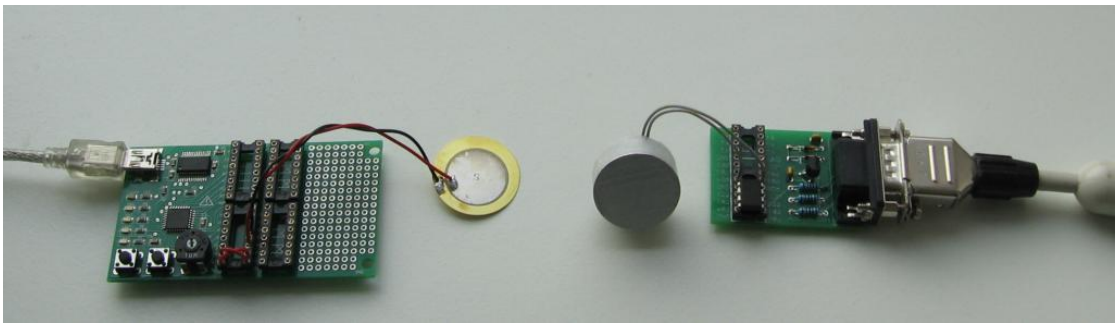
Seriell- Morse- Wandler

Ralf Beesner, DK5BU

26.11.2010



1 Einleitung

Serielle Schnittstellen gelten zwar als veraltet, sind aber trotzdem nicht „totzukriegen“. Zahlreiche Geräte geben ihre Daten immer noch über serielle Schnittstellen aus.

Geht es nur um sehr geringe Datenmengen, kann man auf einen Terminal- PC als Ausgabegerät verzichten und stattdessen die Informationen mit einem Microcontroller aufbereiten und auf einem LCD- Display anzeigen - im Web findet man einige Projekte, die sich damit beschäftigen. Viel „cooler“ finde ich jedoch, die seriellen Informationen in Morsezeichen umzuwandeln und akustisch auszugeben.

Ich habe einige Programme beigefügt; das erste ist für den ATmega8. An Hardware werden neben dem obligatorischen Pegelwandler für die serielle Schnittstelle (z.B. MAX 232) nur ein Piezoschwinger an PB3 und zwei Potentiometer an PC0/ADC0 und PC1/ADC1 (Tonhöhe und Morsegeschwindigkeit) benötigt.

Mit minimalen Änderungen läuft es auch auf dem ATmega88. Das zweite Programm ist auf die Atmega88- Hardware des Franzis- Lernpakets „Mikrocontroller-Technik mit Bascom“ zugeschnitten. Auch hier ist der Piezoschwinger an PB3 anzuschließen; das

bereits auf der Platine vorhandene Poti (an ADC7) dient zur Geschwindigkeitseinstellung. Die Tonhöhe ist über eine Konstante fest eingestellt. Der USB-Anschluß bildet die serielle Schnittstelle. Allerdings ist Hyperterm als Terminalprogramm ungeeignet, weil es die DTR- Leitung aktiviert und damit den ATmega88 auf Reset schaltet. Das interne Bascom- Terminalprogramm funktioniert jedoch.

Die restlichen 3 Programme sind für den ATtiny13 und die Hardware des Franzis-Lernpakets „Mikrocontroller“ gedacht. Es muß lediglich ein Piezoschwinger an PB0 angeschlossen werden; Tonhöhe und Geschwindigkeit sind konstant.

2 Gemeinsamkeiten und Unterschiede der Programme

Der zentrale Teil - Umwandeln von ASCII- Zeichen in Morsesignale - erfolgt ähnlich wie bei meinen Projekten „Morsebake“, „Morsethermometer“ und „Morseuhr“. Da hier nicht nur eine Teilmenge des Morsealphabets benutzt wird, sondern alle Zeichen, ist die Umcodierung abweichend gelöst. Die Umwandlungstabelle steckt in mehreren DATA-Zeilen, in denen mit dem Befehl „Lookup“ nachgeschlagen wird.

Es wäre auch möglich, diese Daten bei Programmstart in eine Array-Variable von 91 bzw. 122 Bytes einzulesen und dort nachzuschlagen, aber RAM ist ja besonders bei den kleinen Microcontrollern knapp und kostbar.

Da die Morseausgabe sehr langsam erfolgt und die Bytes sehr viel schneller über die serielle Schnittstelle „eintrudeln“, müssen die Daten zwischengespeichert werden. Ist ein Hardware-UART vorhanden, nimmt einem BASCOM die Arbeit ab: mit dem Befehl „Config Serial“ kann man einen Buffer von max. 254 Zeichen konfigurieren und muß nur noch Interrupts allgemein freigeben, da BASCOM für das Puffern im Hintergrund eine Interrupt- Routine einrichtet.

Beim ATmega8 und ATmega88 ist man also „fein raus“; beim ATtiny13 muß man jedoch den Software- UART benutzen, und der stellt keine Buffer zur Verfügung.

Das Programm „T13-Seriell-Morse-Wandler-1.bas“ demonstriert das Problem: mit „GET“ und „PUT“ kann man sehr einfach Bytes von der Software- UART abholen und dort wieder abliefern, aber während die Morseausgabe läuft, wird der Software-UART nicht abgefragt - die zwischenzeitlich eintrudelnden Bytes gehen ins Leere.

