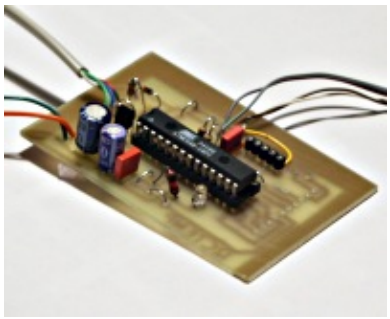
Ein digitales Thermometer oder rede I2C mit deinem Microcontroller

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:

Der Atmega8 Microcontroller von Atmel hat jede Menge digitale und analoge Ein−/Ausgänge. Es ist das ideale Bauteil um Meßelektronik zu entwickeln.

In diesem Artikel werden wir sehen wie man einen Microcontroller mit einem Linux PC über eine physikalische RS232 Schnittstelle verbinden kann, ohne aufwendige Adapterschaltungen wie den MAX232 Chip zu benutzen.

Einführung

Eine Voraussetzung für diesen Artikel ist, daß du die GCC AVR Programmierumgebung, wie in den dem Artikel "Programmieren eines AVR Microcontrollers mit GCC, libc 1.0.4" beschrieben, installiert hast. Natürlich kannst du, um Ärger bei der Installation zu vermeiden, auch die "AVR programming CD" von http://shop.tuxgraphics.org/ benutzen.

Wenn man so ein enorm mächtiges Bauteil wie einen Microcontroller benutzt, um analoge oder digitale Signale zu messen, dann möchte man natürlich Schnittstellen zum Auswerten haben und Befehle an den Microcontroller schicken können. In allen vorangegangenen Artikeln haben wir immer die RS232 Schnittstelle mit einem UART zur Kommunikation benutzt. Der Nachteil ist hierbei, daß man einen zusätzlichen MAX232 Chip und 4 Kondensatoren braucht. Atmel schlägt sogar vor, daß man einen externen Quarz für den Oszillator benutzen soll damit die UART Kommunikation zuverlässig funktioniert. In jeden Fall sind das eine Menge extra Bauteile.... und diese kann man vermeiden!

Die Menge an Daten, die zwischen PC und Microcontroller übertragen werden müssen ist normalerweise klein (nur einige Bytes). Die Übertragungsgeschwindigkeit spielt daher praktisch keine Rolle. Das macht den I2C Bus attraktiv für diese Aufgabe.

I2C (ausgesprochen "ei−squär−sie") ist ein bidirektionales Kommunikationssystem mit nur zwei Drähten. Es wurden von Philips entwickelt und sie haben sich den Namen schützen lassen. Daher nennen es andere Hersteller nicht I2C. Bei Atmel heißt es TWI ("two wire interface").

Viele von euch benutzen möglicherweise I2C in ihrem PC ohne es zu wissen. Alle modernen Motherboards haben einen I2C−Bus um Temperatursensoren, Lüftergeschwingigkeit, verfügbaren Speicher... und alle möglichen anderen Daten zu lesen. Dieser I2C−Bus ist leider am normalen PC nicht nach außen geführt (es gibt außen keine physikalischen Anschluß dafür). Wir müssen uns deshalb etwas neues überlegen.

Erst wollen wir aber sehen, wie das "two wire interface" (=TWI = anderer Name für I2C) funktioniert.

Wie I2C/TWI funktioniert

Das Datenblatt des Atmega8 (siehe unten) hat eine sehr detaillierte Beschreibung, die auf Seite 160 beginnt. Ich werde deshalb hier nur einen Überblick geben. Danach solltest du in der Lage sein, das Datenblatt zu verstehen.

An einem I2C−Bus hat man immer einen "Master" und verschiedene "Slaves". Die Kommunikation findet zu einem Zeitpunkt immer zwischen dem Master und genau einem Slave statt. Der Master ist das Gerät, das die Kommunikation beginnt und und den Bit−Takt (clock) kontrolliert. Die zwei Drähte des I2C−Bus nennen sich SDA (Datenleitung) und SCL (Clock line, Taktleitung). Jedes der Geräte muß von einer separaten Stromversorgung (genau wie bei normaler rs232 Kommunikation) gespeist werden. Die zwei Drähte des Busses sind normalerweise über 4.7K Widerstände mit der logischen Eins (+5V für 5V ICs) verbunden. Das ergibt eine elektrische Veroderung zwischen den Geräten. Ein Gerät braucht dann nur die Leitung auf GND ziehen, wenn es eine 0 übertragen will oder nichts machen, wenn es eine 1 senden will.

