dresden elektronik

# Praktikumsarbeit

# AT Kommandos

**Tobias Ehrlich**
**Dokumentenversion 0.1**
**2016-11-08**

# Inhaltsverzeichnis

# 1 Einleitung

In Rahmen des Diplom Studiengangs ist ein Semester für das Praktikum vorgesehen. Dieses hat den Sinn und Zweck das in der Universität erlernte theoretische Wissen in der Praxis an ein komplexes Projekt anzuwenden und somit den Erfahrungswert zu Steigern und zu festigen.

Mein persönliches Ziel war es dabei ein Projekt in der C Programmierung zu finden, meine Kenntnisse in der Programmiersprache zu verbessern, neue Erkenntnisse zu ziehen und ungeklärte Lücken zu schließen.

Das Praktikum bei dresden-elektronik bietet mir nicht nur die Möglichkeit in C zu Programmieren sondern auch das Wissen aus dem Modul Rechnernetze anzuwenden und die Umsetzung dieser Kenntnisse auf die Mikrocontroller-Technologie zu erlernen.
Ein wesentlicher Bestandteil der Mikrocontroller ist die Steuerung der Module durch einfache Befehle, den AT Kommandos.

Nach der Schließung, bzw. den Aufkauf von Atmel durch die Firma Microchip wurde die einheitliche Weiterentwicklung an diesen Befehlssatz eingestellt und die einzelnen Hersteller erweitern, modifizieren oder entwickeln neue Kommandos und implementieren diese.
Einer dieser Hersteller ist Digi. Diese Firma stellt die XBee RF-Funkmodule her, die Weltweit von Hobbybastlern bis hin zu Großfirmen und Forschungsstationen wie zum Beispiel die NASA eingesetzt werden.

Meine Aufgabe ist es für die Firma dresden-elektronik die alten AT Kommandos zu sichten, zu überprüfen ob diese kompatibel zur Referenzfirma Digi sind und einen neuen Prototyp zu entwickeln, der es ermöglicht mit den Hausinternen Produkt ConBee die XBee Module anzusteuern.

# 2 Inhalt

Die AT Kommandos sollen die auf AVR und potentiell ARM R21 basierenden Funkprodukte wie ConBee, RaspBee und ausgewählte Funkmodule für die Entwickler interessant machen.

Die Zielgruppe reicht von der DIY Makerszene bis zum Industriellen Kunden der sich mit den DE Funkmodulen, mit möglichst geringen Einstiegshürden beschäftigen will.

Da in der DIY Szene die XBee Module und deren AT Kommandos sehr weit verbreitet sind, wird eine 100% Kompatibilität angestrebt. Der bei XBee vorhandene API Mode mit binärem Protokoll soll als Erweiterung vorgesehen werden die später ergänzt wird.

# 3 Aufgaben

Um spätere Kompatibilitätsprobleme zu vermeiden wird die Aufgabenstellung allgemein gestellt, die Auswertung basiert zunächst auf ein XBee S1 XB24 Modul mit den meist verwandtesten unterschiedlichen Funktionssets und der Firmware Version.

XBEE 802.15.4, 10ec
XBEE 802.15.4 Analog IO-Adapter, 16e6
XBEE 802.15.4 Digital IO-Adapter, 17e6
XBEE 802.15.4 AIO-Adapter, 16ec

XBEE 802.15.4 DIO-Adapter, 17ec

## 3.1 Die XBee Varianten

Die XBee Module kommen in verschiedenen Varianten. Die einfachste S1 Form stellt 802.15.4 Kommunikation für Punkt-zu-Punkt und Sternnetzwerke zur Verfügung. Die S2 Module hingegen ermöglichen ZigBee Meshnetzwerk Funktionalität. Die Schnittstelle für AT Kommandos und API Mode sind bei S2 Modulen entsprechend umfangreicher.

Für die erste funktionelle Implementierung ist daher die S1 Variante sinnvoll. Die Erweiterung der Implementation um die S2 ZigBee/Meshnetzwerk Funktionen muss jedoch möglich sein.

## 3.2 Statistik der verbreitetsten AT Kommandos

Anhand von XBee Tutorials und Videos soll eine Statistik erarbeitet werden, welche die am meisten verwendeten AT Kommandos ermittelt.

Die Statistik soll tabellarisch mit Quellenangaben angelegt werden und 15-20 Datensätze umfassen.

## 3.3 X-CTU Modulkonfiguration Analyse

Zur Konfiguration und Testzwecken soll das XBee Tool X-CTU verwendet werden. Damit eine eigene XBee AT Kommando Implementation, z.B. auf einem ConBee, mit diesem Tool funktionieren muss wahrscheinlich ein XBee Modul simuliert werden (ID, MAC Adresse, Version, . . . ?).

Um zu ermitteln welche Befehle zwischen X-CTU und einem S1 XBee ausgetauscht werden soll ein USB Serialanalyzer Tool verwendet und die Ergebnisse umfangreich dokumentiert werden. Die Kommandos die zusätzlich zu denen in der Statistik verwendet werden, sollen in diese mit dem Vermerk „X-CTU“ aufgenommen werden.

Die Statistik, Analyse und Dokumentation bilden die Basis für die Implementation des Prototypen.

## 3.4 XBee Robustheit Analyse

Für die angestrebte Kompatibilität sind neben der korrekten Implementation der definierten Kommandos auch die Fehlerfälle und deren Verhalten auf dem XBee Modul zu bewerten.

Für alle ermittelten Kommandos in der Statistik sollen daher Fehlerfälle definiert, getestet und dokumentiert werden.

Hierzu zählen beispielsweise Falscheingaben, unvollständige Eingaben, Zeitliches Verhalten bei Falscheingaben. Auslassen von Parametern. Timeouts usw.

Die Analyse muss für die spätere Implementation betrachtet werden, so dass man entscheiden kann ob man das Verhalten des Moduls für einen konkreten Fehlerfall imitiert oder ob das Verhalten in einem konkreten Fehlerfall mit einer eigenen Lösung verbessert werden kann, um z.B. das blockieren des Moduls zu verhindern.

## 3.5 Plattformunabhängige Definition einer abstrakten AT Kommando C-API

Der besondere Anspruch der AT Kommandos ist das diese auf beliebigen Plattformen eingesetzt werden können. Es sollen daher mehrere C Module entstehen die das AT Kommandohandling unabhängig vom verwendeten Microcontroller umsetzen.

Die API muss daher Serielle Kommunikation, Timer und Taskhandler derart abstrahieren das diese mit geringem Aufwand auf die Zielplattform portiert werden können. Als zukünftig mögliche unterliegende Software Stacks können Atmel BitCloud, Atmel MAC Stack, Atmel Lightweight Mesh, Uraculi, TI- Z-Stack und ähnliche angenommen werden.

Alle Kommandos inklusive die zur Konfiguration müssen asynchron verarbeitet werden.

## 3.6 Planen des Prototypen

Der Prototyp soll auf dem ConBee mit AVR ATmega256RFR2 lauffähig sein. Die Implementation wäre somit ebenfalls auf dem RaspBee und deRFmega256* Modulen einsetzbar, was für den Prototypen jedoch nicht der Fokus ist.

Der zu verwendende Software Stack ist noch zu ermitteln. Denkbar sind Atmel BitCloud 3.3.0 oder Uraculi. Letzterer hat den Vorteil, dass er der Open Source Stack weiterentwickelt wird und potentiell Support für den ARM R21 erhält.

# 4 Die XBee Varianten

Digi bietet die XBee Module nicht nur in varianten an welche die Marke im Namen haben sondern auch in angepassten Versionen für einen jeweiligen speziellen Aufgabenbereich. Im Folgenden werden nur Module aufgelistet, die entsprechend mit den Namen XBee bezeichnet wurden. Alle Module erscheinen sowohl in einer *Standard*, als auch in einer *PRO* Version, die hier aufgelisteten Spezifikationen stellen die Maximalleistungen der Module dar und sind daher nicht eindeutig einer Verkaufsversion zu zuordnen.

Es werden auch nicht alle Spezifikationen aufgelistet, denn diese Übersicht soll nur einen Überblick Schaffen welche verschieden Versionen es gibt. Sind weitere Informationen erforderlich können diese der Datenspezifikation auf der Herstellerseite oder der Tabelle des englischsprachigen Wikipedia Eintrages[1] entnommen werden.

## 4.1 Die Module

### 4.1.1 Module mit DigiMesh Protokoll Unterstützung

**XBee SX** ist die Energiesparvariante bei maximaler Effizienz. In ihr arbeitet ein 1 Watt 900MHz XBee PRO Modul. Aufgrund SMT (surface mount) Technologie sind sie zudem platzsparend.

| | |
|---|---|
| Frequenz: | 900MHZ |
| Modell: | SMT |
| Reichweite: | bis ca. 105km |
| Sendeleistung: | 55mA (20mW) @ 13dBm |
| | 900mA (1W) @ 30dBm |
| Verbrauch im Sleep Mode: | 2.5µA |
| Verschlüsselung: | 256-bit |

**XBee 900HP** besitzen im Vergleich zur 802.15.4 und ZigBee Version nur einen vereinfachten AT Befehlssatz, dafür aber auch eine erweiterten API.

| | |
|---|---|
| Frequenz: | 900MHz |
| Modell: | Through-hole |
| Reichweite: | bis ca. 14km |
| Sendeleistung: | 229mA (250 mW) @ 24dBm |
| Verbrauch im Sleep Mode: | 2.5µA |
| Verschlüsselung: | 128-bit |

**XBee XSC** besitzen dieselben Spezifikationen wie die 900HP Variante aber mit einer höheren Reichweite, die bis zu 45 km beträgt. Es fehlt jedoch die Verschlüsselung und benötigen keine 9XStream[2] Sender.

