Morse-Terminal

Ralf Beesner

11.12.2010

1 Einleitung

Mein vorheriges Projekt „Seriell-Morse-Wandler“ deckt nur eine Richtung ab - die Ausgabe serieller Signale als Morsecode. In diesem Projekt wird nun auch die Gegenrichtung realisiert - es entsteht ein Halbduplex- Morseterminal, das sowohl serielle Signale in Morse ausgeben als auch Morseeingaben in serielle Signalen wandeln kann. Die Morseeingabe erfolgt (wie bei einem Squeeze- Elbug) mit zwei Paddles.

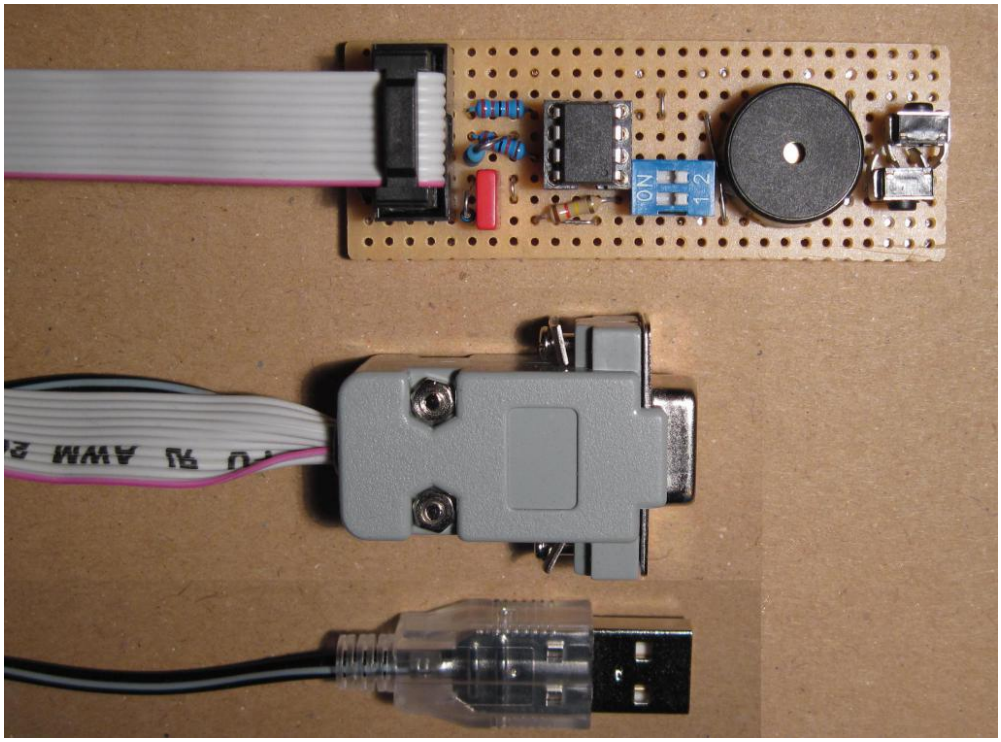
Damit wird es möglich, sich in Morse auf einem Rechner einzuloggen, der den Login über eine serielle Schnittstelle unterstützt. Auf einem Linux-Rechner läßt sich das mit Bordmitteln einrichten. Allerdings zeigt sich, daß die Kommunikation doch recht mühsam ist, weil der Login- Vorgang einerseits recht geschwätzig ist (er ist halt auf einen vergleichsweise schnellen Modemzugang und nicht auf langsames Morse ausgerichtet) und andererseits keine kleinen Fehler verzeiht, so daß der Login leicht in die „Grütze“ geht.

Da nur wenige I/Os benötigt werden und um die Sache spannender zu machen, basiert das Projekt auf einem ATtiny 45. Der „Seriell-Morse-Wandler“ mit einem ATtiny 13 hatte gezeigt, daß ein Software-UART mit selbstgestricktem Zeichenpuffer schnell genug ist, und der ATtiny 45 bietet mit seinen 256 Byte SRAM Platz für einen Zeichenpuffer von etwa 180 Bytes.

Da auch der Flashspeicher grosszügig bemessen ist, war Platz für eine interaktive Geschwindigkeitseinstellung.

