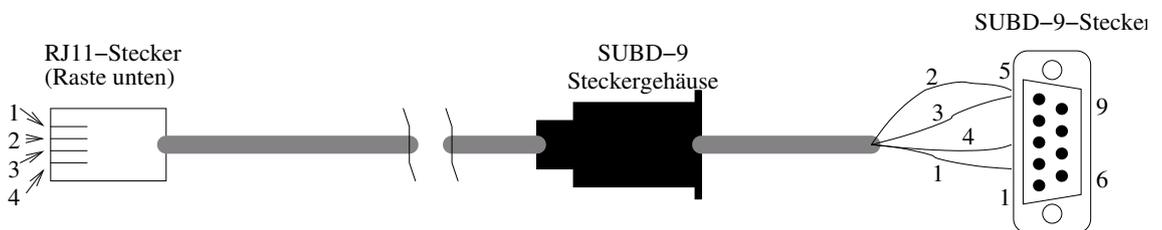


Abbildung 2.9: Aufbau des Flasherkabels

Das Verbindungskabel vom AKSEN-Board zum Computer ist ein normales vieradriges Kabel mit einem RJ11-Stecker an dem einen und einem 9-poligen SUB-D-Stecker an dem anderen Ende. Die obige Grafik verdeutlicht die Anschlussbelegung. Jeweils ein Draht wird an die Pins 2 und 3 und zwei Drähte werden an Pin 5 des SUB-D-Steckers angelötet.

Die serielle Schnittstelle des AKSEN-Boards läuft mit einer Geschwindigkeit von 9600 Baud. Es werden 8 Bit breite Datenworte ohne Parität und mit einem Stopbit benutzt. Die AKSEN-Bibliothek enthält Funktionen, die eine Kommunikation über die serielle Schnittstelle erlaubt.

3 Tutorials

3.1 Das erste Programm

Das AKSEN-Board wird in C programmiert. Es stehen die normalen Standardbibliotheken zur Verfügung. Zu einem normalen, für DOS, Linux oder Windows bestimmten, Programm gibt es jedoch einige Unterschiede. Die wichtigsten Punkte betreffen dabei die Hauptprogrammroutine. In normalen Programmen `main` genannt, heisst Sie hier `AksenMain` und hat keine Parameter. Es wird auch kein Rückgabewert gefordert. Die Programmausführung beginnt bei der ersten Zeile dieser Routine und endet mit der letzten Zeile.

Wenn das Ende der Funktion erreicht wird, dann gibt das AKSEN-Board eine Meldung aus „- AKSENMain - wurde verlassen“. Wenn diese Meldung nicht gewünscht wird, kann am Ende des Programms eine Endlosschleife platziert werden. Im folgenden ist ein kleines, aber berühmtes Programmbeispiel aufgeführt. Sämtliche Beispiele finden Sie auf der AKSEN-CD im Verzeichnis `aksen-lib\beispiele`.

Listing 3.1: helloworld.c

```
1 //Autor: K.-U. Mrkor
2
3 //Standard-Include-Files
4 #include <stdio.h>
5 #include <regc515c.h>
6
7 //Diese Include-Datei macht alle Funktionen der
8 //AkSen-Bibliothek bekannt.
9 //Unbedingt einbinden!
10 #include <stub.h>
11
12 //Hauptprogrammroutine
13 void AksenMain(void)
14 {
15     //Der erste Satz
16     lcd_puts("Hallo_Welt");
17
18     // Die folgende Endlosschleife "erzeugt" das Programmende
19     while(1);
20 }
```

Wie schon an den Quelltextkommentaren ersichtlich, ist die Include-Datei `stub.h` sehr wichtig. Sie stellt die Schnittstelle zur AKSEN-Bibliothek dar. Die von dieser Bibliothek bereitgestellten Funktionen werden im Folgenden in Tutorials vorgestellt. Eine komplette Übersicht finden Sie im Anhang.

3.2 Anschlüsse

3.2.1 Digitale Eingänge

Die digitalen Eingänge liefern 0 oder 1 als Ergebnis. Die Funktionen zum Einlesen eines Wertes lautet `unsigned char digital_in(unsigned char port)`. Zum byteweisen Einlesen von 8 Portpins kann die Funktion

```
unsigned char digital_bytein(unsigned char port)
```

benutzt werden, `port` ist hierbei die Konstante `PORT_1` oder `PORT_2`.

Listing 3.2: digitalin.c

```
1 #include <stub.h>
2
3 //Einlesen eines Wertes von Digital-Port 0
4 //und Anzeige auf dem LCD
5
6 void AksenMain(void)
7 {
8     unsigned char pin=0;
9     unsigned char wert=0;
10
11     while(1)
12     {
13         //Wert einlesen
14         wert=digital_in(0);
15
16         // Cursor auf linke, obere Ecke setzen
17         lcd_setxy(0,0);
18
19         //Wert ausgeben
20         lcd_ubyte(wert);
21     }
22 }
```

3.2.2 Digitale Ausgänge

Über die digitalen Ausgänge können externe Geräte geschaltet werden. Hier kommt der Befehl `void digital_out(unsigned char port)` zum Einsatz. Zur byteweisen Ausgabe von 8 Portpins kann die Funktion

```
void digital_byteout(unsigned char port, unsigned char wert)
```

benutzt werden, `port` ist hierbei die Konstante `PORT_1` oder `PORT_2`.

Listing 3.3: digitalout.c

```
1 #include <stub.h>
```

```

2
3 //Erzeugen einer Rechteck-Schwungung
4 //auf Digital-Port 0
5
6 void AksenMain(void)
7 {
8     unsigned char pin=0;
9     unsigned char wert=1;
10
11     while(1)
12     {
13         //zwischen 1 und 0 umschalten
14         if (wert==1) wert=0; else wert=1;
15
16         //Wert setzen
17         digital_out(pin,wert);
18     }
19 }

```

3.2.3 Analoge Eingänge

Die analogen Eingänge messen und digitalisieren die anliegende Spannung. Der Wertebereich reicht dabei von 0 (0V) bis 255 (5V). Die Abfrage eines analogen Ports erfolgt mit dem Befehl `unsigned char analog(unsigned char port)`.

Listing 3.4: analogin.c

```

1 #include <stub.h>
2
3 //Einlesen eines Wertes von Analog-Port 0
4 //und Anzeige auf dem LCD
5
6 void AksenMain(void)
7 {
8     unsigned char pin=0;
9     unsigned char wert=0;
10
11     while(1)
12     {
13         //Wert einlesen
14         wert=analog(0);
15
16         // Cursor auf linke, obere Ecke setzen
17         lcd_setxy(0,0);
18
19         //Wert ausgeben
20         lcd_ubyte(wert);
21     }
22 }

```

3.2.4 LED-Ausgänge

Die LED-Ausgänge können ähnlich wie die digitalen Ausgänge angesprochen werden. Eine interessante Anwendungsmöglichkeit besteht in der Implementation einer Lichtschranke. So kann mit einem IR-Empfänger an einem analogen Port das Signal einer an einem LED-Port angeschlossenen Infrarot-LED empfangen werden. Das Ergebnis ist jedoch stark abhängig von der Umgebungshelligkeit.

Eine Lösung stellt ein Differenzverfahren da. Zunächst wird die Helligkeit bei abgeschaltetem IR-Sender gemessen, danach bei eingeschaltetem Sender. Die Differenz dieser beiden Werte ergibt die Lichtmenge, die der Empfänger von der Sende-Diode empfangen hat.

Werden Sender und Empfänger so angeordnet, dass sie in eine Richtung zeigen, können in einem Bereich zwischen 3 cm und 40 cm (abhängig von Form, Farbe und Beschaffenheit der angestrahlten Oberfläche) Entfernungen gemessen werden. Der Empfänger darf dabei natürlich den Sender nicht direkt sehen und muss daher abgeschirmt werden.

Listing 3.5: led.c

```
1 //Autor: Ronny Menzel
2 //Lauflicht mit den LED-Ausgaengen
3
4 #include <stdio.h>
5 #include <stub.h>
6
7 void AksenMain(void)
8 {
9
10     // Laufvariable
11     char lampe;
12
13     // Zeitverzoeigerung des Lauflichts
14     unsigned char time=200;
15
16
17
18     // Endlosschleife fuer das Programm
19     do{
20
21         // Vorwaertsschleife (LED hochzaehlen)
22         for (lampe=0; lampe<=3; lampe++)
23         {
24             //LED einschalten
25             led(lampe,1);
26
27             sleep(time);
28
29             // LED ausschalten
30             led(lampe,0);
31         }
32         // nachdem die 3. Led angesprochen wurde,
```

```

33     // wieder runterzaehlen
34
35     // Rueckwaertsschleife
36     for (lampe=3; lampe>0; lampe--)
37     {
38         led(lampe,1);
39         sleep(time);
40         led(lampe,0);
41     }
42
43     }while(1);
44 }

```

3.2.5 IR-Ausgang

Der IR-Ausgang kann in zwei sich ausschließenden Modi betrieben werden:

- Einzelschaltung, z.B. für die Generierung von RC5-Code
- Modulation mit Rechteck-Signal, z.B. für Lokalisation

Einzelschaltung

Der modulierte IR-Ausgang kann manuell ein- und ausgeschaltet werden, dazu dienen die Makros `mod_ir_an()` und `mod_ir_aus()`:

Listing 3.6: `mod_ir_an.c`

```

1  /*****
2  /* "Manuelles"
3  /* Ein-/Auschalten des Infrarot-Senders durch
4  /* mod_ir_an() und mod_ir_aus()
5  /*
6  /* K.-U. Mrkor
7  /*
8  /*****
9  #include <stdio.h>
10 #include <regc515c.h>
11 #include "stub.h"
12 void AksenMain(void)
13 {
14     do
15     {
16         lcd_cls();
17         lcd_puts("mod_ir_an()");
18         lcd_setxy(1,0);
19         lcd_puts("Modul._IR_an");
20         mod_ir_an();
21         sleep(2000);
22

```

3 Tutorials

```
23         lcd_cls();
24         lcd_puts("mod_ir_aus()");
25         lcd_setxy(1,0);
26         lcd_puts("Modul._IR_aus");
27         mod_ir_aus();
28         sleep(2000);
29     }
30     while(1);
31 }
```

Modulation mit Rechteck-Signal

Das in Abschnitt 2.2.3 erwähnte Modulieren des 40-kHz-Signals mit einer Rechteckschwingung leistet die Funktion `setze_ir_senden(takt)`. Der Anwender übergibt dazu mit `takt` den halben Wert der Periodendauer des gewünschten Signals (Angabe in ms). Ein Wert ungleich 0 startet den automatischen Vorgang.

Listing 3.7: irsenden.c

```
1  #include <stdio.h>
2  #include <regc515c.h>
3  #include "stub.h"
4
5  void AksenMain(void)
6  {
7      lcd_puts("_Infrarot-Test_");
8      lcd_setxy(1,0);
9      lcd_puts("_125_Hz_");
10
11     /******
12     /* Wert fuer setze_ir_senden() ergibt sich aus halber */
13     /* Periodendauer der gewuenschten Frequenz in ms      */
14     /*                                                     */
15     /* mod_ir_senden | Frequenz                          */
16     /* ----- */
17     /* 0           | deaktiviert                          */
18     /* 4           | 125 Hz                               */
19     /* 5           | 100 Hz                               */
20     /******
21
22     setze_ir_senden(4);
23     while(1);
24 }
```

Das Beispiel `irempfang` zeigt die automatische Frequenzerkennung. Das AKSEN-Board kann 4 Empfänger gleichzeitig bedienen, wobei die 4 Empfänger jeweils nach einer eigenen Frequenz suchen können. Die Fernsteuerungsempfänger werden dazu an die Digital-Ports 04 bis 07 angeschlossen. Mit dem Makro `mod_irX_takt()` wird die zu erkennende Frequenz (analog

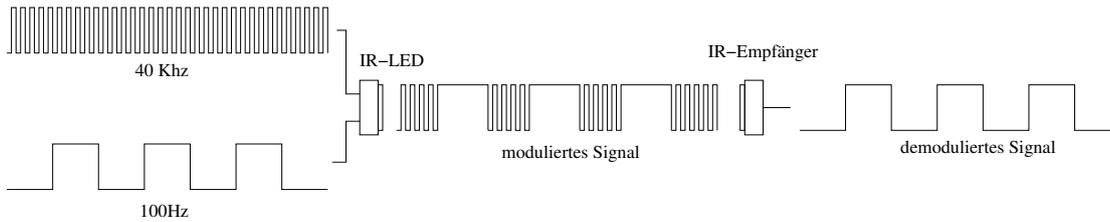


Abbildung 3.1: Senden und Empfang von modulierten IR-Signalen

zu `ir_senden()` eingestellt. Das große X steht dabei für die Nummer des IR-Empfängers:

Befehl	Empfänger an Port
<code>mod_ir0_takt(takt)</code>	04
<code>mod_ir1_takt(takt)</code>	05
<code>mod_ir2_takt(takt)</code>	06
<code>mod_ir3_takt(takt)</code>	07

In `mod_irX_maxfehler()` kann der maximale Fehler pro Periode übergeben werden (siehe auch Seite 13). Zur Abfrage der Empfangsqualität dient das Makro `mod_irX_status()`, es übergibt die Anzahl der letzten zusammenhängend erkannten Perioden. Wird auch nur eine einzige nicht erkannt, wird der entsprechende Zähler sofort gelöscht.

Grundlage der Erkennung ist ein permanenter Soll-Ist-Vergleich. Dazu ein Beispiel: Bei `takt=5` sind es insgesamt 10 Zustände pro Periode. Nach einer Anfangssynchronisierung (fallende Flanke=Zustand 0) wird nun jeder Zustand überprüft (0 oder 1). Abweichungen vom erwarteten Wert werden aufsummiert. Ist die Summe am Ende der Periode größer als der vereinbarte Maximalwert, wird der Status zurückgesetzt. Anderenfalls (und wenn er kleiner 255 ist) incrementiert.

