Teil 2 –– Ein digitales Thermomether order rede I2C mit deinem Microcontroller

von Guido Socher (homepage)

Über den Autor:

Guido mag Linux weil es ein wirklich gutes System ist, um eigene Hardware zu entwickeln.

Übersetzt ins Deutsche von:
Guido Socher (homepage)

Zusammenfassung:

In diesem zweiten Teil werden wir ein LCD display in unser Thermomether einbauen und ich werde erklären wie die Software funktioniert.

Die Leser, die neu hinzugekommen sind, sollten zuerst den ersten Teil (Februar 2005 Artikel 365) lesen.

Die neuen Teile

In dem vorrangegangenen Artikel haben wir schon den wichtigsten Teil der Hardware, die nötig war um Temperaturen zu messen, aufgebaut. In diesem Artikel gibt es nun ein sehr einfaches gtk GUI und ein LCD display.

Es ist sehr einfach diese beiden Dinge hinzuzufügen, deshalb werde ich den Rest des Artikels darauf verwenden die I2C Software und die Analogdigitalwandler Software zu erklären.

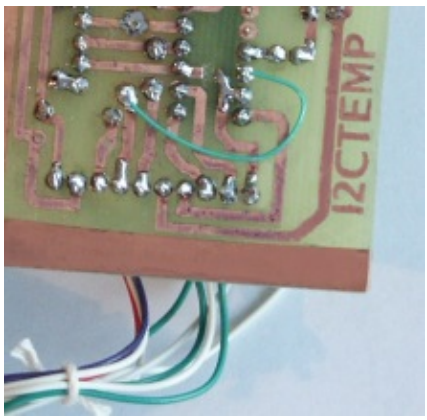
Das LCD–Display

Als Display benugtzen wir ein HD44780 kompatibles Display, wie ich es schon in älteren Artikeln eingesetzt hatte. Diese Displays sind sehr einfach in Verbindung mit Microcontrollern zu benutzen, da man einfach ASCII Zeichen an das Display schicken kann.

Wie für alle anderen Artikel in dieser Serie gibt es auch hier wieder die Bauteile bei shop.tuxgraphics.org
Ich benutze den gleichen LCD−Treiber wie in den älteren Artikeln. Die Dateien, die den Treiber implementieren sind lcd.c, lcd.h und lcd_hw.h. Sie sind Teil des Paketes, welches sich am Ende des Artikels herrunterladen läßt. Das Interface für diesen Code ist wirklich einfach zu benutzen:

```
// call this once:
// initialize LCD display, cursor off
lcd_init(LCD_DISP_ON);

// to write some text we first clear
// the display:
lcd_clrscr();
lcd_puts("Hello");
// go to the second line:
lcd_gotoxy(0,1);
lcd_puts("LCD");
```