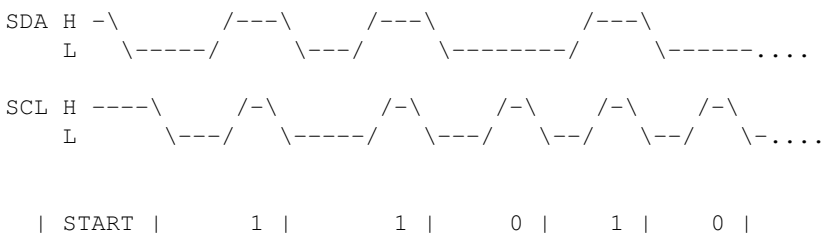
Der Master beginnt einen Dialog indem er ein Bitmuster, das als "start condition" gefolgt von der Adresse des Gerätes das er ansprechen möchte, schickt. Jedes Gerät an dem Bus hat eine eindeutige 7 Bit Adresse. Danach schickt der Master ein Bit das anzeigt ob er lesen oder schreiben möchte. Der Slave (das angesprochene Gerät) wird nun bestätigen, daß es den Master verstanden hat und schickt ein ack−bit (ack wie acknowledge). Mit anderen Worten wir haben jetzt 9 Datenbits auf dem Bus gesehen (7 Adressbits + read_bit + ack−bit):

```
| start | 7-bit slave adr | read_data bit | wait for ack | ... data comes here
```

Was kommt danach?

Als nächstes können wir Daten senden oder empfangen. Die Daten sind immer ein Vielfaches von 8 Bit (1 Byte) und müssen durch ein Ack−Bit bestätigt werden. Es werden also immer 9 Bit über den Bus geschickt. Wenn die Kommunikation zu ende ist, schickt der Master ein Bitmuster namens "stop condition". Mit anderen Worten der Master muß wissen, wie viele Daten der Slave schicken möchte. Das ist aber kein Problem, denn man kann diese Information immer in den Dialog zwischen Master und Slave mit den Daten übertragen. In unserem Fall werden wir z.B. das Null−Byte am Ende eines Strings benutzen um anzuzeigen, daß der übertragene String zu ende ist.

Die Daten auf dem SDA Draht sind immer solange gültig wie SCL 1 ist. SDA darf sich also nicht ändern während SCL 1 ist:



Das Beste an diesem Protokoll ist, daß man kein genaues synchrones Taktsignal braucht. Das Protokoll wird immer noch funktionieren wenn der Takt etwas wackelt (Jitter).

Genau diese Eigenschaft macht es möglich, I2C in einer ganz normalen Benutzerapplikation zu implementieren. Man braucht kein Kernelmodul oder spezielle Hardware (wie UART). Genial oder?

Die Idee

Wie schon gesagt können wir den PC internen I2C Bus nicht benutzen. Stattdessen benutzen wir einfach einige Leitungen der RS232 Schnittstelle. Unsere Kommunikationsschnittstelle ist also immer noch RS232 aber wir brauchen keine MAX232 Hardware, keine Kondensatoren, ... etc

Der schwierige Teil ist nun das I2C Protokoll neu in einer Linuxapplikation zu implementieren und zu testen. Ich habe dafür 5 Wochen gebraucht bis es 100%ig funktionierte und du kannst es einfach kopieren :−). Ich hoffe du erkennst den Wert dieses Programms, wenn du es benutzt.
Als Anwendung für unser neues I2C Protokoll werden wir ein Thermometer bauen. Du kannst natürlich auch irgend etwas anderes messen oder einfach Lichter ein/aus−schalten. Es ist deine Entscheidung.

In einem zweiten Artikel werden wir ein LCD−Display hinzufügen. Mit anderen Worten es wird ein Thermometer bei dem man die Temperatur über den Linux PC oder direkt über eine Anzeige ablesen kann. Das Display kommt in einem zweiten Artikel um diesen hier nicht zu überladen.

NTCs sind klein, billig und ausreichend genau

Der Temperatursensor

Es gibt schon fertig kalibrierte Temperatursensoren (einige von ihnen reden sogar I2C ;−) aber diese sind recht teuer. NTCs sind viel billiger und fast genauso gut selbst ohne Kalibrierung. Wenn man sie kalibriert kann man Genauigkeiten bis hinter dem Dezimalpunkt erreichen.

