XBee for NewBees (DOS-Tool zum Testen von XBee's im API-Modus)
(Überarbeitung vom 28.12.10)

Hallo allerseits,

hier noch einmal zurück zum Ausgangspunkt/Betreff dieses Threads.

Das dort vorgestellte Tool war mir eine große Hilfe bei der 'Erforschung' der XBee's im API-Modus, wenngleich es nicht sonderlich komfortabel war.
Seine Aufgabe ist, lokale bzw. remote XBee's mit AT-Commands zu konfigurieren und auszulesen bzw. Zeichenfolgen an ein XBee zu senden, die von einem angeschlossenen Controller weiterverarbeitet werden.

Zwischenzeitlich habe ich die Bedienung und die Funktionalität noch einmal überarbeitet.

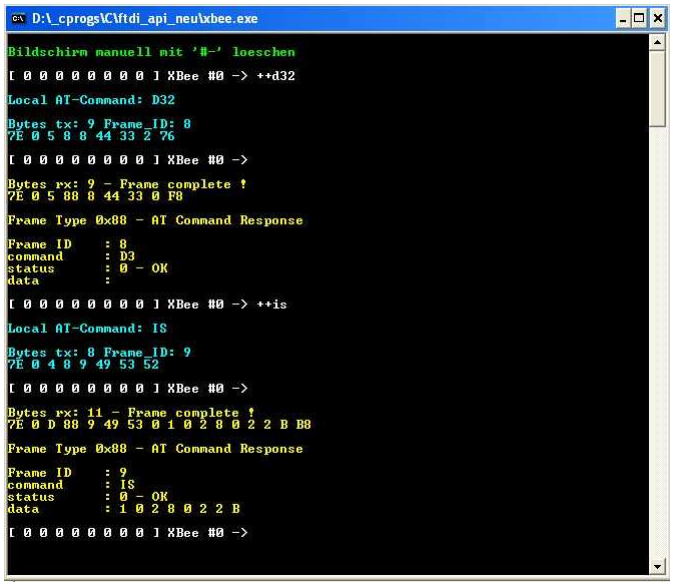
Die wesentlichen Veränderungen:

Das Programm fragt nun ständig die serielle Schnittstelle und die Tastatur ab (kbhit() sei es gedankt!), eingehende Daten werden ohne weiteres Zutun auf den Schirm manipuliert.

Erst dann, wenn eine Tastatureingabe erkannt wird, verzweigt das Programm in scanf() - und solange die Eingabe nicht abgeschlossen ist, werden keine eingehenden Daten erkannt !

Auch die Eingabe ist flexibler geworden:
Zahlen können dezimal, hexadezimal, oktal oder binäre eingegeben werden.
Das Ergebnis wird entsprechend des Wertes in 1, 2 oder 4 Byte umgewandelt.
Durch ein Komma getrennt, können mehrere Zahlen hintereinander eingegeben werden.
Auch das Mischen mit Textfolgen ist möglich.

Die Bildschirmausgabe ist zur Unterscheidung von Eingaben, gesendeten und empfangenen Daten farbig geworden:



Wenn das 'conio-2.0-1mol.DevPak' in Dev-C installiert ist, kann die Konstante IS_CONIO2_LIB in 'global.h' aktiviert werden.

Wenn dann beim Kompilieren die libconio.a gefunden wird, wird bei der Ausgabe unterschieden:

- Prompt + Tastatureingabe: weiß
- gesendete Daten: cyan,
- empfangene Daten: gelb,
- Meldungen des Programms: grün.

Ist die Bibliothek nicht installiert, dann bleibt die Ausgabe monochrom - die Konstante IS_CONIO2_LIB muss auskommentiert werden.

Die Programmänderungen und Bedienung habe ich in der Readme.pdf dokumentiert bzw. korrigiert.

Anmerkung:
Das Programm benötigt für den Zugriff auf die serielle Schnittstelle unverändert die ftd2xx.lib und einen FTDI USB-Adapter und ist bei mir unter XP SP3 getestet.

Viel Spaß beim Funken,

Michael S.

XBee for NewBees (DOS-Tool zum Testen von XBee's im API-Modus)

Benötigt werden: FTDI-USB-Seriell-Adapter / ftd2xx.dll
ggf: C-Compiler: z.B. Dev-C++ *(optional mit conio-2.0-1mol.DevPak)*

Während die Arbeit mit den XBee's im AT-Modus fast ohne eigenes Zutun funktioniert, ist die Latte für den Einstieg in den API-Modus höher gehängt.

