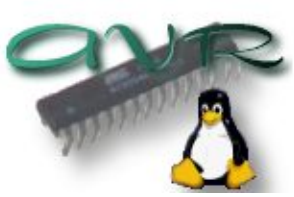
Den AVR Microcontroller mit GCC programmieren



by Guido Socher (homepage)

About the author:

Guido liebt Linux nicht nur, weil es Spaß macht, die großartigen Möglichkeiten dieses Systems zu entdecken, sondern auch wegen den Leuten, die an seiner Entwicklung teilhaben.

Abstract:

Achtung: Es gibt jetzt eine neue Version diese Artikels: November2004/article352.shtml

Der AVR 8−Bit RISC Microcontroller von Atmel ist ein sehr verbreiteter Microcontroller. Er ist ein einfache integrierte Halbleiterschaltung mit EEPROM, RAM, A/D−Wandler, einer Menge digitaler Input− und Outputleitungen, Zeitschaltungen, UART für RS232−Datenübertragung und vielen weiteren Eigenschaften.

Das Beste ist jedoch, daß es unter Linux eine vollständige Programmierumgebung gibt: Man kann diesen Microcontroller unter GCC mit C programmieren. In diesem Artikel werde ich erklären, wie man es installiert und GCC benutzt. Ich werde ebenfalls erklären, wie man die Software auf den Microcontoller lädt. Alles, was man dafür braucht, ist ein AT90S4433 Microcontroller, ein 4MHz−Quarz, etwas Kabel und einige andere sehr billige Teile.

Dieser Artikel wird nur eine Einführung sein. In einem späteren Artikel bauen wir eine LCD−Anzeige mit ein paar Drucktasen, Analog− und Digitaleingabe, einer Hardwarelaufzeitüberwachung und LEDs. Der Plan ist, daraus eine Mehrzweckkonsole für einen Linuxserver zu machen, aber zuerst werden wir lernen, wie man die Programmierumgebung einrichtet. Darüber geht dieser Artikel.

Installation der Software: Was man braucht.

Um die GNU C − Entwicklungsumgebung zu nutzen, braucht man die folgende Software::

binutils−2.11.2.tar.bz2	Zum Herunterladen bei: ftp://ftp.informatik.rwth−aachen.de/pub/gnu/binutils/

	oder ftp://gatekeeper.dec.com/pub/GNU/binutils/
gcc−core−3.0.3.tar.gz	Zum Herunterladen bei: ftp://ftp.informatik.rwth−aachen.de/pub/gnu/gcc/ oder ftp://gatekeeper.dec.com/pub/GNU/gcc/
avr−libc−20020106.tar.gz	Die AVR C−Bibliothek ist vorhanden bei: http://www.amelek.gda.pl/avr/libc/ Man kann sie auch von diesem Server herunterladen: Download
uisp−20011025.tar.gz	Die AVR−Programmierschaltung ist vorhanden bei: http://www.amelek.gda.pl/avr/libc/ Man kann sie auch von diesem Server herunterladen: Download

Wir installieren alle Programme nach /usr/local/atmel. Das machen wir, um das Programm vom normalen Linux−C−Compiler zu trennen. Erzeugen Sie dieses Verzeichnis mit diesem Befehl:

mkdir /usr/local/atmel

Installation der Software: Die GNU binutils

Das Paket binutils liefert alle Low−Level−Hilfsprogrammme, die man braucht, um Objektdateien zu erzeugen. Es enthält einen AVR−Assembler (avr−as), einen Linker (avr−ld), Programme für die Bearbeitung von Bibliotheken (avr−ranlib, avr−ar), Programme, um die Objektdateien zu erzeugen, die auf die EEPROM des Microcontrollers geladen werden können (avr−objcopy), Disassembler (avr−objdump) und Programme wie avr−strip und avr−size.