Hinweis: Die Infrarot-Detektion verträgt sich nicht ohne weiteres mit dem Multitasking. Eine elegante Lösung finden Sie im Abschnitt 3.5.1.

Listing 3.8: irempfang.c

```

1  /*****
2  /*
3  /* Erkennen einer auf den 40kHz-Träger mod.
4  /* Frequenz
5  /* 19.01.2004
6  /* K.-U. Mrkor
7  /*
8  /*****
9  #include <stdio.h>
10 #include <regc515c.h>
11 #include "stub.h"
12
13 void AksenMain(void)
14 {

```

3 Tutorials

```
15  /*****
16  /* Zuordnung der Empfangseingaenge: */
17  /*
18  /* D04 -> mod_ir0_empf
19  /* D05 -> mod_ir1_empf
20  /* D06 -> mod_ir2_empf
21  /* D07 -> mod_ir3_empf
22  /*
23  *****/
24
25  mod_ir0_takt(4);          /* 125 Hz */
26  mod_ir0_maxfehler(3);   /* maximal 3 Fehler je Periode */
27
28  do
29  {
30      lcd_cls();
31      lcd_setxy(0,0);
32      lcd_puts("IR-Empfang_\_IR0");
33
34      lcd_setxy(1,0);
35      lcd_puts("125Hz_\->");
36      lcd_ubyte( mod_ir0_status() );
37      sleep(200);
38
39  }
40  while(1);
41 }
```

3.2.6 Motor-Ports

DC-Motoren

Die vier Motor-Ports werden über zwei Befehle gesteuert. Die Drehrichtung wird über den Befehl `motor_richtung(unsigned char motor, unsigned char richtung)` eingestellt. Wird als Richtung eine 0 übergeben läuft der Motor vorwärts, eine 1 lässt ihn in die andere Richtung laufen.

Mit `motor_pwm(unsigned char motor, unsigned char power)` kann die Geschwindigkeit des Motors eingestellt werden. Hier sind für Power die Werte 0 (aus) bis 10 (volle Geschwindigkeit) zulässig. Die tatsächliche Drehgeschwindigkeit des Motors in den einzelnen Geschwindigkeitsstufen hängt jedoch stark von dessen Konstruktion ab. So ist es möglich, dass die bei einer niedrigen Stufe am Motor anliegende Spannung zu niedrig ist, um ihn überhaupt zu bewegen.

In beiden Befehlen muss der Motor angegeben werden, der angesprochen werden soll. Dabei ist der Port ganz aussen auf dem Board dem Wert 0 zugeordnet, der Port ganz innen hat die Nummer 3.

Listing 3.9: motor.c

```
1 //Autor: Ronny Menzel
```

```

2 #include <stdio.h>
3 #include <stub.h>
4
5 //Das Programm faehrt zunaechst alle vier Motoren
6 //nacheinander langsam an. Danach werden Sie
7 //verlangsamt und steuern in die andere Richtung
8
9 void AksenMain(void)
10 {
11     unsigned char mot, power, richtung=0;
12
13     // Endlosschleife fuer das Programm
14     while(1)
15     {
16         //Richtung auswaehlen
17         for (richtung=0;richtung<=1;richtung++)
18         {
19             lcd_int(richtung);
20             // Motor auswaehlen
21             for (mot=0; mot<=3; mot++)
22             {
23                 //Geschwindigkeit auswaehlen
24                 for (power=0; power <= 10; power++)
25                 {
26                     //Drehrichtung fuer Motor mot setzen
27                     motor_richtung(mot, richtung);
28                     //Geschwindigkeit fuer Motor mot setzen
29                     motor_pwm(mot, power);
30
31                     sleep(200);
32                 }
33                 for (power=10; power > 0; power--)
34                 {
35                     //Drehrichtung fuer Motor mot setzen
36                     motor_richtung(mot, richtung);
37                     //Geschwindigkeit fuer Motor mot setzen
38                     motor_pwm(mot, power);
39
40                     sleep(200);
41                 }
42                 motor_pwm(mot,0);
43             }
44         }
45     }
46 }

```

Schrittmotoren

Ein Schrittmotor ist ein Elektromotor, der sich präzise um eine bestimmte Gradzahl mit x Schritten drehen kann. Man unterscheidet bei den Schrittmotoren zwischen Unipolaren und Bipolaren, welche man jeweils im Voll- bzw. Halbschrittbetrieb laufen lassen kann. Mit dem AKSEN-Board kann man einen bipolaren Schrittmotor mit Hilfe zweier Motorports ansteuern.

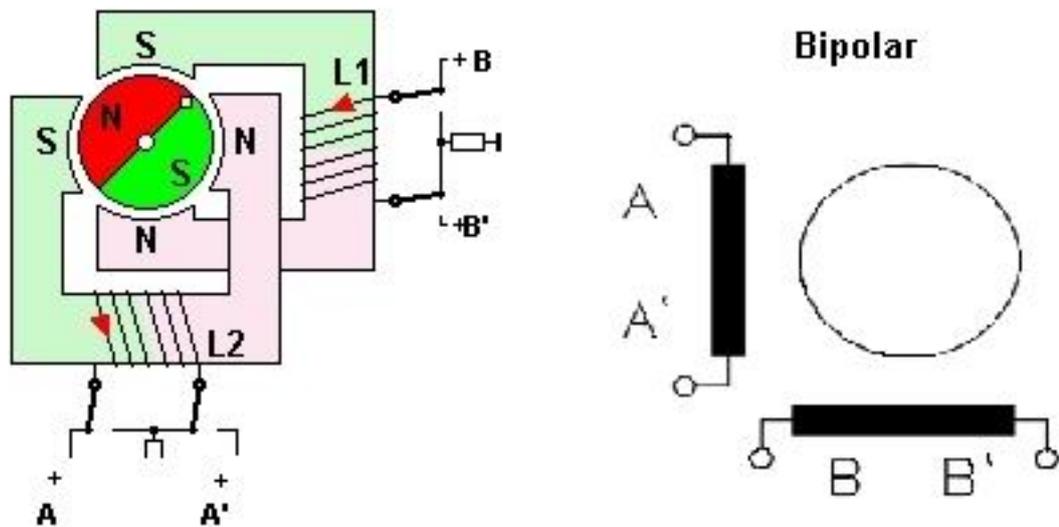


Abbildung 3.2: Bipolarer Schrittmotor

Vereinfacht besteht der Schrittmotor aus mehreren Elektromagneten und einem Dauermagneten (Rotor). Ein bipolarer Schrittmotor besitzt 4 Anschlusskabel, wobei 2 jeweils zusammengehören (A und A' sowie B und B'). Die zusammengehörigen Anschlüsse können mithilfe eines Widerstands-Messgerätes gefunden werden, da sie im Inneren des Motors miteinander verbunden sind (siehe Abbildung 3.2).

Es gibt auch teure Schrittmotore mit 8 Anschlüssen, bei denen durch externe Verschaltung bipolare oder unipolare Motore gebildet werden können. Unipolare Schrittmotore haben i.R. 6 Anschlüsse. Eine gute Quelle für Schrittmotore sind ausrangierte elektromechanische Geräte z.B. Diskettenlaufwerke. So ist in einem 3,5“-Laufwerk ein kleiner bipolarer Schrittmotor mit 18°-Schritten zur Kopfpositionierung verbaut.

Die miteinander verbundenen Anschlüsse bilden je ein Motorportanschluss, so dass die Spulen über Motorkommandos angesteuert werden können. Hierbei ist der meist hohe Stromfluss bei Schrittmotoren zu beachten. Zunächst sollte man beide Motorports aktivieren bzw. einschalten (z.B. `motor_pwm(0,10);` und `motor_pwm(1,10);`) Zum Schreiten im Vollschrittbetrieb sind die folgenden 4 Zustände von der Schrittmotorsteuerung zu durchlaufen:

Zustand	Motor1	Motor2
1	Vorwärts	Vorwärts
2	Vorwärts	Rückwärts
3	Rückwärts	Vorwärts
4	Rückwärts	Vorwärts

Jeder dieser Zustände bewirkt eine bestimmte Stellung des Rotors. Die Zustände können in beide Richtungen durchlaufen werden und erzeugen damit eine Rechts- bzw. Linksdrehung.

Ein Schrittmotor hat eine untere Grenzfrequenz (Anlauffrequenz), so dass zwischen zwei Schritten eine Pause von 2-10 Millisekunden (z.B. mit `sleep()`) eingefügt werden muss. Ohne diese Pause funktioniert der Schrittmotor evtl. nicht korrekt. Der nachfolgende Quelltext zeigt die Ansteuerung eines Schrittmotors.

Listing 3.10: stepper.c

```

1 #include <stdio.h>
2 #include <stub.h>
3 #include <regc515c.h>
4
5 short int zust = 0;
6
7 void smotor_vw()
8 {
9     motor_pwm(0,10);           // den Strom für
10     beide Motoren einschalten
11     motor_pwm(1,10);
12     switch(zust)
13     {
14         case 3:
15             motor_richtung(0,0);
16             motor_richtung(1,0);
17             break;
18         case 0:
19             motor_richtung(0,0);
20             motor_richtung(1,1);
21             break;
22         case 1:
23             motor_richtung(0,1);
24             motor_richtung(1,1);
25             break;
26         case 2:
27             motor_richtung(0,1);
28             motor_richtung(1,0);
29             break;
30     }
31     zust++;

```

3 Tutorials

```
32     zust %= 4;
33     sleep(2);
34     motor_pwm(0,0); // den Strom für
        beide Motoren einschalten
35     motor_pwm(1,0);
36 }
37
38 void smotor_rw()
39 {
40     motor_pwm(0,10); // den Strom für
        beide Motoren einschalten
41     motor_pwm(1,10); //
42
43     switch(zust)
44     {
45         case 3:
46             motor_richtung(0,1);
47             motor_richtung(1,1);
48             break;
49         case 0:
50             motor_richtung(0,1);
51             motor_richtung(1,0);
52             break;
53         case 1:
54             motor_richtung(0,0);
55             motor_richtung(1,0);
56             break;
57         case 2:
58             motor_richtung(0,0);
59             motor_richtung(1,1);
60             break;
61     }
62
63     zust--;
64     zust = (zust+4)%4;
65
66     sleep(2);
67
68     motor_pwm(0,0); // den Strom für
        beide Motoren einschalten
69     motor_pwm(1,0);
70
71 }
72
73 void smotor(int schritte)
74 {
75     if(schritte > 0)
76     {
77         for(;schritte > 0;schritte--)
```

```

78     {
79         smotor_vw();
80     }
81 }
82 else if(schritte < 0)
83 {
84     for(;schritte < 0;schritte++)
85     {
86         smotor_rw();
87     }
88 }
89 }
90 void AksenMain(void)
91 {
92 smotor(/*hier die anzahl der Schritte als ganze Zahl eingeben*/);
93 }

```

3.2.7 Servo-Ports

Das AKSEN-Board besitzt Anschlüsse für drei Servos sowie eine zuschaltbare Servobank mit weiteren 8 Servoausgängen auf den Digitalports 8 bis 15. Die alte Funktion `servoD0()` wurde aus der Bibliothek entfernt

Die drei Standard-Servoausgänge

Diese können über zwei Funktionen angesteuert werden.

`void servo_arc(unsigned char sv, unsigned char Winkel)` dreht den Servo auf einen bestimmten Winkel. Über den Parameter `sv` wird der Servo ausgewählt, hier sind die Werte 0 bis 2 zulässig (die Nummer entspricht der Kennzeichnung auf dem AKSEN-Board). Der zweite Parameter stellt einen absoluten Winkel im Bereich von 0 bis 90 Grad ein, wobei die Gradzahl direkt eingegeben werden kann. Achten Sie hierbei auf die maximale Drehweite Ihres Servos, um ihn nicht zu überdrehen.

Die Funktion `void servo(unsigned char sv, unsigned char laufzeit)` lässt den Servo auf einen absoluten Winkel drehen. Dieser wird jedoch nicht über die Gradzahl angegeben. Vielmehr wird die Zeitdauer eines Pulses in Mikrosekunden angegeben. Die aktuelle Winkelstellung wird Servos als pulswertenmoduliertes Rechtecksignal geliefert. Die Pulsweite des Signal liegt im Bereich von 0 bis 2100 μs . Mit diesem Befehl ist die Ansteuerung von Servos über den gesamten mechanischen Bewegungsspielraum (0 bis 180°) und darüber hinaus möglich.

Zur individuellen Einstellung der beidseitigen Anschlagpunkte von Servos kann das Beispiel `servo_ms` benutzt werden. Hierbei wird mit einem Poti an Analog(0) der Mikrosekunden-Impuls verändert, gleichzeitig kann am Stromverbrauch des Systems der Anschlag eines Servos an seiner mechanischen Endstellung erkannt werden.

Listing 3.11: servo.c

```
1 //Autor: Ronny Menzel
```

3 Tutorials

```
2 #include <stdio.h>
3 #include <stub.h>
4
5 //Das Programm dreht die drei Servos nacheinander
6 //von 0 auf 90 Grad.
7 //Danach werden alle Servos nacheinander wieder
8 //auf 0 Grad zurueckgedreht.
9
10 void AksenMain(void)
11 {
12     char servo, winkel;
13
14     do
15     {
16         // Servos nacheinander vorwaerts drehen
17         for (servo=0; servo <=2; servo++)
18         {
19             // Winkel auswaehlen
20             for (winkel=0; winkel<=90; winkel++)
21             {
22                 // Servo auf einen bestimmten Winkel drehen
23                 servo_arc(servo, winkel);
24
25                 // 5 ms warten
26                 sleep(5);
27             }
28         }
29
30         //Servos nacheinander rueckwaerts drehen
31         for (servo=2; servo >=0; servo--)
32         {
33             // Winkel auswaehlen
34             for (winkel=90; winkel>=0; winkel--)
35             {
36                 // Servo auf einen bestimmten Winkel drehen
37                 servo_arc(servo, winkel);
38
39                 // 5 ms warten
40                 sleep(5);
41             }
42         }
43     }while(1);
44 }
45 }
```

Die Servobank

Mit der Funktion `servobank_start()` kann eine Servo-ISR der AKSEN-Bibliothek (die sog. Servobank) aktiviert werden, die auf die Digital-Ports 8 bis 15 ebenfalls Servo-Signale aus-

gibt. Diese Digital-Ports stehen dann nicht mehr für andere Ein- und Ausgaben zur Verfügung. Mit `servobank_stop()` wird die Servobank deaktiviert.

