dresden elektronik

Praktikumsarbeit

AT Kommandos

für IEEE 802.15.4 Funkmodule

Tobias Ehrlich

Dokumentenversion 0.1

24.02.2017

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Praktikumsarbeit unterstützt und motiviert haben.

Ein besonderer Dank gilt dabei der Firma dresden elektronik und Herrn Professor Arnold Beck, durch deren Unterstützung diese Arbeit erst ermöglicht wurde.

Dank Herrn Professor Arnold Becks fachlicher Unterstützung während des Studiums konnte ich mit einem reichhaltigen Fachwissen an diese Praktikumsarbeit herangehen. Zudem danke ich ihm auch für die Vermittlung von diesem Praktikumsplatz.
In der Firma dresden elektronik gilt mein besonderer Dank Manuel Pietschmann, der viel Zeit und Mühen investiert hat, mich durch stetig kritisches Hinterfragen und konstruktive Kritik, zu einer durchdachten These und Fragestellung zu verhelfen.

Außerdem gilt mein Dank meiner Mutter und meiner Kommilitonin, welche in zahlreichen Stunden Korrektur gelesen haben.

Inhaltsverzeichnis

1 Inhalt

In diesem Praktikum soll eine Programmschnittstelle entwickelt werden, die es ermöglicht sowohl mit AT Kommandos, als auch mit API Frames, Radio Frequenz Module (RF Modul) zu steuern und zu konfigurieren.

Die AT Kommandos, oder auch Hayes Befehlssatz, wurde von der Firma Hayes in den achtziger Jahren entwickelt, um die aus eigener Produktion hergestellten Modems zu steuern. Dieser Befehlssatz wurde später von der ITU (The International Telecommunication Union) aufgenommen und zum Betriebsstandard. Auch heute findet er immer noch Verwendung.

Das AT steht für ‚attention' (Achtung) und ist daran angelehnt, dass der nachfolgende Befehl eine höhere Priorität vor der Verarbeitung anderer Daten hat. Zudem vereinfachen diese Kommandos den Zugriff auf die interne Modulkonfiguration und ermöglichen dem Nutzer das Gerät schnell und einfach zu konfigurieren. Da jedoch die Weiterentwicklung des Standards nur sehr langsam voran geht, haben einige Firmen ihren eigenen AT Kommando Standard entwickelt und weiterentwickelt. Dazu gehört auch die Firma Digi International aus den USA.

Digi entwickelte, um unabhängig vom Standard zu sein, eine eigene *AT Kommando Implementation*, die um eine *API Frame Implementation* erweitert wurde. Die binär kodierten API Frames ermöglichen es mit einer kurzen Befehlszeile mehrere Informationen an das Gerät zu schicken oder vom Gerät zu empfangen, ohne mehrere Kommandos hintereinander auszuführen zu müssen.

Die am weit verbreitetsten RF Module der Firma Digi, sind die XBee Module [Abb. 1], welche vom Industriellen Kunden bis hin zur Do It Yourself (DIY) Szene verwendet werden.

Die zu entwickelnde API soll sich an diesen Kommandos anlehnen, sowie auf zukünftigen Produkten der Firma dresden elektronik laufen. Damit soll das Interesse verschiedener Entwickler auf Atmel AVR und Atmel SMART SAM R21[1] geweckt werden. Diese basieren auf dresden elektronik Funkprodukten, wie den USB Stick ConBee und dem Aufsatzmodul RaspBee für den Raspberry Pi, welche für die Lichtsteuerung über Funk Verwendung finden.

Die Zielgruppe reicht vom Industriellen Kunden bis hin zur DIY Szene, welcher sich mit den dresden elektronik Funkmodulen und 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, welche später ergänzt wird.

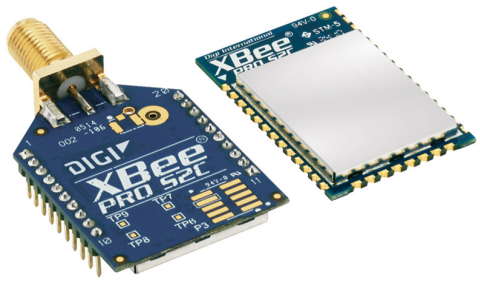

Abb. 1: XBee 802.15.4 Modul

[1] der SAM R21 besteht aus einen Atmel ARM Cortex-M0+ Controller (2017-02-15_Atmel-42223–SAM-R21_Datasheet.pdf)

2 Die Aufgabenstellung

Um spätere Kompatibilitätsprobleme zu vermeiden, wird die Aufgabenstellung allgemein gestellt. Die Auswertung der Kommandos, die in den verschiedenen Funktionssets verwendet werden, basiert zunächst auf einem XBee Modul (XB24) der Serie 1 (S1) mit dem jeweiligen Funktionsset und der dazugehörigen Firmware Version.

Die XBee Varianten Die XBee Module sind in verschiedenen Varianten erschienen. Es sollen alle XBee Versionen zum Vergleich aufgelistet werden, um einen groben Überblick über die Vielfältigkeit zu bekommen. Für die Analyse sollen aber vorrangig die Module zum Einsatz kommen, die auf dem Markt am meisten Verwendung finden und die Basis der anderen Module stellt.

Protokoll Typen Aufgrund der Vielfältigkeit der XBee Module verwendet Digi unterschiedliche Funkprotokolltypen. In diesem Abschnitt sollen die von Digi verwendeten Protokoll Typen kurz vorgestellt werden, um später bei der Planung des Prototyps zu entscheiden, ob eine Programmbibliothek verwendet wird, die ein solches Protokoll beinhaltet oder nicht.

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.

X-CTU Modulkonfiguration Analyse Zur Konfiguration und Testzwecken soll für die XBee Module das Tool *X-CTU* verwendet werden, welches von der Firma Digi kostenlos zur Verfügung gestellt wird. Damit eine eigene AT Kommando Implementation, zum Beispiel auf einem USB-Stick ConBee, mit diesem Tool funktioniert, muss vermutlich ein XBee Modul simuliert werden (ID, MAC Adresse, Version, . . . ?).

Um zu ermitteln welche Befehle zwischen dem X CTU und einem X Bee ausgetauscht werden sollen, muss ein *USB Serial Analyzer Tool* verwendet und die Ergebnisse umfangreich dokumentiert werden. Die Kommandos, die zusätzlich zu denen in der Statistik verwendet werden, sollen in dieser mit dem Vermerk „X-CTU“ aufgenommen werden.

Diese Statistiken, Analysen und Dokumentationen bilden die Basis für die Implementation des Prototyps.

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 eine Auswahl ermittelter Kommandos aus der Statistik, sollen daher Fehlerfälle definiert, getestet und dokumentiert werden.

Hierzu zählen beispielsweise Falscheingaben, unvollständige Eingaben, auslassen von Parametern, Timeouts, so wie zeitliches Verhalten bei Falscheingaben, 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 mit einer eigenen Lösung verbessern kann, um zum Beispiel das Blockieren des Moduls zu verhindern.

Programmbibliotheken Als zukünftig mögliche unterliegende Programmbibliotheken (Software Stacks) können Atmel BitCloud, Atmel MAC Stack, Atmel Lightweight Mesh, µracoli, TI Z-Stack und ähnliche angenommen werden.

Der zu verwendende Software Stack ist noch zu ermitteln. Denkbar sind Atmel BitCloud 3.3.0 oder µracoli. Letzterer hat den Vorteil, dass der Open Source Stack weiterentwickelt wird und potentiell Support für den ARM R21 erhält. In diesem Abschnitt sollen daher beide Stacks kurz vorgestellt werden.

Plattform unabhängige Definition einer abstrakten
AT Kommando API Der besondere Anspruch der AT Kommandos ist, dass diese auf beliebigen Plattformen eingesetzt werden können. Es sollen daher mehrere C-Module entstehen, die das AT Kommandohandling unabhängig vom verwendeten Mikrocontroller umsetzen.

Die API muss daher serielle Kommunikation, Timer und Taskhandler derart abstrahieren, dass diese mit geringem Aufwand auf die jeweilige Zielplattform portiert werden kann.

Alle Kommandos, inklusive die zur Konfiguration, müssen asynchron verarbeitet werden.

Planen des Prototypen Der Prototyp soll zunächst auf dem USB-Stick ConBee mit AVR ATmega256RFR2 lauffähig sein. Die Implementation wäre somit ebenfalls auf dem RaspBee und deRFmega256[2] Modulen einsetzbar, was für die Prototypen jedoch nicht der Fokus ist.

[2] dresden elektronik bietet mehrere Varianten an, unter anderen den RFR2 oder den RFA1 mit verschiedenen Atmel ARM Cortex-M Controllern

3 Die Protokolle

DigiMesh
Dieses Protokoll arbeitet nur mit einem Kontentyp, den Routern im Netzwerk, das heißt alle Knotenpunkte können Daten zu allen anderen Knoten über unterschiedliche 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 Netzwerkstabilität wird ebenfalls erhöht. Fällt ein Knoten aus, bedeutet das nicht, dass das ganze Netzwerk ausfällt. Aufgrund der Mesh-Struktur können die Daten unterschiedliche Routen durch das Netzwerk nehmen. Diese werden durch Eigendiagnosen, Selbstheilung, Netzwerkkonfigurationen und engmaschige Netzwerkoperationen mit möglichst wenigen Knotensprüngen umgesetzt.

DigiMesh arbeitet auf Basis eines proprietären Protokolls, das erlaubt eine größere Kontrolle der Programmierbarkeit und mehr Raum für Erweiterungen.[3]

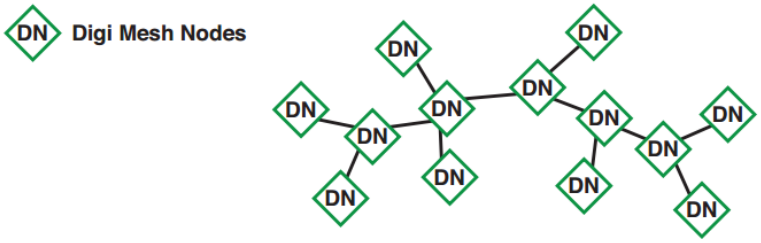

Abb. 2: DigiMesch Netzwerk

ZigBee
ZigBee ist eine Spezifikation, welche ein Framework für drahtlose Funknetzwerke beschreibt. ZigBee baut auf den IEEE 802.15.4 Standard auf und erweitert dessen Funktionalität insbesondere um die Möglichkeit des Routings und des sicheren Schlüsselaustausches.[3,4,5]

Das ZigBee Protokoll erweitert die Media Access Control (MAC) und Physical (PHY) Schicht um eine Application (APL), Security (SEC) und Network (NWK) Schicht. Anwendungen werden vom Programmierer in der Applikationsschicht implementiert.

Weitere Technologien, die es zum Beispiel ermöglichen Over-The-Air Firmware Updates, Energie- und Lichtkontrollapplikationen, sowie verschiedene Diagnose Programme anzuwenden, erweitern das Funktionsspektrum der Module.

