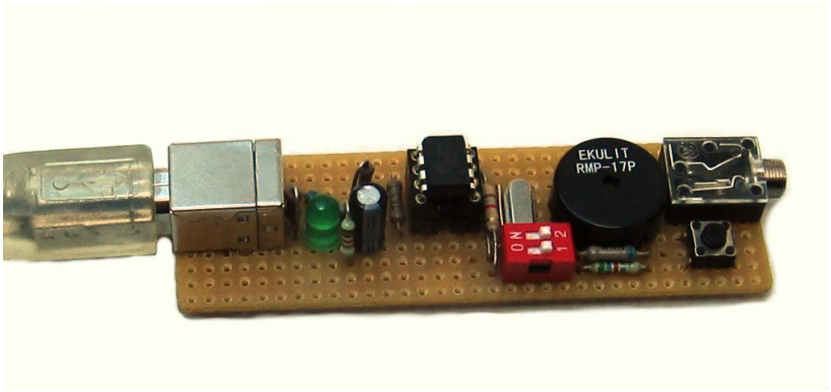
inverses Morsekeyboard mit Bascom-SWUSB

Ralf Beesner

3. September 2012



1 Überblick

In einem früheren Beitrag (siehe:
`http://www.elektronik-labor.de/AVR/Morseterminal2.html`)
hatte ich einen BASCOM- Morsedecoder mit einem Serial Keyboard auf Basis der VUSB- Software `www.obdev.at/products/vusb/index-de.html` kombiniert. Der AtTiny 13- Morsedecoder wandelt eingegebene Morsezeichen in ein serielles Signal, dieses wird an das Serial Keyboard (einen AtTiny 2313) weitergeleitet, der als virtuelle USB- Tastatur fungiert und die Zeichen in Tastatureingaben umsetzt.

Allerdings ärgerte mich doch ein wenig, dass die vielen IOs des 20-poligen Attiny2313 brachlagen (der AtTiny 2313 kam einzig deshalb ins Spiel, weil er der kleinste AVR mit einem Hardware- UART ist).

Eigentlich wollte ich immer mal versuchen, den Morsedecoder von BASCOM nach C zu portieren und in die VUSB- Software einzubauen, damit das ganze in einen einzigen AtTiny 45 passt.

Mit der BASCOM- SWUSB- Library von Rick Richards (ollopa) `www.sloservers.com/swusb` wurde nun eine viel einfachere Lösung möglich: ich habe in seine Keyboard- Beispielimplementation mit recht wenig Aufwand den Morsedecoder "hineingefummelt".

2 Hardware

Die USB- Anschaltung an den Mikrocontroller ist die gleiche wie bei den VUSB-Projekten. Da die SWUSB-Implementation stets einen Quarz benötigt, hat man bei Verwendung eines AtTiny 45 nur einen Pin frei.

Benötigt werden eigentlich zwei, besser sogar vier: Lautsprecher (Mithörton), Morsetaste und zwei Dipschalter für eine grobe Geschwindigkeitsvorwahl und die Umschaltung zwischen manuellem und automatischem Einfügen der Leerzeichen.

Man könnte statt eines AtTiny 45 einen AtTiny44 nehmen, der 6 Pins mehr als der AtTiny 45 hat. Da ich keinen zur Hand hatte, wird (nach einem Vorschlag von Hermann Nieder) der Reset- Pin über einen Trick nutzbar gemacht: wenn der Reset- Pin aktiv ist, kann man ihn zwar nicht als digitalen Eingang verwenden, aber als ADC- Eingang (ADC0). Man muss nur Sorge tragen, dass das angelegte Signal stets größer ist als 1/2 VCC, damit kein Reset ausgelöst wird.

Die beiden DIP- Schalter und die Morsetaste werden daher über einen Spannungsteiler an PB5/ADC0 angeschlossen. Die Widerstände sind so bemessen, dass die Spannung stets höher als 1/2 VCC ist. ADC0 gibt bei offenen Tastern 1023 aus; wird der erste DIP-Schalter betätigt, sinkt der Wert knapp unter 950, wird der zweite DIP-Schalter betätigt, sinkt er knapp unter 900. Bei Drücken der Morsetatste sinkt er unter 750.