Um den Anfang und das Debuggen nicht allzu frustrierend werden zu lassen, habe ich mich entschlossen, zunächst ohne AVR ausschließlich am PC unter Dev-C++ zu arbeiten.
Damit habe ich die grundlegenden Routinen erstellt und getestet - und nebenbei ist ein kleines Tool entstanden, mit dem man XBee's im API-Modus konfigurieren und auslesen kann.

Das Programm läuft in einem schlichten DOS-Fenster. Die Eingabe erfolgt über die DOS-Befehlszeile.

Der PC ist über einen FTDI USB-Seriell-Adapter mit einer der XBee's verbunden.
Die Software greift über die ftd2xx.dll von FTDI auf den Adapter zu
(die Datei "cdm 2.06.00 whql certified\cdm 2.06.00 whql certified\i386.zip" muss ggf. von der Herstellersite ftdichip.com geladen werden).

Getestet habe ich in einem kleinen Netzwerk mit 1 Koordinator und 2 Routern und 1 Enddevice (Modem XB24-ZB, Firmware 2164, 2364, 2964), die zugehörige Dokumentation 90000976_F.pdf ist erschöpfende Pflichtlektüre.

Nach dem Laden der jeweiligen Firmware auf die XBee's (in ihrer Default-Einstellung) wurden alle weiteren Schritte aus der Befehlszeile des Tools ausgeführt.

Folgende Szenarien habe ich durchgespielt und ihre Tücken erforscht:

- beliebige Konfiguration von lokalen und der remote-Devices mittels AT-Kommandos,
  z.B. Setzen/Auslesen von Systemvariablen und Portpins, Auslesen von ADC-Werten ...
- ein Enddevice läuft im sleep-Modus und liefert beim Aufwachen IO-Daten an ein anderen Router.

Mit den Erfahrungen aus der Arbeit mit diesem Tool sollte der Einsatz eines Microcontrollers zur Steuerung des Netzes deutlich leichter fallen.

Dazu können Teile des C-Programmcodes in angepasster Form für den AVR genutzt werden.

Und genau das war das eigentliche Ziel dieser Übung.

Viel Spaß beim Funken,

Michael S.

Für die Arbeit im API-Modus ist weder das X-CTU noch ein Terminal-Programm eine echte Hilfestellung, da die Nachrichten in eine fest definierte Form (Frames) gebracht werden und mit Längenangaben und einer Prüfsumme versehen werden müssen.
Manuell ist die Zusammenstellung dieser Frames extrem umständlich.

Um aber auch im API-Modus mit wenig Aufwand ein Netzwerk konfigurieren zu können, habe ich mir ein kleines Tool zusammengehäkelt, das sozusagen als 'Frame-Builder' Daten in die notwendigen API-Frame-Formate zwingt.
Bzw. eingehende Daten in ihrer Frame-Struktur wiedergibt.

Achtung: die Oberfläche des Programms hat den Charme eines Shell-Scripts.
Die Ausgabe erfolgt in einem einfachen DOS-Fenster.

Nach der Eingabe eines Commands wird das Fenster mit den Meldungen einfach gelöscht und mit den neuen Daten neu aufgebaut *(kann durch '#A' geändert werden).*

Bei AT-Commands im API-Modus ist keine Guard Time wie im AT-Modus zu beachten.
Auch darf kein '+++' gesendet werden.

Das 'AT' des AT-Commands ist bei der Verwendung des Programmes zu ersetzen durch '++' (für ein lokales AT-Command) oder '##' (für ein Remote AT-Command) !

Die Eingabeoptionen am Prompt sind :

XX + AT-Command gemäß der Xbee Command Reference Table

wobei XX anstelle von 'AT' für '++' oder '##' steht.
'++' sendet ein lokales AT-Command (Frame 0x08) an das am USB-Adapter angeschlossene Radio.
'##' sendet ein remote AT-Command (Frame 0x17) über Funk an ein entferntes Radio.
Alle anderen Eingaben werden als ZIGBEE TRANSMIT REQUEST (Frame 0x10) versandt.

wichtige Änderung:
In der Eingabe ist zwischen Zeichenfolgen und Zahlen zu unterscheiden:
Zahlen müssen aus Ziffern bestehen + ggf. dem vorangestellten
- 0x für ein hexadezimales,
- 0b für ein binäres oder
- O für ein oktales Eingabeformat.