Ein Problem von NTCs ist ihre Nichtlinearität. Es ist aber eigentlich nur eine Frage der Halbleiterphysik hier die richtige Formel zu finden und diese nicht lineare Kurve zu korrigieren. Der Microcontroller ist ein kleiner Computer und mathematische Operationen sind überhaupt kein Problem. NTCs sind temperaturabhängige Widerstände. Der Widerstandswert R eines NTCs bei einer gegebenen Temperatur ist:

$$R = R_N \cdot e^{B(\frac{1}{T} - \frac{1}{T_N})}$$

where

R_N = Value of NTC at T_N

$T_N = 25\ ^{o}/_{K} + 273K$

B = see datasheet of NTC

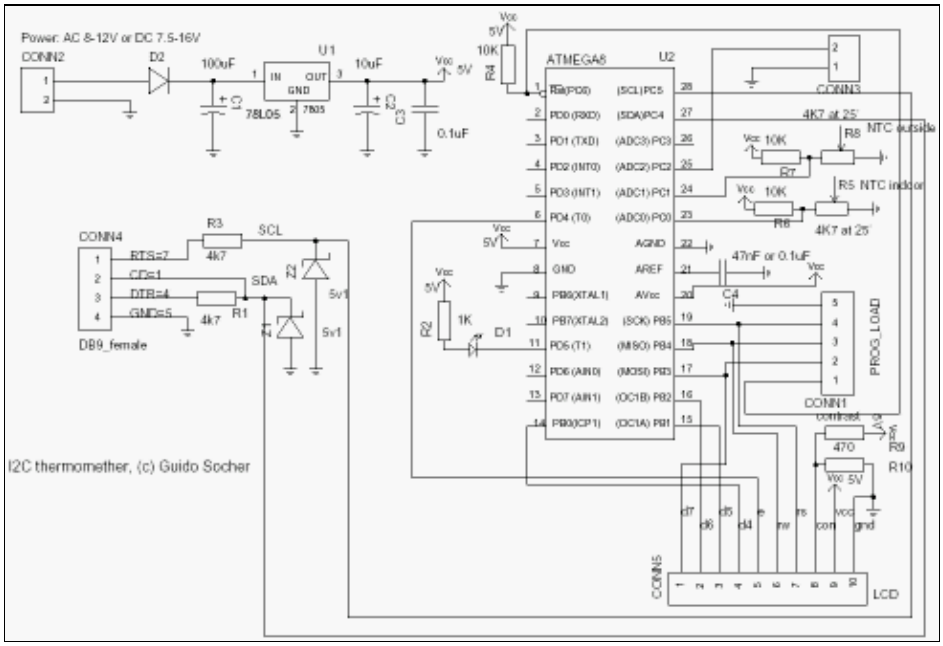
$T = T_C + 273K$

thus T_C *can be written as*

$$T_C = \frac{1}{\frac{1}{B}\ln(\frac{R}{R_N}) + \frac{1}{T_N + 273}} - 273$$

T oder Tc ist die Temperatur, die wir suchen. Rn ist der Widerstandswert bei 25'C. Man kann NTCs mit 4.7K, 10K, ... kaufen. Rn ist dieser Wert.

Die Schaltung



Schaltplan. Klick auf den Plan um eine genauere Ansicht zu erhalten.

Nun haben wir alles um das digitale Thermometer zu bauen. Wir nehmen zwei NTCs. Einen für innen und einen für die Außentemperatur. Du kannst noch mehr Sensoren anschließen. (conn3, Pin PC2 ist z.B frei). In dem Schaltbild sieht man schon das LCD−Display. Ich wollte vermeiden, daß für den nächsten Artikel wieder eine neue Schaltung gebaut werden muß.

In der Schaltung findet sich eine ein LED. Das kostet nicht viel und ist sehr nützlich zum debuggen. Ich habe es oft benutzt als ich die I2C State−machine für die Kommunikation zwischen PC und Microcontroller entwickelt habe. Im normalen Betrieb lassen wir die LED einfach blinken um zu zeigen, daß Messwerte genommen werden.

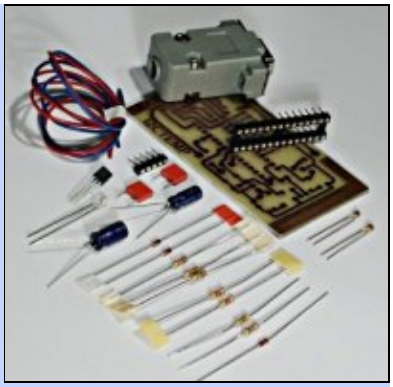
Die Schaltung ist ansonsten ganz einfach. Der Analogdigitalwandler in dem Microcontroller wird benutzt um den Spannungsabfall an einem NTC zu messen. Dieser wird dann in einen Temperaturwert übersetzt.

