Kommunizieren mit Hilfe von rfm12-Funkmodulen

In diesem Kapitel wollen wir zeigen, wie man eine Funkverbindung zwischen zwei Computern aufbauen kann. Dabei benutzen wir den Transceiver-Baustein rfm12. Dieser Baustein ist in der Lage, zu senden und zu empfangen - und dies über eine Strecke von bis zu 100 m. Er ist preisgünstig: Bei Pollin kostet er z. Zt. 4,95 Euro. Leider ist die mitgelieferte Anleitung sehr knapp gehalten, und die Original-Manuals von HopeRF sind für den Einsteiger eine recht schwere Kost, beschränken sie sich im Wesentlichen doch auf die Beschreibung der einzelnen Kontrollbits. Und davon gibt es sehr, sehr viele!

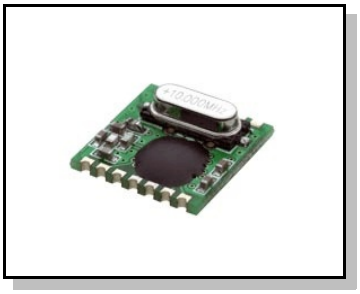

Abb. 1: Funkmodul rfm12

Wir wollen hier auch nicht eine Luxus-Funkverbindung vorstellen; vielmehr kommt es uns darauf an, die wesentlichen Grundzüge aufzuzeigen. Auf diesen aufbauend kann dann der Leser selbst Verbesserungen nach eigenen Vorstellungen vornehmen.

Abbildung 2 zeigt, wie die Verbindung aussehen soll: Beide PCs sind über ein COM-Kabel (oder USB-COM-Kabel) jeweils mit einer Attiny-Platine verbunden; an diese wiederum ist ein rfm12-Modul angeschlossen. Der Sende-PC schickt über ein Terminal-Programm einen Datenstrom (bei uns maximal 10 Byte) an den Attiny. Der Mikrocontroller sendet die Informationen an das angeschlossene rfm12-Modul; dieses muss natürlich als Sender konfiguriert sein. Die Kommunikation zwischen Attiny und rfm12 erfolgt dabei über das SPI-Protokoll. Das Sende-Modul sendet die Daten in so genannten Paketen (s. u.) aus. Diese werden von dem zweiten rfm12-Modul, welches als Empfänger eingestellt ist, aufgenommen und von dem zweiten Attiny abgefragt. Der wiederum gibt die empfangenen Bytes über die COM-Schnittstelle an das Terminal-Programm auf dem Empfangs-PC weiter, und so haben wir letztlich eine Funkverbindung zwischen den beiden PCs. Natürlich könnten wir auf Sender- oder Empfängerseite auch auf einen PC verzichten und die Mikrocontroller selbstständig arbeiten lassen. Für unsere einführenden Betrachtungen wäre das aber nicht so sinnvoll.

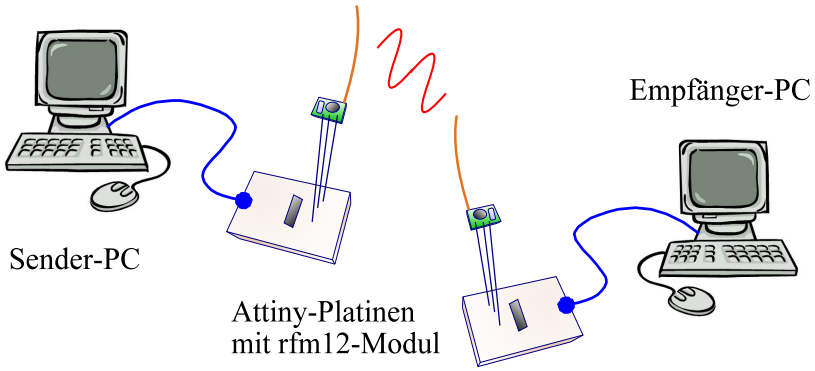

Abb. 2: Funkübertragung zwischen zwei Computern

Der Attiny2313 als SPI-Master

Um Kontakt mit dem rfm12-Modul aufnehmen zu können, muss der Attiny2313 als SPI-Master eingesetzt werden. Dabei muss man beachten, dass PB5 weiterhin als Dateneingang DI arbeitet und daher jetzt als MISO angesehen werden muss. An dieser Stelle ist die Pin-Beschreibung des Attiny2313-Datenblattes (S. 2) nicht ganz korrekt. Dementsprechend ist PB6 = DO = MOSI; PB7 bleibt SCK.