[1] Wikipediaeintrag zu den XBee Modulen: https://en.wikipedia.org/wiki/XBee
[2] Die XStream Serie ist eine veraltete Vorgängerversion der XBee Module

**XBee 868LP/865LP** sind die *Low Power* Module und unterscheiden sich nur am Energieverbrauch und an der Sendeleistung zu den 900HP Modulen

| | |
|---|---|
| Frequenz: | 868MHz/865MHz |
| Modell: | SMT |
| Reichweite: | bis ca. 8km |
| Sendeleistung: | 62mA (25 mW) @ 14 dBm |
| Verbrauch im Sleep Mode: | 1.7µA |
| Verschlüsselung: | 128-bit |

**DigiMesh 2.4** verwendet XBee und XBee-PRO 802.15.4 Module und kann entsprechend mit beiden Firmware Versionen verwendet werden.

| | |
|---|---|
| Frequenz: | 2.4GHz |
| Modell: | Through-hole |
| Reichweite: | bis ca. 1.6km |
| Sendeleistung: | 340mA (63mW) @ 18dBm |
| Verbrauch im Sleep Mode: | < 50µA |
| Verschlüsselung: | 128-bit |

### 4.1.2 Module mit ZigBee Protokollunterstützung

Die XBee ZigBee Module sind in zwei Varianten erhältlich, einmal in der älteren *ZNet 2.5* und in der neueren *ZigBee 2007* Version, beide sind kompatibel zu anderen ZigBee Produkten anderer Hersteller. Zudem sollen die neuen Modelle das Thread, IPv6 (6LoWPAN) Protokoll unterstützen.

| | |
|---|---|
| Frequenz: | 2.4GHz |
| Modell: | Through-hole, SMT |
| Reichweite: | bis ca. 3.2km |
| Sendeleistung: | 120mA (63mW) @ 18dBm |
| Verbrauch im Sleep Mode: | < 1µA |
| Verschlüsselung: | 128-bit |

### 4.1.3 Module mit 802.15.4 Protokollunterstützung

Diese Modelle sind seit der Serie S2C die Allrounder und können neben den Standard 802.15.4 Protokoll auch zu ZigBee und DigiMesh Versionen portiert werden. Die Ältere S1 Serie unterstützt hingegen nur die DigiMesh Firmware.

| | **S2C Serie** | **S1 Serie** |
|---|---|---|
| Frequenz: | 2.4GHz | 2.4GHz |
| Modell: | Through-hole | Through-hole |
| Reichweite: | bis ca. 3.2km | 1.6km |
| Sendeleistung: | 120mA (63mW) @ 18dBm | 215mA (60mW) @ 18dBm |
| Verbrauch im Sleep Mode: | < 1 µA | <10µA |
| Verschlüsselung: | 128-bit | 128-bit |

### 4.1.4 Wi-Fi 802.11.b/g/n

Wi-Fi Modelle der Firma Digi sind dazu Optimiert um sie über den DigiCloud Service zu konfigurieren und die M2M Kommunikation zu beschleunigen. Zudem wird dem Nutzer ermöglicht dank SoftAP[3] Technologie die Module mit einen Smartphone zu verbinden, Veränderungen vorzunehmen oder Daten zu empfangen. Digi bietet zudem eine Open-Source Webapplikation mit verschieden Widgets an.

Fügt man einen XBee Socket hinzu, können sich die die Module auch mit anderen XBee Varianten verbinden.

| | |
|---|---|
| Frequenz: | 2.4GHz |
| Modell: | Through-hole, SMT |
| Reichweite: | bis ca. 3.2km |
| Sendeleistung: | 309mA @ 16dBm |
| Verbrauch im Sleep Mode: | < 6µA |
| Verschlüsselung: | WPA-PSK, WPA2-PSK und WEP |

## 4.2 Die Protokolle

**DigiMesh** arbeitet nur mit einen Kontentyp im Netzwerk, das heißt alle Knotenpunkte können Daten zu allen anderen Knoten über unterschiedlichen Routen senden. Dies bringt den Vorteil, dass alle Geräte so konfiguriert werden können, dass sie Energie sparen und zu unterschiedlichen Zeiten schlafen gelegt werden können. Der Endverbraucher hat dadurch nicht nur Energiesparoptionen sondern auch mehrere Möglichkeiten das Netzwerk zusammenzustellen und zu erweitern.

Die Netzwerk Stabilität wird ebenfalls erhöht, fällt ein Knoten aus heißt das nicht, dass das ganze Netzwerk ausfällt, da die Daten unterschiedliche Routen durch das Netzwerk nehmen können. Dies wird durch Eigendiagnosen, Selbstheilung, enge Netzwerkoperationen und Netzwerkkonfigurationen umgesetzt.

DigiMesh arbeitet auf Basis eines proprietären Protokolls, das erlaubt eine größere Kontrolle der Programmierbarkeit und mehr Raum für Erweiterungen.

„**ZigBee** ist eine Spezifikation, welche ein Framework für drahtlose Funknetzwerke beschreibt. ZigBee baut auf dem IEEE 802.15.4 Standard auf und erweitert dessen Funktionalität insbesondere um die Möglichkeit des Routings und des sicheren Schlüsselaustausches.“ [4]

Das ZigBee Protokoll erweitert den MAC und PHY Layer um eine APL (Application) SEC (Security) und NWK (Network) Layer. Anwendungen werden vom Programmierer in der Applikationsschicht implementiert. Standardmäßig stehen den Entwickler drei Gerätetypen zur Verfügung:

einen *Koordinator*, der niemals in den Energiesparmodus geht und das Netzwerk steuert
*Router* die, die Endgeräte mit dem Koordinator verbinden
*Endgeräte*, die Aufgaben durchführen

[3] Wikipediaeintrag zu SoftAP: https://en.wikipedia.org/wiki/SoftAP
[4] Wikipediaeintrag zu ZigBee: https://de.wikipedia.org/wiki/ZigBee

Mittlerweile bieten verschiedene Hersteller ZigBee Modelle an, die rein auf theoretischer Basis miteinander verknüpft werden können. Des Weiteren wurden Technologien entwickelt, die es zum Beispiel ermöglichen Over-The-Air Firmware Updates, Energie- und Lichtkontrollapplikationen sowie verschiedene Diagnose Programme anzuwenden.

**Thread, IPv6**
Die neuen XBee Module, basierend auf den Chips von Silicon Labs mit der Bezeichnung EM3587, sollen laut Digi das neue Thread 6LoWPAN Netzwerkprotokoll unterstützen.

Das Netzprotokoll von der Thread Group ist ein Low Power Protokoll basierend auf IPv6 (6LoWPAN). Es kombiniert die Vorteile eines Wi-Fi Netzwerkes mit denen einen energiesparenden Netzwerkes und ist für spezielle Aufgaben optimiert.

Als Basis werden die Schichten eines 802.15.4 Netzes genutzt. Thread fügt eine UDP, IP Routing und eine 6LoWPAN Schicht hinzu.

Da damit das Thread Mesh die Netzwerkkommunikation übernimmt können unterschiedliche Applikationsprotokolle unabhängig voneinander, vom Hersteller oder vom Typ miteinander kommunizieren.

Wird ein Netzwerk eröffnet oder ein neuer Knoten hinzugefügt, wird nur ein Smartphone oder ein Computer benötigt. Das Modul sendet den Beitrittswunsch und wie beim Verbinden mit einen Wi-Fi Router, wird ein Passwort abgefragt.

Bietet aber gleichzeitig die Sicherheit, die auch ein normaler Rechner mit IPv6 besitzt. Aufgrund das es auf ein 802.15.4 Protokoll läuft, besitzt man schon dessen Vorteile. In Verbindung mit dem IPv6 bekommt man ebenfalls Zugang zu allen anderen gängigen Sicherheitszertifikaten und Protokollen wie zum Beispiel TLS, DTLS, SLS, SLL, etc.

Im Netzwerk haben alle Knoten denselben Status, bei der Erstellung eines Netzwerkes, wird ein Gerät von allen Routern zum Koordinator ernannt. Wird dieser Knoten wieder entfernt, so ernennen die verblieben Router wieder einen neuen Router. Das bedeutet es gibt kein Single Point Of Failure.

## 4.3 Unterschiedliche Antennen

Die XBee Module unterscheiden sich nicht nur in den Modelltypen sondern auch in den Antennen, es gibt die PCB, RPSMA und U.FL Typen.

**RPSMA** ist abgeleitet von den alten SMA Standard und bietet die Möglichkeit eine Externe Antenne auf zu schrauben.

**U.FL** ist eine Miniatur, nur wenige mm große Hochfrequenzsteckverbindung, auf ihr können je nach Modell kleine Antennen aufgeschraubt werden oder sind als Chip fest auf der Platine aufgelötet. Diese Module sind jedoch sehr empfindlich und nur für Produkte gedacht die nicht viel bewegt werden.

**PCB** ist vereinfacht gesagt eine aufgedruckte Kupferspulenantenne

**RF Pad bzw. RF Wiederstand,**reduziert das Rauschen und trennt dadurch die Nutzsignale vom Rest.

**Wire** ist in Falle des XBee‘s eine kleine 1,5 cm lange Drahtantenne, die auf die Leiterplatte aufgelötet wird.

**Chip** ist eine 7x7 mm große RF Keramik Antenne, die auf die Leiterplatte aufgelötet wird.

Trotz unterschiedlicher Antennen sind alle identisch mit der Leistung. Sie besitzen laut Hersteller weniger als 0.1 dB Verlustrate, das Receiveransprechverhalten liegt bei rund -110 dBm und +30 dBm TX power. [5]

# 5 Statistik der verbreitetsten AT Kommandos

Die Anordnung der Befehle in der Auswertungstabelle[6] entspricht der Kommandotabelle des Digi User Guides, *ZigBee RF Modules*. Diese Tabelle besitzt eine andere Befehlsaufteilung als das X-CTU, zudem sind im Manual nicht alle Befehle gelistet, daher wurde die Auswertungstabelle entsprechend angepasst und erweitert. Zum Nachschlagen der Kommandoreferenzen empfiehlt sich das entsprechende Digi User Guid, XBee / XBee-PRO S1 802.15.4 (Legacy) hinzu zunehmen. Wenn wir die AT Kommandos aus beiden Modulvarianten in einer Menge abbilden, existieren somit 116 AT Kommandos.

**Tabelle 1: Erfassung** listet die Einzelnen Tutorials auf und den Zähler, welches Kommando wie oft genutzt wird. Dabei sind die Befehlsbereiche farbig markiert um sie schneller den einzelnen Abschnitten in der Command Reference Table des User Guids zuzuordnen zu können.

Ebenfalls sind die Einstellungen der vorgefertigten Funktionssets des XBee S1 802.15.4 Moduls unter X-CTU Default Settings (XBee 802.15.4) aufgelistet. Das X-CTU setzt Voreinstellungen für ca. 50 Parameter, während der Nutzer nur noch wenige Feineinstellungen, wie etwa die PANID, die Ziel Adresse in Destination High und Low sowie die 16-bit Elternknotenadresse festlegen muss.

**Tabelle 2: Ergebnisse** beinhaltet die statistischen Summen der einzelnen Befehle (keine Prozentwerte). Am häufigsten wird des AT Kommando ID (setzen der PAN Netzwerk ID) verwendet, in den 19 Tutorials wird es 23-mal eingegeben dicht gefolgt von den Zieladresskommandos DH und DL mit 17 und 16 Zähler.

[5] Technische Antennenspezifikation: http://www.digi.com/resources/documentation/Digidocs/90000991/concepts/c_xbee_digimesh_antenna_options.htm
[6] Auswertungstabelle: Stat-auswerung_xbee-befehle.xlsx (zuletzt aktualisiert am 22.09.2016)

# 6 X-CTU Modulanalyse

## 6.1 Das Auswertungswerkzeug

Zur Auswertung wird das Tool *Free Device Monitoring Studio* verwendet, welches als Freeware herunter geladen werden kann. [7]

Nach dem Start des Programms gibt es ein Popup mit dem Hinweis, dass wenn man alle Funktionen verwenden möchte zahlen soll [Abb. 1]. Für die Auswertung ist dies jedoch nicht von Nöten und das Fenster wird über den Schriftzug *Continue with limited features* geschlossen.