2 Prinzip der Codierung:

Die Codierung erfolgt wie in meinen älteren Projekten:



- 1 Byte pro Zeichen, niederwertiges Bit zuerst
- hat das Bit den Wert 0: Morsepunkt (dit)
- hat das Bit den Wert 1: Morsestrich (dah)
- da Morsezeichen unterschiedlich lang sind, ist ein Ende- Zeichen erforderlich; es ist ebenfalls ein 1 - Bit
- Beispiel Morsezeichen „a“ (dit dah): `&B00000110`
- Beispiel Morsezeichen „9“ (dah dah dah dah dit): `&B00101111`

Das Umsetzen empfangener serieller Signale in Morsezeichen erfolgt nach dem bekannten Muster:

In der Tabelle wird das zu dem empfangenen ASCII- Zeichen gehörende Morsebyte nachgeschlagen. Es wird das niedrigstwertige Bit ausgelesen; ist es eine „0“, wird ein Morsepunkt erzeugt, ist es eine „1“, wird ein Morsestrich gebildet. Danach wird der Byteinhalt nach rechts geschoben und wieder das niederwertigste Bit ausgewertet. Hat

das Byte nur noch den Gesamtwert „1“, ist das Ende- Zeichen rechts angekommen; es wird eine Zeichenpause eingefügt und das nächste ASCII- Zeichen ausgewertet.

Für die Senderichtung ist eine Hilfsvariable erforderlich:

Die Byte- Hilfsvariable „K“ wird zunächst auf „1“ gesetzt. Mit jedem Morsetakt wird das „1“ - Bit um eine Stelle nach links geschoben. Wurde der Morsetakt durch den Strich- Hebel ausgelöst, wird der Wert von „K“ in einer Variablen „D“ aufaddiert. In „D“ befindet sich also - von rechts nach links gelesen - pro Punkt eine „0“ und pro Strich eine „1“.

Wird kein Hebel mehr gedrückt, weil das Zeichen komplett ist, wird die Trennzeit zwischen zwei Morsezeichen abgewartet und dann das Ende- Zeichen durch letztmaliges Addieren von „K“ zu „D“ angefügt. Anschließend wird per Lookdown in der Zeichentabelle das zu dem Morsebyte passende ASCII- Zeichen nachgeschlagen und über die serielle Schnittstelle ausgesandt.

3 Hardware

Die Hardware ist an das LP Microcontroller angelehnt und auf einer Lochrasterplatine aufgebaut. Die serielle Schnittstelle „hängt“ an den Ports PB0, PB1 und PB2; allerdings habe ich, weil die Pins der D-Sub- Leiterplattensteck- verbinder nicht im 2,54mm- Raster liegen, eine 10- poligen Wannensteckverbinder verwendet. Früher hatte ich Wannenstecker gemieden, weil die beiden Stiftreihen nur 2,54 mm auseinander liegen und das Auftrennen der Leiterbahnen zwischen den Reihen daher schwierig ist, aber wenn man die Leiterbahnen bereits vor dem Verlöten des Wannensteckers mit einem Messer aufritzt und dann mit einer Schraubendreherklinge den Spalt etwas verbreitert, lassen sich die Wannenstecker auch auf Lochrasterplatinen recht gut verarbeiten.

Die Verbindung zur PC- seitigen D-Sub- Steckbuchse erfolgt mit 10- poligem Flachbandkabel; die zehnte Ader an Pin 10 des Wannensteckers dient der Zuführung der Betriebsspannung, die an einer USB- Buchse des PC abgegriffen wird.

Zu beachten ist die unterschiedliche Zählweise der Wannensteckerpins und der D-Sub-Pins.

Die Betriebsspannung kann wahlweise auch über die RTS- Leitung und eine Diode 1N4148 bereitgestellt werden, um die Kompatibilität zum LP Microcontroller zu wahren.

An PB4 und PB3 liegen zwei Mikrotaster, mit denen die Striche und Punkte eingegeben werden; an PB0 der Buzzer. Ein „Mäuseklavier“ mit zwei Schaltern legt den Reset auf Masse bzw. unterbricht die Verbindung PB0 - Buzzer. Schließt man den Reset- Schalter und öffnet man den Buzzer- Schalter, läßt sich der ATtiny über die serielle Schnittstelle programmieren.