Standardmäßig stehen dem Entwickler drei Gerätetypen zur Verfügung:

Koordinator, der niemals in den Energiesparmodus geht und das Netzwerk steuert
Router, der Endgeräte mit dem Koordinator verbindet
Endgerät, welches spezifische Aufgaben durchführt

[3] What a Mesh! Part 2-Networking Architectures and Protocols
[4] What is ZigBee Hauptseite der ZigBee Alliance: http://www.zigbee.org/what-is-zigbee/
[5] ZigBee Spezifikationen: Smart Energy Profile 2 (SEP 2)

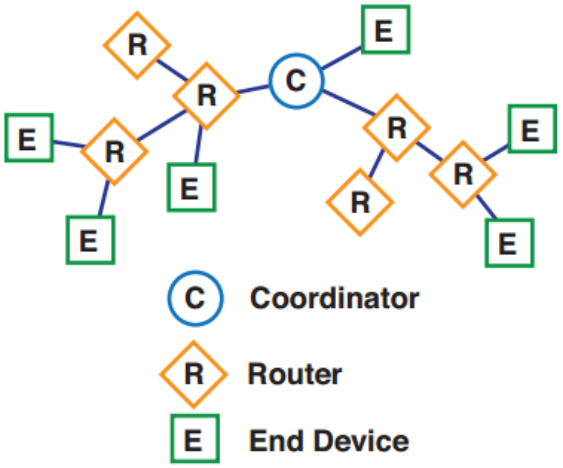

Abb. 3: ZigBee Netzwerk

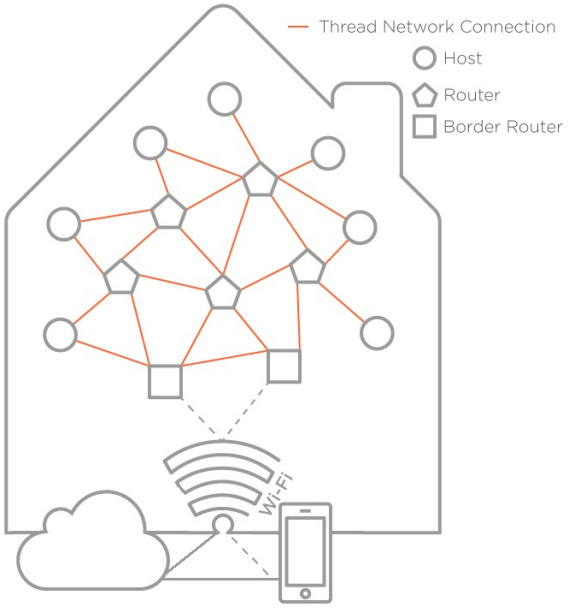

Abb. 4: Thread IPv6 Netzwerk

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.[6,7]

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 eines energiesparenden 802.15.4 Netzwerkes und ist für spezielle Aufgaben optimiert.

Als Basis werden die Schichten des 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, benötigt man nur ein Smartphone oder einen Computer und eine Border Router Schnittstelle. Das Modul sendet den Beitrittswunsch und wie beim Verbinden mit einem Wi-Fi Router, wird ein Passwort abgefragt.

Dies bietet gleichzeitig die Sicherheit, die auch ein normaler Rechner mit IPv6 besitzt. Die Netzwerke besitzen aufgrund der 802.15.4 Protokolle, die als Basis verwendet werden, schon dessen Vorteile. In Verbindung mit dem IPv6 bekommt man ebenfalls zusätzlichen Zugang zu allen anderen gängigen Sicherheitszertifikaten und Protokollen, wie zum Beispiel TLS, DTLS, SLS, SLL, etc.

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

[6] Digi Pressemitteilung zu Thread IPv6: https://www.digi.com/news/press-releases/digi-international-announces -Thread-ready-xbee-mod

[7] Thread Technical Brief

4 Die XBee Varianten

Digi bietet die XBee Module in unterschiedlichsten Varianten an, dabei tragen diese nicht immer die Marke im Namen. Es gibt angepasste Versionen, die für einen jeweiligen, speziellen Aufgabenbereich optimiert und zum Teil auf anderen Platinen vormontiert sind. Im Folgenden werden nur Module aufgelistet, die entsprechend mit dem Namen XBee bezeichnet wurden. Alle Module sind bisher in einer *Standard*, als auch in einer *PRO* Version erschienen. Ebenfalls verzeichnen die hier aufgelisteten Spezifikationen nur die Maximalleistungen der Module auf und sind daher nicht eindeutig einer Verkaufsversion zuzuordnen.

Diese Übersicht soll nur einen Überblick verschaffen, welche verschieden Versionen es gibt, daher werden nicht alle Spezifikationen aufgelistet. Sind weitere Informationen erforderlich, können diese der Datenspezifikation auf der Herstellerseite[8] entnommen werden!

4.1 Die Module

Module mit DigiMesh Protokoll Unterstützung

XBee SX ist die Energiesparvariante bei maximaler Effizienz. In ihr arbeitet ein 1 W 900 MHz XBee PRO Modul. Aufgrund SMT (Surface Mount) Technologie sind sie zudem platzsparend.

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

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

Frequenz:	900 MHz
Modell:	THT (Through-hole)
Reichweite:	bis ca. 14 km
Sendeleistung:	229 mA (250 mW) @ 24 dBm
Verbrauch im Sleep Mode:	2.5 µA
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 es werden keine zusätzlichen *9XStream*[9] Module benötigt.

[8] Digi Produktseite für XBee Module:https://www.digi.com/products/xbee-rf-solutions

[9] Die XStream Serie ist eine veraltete Vorgängerversion der XBee Module. Größtenteils besteht die Produktreihe aus Modems

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

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

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

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

Module mit ZigBee Protokollunterstützung

Die XBee ZigBee Module sind in zwei Varianten erhältlich, 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.[10]

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

Module mit 802.15.4 Protokollunterstützung

Diese Modelle sind seit der *Serie S2C* die *Alleskönner* 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:	2.4 GHz	2.4 GHz
Modell:	THT	THT
Reichweite:	bis ca. 3.2 km	1.6 km
Sendeleistung:	120 mA (63 mW) @ 18 dBm	215 mA (60 mW) @ 18 dBm
Verbrauch im Sleep Mode:	< 1 µA	< 10 µA
Verschlüsselung:	128-bit	128-bit

[10] Quelle: http://www.digi.com/blog/iot/what-makes-thread-a-secure-wireless-protocol/

Wi-Fi 802.11.b/g/n

Wi-Fi Modelle der Firma Digi sind dazu optimiert, sie über den DigiCloud Service zu konfigurieren und die M2M Kommunikation zu beschleunigen. Zudem wird dem Nutzer ermöglicht, Dank SoftAP[11] Technologie die Module mit einem 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 Module auch mit anderen XBee Varianten verbinden.

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

4.2 Unterschiedliche Antennen

Die XBee Module unterscheiden sich nicht nur in den Modelltypen, sondern auch in den Antennentypen. Sie besitzen jedoch laut Hersteller alle ähnliche Leistungen, unter anderem weniger als 0.1 dB Verlustrate, das Receiveransprechverhalten liegt bei rund -110 dBm und +30 dBm TX power. [12]

4.3 Relevanz für den Prototyp

Anhand der Spezifikationen läuft auf den meisten Modulen das IEEE 802.15.4 Protokoll im Hintergrund, die einzige Ausnahme ist das Wi-Fi Modul, welches mit einem 802.11.x Protokoll arbeitet.

Die einfache XBee S1 Form, stellt 802.15.4 Kommunikationen für Punkt-zu-Punkt und Sternnetzwerke zur Verfügung, während die S2 Module hingegen erweiterte Protokoll Funktionalitäten ermöglichen, wie zum Beispiel das ZigBee Mesh-Netzwerk. Die Schnittstelle für AT Kommandos und API Mode, sind bei S2 Modulen entsprechend umfangreicher.

Für die erste funktionelle Implementierung sind daher die S1 Module sinnvoll. Das heißt, die erste Entwicklung für den ersten Prototyp und dieses Dokument, werden zunächst mit dem IEEE 802.15.4 Protokoll erarbeitet. Die Erweiterung der Implementation, um die ZigBee/Mesh-Netzwerk Funktionen kann zu einem späteren Zeitpunkt ergänzt werden.

[11] Software, die ein nicht spezifiziertes Modul zu einen virtuellen Router macht. Wikipedia-Eintrag zu SoftAP: https://en.wikipedia.org/wiki/SoftAP

[12] Technische Antennenspezifikation: http://www.digi.com/resources/documentation/Digidocs/90000991/concepts/c_xbee_digimesh_antenna_options.htm

5 Modulanalyse

5.1 Das X-CTU Programm

Zur Konfiguration der Digi RF Module (XBee, XTend, XLR) stellt die Firma ein hauseigenes Konfigurationsprogramm zur Verfügung, welches mit der aktuellen Firmware der einzelnen Module und zum Teil mit vorkonfigurierten Sets die Benutzung nicht nur vereinfacht, sondern auch durch die integrierte Konsole, AP Frame Generator/Interpreter, Reichweitentest und Spektrum Analyse die Module komfortabel bedienen lässt.

Zum Zeitpunkt der Erstellung des Dokumentes gibt es zwei verschiedene Varianten, die *Next Gen*, in der Version 6.3.2 mit moderner grafischer Oberfläche (Java) für die Plattformen Windows, Linux und MacOS, sowie die *Legacy Edition* in der Version 5.2.8.6 für Windows.
Die Legacy Edition ist die ältere X-CTU Plattform von 2012. Diese Version hat nicht den ganzen Funktionsumfang und besitzt ein altes Oberflächendesign, unterstützt jedoch fast alle neueren RF Module. Wird demnach nicht der gesamte Funktionsumfang und keine moderne Oberfläche benötigt, so kann die ältere Programmversion verwendet werden, denn eine Konsole und die Konfigurationstabelle gibt es auch in dieser Edition.

Beide Versionen können kostenlos von der Digi Homepage heruntergeladen werden. [13]

5.2 Die Firmware Versionen

Mit den Konfigurationstools bekommen die Nutzer *eine Vielzahl* von Firmware Versionen mitgeliefert, diese unterscheiden sich jedoch nur in den vorkonfigurierten Sets, die einige Einstellungen schon beim Flashen übernehmen, die Basisfirmware bleibt dabei gleich.

Zur Analyse und Vorlage für den Prototyp werden von dem 802.15.4 Modul die Firmware aus der Produktfamilie XB24 und das Funktionsset 802.15.4 in der Version 10EF verwendet.
Für die ZigBee Module sollten zwei verschiedene Funktionssets aus der Produktfamilie XB24-ZB verwendet werden, denn im AT Modus ist das Funktionsfeld für den API Modus nicht verfügbar und auch weitere Konfigurationsfelder fehlen je nachdem, ob das Modul Koordinator oder Endgerät ist.

Die Folgende Tabelle Zeigt die wichtigsten Funktionssets, welche zur ersten Grobanalyse verwendet werden können, alle weiteren nicht aufgeführten Funktionssets benötigen noch ein zusätzliches Entwicklerboard.

Modul	Funktionsset (Name)	Firmware Version
XBee 802.15.4	802.15.4	10EF
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

[13] Download Seite vom X-CTU Konfigurationstool, https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu, Abgerufen am 22. September 2016

5.3 Das Auswertungstool

Zur Auswertung wird das Tool *Free Device Monitoring Studio* verwendet, welches als Freeware heruntergeladen werden kann. [14]

Nach dem Start des Programms gibt es ein Popup mit dem Hinweis dass, wenn man alle Funktionen verwenden möchte, gezahlt werden soll [Abb. 5]. Für die Auswertung ist dies jedoch nicht notwendig und das Fenster wird über den Schriftzug *Continue with limited features* geschlossen.