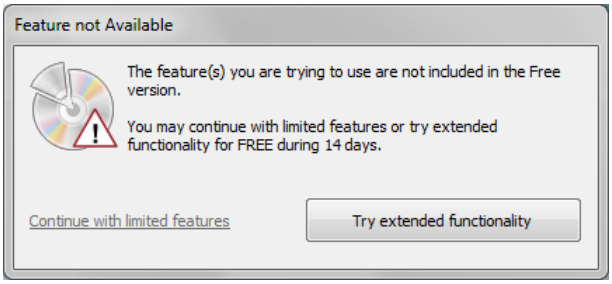


Abb. 1: Kaufaufforderung

Im ersten großen Fenster [Abb. 2] findet man auf der rechten Seite den Device Tab in dem alle Serial Ports, USB Ports und Netzwerk Ports aufgelistet. In der Kostenlosen Variante stehen jedoch nur die USB Ports zur Verfügung. Um eine neue Aufzeichnung zu starten wird auf den gewünschten USB Port doppelt oder rechts und Start Monitoring geklickt. Im folgenden Session Configuration Fenster [Abb. 3] wählen wir die Paket View (Pakete mit schneiden) aus und klicken auf Start.

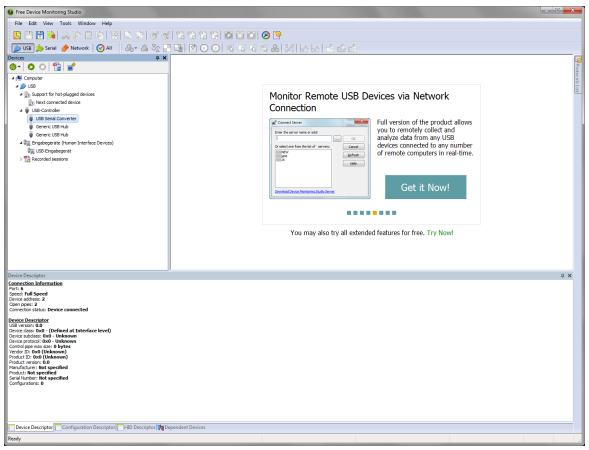


Abb. 2: Hauptfenster des Free Device Monitoring Studios

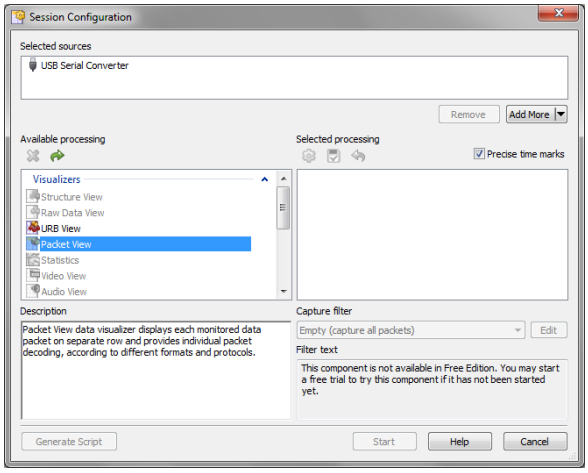


Abb. 3: *Session Configuration* Fenster zum Einstellen der Verfügbaren Ausleseprozesse

Nun kann das X-CTU gestartet und das XBee Modul geladen werden. Das Free Devices Monitoring Studio zeichnet jetzt den Datenaustausch auf. Für diesen Test wurde das NextGen X-CTU

[7] Link zur Software http://freeusbanalyzer.com/

Tools mit der Version 6.3.2 und Build ID: 20160819-3 verwendet.

Im Auswertungsfenster [Abb. 4] finden wir im oberen Tab die Paketmitschnitte, klicken wir auf eines der Pakete, sehen wir im unteren Tab den Inhalt sowohl Hexadezimal als auch ASCII Zeichen, sowie die Anzahl an Bytes, die gesendet wurden. Um die Richtung zu identifizieren sind ausgehende Pakete (DOWN) mit blau gekennzeichnet und eingehende (UP) mit rot.

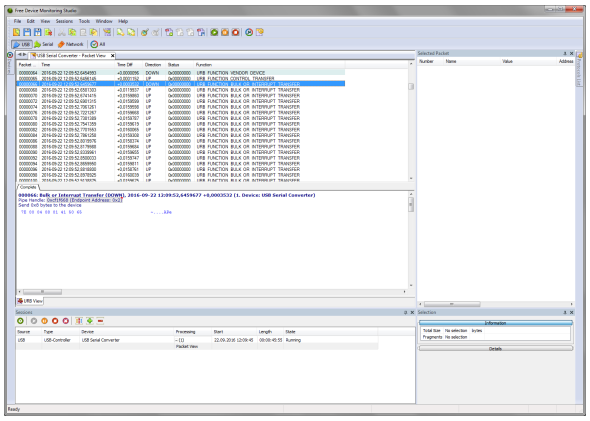

Abb. 4: Das Auswertungsfenster

## 6.2 Die Auswertung

Die XBee Kommandos können direkt über die Konsole als auch durch externe Schnittstellen gesendet werden. Dabei können die Nutzer die AT Kommandos oder die mächtigeren API Frames benutzen. Um jedoch in den *Command Mode* zu gelangen muss der Nutzer erst drei Plus Zeichen an das Modul senden, da standardmäßig der API Empfang eingestellt ist. Die API Frames bestehen aus Hexadezimal Zahlen und beginnen stehst mit dem Wert 0x7E. [8]

Um den ersten Kontakt mit den Modulen aufzubauen sendet das Next Gen X-CTU eine solche API Anfrage, die Legacy Edition tut dies nicht. Weitere Abfragen erfolgen dann mit AT Kommandos.

Das erste Kommando enthält eine AT Anfrage. Mit dieser überprüft das Tool ob sich das Angeschlossene Gerät im API Modus befindet.

Der Befehl `7E 00 04 08 01 41 50 65` löst sich wie folgt auf:

| Anz. Byte | Hex-Wert | Beschreibung |
|---|---|---|
| 1 | 7E | Delimiter ‘~’ |
| 2 | 00 04 | Länge 4 (MSB,LSB) |
| 1 | 08 | API Frame Typ ‘AT Command’ |
| 1 | 01 | ein Fenster das gesendet wird |
| 2 | 41 50 | AT Kommandoinhalt ‘AP’, kann n Byte groß sein (Frame Typ abhängig) |
| 1 | 65 | Checksumme 65(Hex) = 101(Dez) |

[8] Digi User Guid - ZigBee RF Modules, Seite 108.

Zwischen dem Aussenden des API Befehls und den Sende der drei Pluszeichen vergehen konstant in jeden Test 1,7ms. Der USB Stick sendet in dieser Zeit 107 Pakete mit dem Code 11 60 (Device Control 1 und ‘ ` ‘).

Die Zeit zwischen den senden der drei Pluszeichen und dem empfangen des OKs vergeht beträgt 1ms. Es werden dazwischen 64 Pakete versendet.
Nach dieser Bestätigung Startet die Abfrage des Moduls:

| | **Befehlsbeschreibung** | **Befehl** | **Empfangen** |
|---|---|---|---|
| 1 | API Aktiv | ATAP | 0<CR> (aus) |
| 2 | Hardwareversion | ATHV | 1743<CR> |
| 3 | Firmware Version | ATVR | 10EC<CR> |
| 4 | SN-Nr. oberen 64-bit | ATSH | 13A200<CR> |
| 5 | SN-Nr. unteren 64-bit | ATSL | 409A5C81<CR> |
| 6 | Knotenname (String) | ATNI | Space<CR> |
| 7 | Unbekannt | ATR? | ERROR<CR> |
| 8 | Unbekannt (könnte jedoch der Befehl zur Kalibrierung des Temperatursensors sein, XBee ZigBee) [9] | AT%C | 1<CR> |
| 9 | Koordinator Aktiv | ATCE | 0<CR> (End Device) |
| 10 | Sleep Status (XBee-PRO 900 DigiMesh) [10] | ATSS | ERROR<CR> |
| 11 | Baudrate | ATBD | 3<CR> (9600 Bd) |
| 12 | Parität | ATNB | 0<CR> (keine) |
| 13 | Stopbits | ATSB | ERROR<CR> |
| 14 | API Aktiv | ATAP | 0<CR> (aus) |
| 15 | Verlasse Kommandomodus | ATCN | OK<CR> |

Die Werte, die mit den AT Befehlen beim Read gelesen werden richten sich nach der Spezifischen Firmware. Das heißt, liest das Modul die Firmware 10EC beim Verbinden, so ruft er nur die spezifischen Werte für die allgemeinen Einstellungen, bei 16E6 die der Analog IO-Adapter Einstellungen. Welche Befehle ausgelesen werden, können der Tabelle im Abschnitt Erfassung: *X-CTU Default Settings (XBEE 802.15.4)* im Stat-auswerung_xbee-befehle.xlsx Dokument entnommen werden.

Die Zeit zwischen den einzelnen Requests, ist in mehreren Testverläufen unterschiedlich, im Schnitt liegt sie jedoch bei etwa *0,050121ms*.

Um zu überprüfen ob dieser Vorgang auch bei anderen Modultypen gleich bleibt wurde noch ein Weiterer Test mit eine XBee S2 ZigBee Modul getestet. Bei diesem ergibt sich eine etwas andere Tabelle.