Der Atmega8 hat verschiedene Möglichkeiten die Referenzspannung des Analogdigitalwandlers zu wählen. Man kann entweder 5V (AVcc) oder einen interen 2.56V Referenz benutzen. Für die Innentemperatur brauchen wir einen viel kleineren Bereich. +10'C bis +40'C sollten ausreichen. Wir nehmen daher hier die 2.56V Referenz und erhalten die doppelte Genauigkeit, da jetzt 1024 mögliche digitale Werte auf 0−2.56V verteilt sind. Das ergibt eine Genauigkeit von 2.5mV (genauer als die meisten digitalen Voltmeter!).

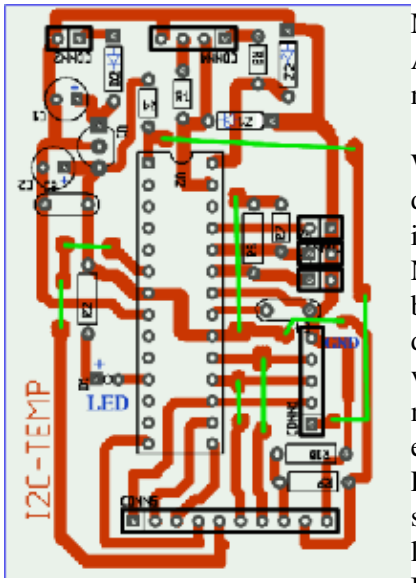
Der CD Anschluß an der RS232 Schnittstelle ist ein Eingang und wir schließen ihn daher an die SDA Leitung des I2C Bus an. DTR und RTS sind Ausgänge. Wenn der PC Daten−bits auf den Bus gibt, dann ändert er einfach DTR. Der I2C Master, hier der Linux PC, kontrolliert die SCL (Takt) Leitung. Mit anderen Worten SCL ist ein Ausgang an dem RS232 Anschluß.

Der 78L05 wird benutzt um eine stabile Stromversorgung und eine Referenzspannung zu erhalten. Du kannst daher fast jede Art von Stromversorgung benutzen. DC oder AC zwischen 7.5V ond 12V. 9V ist eine gute Wahl.

Die Platine erstellen



tuxgraphics.org verkauft alle nötigen Bauteile zusammen mir einer geätzten Platine.



Man kann natürlich auch die Lochrasterplatine aus dem vorangegangenen Artikel benutzen. Dazu muß man die LED auf Pin 11 umlöten und die neuen Bauteile hinzufügen.

Wenn du jedoch eine Schaltung bauen möchtest, die auch gut aussieht, dann macht es mehr Sinn eine neue Platine zu ätzen, denn diese Schaltung ist einfach viel komplexer als die Testschaltung aus dem ersten Artikel. Nachdem ich Iznogood's Linuxfocus Artikel über gEDA gelesen hatte, beschloss ich dieses Mal gEDA anstelle von Eagel zu benutzen. gschem, das Schaltplanzeichenprogramm von gEDA ist sehr gut. Die Bibliothek der vorhandenen Symbole ist wesentlich kleiner als bei Eagle und so mußte ich mein eigenes Symbol für den Atmega8 entwerfen, aber ging erstaunlich einfach. Viel problematischer ist pcb, das Programm zum Zeichnen der Platine. Wenn man vorher Eagle benutzt hat wird man zunächst irritiert sein, daß sich die Gummibandverbindungen von den Bauteilen entfernen lassen. Um sicher zu sein, daß das richtige Gummiband an dem richtigen Bauteil hängt muß man sie von Zeit zu Zeit neu erzeugen (Connects−>Optimize rats−nest). Man sollte sich auch sehr sicher sein, daß der Schaltplan komplett ist, denn nachträglich Änderungen werden nur halb manuell übertragen und es einfach dabei Fehler zu machen.

Ich benutzte den orangen Layer zum zeichnen. Irgendwie erzeugten die anderen verfügbaren Farben einfach keine Ausgabe beim Drucken. Das Problem ist jedoch, das dieser orange Layer offenbar auf der Seite liegt, wo die Bauteile sind. Man muß das Ergebnis also spiegeln. Letztendlich entschloss ich mich, die wesentlichen Teile des Layout mit pcb und den Rest mit gimp zu zeichnen.

Dank shop.tuxgraphics.org brauchst du nicht mit gefährlichen Chemikalien zu hantieren oder in verschiedenen Geschäften nach den richtigen Komponenten suchen. tuxgraphics verkauft alle Teile, die man für diesen Artikel braucht.