Als Master muss der Attiny nun selbst die Taktsignale erzeugen. Dies kann auf verschiedene Weisen geschehen. Wir wählen hier den Weg über die Software; hierbei muss jedes Taktsignal einzeln durch einen entsprechenden Befehl hervorgerufen werden. Zuvor muss der Mikrocontroller aber erfahren, dass wir diese Methode gewählt haben; dazu führen wir folgende Initialisierungen durch:

Zunächst löschen wir wieder den SPI-Zähler und das USIOIF-Bit: Usisr = &B01000000 (vgl. auch das Kapitel über SPI). Dann stellen wir mit USICR.USIWM0 = 1 und USICR.USIWM1= 0 den 3-wire-mode ein. Mit dem USITC-Bit des Registers USICR kann der Zustand des Taktsignals getoggelt werden; genauer: Jedesmal, wenn das USITC-Bit auf 1 gesetzt wird, dann wird das USICLK-Bit von 0 auf 1 oder von 1 auf 0 gesetzt. Damit der Mikrocontroller auch genau so arbeitet, muss er entsprechend eingestellt sein: USICR.USICS1 = 1, USICR.USICS0 = 0 und USICR.USICLK=1. Das entspricht der vorletzten Zeile (software clock strobe über USITC) in der folgenden Tabelle:

USICS1	USICS0	USICLK	Shift Register Clock Source	4-bit Counter Clock Source
0	0	0	No Clock	No Clock
0	0	1	Software clock strobe (USICLK)	Software clock strobe (USICLK)
0	1	X	Timer/Counter0 overflow	Timer/Counter0 overflow
1	0	0	External, positive edge	External, both edges
1	1	0	External, negative edge	External, both edges
1	0	1	External, positive edge	Software clock strobe (USITC)
1	1	1	External, negative edge	Software clock strobe (USITC)

Abb. 3: Die Taktmöglichkeiten der USI-Einheit (*aus: Attiny2313-Manual*)

Der Rest ist jetzt einfach und kann dem folgenden, kommentierten Programm entnommen werden. Die Schleife wird dabei genau 16 mal durchlaufen. Mit jeder positiven und jeder negativen Flanke des Taktsignals wird nämlich der 4-Bit-Zähler des USISR hochgezählt. Beim 16. Mal kommt es zu einem Überlauf und das USIOIF-Bit wird gesetzt.

```
Function Spi(s As Byte) As Byte
  Ddrb = &B11011111
  Usidr = S
  Usisr = &B01000000          'USIOIF und Zähler löschen
  Usicr = &B00011010          'USIWM0, USICS1, USICLK auf 1
  Do
    Usicr.usitc = 1           'Takten durch Setzen von usitc
  Loop Until Usisr.usioif = 1 '16 mal toggeln
  Spi = Usidr
End Function
```

Alternativ könnte man auch die BASCOM-eigenen SPI-Befehle benutzen; allerdings greifen diese nicht auf die USI-Komponente zurück und benötigen deswegen mehr Programmspeicher.

Anschuss des rfm12-Moduls an die Attiny-Platine

Bezeichnung	Typ	Funktion
nINT/VDI	DI/DO	Interrupt input(active low)/Valid data indicator
VDD	S	Positive power supply
SDI	DI	SPI data input
SCK	DI	SPI clock input
nSEL	DI	Chip select(active low)
SDO	DO	Serial data output with bus hold
nIRQ	DO	Interrupts request output (active low)
FSK/DATA/ nFFS	DI/ DO/DI	Transmit FSK data input/ Received data output (FIFO not used)/ FIFO select
DCLK/CFIL/ FFIT	DO/AIO/ DO	Clock output (no FIFO)/ external filter capacitor (analog mode)/ FIFO interrupts(active high) when FIFO level set to 1, FIFO empty interruption can be achieved
CLK	DO	Clock output for external MCU
nRES	DIO	Reset output (active low)
GND	S	Power ground

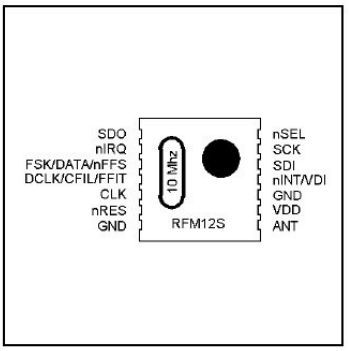

Abb. 4: Die Anschlüsse des rfm12 (*nach: Pollin-Anleitung*)

Für die von uns gewählte Betriebsart benötigen wir nur wenige Anschlüsse des rfm12-Moduls: VDD, GND, SDI (MOSI), SCK, SDO (MISO), nSEL und FSK. Sie werden wie in der folgenden Tabelle angegeben mit der Attiny-Platine verbunden:

rfm12-Modul	**Attiny-Platine**
VDD	+ 5 V
GND	Masse
SDI	PB6
SDO	PB5

rfm12-Modul	Attiny-Platine
SCK	PB7
nSEL	PB4
FSK	über 10K-Pullup-Widerstand an + 5 V

An den Anschluss ANT des rfm12-Moduls löten wir einen Draht als Antenne an. Als Länge *l* dieser Antenne sollte man ein Viertel der Wellenlänge wählen. Da wir mit einer Frequenz von *f* = 433 MHz arbeiten werden, erhalten wir dafür den Wert

$$l = \frac{\lambda}{4} = \frac{c}{4 \cdot f} = \frac{300\,000\,000\ \frac{\mathrm{m}}{\mathrm{s}}}{4 \cdot 433\,000\,000\ \mathrm{Hz}} \approx 0{,}17\ \mathrm{m}.$$

In dieser Formel steht *c* für die Lichtgeschwindigkeit, die auch gleichzeitig die Ausbreitungsgeschwindigkeit der elektromagnetischen Wellen ist.