[9] Digi Customer Release Notes - 93009373 G1
[10] Digi Customer Release Notes - 93009377 B1

| | Befehlsbeschreibung | Befehl | Empfangen |
|---|---|---|---|
| 1 | API Aktiv | ATAP | ERROR<CR> |
| 2 | Hardwareversion | ATHV | 194A<CR> |
| 3 | Firmware Version | ATVR | 22A7<CR> |
| 4 | Seriennummer oberen 64-bit | ATSH | 13A200<CR> |
| 5 | Seriennummer unteren 64-bit | ATSL | 409029CF<CR> |
| 6 | Knotenname (String) | ATNI | Space<CR> |
| 7 | Unbekannt | ATR? | ERROR<CR> |
| 8 | Unbekannt (könnte jedoch der Befehl zur Kalibrierung des Temperatursensors sein, XBee ZigBee)[11] | AT%C | ERROR<CR> |
| 9 | 16-bit Netzwerkadresse | ATMY | FFFE<CR> |
| 10 | AT Command Mode noch aktiv? | AT | OK<CR> |
| 11 | Netzwerk Erkennungsoption | ATNO | 0<CR> |
| 12 | Netzwerk Erkennungsoption auf 3. Für die 3 ist keine Option gelistet, möglicher Bereichstest | ATNO3 | OK<CR> |
| 13 | Node Discovery Timeout | ATNT | 3C<CR> |
| 14 | | ATNT00000020 | OK<CR> |
| 15 | Änderung Bestätigen | ATAC | OK<CR> |
| 16 | Knoten mit der Adresse SH SL suchen | ATND0013A200409029CF | ERROR<CR> |
| 17 | | ATNT3C | OK<CR> |
| 18 | | ATNO0 | OK<CR> |
| 19 | | ATAC | OK<CR> |
| 20 | Baudrate | ATBD | 3<CR> (9600 Bd) |
| 21 | Parität | ATNB | 0<CR> (keine) |
| 22 | Stopbits | ATSB | 0<CR> |
| 23 | API Aktiv | ATAP | ERROR<CR> |
| 24 | Verlasse Kommandomodus | ATCN | OK<CR> |

Wenn man die Tabellen vergleicht scheint das Programm nach dem Auslesen der entsprechenden Firmware und der Seriennummer die Abfragen anzupassen. In der Tabelle für das ZigBee Modul wird erkenntlich, dass das X-CTU Tool zunächst die Netzwerkeinstellungen überprüft und ggf. zu modifiziert (Zeile 11-18).

Wenn unser Modul mit dem X-CTU kompatibel sein soll muss dem Programm ein XBee Modul vorgetäuscht werden, dafür wird mit hoher Wahrscheinlichkeit die Hardwareversion und Firmware Version, möglicher weise auch die Seriennummer benötigt.

[11] Digi Customer Release Notes - 93009373 G1

# 7 RF Paket Sniffing-Analyse

## 7.1 Vorbereitung

Benötigte Materialien:

- 2 XBee S2 Module (Sensor [Sender], Coordinator [Empfänger])
- 2 XBee S1 Module (Punkt-zu-Punkt Nachrichtenübertragung)
- 1 ConBee USB Stick (als Sniffer)
- die BitCatcher Firmware für den ConBee (ZB-USB_BitCatcher_FW_0x27030500.bin)
- die Software BitCatcher von Luxoft
- den GCFFlasher von dresden-elektronik (zum flashen der Firmware)

**Achtung!** Da der ConBee Stick nicht mehr verkauft und die BitCatcher Software nicht mehr weiterentwickelt wird, gibt es die BitCatcher Firmware nicht mehr zum Download auf der Homepage von dresden elektronik.

Die XBee Module müssen so konfiguriert werden, das sie miteinander kommunizieren, dabei kann man wie im Video Tutorial *XBee Basics - Lesson 2 - Simple Chat Program Between Two XBees* in AT Mode[12] von tunnelsup vorgehen.

Ein S2 ZigBee Modul kann zwar nicht mit einen S1 802.15.4 Modul kommunizieren aber um die Beiden Log-Dateien besser zu vergleichen werden für beide Geräte, so weit wie möglich, dieselben Konfigurationen übernommen.

Gerät A (Koordinator)

| | |
|---|---|
| CH | 0x17 |
| DH | 0x13A200 |
| DL | 0x414640C0 (Modell S1)<br>0x40E97AE3 (Modell S2) |
| ID | 0x4790 |
| MY | 0x101 (nur bei Modell S1 Möglich) |
| CE | 0x1 |
| NI | Coordinator |

Gerät B (Endgerät)

| | |
|---|---|
| CH | 0x17 |
| DH | 0x13A200 |
| DL | 0x409A5C81 (Modell S1)<br>0x409029CF (Modell S2) |
| ID | 0x4790 |
| MY | 0x110 (nur bei Modell S1 Möglich) |
| CE | 0x0 |
| NI | End Device |

Dabei ist zu beachten, dass die Zieladressen (DH/DL), die 16-bit Quelladresse (MY) sowie der Knotenidentifizierung (NI) optional sind und in diesen Test nur zur besseren Visualisierung des Datenflusses dienen.

Als nächstes muss der ConBee geflasht werden. Hierzu kopiert man zur Vereinfachung die Firmware in den GCFFlascher Ordner. Das Programm wird über die Kommandokonsole (CMD) mit den Parameter `-l` gestartet, im Fenster werden alle unterstützten und verfügbaren Geräte aufgelistet. Mit dem den Parametern `-d <Geräteslot> -f ZB-USB_BitCatcher_FW_0x27030500.bin` wird die Firmware auf den USB geflasht.

Im BitCatcher muss zunächst das Gerät hinzugefügt werden, dazu klickt man im *< No Device Selected >* Dropdown Menü auf *-Add/Remove/Edit device* und im folgenden Fenster auf *Add*

[12] Tutorial Lesson 2 von tunnelsub: https://www.youtube.com/watch?v=mPx3TjzvE9U

und *DE Sniffer*. Hier wird der entsprechende COM Port ausgewählt und die Baudrate auf 38400 gestellt, der Rest bleibt Standardmäßig so wie er ist. Um nun die Aufzeichnung zu starten, wählt man den Sniffer aus und klickt daneben auf das lilane Symbol.

Als nächstes Benötigt man die Richtige Kanalnummer, die kann man am schnellsten ermitteln, wenn man via AT Kommando oder per X-CTU den Befehl CH aufruft, im Testfall ist dieser auch oben in der Tabelle gelistet. ZigBee Modelle können von den Werten in der Tabelle abweichen, da sich die Module selbstständig den besten Kanal wählen. Durch das verändern Channel-Bitmask kann man auch diese Module in einen Kanal zwingen.

Um jeweils ein gleiches Bild von den XBee Modulen zu bekommen wird im X-CTU Tool ein vorgefertigtes Chat Fenster erstellt und automatisch aller 60000ms (1min) gesendet.

Als Nachricht wird ein *Hallo XBee<CR>* (48 61 6C 6C 6F 20 58 42 65 65 0D = 11 Byte) gesendet.

## 7.2 Auswertung des XBee S2 Logs

Die S2 Module werden mit dem Funktionssets:
XB24-ZB ZigBee Coordinator AT 20A7,
XB24-ZB ZigBee End Device AT 28A7
vorkonfiguriert und die einzelnen Parameter Gesetzt (vgl. Tabelle).

Gerät A (Koordinator)

| | |
|---|---|
| CH | 0x17 |
| DH | 0x13A200 |
| DL | 0x40E97AE3 |
| ID | 0x4790 |
| MY | automatisch |
| CE | n.a. |
| NI | Coordinator |

Gerät B (Endgerät)

| | |
|---|---|
| CH | 0x17 |
| DH | 0x13A200 |
| DL | 0x409029CF |
| ID | 0x4790 |
| MY | automatisch |
| CE | n.a. |
| NI | End Device |

Um die Channel Mask bei Zigbee Modulen manuell zu setzten muss die Entsprechende Bitmaske in Scan Channels (SC) gesetzt werden, für den Test wird diese auf 0x40 eingestellt.

Die wichtigen von uns gesendeten Daten finden wir in unseren Fall in den APS Frames.

Zum Beispiel Zeile 4836 aus der Log Datei log_xbS2-log.dcf

**MAC Header**

{ 61 88 BA 0E D2 00 00 AF 48 }

| Anz. Byte | Hex-Wert | Bedeutung |
|---|---|---|
| 2 | 88 61 | frameCtl |
| 1 | BA | seqNo |
| 2 | D2 0E | dstPanID |
| 2 | 00 00 | dstShortAddr |
| 2 | 48 AF | srcShortAddr |

**NWK Header**

{ 48 18 00 00 AF 48 1E CD CF 29 90 40 00 A2 13 00 E3 7A E9 40 00 A2 13 00 }

| Anz. Byte | Hex-Wert | Bedeutung |
|---|---|---|
| 2 | 18 48 | frameCtl |
| 2 | 00 00 | 16-bit Zieladresse |
| 2 | 48 AF | 16-bit Quelladresse |
| 1 | 1E | Radius |
| 1 | CD | Sequenz Nummer |
| 8 | 00 13 A2 00 40 90 29 CF | 64-bit Zieladresse |
| 8 | 00 13 A2 00 40 E9 7A E3 | 64-bit Quelladresse |

**APS Header**

{ 40 E8 11 00 05 C1 E8 15 }

| Anz. Byte | Hex-Wert | Bedeutung |
|---|---|---|
| 1 | 40 | frameCtl |
| 1 | E8 | dstEp |
| 2 | 00 11 | clusterID |
| 2 | C1 05 | profileID |
| 1 | E8 | srcEP |
| 1 | 15 | counter |

Digi hat verschiedene Cluster implementiert, daher könne die Geräte auf jeder Cluster ID mit jeder Profil ID oder Endpunkt Daten senden oder empfangen.[13]

**Data**

| { | 48 | 61 | 6C | 6C | 6F | 20 | 58 | 42 | 65 | 65 | 0D | } |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| { | H | a | l | l | o | | X | B | e | e | <CR> | } |

Des Weiteren gibt es die ZDO Fenster.
Zum Beispiel Zeile 20 aus der Log Datei log_xbS2-log.dcf

**MAC Header**

{ 61 88 81 0E D2 00 00 AF 48 }

| Anz. Byte | Hex-Wert | Bedeutung |
|---|---|---|
| 2 | 88 61 | frameCtl |
| 1 | BA | seqNo |
| 2 | D2 0E | dstPanID |
| 2 | 00 00 | dstShortAddr |
| 2 | 48 AF | srcShortAddr |

[13] Digi User Guid - ZigBee RF Modules, Seite 37, Clusters, dritter Absatz

### NWK Header

{ 08 00 FD FF AF 48 1E 9B }

| Anz. Byte | Hex-Wert | Bedeutung |
|---|---|---|
| 2 | 00 08 | frameCtl |
| 2 | FF DF | 16-bit Zieladresse |
| 2 | 48 AF | 16-bit Quelladresse |
| 1 | 1E | Radius |
| 1 | 9B | Sequenz Nummer |

### APS Header

{ 08 00 13 00 00 00 00 00 }

| Anz. Byte | Hex-Wert | Bedeutung |
|---|---|---|
| 1 | 08 | frameCtl |
| 1 | 00 | dstEp |
| 2 | 00 13 | clusterID (DeviceAnnce) |
| 2 | 00 00 | profileID |
| 1 | 00 | srcEP |
| 1 | 00 | counter |

{ 81 AF 48 E3 7A E9 40 00 A2 13 00 80 }

| Anz. Byte | Hex-Wert | Bedeutung |
|---|---|---|
| 1 | 81 | transactionSeqNo |
| 2 | 48 AF | shortAddr |
| 8 | 00 13 A2 00 40 E9 7A E3 | extAddr |
| 1 | 80 | capabilityInfo |

Der NWK Link Status kann zum Beispiel der Zeile 806 aus der Log Datei log_xbS2-log.dcf entnommen werden und wurde nur vom Koordinator aus gesendet.

