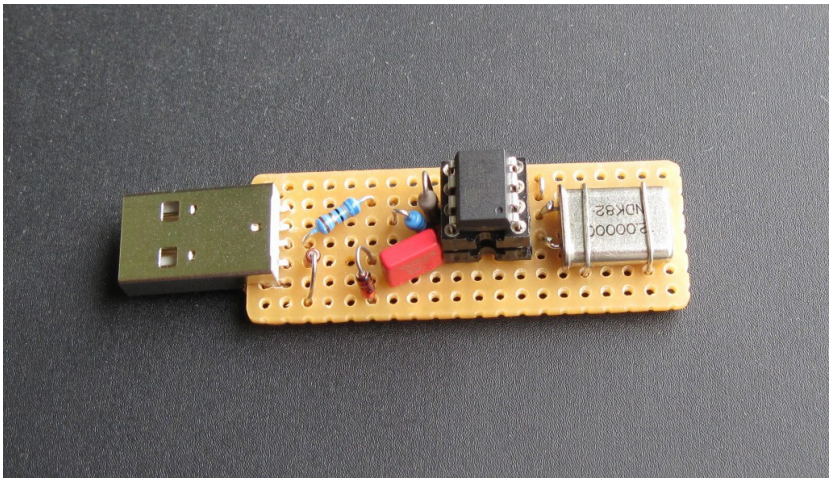
SWUSB: AVR-Software-USB mit Bascom

Ralf Beesner

9. September 2012



1 Überblick

Es gibt einige AVR- Chips mit USB- Hardware. Wenn es gilt, die übrigen 8bit-AVRs USB- fähig zu machen, ist die VUSB- Software `www.obdev.at/products/vusb/projects-de.html` die etablierte Lösung. Schon seit einigen Jahren ist Rick Richards aktiv, um eine vergleichbare Lösung unter BASCOM zu entwickeln. Er tritt im BASCOM- Forum unter dem Nick "ollopa" auf; dort wird der Fortschritt seiner Lösung begleitet und diskutiert:

`http://www.mcselec.com/index2.php?option=com_forum&Itemid=59&page=viewtopic&t=7537`

Offenbar hat er wenig Zeit, um sich dem Projekt zu widmen, aber inzwischen ist seine Software recht brauchbar. Der Kern, die Library swusb.lbx und ein include- File sowie (derzeit) zwei Beispielimplementationen (ein Gamepad und ein Keyboard) finden sich unter `www.sloservers.com/swusb`.

Die Library steht unter einer restriktiven Lizenz, ist aber in der kompilierten Version (*.lbx) derzeit frei für nichtkommerziellen Einsatz (vielleicht wird sie mal ein kostenpflichtiges Addon für BASCOM).

Sie ist nicht ganz so universell wie VUSB, z.B. wird der PLL- Takt der AtTinies 25/45/85 nicht unterstützt, sondern es ist stets ein Quarz erforderlich (12 oder 15 MHz). Außerdem ist der Code so umfangreich, dass mindestens 4 kB Flashspeicher erforderlich sind, also mindestens ein AtTiny 45 oder AtTiny 44 eingesetzt werden muss.

Mit SWUSB kann man jedoch in der vertrauten BASCOM- Umgebung ähnlich "coole" Sachen machen wie mit VUSB, ohne sich mit der sperrigen Programmiersprache "C" herumzuplagen.

Ich habe mit ollopas Keyboard- Beispielimplementation einige VUSB- Projekte in BASCOM nachgebaut:

- Serial Keyboard
- Slideshow Presenter
- Tremor Mouse

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.

Dies reicht so gerade eben für das Serial Keyboard (es nimmt ein Signal über seine serielle Schnittstelle auf und leitet es über den USB als Tastatureingaben weiter, so dass man z.B. Messwerte direkt in eine Tabellenkalkulation eintragen kann).

PB1 wird als Software-UART verwendet und benötigt keinen Pegelwandler, da die Polaritätsumwandlung durch den Software-UART erfolgt. Es ist lediglich ein Vorwiderstand zur Strombegrenzung (in Verbindung mit den internen Schutzdioden des AtTiny) erforderlich.