Die Einstellung eines Servobank-Servomotors `servo` auf `zeit` Mikrosekunden erfolgt mit `servobank(servo, zeit)`. Beispiele zur Nutzung der Servobank sind `servo_ms`, `servobank`, `mt+servo` und `bench_sim`. Da die Servobank (Digital 8 - 15) und die Infrarot-Detektion (Digital 4 - 7) auf verschiedenen Ports arbeiten ist ein problemloser Parallelbetrieb möglich. Die Servobank ist kompatibel mit Multitasking.

Zusammen mit den 3 Standard-Servoports sind damit 11 Servomotoren ansteuerbar. Die Stromversorgung der 8 Servobank-Servos sollte dann nicht mehr über den 5V-Spannungsregler des AKSEN-Boards vorgenommen werden.

Abbildung 3.3 zeigt einen Beispiel-Roboter mit allen 11 Servomotoren.

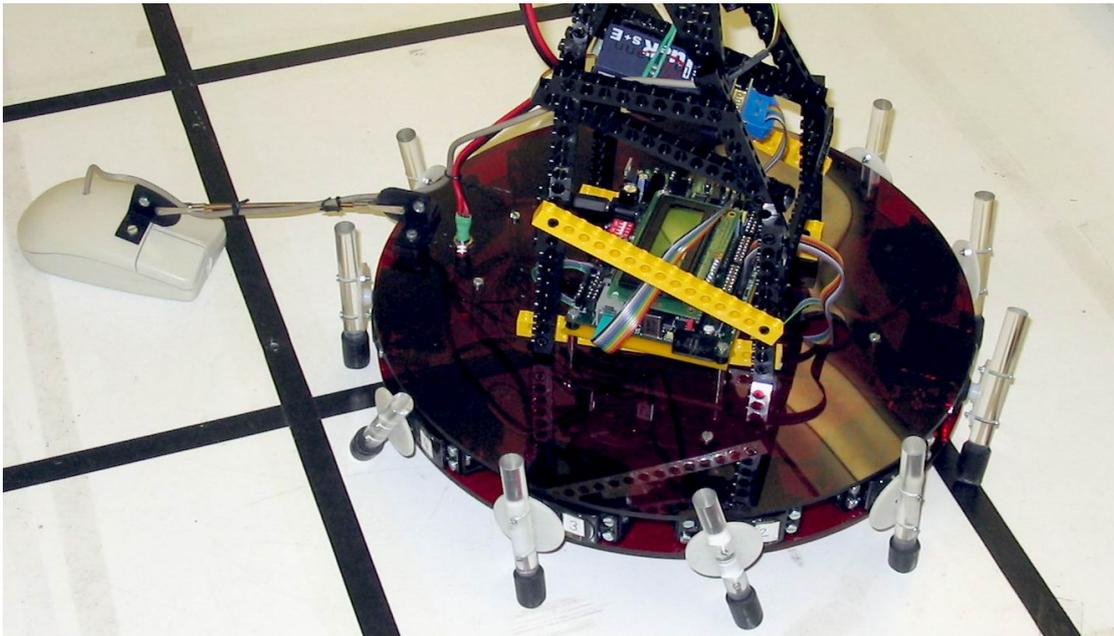


Abbildung 3.3: Roboter ELFE zur automatischen Generierung von Bewegungsmustern mit künstl. Evolution (KI-Labor der FH Brandenburg, WS04/05)

3.2.8 Encoder-Ports

Die Encoder zählen negative Flanken, also die Übergänge von High nach Low. Sie werden über drei Befehle angesprochen: `unsigned int encoder0()`, `unsigned int encoder1()` und `unsigned int encoder2`. Die Ziffer im Funktionsnamen entspricht der Encoder-Bezeichnung auf dem Board. Jeder Befehl gibt die Anzahl an Ticks (also negativen Flanken) seit seinem letzten Aufruf zurück. Der Programmierer kann so die Zeitbasis zur Ermittlung der Encoderwerte selbst bestimmen.

Listing 3.12: `encoder.c`

```
1 #include <stub.h>
```

3 Tutorials

```
2
3 //Abfragen des Encoders 0 im Sekundentakt
4 //und Anzeige auf dem LCD
5
6 void AksenMain(void)
7 {
8     unsigned char pin=0;
9     int wert=0;
10
11     while(1)
12     {
13         //Display loeschen
14         lcd_cls();
15
16         //Wert einlesen
17         wert=encoder0();
18
19         //Wert ausgeben
20         lcd_int(wert);
21
22         //eine Sekunde warten
23         sleep(1000);
24     }
25 }
```

Wird ein Encoder lange nicht abgefragt, kann es vorkommen dass ein Überlauf stattfinden. Wenn es also mehr als 65536 Flanken seit der letzten Abfrage gab, springt der Zähler auf null und zählt dann einfach weiter.

3.2.9 CAN-Bus

Da das CAN-Interface optional ist, sind die Funktionen zur Ansteuerung in einer eigenen Datei untergebracht. Um die Funktionen nutzen zu können, müssen die Dateien `can.c` und `can.h` aus dem CD-Verzeichnis `Bibliothek\Can` in das Projektverzeichnis kopiert werden. Die Datei `can.c` muss compiliert und zum Linkvorgang hinzugefügt werden, während `can.h` per `include`-Anweisung in in den Quelltext eingebunden wird.

Die Initialisierung des CAN-Interface übernimmt die Funktion `void CanInit(void)`. Die Konfiguration der Schnittstelle muss manuell in der gut dokumentierten `can.c` erfolgen. Für weitere Informationen zur Konfiguration und Benutzung des CAN-Bus sei auf [4] verwiesen. Die Kommunikation erfolgt über die Funktionen `int CanEmpfang(struct CAN_MSG xdata *CanBotschaft)` und `int CanSenden(struct CAN_MSG xdata *CanBotschaft)` wobei beiden ein Zeiger auf die Datenstruktur `CAN_MSG` übergeben werden muss. Beide Funktionen geben im Erfolgsfall 1 (also TRUE), ansonsten 0 (FALSE) aus.

Listing 3.13: `cantest.c`

```
1 //Autor: K.-U. Mrkor
```

```

2 #include <stdio.h>
3 #include <regc515c.h>
4 #include <stub.h>
5
6 #include "can.h"
7
8
9 struct CAN_MSG xdata CanBotschaft;
10
11
12 void AksenMain(void)
13 {
14     lcd_puts("CAN-Init");
15     CanInit();
16     lcd_setxy(1,0);
17     lcd_puts("erfolgt");
18
19     do
20     {
21         /* Empfangspuffer ueberpruefen und gegebenenfalls leeren */
22         if ( CanEmpfang( &CanBotschaft)==TRUE)
23         {
24             /* LCD loeschen*/
25             lcd_cls();
26
27             /* uebergebenen Wert ausgeben */
28             lcd_setxy(0,0);
29             lcd_puts("wert1:_");
30             lcd_byte(CanBotschaft.Puffer[0]);
31             lcd_setxy(1,0);
32
33             /* Adresse ausgeben */
34             lcd_hbyte(CanBotschaft.ID[0]); lcd_putchar('_');
35             lcd_hbyte(CanBotschaft.ID[1]); lcd_putchar('_');
36             lcd_hbyte(CanBotschaft.ID[2]); lcd_putchar('_');
37             lcd_hbyte(CanBotschaft.ID[3]); lcd_putchar('_');
38         }
39     }
40     while(1);
41 }

```

Die Datenstruktur CAN_MSG hat folgenden Aufbau:

```

struct CAN_MSG
{
    unsigned char Puffer[8];
    unsigned char DatenLaenge;
    unsigned char ID[4];
};

```

Die 8 Bytes von `Puffer` stehen für die Nachricht zur Verfügung. In das Byte `DatenLaenge` muss die Anzahl der tatsächlich in `Puffer` geschriebenen Bytes eingetragen sein. Das Feld `ID` nimmt die beim CAN-Bus üblichen 4 Identifikationsbytes auf, die den Zielcontroller der Nachricht adressieren. Soll der Identifikationsmechanismus genutzt werden, muss er in der Datei `can.c` eingeschaltet werden. In der vorgegebenen Konfiguration werden alle Nachrichten unabhängig von der ID empfangen.

3.3 LCD-Display

Das folgende Programm gibt auf dem Display in der oberen Zeile `LCD Tutorial` und in der unteren abwechselnd `Hallo Welt` und `AKSEN-Board` aus. Für jeden Datentyp gibt es spezielle Ausgabefunktionen. Eine Übersicht finden Sie im Anhang.

Ähnlich wie bei einer Konsole, hat das LCD-Display einen Cursor. Dieser befindet sich bei Programmstart in der oberen linken Ecke. Er kann über `lcd_setxy(int x,int y)` versetzt werden. Nach einer Ausgabe auf dem Display befindet sich der Cursor hinter dem letzten ausgegebenen Zeichen. Die nächste Ausgabe erfolgt an der Position des Cursors.

Um das Display zu löschen kann die Funktion `lcd_cls()` genutzt werden. Bei Aufruf dieser Funktion wird der Cursor wieder in die linke obere Ecke versetzt.

Listing 3.14: `lcd.c`

```
1 //Autor: Ronny Menzel
2 #include <stdio.h>
3 #include <stub.h>
4
5 void AksenMain(void)
6 {
7     // Endlosschleife fuer das Programm
8     do{
9
10         // Lcd loeschen und Cursor auf Home setzen
11         lcd_cls();
12
13         // 400 ms warten
14         sleep(400);
15
16         // Ausgabe des Textes! Cursor ist oben links
17         lcd_puts("LCD_Tutorial");
18
19         // Cursor auf Zeile 1 (untere Zeile) und Spalte 0 setzen
20         lcd_setxy(1,0);
21
22         // Ausgabe des Textes in die untere Zeile
23         lcd_puts("Hallo_Welt");
24
25         // 1000 ms warten
26         sleep(1000);
27
```

```

28     // Cursor auf Zeile 1 (untere Zeile) und Spalte 0 setzen
29     lcd_setxy(1,0);
30
31     // Ausgabe des Textes in die untere Zeile
32     lcd_puts("AkSen_Board");
33
34     // 1000 ms warten
35     sleep(1000);
36
37     }while(1);
38 }

```

Die Funktion `void lcd_setup(unsigned char DisplayOnOff, unsigned char CursorOnOff, unsigned char CursorBlink)` bietet die Möglichkeit, einige Grundparameter des Displays einzustellen. Über den ersten Parameter kann mit den Konstanten `DISPLAY_ON` und `DISPLAY_OFF` das Display an oder ausgeschaltet werden. Der zweite Parameter erlaubt das Einschalten (`CURSOR_ON`) bzw. Ausschalten (`CURSOR_OFF`) eines Cursorzeichens. Mit dem dritten Parameter kann die Darstellung des Cursorzeichens von statisch (`CURSOR_NOBLINK`) auf blinkend (`CURSOR_NOBLINK`) umgeschaltet werden.

3.4 Serielle Schnittstelle

Die serielle Schnittstelle kann mit den folgenden fünf Befehlen denkbar einfach genutzt werden. Diese Befehle sind dabei in die Dateien `serielle.h` bzw. `serielle.c` ausgelagert, die zum Projekt hinzugelinkt werden müssen.

Der Befehl `void serielle_init(void)` initialisiert die serielle Schnittstelle mit 9600 Baud, 8 Datenbits, ohne Paritätsbit und mit einem Stoppbit. Danach kann mit dem Befehl `void serielle_putchar(char c)` ein einzelnes Zeichen, mit dem Befehl `void serielle_puts(const char *string)` ein kompletter null-terminierter String gesendet werden. Der Befehl `char serielle_getchar(void)` liest ein Zeichen ein. Ein ganzer String kann mit `void serielle_gets(char * string)` eingelesen werden. Das Stringende kann durch die ASCII-Werte `0x00,0x0A (LF)` bzw. `0x0D (CR)` markiert werden. Die Lesefunktionen von der seriellen Schnittstelle arbeiten blockierend, warten also auf ein empfangenes Zeichen. Zur Nutzung nichtblockierender Lesefunktionen kann die Auswertung des Ringindikators `RI` (zeigt an, dass ein Datum im Empfangspuffer liegt) genutzt werden. Zum Zugriff auf den Ringindikator werfen Sie einen Blick in die `serielle.c`.

Listing 3.15: `seriell.c`

```

1 // Autor: K.-U. Mrkor
2 // Testausgabe auf dem seriellen Port
3
4 #include <stdio.h>
5 #include <regc515c.h>
6 #include <stub.h>
7
8 #include "serielle.h"

```

```
9
10 void AksenMain(void)
11 {
12     serielle_init();
13
14     lcd_cls();
15     lcd_puts("Hallo_Welt");
16
17     do
18     {
19         serielle_puts("Hallo_Welt");
20         sleep(2000);
21
22         lcd_setxy(1,0);
23         //Zeichen von seriell einlesen und
24         //ausgeben
25         lcd_putchar(serielle_getchar());
26     }
27     while(1);
28 }
```

3.5 Multitasking

Multitasking ist erst ab der Bibliotheksversion 0.965 als zuverlässig und schnell zu empfehlen. Bei der gleichzeitigen Verwendung der Infrarot-Detektion und Multitasking beachten Sie bitte Abschnitt 3.5.1.