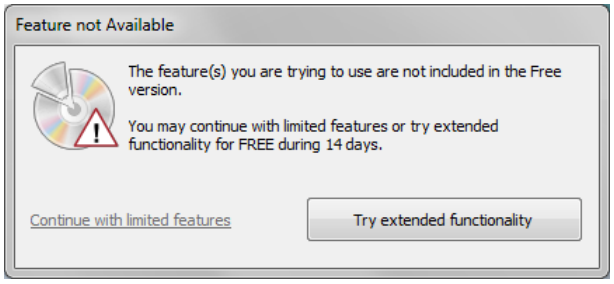

Abb. 5: Kaufaufforderung

Im ersten großen Fenster [Abb. 6] findet man auf der rechten Seite den *Device Tab* in dem alle seriellen, USB und Netzwerk Ports aufgelistet werden. 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. 7] wird die *Paket View* (Pakete mit schneiden) ausgewählt und man klickt auf Start. Während des Praktikums wurden weitere Versionen des Programms veröffentlicht und mittlerweile beinhaltet die kostenlose Version auch einen *Data View*, der den Datenfluss in getrennten Fenstern nach eingehenden und ausgehenden Daten anzeigt.

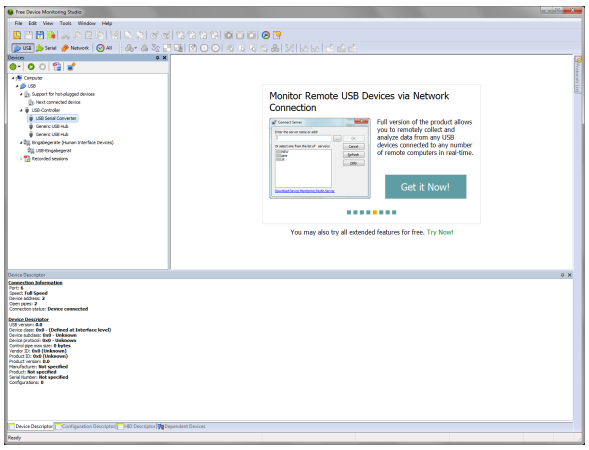

Abb. 6: Hauptfenster des Free Device Monitoring Studios

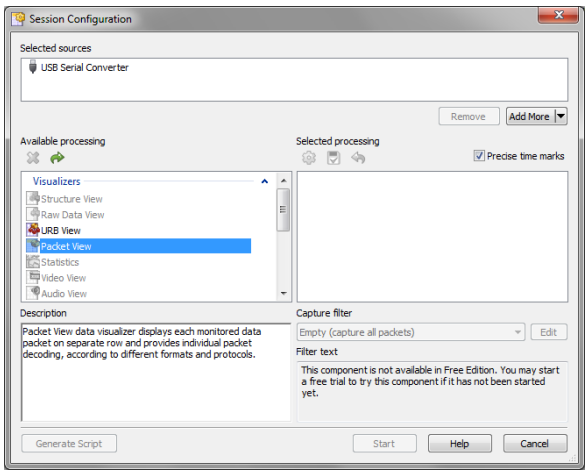

Abb. 7: *Session Configuration* Fenster zum Einstellen der verfügbaren Auslese-prozesse

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

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

Für diesen Test wurde das NextGen X-CTU Tools mit der Version 6.3.2 und Build ID: 20160819-3 verwendet.

Im Auswertungsfenster [Abb. 8] befinden sich im oberen Tab die Paketmitschnitte. Wird auf eines der Pakete geklickt, dann erscheint im unteren Tab der Inhalt, sowohl Hexadezimal, die ASCII-Zeichen, als auch die Anzahl an Bytes, die gesendet wurden. Um die Richtung zu identifizieren sind ausgehende Pakete (DOWN) mit blau gekennzeichnet und eingehende (UP) mit rot.

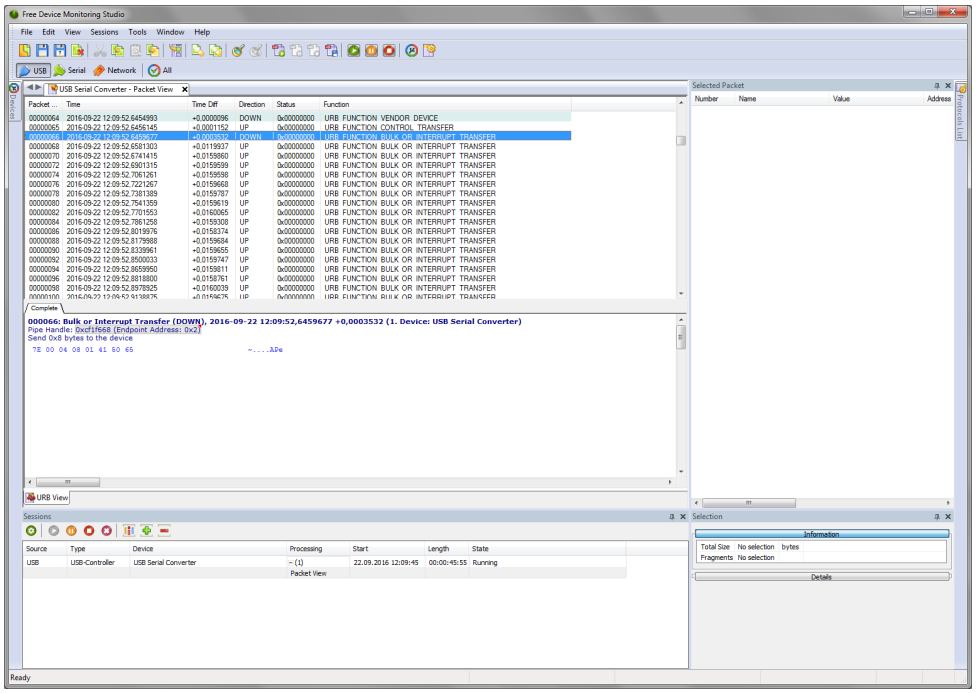

Abb. 8: Das Auswertungsfenster

5.4 Die Auswertung

Die XBee Kommandos können direkt über die Konsole, als auch durch andere Konsolenprogramme, wie etwa HTerm oder PuTTY gesendet werden. Dabei können die Nutzer die AT Kommandos, oder die mächtigeren API Frames benutzen.

Da im *Transparentmodus* standardmäßig der Transceiver Empfang eingestellt ist, müssen *drei Plus Zeichen* an das Modul gesendet werden um in den *Command Mode* zu gelangen.
Die API Frames bestehen hingegen aus Hexadezimal Zahlen und beginnen stets mit dem Standard Delimiter Wert 0x7E. [15]

Um den ersten Kontakt mit den Modulen aufzubauen, sendet das Next Gen X-CTU eine solche API Anfrage. Dadurch wird überprüft, in welchem Status sich das Modul befindet, die Legacy Edition tut dies nicht. Hier muss zunächst ausgewählt werden, in welchem Modus die Abfrage gestartet werden soll. Weitere Abfragen erfolgen dann mit AT Kommandos oder API Frames, je nach dem in welchen Status sich das Modul befindet.

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

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	Frame ID
2	41 50	AT Kommandoinhalt 'AP', kann n Byte groß sein (Frame Typ abhängig)
1	65	Checksumme 65(Hex) = 101(Dez)

Zwischen dem Aussenden des API Befehls und dem Senden der drei Pluszeichen vergehen konstant in jeden Test 1.7 ms. Das XBee Modul sendet in dieser Zeit 107 Pakete mit dem Code 11 60 (Device Control 1 und ' ` ').

Die Zeit, die zwischen dem Senden der drei Pluszeichen und dem Empfangen des OKs vergeht, beträgt 1 ms. Es werden dazwischen ca. 64 Pakete versendet.

Nach dieser Bestätigung startet die Abfrage des Moduls (in AT Kommandos):

	Befehlsbeschreibung	Befehl	Empfangen
1	API Aktiv	ATAP	0<CR> (aus)
2	Hardwareversion	ATHV	1743<CR>
3	Firmware Version	ATVR	10EF<CR>
4	SN-Nr. oberen 32-bit	ATSH	13A200<CR>
5	SN-Nr. unteren 32-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) [16]	AT%C	1<CR>
9	Koordinator Aktiv	ATCE	0<CR> (End Device)
10	Sleep Status (XBee-PRO 900 DigiMesh) [17]	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 mit den AT Befehlen, welche mit dem read-Befehl gelesen werden, richten sich nach der spezifischen Firmware. Das heißt, liest das Modul die Firmwareversion 10ef beim Verbinden, so ruft er nur die spezifischen Werte für die allgemeinen Einstellungen, bei 16e6, die der Analog IO-Adapter Einstellungen usw. ab. Welche Befehle ausgelesen werden, können der Tabelle im Abschnitt Erfassung, auf Zeile 22 bis 23: *X-CTU Default Settings (XBEE 802.15.4)* im Stat-auswerung_xbee-befehle.xlsx Dokument entnommen werden.

Die Zeit zwischen den einzelnen Anfragen, ist in mehreren Testverläufen unterschiedlich, im Schnitt liegt sie jedoch bei etwa 0.050 121 ms.

[16] Digi Customer Release Notes - 93009373 G1
[17] Digi Customer Release Notes - 93009377 B1

Um zu überprüfen, ob dieser Vorgang auch bei anderen Modultypen gleich bleibt, wurde noch ein weiterer Test mit einem XBee S2 ZigBee Modul durchgeführt. Bei diesem ergibt sich eine etwas andere Tabelle (siehe Seite 15).

Vergleicht man die Tabellen, wird erkenntlich, dass das Programm die Abfrage anpasst, nach dem die entsprechende Firmware und eventuell auch die Hardwareversion ausgelesen sind.

In der Tabelle für das ZigBee Modul wird zudem erkenntlich, dass das X-CTU Tool zunächst die Netzwerkeinstellungen überprüft und ggf. modifiziert (Zeile 11–18).

Wenn die zu realisierende Implementation mit dem X-CTU kompatibel sein soll, muss dem Programm ein XBee Modul vorgetäuscht werden, dafür wird mit hoher Wahrscheinlichkeit die Firmware Version und möglicherweise auch die Hardwareversion benötigt.

	Befehlsbeschreibung	Befehl	Empfangen
1	API Aktiv	ATAP	ERROR<CR>
2	Hardwareversion	ATHV	194A<CR>
3	Firmware Version	ATVR	22A7<CR>
4	Seriennummer oberen 32-bit	ATSH	13A200<CR>
5	Seriennummer unteren 32-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)[18]	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>

[18] Digi Customer Release Notes - 93009373 G1

6 Statistik der verbreitetsten AT Kommandos

Die Anordnung der Befehle in den Auswertungstabellen, aus der Datei *Stat-auswerung_xbee-befehle.xlsx*[19], entspricht der Kommandotabelle des Digi User Guides *ZigBee RF Modules*. Diese Tabelle besitzt eine andere Befehlsaufteilung als das X-CTU Programm, zudem sind im Manual nicht alle Befehle gelistet, daher wurde die Auswertungstabelle entsprechend angepasst und erweitert.

Da die Implementation, genau wie der Großteil der XBee Module, auf dem IEEE 802.15.4 Protokoll aufbauen sollen, wurden ebenfalls die fehlenden Kommandos der S1 802.15.4 Module mit aufgenommen. Es empfiehlt sich daher, zum Nachschlagen der Kommandoreferenzen das entsprechende Digi User Guide, XBee / XBee-PRO S1 802.15.4 (Legacy) mit hinzuzunehmen.

Wird die Menge aller Kommandos aus den beiden Modulvarianten und den X-CTU abgebildet, so umfasst diese insgesamt 119 AT Kommandos, die in 14 Kategorien aufgeteilt sind. Dies sind die Addressing Commands, Networking Commands, Security, RF Interfacing Commands, Serial Interfacing (I/O) Commands, I/O Commands, Diagnostic Commands, AT Command Option, Sleep Commands, Execution Commands, 802.15.4 related Commands, IO Line Passing und den X-CTU Kommandos, welche beim Einlesen der Module zusätzlich abgefragt werden.

Aufgrund der resultierenden Tabellengröße sind diese Tabellen nicht in diesen Dokument aufgeführt.