Für den Slideshow Presenter werden jedoch zwei freie Pins benötigt (der Slideshow Presenter hat zwei Tasten (Bild rauf / Bild runter), mit denen man durch eine Powerpoint- Präsentation oder ein PDF blättert).

Zunächst wollte ich den Reset- Pin opfern und zu einem normalen digitalen Eingang umfusen, aber ich erinnerte mich an Schaltungsbeiträge von Hermann Nieder, die den Reset- Pin über einen Trick nutzbar machen: 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 als 1/2 VCC ist, damit kein Reset ausgelöst wird.

Die beiden Tasten werden daher über einen Spannungsteiler an PB5/ADC0 angeschlossen. Die Widerstände sind so bemessen, dass die Spannung stets höher als 1/2 VCC ist. Der ADC gibt bei offenen Tastern 1023 aus; wird der Taster an PAD2 betätigt, sinkt der Wert knapp unter 950, wird der Taster an PAD3 betätigt, sinkt er knapp unter 900.

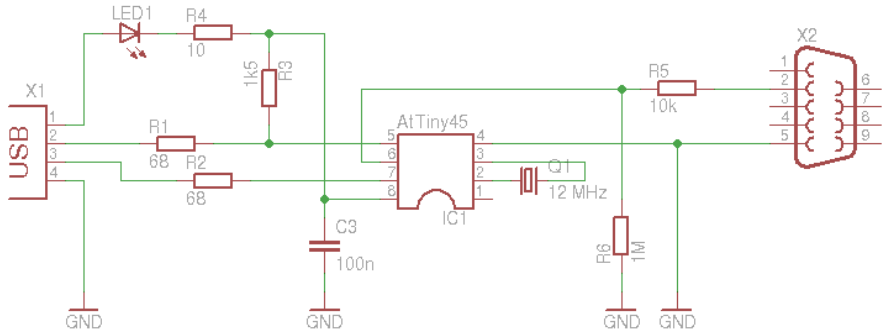


Abbildung 1: Schaltbild Serial Keyboard

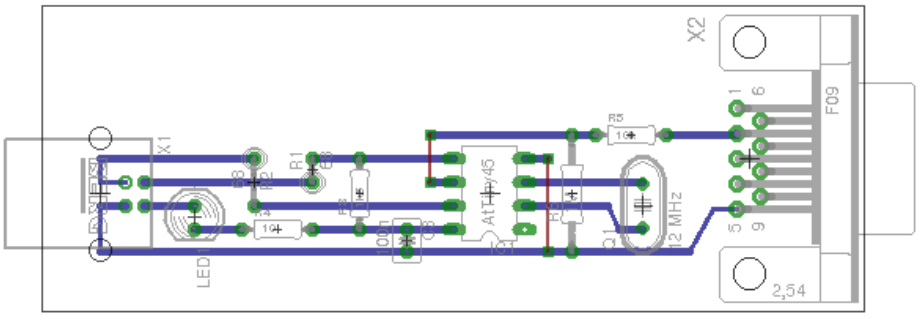


Abbildung 2: Streifenraster- Layout

3 Software des Serial Keyboard

Ollopas Keyboard- Beispiel- Implementation habe ich nur ansatzweise verstanden, aber glücklicherweise erkennt man leicht die Stellen, an denen man sich in seinen 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. Ich habe einfach mal versucht, an dieser Stelle auf den Befehl "Inkey" zu verzweigen, vor dem eigentlich immer gewarnt wird, weil er den Prozessor blockiert, wenn er kein Signal empfängt. Die Gefahr besteht hier jedoch nicht, weil das "Low", bei dem auf "Inkey" verzweigt wird, der Beginn eines Startbits ist.