3 Buffern mit BASCOM- Bordmitteln

Im Programm „T13-Seriell-Morse-Wandler-2.bas“ habe ich versucht, den Buffer „zu Fuß“ zu bauen.

Der RX- Pin des Software-UART (PB2) wird mit einem Pin Change Interrupt überwacht. Ändert sich das Potential, weil gerade ein neues Startbit einläuft, verzweigt

das Programm kurz in eine Interruptroutine. Sie nimmt das Zeichen entgegen, gibt es als Echo auf dem TX- Pin des Terminals gleich wieder aus und hängt es an einen String namens „Pufferstring“ an. Das geht so schnell, daß die parallel laufende Morseausgabe nicht merklich verzögert wird.

Im Hauptprogramm, der Routine „Zeichenlesen“, wird das älteste „linke“ Zeichen des Pufferstrings ausgelesen und dann der Pufferinhalt um eine Stelle nach „links“ geschoben. Zuvor wird jedoch die Länge des Pufferstrings geprüft, ist er leer, muß das Programm „auf die Bremse treten“.

Leider benötigt die Interrupt- Routine 32 Bytes des knappen SRAM, da sie alle 32 Register auf den Stack retten muß. Daher ist der Puffer sehr klein (nur 15 Bytes).

Eigentlich ist die Verwendung eines Strings ein Umweg, da es ja Bytes sind, die gepuffert werden sollen. Aber Strings lassen sich mit BASCOM leichter bearbeiten, da ist die Hin- und Rückwandlung in Kauf zu nehmen.

In Roland Walters „AVR Mikrocontroller Lehrbuch“ habe ich einen Weg gefunden, zumindest die Rückwandlung mit Hilfe einer Overlay- Variable bytesparend zu umgehen (sein Beispiel beschreibt einen Buffer für die Senderichtung eines Hardware-UART). Im Programm „T13-Seriell-Morse-Wandler-2-overlay.bas“ ist das sinngemäß übernommen und spart ein Byte SRAM und ein paar Prozent Flash.

Das Buffern mit BASCOM- Befehlen ist allerdings recht langsam; bei den weiter unten beschriebenen Experimenten mit der Kommandozeile „verschluckte“ sich der ATtiny13 an den zu rasch eintreffenden Zeichen (nach Umstellung der Taktrate von 1,2 MHz auf 4,8 MHz funktionierte es jedoch). Die Windows- Eingabeumleitung legt auch nicht die RTS- Leitung und die DTR- Leitung auf High; man muß den ATtiny13 (bzw. den Eingang des Spannungsreglers) extern mit 5 V speisen.

4 Weitere Hinweise:

Die Programme wurden unter Windows XP im Zusammenspiel mit dem Windows-Standard- Programm „Hyperterminal“ getestet. 9600 bit/s 8N1, „doofes“ TTY, nur CR, kein LF, kein lokales Echo (der Mikrocontroller sendet deshalb die Zeichen zurück, damit sie auf dem Terminal erscheinen).

Je nach Terminalprogramm bzw. serieller Hardware wird es erforderlich sein, im Seriell-Morse-Wandler das Echo abzuschalten (auszukommentieren) und/oder mit dem Zeilenendezeichen zu experimentieren.

Windows schleppt ein paar nette „DOS- Altlasten“ mit sich herum (die letztendlich aus der UNIX- Welt stammen). Auf der Kommandozeile kann man per Ausgabeumleitung Texte oder die Inhalte von Dateien auf die serielle Schnittstelle umleiten. So kann man mit dem Befehl

```
echo test 12345 > com1
```

den String „test 12345“ auf den Morsewandler senden. Mit

```
cat 'Dateiname' > com1
```

kann man sich eine Text- Datei in Morse vorlesen lassen, und in Batchdateien lassen sich kleine Programme zusammenfügen, die ihre Ausgaben jeweils auf den Morsewandler leiten.

Solche Spielchen ermöglicht Windows anscheinend nur mit echten seriellen Schnittstellen, nicht mit virtuellen USB- Com- Ports.

Falls es mit einem echten Com- Port Probleme gibt, sollte man unter „Systemsteuerung“ - „System“ - „Gerätemanager“ - „Hardware“ - „Anschlüsse ...“ - „Kommunikationsanschluss (COM1) - Eigenschaften“ nachschauen, ob dort „9600 bit/s 8N1, Flussteuerung:keine“ eingestellt ist.

Unter Linux ist es recht einfach, Hardware- Meldungen auf den Morsewandler zu geben. Zum Beispiel kann man sich den Ladezustand der Notebook- Batterie mit

```
cat /proc/acpi/battery/CMB1/state | grep remaining > /dev/ttyS0
```

in Morse ausgeben lassen; der Vorgang läßt sich auch leicht automatisieren und zu festen Zeiten wiederholen.

Unter Linux funktioniert die Ausgabeumleitung eigentlich auch mit seriellen USB-Schnitt- stellen, allerdings nur, wenn lediglich RXD, TXD und Masse angeschlossen werden. Bei der Hardware des „Franzis- Lernpakets BASCOM“ tritt auch unter Linux das Problem auf, daß die DTR- Leitung Resets auslöst.