Der Aufbau

Beim Zusammenbau sollte man auf die Teile achten, bei denen die Polarität wichtig ist: Elektrolytkoindenstoren, Dioden, Z−Dioden, 78L05, LED und Microcontroller.

Bevor man den Microcontroller in den Sockel steckt sollte man die Stromversorgung mit einem Meßgerät testen. Falls diese nicht funktioniert bekommt man nicht nur falsche Werte, sondern zerstört möglicherweise auch den Microcontroller. Als nächstes kann man die Schaltung an den RS232 Anschluß des Linux PCs anschließen un mit dem i2c_rs232_pintest Programm die verschiedenen Signalkombinationen auf den SDA und SCL Anschlüssen zu testen.

```
i2c_rs232_pintest -d 1 -c 1
i2c_rs232_pintest -d 0 -c 1
i2c_rs232_pintest -d 1 -c 0
```

Dieses Programm setzt die Spannungspegel auf den RTS (als SCL genutzt, Option −c) und DTR (als SDA genutzt, Optio −d) an der RS232 Schnittstelle. RS232 hat Spannungen von +/− 10V. Hinter den Z−Dioden sollte man jedoch nur −0.7 für eine logische Null und +4−5V für eine logische Eins messen.

Erst wenn diese Tests bestanden sind, sollte man den Microcontroller in den Sockel setzen.

I2C Kommunikation nutzen.

Lade das linuxI2Ctemp tar.gz File herunter (siehe Referenzen) und entpacke es. Die I2C Kommunikation ist in 2 Dateien implementiert:

```
i2c_avr.c -- die i2c Zustandsmaschine für den atmega8
i2c_m.c   -- das komplette i2c Protokoll für die Linuxseite.
```

Ich habe dem atmega8 die Slave Adresse "3" gegeben. Um den String "hello" and den atmega8 zu senden würde man folgende C Funktionen ausführen:

```
address_slave(3,0); // tell the slave that we will send something
i2c_tx_string("hello");
i2cstop(); // release the i2c bus

Auf der Microcontrollerseite würde man den String mit
der folgenden Funktion empfangen:
i2c_get_received_data(rec_buf);
```

Ganz einfach. Das lesen von Daten aus dem Microcontroller geht ähnlich.

Wie warm ist es?

Um den Code für den Microcontroller zu kompilieren und zu laden, sind die folgenden Befehle nötig:

```
make
make load
```

Um die Programme i2c_rs232_pintest und i2ctemp_linux zu compilieren tippt man:

```
make i2c_rs232_pintest
make i2ctemp_linux
```

... oder man kann die vorkompilierten Versionen aus dem "bin" Verzeichnis benutzen.

Um Temperaturwerte zu lesen nimmt man einfach:

```
i2ctemp_linux
```

... und die Werte für Innen– und Außentemperatur werden auf den Bildschirm gedruckt. Um diese Daten über einen Webserver zur Verfügung zu stellen empfehle ich nicht das i2ctemp_linux Programm direkt vom Webserver aus laufen zu lassen, da es recht langsam ist. Besser ist es einen cron–Job zu schreiben, der regelmäßig eine html–Datei erzeugt. Ein Beispiel dafür findet sich in der README Datei des linuxI2Ctemp Paketes.

Zusammenfassung

Das I2C Protokoll braucht sehr wenig extra Hardware und ist optimiert für kleine Datenmengen. Das ist genau das, was wir brauchen, wenn wir Meßdaten von dem Microcontroller lesen. Eine elegante Lösung.

In diesem Artikel lag der Schwerpunkt auf der Hardware. Wenn dir dieser Artikel gefallen hat, dann schreibe ich noch einen zweiten Artikel, in dem die Software beschrieben wird. Speziell wie die Analogdigitalwandlung und die Implementation des I2C Protokolls funktionieren. In diesem zweiten Artikel

kommen auch das LCD Display und Celsius nach Fahrenheit Konvertierung hinzu.

Referenzen

- **Download**−Seite für diesen Artikel: die linuxI2Ctemp Software, Zeichnungen, Software−updates
- Wie man den Atmega8 mit gcc programmiert: November2004 Artikel 352
- Datenblatt für den Atmega8: gehe nach http://www.atmel.com/ und selektiere products−>Microcontrollers −>AVR−8 bit RISC−>Documentation−>datasheets (lokale Kopie, pdf, 2479982 Bytes)
- Der tuxgraphics shop. Ein wirklich toller Onlineshop :−): shop.tuxgraphics.org
  Hier bekommt man die Linux AVR Programmier−CD, alle Teile für diesen Artikel, LCD Displays und Microcontroller

---

2005−03−26, generated by lfparser_pdf version 2.51