3.1 Atmega-Programmierung in C/Einführung

3.1.1 Installation

Zunächst müssen bestimmte Pakete installiert sein (als Administrator `root`):

Terminal

```
root@debian964:~# apt-get install gcc-avr binutils-avr avr-libc make
```

3.1.2 Beispielprogramm

Für die RN-Hardware sieht das Beispielprogramm `avrhello.c` so aus:

```
1  #define F_CPU 16000000UL
2  #include <avr/io.h>
3  #include <util/delay.h>
4
5  int main(void)
6  {
7              DDRC|= (1<<DDC7);
8              while (1)
9              {
10                    PORTC &= ~(1<<PC7);
11                    _delay_ms(500);
12                    PORTC|= (1<<PC7);
13                    _delay_ms(500);
14             }
15             return 0;
16 }
```

Und für die CSB-Hardware wie folgt:

```
1  #define F_CPU 18432000UL
2  #include <avr/io.h>
3  #include <util/delay.h>
4
5  int main(void)
6  {
7              DDRB|= (1<<DDB7);
8              while (1)
9              {
10                    PORTB &= ~(1<<PB7);
11                    _delay_ms(500);
12                    PORTB|= (1<<PB7);
13                    _delay_ms(500);
14             }
15             return 0;
16 }
```

Im Vergleich zu einem PC-Programm fällt auf, dass man nicht mehr `<stdio.h>` einbinden muss. Aber das ist verständlich, weil keine Bildschirmausgabe mit `printf()` und keine Tastatureingabe mit `scanf()` gebraucht wird.

Stattdessen wird `<avr/io.h>` eingebunden. Mit dieser Include-Datei werden die Adressen der SFR-Register wie `DDRB` und `PORTB` eingebunden. Aber woher weiß der Compiler z. B. , dass diese Register beim ATMega32 anders liegen als beim ATMega1284p? Bei Assembler-Programm musste man doch zu diesem Zweck die Datei `m32def.inc` oder `m1284pdef.inc` einbinden. Dies wird hier beim Compiler-Aufruf mit der Option `-mmcu=atmega32` verursacht. Durch die Option wird

ein Makro __AVR_ATmega32__ gesetzt, und das wiederum wird in <avr/io.h> abgefragt und die entsprechend passende Datei <avr/iom32.h> eingebunden.

Mit der Datei <util/delay.h> wird u. a. der Header für die Funktion _delay_ms() eingebunden, mit der eine Verzögerungsschleife realisiert werden kann. Damit die Verzögerungszeit stimmt, muss man vor dem Einbinden von <util/delay.h> die symbolische Konstante F_CPU auf den Wert der Taktfrequenz in Hertz setzen.

3.1.3 Compiler

Terminal

```
schueler@debian964:~$ avr-gcc -mmcu=atmega32 -Wall -Os \
> -o avrhello.out avrhello.c
```

Der Compiler heißt nun avr-gcc statt gcc. Man schreibt das Programm auf einem PC und compiliert es dort für einen AVR-Controller. So ein Compiler heißt *Cross-Compiler*. Eigentlich kann der Gnu-C-Compiler gcc schon von Hause aus für alle möglichen CPUs und Mikrocontroller Code erzeugen. Aber der avr-gcc ist einfach speziell für AVR-Systeme angepasst, so dass meine keine besonderen Optionen mehr anzugeben braucht.

Mit der Option -mmcu muss der Typ des Mikrocontrollers angegeben werden (hier -mmcu=atmega32 oder -mmcu=atmega1284p). Mit der Option -O kann man die Art der Optimierung angeben. Die Einstellung -Os bedeutet dabei eine Optimierung hinsichtlich des Speicherbedarfs des erzeugten Codes (*size*).

3.1.4 Hex-Converter

Terminal

```
schueler@debian964:~$ avr-objcopy -O ihex avrhello.out avrhello.hex
```

Das Programm avr-objcopy kann compilierte Dateien von einem Format in ein anderes bringen. Hier soll es die Datei avrhello.out, die im Binärformat vorliegt, in die Datei avrhello.hex konvertieren, die im Intel-Hex-Format (Option -O ihex) sein muss.

3.1.5 Übertragungsprogramm

Terminal

```
schueler@debian964:~$ avrdude -p m1284p -c stk500v2 -P /dev/ttyUSB0 \
> -e -U flash:w:avrhello.hex
```

Für die Übertragung wird wie bisher das Programm avrdude genommen. Die Option -p m32 oder -p m1284p gibt wie gehabt den Controller-Typ an.

3.1.6 Makefile

Mit dem Programm make kann man die Erstellung und Übertragung des Programms automatisieren. Alles, was man dazu braucht, muss in der Datei mit dem Namen Makefile stehen:

```
1  # Makefile avr-gcc
2  ####################
3  ZIEL=avrhello
4  UPROZ=m1284p
5  # UPROZ=m32
6  CPROZ=atmega1284p
7  # CPROZ=atmega32
8  GERAET=stk500v2
9  SCHNITTSTELLE=/dev/ttyUSB0
10
11 all: install
```

```
12
13 $(ZIEL).out: $(ZIEL).c
14         avr-gcc -mmcu=$(CPROZ) -Wall -Os -o $(ZIEL).out $(ZIEL).c
15
16 $(ZIEL).hex: $(ZIEL).out
17         avr-objcopy -O ihex $(ZIEL).out $(ZIEL).hex
18
19 install: $(ZIEL).hex
20         avrdude -p $(UPROZ) -c $(GERAET) -P $(SCHNITTSTELLE) -e \
21         -U flash:w:$(ZIEL).hex:i
22
23 clean:
24         rm -f $(ZIEL).out $(ZIEL).hex
```

Das Makefile hat eine Besonderheit: Die eingerückten Zeilen müssen mit dem Tabulator eingerückt sein, nicht mit Leerzeichen. Andernfalls funktioniert `make` nicht.

Will man dasselbe Makefile auch für ein anderes Programm, z. B. `nocheins.c`, benutzen, muss man die Variable `ZIEL` für das Makefile explizit (Option `-e`) setzen:

Terminal

```
schueler@debian964:~$ make -e ZIEL=nocheins
```