Führen Sie folgende Befehle aus, um binutils zu erzeugen und zu installieren:

```
bunzip2 −c binutils−2.11.2.tar.bz2 | tar xvf −
cd binutils−2.11.2
./configure −−target=avr −−prefix=/usr/local/atmel
make
make install
```

Fügen Sie die Zeile /usr/local/atmel/lib in die Datei /etc/ld.so.conf ein und führen Sie den Befehl /sbin/ldconfig aus, um den Linker Cache neu zu erzeugen.

Installation der Sofware: Der gcc für den AVR

avr−gcc wird unser C−Compiler sein.

Führen Sie folgende Befehle aus, um es zu erzeugen und zu installieren:

```
tar zxvf gcc−core−3.0.3.tar.gz
cd gcc−core−3.0.3
./configure −−target=avr −−prefix=/usr/local/atmel −−disable−nls −−enable−language=c
make
make install
```

Installation der Software: Die AVR C−Library

Die C−Library wird noch entwickelt. Die Installation kann sich von Version zu Version noch ein bisschen ändern. Ich empfehle, die oben in der Tabelle gezeigte Version, wenn Sie der Anleitung Schritt für Schritt folgen wollen. Ich habe diese Version getestet und sie läuft gut für alle Programme, die wir in diesem und den folgenden Artikeln schreiben werden.

Einige Umgebungsvariablen setzen (Bash−Syntax):
export CC=avr−gcc
export AS=avr−as
export AR=avr−ar
export RANLIB=avr−ranlib
export PATH=/usr/local/atmel/bin:${PATH}

./configure −−prefix=/usr/local/atmel −−target=avr −−enable−languages=c −−host=avr
make
make install

Installation der Software: Programmiersoftware

Die Programmiersoftware lädt den speziell vorbereiteten Maschinencode in das EEPROM unseres Microcontrollers.

Die uisp−Programmiersoftware für Linux ist sehr gut. Sie kann direkt in einem Makefile verwendet werden. Man kann einfach eine "make load"−Regel hinzufügen und in einem Schritt die Software kompilieren und hochladen.

uisp wird wie folgt installiert:

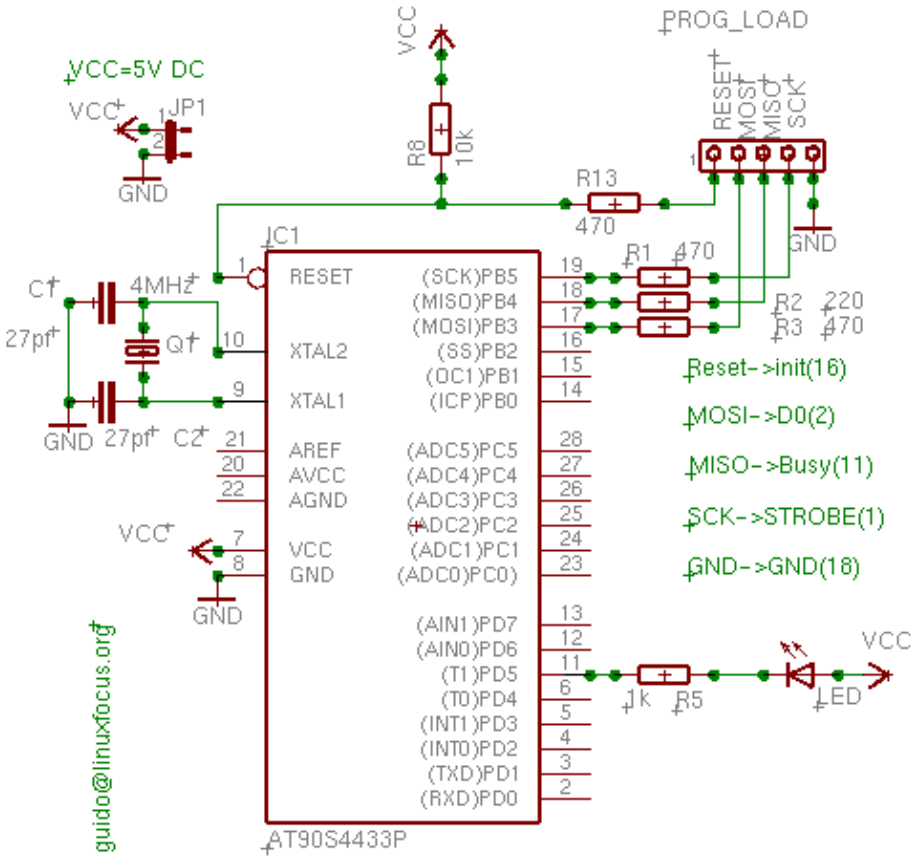
tar zxvf uisp−20011025.tar.gz
cd uisp−20011025/src
make
cp uisp /usr/local/atmel/bin

