Attiny-Platine & GPS-Modul

Viele GPS-Module können ihre Daten als TTL- oder auch als RS232-Signale abgeben. Solche Module sind bestens für Experimente mit Mikrocontrollern geeignet. Als Beispiel soll hier das GPS-Modul NL-552ETTL der Firma Navilock vorgestellt. Dieses GPS-Modul kostet ca. 30 Euro, kommuniziert über TTL-Signale und sendet seine Daten gemäß dem verbreiteten NMEA-Protokoll (s. u.). Dieses Modul ist auch in einer RS232-Version unter der Bezeichnung NL-550RS erhältlich.

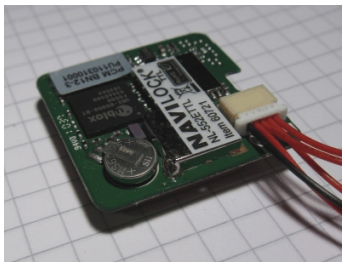

Abbildung 1

Die TTL-Version hat zwar eine Betriebsspannung von 5 V, sie arbeitet aber mit TTL-Signalen von 3,3 V; wenn man dieses Modul mit der Attiny-Platine betreiben möchte, ist daher etwas Vorsicht geboten: Das GPS-Modul kann nämlich Schaden nehmen, wenn es mit 5-V-Signalen der Attiny-Platine betrieben wird; die Firma Navilock empfiehlt für solche Fälle Pegelwandler.

Wird das GPS-Modul an die Stromversorgung angeschlossen, beginnt es unverzüglich, seine Datenströme über seine TxD-Leitung auszusenden. Eine Aufforderung durch den Mikrocontroller - etwa in Form spezieller Signale - ist nicht erforderlich. Wir benötigen also nur eine einzige Datenleitung vom TxD-Ausgang des GPS-Moduls zum RxD-Eingang des Mikrocontrollers. Da dieser Mikrocontroller-Eingang im USART-Betrieb hochohmig ist, erübrigt sich in diesem Fall der oben erwähnte Pegelwandler.

Allerdings muss unserer Attiny-Platine leider doch noch mit dem Lötkolben zu Leibe gerückt werden. Für die Invertierung des RS232-Signals der COM-Schnittstelle ist der RxD-Eingang des Mikrocontrollers nämlich über einen 4,7-K-Widerstand mit +5V verbunden. Dieser Widerstand sollte für den GPS-Betrieb “entfernt” werden. Damit die Programmierung über die COM-Schnittstelle (RS232) weiterhin möglich bleibt, empfiehlt sich folgende Vorgehensweise:

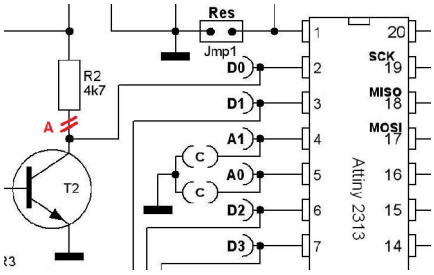

Abbildung 2

- den zum Emitter des Transistors weisenden Anschluss des 4,7-K-Widerstands (bei A in Abb. 2) auslöten und mit einem kurzen Patchkabel versehen,

- im GPS-Betrieb dieses Patchkabel unverbunden lassen und den TxD-Ausgang direkt mit dem RxD-Eingang (Pin D0) verbinden,

- im Programmierbetrieb das Patchkabel mit Pin D0 verbinden.

Wer derartige Umbauten an seiner Attiny-Platine vermeiden möchte, der greife lieber auf die RS232-Version zurück. Allerdings ist dieses Modul auch ein paar Euro teurer.

Das GPS-Modul von Navilock sendet seine Daten mit einer Rate von 38400 Baud (also nicht mit der im NMEA-Standard vorgegebenen Rate von 4800 Baud). Für dieses Tempo ist der im Attiny eingebaute Taktgenerator nicht präzise genug. Um eine zuverlässige Kommunikation zwischen GPS-Modul und Attiny zu gewährleisten, muss ein Quarz eingesetzt werden. Welche Schritte

erforderlich sind, um unseren Attiny mit einem Quarz zu betreiben sind, ist unter http://www.g-heinrichs.de/attiny/Einbau_und_Aktivieren_eines_Quarzes.pdf nachzulesen. Allerdings benötigen wir diesmal keinen 4,000 MHz-Kristall, sondern einen mit 3,6463 MHz. (Der Grund: Für 4,000 MHz ergibt sich ein "krummer" Wert für die Frequenzuntersetzung; dies führt zu Fehlern von 7%. Bei 3,6463 MHz gibt es diesen Fehler nicht. Vgl. S. 135 des Attiny2313-Manuals!)