Tabelle 1: Erfassung, listet die einzelnen Tutorials und die Häufigkeit, welches Kommando wie oft genutzt wird auf. Dabei sind die Befehlsbereiche farbig markiert, um sie schneller den einzelnen Abschnitten in der *Command Reference Table* des User Guides 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 PAN ID, die Zieladresse in Destination High und Low, sowie die 16-bit Adresse festlegen muss.

Tabelle 2: Ergebnisse, beinhaltet die statistischen Summen der einzelnen Befehle (keine Prozentwerte). Am häufigsten wird das 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.

[19] Auswertungstabellen: Stat-auswerung_xbee-befehle.xlsx (zuletzt aktualisiert am 16.02.2017)

7 XBee Robustheit Analyse

Zum Testen der Robustheit werden verschiedene Befehle eingegeben und deren Rückgabelwert kontrolliert. Auch hier wird zum Protokollieren das *Tool Free Device Monitoring Studio* verwendet und 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 IDs
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

7.1 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 einem 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

7.2 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 Standardwert 0

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

T015: Eingabe von einem 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.

7.3 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

7.4 T040 : (ID) Verschiedene PAN IDs

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 einem Wert, der außerhalb des Wertebereichs liegt.
Ergebnis: ERROR

7.5 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 einem Wert, der außerhalb des Wertebereichs liegt.
Ergebnis: ERROR

7.6 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

7.7 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

7.8 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.

7.9 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 einem String mit Sonderzeichen.
Ergebnis: OK, wird problemlos gespeichert, auch Sonderzeichen wie @, € und µ

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

7.10 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

7.11 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 einen ungültigen Wert einzuspeichern.
Ergebnis: ERROR

7.12 T120 : Falsche Eingabe der AT Kommandos

Damit die AT Kommandos Problemlos funktionieren, muss die Reihenfolge der Eingabe beachtet werden. In diesen Versuch wird diese Abfolge absichtlich missachtet.

T121: Eingabe von AP (Befehl, der zum Aufwecken des Moduls geschickt wird, wenn man sich mit diesem 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 Befehls, der Buchstabenreihenfolge gemischten ist: DJET1, AETE1
Ergebnis: keine Antwort

7.13 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 Standardwerte zu setzen. Zu dem musste das Modul nach dem Starten des X-CTU Programms zurückgesetzt werden. Daraus ist zu schlussfolgern, dass einige nicht erlaubte Werte gespeichert werden 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. Außerdem reagieren die Module zum Teil nicht mehr auf Eingaben und die Entschlüsselung ist fehlerhaft.

8 RF Paket Sniffing-Analyse

8.1 Vorbereitung

Benötigte Materialien:

- 2 XBee S2 Module (Sensor [Sender], Coordinator [Empfänger])
- 3 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)

Hinweis: Die ersten Tests wurden nur mit zwei S1 Modulen durchgeführt, um einen groben Überblick zu bekommen, wie die XBee Module miteinander kommunizieren. Im späteren Verlauf wurden umfangreichere Testläufe angefertigt, um die Details näher zu betrachten. (vgl. 8.4)

Ebenfalls wurden im ersten Test einige Durchläufe mit den S2 ZigBee Modulen durchgeführt Ein S2 Modul kann zwar nicht mit einem S1 802.15.4 Modul kommunizieren, aber um die beiden Log-Dateien besser vergleichen zu können und Rückschlüsse auf die später folgende Prototypenplanung zu schließen, wurden für beide Geräte, so weit wie möglich, die selbe Konfiguration übernommen. Die Auswertung der S2 Module kann in Abschnitt 8.5 nachgelesen werden.

Konfiguration: Die XBee Module müssen so konfiguriert werden, dass sie miteinander kommunizieren, dabei kann man wie im Video Tutorial *XBee Basics – Lesson 2 – Simple Chat Program Between Two XBees* im Transparentmodus[20] von *tunnelsup* vorgehen.

Gerät A (Koordinator)

CH	0x17
DH	0x13A200
DL	0x414640C0 (Modell S1) 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) 0x409029CF (Modell S2)
ID	0x4790
MY	0x110 (nur bei Modell S1 Möglich)
CE	0x0
NI	End Device

Es ist zu beachten, dass die Zieladressen (DH/DL), die 16-bit Quelladresse (MY), sowie die Knotenidentifizierung (NI) optional sind und in diesem 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 den Parametern

```
1 -d <COM Port> -f ZB-USB_BitCatcher_FW_ 0x27030500.bin
```

wird die Firmware auf den USB geflasht.

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

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* und *DE Sniffer*. Hier wird der entsprechende COM Port ausgewählt und die Baudrate auf 38 400 Bd 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 lila Symbol.

Als Nächstes benötigt man die richtige Kanalnummer, diese kann man am schnellsten ermitteln, wenn man via AT Kommando, oder per X-CTU den Befehl CH aufruft, im Testfall ist dieser ebenfalls 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 der Channel-Bitmask kann man ebenfalls 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 60 000 ms (1 min) gesendet.

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

8.2 Sendelogik der XBee Module

Bei großen Datenmengen, jedes Byte einzeln zu schicken, ist genau so wenig sinnvoll, wie wenn man für einen Großeinkauf, für jedes einzelne Produkt immer wieder in denselben Laden geht und dieses zuerst nach Hause bringt, bevor man das nächste kauft. Im Falle der RF Funkmodule würden bei dieser Logik nicht nur gewaltig viele Verwaltungsdaten entstehen, sondern es würde auch bedeuten einen erhöhten Energieverbrauch, so wie längere Übertragungsraten zu erzeugen. Daher warten die Module einen kleinen Moment bis sie das Paket packen und versenden.

Digi gibt hierbei den Nutzer ebenfalls die Möglichkeit diesen Faktor zu beeinflussen. Mit dem AT Kommando RO (Packetization Timeout), welcher unter den seriellen Einstellungen zu finden ist, stellt man den Faktor ein, welchen das Modul warten soll bevor es das Paket versendet. Digi beschreibt das Kommando dabei wörtlich: „RO setzt/liest die Anzahl der ‚character times' der Zwischenzeichenverzögerung, bevor die Übertragung beginnt. [. . .]" Dabei besteht die *Zwischenzeichenverzögerung* aus dem Start-Bit, den zu übertragenen Zeichen und dem Stopp-Bit. *Character times* ist demnach die Anzahl der *Zwischenzeichenverzögerungen*, wie lange das Modul nach dem letzten Zeichen warten soll, bis es das Paket versendet wird. Als mögliche Spanne kann man hexadezimale Werte zwischen 0x0 und 0xFF (255 Dezimal) angeben, der Standardwert ist 0x3. Also wartet das Modul dreimal so lange, wie ein Zeichen inklusive Start und Stopp-Bit zur Übertrag bräuchte. Eine genaue Zeitangabe gibt Digi dabei nicht an, es ist aber davon auszugehen, dass diese von der Baudrate abhängig ist.

Beispiel: Es soll der String *Hallo XBee* (ohne Carriage Return) versendet werden. Des Weiteren wird angenommen, dass es keine Verzögerung durch menschliche Reaktionszeiten gibt, sowie dass die Verarbeitung eines Paketes bis es versendet wurde 3 ms benötigt. Ebenfalls sind folgende Werte gegeben:

Baudrate: 38400 Baud
8 Daten-bits + 1 Start-bit + 1 Stopp-bit = 10-bit
Standardwert für ATRO: 0x3
Stringlänge: 10 Bytes

Daraus resultiert folgende Übertragungsrate:

$$\frac{38\,400\,\text{Bd}}{10\text{-bit}} = 3840\,\text{Bd/bit} \rightarrow 3840\,\text{Byte/s}$$

In der Rechnung (.1) wird zunächst die Sendung als Gesamtpaket behandelt, während in Rechnung (.2) jedes Byte einzeln versendet wird. Anhand der Ergebnisse wird erkenntlich, dass der String als Gesamtpaket 5,11 mal schneller verschickt ist. Dazu kommt noch, dass der Sender nur ein ACK abwarten muss und keine zehn. Auch das Empfängermodul hat eine kürzere Verarbeitungszeit, denn es muss ebenfalls nur ein Paket auspacken und verarbeiten.

$$\begin{aligned} 3 \times \frac{1\,\text{Byte}}{3840\,\text{Byte/s}} &= 0.781\,\text{ms} \\ 10 \times \frac{1\,\text{Byte}}{3840\,\text{Byte/s}} &= 2.604\,\text{ms} \\ 2.604\,\text{ms} + 0.781\,\text{ms} &= 3.385\,\text{ms} \\ 3.385\,\text{ms} + 3\,\text{ms} &= 6.385\,\text{ms} \end{aligned} \quad (.1)$$

$$\begin{aligned} \frac{1\,\text{Byte}}{3840\,\text{Byte/s}} &= 26.04\,\mu\text{s} \\ 10 * (26.04\,\mu\text{s} + 3\,\text{ms}) &= 32.604\,\text{ms} \end{aligned} \quad (.2)$$

8.3 Auswertung des XBee S1 Logs (2 Module)

Der Datenaustausch zwischen den XBee Modulen kann über verschiedene Wege erfolgen, als einfache Textnachricht im Transparenten Modus, oder als ein bestimmter Frame Typ im API Modus. Es wird vermutet, dass die übertragenen Frames nahezu identisch sind und es keinen Unterschied zwischen Transparent und API Modus gibt.

Auswertungsbeispiel Zeile 169 aus der BitCatcher Log Datei log_xbS1-log.dcf.

Der BitCatcher ist normalerweise für ZigBee Module ausgelegt, daher kann es bei dem Sniffen dazu kommen, dass Frames mit unterschiedlichen Typenbezeichnungen im Sniffer Tool ausgegeben werden.
Die *APS Frames* enthalten daher ebenfalls die Daten, jedoch beginnen die Nutzdaten schon im nwkHeader, der nur bei ZigBee Modulen vorhanden ist. Je nach Nutzdatenlänge wird nach dem *apsHeader* noch ein *Data* Feld angezeigt , welches 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)

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>	}

Wie eingangs erwähnt, werden im BitCatcher sowohl unbenannte, als auch Netzwerk Fenster angezeigt, diese enthalten ebenfalls die Nutzdaten und werden nur vom BitCatcher/Sniffer fehlerhaft deklariert.

Zudem befinden sich im Datenfeld zwei Byte, die zunächst unklar erscheinen. Nach einigen Testläufen wird jedoch klar, dass diese beiden Bytes den MaxStream Header von Digi darstellen.

MaxStream Header

Anz. Byte	Hex-Wert	Bedeutung
1	28	frame counter im relativen Abstand zu seqNo*
1	00	Option [00] kein Request [04] Remote AT Command [05] Remote AT Request

*die *Distanz* ist von Modul zu Modul und nach jedem Neustart unterschiedlich, jedoch innerhalb der Lebensspanne des Hauptprozesses immer gleich.

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 im Dokument nicht weiter dokumentiert, da diese während Folgetests nicht reproduzierbar waren.

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). Der Versuch mit API Frames wird hier nicht näher dokumentiert, da dieser identisch zu den AT Frames ist. Mit den verwendeten Testdaten würden die API Frames 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

Hex-Wert	Beschreibung
7E	Delimiter
00 16	Länge
00	API Typ
01	Frame ID
00 13 A2 00 40 E9 7A E3	Zieladresse
00	Optionen*
48 61 6C 6C 6F 20 58 42 65 65 0D	Textinhalt
42	Checksumme

(16-bit Adresse):

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

Hex-Wert	Beschreibung
7E	Delimiter
00 10	Länge
01	API Typ
01	Frame ID
01 01	Zieladresse
00	Optionen*
48 61 6C 6C 6F 20 58 42 65 65 0D	Textinhalt
7A	Checksumme

* Optionen: [00] ACK / [01] kein ACK [04] mit Broadcast PANID

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 über die Daten (alle Byte nach der Länge) ergeben das Ergebnis 0xFF.