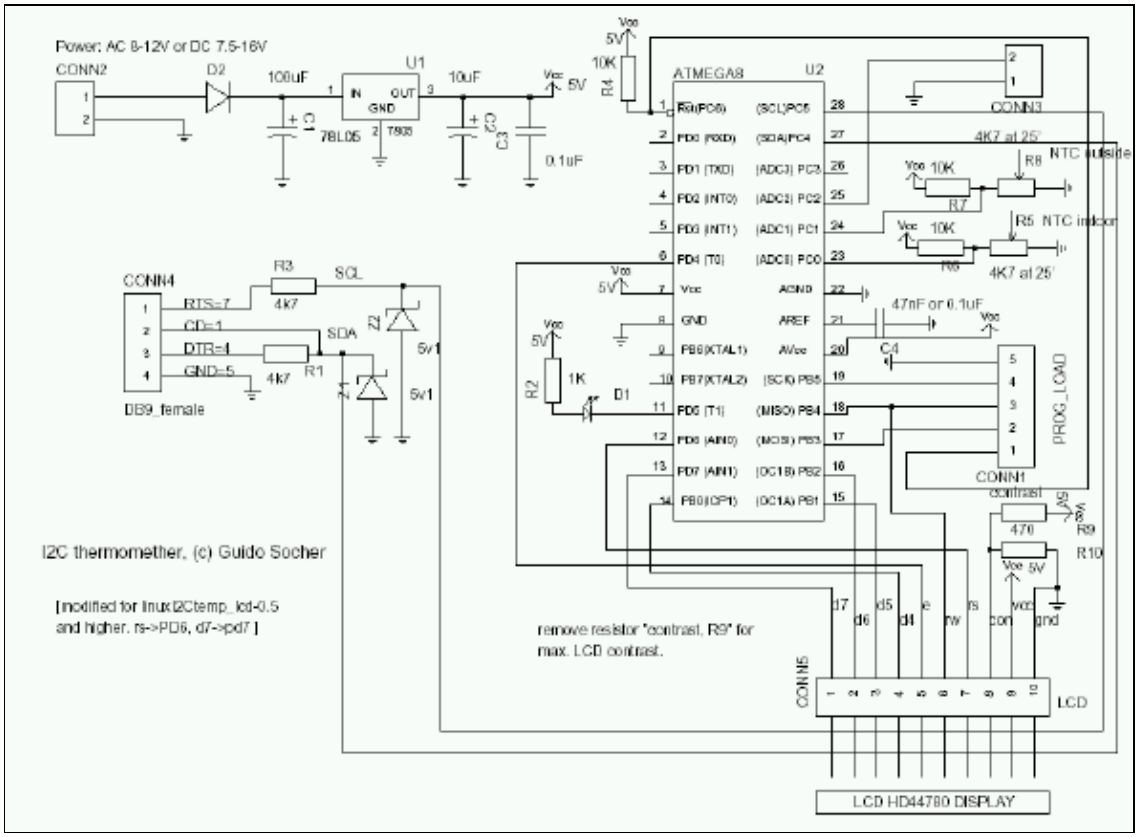
Wie diese HD44780 Displays genau funktionieren ist in dem linuxfocus September2002 Artikel "HD44780 kompatible LCD−Displays verstehen" erklärt.

Die Software ist so geschrieben, daß sie mit 16x2 und 20x2 Zeichen Displays funktioniert.

Es gibt leider im Schaltbild ein Änderung. Ich habe entdeckt, daß einige LCD Displays einen höheren kapazitiven Innenwiderstand haben als andere. Das ist vermutlich so, weil sie einen besseren ESD Schutz haben. Diese höhere Last kann Bit−Errors beim Programmieren des Microcontrollers verursachen, wenn das Display an die SCK und MOSI Leitungen des Microcontrollers angschlossen ist.

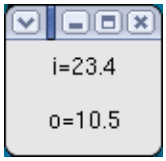
Als erste Lösung für dieses Problem habe ich einfach ein paar Widerstände in die Leitungen zum Display eingelötet. Das funktionierte problemlos bei mir, aber einige Leser, speziell Leute mit Laptops hatten immer noch Probleme.

Um das Problem vollständig zu vermeiden habe ich dann das Schaltungsdiagramm geändert und die D7 und RS Pins des Display an PD7 und PD6 angeschlossen. Es ist kein Problem das auf der Platine zu ändern. Dazu ist lediglich ein kurzes Stück Draht unter der Platine nötig und die Verbindung zu PB3 muß mit einem Messer unterbrochen werden.



Ein einfaches GUI

Für die Leser, die gerne ein GUI auf Ihrem Desktop haben, habe ich ein wirklich einfaches GUI entwickelt. Es besteht aus 2 Textzeilen die die Ausgabe es Befehls i2ctemp_linux wiedergeben (das GUI ruft einfach den Befehl i2ctemp_linux auf und dieser liest die Temperatur via I2C).



Jetzt haben wir ein richtig cooles Thermomether:

- Man kann die Temperatur lokal am Display ablesen.
- Man kann ein kleines GUI auf seinem Desktop haben.
- Man kann die Temperaturwerte über einen cron−job in eine Datei schreiben und damit Statistiken über längere Zeit haben.

Den Rest des Artikels werde ich jetzt damit verwenden die Interna der Software zu erklären.

Wie es funktioniert: Analogdigitalwandler

Der Atmega8 hat zwei Modi. Einen in dem er permanent analog zu digital wandelt und immer einen Interrupt auslöst, wenn ein Konvertierung fertig ist. Die Applikation kann dann dem Interrupt benutzen um schnell des Ergebnis aus den Registern in Variablen zu kopieren.