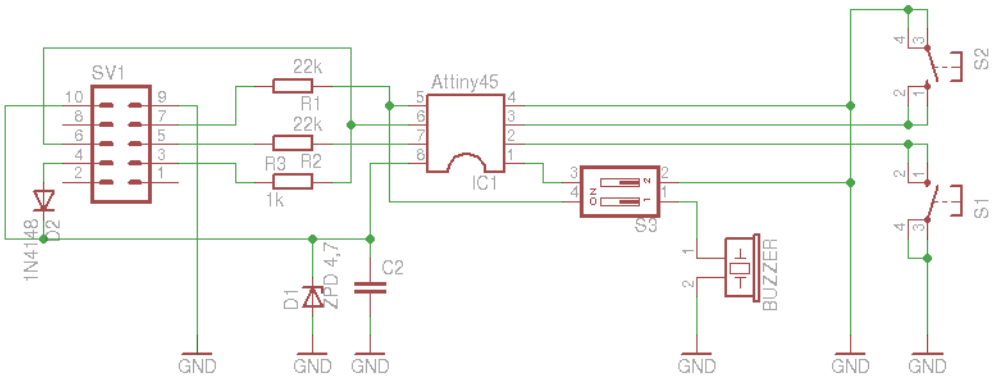


Abbildung 1: Schaltplan

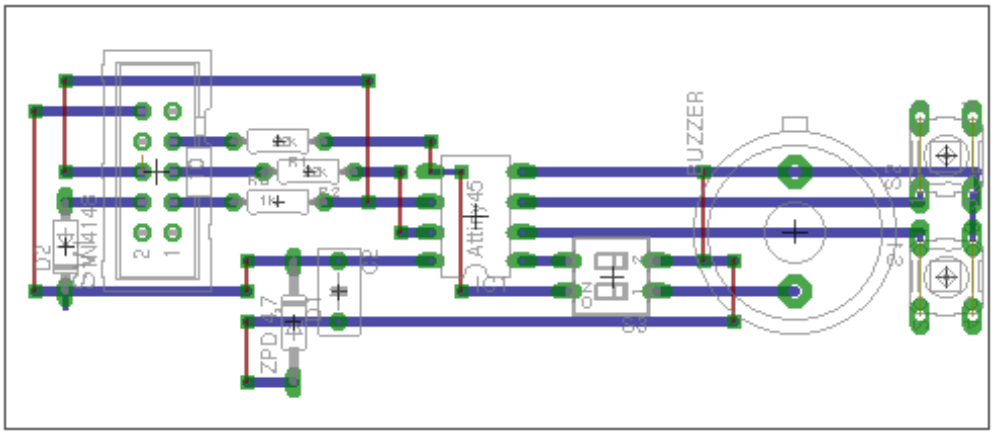


Abbildung 2: Layout

4 Flashen

ATtiny 13 lassen sich in dieser Schaltung mit Burkhard Kainkas LPMicroISP.exe programmieren. Für ATtiny 45 eignet sich das Kommandozeilenprogramm avrdude.exe; ich beschreibe in einem separaten Beitrag, wie man ihm - auch wenn man mit der Kommandozeile auf Kriegsfuß steht - mit passenden Batchdateien seinen Schrecken nehmen kann.

5 Software

Die Software gliedert sich in folgende Blöcke:

- Interrupt- Routine „seriellelesen“, sie buffert die Zeichen, die über die serielle Schnittstelle eingehen
- „Speed“ fragt nach einem Reset die gewünschte Morsegeschwindigkeit ab (Frage und Antwort in Morse)
- „Zeichenlesen“ und „Morse“ geben die Zeichen aus dem seriellen Buffer akustisch in Morse aus
- „Decodekeyer“ wandelt die mit den Paddles eingegebenen Morsezeichen in Ascii und sendet sie auf die serielle Schnittstelle
- „Main“ ruft „Zeichenlesen“ und „Decodekeyer“ abwechselnd auf

Für die Eingabe der Morsezeichen wird die Geschwindigkeit auf 2/3 herabgesetzt, um Gebefehler zu minimieren, außerdem wird die Tonhöhe etwas abgesenkt, damit man eigene Eingaben besser von Ausgaben unterscheiden kann.

Da das Morsealphabet keinen Zeilenumbruch kennt, dieser aber für die Kommunikation mit einer Maschine wichtig ist, habe ich das Zeichen „KN“ (ohne Zwischenraum gegeben) als Zeilenumbruch definiert.