8.4 Auswertung des XBee S1 Logs (3 Module)

In einem weiteren Testlauf sollen drei XBees gleichzeitig angesteuert werden. Im ersten Versuch bekommen alle Module die gleiche Firmware Version und die gleiche Konfiguration für PAN ID und Channel Nummer. Im Zweiten, unterschiedliche Kanalnummern, im Dritten diverse PAN ID's und im Vierten eine Kombination aus der zweiten und dritten Variation. Alle vier Test Variationen werden einmal ohne und einmal mit Security-Optionen durchgeführt. Ziel ist es herauszufinden, in welcher Konstellation die XBee Module den MAC Header zusammensetzten und Nachrichten empfangen.

Als Module kommen zum Einsatz (SL ID):

- 414640C2 (als Coordinator) Bezeichnung: XBee0 MY: BEE0
- 414640C0 (als End Device) Bezeichnung: XBee1 MY: BEE1
- 41464103 (als End Device) Bezeichnung: XBee2 MY: BEE2

Variante 1 – CH: 17, PAN ID: DE16

Die Fallnotation

Aufgrund eines größeren Analyseaufwandes wird für die Übersichtlichkeit und der genaueren Referenzierung eine Fallnotation eingeführt. Sie besteht aus dem Kürzel AL (Analyselauf) und einer vierstelligen Nummer. Die 1000er, 100er und 10er Stelle stellt die Test-Durchlauf-Nummer dar, die 1er die Wiederholungsnummer, falls aufgrund fehlender Analyseinformationen ein Testlauf wiederholt wird. Sollten mehr als 9 Wiederholungen benötigt werden, so ist ein neuer Analyselauf zu definieren und eine Referenz zum ursprünglichen Test einzufügen.

Beispiel: AL0256

AL	Fallbeschreibung
025	Fallnummer (dreistellig)
6	Wiederholung (einstellig)

Nach der ursprünglichen Planung war hier die Auflistung der Tests angedacht, da sich aber rasch herausstellte, dass viele der Durchläufe fast identisch waren, fasse ich im Folgenden die allgemeinen Erkenntnisse zusammen und liste im späteren Verlauf nur eine Auswahl an Durchläufen auf.

Der MAC und MaxStream Header

Wichtig für die Tests ist der integrierte MAC Mode (MM) der XBee Module.
Es gibt vier verschiedene Modi:

1. 802.15.4 + Max Steam Header with ACKS [0]
2. 802.15.4 no ACKs [1]
3. 802.15.4 with ACKs [2]
4. 802.15.4 + Max Steam Header no ACKS [3]

Alle Module müssen sich ausnahmslos im selben MAC Mode befinden, um Nachrichten von dem jeweiligen anderen Modul zu empfangen. Je nach Eingabemodus, Transparent oder API, ist die Unterscheidung im eigentlichen MAC Header jedoch nur minimal im Frame Control Bit Feld erkennbar.
Der entscheidende Abschnitt ist der MaxStream Header, welcher mit zum Frame Payload gehört, aber wichtige Informationen über die Funktionalität übermitteln kann.

Dieser besteht aus zwei Bytes, einem zusätzlichen Zähler-Byte und einem Optionen-Byte.

Der Abstand vom Zähler-Byte zur Frame Sequenz Nummer wird zum Start des Moduls zufällig gewählt. Mit jedem Trennen und Verbinden der Stromzufuhr wird ein neuer Abstand festgelegt, welcher solange das Modul mit Strom versorgt wird, konstant bleibt. Durch das Auswürfeln, haben die Module meistens einen unterschiedlichen Abstand. Dopplungen sollten nur bei extrem vielen Modulen, oder nur mit einer geringen Wahrscheinlichkeit bei langer Laufzeit von zwei Modulen auftreten. Mit diesem Frame Counter hat man eine zusätzliche Kontrolle, ob ein Frame doppelt angekommen ist. Applikationen können diese zusätzliche Information nutzen um ihre Daten zu validieren.

Das Optionen-Byte ist von hoher Relevanz, wenn Daten im API Modus versendet werden, denn mit diesem Feld wird ermittelt, ob es sich um einen einfachen *TX Transmit (0x00)*, um ein *Remote AT Command (0x04)*, oder um ein *Remote AT Command Response (0x05)* handelt. Fehlt dieses Optionen-Byte, werden alle Frames als TX Transmit interpretiert und auf dem Empfängermodul mit ein RX Receive an den Nutzer ausgegeben.

In den Einstellungen ohne dem Max Stream Header, ist das *Frame Control Bit Feld* identisch zu den Einstellungen mit dem Max Stream Header, jedoch fehlen im Payload die zwei zusätzlichen Byte, was dazu führen kann, dass im API Modus die Anweisungen nicht ausgeführt werden.

Die Unterschiede zeigen folgendes Beispiel Frame. In diesem werden die besprochen Bytes verändert und die CRC Checksumme, sowie der Frame Zähler belassen, um die Veränderungen visuell besser hervorzuheben.

MM0: 61 8C 1B 16 DE C0 40 46 41 00 A2 13 00 0E BE 0B 00 44 46 2A
MM2: 61 8C 1B 16 DE C0 40 46 41 00 A2 13 00 0E BE 44 46 2A
MM1: 41 8C 1B 16 DE C0 40 46 41 00 A2 13 00 0E BE 44 46 2A
MM3: 41 8C 1B 16 DE C0 40 46 41 00 A2 13 00 0E BE 0B 00 44 46 2A

Das Frame Control Bit Feld

Ist ein Teil des IEEE 802.15.4 Standards, um in einen übertragenen Datensatz die Position und Zugehörigkeit der einzelnen Bytes zu bestimmen und damit Auskunft zu geben, wie der Frame zusammengebaut ist. Das Frame Control Bit Feld der XBee Module gibt Auskunft, an welche Adresse ein Paket gesendet wurde, woher es kommt, welcher Adresstyp verwendet wurde (16 oder 64-bit), ob ein ACK erwünscht ist, den Security Status und ob es sich um eine Broadcast Nachricht handelt. Alle anderen Felder bleiben bei den nachfolgenden Tests auf Standartwerten, und wurden nicht weiter verändert.

Testfälle	Einstellungen	Frame Control
AL0010 – AL0060 AL0010 – AL0060 AL0130 – AL0180 AL0250 – AL0300 AL0370 – AL0420	MAC Mode 0 und 2, Daten werden mit einen ACK Request geschickt	0x8C61 0x8861 0xCC61* 0xC861*
AL0070 – AL0120 AL0190 – AL0240 AL0310 – AL0360 AL0430 – AL0480	MAC Mode 1 und 3, Daten werden ohne einen ACK Request geschickt	0x8C41 0x8841 0xCC41* 0xC841*
AL0730	Mit der zusätzlichen Änderung des Security Status, werden von den XBee Modulen stets die 64-bit Adressen verwendet und nur wenn die entsprechende Adresse nicht zur Verfügung steht, wird auf die 16-bit Adresse zurückgegriffen (vgl. 8.4).	0xCC69 0xC869 0xCC49 0xC849 0xCC29 0xC829 0xCC09 0xC809

* (wird von XBee nicht genutzt)

Nicht in den Tests auftauchende Varianten des Frame Control Feldes sind:

- 0xCC21
- 0x8821
- 0x8C21
- 0xC821
- 0xCC01
- 0x8C01
- 0xC801
- 0x8801

Daraus resultiert folgende Bit Feld Verteilung[21]:

Bit	0 – 2	3	4	5	6	7	8	9	10 – 11	12 – 13	14 – 15
Zweck	Frame Type	Security Enabled	Frame Pending	AR (ACK)	PAN ID Compression	Reserved	Seq. Num. Suppression	IE Present	Dest. Addr. Mode	Frame Version	Src. Add. mode
8C 61	001	0	0	1	1	0	0	0	11	00	10
88 61	001	0	0	1	1	0	0	0	10	00	10
CC 61	001	0	0	1	1	0	0	0	11	00	11
C8 61	001	0	0	1	1	0	0	0	10	00	11
8C 41	001	0	0	0	1	0	0	0	11	00	10
88 41	001	0	0	0	1	0	0	0	10	00	10
CC 41	001	0	0	0	1	0	0	0	11	00	11
C8 41	001	0	0	0	1	0	0	0	10	00	11
8C 21	001	0	0	1	0	0	0	0	11	00	10
88 21	001	0	0	1	0	0	0	0	10	00	10
CC 21	001	0	0	1	0	0	0	0	11	00	11
C8 21	001	0	0	1	0	0	0	0	10	00	11
8C 01	001	0	0	0	0	0	0	0	11	00	10
88 01	001	0	0	0	0	0	0	0	10	00	10
CC 01	001	0	0	0	0	0	0	0	11	00	11
C8 01	001	0	0	0	0	0	0	0	10	00	11

Mit AES Verschlüsselung:

Bit	0 – 2	3	4	5	6	7	8	9	10 – 11	12 – 13	14 – 15
CC 69	001	1	0	1	1	0	0	0	11	00	11
C8 69	001	1	0	1	1	0	0	0	10	00	11
CC 49	001	1	0	0	1	0	0	0	11	00	11
C8 49	001	1	0	0	1	0	0	0	10	00	11
CC 29	001	1	0	1	0	0	0	0	11	00	11
C8 29	001	1	0	1	0	0	0	0	10	00	11
CC 09	001	1	0	0	0	0	0	0	11	00	11
C8 09	001	1	0	0	0	0	0	0	10	00	11

[21] vgl. IEEE Standard for Low-Rate Wireless Networks, 802.15.4-2015.pdf, Seite 151

Analyse Fälle AL0010 – AL0240

In den ersten Testfällen wird keine AES Verschlüsselung eingesetzt und Nachrichten im AT Modus übermittelt.
Alle Paketzeilen sind in der Wireshark Log Datei 2016-12-20_xb-s1_log.pcapng gelistet.

AL0010 und AL0190

Konfiguration:

- DH und DL gegeben
  - 0x0013A200 und 0x414640C0
- Nachrichten Uhrsprung: Koordinator
- MY (16-bit Adr.) nicht gegeben
- MAC Mode = 0 und MAC Mode = 3

Ergebnis:
Die Nachricht wird nur von Punkt zu Punkt gesendet, alle nicht adressierten Geräte registrieren die Nachricht nicht. Es ist dabei auch irrelevant, ob die Nachricht von Koordinator zu Koordinator gesendet wird, oder ob der reguläre Fall auftritt: Koordinator zum Endgerät. Zudem wird die Nachricht Zeichenweise versendet, daraus ergibt sich, dass zwei Pakete analysiert werden müssen. Diese unterscheiden sich jedoch nur im Dateninhalt und in den Frame Counter, daher wird nur die Nachricht mit dem ‚D' aufgelistet.

MM0 (Paket Nr. 17) { 61 8C 1B 16 DE C0 40 46 41 00 A2 13 00 00 00 0B 00 44 46 2A }
Zum MM3 ändert sich nur der Frame Control Wert (grün markiert) und die variablen Bytes für die Paketzähler und CRC-Checksummme.