Die 3 Leitungen an SDI, SDO und SCK bilden die bekannte 3-wire-Verbindung des SPI-Busses. Mit dem Anschluss nSEL wird der rfm12-Baustein aktiviert (nSEL = 0) oder deaktiviert (nSEL = 1). Der Anschluss FSK wird von uns nicht benutzt; das Datenblatt des rf12-ICs, der auf dem rfm12-Modul montiert ist, schreibt für diesen Fall vor, dass der Anschluss mit einem Pullup-Widerstand auf High gezogen werden muss.

Bemerkungen zu Funktionsweise und Ansteuerung des rfm12-Moduls

Das rfm12-Modul kann sowohl als Sender als auch als Empfänger benutzt werden. Dabei kann er mit unterschiedlichen Frequenzen betrieben werden. Auch für die Datenübertragung zwischen Mikrocontroller uns rfm12-Modul gibt es verschiedene Möglichkeiten. Ferner kann das Modul auch ein Taktsignal für den Mikrocontroller erzeugen. Diese und weitere Einstellungen müssen im Rahmen einer Konfiguration vorgenommen werden.

Dazu dienen spezielle Kommandos. Diese bestehen immer aus einem Wort (Doppelbyte). Meistens stellt das erste Byte den Befehlscode dar und das zweite Byte die Parameter. Am Beispiel des Power Management Command (Abb. 5) wollen wir dies erläutern: Sein Befehlscode ist $82.Die Starteinstellung für die Parameter $08 hat das Bitmuster &B00001000. Das heißt: Beim Einschalten sind lediglich der Synthesizer und das Taktsignal an CLK aktiviert.

Wir wollen den Baustein zunächst als Sender benutzen. Deswegen müssen wir das Bit `et` aktivieren (enable transmitter = Sender aktivieren); außerdem muss für den Sendebetrieb auch der Synthesizer mitsamt dem Quarz-Oszillator eingeschaltet werden (`es` = 1 und `ex` = 1). Auf das Clock-Signal am CLK-Pin verzichten wir, also setzen wir `dc` = 1. Die Bitkombination &B00111001 ist im Hexadezimalsystem gerade $39. Das gesamte Kommando zum Einstellen dieser Eigenschaften ist demnach $8239.

3. Power Management Command

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	POR
	1	0	0	0	0	0	1	0	er	ebb	et	es	ex	eb	ew	dc	8208h

er: Enable receiver
ebb: Enable base band block
et: Enable transmitter
es: Enable synthesizer
ex: Enable crystal oscillator
eb: Enable low battery detector
ew: Enable wake-up timer
dc: Disable clock output of CLK pin

Abb. 5: Die einzelnen Parameter des Power Management Command (*nach: rfm12-Datasheet*)

Wir werden hier nicht auf alle einzustellenden Parameter eingehen können. Unsere Darstellung orientiert sich an den Empfehlungen der Kurzanleitung von Pollin (vgl. Abb. 7). Die einzelnen Kommandos sind hier in hexadezimaler Form angegeben und mit den zugehörigen Parameterkürzeln kommentiert, sofern sie von der Starteinstellung (initial value) abweichen. In unserem Fall sieht das so aus:

Kommando	**Kommentar**
$8239	!er, !ebb, et, es, ex, !eb, !ew, dc

Dabei bedeutet das Ausrufezeichen, dass das entsprechende Bit *nicht* gesetzt ist.

Nun folgt der eigentliche Sendevorgang. Jedes einzelne Byte, welches gesendet werden soll, wird mit dem Byte $B8 kombiniert: Soll z. B. das Byte $3F gesendet, muss dazu der Befehl $B83F an das rfm12-Modul geschickt werden. Mit anderen Worten: $B8 ist die Befehlsnummer des Sendekommandos und $3F ist hier als Sendeparameter zu sehen. Vor jedem Sendekommando muss überprüft werden, ob das rfm12-Modul sendebereit ist. Dazu wird kontrolliert, ob SDO = 1 ist.

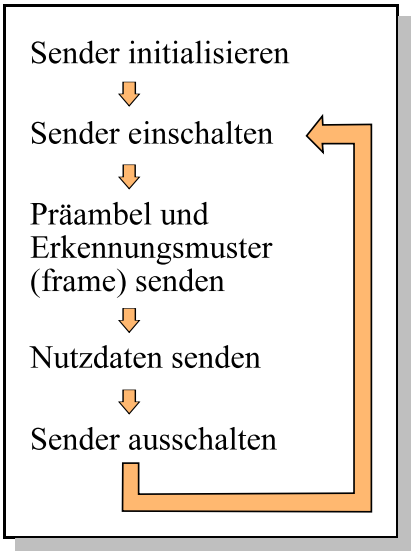

Abb. 6: Sende-Schema

Nun kann man aber nicht sofort mit dem Senden der Nutzdaten beginnen. Diesen muss nämlich eine sogenannte Präambel vorangestellt werden. In unserem Fall besteht diese Präambel aus der Bytefolge $AA-$AA-$AA. Durch das Senden dieser Bytes soll eine Synchronisation zwischen Sender und Empfänger hergestellt werden. Anschließend

muss das Empfängermodul noch über den Beginn eines neuen Datenpakets informiert werden. Dazu werden die beiden Identifikationsbytes $2D und $D4 gesendet. Jetzt erst können die Nutzdaten Byte für Byte gesendet werden. Die Übertragung des Datenpakets endet mit dem Kommando $8201; dadurch wird alles ausgeschaltet. Für das Senden eines weiteren Datenpakets muss der Transmitter natürlich wieder aktiviert werden; die restlichen Konfigurationen müssen nicht erneut vorgenommen werden.