Zum Anschließen benötigt man ein Anschlusskabel wie in Abb. 3 (Navilock Artikel 95843). Standardmäßig wird das GPS-Modul NL-552ETTL ohne dieses Anschlusskabel ausgeliefert. Der Anschluss an die Attiny-Platine erfolgt gemäß der folgenden Tabelle:

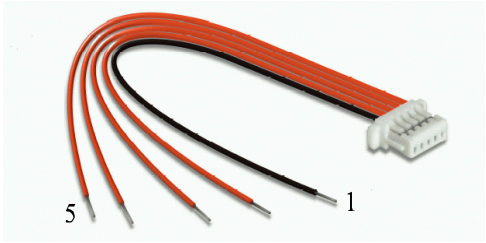

Abbildung 3

Nr	GPS-Modul	Attiny-Platine
1	+ 5 V	+ 5 V
2	Masse	Masse
3	Abschirmung	
4	TxD (3,3 V)	RxD (hochohmig)
5	RxD (3,3 V)	

Widmen wir uns jetzt der Software. Hier spielt das NMEA-Protokoll eine entscheidende Rolle. Nach diesem Protokoll werden die Daten blockweise übertragen. Ein kompletter Block Daten sieht dann beispielsweise beim Garmin etrex Vista (Software Version 2.41) folgendermaßen aus (nach http://www.kowoma.de/gps/zusatzerklaerungen/NMEA.htm):

```
$GPRMC,191410,A,4735.5634,N,00739.3538,E,0.0,0.0,181102,0.4,E,A*19
$GPRMB,A,9.99,L,,Exit,4726.8323,N,00820.4822,E,29.212,107.2,,V,A*69
$GPGGA,191410,4735.5634,N,00739.3538,E,1,04,4.4,351.5,M,48.0,M,,*45
$GPGSA,A,3,,,,15,17,18,23,,,,,,4.7,4.4,1.5*3F
$GPGSV,2,1,08,02,59,282,00,03,42,287,00,06,16,094,00,15,80,090,48*79
$GPGLL,4735.5634,N,00739.3538,E,191410,A,A*4A
$GPBOD,221.9,T,221.5,M,Exit,*6B
$GPVTG,0.0,T,359.6,M,0.0,N,0.0,K*47
$PGRME,24.7,M,23.5,M,34.1,M*1D
$PGRMZ,1012,f*36
$PGRMM,WGS 84*06
$HCHDG,170.4,,,0.4,E*03
$GPRTE,1,1,c,*37
```

Jeder Block besteht also aus einzelnen Datensätzen, die jeweils mit einem entsprechenden Kopf versehen sind. Dieser Kopf besteht immer aus einer Abkürzung, die durch ein Dollarzeichen eingeleitet wird. Die einzelnen Daten (Items) werden diesem Kopf - jeweils durch ein Komma getrennt - angefügt.

Der GPRMC-Datensatz (RMC = recommended minimum sentence C, empfohlener Minimumdatensatz) ist eine Empfehlung für das Minimum, was ein GPS-Empfänger ausgeben soll (NMEA 0183 V2.3).

```
$GPRMC,191410,A,4635.5634,N,00739.3538,E,0.0,0.0,181102,0.4,E,A*19
       ^      ^ ^           ^            ^   ^   ^      ^     ^
       |      | |           |            |   |   |      |     |
       |      | |           |            |   |   |      |     Neu in NMEA 2.3:
       |      | |           |            |   |   |      |     Art der Bestimmung
       |      | |           |            |   |   |      |     A=autonomous (selbst)
       |      | |           |            |   |   |      |     D=differential
       |      | |           |            |   |   |      |     E=estimated (geschätzt)
       |      | |           |            |   |   |      |     N=not valid (ungültig)
       |      | |           |            |   |   |      |     S=simulator
       |      | |           |            |   |   |      |
       |      | |           |            |   |   |      Missweisung (mit Richtung)
       |      | |           |            |   |   |
       |      | |           |            |   |   Datum: 18.11.2002
       |      | |           |            |   |
       |      | |           |            |   Bewegungsrichtung in Grad (wahr)
       |      | |           |            |
       |      | |           |            Geschwindigkeit über Grund (Knoten)
       |      | |           |
       |      | |           Längengrad mit (Vorzeichen)-Richtung (E=Ost, W=West)
       |      | |           007° 39.3538' Ost
       |      | |
       |      | Breitengrad mit (Vorzeichen)-Richtung (N=Nord, S=Süd)
       |      | 46° 35.5634' Nord
       |      |
       |      Status der Bestimmung: A=Active (gültig); V=void (ungültig)
       |
       Uhrzeit der Bestimmung: 19:14:10 (UTC-Zeit)
```

Das Navilock-GPS NL-552ETTL arbeitet mit der Version 3.1. Beim GPMRC-Datensatz zeigen sich nur geringe Abweichungen gegenüber der Version 2.3. Zum Beispiel werden bei der Uhrzeit nun auch die Millisekunden angegeben.