Beispiel:	*++dh0x0013a200*	*setzt die Destination-High-Adresse auf dem lokalen device*
	##dh0x0013a200	*setzt die Destination-High-Adresse auf dem remote device*

~~Die Parameter hinter dem AT-Command werden hexadezimal ohne '0x' erwartet und vor dem Senden in binäre Form umgwandelt - solange die Zeichenfolge aus gültigen hexadezimalen Ziffern besteht.~~
~~Ausnahme ist nur ATNI. Hier werden alle folgenden Zeichen als String gesendet.~~
~~+ nach eingehenden Daten suchen~~
~~(Das ist eine Krücke, da im Eingabemodus im Hintergrund empfangene Bytes nicht erkannt werden)~~
~~- löscht das DOS-Fenster~~
~~# beenden (alternativ Ctrl-C)~~

#x ein Target (Zielstation) für die nachfolgenden Frames wählen (x = Nr. des Targets)
#- Löscht den Bildschirm
#A Umschalten zwischen manuellem Löschen (mit #-) oder automatischem Löschen des Bildschirms jeweils nach einer Eingabe
#F Schaltet um zwischen fester Frame_ID 0 und hochzählender Frame_ID 1...255
(bei der Frame_ID 0 liefert die Gegenstelle keine Quittung !)
#V Schalten die Ausgabe der gesendeten und empfangen Bytes in drei unterschiedliche Modes:
AUS - es wird nur der Frametype der gesendeten Daten + Kommando sowie die Framemaske der empfangenen Daten ausgegeben.
kurz - zusätzlich wird die Anzahl der gesendeten/empfangenen Bytes ausgegeben,
lang - zusätzlich werden alle gesendeten/empfangenen Bytes ausgegeben
#? gibt eine knappe Hilfe aus
CTRL-C Beendet das Programm

Alle anderen Eingaben werden als "transparente" Daten via Frame 0x10 versandt.
Der Sinn ist, dass diese Daten vom Empfänger an einen angeschlossenen Controller weitergereicht werden. Daher war es auch erforderlich, das Versenden von Binärdaten zu ermöglichen.

Die Tastatureingabe wird grundsätzlich wie folgt interpretiert:

Wenn als die beiden ersten Zeichen '++' oder '##' erkannt wird und die Zeichenfolge länger als 3 Zeichen ist, dann wird von der Eingabe eines Kommandos ausgegangen, der nachfolgende Reststring ab Position 4 wird dann weiter untersucht. Ist die genannte Bedingung nicht erfüllt, dann wird die gesamte Eingabe ungekürzt interpretiert und als Frame 0x10 versendet.

Der (ggf. verbleibende) Eingabestring wird durch ',' in Teilstrings unterteilt, die jeweils für sich analysiert werden.

- Ist das erste Zeichen ein '|', dann wird der Teilstring als Text übernommen (ohne das '|').

- Ist das letzte Zeichen ein 's/S' oder 'l/L' oder 'w/W', dann wird - falls der Reststring als eine gültige Zahl interpretiert werden kann, diese Zahl als 8, 16 oder 32 Bit-Wert übernommen, d.h. kleinere Werte werden erweitert, größere Werte auf die geforderte Größe beschnitten.

- Nun wird versucht, den Teilstring in eine gültige Zahl umzuformen.

- Gelingt die Umwandlung, dann wird die Anzahl der zurückgegebenen Bytes entsprechend des Wertes der Zahl gewählt - es sei denn, der Zeichenfolge war ein S/L/W hintangestellt - dann ist diese Vorgabe für die Länge maßgebend.

- Gelingt die Umwandlung in eine Zahl nicht (weil als Ziffer unzulässige Zeichen enthalten sind), dann wird der String als reiner Text übernommen.

Für Sonderfälle gibt es folgende Regelung:

- soll ein ',' nicht als Trennelement (sondern als ',' erhalten bleiben), dann ist ein '|' voranzustellen.

- soll der '|' vor dem Komma auch erhalten bleiben, dann sind '||' voranzustellen.