In dem anderen Modus, der sich "single shot mode" nennt wird nur ein Konvertierung durchgeführt. Dieser Mode ist immer noch recht schnell. Mit der Zeit, die man braucht um alle Voreinstellungen richtig zu setzen kann man immer noch 100 Konvertierungen pro Sekunde erreichen. Das ist mehr als schnell genug für unsere Zwecke. Wir nehmen deshalb diesen Modus.

Als Eingänge kann der Atmega8 die Pins ADC0 bis ADC3 benutzen. Daneben gibt es noch die Pins AGND (analog ground, Verbunden mit der normalen Masse), AREF (reference voltage) und AVCC (verbunden mit +5V).

Während der AD−Wandlung wird das analoge Signal mit AREF verglichen. Ein analoges Signal das gleich mit AREF ist, entspricht einem digitalen Wert von 1023. AREF kann eine externe Referenz zwischen 0 und 5V sein. Ohne externe Referenz kann man immer noch exakte Messungen machen, indem man entweder die interne Referenzspannung (2.56V), oder AVCC benutzt. Welche dieser Optionen benutzt werden soll, läßt sich über die REFS0 und REFS1 Bits im ADMUX Register einstellen.

Der Analogdigitalwandler kann zu einem Zeitpunkt nur einen der Eingänge ADC0−ADC3 konvertieren. Bevor die Konvertierung beginnt muß man festlegen, welcher Eingang benutzt werden soll und das geht auch über Bits im ADMUX Register.

Eine einfache AD−Wandlung würde dann so aussehen:

```
volatile static int analog_result;
volatile static unsigned char analog_busy;

analog_busy=1; // busy mark the ADC function
channel=0; // measure ADC0
// use internal 2.56V ref
outp((1<<REFS1)|(1<<REFS0)|(channel & 0x07),ADMUX);
outp((1<<ADEN)|(1<<ADIE)|(1<<ADIF)|(1<<ADPS2),ADCSR);
sbi(ADCSR,ADSC); // start conversion
```

Jetzt wird der Microcontroller die AD−Wandlung durchführen und dann die Funktion SIGNAL(SIG_ADC) aufrufen sobald die AD−Wandlung fertig ist. Als Programmierer muß man darauf achten, daß man die unteren 8 Bits der Ergebisregister immer zuerst lesen muß.

```
SIGNAL(SIG_ADC) {
        unsigned char adlow,adhigh;
        adlow=inp(ADCL); /* read low first, two lines. Do not combine
                            the two lines into one C statement */
        adhigh=inp(ADCH);
        analog_result=(adhigh<<8)|(adlow & 0xFF);
        analog_busy=0;
}
```

Jetzt haben wir das Ergebnis in der Variablen analog_result. Dieser Wert kann nun anderswo im Programm benutzt werden. Ganz einfach.

Wie bei allen Interrupts muß man einmal global sei(); aufrufen um Interrupts einzuschalten. Das wird im Haupprogramm gemacht und ist deshalb hier nicht aufgeführt.

Es gibt eine Menge Bits und Flags, die ich kurz erklären werde:

- ADEN: Analog Digital Converter Enable, muß vor ADSC gesetzt werden
- ADIE: Enable ADC Interrupt (=schaltet Interrupts ein, daruch wird der Aufruf von SIGNAL(SIG_ADC) möglich)
- ADIF: ADC Interrupt Flag (muß auf 1 vor der Konvertierung gesetzt werden)
- ADPS: ADC clock pre−scaler bits: Muß so eingestellt werden, das die Taktfrequenz geteilt durch den pre−scale Faktor einen Wert zwischen 50 und 200 kHz ergibt. Der pre−scale Faktor ist 2^ADPS (zwei hoch ADPS Bits Wert). Die oben benutzte Einstellung (ADPS2=1, ADPS1=0, ADPS0=0 = dezimal 4 −> 2^4 = 16 −> Teilerfaktor = 16) ist gut für eine Taktfrequenz von 1MHz.

Der Atmega8 hat verschiedene Möglichkeiten was die Referenzspannung betrifft:

REFS0=0, REFS1=0	benutze eine externe AREF Spannung, die interne Vref ist aus
REFS0=0, REFS1=1	AVCC mit optionalen Kondensator an dem AREF Pin
REFS0=1, REFS1=1	interne 2.56V Referenz with (optionalem) externem Kondensator an dem AREF Pin

Der optionale externe Kondensator am AREF Pin kann rauschen unterdrüchen und die AREF Spannung stabilisieren.