Bei der gewählten Multitasking-Strategie handelt es sich um ein Round Robin ohne Prioritätsvergabe. Neu eingereichte Prozesse werden in der Prozess-Liste hinten angehängt. Insgesamt können bis zu 20 Prozesse definiert werden, wobei `AksenMain()` selbst ebenfalls ein Prozess ist.

Zum Erzeugen von Prozessen dient die Funktion

```
process_start( void (*funktionszeiger)(), unsigned char zeit)
```

Im ersten Parameter erwartet sie einen Zeiger auf die Prozess-Funktion. Bei der Prozess-Funktion wird eine ganz normale C-Funktion ohne Parameter und Rückgabewert erwartet. Einzige Besonderheit liegt beim Verlassen einer solchen Funktion. Denn das Ende einer Prozessfunktion wird nicht erkannt!!! Ein Prozess, welcher sich selbst beenden will, muss sich daher auch selbst aus der Prozess-Liste entfernen. Prozesse mit immer wiederkehrenden Aufgaben sind am besten in Endlosschleifen realisierbar. Im zweiten Parameter erwartet `process_start()` die Prozess-Zeit (also die Verweildauer des Prozesses) in ms. Bei einem Rückgabewert größer 19 handelt es sich um eine Fehlermeldung und der Prozess wurde nicht erzeugt. Ansonsten handelt es sich bei dem Rückgabewert um die von `process_start()` dem Prozess zugewiesene ID (mögliche Wert 00...19).

Diese Prozess-ID benötigt man zum Beispiel zum Löschen eines Prozesses mittels

```
process_kill(unsigned char pid) |
```

Diese Funktion nimmt den angegebenen Prozess aus der Prozessliste und setzt evt. nachfolgende Prozesse in dieser Liste auf ihre neue Position. Gibt es keinen Prozess mit der übergebenen ID wird die Konstante `PROCESS_ERROR` zurückgegeben.

Innerhalb eines Prozess kann mittels `process_get_pid()` die eigene Prozess-ID ermittelt werden. So kann sich zum Beispiel ein Prozess mit der folgenden Kombination selbst aus der Prozessliste entfernen:

```
process_kill( process_get_pid());
```

Ein Prozess kann aber auch eine vorzeitige Abgabe seiner Zeitscheibe und damit den vorzeitigen Wechsel zum nächsten Prozess in der Prozessliste mittels `process_defer()` erzwingen. Gerade bei Prozessen, welche nur beim Erreichen bestimmter Zustände die vorher vereinbarte Zeit benötigen, bietet sich diese Vorgehensweise an.

Den umgekehrten Weg ermöglicht `process_hog()`. Hier wird der Tickzähler des gerade aktiven Prozesses auf den Maximalwert 255 gesetzt und dadurch der Prozesswechsel verzögert.

Zur dauerhaften Änderung der Prozess-Zeit innerhalb der Zeitscheibensteuerung eines beliebigen Prozesses dient

```
process_set_ticks( unsigned char pid, unsigned char prozess_ticks)
```

Alle zur internen Organisation des Scheduling benötigten Felder, Variablen und Zeiger befinden sich in der Struktur `proz_tab` (siehe Abb.3.4). Das Auslesen der darin enthaltenen Informationen ist jederzeit problemlos möglich. Schreibender Zugriff auf diese Struktur ist aber nicht zulässig!!!

Im Feld `proz_liste[]` befindet sich eine geordnete Liste aller Prozesse. Die Reihenfolge in der Liste entspricht der Aufrufreihenfolge. In der Variablen `anz_proz` steht die Anzahl aller gestarteten Prozesse. `akt_proz_index` zeigt auf den gerade aktiven Prozess. In `ticks_aktuell` steht die noch verbliebene Anzahl von Ticks des gerade aktiven Prozess. Dessen Wert wird jede Millisekunde um 1 verringert.

Beim Erreichen von `ticks_aktuell=0` hat der Prozess seinen Anteil an der Zeitscheibe für diesen Durchlauf verbraucht und es wird der Scheduler aufgerufen. Dieser sichert zunächst den kompletten Kontext (siehe Abb. 3.5) des gerade aktiven Prozesses und setzt den Kontext des nächsten Prozesses in der Prozess-Liste (`akt_proz_index + 1`). Beim Erreichen des Listen-Endes (`akt_proz_index = anz_proz`) folgt ein Verweis auf Eintrag 0 der Liste (`akt_proz_index=0`). Außerdem wird mit `akt_proz=proz_liste[akt_proz_index]` die Prozess-Nummer des nun aktuellen Prozesses abgelegt und `ticks_aktuell` mit dem für diesen Prozess in `proz_ticks[akt_proz]` eingetragenen Wert überschrieben. Danach ist der Prozesswechsel abgeschlossen.

Listing 3.16: `prozess2.c`

```
1 /*****
2 /* 3 paralle Prozesse */
3 /* - es beendet sich der initiale Prozess - */
4 /* AksenMain() selbst */
```

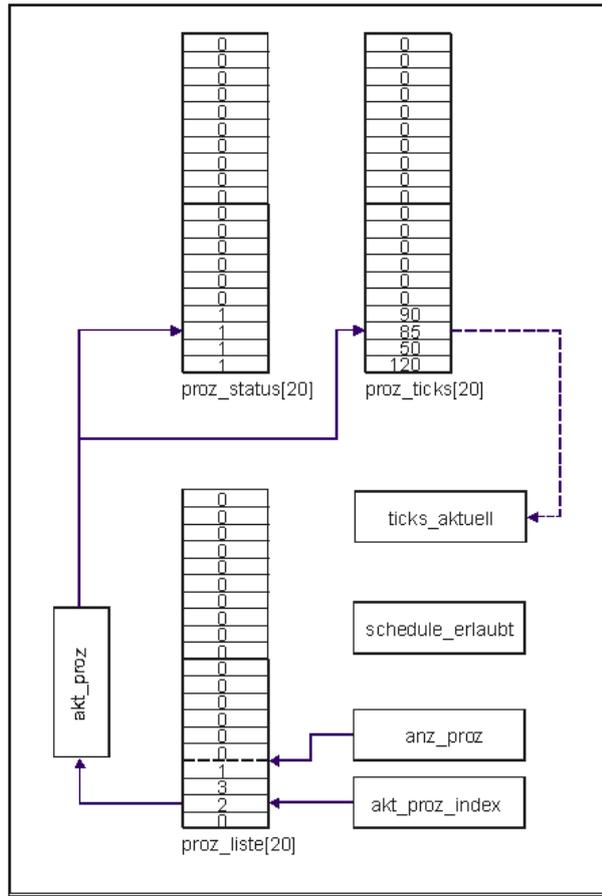


Abbildung 3.4: Aufbau von proz_tab

```

5  /*          */
6  /*          */
7  /* 23.02.2004          */
8  /* K.-U. Mrkor          */
9  /******
10
11 #include <stdio.h>
12 #include <regc515c.h>
13 #include <stub.h>
14
15 unsigned char j=0;
16 unsigned char pid_test1=25, pid_test2=25, pid_test3=25;
17
18 /******
19 /* Blinksignal auf IR */
20 /******
21 void test1( void)

```

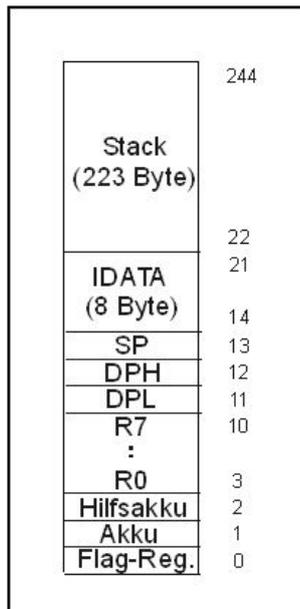


Abbildung 3.5: Prozesskontext

```

22 {
23 do
24 {
25     mod_ir_an();
26     sleep(350); /* 350ms+x */
27     mod_ir_aus();
28     sleep(350); /* 350ms+x */
29 }
30 }
31 while(1);
32 }
33
34 /*****/
35 /* Blinksignal auf LED 0-3 */
36 /*****/
37 void test2( void)
38 {
39 do
40 {
41     led(0,0);
42     led(1,0);
43     led(2,0);
44     led(3,0);
45     sleep(500); /* 500ms+x */
46     led(0,1);
47     led(1,1);

```

3 Tutorials

```
48     led(2,1);
49     led(3,1);
50     sleep(500); /* 500ms+x */
51
52 }
53 while(1);
54 }
55
56 /*****/
57 /* Kontroll-Prozess */
58 /*****/
59 void aufpasser( void)
60 {
61     do
62     {
63         j++;
64
65         if (j==128)
66             pid_test1=process_start( test1, 150);
67
68         if (j==255)
69             process_kill( pid_test1);
70
71         lcd_cls();
72
73         /* einige interne Daten anzeigen */
74         lcd_puts("P-Liste:_");
75         lcd_ubyte(proz_tab.proz_liste[0]);
76         lcd_puts("_");
77         lcd_ubyte(proz_tab.proz_liste[1]);
78         lcd_puts("_");
79         lcd_ubyte(proz_tab.proz_liste[2]);
80         lcd_puts("_");
81         lcd_ubyte(proz_tab.proz_liste[3]);
82         lcd_setxy(1,0);
83         lcd_puts("ProzAnz:_"); lcd_ubyte(proz_tab.anz_proz);
84         lcd_puts("_j=");
85         lcd_ubyte(j);
86
87         sleep(40);
88     }
89     while(1);
90 }
91 }
92
93 /*****/
94 /* MAIN-Routine */
95 /*****/
96 void AksenMain(void)
```

```

97 {
98     /* Starten von zwei Prozessen */
99     pid_test2=process_start( test2, 150);
100    pid_test3=process_start( aufpasser, 150);
101
102    /* Der Prozess AkSenMain() beendet sich selbst */
103    process_kill( process_get_pid());
104 }

```

3.5.1 Multitasking und Infrarot-Detektion

Die Infrarot-Detektion läuft in der Millisekunden-Hauptschleife der AKSEN-Bibliothek und ist darauf angewiesen, nicht durch Taskwechsel unterbrochen zu werden, da der Erkennungsautomat sonst die Synchronisation mit dem empfangenen Signal verliert. Um die Infraroterkennung zusammen mit Prozessen zu benutzen, wurde beispielhaft folgende Grundidee implementiert:

Grundidee: Ein spezieller Infrarotprozess blockiert durch Verlängern seiner Zeitscheibe den Prozesswechsel und versucht sich einige Signalperioden auf das Infrarot-Signal zu synchronisieren. Sobald die abgelaufene Zeit oder die Anzahl synchronisierter Signalperioden eine Schwelle überschreitet, wird die Messung beendet und die Zeitscheibe abgegeben. Das Detektionsergebnis kann in globalen Variablen für andere Prozesse zur Verfügung gestellt werden.

Die Infrarot-Detektion kann damit problemlos mit anderen Prozessen zusammen verwendet werden. Das folgende Beispiel `mt_ir` demonstriert dieses Vorgehen, ein etwas ausführlicheres Beispiel mit 4 Detektoren und Servomotoren ist in `bench_sim` zu finden.

Listing 3.17: `mt_ir.c`

```

1  /*****
2  /*
3  /* Blinker mit 16 Prozessen
4  /* dazu Empfang von 125Hz Infrarot
5  /* 12.05.2004
6  /* I. Boersch
7  /*
8  /*****
9
10 /* Grundidee: Der Detektor des 125Hz-IR läuft
11 /* in der Systemzeitscheibe und wird durch
12 /* Prozesswechsel gestört, daher ist es
13 /* notwendig dem Detektor genügend lange Zeit
14 /* (ca. 5-10 IR-Signal-Perioden reichen) zu
15 /* verschaffen. Der Prozess ir_proc blockiert
16 /* hierzu den Prozesswechsel für diese Zeit
17 /* und gibt dann möglichst schnell den
18 /* Prozessor weiter.
19 /*****
20
21 #include <stdio.h>

```

3 Tutorials

```
22 #include <regc515c.h>
23 #include <stub.h>
24
25 #define WAIT_BEFORE_START 1000
26 #define HALF_PERIOD 500
27
28 // Detektor für 125 Hz an D4
29 void ir_proc(void)
30 {
31     // Frequenz
32     #define IR_PROC_HALF_PERIOD 4                /* 125 Hz, Periode
        damit 8 ms */
33
34                                           /* mindestens 5-..
        x Periode */
35     #define IR_PROC_WAIT (10 * 2 * IR_PROC_HALF_PERIOD)
36     #define IR_PROC_PERIODS_FOR_OK 3            /* notw. Anzahl für
        Detektion */
37     #define IR_PROC_MAX_ERROR_PER_PERIOD 3      /* erlaubte Fehlertakte
        pro Periode */
38
39     unsigned char empfang;
40     unsigned long endezeit;
41
42     mod_ir0_maxfehler(IR_PROC_MAX_ERROR_PER_PERIOD);
43     motor_pwm(3,10);
44
45     do
46     {
47         mod_ir0_takt(IR_PROC_HALF_PERIOD);      /* Detektor an */
48         mod_ir_empf0_gutzaehler = 0;           /* mod_irX_status
        auf 0 */
49
50         // exclusive busy waiting
51         motor_richtung(3,1);
52         endezeit = akt_time() + IR_PROC_WAIT;
53         process_hog();
54         while (endezeit > akt_time())
55         {
56             if ((empfang = mod_ir0_status()) >=
                IR_PROC_PERIODS_FOR_OK) break;
57         }
58         mod_ir0_takt(0);                        /* Detektor aus */
59         motor_richtung(3,0);
60
61         //Anzeige mit LCD
62         lcd_setxy(1,0);
63         lcd_ubyte( empfang );
64         if (empfang < 100) lcd_putchar('_');
```

```

65     if (empfang < 10) lcd_putchar('_');
66
67     // Anzeige mit LED
68     if (empfang >= IR_PROC_PERIODS_FOR_OK ) {mod_ir_an();}
69     else { mod_ir_aus(); }
70
71     sleep(20);                               /* enthält
        process_defer() */
72
73     // Ausgabe in globale Variable
74     // ...
75 }
76 while(1);
77 }
78
79
80 void blink(unsigned char id)
81 {
82
83     while(1)
84     {
85         lcd_setxy(0,id);
86         lcd_ubyte(0);
87         if (id < 4 ) led(id,0);
88
89         sleep(HALF_PERIOD);
90
91         if (id < 4 ) led(id,1);
92         lcd_setxy(0,id);
93         lcd_ubyte(1);
94
95         sleep(HALF_PERIOD);
96     }
97 }
98
99
100 /*****/
101 /* Blinkprozesse */
102 /*****/
103 void p0( void) {blink(0);}
104 void p1( void) {blink(1);}
105 void p2( void) {blink(2);}
106 void p3( void) {blink(3);}
107 void p4( void) {blink(4);}
108 void p5( void) {blink(5);}
109 void p6( void) {blink(6);}
110 void p7( void) {blink(7);}
111 void p8( void) {blink(8);}
112 void p9( void) {blink(9);}