### MAC Header

{ 41 88 CA 0E D2 FF FF 00 00 }

| Anz. Byte | Hex-Wert | Bedeutung |
|---|---|---|
| 2 | 88 41 | frameCtl |
| 1 | CA | seqNo |
| 2 | D2 0E | dstPanID |
| 2 | FF FF | dstShortAddr |
| 2 | 00 00 | srcShortAddr |

**NWK Header**

{ 09 10 FC FF 00 00 01 97 CF 29 90 40 00 A2 13 00 }

| Anz. Byte | Hex-Wert | Bedeutung |
|---|---|---|
| 2 | 00 09 | frameCtl |
| 2 | FF FC | 16-bit Zieladresse |
| 2 | 00 00 | 16-bit Quelladresse |
| 1 | 01 | Radius |
| 1 | 97 | Sequenz Nummer |
| 8 | 00 13 A2 00 40 90 29 CF | 64-bit Quelladresse |

{ 08 60 }

| Anz. Byte | Hex-Wert | Bedeutung |
|---|---|---|
| 1 | 08 | nwkCommandType |
| 1 | 60 | options |

## 7.3 Auswertung des XBee S1 Logs

Zum Beispiel Zeile 169 aus der Log Datei log_xbS1-log.dcf

Die APS Frames enthalten zwar ebenfalls die Daten, jedoch beginnen die Nutzdaten schon im nwkHeader. Je nach Nutzdatenlänge wird nach dem apsHeader noch ein Data Feld angezeigt, die den restlichen Dateninhalt beinhaltet.

**MAC Header** { 61 8C 78 90 47 03 41 46 41 00 A2 13 00 01 01 }

| Anz. Byte | Hex-Wert | Bedeutung |
|---|---|---|
| 2 | 8C 61 | frameCtl |
| 1 | 78 | seqNo |
| 2 | 47 90 | dstPanId (ID) |
| 8 | 00 13 A2 00 41 46 41 03 | dstExtAddr (DH/DL) |
| 2 | 01 01 | srcShortAddr (MY) |

Im BitCatcher werden die Daten als NWK und APS Header angegeben, dies liegt an der Übergabe vom Sniffer zum Programm und entspricht nicht ganz des gesendeten Strings, daher kann wohl die 0x0028 ignoriert werden, denn die dürften vom Sniffer stammen.
Danach folgen die Nutzdaten:

| { | 28 | 00 | 48 | 61 | 6C | 6C | 6F | 20 | 58 | 42 | 65 | 65 | 0D | } |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| { | . | . | H | a | l | l | o | | X | B | e | e | <CR> | } |

Im BitCatcher werden sowohl unbenannte als auch Netzwerk Fenster Angezeigt. Diese enthalten aber ebenfalls die Nutzdaten und werden nur vom BitCatcher/Sniffer fehlerhaft deklariert.

Scan Requests stören zudem die Übertragung, die Ursprüngliche Nachricht kann dabei eliminiert oder verändert werden, zudem Fangen die Module diese Nachrichten auf und geben diese über die Konsole wieder. Im Log können die Daten näher betrachtet werden. Betroffene Pakete sind #64 – #67 sowie #72 – #73 und #96 – #99. Dieser Vorgang wird nicht weiter Dokumentiert, da diese während Folgetests nicht reproduzierbar waren.

Ein Versuch mit API Frames wird ebenfalls nicht aufgelistet, da dieser identisch zu den AT Frames ist, was wieder ein Beweis dafür ist, dass die Daten vor dem Versenden in API Frames umgewandelt werden. Bei den 802.15.4 Modulen kann eine Textnachricht über ein TX Request übermittelt werden, wahlweise über die 64-bit (0x00) oder über die 16-bit Adresse (0x01).

Mit den verwendeten Testdaten würde dieses Fenster wie folgt aussehen.

(64-bit Adresse):

7E 00 16 00 01 00 13 A2 00 40 E9 7A E3 00 48 61 6C 6C 6F 20 58 42 65 65 0D 42

(16-bit Adresse):

7E 00 10 01 01 01 01 00 48 61 6C 6C 6F 20 58 42 65 65 0D 7A

| 64-bit Zieladresse | 16-bit Zieladresse | |
|---|---|---|
| 7E | 7E | Delimiter |
| 00 16 | 00 10 | Länge |
| 00 | 01 | API Typ |
| 01 | 01 | Frame ID |
| 00 13 A2 00 40 E9 7A E3 | 01 01 | Zieladresse |
| 00 | 00 | Optionen<br>[00] ACK / [01] kein ACK<br>[04] mit Broadcast PANID |
| 48 61 6C 6C 6F 20 58 42 65 65 0D | 48 61 6C 6C 6F 20 58 42 65 65 0D | Textinhalt |
| 42 | 7A | Checksumme |

## 7.4 Berechnung der Checksumme

0xFF - ( API Typ + Frame ID + Zieladresse + Optionen + Textinhalt ) = Checksumme
0xFF - ( 0x00 + 0x01 + 0x33B + 0x00 + 0x381 ) = 0xFFFFFFFFFFFFA42
0xFF - ( 0x01 + 0x01 + 0x02 + 0x00 + 0x381 ) = 0xFFFFFFFFFFFFD7A

Von den Ergebnissen werden jeweils nur die letzten zwei Zeichen verwendet und ein Byte gefüllt. Die Summe der Byte-Werte (alle Byte nach der Länge) ergeben das Ergebnis 0xFF.

# 8 XBee Robustheit Analyse

Zum Testen der Robustheit werden verschiedene Befehle Eingeben und deren Rückgabewert kontrolliert. Auch hier wird zum Protokolieren das Tool Free Device Monitoring Studio verwendet. Da das Programm X-CTU Fehlereingaben schon vorher abfängt wird stattdessen PuTTY genutzt um eine Hardware nähere Abfrage durchzuführen.

Es werden folgende Testfälle durchgeführt:

T010 : (CH) Eingabe einer Falschen Kanalnummer
T020 : (DH / DL) Eingabe einer falschen Zieladresse
T030 : (SH/SL) Versuch die Seriennummer zu Ändern
T040 : (ID) Verschiedene PAN ID‘s
T050 : (MY) Eingabe einer nichtzulässigen 16-bit Netzwerkadresse
T060 : (MM) Eingabe eines Falschen Wertes für den MAC Mode
T070 : (EE) Eingabe eines ungültigen Wertes für die AES Verschlüsselung
T080 : (KY) Eingabe eines falschen AES Schlüssels
T090 : (NI) Eingabe eines nichtzulässigen Knotennamens
T100 : (BD) Test einer nichtzulässigen Baudrate
T110 : (JV) Eingabe eines nicht für dieses Modul zugelassene AT Kommandos
T120 : Falsche Eingabe der AT Kommandos

## 8.1 T010 : (CH) Eingabe einer Falschen Kanalnummer

Die Reichweite der Kanäle liegt zwischen 0x0B und 0x1A.
Die Eingabe muss als Hex-Wert übergeben werden.

T011 : Eingabe von einen Wert < B.
Ergebnis: ERROR

T012 : Eingabe eines Wertes > 1A.
Ergebnis: ERROR

T013 : Eingabe von alphabetischen Zeichen die nicht im Hex-Alphabet vorkommen.
Ergebnis: ERROR

T015: Eingabe von Sonderzeichen.
Ergebnis: ERROR

## 8.2 T020 : (DH / DL) Eingabe einer falschen Zieladresse

Die Reichweite der Kanäle liegt jeweils zwischen 0x0 und 0xFFFFFFFF.
Die Eingabe muss als Hex-Wert übergeben werden.

T021 : Eingabe von alphabetischen Zeichen die nicht im Hex-Alphabet vorkommen
Ergebnis: OK, aber bei Abfrage immer noch Standartwert 0

T022: Eingabe von Sonderzeichen.
Ergebnis: OK, aber bei Abfrage immer noch Standartwert 0

T015: Eingabe von einen Wert der außerhalb des Wertebereichs liegt.
Ergebnis: OK, es werden nur die ersten 8 Zeichen gespeichert, die restlichen werden nicht in das zugehörige zweite Feld gespeichert.

## 8.3 T030 : (SH/SL) Versuch die Seriennummer zu Ändern

Die AT Befehle SH und SL sind read only Befehle.

T031: Versuch unter SH einen Wert zu speichern.
Ergebnis: ERROR

T032: Versuch unter SL einen Wert zu speichern.
Ergebnis: ERROR

## 8.4 T040 : (ID) Verschiedene PAN ID's

Die Reichweite der Kanäle liegt zwischen 0x0 und 0xFFFF.
Die Eingabe muss als Hex-Wert übergeben werden.

T041 : Eingabe von alphabetischen Zeichen die nicht im Hex-Alphabet vorkommen
Ergebnis: OK, bei der ersten Eingabe „koptz“ speichert er AD, bei der zweiten 0

T042: Eingabe von Sonderzeichen.
Ergebnis: OK, Wert wird jedoch auf 0 gesetzt

T045: Eingabe von einen Wert der außerhalb des Wertebereichs liegt.
Ergebnis: ERROR

## 8.5 T050 : (MY) Eingabe einer nichtzulässigen 16-bit Netzwerkadresse

Die Reichweite der Kanäle liegt zwischen 0x0B und 0xFFFF.
Die Eingabe muss als Hex-Wert übergeben werden.

T051 : Eingabe von alphabetischen Zeichen die nicht im Hex-Alphabet vorkommen
Ergebnis: OK, Wert wird jedoch auf 0 gesetzt

T052: Eingabe von Sonderzeichen.
Ergebnis: OK, Wert wird jedoch auf 0 gesetzt

T055: Eingabe von einen Wert der außerhalb des Wertebereichs liegt.
Ergebnis: ERROR

## 8.6 T060 : (MM) Eingabe eines Falschen Wertes für den MAC Mode

Es gibt 4 Möglichkeiten mit den Werten 0 – 3.

T061: Eingabe eines ungültigen Wertes.
Ergebnis: ERROR

## 8.7 T070 : (EE) Eingabe eines ungültigen Wertes für die AES Verschlüsselung

Zwei Zustände möglich aus (0) oder an (1).

T071: Eingabe eines ungültigen Wertes.
Ergebnis: ERROR

## 8.8 T080 : (KY) Eingabe eines falschen AES Schlüssels

Die Zeichenkette muss genau 32 Zeichen lang sein.
Die Eingabe muss als Hex-Wert übergeben werden.

T081: Eingabe Eines zu kleinen Wertes.
Ergebnis: OK Key kann aber nicht abgefragt werden, damit ist es unmöglich zu sagen, ob der Wert gespeichert wurde oder nicht.