Ich habe den Code auch auf den ATmega8 und den ATmega88 portiert; die ATmega88-Version orientiert sich am Franzis- LP Bascom (die beiden vorhandenen Taster werden als Punkt- und Strichkontakt genutzt; der Trimmwiderstand dient zur Geschwindigkeitseinstellung).

Einrichten des Linuxrechners

Da Linux einige Sonderzeichen verwendet, die das Morsealphabet nicht kennt und da es auch „zu geschwätzig“ ist, habe ich einen neuen User „m“ eingerichtet und versucht, einige Probleme zu umschiffen.

Der User „m“ hat ein leeres Passwort. Im Homeordner liegen ein paar Dateien, die den User „morsekompatibler“ machen sollen. Eine 0-Byte Datei namens `.hushlogin` verkürzt den Login- String auf das nötigste, und in der Datei `.bashrc` wird der Prompt vom schwer zu morsenden

```
m@robbi:/home/m>
```

auf

```
++ <KN>
```

abgeändert.

Außerdem muß man den login über eine serielle Schnittstelle freischalten. Linux stellt auf dem Rechner (neben dem Grafikmodus) mehrere Textkonsolen bereit, zwischen denen man mit F1 ... F6 hin- und herschalten kann. Sie sind in der Datei `/etc/inittab` konfiguriert. Analog zu den virtuellen Textkonsolen kann man über einen inittab-Eintrag ein echtes serielles Terminal einrichten (Neustart des Rechners erforderlich, damit der Eintrag übernommen wird):

```
# Local serial lines:
s1:12345:respawn:/sbin/agetty -L -w -i -t 300 ttyS0 9600 dumb
```

`agetty` ist das Programm, das den login entgegennimmt. Die Parameter für `agetty` unterdrücken den Carrier Detect, den ein Modem ausgeben würde (`agetty` wartet stattdessen auf einen Carriage return, der den login startet) verlängern die Anmeldezeit auf 300 Sekunden und lassen `agetty` auf der ersten seriellen Schnittstelle horchen; die Datenrate ist 9600 bit/s und es werden keine speziellen Terminal- Steuerzeichen ausgewertet.

Unter Ubuntu funktioniert das leider so nicht, weil Ubuntu nicht die klassische Unix-Start- Maschinerie „Init“, sondern einen neueren Ansatz namens „Upstart“ verwendet.

Man kann `/sbin/agetty` testweise auch als root direkt in einem Terminal starten, aber es verhält sich dann etwas sonderbar.

praktische Demonstration

Das ganze ist ziemlich umständlich und nicht wirklich praktikabel, aber funktioniert grundsätzlich, wie die beigefügte MP3- Datei zeigt. Hier ist der Dialog in plain text; die Morse- Ausgaben des Systems sind in Kleinschrift, meine Morse- Eingaben in Großschrift:

```
robbi login:
M
password:
login incorrect

robbi login:
```

```
M


++
LS
hello.txt  profile  vertrag.txt
++
CAT HELLO.TXT

this is slackware 13.1 morse shell
how are you ?
have fun.
++
EXIT
abgemeldet
```

Erläuterung: Der erste login ist fehlerhaft, der erneute klappt dann; mit „ls“ werden anschließend die Dateien im Heimatordner aufgelistet. Anschließend wird mit „cat hello.txt“ der Inhalt der Datei „hello.txt“ ausgegeben. Danach erfolgt mit dem Befehl „exit“ die Abmeldung

Fazit

Zur Kommunikation mit einem PC ist das Morse-Terminal nicht wirklich brauchbar; es ist einfach zu langsam und hinsichtlich des Zeichensatzes zu beschränkt .

Es eignet sich aber durchaus zur Kommunikation mit einem anderen Microcontroller-system, das nur wenige Ein- und Ausgaben erfordert (z.B. einen Konfigurations- Dialog)

Die Subroutinen des Programms lassen sich in anderen Projekten (z.B. Speicher- morsetaste, Codeschloss oder ATtiny-45-Morseuhr) zur direkten Steuerung des Mikrocontrollers verwenden, da man über nur zwei IO-Pins Befehle an den Mikrocontroller senden kann und nur einen Pin für die „Gegenrichtung“ benötigt.