```

3 Tutorials

```
113 void p10( void) {blink(10);}
114 void p11( void) {blink(11);}
115 void p12( void) {blink(12);}
116 void p13( void) {blink(13);}
117 void p14( void) {blink(14);}
118 void p15( void) {blink(15);}
119 void p16( void) {blink(16);}
120
121 /*****
122  /* MAIN-Routine
123  *****/
124 void AksenMain(void)
125 {
126
127     /* Starten der Prozesse */
128     process_start( ir_proc, 10); sleep(10000);
129
130     process_start( p0, 10); sleep(WAIT_BEFORE_START);
131     process_start( p1, 10); sleep(WAIT_BEFORE_START);
132     process_start( p2, 10); sleep(WAIT_BEFORE_START);
133     process_start( p3, 10); sleep(WAIT_BEFORE_START);
134     process_start( p4, 10); sleep(WAIT_BEFORE_START);
135     process_start( p5, 10); sleep(WAIT_BEFORE_START);
136     process_start( p6, 10); sleep(WAIT_BEFORE_START);
137     process_start( p7, 10); sleep(WAIT_BEFORE_START);
138     process_start( p8, 10); sleep(WAIT_BEFORE_START);
139     process_start( p9, 10); sleep(WAIT_BEFORE_START);
140     process_start( p10, 10); sleep(WAIT_BEFORE_START);
141     process_start( p11, 10); sleep(WAIT_BEFORE_START);
142     process_start( p12, 10); sleep(WAIT_BEFORE_START);
143     process_start( p13, 10); sleep(WAIT_BEFORE_START);
144     process_start( p14, 10); sleep(WAIT_BEFORE_START);
145     process_start( p15, 10); sleep(WAIT_BEFORE_START);
146     process_start( p16, 10); sleep(WAIT_BEFORE_START);
147
148     /* harakiri */
149     process_kill( process_get_pid());
150 }
```

3.5.2 Semaphore

Der exklusive Zugriff auf gemeinsame Betriebsmittel kann im Multitasking über Semaphore gelöst werden. Ein Beispiel wären mehrere Prozesse, die jeweils an bestimmte Stellen im LCD-Display schreiben wollen. Jeder Prozess hat dazu eine eigene `id`, über die mit folgendem Code seine Ausgabeposition beschrieben wird:

Listing 3.18: Auszug 1 aus `bench_sim.c`

```
1         lcd_setxy(0,id);
```

```
2     lcd_putchar('A'+id);
```

Ein Prozess mit `id = 1` würde also an die LCD-Position (0,1) ein 'B' schreiben. Wenn er dabei zwischen Zeile 1 und 2 durch einen Taskwechsel unterbrochen wird, und ein anderer Prozess den Cursor versetzt, erscheint die Ausgabe von Prozess1 an der falschen Stelle. Obwohl das LCD-Display hier kein exklusives Betriebsmittel ist, ist es sehr wohl der Cursor des LCD-Displays. Es ist also sinnvoll diese Anweisung durch einen kritischen Abschnitt mithilfe eines Semaphores zu schützen:

Listing 3.19: Auszug 2 aus `bench_sim.c`

```
1     S_down(LCD_MUTEX);
2     lcd_setxy(0,id);
3     lcd_putchar('A'+id);
4     S_up(LCD_MUTEX);
```

Die Realisierung eines Semaphores benötigt in der Regel eine atomare Anweisung, die der Mikrokontroller nicht zur Verfügung stellt. Allerdings kann ein Prozess seine Zeitscheibe mit `process_hog()` verlängern und damit für eine kurze Zeit garantieren, dass er nicht unterbrochen wird. Diese Idee ist im Beispiel `2semaphore` für 10 binäre Semaphore in den Dateien `sema.h` und `sema.c` umgesetzt und funktioniert. Durch Einbinden dieser Dateien stehen die 10 Semaphore mit ihren Zugriff-Funktionen (`sema.h`) für eigene Programme zur Verfügung.

Die beschriebene Ausgangssituation mit den auf das LCD-Display schreibenden Prozessen ist in dem Beispiel `2bench_sim` zu finden. Hier versuchen 13 Prozesse auf das Display zuzugreifen und garantieren die Exklusivität mit einem Semaphore. In diesem Beispiel wird das Betriebsmittel zum Engpass und bremst die darauf angewiesenen Prozesse ab. Bei Freigabe eines Betriebsmittels erhält durch die Round-Robin-Strategie des Multitasking der in der Prozessliste nächste auf das Betriebsmittel wartende Prozess das Betriebsmittel. Es ist also garantiert, dass jeder Prozess irgendwann Zugriff erhält, falls jeder Prozess den kritischen Abschnitt in endlicher Zeit verlässt.

3.6 Weitere Komponenten

3.6.1 DIP-Schalter

Der auf dem AKSEN-Board befindliche DIP-Schalter kann über zwei Funktionen abgefragt werden: `unsigned char dip()` liefert in den ersten vier Bits die Schalterstellung zurück (0 ist off, 1 ist on). Über `unsigned char dip_pin(unsigned char dip_schalter)` kann ein bestimmter Schalter abgefragt werden. Rückgabewert ist hier 0 für off und 1 für on. Der DIP-Schalter eignet sich gut zur Auswahl von verschiedenen Programmteilen oder Betriebsarten.

3.6.2 Zeitsteuerung

Die AKSEN-Bibliothek liefert einige Funktionen, um eine einfache Zeitsteuerung zu implementieren. Dazu wird ein interner Timer benutzt, der im Millisekundentakt zählt. Die Funktion

3 Tutorials

`void clear_time()` setzt diesen Timer auf 0 zurück. Mit `unsigned long akt_time()` kann die aktuelle Zeit in Millisekunden seit Systemstart oder Zurücksetzen des Timers abgefragt werden.

Die Funktion `void sleep(unsigned long wartezeit)` legt das Programm für `wartezeit` Millisekunden schlafen, beim Multitasking wird die Zeitscheibe dabei an den nächsten Prozess übergeben. Alle Hintergrundfunktionen, wie Encoder-Zählung, Zeitmessung und Servo-Steuerung werden weitergeführt.

3.6.3 Versionsabfrage

Die aktuelle Version der AKSEN-Bibliothek kann mit der Funktion `version()` abgefragt werden. Die Funktion gibt die Nummer der aktuellen Version als Zeichenfolge zurück.

3.7 Verfügbarer Speicher

Generell gilt: Es ist genug für alle da, aber wir sparen trotzdem: Also keine `int`-Variable deklarieren, wenn auch `inchar` ausreicht.

3.7.1 Lokale Variablen

Jeder Prozess, auch `AkSenMain()` verfügt über 221 Byte lokalen Stack. In diesem Stack werden lokale Variablen, aber auch alle Parameter und Rücksprungadressen beim Aufruf von Funktionen abgelegt. Das Überschreiten des Stackrahmens wird zur Laufzeit durch die AKSEN-Bibliothek kontrolliert. Bei Verlassen des gültigen Speicherbereiches wird das Programm mit der Fehlermeldung 'Stack-Ueberlauf in Prozess XX' abgebrochen. Der Zugriff auf lokale Variablen ist mehr als dreimal so schnell wie der auf globale Variablen, allerdings werden lokale Variablen beim Taskwechsel gesichert, dies dauert um so länger je mehr es sind.

3.7.2 Globale Variablen

Von den verfügbaren 8 KB Speicher nutzt die Bibliothek 512 Byte, die verbleibenden 7.5 KB teilen sich

- die laufenden Prozesse (jeweils 256 Byte),
- globale Variablen des `sdcc`-Compilers (maximal 100 Byte, meist 10 oder weniger),
- Reservierung für Bibliotheksparameter (16 Byte)
- sowie der Speicherplatz für globale Variablen.

Der Speicher für den `sdcc` wird nur benötigt bei Divisionen etc., die vollständige Liste ist noch nicht bekannt. Der verfügbare Speicher für globale Variablen berechnet sich daher wie folgt:

$$\text{Globaler Speicher} = 7680 - \text{Prozessanzahl} * 256 - 100 - 16$$

Beispiel mem: AkSenMain startet 2 weitere Prozesse, somit laufen insgesamt 3 Prozesse, der sdcc-Compiler legt keine Variablen an (sichtbar in der *.map-Datei, dazu im Makefile das Löschen auskommentieren). Der verfügbare globale Speicher berechnet sich damit als

$$\text{Globaler Speicher} = 7680 - 3 * 256 - 16 = 6896 \text{ Bytes}$$

Es ist also folgende globale Variable möglich:

```
unsigned char daten [2][3448];
```

Sie finden dieses Beispiel unter mem. Die Überschreitung des globalen Speichers wird nicht abgefangen und erzeugt zur Laufzeit Programmfehler, die meist schwer zu entdecken sind. Im Beispiel genügt die Vergrößerung der globalen Variable um ein Byte, um einen (hier durch Speichertest erkannten) Überlauf hervorzurufen.

3.7.3 Programmgröße

Der Programmspeicher beträgt 32 KB, wieviel Speicher ein Programm belegen wird, ist aus dem C-Quelltext schwer abzuschätzen. Eine Schätzung für den benötigten Speicher lässt sich aus den Adressen des .ihx-Files ableiten - im 2. und 3. Byte einer Zeile steht die Zieladresse. In Abbildung 3.6 finden Sie eine Übersicht der Beispiele hinsichtlich Größe des Quelltextes (inkl. Kommentaren), Größe des damit belegten Speicherplatzes und den Quotienten aus beiden (d.h. wieviel Speicherplatz wird durch 1 Byte Quellcode belegt). Aus dem Diagramm lassen sich 2 Dinge ablesen:

1. Größere Programme haben einen kleineren Kommentaranteil (konvergiert)
2. Ein Byte Quelltext erzeugt im Mittel ca. ein halbes Byte Code

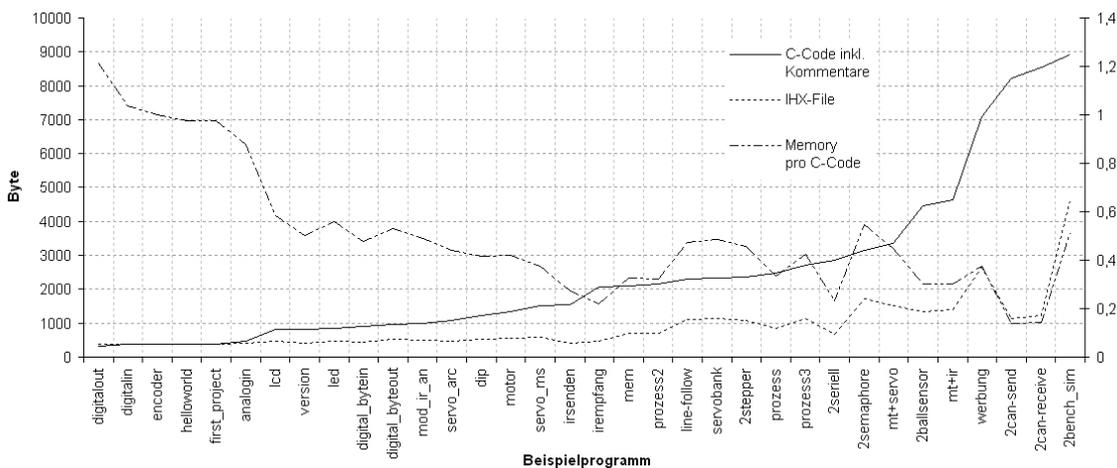


Abbildung 3.6: Quelltext und Programmgröße

Daher die Prognose: Wenn sich die Gesamtgröße eines Programm-Quelltextes der 64 KB-Marke nähert, könnte der Programmspeicher auf dem Board knapp werden. Das größte Beispielprogramm hat 8914 Quellbytes.

3.8 Testprogramm

Die CD enthält ein Testprogramm, mit dem es möglich ist, alle Funktionen des Boards zu testen. Die einzelnen zu testenden Funktionsgruppen werden dabei über die DIP-Schalter eingestellt. Um eine andere Funktionsgruppe zu testen, müssen die DIP-Schalter in die richtige Position gebracht werden. Danach muss ein Reset ausgelöst werden.

Folgende Tests sind verfügbar. Die Stellung der DIP-Schalter ist binärcodiert angegeben. Das erste Bit entspricht DIP-Schalter 1. Ein Bitwert von eins entspricht der Schalterstellung on.

DIP-Stellung	Funktion
0000	Abfrage der Digitalports 00 bis 07. Die Werte der Ports werden nebeneinander auf dem Display ausgegeben.
0001	Abfrage der Digitalports 08 bis 15.
0010	Abfrage der Analogports 00 bis 07. Auch hier werden die Portwerte nebeneinander auf dem Display ausgegeben.
0011	Abfrage der Analogports 08 bis 14.
0100	Ausgabe auf den Digitalports 00 bis 15. Die Ports werden einfach eingeschaltet.
0101	LED-Anschluss-Test
0110	Motor-Port-Test
0111	Infrarot-Test
1000	Servo-Test
1001	Encoder-Test

4 Anhang

4.1 FAQ

Q: Mein AKSEN-Board funktioniert überhaupt nicht!