Der Software- UART ist wesentlich weniger leistungsfähig als der Hardware-UART, weil er sich um jedes Bit einzeln kümmern muss, statt ein komplettes Byte bei der Hardware "abholen" zu können. Dennoch funktioniert die Lösung

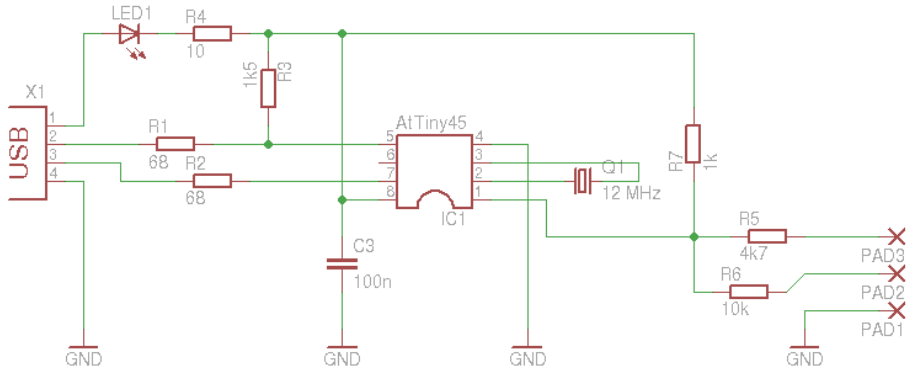


Abbildung 3: Schaltbild Slideshow Presenter

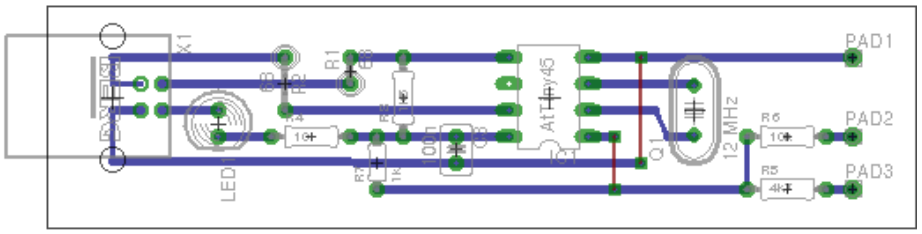


Abbildung 4: Streifenraster- Layout

besser als erwartet.

Man darf allerdings nicht mehrere Bytes auf einmal an den Software- UART senden, dann werden Zeichern verstümmelt oder verschluckt, sondern man muss kleine Pausen zwischen den Einzelzeichen lassen, wie es z.B. der Fall ist, wenn das serielle Signal aus "händisch eingegebenen" Zeichen besteht (serielles Terminal).

Stammen die Zeichen aus einem zweiten Mikrocontroller, muss man gezielt kleine Pausen zwischen den Zeichen einlegen. Die Bitrate beträgt 1200 bit/s (maximal sind 4800 bit/s möglich), und man kann etwa 20 - 30 Bytes pro Sekunde übermitteln. Das klingt nach wenig, dürfte aber für viele Fälle reichen (z.B. die automatische Eintragung von Messwerten in eine Excel- Tabelle).

Eine Eigenheit der Ollopa- Software ist, dass man erst "lossenden" darf, wenn der AtTiny und der PC die USB- Verbindung vollständig ausgehandelt haben, sonst wird das Serial Keyboard nicht erkannt.

Wie beim VUSB Serial Keyboard reicht es nicht, die empfangenen ASCII- Zeichen unmodifiziert weiterzugeben, da der USB- Keyboard- Standard auf der US-Keyboard- Belegung beruht; der Lookup- Befehl und eine Tabelle sorgen für die Umcodierung, und für einige Zeichen müssen auch Modifier geändert werden.

4 Software des Slideshow Presenter

Hier besteht die eigene Task aus der Abfrage des ADC; sinkt der ADC- Wert unter 950 bzw. unter 900, wird der HID- Key 75 (dezimal) bzw. 78 (dezimal) gesendet.

5 Tremor Mouse

Die virtuelle Maus ist aus Ollopas Gameport- Beispiel abgeleitet. Der "usb hid report descriptor" (ab Zeile 352) muss abgeändert werden; nach einigen vergeblichen Versuchen, ihn selbst zu schreiben, habe ich irgendwo einen korrekten Descriptor für eine Maus mit 3 Tasten - aber ohne Scrollrad - gefunden und "geklaut".

Zu beachten ist, dass immer auch der Wert in Zeile 351 und Zeile 332 angepasst werden muss, wenn sich die Länge des Descriptors ändert.

Die Bytes für Tastendrücke, X- und Y- Koordinaten werden ab Zeile 260 aufbereitet. Damit der Mauszeiger wild zappelt, wird der Zufallsgenerator ein paar mal herangezogen.

6 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.

7 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