Byte	Hex-Wert	Bedeutung
2	8C 61	(MM0) frameCtl
1	1B	seqNo
2	DE 16	dstPanId (ID)
8	00 13 A2 00 41 46 41 C0	dstExtAddr (DH/DL)
2	00 00	srcShortAddr (MY)
1	0B	Frame Counter d = 0x10 (Max Stream Header)
1	00	(no Request) Option (Max Stream Header)
1	44	(D) Message
2	2A 46	CRC

AL0020 und AL0200

Konfiguration:

- DH und DL gegeben
  - 0x0013A200 und 0x414640C0
- Nachrichten Uhrsprung: Endgerät
- MY (16-bit Adr.) nicht gegeben
- MAC Mode = 0 und MAC Mode = 3

Ergebnis:
Die Nachricht wird nur von Punkt zu Punkt gesendet, alle nicht adressierten Geräte registrieren die Nachricht nicht. Es ist dabei auch irrelevant, ob die Nachricht von Endgerät zu Endgerät gesendet wird oder ob der reguläre Fall auftritt: Endgerät zum Koordinator. Zudem wird die Nachricht Zeichenweise versendet, daraus ergibt sich, dass zwei Pakete analysiert werden müssen. Diese unterscheiden sich jedoch nur im Dateninhalt und in den Frame Counter, daher wird nur die Nachricht mit dem ‚D' aufgelistet.

MM0 (Paket Nr. 31) { 61 8C 12 16 DE C0 40 46 41 00 A2 13 00 00 00 06 00 44 ED 0B }
Zum MM3 ändert sich nur der Frame Control Wert (grün markiert) und die variablen Bytes für die Paketzähler und CRC-Checksummme.

Byte	Hex-Wert	Bedeutung
2	8C 61	(MM0) frameCtl
1	12	seqNo
2	DE 16	dstPanId (ID)
8	00 13 A2 00 41 46 41 C0	dstExtAddr (DH/DL)
2	00 00	srcShortAddr (MY)
1	06	Frame Counter d = 0x6 (Max Stream Header)
1	00	(no Request) Option (Max Stream Header)
1	44	(D) Message
2	0B ED	CRC

In diesen ersten beiden Testläufen sieht man, dass die 16-bit Quelladresse (MY) immer mit angegeben wird, auch wenn der Wert 0x0 entspricht. Um diese Bytes besser zuordnen zu können werden die Adressen in den Folgetests mit übergeben.

AL0030 und AL0210

Konfiguration:

- DH = 0 und DL < 0xFFFE
  - 0xBEE1
- Nachrichten Uhrsprung: Koordinator
- MAC Mode = 0 und MAC Mode = 3

Ergebnis:
Die Nachricht wird nur von Punkt zu Punkt gesendet, alle nicht adressierten Geräte registrieren die Nachricht nicht. Es ist dabei auch irrelevant, ob die Nachricht von Koordinator zu Koordinator gesendet wird, oder ob der reguläre Fall auftritt: Koordinator zum Endgerät. Zudem wird die Nachricht Zeichenweise versendet, daraus ergibt sich, dass zwei Pakete analysiert werden müssen. Diese unterscheiden sich jedoch nur im Dateninhalt und in den Frame Counter, daher wird nur die Nachricht mit dem ‚D' aufgelistet.

MM0 (Paket Nr. 35) { 61 88 1D 16 DE E1 BE E0 BE 0D 00 44 E3 37 }
Zum MM3 ändert sich nur der Frame Control Wert (grün markiert) und die variablen Bytes für die Paketzähler und CRC-Checksummme.

Byte	Hex-Wert	Bedeutung
2	8C 61	(MM0) frameCtl
1	1D	seqNo
2	DE 16	dstPanId (ID)
2	BE E1	dstExtAddr (DL)
2	BE E0	srcShortAddr (MY)
1	06	Frame Counter d = 0x17 (Max Stream Header)
1	00	(no Request) Option (Max Stream Header)
1	44	(D) Message
2	37 E3	CRC

AL0040 und AL0220

Konfiguration:

- DH = 0 und DL < 0xFFFE
  - 0xBEE1
- Nachrichten Uhrsprung: Endgerät
- MAC Mode = 0 und MAC Mode = 3

Ergebnis:
Die Nachricht wird nur von Punkt zu Punkt gesendet, alle nicht adressierten Geräte registrieren die Nachricht nicht. Es ist dabei auch irrelevant, ob die Nachricht von Endgerät zu Endgerät gesendet wird oder ob der reguläre Fall auftritt: Endgerät zum Koordinator. Zudem wird die Nachricht Zeichenweise versendet, daraus ergibt sich, dass zwei Pakete analysiert werden müssen. Diese unterscheiden sich jedoch nur im Dateninhalt und in den Frame Counter, daher wird nur die Nachricht mit dem ‚D' aufgelistet.

MM0 (Paket Nr. 39) { 61 88 14 16 DE E1 BE E2 BE 08 00 44 D8 5D }
Zum MM3 ändert sich nur der Frame Control Wert (grün markiert) und die variablen Bytes für die Paketzähler und CRC-Checksummme.

Byte	Hex-Wert	Bedeutung
2	8C 61	(MM0) frameCtl
1	14	seqNo
2	DE 16	dstPanId (ID)
2	BE E1	dstExtAddr (DL)
2	BE E2	srcShortAddr (MY)
1	08	Frame Counter d = 0xC (Max Stream Header)
1	00	(no Request) Option (Max Stream Header)
1	44	(D) Message
2	5D D8	CRC

AL0050 und AL0230

Konfiguration:

- DH und DL 0
- Nachrichten Uhrsprung: Koordinator
- MAC Mode = 0 und MAC Mode = 3

Ergebnis:
Keines der anderen Geräte empfängt die Nachricht.

MM0 (Paket Nr. 116) { 61 88 55 16 DE 00 00 E0 BE 04 00 44 32 D5 }
Zum MM3 ändert sich nur der Frame Control Wert (grün markiert) und die variablen Bytes für die Paketzähler und CRC-Checksummme.

Byte	Hex-Wert	Bedeutung
2	8C 61	(MM0) frameCtl
1	55	seqNo
2	DE 16	dstPanId (ID)
2	00 00	dstExtAddr (DL)
2	BE E0	srcShortAddr (MY)
1	04	Frame Counter d = 0x51 (Max Stream Header)
1	00	(no Request) Option (Max Stream Header)
1	44	(D) Message
2	D5 32	CRC

AL0051

Nach einer Modifizierung des Tests, indem die 16-bit Quelladresse (MY) bei allen Modulen ebenfalls auf 0x0 gesetzt wird, empfangen alle Geräte diese Nachricht. (Paket Nr. 129)

Außer, dass sich die Zeile *srcShortAddr* (MY) ändert, sind alle anderen statischen Tabellenwerte identisch, daher gibt es keine weitere Tabellenauflistung.

AL0060 und AL024

Konfiguration:

- DH und DL 0
- Nachrichten Uhrsprung: Endgerät
- MAC Mode = 0 und MAC Mode = 3

Ergebnis:
Keines der anderen Geräte empfängt die Nachricht.

> **MM0** (Paket Nr. 116) { 61 88 53 16 DE 00 00 E1 BE 04 00 44 A4 36 }
> Zum MM3 ändert sich nur der Frame Control Wert (grün markiert) und die variablen Bytes für die Paketzähler und CRC-Checksummme.

Byte	Hex-Wert	Bedeutung
2	8C 61	(MM0) frameCtl
1	53	seqNo
2	DE 16	dstPanId (ID)
2	00 00	dstExtAddr (DL)
2	BE E1	srcShortAddr (MY)
1	04	Frame Counter d = 0x4F (Max Stream Header)
1	00	(no Request) Option (Max Stream Header)
1	44	(D) Message
2	36 A4	CRC

AL0061

Identisches Resultat wie in *AL0051*.

Entsprechend wird aus den Testläufen AL0050/51 und AL0060/61 ersichtlich, dass es im AT Modus keine Broadcast Übertragung gibt, sobald die Module unterschiedliche 16-bit Adressen bekommen haben.

AL0070 und AL0130

Konfiguration:

- DH und DL gegeben
  - 0x0013A200 und 0x414640C0
- Nachrichten Uhrsprung: Koordinator
- MAC Mode = 1 und MAC Mode = 2

Ergebnis:
Die Nachricht wird nur von Punkt zu Punkt gesendet, alle nicht adressierten Geräte registrieren die Nachricht nicht. Es ist dabei auch irrelevant, ob die Nachricht von Koordinator zu Koordinator gesendet wird, oder ob der reguläre Fall auftritt: Koordinator zum Endgerät. Zudem wird die Nachricht Zeichenweise versendet, daraus ergibt sich, dass zwei Pakete analysiert werden müssen. Diese unterscheiden sich jedoch nur im Dateninhalt und in den Frame Counter, daher wird nur die Nachricht mit dem ‚D' aufgelistet.

MM1 (Paket Nr. 151) { 41 8C 5d 16 de c0 40 46 41 00 a2 13 00 E0 BE 44 F3 2B }
Zum MM2 ändert sich nur der Frame Control Wert (grün markiert) und die variablen Bytes für die Paketzähler und CRC-Checksummme.

Byte	Hex-Wert	Bedeutung
2	8C 41	(MM1) frameCtl
1	5D	seqNo
2	DE 16	dstPanId (ID)
8	00 13 A2 00 41 46 40 C0	dstExtAddr (DH/DL)
2	BE E0	srcShortAddr (MY)
1	44	(D) Message
2	2B F3	CRC

AL0080 und AL0140

Konfiguration:

- DH und DL gegeben
  - 0x0013A200 und 0x414640C2
- Nachrichten Uhrsprung: Endgerät
- MAC Mode = 1 und MAC Mode = 2

Ergebnis:
Die Nachricht wird nur von Punkt zu Punkt gesendet, alle nicht adressierten Geräte registrieren die Nachricht nicht. Es ist dabei auch irrelevant, ob die Nachricht von Endgerät zu Endgerät gesendet wird oder ob der reguläre Fall auftritt: Endgerät zum Koordinator. Zudem wird die Nachricht Zeichenweise versendet, daraus ergibt sich, dass zwei Pakete analysiert werden müssen. Diese unterscheiden sich jedoch nur im Dateninhalt und in den Frame Counter, daher wird nur die Nachricht mit dem ‚D' aufgelistet.

MM1 (Paket Nr. 153) { 41 8C 23 16 DE C0 40 46 41 00 A2 13 00 E2 BE 44 42 4F }
Zum MM2 ändert sich nur der Frame Control Wert (grün markiert) und die variablen Bytes für die Paketzähler und CRC-Checksummme.

Byte	Hex-Wert	Bedeutung
2	8C 41	(MM1) frameCtl
1	23	seqNo
2	DE 16	dstPanId (ID)
8	00 13 A2 00 41 46 40 C0	dstExtAddr (DH/DL)
2	BE E2	srcShortAddr (MY)
1	44	(D) Message
2	4F 42	CRC

AL0090 und AL0150

Konfiguration:

- DH = 0 und DL < 0xFFFE
  - 0xBEE1
- Nachrichten Uhrsprung: Koordinator
- MAC Mode = 1 und MAC Mode = 2

Ergebnis:
Die Nachricht wird nur von Punkt zu Punkt gesendet, alle nicht adressierten Geräte registrieren die Nachricht nicht. Es ist dabei auch irrelevant, ob die Nachricht von Koordinator zu Koordinator gesendet wird, oder ob der reguläre Fall auftritt: Koordinator zum Endgerät. Zudem wird die Nachricht Zeichenweise versendet, daraus ergibt sich, dass zwei Pakete analysiert werden müssen. Diese unterscheiden sich jedoch nur im Dateninhalt und in den Frame Counter, daher wird nur die Nachricht mit dem ‚D' aufgelistet.

MM1 (Paket Nr. 155) { 41 88 5f 16 DE E1 BE E0 BE 44 9C 9A }
Zum MM2 ändert sich nur der Frame Control Wert (grün markiert) und die variablen Bytes für die Paketzähler und CRC-Checksummme.