T082: Eingabe eines aus 32 „0“ Zeichen bestehen String.
Ergebnis: OK

T083: Eingabe eines Wertes der größer als 32 Zeichen ist.
Ergebnis: OK. Vermutlich werden hier die ersten 32 Stellen gespeichert und alles Übrige abgeschnitten

T084: Eingabe eines Strings der aus nicht Hex-Werten besteht.
Ergebnis: OK. Key kann aber nicht abgefragt werden, damit ist es unmöglich zu sagen, ob der Wert gespeichert wurde oder nicht.

## 8.9 T090 : (NI) Eingabe eines nichtzulässigen Knotennamens

Es kann als String 0 – 20 Zeichen übergeben werden.

T091: Eingabe eines Strings mit einer Zeichenlänge von mehr als 20 Zeichen.
Ergebnis: OK, nach 20 Zeichen wird der String abgetrennt.

T092: Eingabe von einen String mit Sonderzeichen.
Ergebnis: OK, wird problemlos gespeichert, auch Sonderzeichen wie @, € und µ

T093: Eingabe eines Strings der normal in C ein Hex-Wert Darstellt – „\A24B“.
Ergebnis: OK, wird ebenfalls genau so dargestellt und nicht als Hex-Wert interpretiert

## 8.10 T100 : (BD) Test einer nichtzulässigen Baudrate

Die Baudrate wird nicht als Zahl übergeben sondern als Zustand, es gibt 8 verschiedene Möglichkeiten zwischen 0 und 7.

T101: Eingabe eines ungültigen Zustandes mit einer Zahl
Ergebnis: ERROR

T102: Eingabe eines ungültigen Zustandes mit einen Buchstaben.
Ergebnis: ERROR

## 8.11 T110 : (JV) Eingabe eines nicht für dieses Modul zugelassene AT Kommandos

Das Kommando JV ist die Kanal Verifikation und steht nur den Router Modulen der ZigBee Modellen zur Verfügung.
Es gibt nur zwei Zustände 0 und 1.

T111: Es wird versucht den JV Befehl abzufragen.
Ergebnis: ERROR

T112: Es wird versucht einen Gültigen Wert einzuspeichern.
Ergebnis: ERROR

T113: Es wird versucht ein Ungültigen Wert einzuspeichern.
Ergebnis: ERROR

## 8.12 T120 : Falsche Eingabe der AT Kommandos

Damit die AT Kommandos Problemlos Funktionieren muss die Reihenfolge der Eingabe beachtet werden. In diesen Versuch wird die Reihenfolge absichtlich missachtet.

T121: Eingabe von AP (Befehl der zum Aufwecken des Moduls geschickt wird, wenn man sich mit diesen über das X-CTU verbinden möchte.)
Ergebnis: keine Antwort

T122: Eingabe von CHAT, DHAT, SHAT, ATBD, ATJV, und EEAT
Ergebnis: OK, nach etwas Wartezeit kann es sein, dass das Modul einen Wert Zurück gibt. Diese Situation war nicht bei allen Befehlen reproduzierbar und trat nur bei SHAT und EEAT auf.

T123: Eingabe von DHAT0123F
Ergebnis: ERROR

T124: Eingabe eines wild zusammengewürfelten Befehls DJET1, AETE1
Ergebnis: keine Antwort

## 8.13 Zusammenfassung

Es wurden nicht alle Testmöglichkeiten durchgeführt. Anhand der Bestehenden kann man jedoch erkennen, dass nicht alle Fehlermöglichkeiten abgedeckt wurden und entsprechende Händler aufgerufen werden um eine Fehlerstatusmeldung zurückzugeben oder Standartwerte zu setzen. Zu dem musste das Modul nach dem Starten des X-CTU Programmes zurückgesetzt werden. Daraus ist zu schlussfolgern, dass einige nichterlaubte Werte Gespeichert wurden und somit zur Fehlfunktion beim Auslesen führen.

Bei einer Falschen PAN ID ist es immer noch möglich eine Verbindung aufzubauen, so lange man auf beiden Modulen denselben gespeicherten Wert erhält. Dies war jedoch erst nach mehreren Versuchen erfolgreich. Bei einer Eingabe eines falschen AES Schlüssels kann ein Datenaustausch stattfinden muss aber nicht, zu dem reagieren die Module zum Teil nicht mehr auf Eingaben und die Entschlüsselung ist fehlerhaft.

# 9 Die Mikrocontroller Funkkommunikationsbibliothek - µracoli

µracoli ist eine Abkürzung und steht für *microcontroller radio communications library* und ist eine Bibliothek die, die Nutzung mit Atmels IEEE-802.15.4 Funkmodulen vereinfacht und deren Möglichkeiten demonstriert. Die erste Version ging am 16. Februar 2008 online. Am 17. Februar 2014 wurde die Version 0.4.2 veröffentlicht, diese ist Zeitgleich auch die aktuellste Version. Auf dem Repository befindet sich am 14.10.2016 die Version rc0.5.0rc+.

Atmel liefert mit ihrer Bibliothek ebenfalls ausprogrammiert und funktionsfähig ein 802.15.4 Sniffer, einige Beispiele und ein Wireless UART Programm.

## 9.1 Für AT Kommandos wichtige Programm Daten und Infos

### 9.1.1 SCons

Im aktuellen Build wird die Programmbibliothek nicht mehr über Makefiles kompiliert sondern über das Python Zusatztool SCons [14]. Das Programm ist Open Source [15] und kombiniert die verschieden klassischen Make Tools, wie autoconf, automake und ccach in einen. Dabei werden die Konfigurationsdateien in Python geschrieben, unterstützt wird jedoch auch eine ganze Reihe weitere Programmiersprachen sowie ein paar wichtige Markup-Makro-Sprachen, wie z.B. C, C++, Fortan, Java, Yacc, Lex, Qt, Tex und LaTex.

Das Tool soll laut Entwickler nicht nur die verschieden Make-Tools kombinieren sondern auch automatisch Abhängigkeiten erkennen, einfach durch Benutzerdefinierte Builders erweiterbar sein und Microsoft VS .NET so wie cross compiling unterstützen.

Die SConstruct Daten müssen von uns nicht mehr geschrieben werden, sie liegen den aktuellen Build auf dem Repository bei. Benötigt wird jedoch Python mit einer Version, die kleiner als Version drei ist. Der Nachteil an diesem Tool ist, das es nur über den Command Prompt aus dem Hauptverzeichnis, in dem auch die SConstruct Daten liegen ausführbar ist.

Um eine neue Applikation oder ein neues Board hinzuzufügen müssen lediglich die *boards.cfg* und/oder die *applications.cfg* im Ordner Config editiert werden.

[14] Homepage von SCons Projekt: http:://scons.org
[15] Repository von SCons: https://bitbucket.org/scons/scons

**board.cfg:**

| Befehlszeile | Beschreibung |
|---|---|
| [BoardCompilName] | Name, der zum kompilieren verwendet wird |
| Comment: | * Beschreibung des Boards in einen Satz |
| aliases: | * Namen, die im Programm Code ebenfalls für das Board verwendet wurden |
| image: | * Bild vom das Board für die Doxigen Dokumentation |
| url:* | * Händlerhomepage |
| include | Alle Headerdateien, die für das Board benötigt werden (außer board.h) |
| cpu | * verbaute CPU |
| bootoffset | Startposition des Bootloaders |
| ccflags | * Kompiler Kommandos |
| f_cpu | CPU Frequenz als unsigned long |
| baudrate | Standard Baudrate |
| sensors | Namen der Sensoren, für die Headerdateien bereit gestellt wurden (Src\Lib\Inc\sensors) |
| provides | Funktionen, die bereitgestellt werden (müssen) |
| lfuse* | Fuses, Notation in Hexadezimal mit führenden ‘0x’ (Diese Zeilen werden benötigt, wenn man via Python Script den CPU flashen will) |
| hfuse* | |
| efuse* | |

* Optional, muss nicht hinzugefügt werden

**applications.cfg**

| **Befehlszeile** | **Beschreibung** |
|---|---|
| [API NAME] | API Name, der zum kompilieren verwendet wird |
| requires: | Die Funktionen die für die API benötigt werden |
| excludes: | * Boardbezeichnungen, die von der API nicht unterstützt werden (muss identisch mit der Bezeichnung in der board.cfg sein) |
| sources: | Ort und Namen der C-Dateien (ausgehend vom Ordner Src) |
| headers: | Ort und Namen der Headerdateien (ausgehend vom Ordner Src) |
| flags: | * (unbekannte Auswirkung) |
| ingroup: | Bei einer Abstrahierenden API kann hier der Name der Quell API angegeben werden, dadurch werden *sources* und *headers* optional |

* Optional, muss nicht hinzugefügt werden

### 9.1.2 board.h

In der Datei bord.h wird neben unterschiedlichen Default Definitionen unterandern auch der für die Knotenpunkt Konfiguration relevante struct-Typ bereitgestellt.

```
typedef struct {
   /*The short address of the node.*/
   uint16_t short_addr;
   /*The PAN ID (network ID) of the node.*/
   uint16_t pan_id;
   /*The MAC address of the node (EUI64).*/
   uint64_t ieee_addr;
   /*The radio channel.*/
   uint8_t  channel;
   /*For future extensions, but can be used to store user data.*/
   uint8_t _reserved_[2];
   /*Ibutton CRC to validate if the structure is correct.*/
   uint8_t crc;
} node_config_t;
```

Aus diesen Struct sind die 16-bit short_addr, 16-bit pan_id, 64-bit ieee_addr und die 8-bit channel Deklarationen interessant und können auf andere Strukturen übertragen werden. Ebenfalls werden folgende Funktionen bereitgestellt:

`void store_node_config_eeprom(node_config_t *ncfg, uint8_t * offset)`
um die Knotenkonfiguration in den EEPROM Speicher zu schreiben

`uint8_t get_node_config_eeprom(node_config_t *ncfg, uint8_t * offset)`
um die Knotenkonfiguration aus dem EEPROM Speicher zu holen

`uint8_t get_node_config(node_config_t *ncfg)`
holt die Knotenkonfiguration aus dem Flash Speicher

### 9.1.3 board_*.h

Neue Boards werden in einer board_<board type>.h definiert. In ihnen finden sich die Definitionen der Radio Typen (RADIO_TYPES), die für die Kompilierung, setzen der Board spezifischen Pins/Ports und Definitionen wichtig sind.

Zum Beispiel werden für den ConBee die Einstellungen für den Raspberry Pi benötigt, dazu wird beim kompilieren geprüft ob raspbee mit übergeben wird und anschließend der Radio Typ ATmega256rfr2 übergeben/ definiert.

```
...                    #elif defined(raspbee)                    #define R
```

Wichtig ist auch die korrekte und einheitliche Definition zur board.cfg, den die Definition BOARD_TYPE wird mit der boardscfg.py aus dieser Datei ausgelesen und an die C Daten beim kompilieren übergeben.