Ein kleines Testprojekt

Wir fangen mit einer kleinen Testschaltung an. Der Zweck dieser Schaltung ist nur, unsere Entwicklungsumgebung zu testen. Wir können sie benutzen, um zu kompilieren, herunterzuladen und ein einfaches Programm zu testen. Das Programm wird einfach eine LED blinken lassen.

Ich schlage vor, ein kleines bedrucktes Schaltungsbrett für den Microcontroller zu machen. Sie können später diese Schaltung für Ihre eigenen Experimente verwenden. Es ist eine gute Idee, ein Steckbrett dafür zu benutzen. Sie sollten jedoch nicht versuchen, die AVR mit ihrem 4MHz−Quartz direkt auf das Steckbrett zu setzen. Es ist besser, einige kurze Drähte zu verwenden, um die Eingabe− und Ausgabeleitungen mit dem Steckbrett zu verbinden, da solche Steckbretter nicht für schnelle digitale Schaltungen gedacht sind. Der

4MHz−Quarz und die Kondensatoren sollten physisch sehr eng am Microcontroller sein.



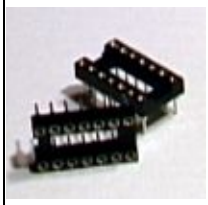
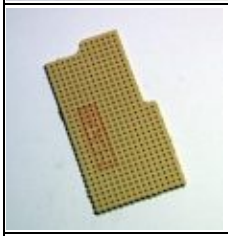
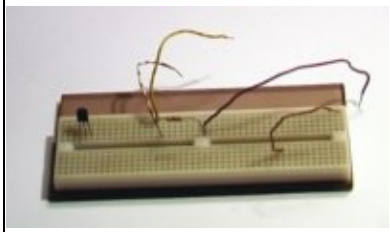
Die Widerstände auf dem Anschluß für die Programmierschaltung werden in unserem Fall eigentlich nicht gebraucht. Sie brauchen sie nur, wenn Sie vorhaben die Port−B Input−/Outputleitungen für andere Zwecke zu verwenden.

Udo Puetz hat einen Schaltplan gezeichnet, der für Anfänger etwas leichter zu lesen ist: avr_layout_newbiefriendly.gif.

Benötigte Hardware

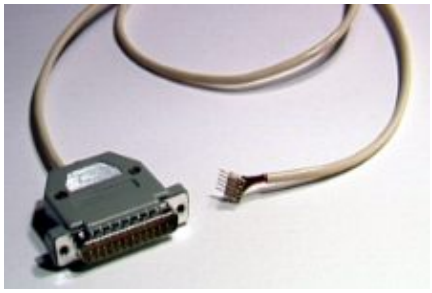
Sie benötigen die Teile der folgenden Tabelle. Davon sind alle sehr verbreitet und billig. Nur der Microcontroller ist ein bisschen teurer, ungefähr 7.50 EUR. Obwohl es ein sehr verbreiteter Microcontroller ist, kann es sein, daß er nicht in jedem örtlichen Elektroladen zu haben ist, aber größere Zwischenhändler für elektronisches Zubehör (wie www.reichelt.de (Deutschland), www.conrad.de (Deutschland), www.selectronic.fr (Frankreich), usw., wahrscheinlich gibt es entsprechende Seiten in Ihrem Land) haben sie alle im Lager.