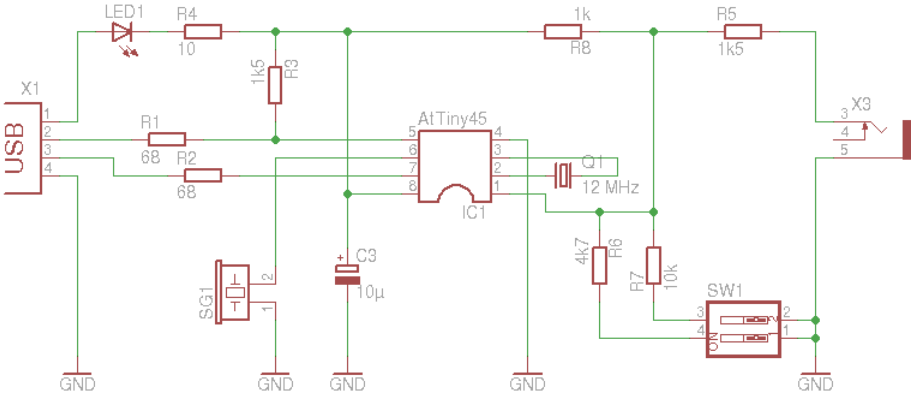


Abbildung 1: Schaltbild Morsekeyboard

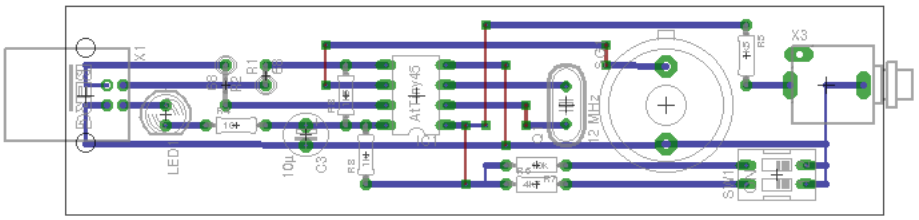


Abbildung 2: Streifenraster- Layout

3 Software

Ollopas Keyboard- Beispiel- Implementation habe ich nur ansatzweise verstanden, aber glücklicherweise erkennt man leicht die Stellen, an denen man sich in den Code mit seiner eigenen Task einhängen kann.

Der USB ist mit seinem relativ hohen Takt (1,5 MHz) sehr zeitkritisch, so dass die Anwendung eigener Interrupts nicht möglich ist. Man kann also in der eigenen Task Eingangspins, ADC- oder Timer- Register und dergleichen nur zyklisch abfragen (pollen).

In Ollopas Beispielimplementation ist vorgesehen, dass die Software einen gleichbleibenden Buchstaben über den USB ausgibt, sobald man einen Pin auf Low zieht. Stattdessen wird nun ADC0 abgefragt, ob der Wert unter 750 gesunken ist. Dann wird in den Morsedecoder verzweigt.

Die Morsezeichen werden zwei mal per Lookup- Befehl umcodiert: zunächst werden sie in ASCII- Bytes umgewandelt und anschließend in deutsche Tastaturbelegung.

Mittlere Geschwindigkeit und Leerzeichenmodus werden nur einmal bei Start des Programms (also bei Einstecken des USB- Steckers) festgelegt.

Die Decodersoftware hat zwar einen großen Fangbereich für unsauber gegebene Zeichen und kann daher auch (in gewissem Rahmen) Geschwindigkeitsunterschiede abfangen, aber die mittlere Geschwindigkeit ist lediglich fest auf 60 oder 90 Zeichen einstellbar (DIP- Schalter am 10 kOhm- Widerstand).

Der andere DIP- Schalter legt fest, ob Leerzeichen automatisch durch Zeitablauf oder ein spezielles Morsezeichen generiert werden (letzteres ist mühsamer, aber verhindert, dass bei zögerlicher Gebeweise ungewollte Leerzeichen eingestreut werden).

Als akustische Rückmeldung über den gewählten Modus wird die Tonhöhe des Mithörtons umgeschaltet.

Da es keine Morsezeichen für Return und Leerzeichen gibt, habe ich die zusammengesetzten Zeichen <kn> und <as> eingebaut.

4 Kompilieren des Codes

Wegen der restriktiven Lizenz für swusb.lbx habe ich dem Sourcecode lediglich das File swusb-includes.bas beigefügt - swusb.lbx ist auf der oben angegebenen Homepage des Projekts herunterzuladen und in den BASCOM- Plugin-Ordner zu verschieben.

5 Fusen des AtTiny

Ein fabrikfrischer Attiny muss für Quarzbetrieb ohne 1:8- Vorteiler gefust werden; Watchdog und Browout sind nicht aktiviert:

Low Fuse = 0xFF High Fuse = 0xDF