++dh0x0013a200	*setzt die Destination-High-Adresse auf dem lokalen device*
++dh0x0013,0xa200	*alternative Adress-Eingabe mit 2x 16-Bit Werten*
++dh0x00,0x12,0xa2,0x00	*alternative Adress-Eingabe mit 4x 8-Bit Werten*
abc,def	*liefert: 'abcdef'*
abc|,def	*liefert: 'abc,def'*
abc12 bzw. abc|,12	*liefert: 'abc12'*
abc,12	*liefert: 'abc'0x0C*
256s	*liefert: 0x00 (weil 1 Byte nicht ausreicht)*
256l	*liefert: 0x0100*
256w	*liefert: 0x00000100*
|256w	*liefert: '256w'*
-128	*liefert: 0x80*
-129	*liefert: 0xFF7F*

~~Nach jeder Eingabe (außer '+') wird der Bildschirm gelöscht.~~
~~In der ersten Zeile wird das erkannte Kommando ausgegeben, in der Folgezeile das versandte Frame in hexadezimaler Darstellung.~~
~~Es folgt dann die Anzahl der zurückgelieferten Bytes sowie die Daten in hexadezimaler Darstellung.~~
~~Vom Empfänger wird normalerweise eine Quittung zurückgegeben.~~
~~Das Programm wartet ca. 6 Sekunden auf eine Antwort.~~
~~Wird eine Antwort erkannt, dann erfolgt die Anzeige des/der Frames und es wird wieder in den Eingabemodus gewechselt.~~
~~Bei manchen Kommandos (etwa ATND oder bei Broadcasts) kann es einige Zeit dauern, bis die Rückmeldungen von allen Stationen eingehen.~~
~~Während das Programm im Eingabemodus läuft, kann nicht erkannt werden, dass weitere Daten empfangen wurden. Das hat zu der folgenden Hilfskonstruktion geführt:~~
~~Wird ein '+' eingegeben, dann prüft das Programm auf zwischenzeitlich empfangene Daten.~~

Die empfangenen Frames werden - soweit möglich - entsprechend ihrer Framestruktur formatiert ausgegeben.

~~Zur Kontrolle werden alle versandten bzw. empfangenen Frames byteweise auf dem Bildschirm ausgegeben.~~

Zur Kontrolle können alle versendeten und empfangenen Daten auf dem Bildschirm ausgegeben werden.
Die Ausgabe kann in drei verschiedenen Formen erfolgen (#A):

aus *Ausgabe des Frametyp und der Parameter der versandten Daten,*
Ausgabe empfangener Daten formatiert in der Framemaske

kurz *zusätzlich Ausgabe der Anzahl der versendeten und empfangenen Bytes*

lang *zusätzlich Ausgabe aller versendeter und empfangener Bytes*

Die versendeten Frames werden (bei einigen Frametyps) zur Identifikation mit einer Frame_ID versehen.
Das Programm vergibt eine fortlaufende ID zwischen 1 .. 255.
Mit (#F) kann umgeschaltet werden auf die feste Frame_ID 0.
Aber Achtung: nun gibt die Gegenstelle keine Quittung zurück !

Die Target-Adressen (also die Seriennummern aller im Netzwerk angemeldeten Devices) müssen in der Datei 'xbee_adr.cfg' bekannt gemacht werden, damit Verbindung zu ihnen aufgenommen werden kann.

Die Default-Adresse für Coordinator und Broadcast sind bereits im Programmcode definiert,
der Coordinator wird als Target#0, ein Broadcast als Target#1 adressiert.

Es ist zu beachten, dass die Array-Größe für die Adressen hinreichend groß ist, um die Anzahl der vorhandenen Stationen aufnehmen zu können (wobei noch 2 Adressen für Coordinator und Broadcast zu addieren sind).

In der global.h ist zunächst Platz für 8 XBees reserviert (#define DEVICES 10).
In der 'xbee_adr.cfg' dürfen weniger Adressen vorhanden sein, ihre genaue Anzahl ist in der ersten Zeile festzulegen (siehe unten).

Die Adressbytes werden immer im Format 'FF' mit genau zwei hexadezimalen Zeichen eingetragen !

Was sich eigentlich von selbst versteht: Die Einträge in der Datei mitgelieferten xbee_ad.cfg müssen durch die Adressen der eingesetzten XBee's ersetzt werden !!