Das Sendeprogramm

Der Händler des rfm12-Moduls (Pollin) gibt folgende Hinweise zum Senderprogramm (Abb. 7). Manche Befehle sind hier allerdings fragwürdig: So ist z. B. unklar, warum der Sendevorgang durch eine Statusabfrage eingeleitet werden sollte. In der Tat arbeitet der Sender auch ohne diese Abfrage einwandfrei. Ähnliches gilt auch für das Kommando $CA81.

Beispielanwendung RFM12 (Senden)

```
0x80D7      ;EL,EF,12.0pF
0x8239      ;!er,!ebb,ET,ES,EX,!eb,!ew,DC
0xA640      ;A140=430.8MHz
0xC647      ;19.2kbps
0x94A0      ;VDI,FAST,134kHz,0dBm,-103dBm
0xC2AC      ;AL,!ml,DIG,DQD4
0xCA81      ;FIFO8,SYNC,!ff,DR
0xC483      ;@PWR,NO RSTRIC,!st,!fi,OE,EN
0x9850      ;!mp,9810=30kHz,MAX OUT
0xE000
0xC800
0xC400      ;1.66MHz CLK-Pin, 2.2V Battery

0x0000      ;Status Register lesen
0x8239      ;!er,!ebb,ET,ES,EX,!eb,!ew,DC

            ;vor jeder Übertragung von zu sendenden Daten an das Modul muss auf nIRQ
            ;gewartet werden um sicherzustellen, dass die vorherige Übertragung
            ;abgeschlossen ist.
            ;die zu senden Daten werden mit dem Befehl 0xB800 kombiniert und über SPI
            ;übertragen, soll z.B. das Nutzdaten-Byte 0x35 gesendet werden, so wird an das
            ;Modul 0xB835 über SPI übertragen, sobald nIRQ bereit ist.
0xAA        ;PREAMBLE an Modul übertragen
0xAA        ;PREAMBLE an Modul übertragen
0xAA        ;PREAMBLE an Modul übertragen
0x2D        ;HI Byte für Frame-Erkennung an Modul übertragen
0xD4        ;LOW Byte für Frame-Erkennung an Modul übertragen
Datenbyte   ;eigentliches Nutzdaten-Byte an Modul übertragen


0x8201
```

Abb. 7: Hinweise zur Programmierung des rfm12 (*nach: Pollin-Anleitung*)