A: Prüfen Sie, ob überhaupt Spannung anliegt und ob die Polung korrekt ist. Das AKSEN-Board sollte ohne Sensorik und Aktorik max. 170 mA verbrauchen. Das Board funktioniert, wenn die grüne oder rote Lampe neben dem Modusschalter leuchtet.

Q: Mein Programm startet nicht.

A: Haben Sie das Programm korrekt geflasht (LCD-Anzeige: Programmierung erfolgreich!)? Haben Sie den Modus-Schalter auf User (grüne LED) gestellt? Haben Sie Reset gedrückt? Drücken Sie Reset mehrmals, auch etwas länger.

Q: Ich sehe nichts auf dem Display.

A: Meistens liegt das am verstellten Display-Kontrast. Schalten Sie das Board ein, am besten im Flash-Modus, da dort eine Ausgabe auf dem Display erfolgt. Drehen Sie dann mit einem Schraubenzieher den Drehwiderstand links unterhalb des LCD-Displays langsam nach links und rechts. Wenn die Schrift dann nicht erscheint, prüfen Sie, ob die Beinchen des LCD-Displays richtig in der Fassung auf der Hauptplatine stecken.

Ein weiterer Fehler, der häufig passieren kann, ist eine zu schnelle Ausgabe. Wenn die Anzeige in zu kurzen Abständen aktualisiert wird, kann es sein, dass nichts oder nur ein kurzes Aufblitzen zu sehen ist. Die Aktualisierungsfrequenz sollte 10Hz nicht überschreiten. Weiterhin sollte der Zeitraum vom Löschen des Displays bis zum Ende des Schreibvorgangs möglichst kurz sein. Es ist ratsam, die Schreibvorgänge an einer Stelle im Programm zusammenzufassen. Da alle Ausgaben dann etwa gleichlang auf den Display erscheinen, sind sie insgesamt besser lesbar.

Q: Die Schrift sieht korrumpiert aus, oder es werden schwarze Blöcke angezeigt.

A: Das passiert manchmal nach dem Start und stellt kein Problem dar. Drücken Sie Reset, danach sollte sich die Anzeige normalisieren. Wenn das Display nur schwarze Blöcke anzeigt, dann ist wahrscheinlich der Display-Kontrast zu hoch eingestellt.

Q: Der Spannungsregler U12 (gleich rechts neben der Stromversorgungsbuchse) wird sehr heiss!

A: Das stellt kein Problem dar, der Kühlkörper leitet die Wärme ab. Zu große Hitzeentwicklung ist meist ein Zeichen zu hoher Versorgungsspannung. Reduzieren Sie, wenn möglich, die Versorgungsspannung.

Q: Ich kann mein Programm unter Linux nicht flashen!

A: Auf neueren Computern, insbesondere bei Verwendung von ACPI kann es zu Problemen

4 Anhang

beim Verwenden der seriellen Schnittstelle. Abhilfe schafft hier beispielsweise ein Kernelupdate auf Version 2.6.0 oder höher. Dadurch wird der Flashvorgang auch wesentlich beschleunigt.

Q: Auf dem Display erscheint immer: „-AKSENMain- wurde beendet“

A: Dies ist kein Fehler, diese Meldung erscheint, wenn das Programm die `AksenMain()` Routine beendet hat. In der Sektion Tutorials auf Seite 17 gibt es dazu weitere Hinweise.

Q: Mein Programm erzeugt Fehlermeldungen beim Compilieren oder Linken, die garantiert nicht von mir kommen!

A: Prüfen Sie die Erreichbarkeit des Compilers `sdcc` und des Verzeichnisses `/usr/local/aksen-lib`. Prüfen Sie die Pfade in der Datei `config.mak`. Beim Linken muss die Datei `aksen.rel` sichtbar sein und als ERSTE Datei beim Linken angegeben werden. Sollten damit weiterhin Fehler auftreten, prüfen Sie Ihre `sdcc`-Installation (siehe Abschnitt 1.2).

Q: Mein Programm bleibt stehen?

A: Sie haben einen Fehler im Programm, das wohl syntaktisch richtig, aber semantisch problematisch ist, z.B. Zuweisung von long-Werten an int-Variablen, defekte Pointer, überlaufende Arrays etc.

```
1   unsigned int zeit;  
2   zeit = akt_time();           // FALSCH!
```

Q: Wieviel Strom darf man über die einzelnen Ports ziehen?

A:

Motor-Port 1000mA pro Kanal
LED Ptot=1W bei 25 °C (Ptot =VCC2*I)
IR Ptot =65W bei 25 °C (Ptot =VCC2*I)
I/O-Pin 3mA(L), -10.. - 80µA(H) Das reicht zum Treiben einer CMOS-Last.

Q: Wieviel Strom darf insgesamt von den angeschlossenen Komponenten (Motoren, Leds, Sensoren usw.) gezogen werden?

A: Man muss hier zwischen den beiden Stromversorgungszweigen VCC und VCC2 unterscheiden. Während der Strom über VCC2 (Servos, Motoren, LEDs) nur von der Leitungsbreite auf dem Board abhängig ist, begrenzt bei VCC der Spannungsregler U12 den maximalen Strom des gesamten 5V-Zweiges auf 1 A. Bei diesem Maximalwert muss aber für ausreichend Kühlung des ICs gesorgt werden!

Q: Sind für die LED-Anschlüsse Vorwiderstände nötig?

A: Ja, die Pegel an den vier geschalteten Ausgängen entspricht der Akkuspannung VCC2.

Q: Wie ist die Signalbelegung der Ausgänge des AKSEN-Boards?

A:

Masse	VCC	Frei	Signal
-------	-----	------	--------

Diese Anordnung minimiert das Risiko des Verpolens. Sowohl optisch durch die Asymmetrie, als auch elektrisch da bei Verpolung nur Signal und Masse vertauscht wurde.

4.2 Funktionen der AKSEN-Bibliothek

4.2.1 Analoge und Digitale Anschlüsse

```
unsigned char analog(unsigned char pin);
```

Gibt den Wert der an Port **pin** anliegenden Spannung zurück. Die Nummerierung der Pins entspricht der auf dem Board aufgedruckten Beschriftung. Der Rückgabewert kann zwischen 0 (entspricht 0V) und 255 (5V) liegen.

```
void led(unsigned char laempchen, unsigned char wert);
```

Schaltet den LED-Port **laempchen** ein (**wert**=1) oder aus (**wert**=0).

```
void digital_out(unsigned char ausgabepin, unsigned char wert);
```

Schaltet den Signal-Pin an Digital-Port **ausgabepin** ein (**wert**=1) oder aus (**wert**=0).

```
unsigned char digital_in(unsigned char pin);
```

Gibt den am Digital-Port **pin** anliegenden Pegel zurück (1 für HIGH bzw. 0 für LOW).

```
void digital_byteout(unsigned char port, unsigned char wert);
```

Gibt das Byte **wert** an dem Digital-Port **port** aus. **port** kann sein PORT_1 oder PORT_2.

```
unsigned char digital_bytein(unsigned char port);
```

Liest ein Byte vom Digital-Port **port** ein. **port** kann sein PORT_1 oder PORT_2.

4.2.2 Infrarot

```
#define mod_ir_an()
```

Schaltet den IR-Port ein.

```
#define mod_ir_aus()
```

Schaltet den IR-Port aus.

```
#define setze_ir_senden( wert)
```

Für **wert** wird die halbe Periodendauer des abzustrahlenden Signals in Millisekunden angegeben. Eine 4 moduliert das Signal mit 125Hz, eine 5 mit 100Hz. 0 schaltet die Modulation ab.

```
#define mod_ir0_takt( takt)
```

Setzt die halbe Periodendauer der zu detektierenden Frequenz in Millisekunden. Ein Wert von 4 entspricht 125Hz, 5 entspricht 100Hz. Mit 0 wird der Empfänger abgeschaltet. Dieser Befehl

4 Anhang

erwartet den Empfänger am Digital-Port 04.

```
#define mod_ir1_takt( takt)
```

Setzt die halbe Periodendauer der zu detektierenden Frequenz in Millisekunden. Ein Wert von 4 entspricht 125Hz, 5 entspricht 100Hz. Mit 0 wird der Empfänger abgeschaltet. Dieser Befehl erwartet den Empfänger am Digital-Port 05.

```
#define mod_ir2_takt( takt)
```

Setzt die halbe Periodendauer der zu detektierenden Frequenz in Millisekunden. Ein Wert von 4 entspricht 125Hz, 5 entspricht 100Hz. Mit 0 wird der Empfänger abgeschaltet. Dieser Befehl erwartet den Empfänger am Digital-Port 06.

```
#define mod_ir3_takt( takt)
```

Setzt die halbe Periodendauer der zu detektierenden Frequenz in Millisekunden. Ein Wert von 4 entspricht 125Hz, 5 entspricht 100Hz. Mit 0 wird der Empfänger abgeschaltet. Dieser Befehl erwartet den Empfänger am Digital-Port 07.

```
#define mod_ir0_status()
```

Liefert einen Wert zurück, der die Stabilität des IR-Signals wiedergibt. Ein Wert über 30 steht für ein stabiles Signal. Wird das Signal nicht richtig erkannt, dann wird dieser Zähler zurückgesetzt und zählt von neuem bis max. 255 hoch. Dieser Befehl erwartet den Empfänger am Digital-Port 04.

```
#define mod_ir1_status()
```

Liefert einen Wert zurück, der die Stabilität des IR-Signals wiedergibt. Ein Wert über 30 steht für ein stabiles Signal. Wird das Signal nicht richtig erkannt, dann wird dieser Zähler zurückgesetzt und zählt von neuem bis max. 255 hoch. Dieser Befehl erwartet den Empfänger am Digital-Port 05.

```
#define mod_ir2_status()
```

Liefert einen Wert zurück, der die Stabilität des IR-Signals wiedergibt. Ein Wert über 30 steht für ein stabiles Signal. Wird das Signal nicht richtig erkannt, dann wird dieser Zähler zurückgesetzt und zählt von neuem bis max. 255 hoch. Dieser Befehl erwartet den Empfänger am Digital-Port 06.

```
#define mod_ir3_status()
```

Liefert einen Wert zurück, der die Stabilität des IR-Signals wiedergibt. Ein Wert über 30 steht für ein stabiles Signal. Wird das Signal nicht richtig erkannt, dann wird dieser Zähler zurückgesetzt und zählt von neuem bis max. 255 hoch. Dieser Befehl erwartet den Empfänger am Digital-Port 07.

```
#define mod_ir0_maxfehler( fehler)
```

Stellt die maximale Zeitdauer in Millisekunden ein, den eine Flanke zu früh oder zu spät kommen kann. Dieser Befehl erwartet den Empfänger am Digital-Port 04.

```
#define mod_ir1_maxfehler( fehler)
```

Stellt die maximale Zeitdauer in Millisekunden ein, den eine Flanke zu früh oder zu spät kommen kann. Dieser Befehl erwartet den Empfänger am Digital-Port 05.

```
#define mod_ir2_maxfehler( fehler)
```

Stellt die maximale Zeitdauer in Millisekunden ein, den eine Flanke zu früh oder zu spät kommen kann. Dieser Befehl erwartet den Empfänger am Digital-Port 06.

```
#define mod_ir3_maxfehler( fehler)
```

Stellt die maximale Zeitdauer in Millisekunden ein, den eine Flanke zu früh oder zu spät kommen kann. Dieser Befehl erwartet den Empfänger am Digital-Port 07.

4.2.3 LCD-Display

```
void lcd_cls(void);
```

Löscht das LCD-Display.

```
void lcd_home(void);
```

Versetzt den Cursor des LCD-Displays in die linke obere Ecke.

```
void lcd_setup( unsigned char DisplayOnOff, unsigned char CursorOnOff, unsigned char CursorBlink);
```

Erlaubt grundlegende Einstellungen des LCD-Displays. **DisplayOnOff** schaltet das Display ein (**DISPLAY_ON**) bzw. aus (**DISPLAY_OFF**). **CursorOnOff** schaltet ein Cursor-Rechteck ein (**CURSOR_ON**) und aus (**CURSOR_OFF**). Der Cursor wird mit **CURSOR_BLINK** zum Blinken gebracht, **CURSOR_NOBLINK** macht die Cursoranzeige statisch.

```
void lcd_setxy(unsigned char zeile, unsigned char spalte);
```

Setzt den Cursor an die Position (**zeile,spalte**) auf dem LCD-Display. Das nächste ausgegebene Zeichen erscheint an dieser Stelle. Die obere linke Ecke ist (0,0).

```
void lcd_putchar(char c);
```

Gibt den Buchstaben **c** auf dem LCD-Display aus.

```
void lcd_puts(const char *string);
```

4 Anhang

Gibt den Text in **string** auf dem LCD-Display aus. Der String muss nullterminiert sein.

```
void lcd_ubyte(unsigned char c);
```

Gibt den 8-Bit-Wert in **c** als vorzeichenlose Dezimalzahl aus.

```
void lcd_byte(char wert);
```

Gibt den 8-Bit-Wert in **wert** als vorzeichenbehaftete Dezimalzahl aus.

```
void lcd_hbyte(unsigned char wert);
```

Gibt den 8-Bit-Wert in **wert** als vorzeichenlose Hexadezimalzahl aus.

```
void lcd_uint(unsigned int i);
```

Gibt den 16-Bit-Wert in **i** als vorzeichenlose Dezimalzahl aus.

```
void lcd_int(int i);
```

Gibt den 16-Bit-Wert in **i** als vorzeichenbehaftete Dezimalzahl aus.

```
void lcd_hint(unsigned int i);
```

Gibt den 16-Bit-Wert in **i** als vorzeichenlose Hexadezimalzahl aus.

```
void lcd_ulong(unsigned long wert);
```

Gibt den 32-Bit-Wert in **wert** als vorzeichenlose Dezimalzahl aus.

```
void lcd_long(long wert);
```

Gibt den 32-Bit-Wert in **wert** als vorzeichenbehaftete Dezimalzahl aus.

```
void lcd_hlong(unsigned long wert);
```

Gibt den 32-Bit-Wert in **wert** als vorzeichenlose Hexadezimalzahl aus.