	1 x AT90S4433, Atmel 8bit AVR RISC−Prozessor.

	2 x 14−Pin IC−Sockel oder 1 x 28−Pin 7.5mm IC−Sockel Den 28−Pin−Sockel bekommt man ein bisschen schwieriger. Üblicherweise sind die 28er−Sockel 14 mm breit, aber wir brauchen einen 7,5 mm breiten Sockel.
	1 x 10K Widerstand (Farbcode: braun, schwarz, orange) 3 x 470 Ohm Widerstand (Farbcode: gelb, violett, braun) 1 x 1K Widerstand (Farbcode: braun, schwarz, rot) 1 x 220 Ohm Widerstand (Farbcode: rot, rot, braun) 1 x 4MHz−Quarz 2 x 27pF Keramikkondensator
	Irgendeinen 5−Pin−Anschluß/−Sockel für die Programmierschaltung. Gewöhnlich kaufe ich diese Steckerleisten und breche 5 davon ab.
	Lochrasterplatine
	1 x DB25−Buchse, für den Parallelport.
	1 x LED
	Ein Steckbrett. Wir benutzen es hier nicht, aber es ist sehr nützlich, wenn Sie weitere Experimente mit dem AVR machen wollen. Ich schlage vor, Sie belassen den Microcontroller zusammen mit dem Quartz und den Kondensatoren auf der Lochrasterplatine und verbinden die Input−/Outputleitungen über kurze Kabel mit dem Steckbrett.

Zusätzlich zu den obigen Teilen brauchen Sie ein 5V, elektronisch stabilisiertes DC−Netzteil oder Sie können eine 4,5V Batterie als Stromversorgung verwenden.

Die Hardware der Programmierschaltung zusammenbauen



Der AT90S4433 erlaubt In Circuit Programming (ICP).
Das heißt: Sie müssen den Microcontroller nicht aus der Schaltung nehmen, um ihn zu programmieren. Sie werden merken, daß Sie fertige Programmier−Hardware für 50−150 EUR kaufen können. Sie müssen nicht soviel in eine Programmierschaltung investieren. Mit Linux, der uisp−Software und einem freien Parallelport können Sie eine sehr gute und einfache AVR−Programmierschaltung bauen. Es ist ein einfaches Kabel. Die Verkabelung der Programmierschaltung muß wie folgt sein:

Pin am AVR	Pin am Parallelport
Reset (1)	Init (16)
MOSI (17)	D0 (2)
MISO (18)	Busy (11)
SCK (19)	Strobe (1)
GND	GND (18)

Das Kabel sollte nicht länger als 70 cm sein.

Software schreiben

Der AT90S4433 kann mit der Hilfe von gcc einfach in C programmiert werden. Ein bisschen AVR−Assembler zu können ist nützlich, aber wird nicht gebraucht. Die AVR−libc hat die avr−libc−reference, die die meisten Funktionen dokumentiert. Harald Leitner hat ein Dokument mit vielen nützlichen Beispielen geschrieben, wie man den AVR und GCC verwendet (haraleit.pdf, 286Kb, ursprünglich von http://www.avrfreaks.net/AVRGCC/). Von Atmel's Webseite, (www.atmel.com, gehen Sie nach: avr products −> 8 bit risc−> Datasheets), Sie können die vollständige Spezifikation runterladen (lokale Kopie: avr4433.pdf, 2361Kb). Sie beschreibt alle Register und wie man die CPU nutzt.