Wie es funktioniert: I2C Kommunikation, Atmega8 Teil

Im Teil I (Februar 2005 Artikel365) habe ich schon erklärt wie das I2C Protokoll funktioniert. Jetzt wollen wir uns die Software etwas näher ansehen. Der Atmega8 hat Hardwareunterstützung für I2C Kommunikation. Man muß deshalb das I2C Protokoll garnicht implementieren. Stattdessen schreibt man eine Zustandsmaschine, die dem Atmega8 sagt was als nächstes getan werden soll. Hier ein Beispiel:

Ein I2C Paket mit unser eigenen Slave−Adresse ist angekommen. Der Atmega8 wird nun die Funktion SIGNAL(SIG_2WIRE_SERIAL) mit dem Statuscode 0x60 (für andere Ereignisse gibt es andere Codes) aufrufen.

−−> Jetzt müssen wir einige Register setzen, um dem Atmega8 zu sagen, was als nächstes zutun ist. In diesem Fall sagen wir ihm: Empfange den Datenteil und bestätige mit einem ack−Bit.

Wenn die eigentlichen Daten empfangen wurden, werden wir mit dem Statuscode 0x80 aufgerufen.

−−> Nun lesen wir das Datenbyte und sagen dem Atmega8, daß er auch das nächste empfangen und bestätigen soll.

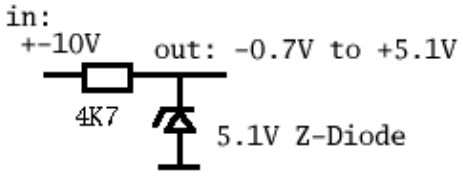
Wenn die Kommunikation zuende ist, erhalten wir den Statuscode 0xA0 (stop condition) und wir können unser Applikation sagen, daß eine vollständige Nachricht empfangen wurde.

Die gesamte Zustandsmaschine für den I2C Slave und alle möglichen Statuscodes sind im Datenblatt Seite 183 erklärt (siehe Link unter Referenzen am Ende des Artikels).

Das Senden von Daten funktioniert ganz änlich. Schau einfach mal in den Code rein!

Wie es funktioniert: I2C Kommunikation, Linux Seite

Zunächst einige Worte zur Hardware. Obwohl I2C ein Bus ist, benutzen wir nur Punkt zu Punkt Kommunikation zwischen einem Slave und dem Linux PC als Master. Wir können uns deshalb die "pullup" Widerstände sparen, solange der Slave die Leitung gegen Masse ziehen kann, ohne eine Kurzschluß zu verursachen. Wir setzen deshalb die 4.7K Widerstände in serie in die Leitung.



Die Spannungspegle müssen angepasst werden. Das machen wir mit Z−Dioden, die negative Spannungen auf −0.7V und positive auf max. +5.1V begrenzen.

Nachdem ich inzwischen mehr über die Internas des Atmeag8 gelesen habe, denke ich daß man sich eigentlich die Z−Dioden sparen könnte, da der Strom über die 4.7K klein genug ist, damit auch die internen Schutzvorrichtungen des Atmeag8 gegen Überspannung ausreichen müßten. Die Z−Dioden schaden aber auch nicht. In der nächsten Schaltung nehme ich sie vielleicht raus.

Die Linux I2C Software implementiert einen kompletten I2C Stack. Dadurch ist die Linux−Seite unabhängig von irgenwelchen Kernelmodulen und Bibliotheken. Es ist einfach ein kleines Programm das unabhängig und problemlos funktioniert.

Wenn man sich die Datei i2c_m.c ansiehst (siehe Download) dann stellt man fest, daß wirklich jede I2C Nachricht Bit für Bit zusammengebaut wird.

Um die "Bits" zu erzeugen müssen wir die physikalischen Leitungen der RC232 Schnittstelle setzen. Das geht über ioctl Aufrufe:

```
// set RTS pin:
int arg=TIOCM_RTS;
ioctl(fd, TIOCMBIS, &arg);
```

... oder um eine "Null" zu erzeugen:

```
// clear RTS pin:
int arg=TIOCM_RTS;
ioctl(fd, TIOCMBIC, &arg);
```

Falls man diesen Code auf ein anderes Betriebsystem portieren möchte, braucht man lediglich diese Zeilen zu ändern. Der Rest ist ganz normaler ANSI C−Code.