Unser BASCOM-Programm für den Attiny2313 überträgt maximal 10 Zeichen, die man über ein Terminal-Programm eingeben kann:

```
' Datei für Attiny-Platine von E. Eube, G. Heinrichs und U. Ihlefeldt
' rfm12-Transceiver mit 10k als Pullup für FSK
' Achtung: MOSI auf PB6, MISO auf PB5,
' CLK auf PB7, CS auf PB4
' LED an PB0 zur Kontrolle, ob rfm12 beschäftigt
'
' ****************************** Senden ***********************************

$regfile = "attiny2313.dat"              'Attiny2313
$crystal = 4000000                       '4 MHz
$baud = 9600
$hwstack = 40
$framesize = 50

'*********************************************************
'******************* Deklarationen ***********************

Declare Function Spi(s As Byte) As Byte
Declare Function Rfcommand(byval W As Word) As Word
Declare Sub Transinit
Declare Sub Senden(byval Laenge As Byte)
Declare Sub Tx(byval S As Byte)
Declare Sub Rf12ready
Dim Highbyte As Byte
Dim Lowbyte As Byte
Dim Temp As Word
Dim Datenbyte As Byte
Dim I As Byte
Dim Lzk As Byte
Dim Zk As String * 10
Dim Puffer(10) As Byte At Zk Overlay    'um auf die einzelnen Bytes von ZK zugreifen
                                         zu können

'****************** Initialisierung ***********************

Ddrb = &B11011111                       'Port B als Ausgangsport, außer B5
Ddrd = &B01110000                       'D4, D5, D6 als Ausgang; Rest als Eingang
Portd = &B10001111                      'Eingänge auf High legen
Portb.4 = 1                             'SPI-Chip-Select (CS) aus


'*********************************************************
'******************** Hauptprogramm ***********************

Waitms 100                              'Sender braucht seine Zeit
Call Transinit
Do
  Input Zk                               'Zeichenkette von Terminal über COM
  Lzk = Len(zk)
  Call Senden(lzk)
Loop


'*********************************************************
'******************* Unterprogramme ***********************

Function Spi(s As Byte) As Byte         'Hardware-SPI mit USI
  Usidr = S
  Usisr = &B01000000                    'USIOIF und Zähler löschen
  Usicr = &B00011010                    'USIWM0, USICS1,USICLK auf 1
  Do
    Usicr.usitc = 1                     'Takten durch Setzen von usitc
  Loop Until Usisr.usioif = 1           '16 mal toggeln
```

```
  Spi = Usidr
End Function


Function Rfcommand(w As Word) As Word
   Lowbyte = W
   Shift W , Right , 8
   Highbyte = W
   Portb.4 = 0                          'Vor dem Übertragen Modul rfm12 auswählen
   Highbyte = Spi(highbyte)
   Lowbyte = Spi(lowbyte)
   Portb.4 = 1                          'CS aus
   Temp = Highbyte
   Shift Temp , Left , 8
   Rfcommand = Temp + Lowbyte
End Function


Sub Transinit
  Temp = Rfcommand($80d7)
  Temp = Rfcommand($a640)
  Temp = Rfcommand($c647)
  Temp = Rfcommand($94a0)
  Temp = Rfcommand($c2ac)
  Temp = Rfcommand($c483)
  Temp = Rfcommand($9850)
  Temp = Rfcommand($e000)
  Temp = Rfcommand($c800)
  Temp = Rfcommand($c400)
End Sub


Sub Tx(s As Byte)
  Temp = $b800 + S
  Call Rf12ready
  Temp = Rfcommand(temp)
End Sub


Sub Senden(laenge As Byte)
  Temp = Rfcommand($8239)

  Call Tx($aa)
  Call Tx($aa)
  Call Tx($aa)
  Call Tx($2d)
  Call Tx($d4)

  If Laenge > 10 Then Laenge = 10       'maximal 10 Zeichen senden
  Call Tx(laenge)                       'zuerst Länge der Zeichenkette senden

  For I = 1 To Laenge
    Datenbyte = Puffer(i)
    Call Tx(datenbyte)
  Next I

  Call Tx(0)
  Call Tx(0)                            'Dummy-Bytes, die verloren gehen (s. u.)

  Temp = Rfcommand($8201)               'ausschalten
End Sub


Sub Rf12ready                           'bereit?
  Portb.0 = 1                           'Kontroll-LED zeigt rfm12-BUSY an
  Portb.4 = 0
  While Pinb.5 = 0
  Wend                                  'warten bis SDO = MISO = 1, dann TX bereit
```

```
  Portb.0 = 0
End Sub

'*******************************************************
```

Wird der Transmitter sofort nach dem Übertragen der Nutzdaten ausgeschaltet, so werden die letzten beiden Bytes nicht mehr aus dem Puffer des rfm12-Moduls abgeholt und sie werden dann auch nicht mehr gesendet. Um diesen Mangel zu beheben, kann man den Mikrocontroller vor dem Ausschalten einige Millisekunden warten lassen; man kann aber auch im Anschluss an die Nutzdaten zwei Dummy-Bytes senden, die dann eben geopfert werden. Wir haben uns hier für die zweite Möglichkeit entschieden.

Das Empfangsprogramm

Auch für den Empfang liefert der Händler einige kurze Informationen (Abb. 8).

Beispielanwendung RFM12 (Empfangen)

```
0x80D7      ;EL,EF,433band,11.5pF
0x82D9      ;!er,!ebb,ET,ES,EX,!eb,!ew,DC
0xA640      ;434MHz
0xC647      ;4.8kbps
0x94A0      ;VDI,FAST,134kHz,0dBm,-103dBm
0xC2AC      ;AL,!ml,DIG,DQD4
0xCA81      ;FIFO8,SYNC,!ff,DR
0xC483      ;@PWR,NO RSTRIC,!st,!fi,OE,EN
0x9850      ;!mp,9810=30kHz,MAX OUT
0xE000
0xC800
0xC400      ;1.66MHz CLK-Pin,2.2V Battery
0xCA81      ;FIFO initialisieren
0xCA83      ;FIFO aktivieren

warten auf nIRQ
0x0000      ;Status-Register
0xB000      ;empfangenes Byte lesen (Datenbyte in 16Bit-Antwort von Bit 0-7)

0xCA81      ;FIFO deaktivieren
```

Abb. 8: Hinweise zum Empfangsprogramm (*nach: Pollin*). Kommentar zu $82D9 ist falsch!

Die Konfiguration ist im Wesentlichen identisch mit der des Senders; lediglich das Sender-Bit beim Kommando $82 ist durch das Empfänger-Bit ausgetauscht. Außerdem muss das FIFO initialisiert und aktiviert werden. Hierbei handelt es sich um ein Schieberegister, welches als Empfangspuffer benutzt wird.

Das FIFO wird vom rfm12-Modul erst dann freigegeben, wenn es die oben erwähnten Identifikationsbytes empfangen hat. Wenn dann das FIFO gefüllt ist, setzt es den SDO-Ausgang auf High und teilt dem angeschlossenen Mikrocontroller auf diese Weise mit, dass ein Empfangsbyte abgeholt werden kann. Der Mikrocontroller wird daraufhin den Befehl $B000 an das Empfangsmodul schicken und gleichzeitig eine Antwort in Form eines Wortes (Doppelbyte) erhalten; von diesem stellt das niederwertige Byte das Empfangsbyte dar.