Eines, was man im Hinterkopf behalten sollte, wenn man den 4433er benutzt ist, daß er nur 128 Bytes RAM und 4KB EEPROM hat. Das bedeutet, daß Sie keine großen Datenstrukturen oder Strings deklarieren dürfen. Ihr Programm sollte keine tief verschachtelten Funktionsaufrufe oder Rekursion haben. Eine Zeile wie
char string[90];
ist schon zuviel. Ein Integer hat 16 Bit. Wenn Sie ein Small Integer brauchen, dann benutzen Sie
unsigned char i; /* 0−255 */
Sie werden dennoch überrascht sein, was für große Programme man schreiben kann. Es ist ein wirklich mächtiger Prozessor!

Viel besser als alle Theorie ist ein echtes Beispiel. Wir schreiben ein Programm, daß unsere LED veranlaßt in Intervallen von 0,5 Sekunden zu blinken. Es ist nicht sehr nützlich, aber gut, um anzufangen und die Entwicklungsumgebung und die Programmierschaltung zu testen.

```
void main(void)
{
    /* enable PD5 as output */
    sbi(DDRD,PD5);
```

```
    while (1) {
        /* led on, pin=0 */
        cbi(PORTD,PD5);
        delay_ms(500);
        /* set output to 5V, LED off */
        sbi(PORTD,PD5);
        delay_ms(500);
    }
}
```

Der obige Programmierschnipsel zeigt wie einfach es ist, ein Programm zu schreiben. Sie sehen nur das Hauptprogramm. Die delay_ms−Funktion ist im vollständigen Listing (avrledtest.c) enthalten. Um den Pin PD5 als Output zu nutzen, müssen Sie das PD5−Bit im Data Direction Register für D−Anschluß (DDRD) setzen. Danach können Sie PD5 mit der Funktion cbi(PORTD,PD5) (clear bit PD5) auf 0V oder auf 5V mit sbi(PORTD,PD5) (set bit PD5) setzen. Der Wert von "PD5" ist in io4433.h definiert, das mit io.h einbezogen ist. Sie müssen sich keine Gedanken darüber machen. Wenn Sie bereits Programme für Multiuser−/Multitaskingsysteme wie Linux geschrieben haben, wissen Sie, daß man niemals eine nichtblockierende Endlosschleife programmieren darf. Dies wäre eine Verschwendung von CPU−Zeit und würde das System sehr verlangsamen. Beim AVR ist dies anders. Wir haben nicht mehrere Aufgaben und es läuft kein anderes Programm. Es gibt nicht einmal ein Betriebssystem. Es ist daher recht normal, ein Busy Loop für immer laufen zu lassen.

Kompilieren und Hochladen

Bevor Sie anfangen, stellen Sie sicher, daß Sie /usr/local/atmel/bin in der PATH−Variablen haben. Wenn es nötig ist, dann ändern Sie Ihre .bash_profile oder .tcshrc und fügen folgendes hinzu:

export PATH=/usr/local/atmel/bin:${PATH} (bei bash)
setenv PATH /usr/local/atmel/bin:${PATH} (bei tcsh)

Wir benutzen den Parallelport und uisp, um den AVR zu programmieren. Uisp benutzt die ppdev−Schnittstelle des Kernels. Daher müssen Sie folgende Kernelmodule laden:

```
# /sbin/lsmod
parport_pc
ppdev
parport
```

Überprüfen Sie mit dem Befehl /sbin/lsmod, daß Sie geladen sind. Ansonsten laden Sie sie (als root) mit

```
modprobe parport
modprobe parport_pc
modprobe ppdev
```

Es ist eine gute Idee, diese Befehle automatisch während dem Hochfahren auszuführen. Sie können Sie zu dem rc−Skript (z.B. bei Redhat /etc/rc.d/rc.local) hinzufügen.
Um die ppdev−Schnittstelle als normaler Benutzer zu nutzen, muß root Ihnen Schreibzugriff geben, indem er einmal den Befehl

```
chmod 666 /dev/parport0
```

ausführt. Stellen Sie ebenso sicher, daß kein Drucker−Daemon auf dem Parallelport läuft. Wenn einer läuft, dann stoppen Sie es, bevor Sie das Kabel der Programmierschaltung anschließen. Jetzt ist alles fertig, um zu kompilieren und unseren Microcontroller zu programmieren.