Byte	Hex-Wert	Bedeutung
2	8C 41	(MM1) frameCtl
1	1D	seqNo
2	DE 16	dstPanId (ID)
2	BE E1	dstExtAddr (DL)
2	BE E0	srcShortAddr (MY)
1	44	(D) Message
2	9A 9C	CRC

AL0100 und AL0160

Konfiguration:

- Security off
- AT Modus
- DH = 0 und DL < 0xFFFE
  - 0xBEE1
- Nachrichten Uhrsprung: Endgerät
- MAC Mode = 1 und MAC Mode = 2

Ergebnis:
Die Nachricht wird nur von Punkt zu Punkt gesendet, alle nicht adressierten Geräte registrieren die Nachricht nicht. Es ist dabei auch irrelevant, ob die Nachricht von Endgerät zu Endgerät gesendet wird oder ob der reguläre Fall auftritt: Endgerät zum Koordinator. Zudem wird die Nachricht Zeichenweise versendet, daraus ergibt sich, dass zwei Pakete analysiert werden müssen. Diese unterscheiden sich jedoch nur im Dateninhalt und in den Frame Counter, daher wird nur die Nachricht mit dem ‚D' aufgelistet.

MM1 (Paket Nr. 159) { 41 88 26 16 DE E1 BE E2 BE 44 5E 0B }
Zum MM2 ändert sich nur der Frame Control Wert (grün markiert) und die variablen Bytes für die Paketzähler und CRC-Checksummme.

Byte	Hex-Wert	Bedeutung
2	8C 41	(MM1) frameCtl
1	26	seqNo
2	DE 16	dstPanId (ID)
2	BE E1	dstExtAddr (DL)
2	BE E2	srcShortAddr (MY)
1	44	(D) Message
2	0B 5E	CRC

AL0110 und AL0170

Konfiguration:

- DH und DL 0
- Nachrichten Uhrsprung: Koordinator
- MAC Mode = 1 und MAC Mode = 2

Ergebnis:
Keines der anderen Geräte empfängt die Nachricht.

MM1 (Paket Nr. 167) { 41 88 62 16 DE 00 00 E0 BE 44 B5 22 }
Zum MM2 ändert sich nur der Frame Control Wert (grün markiert) und die variablen Bytes für die Paketzähler und CRC-Checksummme.

Byte	Hex-Wert	Bedeutung
2	8C 41	(MM0) frameCtl
1	62	seqNo
2	DE 16	dstPanId (ID)
2	00 00	dstExtAddr (DL)
2	BE E0	srcShortAddr (MY)
1	44	(D) Message
2	22 B5	CRC

AL0120 und AL0180

Konfiguration:

- DH und DL 0
- Nachrichten Uhrsprung: Endgerät
- MAC Mode = 1 und MAC Mode = 2

Ergebnis:
Keines der anderen Geräte empfängt die Nachricht.

MM1 (Paket Nr. 169) { 41 88 2A 16 DE 00 00 E2 BE 44 40 DF }
Zum MM2 ändert sich nur der Frame Control Wert (grün markiert) und die variablen Bytes für die Paketzähler und CRC-Checksummme.

Byte	Hex-Wert	Bedeutung
2	8C 41	(MM1) frameCtl
1	62	seqNo
2	DE 16	dstPanId (ID)
2	00 00	dstExtAddr (DL)
2	BE E2	srcShortAddr (MY)
1	44	(D) Message
2	40 DF	CRC

Analyse Fälle AL0250 – AL0480

Keine AES Verschlüsselung und Nachrichten werden im AP Modus übermittelt. Verwendeter Frame: 0x17 Remote AT Command. Alle Paketzeilen sind in der Wireshark Log Datei 2016-12-20_xb-s1_log.pcapng gelistet.

Inhaltlich werden alle Konfigurationen wie in den Analysedurchläufen *AL0010 – AL0240* vorgenommen. Da sich aber die Ergebnisse bis auf den Dateninhalt und den Variablen Bytes gleichen werden nur die unterschiedlichen Fälle und ihre Resultate dokumentiert.

Konfiguration:

- DH und DL gegeben
  - 0x0013A200 und 0x414640C0
- Nachrichten Uhrsprung: Koordinator
- MAC Mode = 0 und MAC Mode = 3

Ergebnis:
Das AT Remote Kommando wird nur von Punkt zu Punkt gesendet, alle nicht adressierten Geräte registrieren die Nachricht nicht. Es ist dabei auch irrelevant, ob die Nachricht von Koordinator zu Koordinator gesendet wird, oder ob der reguläre Fall auftritt: Koordinator zum Endgerät.

MM0 (Paket Nr. 206)
{ 61 8C 6C 16 DE C0 40 46 41 00 A2 13 00 E0 BE 0E 04
01 00 13 A2 00 41 46 40 C0 FF FE 02 53 4C 99 C6 }

Byte	Hex-Wert	Bedeutung
2	8C 61	frameCtl
1	6C	seqNo
2	DE 16	dstPanId (ID)
2	00 13 A2 00 4146 40 C0	dstExtAddr (DH/DL)
2	BE E0	srcShortAddr (MY)
1	0E	Frame Counter
1	04	(AT Command Request) Option
14	01 00 13 A2 00 41 46 40 C0 FF FE 02 53 4C	AP Frame Payload
2	C6 99	CRC

Das Modul 414640C0 antwortet mit einen Remote AT Command Response Frame (0x97)

{ 61 88 5B 16 DE E0 BE E1 BE 09 05 01 00 13 A2 00
41 46 40 C0 BE E1 53 4C 00 41 46 40 C0 2C BF }

Byte	Hex-Wert	Bedeutung
2	8C 61	frameCtl
1	5B	seqNo
2	DE 16	dstPanId (ID)
2	BE E0	dstExtAddr (DL)
2	BE E1	srcShortAddr (MY)
1	09	Frame Counter
1	05	(AT Command Response) Option
22	01 00 13 A2 00 41 46 40 C0 BE E1 53 4C 00 41 46 40 C0	AP Frame Payload
2	BF 2C	CRC

MM3 (Paket Nr. 218)
{ 61 8C 34 16 DE C0 40 46 41 00 A2 13 00 E2 BE
01 00 13 A2 00 41 46 40 C0 FF FE 02 53 4C 82 AF }

Byte	Hex-Wert	Bedeutung
2	8C 61	frameCtl
1	6C	seqNo
2	DE 16	dstPanId (ID)
2	00 13 A2 00 4146 40 C0	dstExtAddr (DH/DL)
2	BE E0	srcShortAddr (MY)
1	0E	Frame Counter
1	04	(AT Command Request) Option
14	01 00 13 A2 00 41 46 40 C0 FF FE 02 53 4C	AP Frame Payload
2	AF 82	CRC

Das Modul erhält keine Antwort, da das Empfängermodul von ein TX Transmit Frame ausgeht und den Nutzer am Empfänger ein RX Receive Frame ausgibt.

Konfiguration:

- 64-bit Zieladresse = 0x0
- 16-bit Zieladresse = 0xFFFE
- MAC Mode = 0 und MAC Mode = 3

MM0 (Paket Nr. 230)
MY ist bei allen Modulen gesetzt
{ 61 88 71 16 de 00 00 E0 BE 13 04 01 00 00 00 00 00 00 00 FF FE 02 53 4c 64 12 }

Byte	Hex-Wert	Bedeutung
2	88 61	frameCtl
1	71	seqNo
2	DE 16	dstPanId (ID)
8	00 00	dstExtAddr (DL)
2	BE E0	srcShortAddr (MY)
1	13	Frame Counter
1	04	(AT Command Request) Option
14	01 00 00 00 00 00 00 00 FF FE 02 53 4C	AP Frame Payload
2	12 64	CRC

Es wird ein TX Failure Frame an den Nutzer ausgegeben.

MM0 (Paket Nr. 243)
MY wurde bei allen Modulen auf den Wert 0x0 gesetzt
{ 61 88 75 16 DE 00 00 00 00 16 04 01 00 00 00 00 00 00 00 00 FF FE 02 53 4C 6E 96 }

Byte	Hex-Wert	Bedeutung
2	88 61	frameCtl
1	75	seqNo
2	DE 16	dstPanId (ID)
8	00 00	dstExtAddr (DL)
2	00 00	srcShortAddr (MY)
1	16	Frame Counter
1	04	(AT Command Request) Option
14	01 00 00 00 00 00 00 00 FF FE 02 53 4C	AP Frame Payload
2	96 6E	CRC

Das AT Kommando wird als Broadcast innerhalb des PAN ID-Bereiches verschickt und die Module Antworten mit dieser Konfiguration auch als Broadcast Nachricht.

Weitere Testfälle werden hier nicht aufgelistet, da sie sich im Wesentlichen nicht von den Testfällen *AL0010 – AL0240* unterscheiden.

Jedoch wird aus den aufgelisteten Fällen ersichtlich, dass der MaxStream Header wichtig für die *Remote AT Commands* und *Remote AT Commands Response* Frames ist. Wird ein Frame ohne den Header gesendet, wird er als TX Transmit Frame erkannt und als RX Receive 16/64 Frame an den Nutzer ausgegeben.

Analyse Fälle AL0490 – AL0720

Keine AES Verschlüsselungen und Nachrichten werden im AP Modus übermittelt.
Verwendeter Frame: 0x00 oder 0x01 TX Transmit 16/64 Frames. Die aufgezeichneten Pakete sind in der Wireshark Log Datei 2016-12-20_xb-s1_log.pcapng und in der Log Datei 2016-12-21_xb-s1_log-txTransmit.pcapng gelistet.

Mit den Transmit Frames können, wie im AT Modus, nur Text Nachrichten versendet werden, sie haben keinerlei Auswirkung auf die empfangenden XBee Module. Daher sind wie im AT Modus die MAC Mode Einstellungen nur für den ACK Request interessant. Wird ein Frame mit einem Max Stream Header gesendet, bekommt das Option-Byte den Wert 0x00 und wird damit als leer interpretiert. Alle Sender und Empfänger müssen jedoch dieselbe MAC Mode Einstellung besitzen, sonst wird der Frame nicht akzeptiert.

Die Transmit Frames haben noch ein zusätzliches Optionsfeld, mit dem sie die zu senden Daten manipulieren können.

- 0x00 – es werden keine Veränderungen vorgenommen
- 0x01 – schaltet den ACK Request aus
- 0x04 – sendet die PAN ID als Broadcast ID, die Originale PAN ID wird mit 0xFFFF aufgefüllt
- 0x08 – ist den XTend Modulen vorbehalten

Das Senden mit einer spezifischen Zieladresse und der Option 0x04 ändert die Punkt zu Punkt Verbindung nicht zu einer Broadcast Verbindung. Nur wenn die Zieladressfelder mit ‚F‘ gefüllt werden, werden die Frames als Broadcast Nachrichten versendet. Die Nachrichten können dann jedoch an Module mit einer anderen PAN ID gesendet werden. Auch Broadcast Nachrichten werden mit dieser Einstellung an alle PAN IDs geschickt.

Analyse Fälle mit SecurityEinstellungen

Da sich in den Analyse Fällen AL0010 – AL0720 gezeigt hat, dass sich viele Frames gleichen und nur die Variablen Bytes sich ständig verändern, wurde der Umfang der Tests reduziert. Es wurden nur Tests mit API Frames durchgeführt, versendet jeweils vom Koordinator und Endgerät.