USB zu RS232

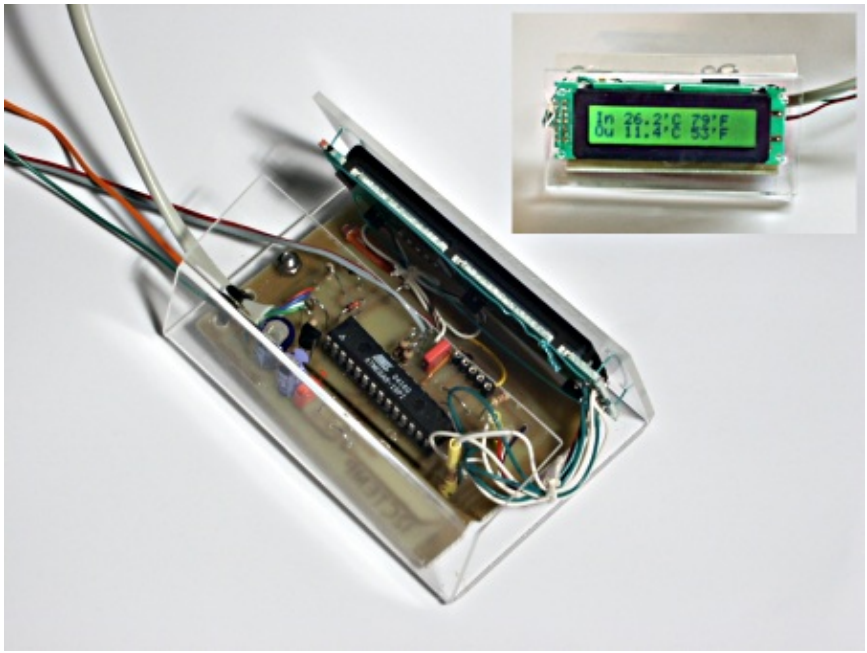
Bei Laptops, die möglicherwiese heute keine RS232 Schnittstelle mehr haben, kann man einfach einen USB zu RS232 Adapter nehmen. Ich benutze, z.B eine No−name Adapter der einen Prolific 2303 Chip enthält. In /proc/bus/usb/devices sieht das bei mir so aus: Vendor=067b ProdID=2303 Rev= 2.02. Näheres zu solchen Adaptern findet sich auch in "Einen ATEN UC−232A USB Adapter unter Linux nutzen (Linuxfocus,

November 2001, Artikel 223)".

Zusammenfassung

Ich benutze das Thermometer nun seit 2 Monaten und es gefällt mir wirklich gut. Ich kann die Temperatur direkt ablesen und die Daten über den PC auswerten.

Der Außensensor muß gut gegen Regen und Sonne geschützt sein. Man kann versuchen ihn in irgendwelches Plastik einzuwickeln, aber ich empfehle das nicht. Egal wie dicht man es verpackt, eines Tages kommt etwas Wasser hinein und kann dann nicht mehr raus. Der NTC ist recht robust und es macht ihm eigentlich nichts aus, wenn er mal für kurze Zeit feucht wird, solange er dann auch wieder trocknen kann. Eine umgedrehte Tablettendose eigent sich ganz gut. Ich habe sie unten offen gelassen und so kann das Wasser auch wieder raus.



Auch für diesen Artikel bekommt man wieder alle Teile (LCD Display, Platine, Microcontroller, ...) über den tuxgraphics Online−shop: shop.tuxgraphics.org.
Viel Spaß!

Referenzen

- **Download** Seite für diesen Artikel: die linuxI2Ctemp_lcd Software, Schaltbilder, Softwareupdates
- Wie man den Atmeag8 mit gcc programmiert: November2004 Artikel 352
- Datenblatt für den Atmega8: Gehe zu http://www.atmel.com/ und selektiere products−>Microcontrollers −>AVR−8 bit RISC−>Documentation−>datasheets (lokale Kopie, pdf, 2479982 Bytes)

Der LinuxFocus Redaktion schreiben http://www.LinuxFocus.org	Autoren und Übersetzer: en −−> −− : Guido Socher (homepage) en −−> de: Guido Socher (homepage)

2005−03−26, generated by lfparser_pdf version 2.51