Ist die Board Definition gesetzt, dann können die LED, PIN/PORT Werte gesetzt werden. Die µracoli Programmierer arbeiten dabei in zwei verschiedenen Varianten (je nach Board Typ), entweder als #define oder als normale C Funktion.

### 9.1.4 const.h

In dieser Datei sind alle unterstützte Board und Radio Typen definiert, ebenfalls wird das TX-Parameter-Struct zur Verfügung gestellt. Dieses Struct ist ein Bit Feld und es könne die chan, txp und die cca Deklarationen für die XBee Kommandos verwendet werden.
Hinweis: die Funktion die, die Werte in den EEPROM speichern findet man in der transmitter.h.

```
1  typedef int8_t  channel_t;
2  typedef struct {
3          /** current channel see sub register @ref SR_CHANNEL*/
4          channel_t chan;
5          /**  TX power index see sub register @ref SR_TX_PWR*/
6          unsigned int txp    : 4;
7          /**  CCA mode see sub register @ref SR_CCA_MODE */
8          unsigned int cca    : 2;
9          /**  ED threshold see sub register @ref SR_CCA_ED_THRES */
10         unsigned int edt    : 4;
11
12         /**  clkm control  see sub register @ref SR_CLKM_CTRL */
13         unsigned int clkm   : 3;
14
15 } trx_param_t;
```

### 9.1.5 hif.h

Das hif im Dateinamen bedeutet Hostinterface, diese Datei stellt Verschiedene Interfaceoperationen bereit:

`void hif_init(const uint32_t baudrate)`
initialisiert das Hostinterface

`uint16_t hif_get_number(int8_t base)`
liest mit hif_getc eine Integer über die UART ein

`int hif_get_dec_number(void)`
liest mit hif_getc eine Dezimalzahl über die UART ein

`int hif_split_args(char *txtline, int maxargs, char **argv)`
teilt ein String auf, kann zum Beispiel verwendet werden um AT Kommandos auszulesen oder API Frames aufzulösen

`uint8_t hif_get_blk(unsigned char *data, uint8_t max_size)`
liest ein Datenblock über die UART

`int hif_getc(void)`
liest ein Char aus der UART

`void hif_dump(uint16_t sz, uint8_t *d) oder als DUMP(sz,ptr)`
schreibt ein Hex-Dump in die UART

`void hif_printf(FLASH_STRING_T fmt, ...) oder als PRINTF(fmt, ...)`
schreibt formatierten String in die UART

`void hif_echo(FLASH_STRING_T str) oder als PRINT(fmt)`
schreibt String in die UART

`int hif_putc(int c)`
schreibt ein Char in die UART

`uint8_t hif_put_blk(unsigned char *data, uint8_t size)`
sendet ein Datenblock an die UART

`void hif_puts(const char *s )`
schreibt ein String in die UART

`void hif_puts_p(const char *progmem_s)`
schreibt String aus dem Programmspeicher in die UART

`HIF_PUTS_NEWLINE() hif_puts_p(FLASH_STRING("\n\r"))`

### 9.1.6 timer.h

Mit der timer.h wird eine allgemein gültige Timervariante implementiert. Sie stellt unteranderen die Millisekunden MSEC(v) zur Verfügung.

```
#define MSEC(v) ((time_t)(v / (1.0e3 * TIMER_TICK)))
```

TIMER_TICK wird Board spezifisch entsprechend in der board_*.h gesetzt. Ebenfalls wird ein Struct (time_stamp_t) mit dem Zeitstempel in Sekunden und Mikrosekunden bereitgestellt.

`timer_init(void)`
initialisiert die Timerfunktion

`timer_hdl_t timer_start(timer_handler_t *thfunc, time_t duration, timer_arg_t arg)`
Startet einen Timer mit einen Timer Handler.

`timer_hdl_t timer_restart(timer_hdl_t th, time_t duration)`
Startet einen Timer neu.

`timer_hdl_t timer_stop(timer_hdl_t th)`
Stopt einen Timer.

`time_t timer_systime(void)`
Gibt die aktuelle Systemzeit in tics zurück.

`void timer_set_systime(time_t sec)`
Setzt die aktuelle Systemzeit in Sekunden. Wie gewöhnlich wird dabei vom Standard Datum, den 1.1.1970 ausgegangen.

`void timer_get_tstamp(time_stamp_t *ts)`
Gibt die aktuelle Systemzeit zurück.

### 9.1.7 transceiver.h

In den folgenden zwei Zeilen wird auf eine Headerdatei referenziert, die erst während des Kompiliervorganges aus einer txt Datei erstellt wird. Die Datei ist im Ordner Templates zu finden.

```
#elif RADIO_TYPE == RADIO_ATMEGA256RFR2 || RADIO_TYPE ==
↪ RADIO_ATMEGA2564RFR2

#include "atmega_rfr2.h"
```

Diese setzt alle relevanten Einstellung für den Transceiver, wie zum Beispiel Registereinträge und Transmitterübertragungsrate.

Die transceiver.h baut auf dieser Datei auf und stellt ähnlich wie die hif.h verschiedene Interfaceoperationen zur Verfügung.

`void trx_io_init (uint8_t spirate)`
Transceiver IO Initialisierung

`void trx_set_irq_handler(trx_irq_handler_t irqhandler)`
Setzt den Pointer für den IRQ Handler

`void trx_reg_write(trx_regaddr_t addr, trx_regval_t val)`
Schreibt in das Register.

`uint8_t trx_reg_read(trx_regaddr_t addr)`
Liest aus dem Register.

`trx_regval_t trx_bit_read(trx_regaddr_t addr, trx_regval_t mask, uint8_t pos)`
Liest das Subregister.

`void trx_bit_write(trx_regaddr_t addr, trx_regval_t mask, uint8_t pos, trx_regval_t value)`
Schreibt in das Subregister.

`void trx_frame_write(uint8_t length, uint8_t *data)`
Sendet ein Frame über den Transreceiver.

`uint8_t trx_frame_read(uint8_t *data, uint8_t datasz, uint8_t *lqi)`
Lies ein Frame über den Transreceiver.

`uint8_t trx_frame_read_crc(uint8_t *data, uint8_t datasz, bool *crc_ok)`
Liest ein Frame inclusive mit CRC check über den Transreceiver. (CRC gespeichert)

`uint8_t trx_frame_read_data_crc(uint8_t *data, uint8_t datasz, uint8_t *lqi, bool *crc_ok)`
Liest ein Frame inclusive mit CRC check über den Transreceiver. (CRC verworfen)

`uint8_t trx_frame_get_length(void)`
Gibt die länge des empfangen Frames wieder.

`void trx_sram_write(trx_ramaddr_t addr, uint8_t length, uint8_t *data)`
schreibt in SRAM

`void trx_sram_read(trx_ramaddr_t addr, uint8_t length, uint8_t *data)`
Liest aus SRAM

`void trx_parms_get(trx_param_t *p)`
Holt Transceiver Parameter.

`uint8_t trx_parms_set(trx_param_t *p)`
Setzt Transceiver Parameter.

`uint8_t trx_set_datarate(uint8_t rate_type)`
Setzt die Datenrate.

`uint8_t trx_get_datarate(void)`
Holt die aktuell verwendete Datenrate.

`uint8_t trx_get_number_datarates(void)`
Gibt die verfügbaren Datenraten aus.

`void * trx_get_datarate_str_p(uint8_t idx)`
Gibt ein Pointer auf den Datenratenstring im Programmspeicher zurück.

`void * trx_decode_datarate_p(uint8_t rhash)`
Decodiert den HASH Wert und speichert den Pointer des Datenratenstring in den Programmspeicher.

`uint8_t trx_get_datarate_str(uint8_t idx, char * rstr, uint8_t nlen)`
Speichert die Kopie einer Datenrate in ein Puffer.

`uint8_t trx_decode_datarate(uint8_t rhash, char * rstr, uint8_t nlen)`
Decodes a hash value and returns a datarate string pointer.

`uint16_t trx_rate_to_byte_us(uint8_t rate_type)`
Gibt die Datenrate als Byte Länge zurück.

`static inline uint8_t trx_init(void)`
`uint8_t trx_check_pll_lock(void)`
Initialisierung der Radio Funktionen.

`int trx_identify(void)`
Identifiziert den Radio Typ.

`void trx_set_panid(uint16_t panid)`
Schreibe PANID in den Adressfilter.

`void trx_set_shortaddr(uint16_t shortaddr)`
Schreibt die 16-bit Adresse in den 16-bit Adressfilter.

`void trx_set_longaddr(uint64_t longaddr)`
Schreibt die 64-bit Adresse in den 16-bit Adressfilter.

## 9.2 µracoli kompilieren

ToDo

## 9.3 Der Sniffer

Der mitgelieferte Sniffer kann eine große Hilfe bei Programmierung darstellen, denn anders als bei der ConBee/Bitcatcher Version kann der auch 802.15.4 Daten sniffen und liefert eine Detailausgabe an das Programm Wireshark.

Was wird benötigt:

- Ein ConBee Stick[16]
- Download von der µracoli Webseite [17]
  - uracoli-sniffer-0.4.2.zip (enthält die benötigten Python Dateien)
  - uracoli-src-0.4.2.zip (enthält Source Code zum kompilieren)
- oder die aktuelle Build Version von Savannah [18]
- Wireshark [19] (Verwendete Version 2.2.0)
- Python [20] mit einer Versionsnummer, die kleiner als Version 3 ist (verwendete Version Python 2.7.12)
- Python Serial 2.7 Plugin [21]
- SCons Python Plugin [22] (wird ab der µracoli Version 5.0.0rc unterstützt)

### 9.3.1 Python

Nach der Installation von Python muss der Pfad zum Python Compiler manuell gesetzt werden, die Installation bietet die Option zwar an, setzt aber den Eintrag nicht.
Die Kommando Zeile für das Terminal:

```
$ set PATH=c:\Python27;c:\Python27\Scripts;%PATH%
```

### 9.3.2 Firmware

**Version 0.4.2**
Die Sniffer Dateien befinden sich in dem Ordner *uracoli-src-0.4.2.\sniffer*, bevor man diese über die Makefile kompiliert muss in Zeile 373 *dwarf-2* eingefügt werden. (Windows)

[16] ConBee von dresden-elektronik: http://www.dresden-elektronik.de/funktechnik/solutions/wireless-light-control/conbee/
[17] Downloadseite von µracoli http://uracoli.nongnu.org/download.html
[18] Savannah Repository von µracoli: hg.savannah.gnu.org/hgweb/uracoli/
[19] Homepage von Wireshark: https://www.wireshark.org/
[20] Download Seite von Python: https://www.python.org/downloads/
[21] Download Seite vom Python Serial Plugin: https://sourceforge.net/projects/pyserial/
[22] SCons Webseite: http://scons.org/

**aktuelle Build Version**
Die aktuelle Version wird nicht mehr über Makefiles kompiliert sondern über das Build Plugin für Python (vgl. Kapitel 9.2). Die Version derfa2 gibt es nicht mehr und wurde in Raspberry Pi und ATmega256rfr2 auf RF Note unterteilt. Da im ConBee der selbe Chipsatz wie auf dem Raspberry verbaut wurde ist für uns die Firmware mit dem Namenszusatz *raspbee* relevant.