Nach dem Empfang aller Nutzdaten wird das FIFO deaktiviert. Um es für den Empfang eines weiteren Datenpaketes vorzubereiten, muss es wieder aktiviert werden.

Unser BASCOM-Programm empfängt eine Zeichenkette und zeigt sie über ein Terminalprogramm an; die Länge der Zeichenkette wird über das erste Byte des Nutzdatenstroms bezogen.

```
' Datei für Attiny-Platine von E. Eube, G. Heinrichs und U. Ihlefeldt
' rfm12-Transceiver
' Achtung: MOSI auf PB6, MISO auf PB5, CS auf PB4, FSK mit 10K-Pullup
'
' ****************************** Empfangen **********************************


$regfile = "attiny2313.dat"       'Attiny2313
$crystal = 4000000                '4 MHz
$baud = 9600

'*****************************************************
'****************** Deklarationen ********************

Declare Function Spi(s As Byte) As Byte
Declare Function Rfcommand(byval W As Word) As Word
Declare Function Empfangen() As Byte
Declare Sub Recvinit
Declare Sub Rf12ready
Dim Highbyte As Byte
Dim Lowbyte As Byte
Dim Temp As Word
Dim Datenbyte As Byte
Dim Puffer(10) As Byte
Dim I As Byte
Dim Laenge As Byte

'****************** Initialisierung ***********************

Ddrb = &B11011111                 'Port B als Ausgangsport außer B5
Ddrd = &B01110000                 'D4, D5, D6 als Ausgang; Rest als Eingang
Portd = &B10001111                'Eingänge auf High legen

Portb.4 = 1                       'SPI-Chip-Select

'*****************************************************
'******************** Hauptprogramm ********************

Waitms 100
Call Recvinit
Do
   Laenge = Empfangen()           'Laenge maximal 10
   For I = 1 To Laenge
     Datenbyte = Puffer(i)      'Printbin Puffer(i) nicht möglich (BUG bei Bascom)
     Printbin Datenbyte
   Next I
Loop
```

```
'*********************************************************
'******************* Unterprogramme **********************

Function Spi(s As Byte) As Byte
  Usidr = S
  Usisr = &B01000000              'USIOIF und Zähler löschen
  Usicr = &B00011010              'USIWM0, USICS1,USICLK auf 1
  Do
    Usicr.usitc = 1               'Takten durch Setzen von usitc
  Loop Until Usisr.usioif = 1     '16 mal toggeln
  Spi = Usidr
End Function


Function Rfcommand(w As Word) As Word
   Lowbyte = W
   Shift W , Right , 8
   Highbyte = W
   Portb.4 = 0
   Highbyte = Spi(highbyte)
   Lowbyte = Spi(lowbyte)
   Portb.4 = 1
   Temp = Highbyte
   Shift Temp , Left , 8
   Rfcommand = Temp + Lowbyte
End Function


Sub Recvinit
  Temp = Rfcommand($80d7)
  Temp = Rfcommand($82d9)
  Temp = Rfcommand($a640)
  Temp = Rfcommand($c647)
  Temp = Rfcommand($94a0)
  Temp = Rfcommand($c2ac)
  Temp = Rfcommand($ca81)
  Temp = Rfcommand($c483)
  Temp = Rfcommand($9850)
  Temp = Rfcommand($e000)
  Temp = Rfcommand($c800)
  Temp = Rfcommand($c400)
End Sub


Function Empfangen() As Byte
  Temp = Rfcommand($ca83)           'FiFo aktivieren
  Call Rf12ready
  Temp = Rfcommand($b000)
  Datenbyte = Temp                  'Lowbyte von Temp -> Länge der Zeichenkette
  For I = 1 To Datenbyte
    Call Rf12ready
    Temp = Rfcommand($b000)
    Puffer(i) = Temp                'Lowbyte von Temp -> Puffer
  Next I
  Temp = Rfcommand($ca81)           'FiFo-Init
  Empfangen = Datenbyte
End Function


Sub Rf12ready
  Portb.0 = 0
  Portb.4 = 0
  While Pinb.5 = 0
  Wend                              'warten bis SDO = MISO = 1, dann TX bereit
  Portb.0 = 1
End Sub
```

Die rfm12-Module im Betrieb

Wir gehen jetzt davon aus, dass alles korrekt angeschlossen ist: die Module an die Attiny-Platinen und diese wiederum an die Computer. Die Programme senden2.hex und empfangen2-hex sind bereits auf die Mikrocontroller hochgeladen und laufen bereits. Sowohl beim Sender-PC als auch beim Empfänger-PC starten wir nun ein Terminalprogramm (mit 9600 Baud, 1 Stoppbit, kein Paritätsbit). Wir benutzen hier das im Uploader-Programm integrierte Terminal-Programm.