Unser Programm geht auf Ideen von C. Kühnel (Programmieren der AVR RISC Mikrocontroller mit BASCOM-AVR: Eine Einführung anhand von Programmbeispielen) zurück. Es soll nur den GPMRC-Datensatz auswerten. Dazu untersucht das Programm den ankommenden Datenstrom und wartet so lange, bis es auf die Zeichenkette “$GPMRC,” stößt. Danach überspringt es eine bestimmte Anzahl von Kommata; diese Anzahl kann mit den Tasten Ta0 und Ta1 vergrößert bzw. verkleinert werden. Das auf das letzte Komma folgende Item wird schließlich auf einem angeschlossenen LCD angezeigt. Nach einer Zeit von zwei Sekunden beginnt alles von vorne. So kann man sich jedes Item des GPMRC-Datensatzes anzeigen lassen. Andere Datensätze lassen sich natürlich auf die gleiche Weise behandeln.

Ganz bewusst wurde hier nicht der komplette Datensatz in einer einzigen Zeichenkette gespeichert; denn dafür besitzt der Attiny2313 einfach nicht genug SRAM-Speicher, zumal der benutzte Software-Puffer für die USART auch etwas Speicherplatz kostet.

Programm-Listing

```
' Navilock an Attiny-Platine
' 4,7K-Widerstand bei RTX auslöten
' spezieller Quarz, s. u.
' zeigt jede 2 Sekunden einen Eintrag von $GPMRC an;
' mit Ta0 und Ta1 Nummer des Items auswählen

$regfile = "attiny2313.dat"
$crystal = 3686400                ' damit USART präzise bei 38400 baud arbeitet
$baud = 38400                     ' NMEA Schnittstelle 34800 baud
$hwstack = 32
$swstack = 10
$framesize = 40

Dim Soutput As String * 20
Dim I As Byte
Dim Ss As Byte
Dim Cc As Byte
Dim Wi As Byte
```

```
Dim Pos As Byte

Declare Sub Wait_for_string(byval S As String)
Declare Sub Wait_for_char(byval C As String)
Declare Sub Read_string()

Ddrd = &B01110000                    ' D4, D5, D6 als Ausgang; Rest als Eingang
Portd = &B10001110                   ' Beginn mit Item 0

Pos = 0

Enable Int0
Config Int0 = Falling
On Int0 Vzaehlen Nosave              ' Rechenregister in ISR nicht retten
Enable Int1
Config Int1 = Falling
On Int1 Rzaehlen Nosave

Config Lcd = 16 * 2                  ' LCD konfigurieren
Config Lcdpin = Pin , Db4 = Portb.0 , Db5 = Portb.1 , Db6 = Portb.2 , Db7 = Portb.3 , E =
       Portb.4 , Rs = Portb.6
Cursor Off

Config Serialin = Buffered , Size = 15  ' Konfig. der ser. Eingabe
Enable Interrupts

Initlcd
Cls
Lcd "Test GPS-Modul"
Wait 1
Cls

Do
   Call Wait_for_string( "$GPRMC,")    ' Warte auf $GPRMC
   Portd.6 = 1
   For I = 1 To Pos
     Call Wait_for_char( ",")
   Next I
   Call Read_string()
   Initlcd
   Cls
   Locate 1 , 1
   Lcd "Item ";
   Lcd Pos
   Locate 2 , 1
   Lcd Soutput
   Wait 1
   Portd.6 = 0
   Wait 1
   Clear Serialin
Loop

End

' ****************************************************************************************

Sub Wait_for_char(byval C As String)
Do
   Cc = Waitkey()
Loop Until Cc = Chr(c)
End Sub

Sub Wait_for_string(byval S As String)
  Wi = 1
  M1:
   Cc = Waitkey()
   If Cc <> Mid(s , Wi , 1) Then
      Wi = 1                           'bei Kuehnel vergessen?!
      Goto M1
   Else
      Incr Wi
      If Wi > Len(s) Then Goto M2
      Goto M1
   End If
M2:
End Sub

Sub Read_string()
Soutput = ""
Do
```

```
   Cc = Waitkey()
   Soutput = Soutput + Chr(cc)
Loop Until Cc = ","
End Sub


Vzaehlen:
  Pos = Pos + 1                  ' hier kaum ein Problem mit Tastenprellern - wegen Kondensator
Return

Rzaehlen:
  If Pos > 0 Then Pos = Pos - 1  ' manchmal kleines Problem bei Tastenprellern :-)
Return
```