Anschließend kann die Firmware mit dem Namen derfa2 oder raspbee auf den ConBee USB Stick gebrannt werden.

### 9.3.3 How to sniff Data

Der Ordner *script* wird aus der *uracoli-src-0.4.2.zip* Datei entpackt oder man verwendet den *install\scripts* Ordner aus dem aktuellen Build. Jetzt kann man über eine Batch Datei oder den Command Terminal (cmd) folgende Zeile ausführen:

```
$ python <path to uracoli-src-0.4.2. folder>/script/sniffer.py -p COM<PORT
NUMBER> | <PATH TO WIRESHARK>/wireshark -ki -
```

Das Programm Wireshark startet mit dem Python Tool

# 10 Prototyp Entwicklung

## 10.1 Ideen die Während der Analyse Entstanden sind

1. Timer könnten über die C Bibliothek time.h und einer while-Schleife gesteuert werden. Jedoch ist zu beachten, dass es unmöglich ist auf jeder Plattform ein und dieselbe Genauigkeit der Zeitfunktion zu garantieren. Um eine exakte Timerverarbeitung zu realisieren sind die Controlerspezifischen Programm, bzw. Hardwarekonfiguraionen nötig. Als alternative bringt die µracoli Bibliothek ebenfalls eine Timerfunktion mit.

   Die Möglichkeit die Timer Funktion der C Bibliothek anzuwenden ist für die Mikrocontroller keine Option, denn die Architektur der Chips ist wesentlich einfacher als die der Desktop PCs. Hier muss mit Timerinterrupts gearbeitet werden, wie sie von der µracoli Bibliothek schon umgesetzt wurde. Es empfiehlt sich daher aus Kompatibilitätsgründen Pointer für die jeweiligen Funktionen anzulegen.

2. Bei asynchrone Verarbeitung ist zu beachten, dass bestimmte Variablen nicht überschrieben werden dürfen um den anschließenden Prozess nicht zu verfälschen oder zu unterbrechen. Es werden also Semaphoren und möglicher weise auch Spinnlocks benötigt.

3. Die Verarbeitung selbst muss in einen neuen Thread erfolgen. Begrenzen wir die Anzahl der Threads Manuell?

4. Punkt 2 und 3 kann mit pthreads kombiniert werden

5. Funktionen sollten nicht unnötige Kopien der variablen anfertigen um keinen Speicherplatz zu verschwenden und die Kompatibilität zu älteren Geräten zu erhöhen

6. Die Umsetzung der der AT befehle kann in zwei verschiedene Arten erfolgen:

a) Die XBee AT Kommandos werden eins zu eins übernommen und entsprechend abgearbeitet.

b) Es werden min zwei AT Bibliotheken hinterlegt, die am Ende Programmintern zusammen führen und ein und dieselbe Funktion ausführen.

## 10.2 AT Kommandos XBee

Mit *+++* initialisiert man den Zeitlich begrenzten Kommandomodus. Die Zeit in der ein Nachfolgendes AT Kommando eingegeben werden muss kann mit CT (Command Mode Timeout) festgelegt werden. Der Standartmodus ist allgemein der API Eingabemodus.

Ein AT Kommando sieht immer nach folgenden Schemata aus:

ATDT 1F<CR> oder ATDT1F<CR> [23]

Diese Zeile wird als String über den UART Port geschickt und auf dem Mikrocontroller sofort verarbeitet, wenn der Befehl lokale Änderungen vornimmt und zu ein API Frame zusammengesetzt wenn er weiter versendet wird.

Es gibt zwei verschiedene AP Modes. Ein die jeden Befehl sofort umsetzt und speichert (0x08) und ein, der alle Kommandos in eine Warteschlange einreiht (0x09). Diese Veränderung müssen mit *AC* (Apply Changes) bestätigt werden. Es gibt jedoch ein paar wenige Kommandos, die können nur einzeln ausgeführt und müssen immer Bestätigt werden, dazu gehören z.B. die Security Kommandos.
Mit ATCN oder nach der Zeit CT, nach dem der letzte Befehl eingeben wurde verlässt der Nutzer den Kommandomodus.

Um über den AT Modus Werte auszulesen, wird der Parameterabschnitt einfach weggelassen, ein Wertebereichstestest wie beim wirelessAT gibt es nicht. Sollen AT Befehle über eine Remoteschnittstelle zu ein anderes Gerät weiter versendet werden, dann müssen die Befehle gemäß der API Frame Spezifikation formatiert werden. Die AT Kommandos alleine sind nur Lokal nutzbar.

## 10.3 API Frame

Das API Fenster beginnt mit dem Start Delimiter 0x7E, gefolgt von der Fenstergröße (MSB und LSB), den API Spezifischen Inhalt und der Checksumme. Die Größe des Fensters selbst kann je nach Nachfrage- oder Befehlstyp variieren.

Bsp. eines API Frames als Hex-String, die Hex-Bezeichnung 0x wird hier weggelassen:
7E 0016 11 01 0013A200 40401234 FFFE 00 00 0031 0000 00 00 76 00 CE

[23] Digi User Guid - XBee / XBee-PRO S1 802.15.4 (Legacy), Seite 33

**Header**

| | |
|---|---|
| Start Delimiter | 0x7E |
| Länge | 0x0016 |

**Content**

| | |
|---|---|
| Transmitterabfrage | 0x11 |
| Fenster ID | 0x01 |
| 64-bit Adresse | 0x0013A200 40401234 |
| 16-bit Adresse | 0xFFFE |
| Quellendpunkt | 0x00 |
| Zielendpunkt | 0x00 |
| LQI-, Nachtbarknotenabfrage | 0x0031 |
| Cluster ID oder ZigBee Device Profile ID | 0x00 00 (in dem Fall Command) |
| Broadcast Radius | 0x00 |
| Tx Option | 0x00 |
| Transaktionsnummer | 0x76 |
| Nutzlast für LQI Kommando | 0x00 |
| Checksumme | 0xCE |

Die Transmitterabfrage entspricht den API Frame Typ/ID [24]

| API Frame Names API | API ID |
|---|---|
| AT Command – Setzt und führt Kommandos sofort aus | 0x08 |
| AT Command - Queue Parameter Value – Setzt Kommandos in eine Warteschlange, um sie Auszuführen muss 0x08 genutzt werden oder das AT Kommando AC | 0x09 |
| ZigBee Transmit Request – Sendet Daten als RF Paket | 0x10 |
| Explicit Addressing ZigBee Command Frame – Erlaubt den APP-Layer für Datenübermittlung zu spezifizieren | 0x11 |
| Remote Command Request – Setzt Modulparameter auf ein Remotegerät, der Request muss mit einen AC an das Gerät gesendet und Bestätigt werden | 0x17 |
| Create Source Route | 0x21 |
| AT Command Response – Gibt nach einen Kommando einen Ergebnistext zurück | 0x88 |
| Modem Status | 0x8A |
| ZigBee Transmit Status – Nach einer TX Übertragung wird ein TX Statusnachricht ausgegeben | 0x8B |
| ZigBee Receive Packet (AO=0) – Das Modul sendet das empfangene Paket über UART weiter | 0x90 |
| ZigBee Explicit Rx Indicator (AO=1) – Das Modem sendet das empfangene Paket über UART weiter | 0x91 |
| ZigBee IO Data Sample Rx Indicator – Das Modul sendet das IO Sample über UART weiter | 0x92 |
| XBee Sensor Read Indicator (AO=0) – Das Modul sendet das IO Sample über UART weiter | 0x94 |
| Node Identification Indicator (AO=0) – Die Nachricht wird erhalten wenn der Knoten identifiziert werden soll | 0x95 |

[24] Digi User Guid - ZigBee RF Modules, Seite 111 ff.

| API Frame Names API | API ID |
|---|---|
| Remote Command Response – Sendet ein Remoteantwort | 0x97 |
| Over-the-Air Firmware Update Status | 0xA0 |
| Route Record Indicator – Erhält man wenn ein Routingkommando gesendet wurde | 0xA1 |
| Many-to-One Route Request Indicator – (über UART wenn eine Anfrage besteht) | 0xA3 |

## 10.4 Programmspezifische Grundgedanken

Der UART nimmt sofort jeden Buchstaben der getippt wird an und füllt ein Puffer, wird nach einer bestimmten Zeit nichts mehr hinzugefügt validiert die Firmware und gibt ggf. den Return-Wert oder Feedback (OK/ERROR) zurück.

Ist der API Frame bei den XBee Modulen aktiviert, dann reagiert das Programm auf eingegebene Delimiter.

Mögliche Implementierung (Pseudocode):

```
1  FRAMESTRUCT readUART(uint8_t *APmode) {
2      switch( uart_getc() ) {
3        case '+' : counted 3 times '+' ? yes -> return OK & enter
          ↪  command mode :   no -> return ERROR or nothing;
4                       break;
5        case '~' : fill buffer until '\n' & handle the Frame;
6                   if      ( APmode == 1 )
7                   else if( APmode == 2 ) /* frame handling with
                     ↪  different checksum calculation and escaping
                     ↪  character elimination */
8                   break;
9  }
10 }
```

## 10.5 Arbeitsschritte

* Senden Empfangen UART
Senden über TX an XBee Modul
Empfangen über RX Modul
Implementierung des Command Modes
Speichern und auslesen von Informationen in/aus den S-Registern

# Anhang

## Grafiken

## Literatur

1. Digi. (2015). *Customer Release Notes 93009373 G1*. http://ftp1.digi.com/support/firmware/93009373_G1.txt. (Besucht am 22.09.2016).

2. Digi. (2015). *Customer Release Notes 93009377 B1*. http://ftp1.digi.com/support/firmware/93009377_B1.txt.(Besucht am 22.09.2016).

3. Digi. (Mai 2016). *XBee / XBee-PRO S1 802.15.4 (Legacy)*. RF Modules. User Guid. PDF: 2016-09-23_90000982.pdf. https://www.digi.com/support/productdetail?pid=3257. (Besucht am 23.09.2016).

4. Digi. (Juni 2016). *ZigBee RF Modules*. XBee2, XBeePRO2, PRO S2B. User Guid. PDF: 2016-09-02_90000976.pdf. https://www.digi.com/support/productdetail?pid=3430. (Besucht am 02.09.2016).