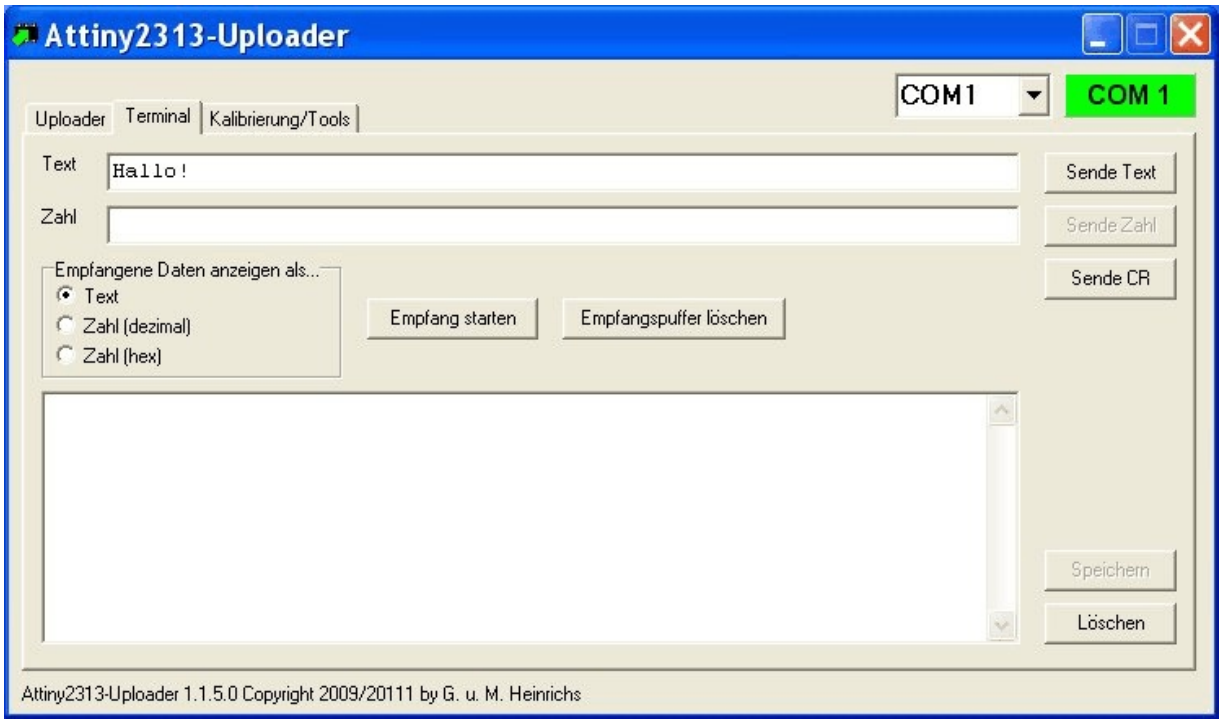

Abb. 9: Das Terminal von Uploader.exe

Zunächst aktivieren wir den Empfang im Terminalprogramm des Empfänger-PCs, indem wir die Schaltfläche <Empfangen starten> betätigen. Sollte sich jetzt schon ein Buchstabe im Empfangsbereich zeigen, können wir diesen mit der Schaltfläche <Löschen> entfernen. Nun geben wir einen Text mit maximal 10 Zeichen in das Texteingabefeld des Terminal-Programms beim Sender ein. Die Eingabe des Textes schließen wir ab mit den Schaltflächen <Sende Text> und <CR>. Die letzte Aktion sorgt dafür, dass das Terminal-Programm ein Carriage-Return-Zeichen sendet; ohne dieses Zeichen nimmt der `input`-Befehl des Sende-Programms auf dem Attiny die Zeichenkette nicht an.

Wenn Sie nun alles richtig gemacht haben, erscheint der eingegebene Text unverzüglich im Empfangsfeld des Empfänger-Terminals. Versuche zeigen, dass die Funkmodule sehr zuverlässig arbeiten und in der Reichweite mit Schnurlostelefonen oder WLAN-Netzen mithalten können. Bei mir funktionierte die Übertragung problemlos über zwei Stockwerke hinweg; dabei mussten die Funkwellen zwei Stahlbetondecken durchdringen.

Sollte die Funkverbindung nicht funktionieren, bietet sich folgende Vorgehensweise an: Zuerst kontrolliert man noch einmal alle Anschlüsse nach: Sind sie korrekt gesetzt? Haben sie guten Kontakt? Außerdem: Sind die Batterien in Ordnung?

Gegebenenfalls ist eine weitere Fehlersuche erforderlich: Dazu stecken wir auf der Sender-Platine eine LED an PB0. Diese sollte kurz aufblinken, wenn die Schaltfläche <Sende Text> betätigt wird. Wenn sie gar nicht erst angeht, bedeutet dies, dass das Programm erst gar nicht zum eigentlichen Sendevorgang gekommen ist. Geht die LED hingegen an, erlischt aber nicht mehr, hat das Funkmodul keine Bereitschaft signalisiert.