4.2.4 Encoder, Servos und Motoren

```
void motor_pwm( unsigned char motor, unsigned char geschw);
```

Schaltet Motor-Port **motor** auf die Geschwindigkeit **geschw**. Die Nummerierung der Motor-Ports entspricht der Beschriftung auf dem Board. Als Geschwindigkeit können die Werte 0 (stopp) bis 10 (volle Geschwindigkeit) eingesetzt werden.

```
void motor_richtung( unsigned char motor, unsigned char richtung);
```

Schaltet die Drehrichtung des Motors **motor** um. **richtung** kann 0 oder 1 sein.

```
void servo(unsigned char sv, unsigned int laufzeit);
```

Stellt die Pulsweite **laufzeit** in Mikrosekunden am Servo-Port **sv** ein. Der gesamte 16-Bit-Wertebereich kann genutzt werden.

```
void servo_arc(unsigned char sv, unsigned char winkel);
```

Dreht den am Servo-Port **sv** angeschlossenen Servo auf die Position **winkel**. Möglich sind Winkel im Bereich von 0 bis 180 Grad.

```
void servobank_start(void);
```

Startet die Servobank zur Ausgabe von Servo-Signalen auf den Digital-Ports 8 bis 15.

```
void servobank_stop(void);
```

Stoppt die Servobank.

```
unsigned char servobank(unsigned char servo, unsigned int zeit);
```

Setzt die Pulsweite des Servoimpulses von Servo **servo** der Servobank auf **zeit** Mikrosekunden. **servo** kann hierbei die Werte 0 bis 7 annehmen, wobei 0 dem Digitalport 8 und 7 dem Digitalport 15 entspricht. Rückgabewert ist die Nummer des Servos, 11 bei ungültigem Servo und 12 bei ungültiger Zeit (größer als 2100 Mikrosekunden). Vorher ist einmalig die Funktion `servobank_start()` zu rufen.

```
unsigned int encoder0( void);
```

Gibt die Anzahl der negativen Flanken an Encoder-Port 0 zurück. Bei jeder Abfrage wird der Zähler auf 0 zurückgesetzt.

```
unsigned int encoder1( void);
```

Gibt die Anzahl der negativen Flanken an Encoder-Port 1 zurück. Bei jeder Abfrage wird der Zähler auf 0 zurückgesetzt.

```
unsigned int encoder2( void);
```

Gibt die Anzahl der negativen Flanken an Encoder-Port 1 zurück. Bei jeder Abfrage wird der Zähler auf 0 zurückgesetzt.

4.2.5 Multitasking

```
unsigned char process_start( void (*funktionszeiger)(), unsigned char zeit );
```

Reiht die Funktion als neuen Prozess in die Prozessliste ein.

4 Anhang

```
unsigned char process_kill( unsigned char pid );
```

Entfernt Prozess aus der Prozessliste.

```
void process_hog();
```

Setzt Tickzähler des aktuellen Prozesses auf 255.

```
void process_defer();
```

Veranlasst vorzeitigen Prozesswechsel.

```
void process_set_ticks( unsigned char pid, unsigned char ticks);
```

Neuzuweisung einer Prozess-Zeit für die Zeitscheibe eines beliebigen Prozesses.

```
unsigned char process_get_pid();
```

PID des aktiven Prozesses ermitteln.

4.2.6 Zeitsteuerung

```
void sleep(unsigned long wartezeit);
```

Wartet für **wartezeit** Millisekunden.

```
unsigned long akt_time( void);
```

Gibt die aktuelle Zeit in Millisekunden seit Systemstart bzw. dem letzten Aufruf von **clear_time** zurück.

```
void clear_time( void);
```

Setzt den Zeitzähler auf 0 zurück.

4.2.7 Diverses

```
unsigned char dip( void);
```

Gibt die Binärcodierte Stellung der Dip-Schalter zurück. Bit 0 entspricht Dip-Schalter 0 usw.

```
unsigned char dip_pin( unsigned char dip_schalter);
```

Gibt die Stellung von Dip-Schalter **dip_schalter** zurück. 0 bedeutet Off, 1 bedeutet On.

```
void version(char *versionstext);
```

Liefert einen Zeiger auf den aktuellen Versionsstring zurück.

4.2.8 Erweiterungen

Die folgenden Funktionen sind nicht in der Bibliothek enthalten. Um sie benutzen zu können muss die unten angegebene C-Datei zum Projekt hinzugefügt und mit `gcc` kompiliert, sowie die entsprechende Header-Datei eingebunden werden.

<code>void CanInit(void);</code>	<code>can.c;can.h</code>
--	--------------------------

Initialisiert die CAN-Schnittstelle. Die Konfigurationseinstellungen in dieser Funktion müssen manuell geändert werden. Die Funktion befindet sich in `can.c`.

<code>int CanEmpfang(struct CAN_MSG xdata *CanBotschaft);</code>	<code>can.c;can.h</code>
---	--------------------------

Holt eine empfangene CAN-Botschaft von der CAN-Schnittstelle ab und füllt damit die übergebene `CAN_MSG`-Struktur.

<code>int CanSenden(struct CAN_MSG xdata *CanBotschaft);</code>	<code>can.c;can.h</code>
--	--------------------------

Sendet die übergebene **CanBotschaft**.

<code>void serielle_init(void);</code>	<code>serielle.c;serielle.h</code>
--	------------------------------------

Initialisiert die serielle Schnittstelle mit 9600 Baud, 8 Datenbit, ohne Parität und mit 1 Stopbit.

<code>void serielle_putchar (char c);</code>	<code>serielle.c;serielle.h</code>
--	------------------------------------

Sendet das Zeichen in `c` über die serielle Schnittstelle.

<code>void serielle_puts (const char* string);</code>	<code>serielle.c;serielle.h</code>
---	------------------------------------

Sendet den null-terminierten String **string** über die serielle Schnittstelle.

<code>char serielle_getchar (void);</code>	<code>serielle.c;serielle.h</code>
---	------------------------------------

Liest ein Zeichen von der seriellen Schnittstelle.

<code>char serielle_gets (char * string);</code>	<code>serielle.c;serielle.h</code>
---	------------------------------------

Liest einen String von der seriellen Schnittstelle in **string** ein. Der String wird mit `0x00`, `0x0D` bzw. `0x0A` terminiert.

4.3 Bauanleitung für Sensoren

Die folgende Anleitung ist als Muster für den Bau eigener Sensoren und Aktoren gedacht. Die hier gezeigte Bauweise hat sich als am stabilsten und einfachsten erwiesen. Exemplarisch wird eine Leuchtdiode passend zum AKSEN-Board verkabelt.

1. Als Sensorkabel eignen sich Streifen von Flachbandkabeln hervorragend. Trennen Sie zunächst so viele Adern, wie Sie benötigen in der gewünschten Länge ab. Trennen Sie die Adern an beiden Enden ein bis zwei Zentimeter auf, isolieren Sie die Adernenden ab und verzinnen Sie sie. Wenn der Sensor ein Vorwiderstand erfordert, ist es am günstigsten, ihn in der Kabelmitte einzubauen. Trennen Sie dazu die Ader in der Kabelmitte voneinander und schneiden Sie ein Stück aus einer Ader heraus. Auch hier müssen dann die Aderenden abisoliert und verzinnt werden.

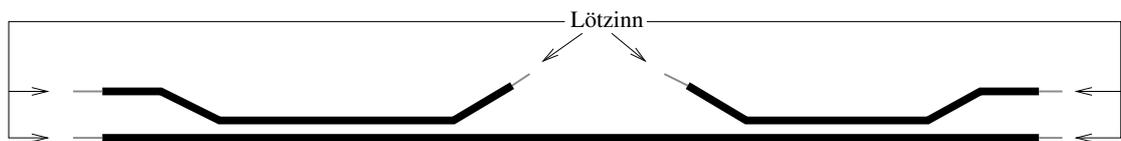


Abbildung 4.1: Sensoraufbau - Schritt 1

2. Um die Lötstellen zu isolieren, stecken Sie auf jedes Aderende ein Stück Schrumpfschlauch. Der Vorwiderstand wird mit einem Schrumpfschlauch über alle Adern isoliert, das macht die Konstruktion stabiler. Genauso kann über die beiden Schrumpfschläuche am Sensorende des Kabels ein weiterer Schrumpfschlauch gezogen werden. Jetzt kann der Vorwiderstand eingelötet werden. Beim Einlöten des Steckers und des Sensors ist auf die richtige Polung bzw. Pinbelegung zu achten.

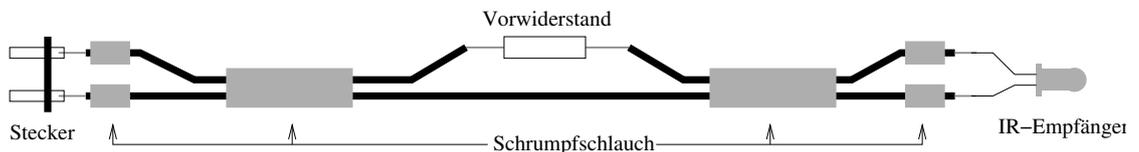


Abbildung 4.2: Sensoraufbau - Schritt 2

3. Wenn alles angelötet ist, ziehen Sie die Schrumpfschläuche über die Lötstellen und schrumpfen Sie sie fest. Dazu eignet sich ein Fön oder Heissluftabbeizer.



Abbildung 4.3: Sensoraufbau - Schritt 3

4.4 Erweiterungen des AKSEN-Boards

4.4.1 RCUBE Architektur

Das AKSEN-Board ist für Steuerungs- und Regelungsaufgaben bestens geeignet. Für weiterführende Aufgaben existiert die Möglichkeit, Erweiterungsboards an das Board anzuschließen. Das kann über mehrere Möglichkeiten geschehen. Die offensichtlichste ist der CAN-Bus. An ihn lassen sich mehrere Geräte gleichzeitig anschließen. Der Programmieraufwand hält sich in Grenzen, es sind aber auch komplexere Kommunikationsprotokolle implementierbar.

Eine weitere Kommunikationsmöglichkeit ist die serielle Schnittstelle. Die Schnittstelle ist direkt im Mikrocontroller implementiert und kann daher recht einfach programmiert werden. Durch die Trennung von Flash-Vorgang und User-Modus kann eine ungestörte Übertragung gewährleistet werden.

Die dritte Möglichkeit wäre der Anschluss von CMOS-basierten Peripheriebausteinen über die digitalen I/O-Pins des Boards. Sie sind direkt mit dem Mikrocontroller verbunden.

Die FH-Brandenburg hat das AKSEN-Board in einen Roboterkernel implementiert [2]. Dieser RCUBE genannte Kernel besteht aus drei Teilen: dem an der Universität Delft in den Niederlanden entwickelten LART-Board, einem im Rahmen einer Diplomarbeit entstandenen Video-Controller-Board und dem AKSEN-Board. Das LART-Board läuft mit ARM-Linux als Betriebssystem und kann über Video4Linux2-Treiber mit dem VIO genannten Video-Controller kommunizieren. Das LART und das AKSEN-Board sind über den CAN-Bus miteinander verbunden. Der rCube hat durch die Fähigkeiten zur Bildverarbeitung und zum Umgang mit Sensorik und Aktorik eine breite Palette an Einsatzfeldern. So ist er zum Beispiel als intelligenter Sensor oder -Kamera oder eben als Roboterkernel einsetzbar. Durch den niedrigen Stromverbrauch (ohne Sensoren weniger als 4 Watt) ist ein effizienter Betrieb auch auf mobilen Geräten möglich.

Eine Beispielapplikation ist der Roboter R2D0. Er wird über einen Lithium-Ionen-Polymer-

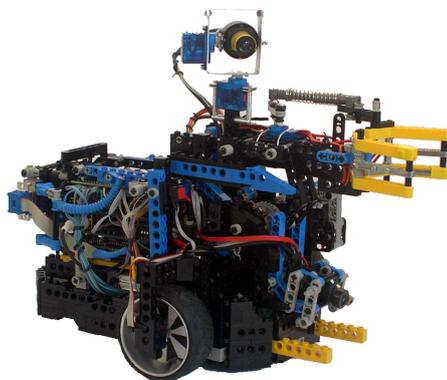


Abbildung 4.4: Beispielapplikation R2D0

Akku mit Strom versorgt und ist so in der Lage, länger zu fahren, als er lädt. Seine primäre Aufgabe besteht im Moment darin, seine Ladestation zu finden. Diese ist durch zwei blaue Bälle gekennzeichnet. Über eine Kamera nimmt der Roboter das Muster wahr und versucht, sich der Ladestation zu nähern. Dabei vermeidet er - über an allen Seiten angebrachte IR-Sensoren -

4 Anhang

Hindernisse.

Die Steuerung ist als State-Machine implementiert. Die Kommunikation wird über ein Shared-Memory-Konzept realisiert. Die momentan längste Zeitdauer, die der Roboter autonom gefahren ist, beträgt ca. 100 Stunden.

Eine weitere Demoapplikation wurde anlässlich der ICIT 2003 in Maribor erstellt. Das AKSEN-Board übernimmt hierbei die Rolle einer Aktorsteuerung. An das LART-Board ist eine Kamera angeschlossen, die auf das projizierte Bild eines Overhead-Projektors gerichtet ist. Das AKSEN-Board steuert über einen Servo eine Signalvorrichtung.

Die Konstruktion soll während eines Vortrags demonstriert werden. Dazu zeichnet der Vortragende ein Bild einer Katze oder eines Hundes auf eine Folie. Der rCube erkennt über bildverarbeitende Algorithmen das projizierte Bild und signalisiert das Ergebnis seiner Beobachtung. Dabei ist die Erkennung (korrekte Zeichnung vorausgesetzt) sehr stabil.