Kleine Änderung in der xbee_adr.cfg:

In der 1. Zeile der 'xbee_adr.cfg' wird die Anzahl der benutzten XBee's mit einem abschließenden '#' eingetragen. In der global.h ist Platz für 8 XBee's voreingestellt.
Sollen mehr als 8 XBee's adressiert werden, muss die Konstante 'DEVICES' vergrößert werden.

Die Adressen der Stationen werden beim Programmstart aus der Datei 'xbee_adr.cfg' eingelesen.

Durch die Eingabe von # + Targetnummer kann eine der Stationen als Ziel für alle nachfolgenden REMOTE Übertragungen ausgewählt werden.

Vorgabe beim Programmstart ist #0 == Coordinator.

Zur Kontrolle der XBee's ist es sinnvoll, sie während der Testphase jeweils mit 3 LED's zu bestücken.
Die zeigen dann (gegen gnd geschaltet) an:

Pin.6 XBee hat Daten gerade empfangen, die Led leuchtet für 4 Sekunden (->ATRP),
genaugenommen sollte die Helligkeit der Led sogar abhängig sein von der Empfangsstärke.
Pin.13 XBee im Sleep-Modus (Led off) oder aktiv (Led on),
Pin.15 XBee hat (irgendwann einmal) ein Netz gefunden (Led blinkend) oder nicht (Led on).

Eine blinkende Led an Pin.15 sagt nicht zwingend aus, dass Coordinator und Router assoziiert sind.
So blinkt die Led am Router, auch wenn der Coordinator noch gar nicht eingeschaltet ist.
Die Led zeigt nur an, dass der Router/Enddevice irgendwann einmal assoziiert war.

Um ggf. eine neue Assoziation zu erzwingen, kann man den Router mit 'ATNR0' reseten.
Dann baut er eine neue Verbindung zum Coordinator auf.

Ein Enddevice hat in der default-Einstellung zunächst irritierende Eigenschaften:
die LED's blinken mehrfach je Sekunde.
Das kommt daher, dass sein SleepMode auf 'cyclic sleep enable' steht und die SleepPeriod auf 1/4 Sekunde voreingestellt ist.

Zur Veranschaulichung der Bedienung einige Beispiele:

++D13 *Schaltet am lokalen XBee den Pin D1 als digitalen Eingang*
++D22 *Schaltet am lokalen XBee den Pin D2 als analogen Eingang*
++IS *Liest die Eingangs-Pin-Konfiguration und den Pin-Status des lokalen XBee*
ein Frame 0x88 mit den Daten wird empfangen
im Feld data steht in codierter Form die Pinconfiguration und der Pinstatus/ADC-Messwert

#2 *Schaltet auf XBee 2 (das sei ein Router, nicht der Coordinator und kein lokales Device!)*
##D13 *Schaltet am remote XBee den Pin D1 als digitalen Eingang*
##D22 *Schaltet am remote XBee den Pin D2 als analogen Eingang*
##IS *Liest die Eingangs-Pin-Konfiguration und den Pin-Status des remote XBee*
ein Frame 0x97 mit den Daten wird empfangen
im Feld data steht in codierter Form die Pinconfiguration und der Pinstatus/ADC-Messwert

##IR1000 *das remote XBee sendet alle 1000ms seinen Pin-Status*
(vermutlich ist nur zu erkennen, das die Gegenstelle sendet,
aber die Sendung geht an DH/DL, per default ist das 0/0 = Coordinator)
##IR0 *daher zunächst einimal das automatische Senden abschalten*
++SH *die eigene SH auslesen und manuell in das Kommando ATDH übertragen*
'##DHxxxx *und an die Gegenstelle senden*
++SL *die eigene SL auslesen und manuell in das Kommando ATDL übertragen*
##DLxxxx *und an die Gegenstelle senden*
##IR1000 *Nun ein neuer Versuch, das Senden wieder aktivieren*
Nun treffen im Sekundentakt die Frames 0x92 mit den Daten ein

##IR0 *das automatische Senden wieder abschalten*
##IC0b10 *(oder ##IC2) aktiviert die IO Digital Change Detection an Pin D1*
wenn der Pinstatus von Pin D1 geändert wird (etwa durch einen Schalter), dann wird ein Frame 0x92 an die Adresse geschickt, die in DH/DL angegeben ist

#1 *Schaltet auf Broadcast (= Sendung an alle)*
##CB1 *an allen XBee's sollten die Leds kurzzeitig blinken*

Viel Spaß beim Funken,

Michael S.