In diesem Fall sollte man der Frage nachgehen, ob das Funkmodul selbst korrekt funktioniert. Leider gibt es keine Status-LED auf dem Funkmodul; damit gibt es auch keinen direkten Hinweis darauf, ob das Modul überhaupt in irgendeiner Form arbeitet. Wenn man aber ein Oszilloskop zur Hand hat, dann kann man einiges über das Funkmodul herausfinden. Dazu entfernen wir zunächst die Taktleitung bei PB7. Dadurch kann der Attiny nach dem Einschalten keine Befehle mehr an das Funkmodul senden. Das Modul wird dann zwar mit Strom versorgt, eine Initialisierung findet aber nicht statt. In diesem Fall sollte das Funkmodul mit den Standardparametern arbeiten. Insbesondere gibt das Funkmodul dann an seinem Ausgang CLK ein eigenes Taktsignal von 1 MHz aus - nicht zu verwechseln mit dem Taktsignal bei PB7, welches vom Attiny stammt und mit dem Eingang SCK des rfm12-Moduls verbunden ist! Dieses Signal an CLK kann man mit einem Oszilloskop überprüfen. Beobachtet man kein solches Signal, könnte das ein Hinweis auf einen Defekt des Moduls sein. Wir gehen jetzt aber einmal davon aus, dass das Modul ein solches Taktsignal erzeugt. Nun schalten wir den Mikrocontroller aus und stecken die Taktleitung wieder in den Anschluss PB7. Anschließend schalten wir den Mikrocontroller wieder ein und beobachten das Signal bei CLK. Kann der Mikrocontroller korrekt mit dem Funkmodul kommunizieren, dann wird beim Starten das Funkmodul so konfiguriert, dass das rfm12-eigene Taktsignal bei CLK ausgeschaltet wird. Dementsprechend sollte auf dem Oszilloskop jetzt kein Taktsignal mehr zu beobachten sein. Im Umkehrschluss bedeutet das: Beobachtet man jetzt trotzdem immer noch ein Taktsignal, dann gibt es Probleme bei der SPI-Übertragung - und ohne diese funktioniert hier leider gar nichts!

Beim Empfänger kann man die Fehlersuche in entsprechender Weise durchführen. Auch hier kann man wieder eine LED zum Testen in PB0 stecken. Blitzt sie nicht auf, gelangt das Programm erst gar nicht zu einer Statusabfrage des Empfänger-Moduls; leuchtet sie auf, ohne auszugehen, dann empfängt das Empfänger-Modul kein Funksignal oder genauer: Der Attiny bekommt vom rfm12 keine Empfangsmeldung.

Funkempfänger mit LCD

Interessant ist es natürlich, die Funkmodule auch autonom, d. h. ohne einen PC mit Terminalprogramm einzusetzen. Auf der Empfängerseite bietet es sich an, das Terminal durch ein LCD zu ersetzen. Bei unserer Attiny-Patine benutzt das LCD den Wannenstecker an PortB. Lediglich die Anschlüsse PB5 und PB7 werden nicht vom LCD benutzt. Deswegen verlegen wir CS von PB4 auf PortD, z. B. PD2. Damit bleibt jetzt nur noch PB6 in einer Doppelfunktion: Einerseits steuert er das R/S-Signal für das LCD, andererseits das MOSI-Signal für die SPI-Kommunikation mit

dem Funkmodul. Diese sollten sich jedoch nicht stören: Wenn nämlich das LCD angesprochen werden soll, dann ist vorher schon nSEL auf high gezogen worden und damit ist das Funkmodul gesperrt. Und wenn umgekehrt das Funkmodul angesprochen wird, gibt es am E-Eingang des LCD-Controller keine Taktsignale; damit ist es egal, ob das R/S-Signal zwischendurch seinen Zustand wechselt.

Allerdings zeigen Experimente, dass an ganz anderer Stelle - ziemlich unerwartet - ein Konflikt zwischen LCD und Funkmodul auftritt. Dieser Konflikt zeigt sich darin, dass das LCD nicht mehr korrekt angesteuert wird, sobald der Mikrocontroller das Funkmodul über SPI angesprochen hat. Auch eine erneute Initialisierung des LCDs hilft hier nicht weiter! Der Grund für dieses merkwürdige Verhalten war gar nicht so einfach zu finden. Jedenfalls kann man feststellen, dass das Problem nicht auftritt, wenn bei jeder SPI-Übertragung der Inhalt des Registers USISR gerettet wird:

```
Function Spi(s As Byte) As Byte
  Retteusicr = Usicr                    'muss gesichert werden
  Usidr = S
  Usisr = &B01000000                    'USIOIF und Zähler löschen
  Usicr = &B00011010                    'USIWM0, USICS1,USICLK auf 1
  Do
    Usicr.usitc = 1                     'Takten durch Setzen von usitc
  Loop Until Usisr.usioif = 1           '16 mal toggeln
  Usicr = Retteusicr
  Spi = Usidr
End Function
```

Dies lässt darauf schließen, dass die LCD-Library von BASCOM dieses Register nicht nur lokal, d. h. für die Laufzeit einer LCD-Routine, sondern auch global einsetzt! Diese unschönen Effekte sollten daher nicht auftreten, wenn die SPI-Kommunikation auf den Einsatz der USI verzichtet, also rein softwaremäßig abgewickelt wird.