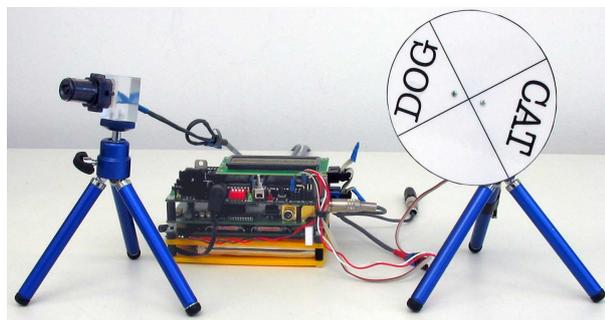


Abbildung 4.5: Demoapplikation R2D1

4.4.2 Bluetooth-Verbindung

Optional kann eine Bluetooth-Verbindung erworben werden, die das serielle Flasher-Kabel zwischen AKSEN-Board und PC ersetzt. Dazu wird ein Bluetooth-Modul an ein Netzgerät und die serielle Buchse des PC angeschlossen und das zweite Bluetooth-Modul an die TAE-Buchse des AKSEN-Boards. Das AKSEN-Board kann über ein mitgeliefertes Kabel diesen Bluetooth-Adapter mit Strom versorgen. Der Stromverbrauch des Bluetooth-Adapter liegt bei ungefähr 70 mA, seine Reichweite bei 10 Metern.

Damit ist es komfortabel möglich, neue Programme auf das AKSEN-Board zu übertragen oder von laufenden mobilen Systemen Laufzeitdaten über die serielle Schnittstelle des AKSEN-Boards zum PC zu schicken und umgekehrt vom PC Befehle entgegen zu nehmen. Die Bluetooth-Module werden in der Konfigurationsphase miteinander bekannt gemacht, so dass sie automatisch eine Verbindung herstellen, sobald sie mit Strom versorgt werden. Das Ersetzen des Flasher-Kabels durch die Funk-Verbindung ist damit sowohl für das AKSEN-Board als auch den PC völlig transparent.

Selbstverständlich kann die Bluetooth-Verbindung auch für die Kommunikation mit den anderen Modulen der RCUBE-Architektur verwendet werden, z.B. mit dem CPU-Board.



Abbildung 4.6: Bluetoothverbindung: Netzgerät, Paarungs-Software, 2 x Bluetooth-Adapter, SV-Kabel BT-AKSEN, Datenkabel BT-AKSEN

4.5 CD-Inhalt

Die CD enthält die komplette Software, die zum Programmieren des AKSEN-Boards unter Linux oder Windows benötigt wird. Im folgenden ist der Inhalt der CD kurz beschrieben:

- Verzeichnis `aksen-lib`
 - Verzeichnis `aksen-lib\beispiele`
Enthält alle im Handbuch vorkommenden Listings. Für jedes Listing existiert ein Verzeichnis gleichen Namens. Diese enthalten jeweils ein passendes Makefile, Quelltexte sowie die compilierten Dateien.
 - Verzeichnis `aksen-lib\lib`
Enthält die aktuelle Version der AKSEN-Bibliothek in compilierter Form, so wie sie auch auf dem AKSEN-Board geflasht ist.
 - Verzeichnis `aksen-lib\include`
Enthält die Header-Dateien, die die Funktionen der AKSEN-Bibliothek deklarieren.
 - Verzeichnis `aksen-lib\stub`
Enthält die zu jedem Programm hinzuzulinkende Objekt-Datei für die AKSEN-Bibliothek.
- Verzeichnis Handbuch
Enthält dieses Handbuch. Weitere relevante Dokumente finden Sie unter `Datenblaetter`

4 Anhang

- Verzeichnis `Linux`
Enthält den Flasher und die aktuelle `sdcc`-Release-Version für Linux
- Verzeichnis `Windows`
Enthält den Flasher, Cygwin-Umgebung mit allen nötigen Paketen und die aktuelle `sdcc`-Release-Version für Windows.

4.6 Bibliotheksversionen

```

0.965: 01-Februar-2005
*****
1. Die Servobank befindet sich nun auf Digital08-15
   (dadurch ist gleichzeitiger Betrieb von Servobank und Infrarot moeglich)

0.964: 12-Januar-2005
*****
1. Bei Makros mod_ir_an() und mod_ir_aus() abschliessendes Semikolon entfernt
2. Globale Interruptsperrre beim Sheduling entfernt, so dass die Servobank dieses
   unterbrechen kann und damit die Servos "ruckelfrei" laufen koennen
3. Interruptprioritaeten neu vergeben (servobank-isr hat hoechste)

0.963: 05-Januar-2005
*****
1. sleep() und interne Funktion timer2() ueberarbeitet (Problembehebung bei Multi-Tasking)

0.962: 14-Dezember-2004
*****
1. Funktionen digital_bytein() und digital_byteout() zugefuegt
2. Beispielprogramm zur Kommunikation über V.24 mit 38.400 Baud zugefuegt

0.961: 23-Juli-2004
*****
1. nochmalige Ueberarbeitung des Multitasking

0.960: 23-Juli-2004
*****
1. Servobank laeuft nun auch im Multitasking
2. Prozesswechsel wurde fuer Prozesse mit geringer Stacktiefe wesentlich beschleunigt
   (Es werden nur noch die "angemeldeten" Bytes auf dem Stack zwischen 0x20 und SP gerettet/geholt)
3. servo() wurde ueberarbeitet, bei laufzeit=0 wird der gewaehlte servo-port auf "Dauer-Null"gestellt

0.959: 14-Juni-2004
*****
1. Encoder-Funktionen ueberarbeitet (Erhoehung der Genauigkeit bei kurzen Intervallen)

0.958: 09-Juni-2004
*****
1. Servobank-Funktionen zugefuegt -> servobank_start(); servobank_stop(); servobank(servo, zeit);
   !!! ACHTUNG - die Servobank vertraegt sich z.Z. noch nicht mit Multitasking !!!
2. die Funktion servod0() wurde aus der Bibliothek entfernt
3. die Funktion sleep() wurde ueberarbeitet (verbesserte Multi-Tasking-Unterstuetzung)

0.957: 26-Mai-2004
*****
1. prozess_hog() ueberarbeitet

0.956: 27-April-2004
*****
1. Analogfunktion nochmals ueberarbeitet

0.955: 15-April-2004
*****
1. zwei Dummyfunktionen in stub.c eingefuegt. Beide sind im Code nur 1 Byte gross (ret-Befehl), "benutzen" aber die
   Registerbaenke 1 und 2, so dass die laestige Warnung ueber 14 freie Byte wegfaellt

0.954: 07-April-2004
*****
1. Alle Bibliotheksvariablen mit festen Adressen im XDATA auf volatile gesetzt

0.953: 31-Maerz-2004
*****
1. Fehler in analog() behoben (Loeschen von ADCON0 hat auch Baudrateneinstellung der Seriellen [ADCON.7,ADCON.6] beeinflusst)

0.952: 15-Maerz-2004
*****
1. prozess_start() ueberarbeitet
2. Umstieg auf sdcc 2.4.0

```

4 Anhang

```
0.951: 10-Maerz-2004
*****
1. Makro process_get_pid() zugefuegt
2. Beispielprog. process2 zugefuegt

0.950: 04-Maerz-2004
*****
1. preemptives Multitasking implementiert
   Funktionen und Makros: process_start(), process_kill(), process_hog(), process_defer(), process_set_ticks

2. IDATA-Beginn wurde auf 0x18 (vorher 0x16) verschoben, muss auch im Makefile des Nutzerprogramms so gesetzt sein
   !!! ---> --data-loc 0x18 <-- !!!

3. Verzeichnisstruktur der Bibliothek geaendert (Test-Progs liegen nun im Unterverzeichnis test)

#####
#####
#####

0.922: 03-Maerz-2004
*****
1. Bibliothek mit dem Schalter --nogcse uebersetzt
   dieser verhindert Optimierungen beim Zugriff auf im XDATA abgelegte globale Variablen
   (leider sind diese Optimierungen im sdcc 2.3.0 fehlerhaft)

2. !!! --nogcse muss auch beim Uebersetzen der Nutzerprogramme benutzt werden !!!

0.921: 05-Februar-2004
*****
1. Anpassung der Makedateien aller Test-Programme an neue Einstellungen (00.913-00.920)
2. Mit dem Aufruf "make test" im Bibliotheksverzeichnis koennen nun alle Test-Programme
   automatisch neu uebersetzt werden. Bei Aenderung der Versionsnummern ist das unbedingt
   erforderlich!

0.920: 03-Februar-2004
*****
1. Aufruf einer Versionskontrolle (Vgl. Bibliotheks- und Stub-Version) in stub eingefuegt
   -> arbeitet sehr restriktiv, Programmstart nur bei absoluter Gleichheit erlaubt
2. Ablage der aktuellen Versionsnummer von Bibliothek und Stub in versionsnummer.h (vorher version.h)
   -> Versionsnummer wird beim Uebersetzen als const jeweils in Bibliothek und Stub eingebunden

0.914: 28-Januar-2004
*****
1. Anordnung der Funktionsdekl. in stub.c geaendert (erst main(), dann die Bibliotheksrount.)
2. Im Unterverzeichnis EXPORT werden nun bei jedem make die zu "exportierenden Dateien" abgelegt
   /EXPORT/HEADER: alle benoetigten Header-Dateien
   /EXPORT/AKSEN: AkSen.rel, AkSen.lst
3. Neue Beispiel-Makedatei in /werbung

0.913: 27-Januar-2004
*****
1. Speicher-Layout wurde geaendert (Data-Variable BP war vorher im Stack-Bereich)
   DATA = 0x16 (fuer Basepointer zum Aufbau des Stackrahmens)
   STACK = 0x20 (der Stack geht von 20h - FFh)

0.912: 26-Januar-2004
*****
1. Timer-ISR laeuft jetzt in Registerbank1
2. Erkennungsroutine fuer Stackueberlauf wurde ueberarbeitet

0.911: 22-Januar-2004
*****
1. Timer-ISR in diversesc wurde um Erkennungsroutine fuer Stackueberlauf ergaenzt

0.910: 19-Januar-2004
*****
1. Timer-ISR in diversesc wurde um Erkennungsroutinen fuer mod. IR (IR0, ...,IR3) ergaenzt
2. Fuer IR0 Makros mod_ir0_takt(), mod_ir0_maxfehler(), mod_ir0_status() zugefuegt
3. Fuer IR1 Makros mod_ir1_takt(), mod_ir1_maxfehler(), mod_ir1_status() zugefuegt
4. Fuer IR2 Makros mod_ir2_takt(), mod_ir2_maxfehler(), mod_ir2_status() zugefuegt
5. Fuer IR3 Makros mod_ir3_takt(), mod_ir3_maxfehler(), mod_ir3_status() zugefuegt
```

4.6 Bibliotheksversionen

3. Makro `ir_senden()` in `digital.h` durch `mod_ir_an()` und `mod_ir_aus()` ersetzt

0.903: 14-Januar-2004

1. Makro `ir_senden()` in `digital.h` zugefügt
2. `Timer1-ISR` um ein-/auschaltbares moduliertes IR-Senden ergaenzt
3. Beispielprogramm fuer serielles Senden ueberarbeitet

0.902: 23-Oktober-2003

1. Fehler in Funktion `analog()` behoben (funktioniert jetzt auch mit SAB 80C515)
2. in Funktion `version()` neue Bibliotheksversion (0.902) eingetragen

0.901: 07-August-2003

1. Funktion `Version()` zur Übergabe der Bibliotheksversion zugefügt
2. Funktionen `servoD0_on()`, `servo_D0_off()`, `servoD0()` und `timer1()` für Ansteuerung eines 4. Servos an Digital 0 zugefügt
3. in Funktion `version()` neue Bibliotheksversion (0.901) eingetragen

Literaturverzeichnis

- [1] A.M.Flynn, J.L. Jones, *Mobile Roboter*, Addison-Wesley, 1996
- [2] I. Boersch, J. Heinsohn, H. Loose, K.-U. Mrkor, *RCUBE A Platform For Intelligent Autonomous Systems*, IEEE Conference on Industrial Technology 2003, Proceedings, pp. 203-206, Maribor/Slowenien, 10.-12.12 2003
- [3] ECONOMIC COMMISSION FOR EUROPE (UNECE), *World Robotics - Statistics, Market Analysis, Forecasts, Case Studies and Profitability of Robot Investment*. Co-authored by the International Federation of Robotics, 2003
- [4] Konrad Eschberger, *Controller Area Network*, Carl Hauser Verlag München, 2002, ISBN 3-446-21776-2
- [5] Fairchild Semiconductor, *Datenblatt zum LM555 Single Timer*, (auf der CD unter Handbuch/Datenblaetter/LM555.pdf)
- [6] Philips, *Datenblatt zum PCA82C251 CAN tranceiver for 24 V Systems*, (auf der CD unter Handbuch/Datenblaetter/PCA82C251.pdf)
- [7] Infineon, *Datenblatt zum BS295 Small Signal Transistor*, (auf der CD unter Handbuch/Datenblaetter/bss295.pdf)
- [8] Siemens, *Datenblatt zum SAB80C515A Microcontroller*, (auf der CD unter Handbuch/Datenblaetter/m80515.pdf)
- [9] Philips, *Datenblatt zum SJA1000 Stand-alone CAN Controller*,(auf der CD unter Handbuch/Datenblaetter/sja1000.pdf)
- [10] Texas Instruments, *Datenblatt zum SN754410 Quadruple Half H-Driver*, (auf der CD unter Handbuch/Datenblaetter/sn754410.pdf)
- [11] Fairchild Semiconductor, *Datenblatt zum TIP 120 Darlington Transistor*, (auf der CD unter Handbuch/Datenblaetter/tip120.pdf)