Das Paket für unser Testprogramm (avrledtest−0.1.tar.gz) beinhaltet ein Makefile. Sie müssen nur folgendes tippen:
make
make load
Dies wird die Software kompilieren und hochladen. Ich gehe nicht tiefer auf alle Befehle ein. Sie können sie im Makefile sehen und sie sind immer gleich. Ich kann mich selber nicht an alle von ihnen erinnern. Ich weiß nur, daß ich "make load" ausführen muß. Wenn Sie ein anderes Programm schreiben wollen, dann ersetzen Sie einfach alle vorkommenden avrledtest im Makefile mit dem Namen Ihres Programmes.

Einige interessante binutils

Interessanter als die tatsächliche Kompilierung sind einige der binutils.

avr−objdump −h avrledtest.out

Zeigt die Größe der verschiedenen Abschnitte in unserem Programm. .text ist Befehlscode und lädt in das Flash EEPROM. .data sind initialisierte Daten wie
static char str[]="hello";
und .bss sind uninitialisierte globale Daten. Beide sind in unserem Fall null. .eeprom ist für Variablen, die in eeprom gespeichert sind. Ich hatte dafür noch nie Verwendung. stab und stabstr sind Debugging−Informationen und kommen nicht bis in den AVR.

```
avrledtest.out:     file format elf32-avr

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000008c  00000000  00000000  00000094  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  00800060  0000008c  00000120  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00800060  0000008c  00000120  2**0
                  ALLOC
  3 .eeprom       00000000  00810000  00810000  00000120  2**0
                  CONTENTS
  4 .stab         00000750  00000000  00000000  00000120  2**2
                  CONTENTS, READONLY, DEBUGGING
  5 .stabstr      000005f4  00000000  00000000  00000870  2**0
                  CONTENTS, READONLY, DEBUGGING
```

Sie können ebensogut den Befehl avr−size ausführen, um dies in einer kompakteren Form zu bekommen:

avr−size avrledtest.out

```
   text    data     bss     dec     hex filename
    140       0       0     140      8c avrledtest.out
```

Wenn Sie mit dem AVR arbeiten, müssen Sie darauf achten, daß text+data+bss nicht mehr als 4k ergibt und data+bss+stack (Sie können die Größe des Stack nicht sehen, es hängt davon ab, wieviele vernestete Funktionsaufrufe Sie haben) darf nicht mehr als 128 Bytes sein.

Ebenso interessant ist der Befehl

avr−objdump −S avrledtest.out

Er generiert ein Assemblerlisting Ihres Codes.

Fazit

Jetzt wissen Sie genug, um Ihre eigenen Projekte mit der AVR−Hardware und dem GCC anzufangen. Ebenso wird es weitere Artikel in LinuxFocus geben, mit komplexerer und interessanterer Hardware.

Quellen

- Libc und uisp: /www.amelek.gda.pl/avr/libc/
- GCC und binutils: ftp://gatekeeper.dec.com/pub/GNU/
- avrfreaks (passen Sie auf, manche Leute auf dieser Website benutzen immer noch Windows !?): http://www.avrfreaks.net/
- Der tavrasm−Assembler für Linux: www.tavrasm.org
- AVR Webring: R.webring.com/hub?ring=avr&list
- Vorkompilierte Versionen von gcc: combio.de/avr/
- Die ganze Software und alle Dokumente, die in diesem Artikel erwähnt werden
- Die atmel Website: www.atmel.com/

2005−01−11, generated by lfparser_pdf version 2.51