Alle Geräte erhalten folgenden Key: 112233445566778899aabbccddeeff11

Die versendeten Frames:

Frame 1. 0x0 TX Transmit 64-bit Address, Option 0 (none)
7E 000D 00 01 0013A200 414640C0 00 43 48 37

Frame 2. 0x0 TX Transmit 64-bit Address, Option 1 (no ACK)
7E 000D 00 01 0013A200 414640C0 01 43 48 36

Frame 3. 0x0 TX Transmit 64-bit Address, Option 4 (Broadcast PAN ID)
7E 000D 00 01 0013A200 414640C0 04 43 48 33

Frame 4. 0x1 TX Transmit 16-bit Address (0xBEE1), Option 0 (none)
7E 0007 01 01 BEE1 00 43 48 D3

Frame 5. 0x1 TX Transmit 16-bit Address (0xFFFE), Option 0 (none)
7E 0007 01 01 FFFE 00 43 48 75

Frame 6. 0x17 Remote AT cmd, keine Adresse
7E 000F 17 01 00000000 00000000 FFFE 02 43 48 5D

Frame 7. 0x17 Remote AT cmd, 64-bit Adresse
7E 000F 17 01 0013A200 414640C0 FFFE 02 43 48 21

Frame 8. 0x17 Remote AT cmd, 16-bit Adresse
7E 000F 17 01 00000000 00000000 BEE1 02 43 48 BB

Frame 9. 0x17 Remote AT cmd, 16-bit und 64-bit Adresse, Option 0 (no ACK)
7E 000F 17 01 0013A200 414640C0 BEE1 00 43 48 81

Laut dem User Manual des XBee 802.15.4 Moduls, wird die Standard IEEE 802.15.4 Verschlüsselung mit einem 128-bit Schlüssel verwendet[22] und damit der Standardablauf nach Kapitel 9.2.1 Outgoing frame security procedure des IEEE Standards abgearbeitet.[23]

Des Weiteren wird immer eine 64-bit Quelladresse verwendet, die 16-bit Quelladresse (MY) wird ignoriert. Dies hat jedoch keine weitere Auswirkung auf die weiteren Funktionalitäten der Adressierungsmöglichkeiten. Genau wie bei dem Versenden der Frames ohne Verschlüsselung, empfängt das Modul nur Broadcast Nachrichten, wenn die 16-bit Adresse genau gleich mit allen andern 16-bit Adressen im Netzwerk ist. Oder kann explizit durch diese angesteuert werden.

Frames, die unverschlüsselt eintreffen, werden nicht interpretiert und ausgegeben. Wenn der Mac Mode 1 oder 2 beträgt, werden die Daten nur über die UART ausgegeben.[24]

[22] vgl. Digi User Guid, XBee / XBee-PRO S1 802.15.4 (Legacy) Modules Seite 67, EE command
[23] vgl. IEEE Standard for Low-Rate Wireless Networks, 802.15.4-2015.pdf, Seite 360/361.
[24] vgl. Digi User Guid, XBee / XBee-PRO S1 802.15.4 (Legacy) Modules Seite 73, MM command

AL0730

Die aufgezeichneten Pakete sind in der Wireshark Log Datei 2016-12-22_xb-s1_log-AES.pcapng und in der Log Datei 2016-12-22_xb-s1_log-AES2.pcapng gelistet.

In dem ersten Testdurchlauf wird der Mac Mode 0 verwendet und die Frames werden nacheinander wie oben aufgelistet ausgeführt, das heißt Paket 1 ist gleich mit Frame 1, Paket 2 mit Frame 2 usw.

(Frame 1, Koordinator) Übertragung erfolgreich an das adressierte Gerät.

Paket 1: { 69 CC 54 16 DE C0 40 46 41 00 A2 13 00 c2 40 46 41 00 A2 13 00 00 00 00 23 00 DF A0 9F 8B 9E 9B D8 62 }

Byte	Hex-Wert	Bedeutung
2	CC 69	frameCtl
1	54	seqNo
2	DE 16	dstPanId (ID)
8	00 13 A2 00 41 46 40 C0	dstExtAddr (DH/DL)
8	00 13 A2 00 41 46 40 C2	srcExtAddr (SH/SL)
5	00 00 00 23 00	Auxiliary Security Header
6	DF A0 9F 8B 9E 9B	AP Frame Payload (verschl.)
2	62 D8	CRC

(Frame 2, Koordinator) Übertragung erfolgreich an das adressierte Gerät.

Paket 2: { 49 CC 55 16 DE C0 40 46 41 00 A2 13 00 C2 40 46 41 00 A2 13 00 00 00 00 24 00 79 44 26 EF B2 13 45 A0 }

Byte	Hex-Wert	Bedeutung
2	CC 49	frameCtl
1	55	seqNo
2	DE 16	dstPanId (ID)
8	00 13 A2 00 41 46 40 C0	dstExtAddr (DH/DL)
8	00 13 A2 00 41 46 40 C2	srcExtAddr (SH/SL)
5	00 00 00 24 00	Auxiliary Security Header
6	79 44 26 EF B2 13	AP Frame Payload (verschl.)
2	A0 45	CRC

(Frame 3, Koordinator) Übertragung erfolgreich an das adressierte Gerät.

Paket 3: { 29 CC 56 FF FF C0 40 46 41 00 A2 13 00 16 DE C2
40 46 41 00 A2 13 00 00 00 00 25 00 74 E7 19 FA
17 8A CE 4A }

Byte	Hex-Wert	Bedeutung
2	CC 29	frameCtl
1	56	seqNo
2	FF FF	dstPanId (nicht veränderbar)
8	00 13 A2 00 41 46 40 C0	dstExtAddr (DH/DL)
2	DE 16	srcPanId (ID)
8	00 13 A2 00 41 46 40 C2	srcExtAddr (SH/SL)
5	00 00 00 25 00	Auxiliary Security Header
6	74 E7 19 FA 17 8A	AP Frame Payload (verschl.)
2	4A CE	CRC

(Frame 4, Koordinator) Übertragung erfolgreich an das adressierte Gerät.

Paket 4: { 69 C8 57 16 DE E1 BE C2 40 46 41 00 A2 13 00 00
00 00 26 00 43 CB 35 F9 FE BC 86 82 }

Byte	Hex-Wert	Bedeutung
2	C8 69	frameCtl
1	57	seqNo
2	DE 16	dstPanId (ID)
2	BE E1	dstAddr (DL)
8	00 13 A2 00 41 46 40 C2	srcExtAddr (SH/SL)
5	00 00 00 26 00	Auxiliary Security Header
6	43 CB 35 F9 FE BC	AP Frame Payload (verschl.)
2	82 86	CRC

(Frame 5, Koordinator) Übertragung an das adressierte Gerät ist fehlgeschlagen.

Paket 5: { 69 C8 58 16 DE FE FF C2 40 46 41 00 A2 13 00 00
00 00 27 00 06 62 2D 68 3E DF ED 62 }

Byte	Hex-Wert	Bedeutung
2	C8 69	frameCtl
1	58	seqNo
2	DE 16	dstPanId (ID)
2	FF FE	dstAddr (DL)
2	00 13 A2 00 41 46 40 C2	srcExtAddr (SH/SL)
5	00 00 00 27 00	Auxiliary Security Header
6	06 62 2D 68 3E DF	AP Frame Payload (verschl.)
2	62 ED	CRC

(Frame 6, Koordinator) Übertragung an die adressierten Geräte ist fehlgeschlagen. Begründung: Bei allen drei Modulen war die 16-bit Adresse noch eingespeichert, wird diese ebenfalls auf Null gesetzt, empfangen die Module das Kommando und liefern eine Antwort zurück.

Paket 6: { 69 C8 59 16 DE 00 00 C2 40 46 41 00 A2 13 00 00
00 00 28 00 5C C9 33 4E 93 09 B0 C2 46 6D 18 82
19 F6 89 7D 9A 9D }

Byte	Hex-Wert	Bedeutung
2	C8 69	frameCtl
1	59	seqNo
2	DE 16	dstPanId (ID)
2	00 00	dstAddr (DL)
8	00 13 A2 00 41 46 40 C2	srcExtAddr (SH/SL)
5	00 00 00 28 00	Auxiliary Security Header
16	5C C9 33 4E 93 09 B0 C2 46 6D 18 82 19 F6 89 7D	AP Frame Payload (verschl.)
2	9D 9A	CRC

(Frame 7, Koordinator) Übertragung erfolgreich an das adressierte Gerät

Paket 7: { 69 CC 5A 16 DE C0 40 46 41 00 A2 13 00 C2 40 46
41 00 A2 13 00 00 00 00 29 00 C3 21 C1 73 F0 CD
D0 F9 58 62 90 56 B3 D1 92 66 2D 19 }

Byte	Hex-Wert	Bedeutung
2	CC 69	frameCtl
1	5A	seqNo
2	DE 16	dstPanId (ID)
8	00 13 A2 00 41 46 40 C0	dstExtAddr (DH/DL)
8	00 13 A2 00 41 46 40 C2	srcExtAddr (SH/SL)
5	00 00 00 29 00	Auxiliary Security Header
16	C3 21 C1 73 F0 CD D0 F9 58 62 90 56 B3 D1 92 66	AP Frame Payload (verschl.)
2	19 2D	CRC

(Frame 8, Koordinator) Übertragung erfolgreich an das adressierte Gerät

Paket 8: { 69 C8 5B 16 DE E1 BE C2 40 46 41 00 A2 13 00 00
00 00 2A 00 C8 BF F3 D9 2E CB 9F E0 CB 74 A9 B8
D9 B9 73 E5 45 BD }

Byte	Hex-Wert	Bedeutung
2	C8 69	frameCtl
1	5B	seqNo
2	DE 16	dstPanId (ID)
2	BE E1	dstAddr (DL)
8	00 13 A2 00 41 46 40 C2	srcExtAddr (SH/SL)
5	00 00 00 2A 00	Auxiliary Security Header
16	C8 BF F3 D9 2E CB 9F E0 CB 74 A9 B8 D9 B9 73 E5	AP Frame Payload (verschl.)
2	BD 45	CRC

(Frame 9, Koordinator) Übertragung erfolgreich an das adressierte Gerät

Paket 9: { 69 C8 5C 16 DE E1 BE C2 40 46 41 00 A2 13 00 00
00 00 2B 00 E8 05 A0 79 45 43 90 0D CC AF 23 78
27 44 62 55 CE 87 }

Byte	Hex-Wert	Bedeutung
2	C8 69	frameCtl
1	5C	seqNo
2	DE 16	dstPanId (ID)
2	BE E1	dstAddr (DL)
8	00 13 A2 00 41 46 40 C2	srcExtAddr (SH/SL)
5	00 00 00 2B 00	Auxiliary Security Header
16	E8 05 A0 79 45 43 90 0D CC AF 23 78 27 44 62 55	AP Frame Payload (verschl.)
2	87 CE	CRC

Die Frameausgabe/-struktur vom Endgerät zum Koordinator ist identisch, lediglich die Counter, SH/SL und die verschlüsselten Bytes sind verschieden, daher werden die Daten nicht mit aufgeführt. Sie können dennoch im Log[25] eingesehen werden.

Wie bei dem MAC Header, ist auch bei dem Auxiliary Header das erste Byte ein Kontrollfeld, welches sich aus dem *Security Level*, *Key Identifier Mode*, *Frame Counter Suppression*, *ASN in Nonce* und einen *Reservierten Bitfeld* zusammensetzt. In allen Testfällen ist dieses Byte immer Null.

Es wird das Security Level 0 verwendet, also gibt es keine Security-Attribute, die komplette Datennutzlast ist vertraulich und somit verschlüsselt, es wird ebenfalls keine Authentifizierung durchgeführt und der MIC (Mmessage Integrity Code) ist 0 Bytes groß.
Die nächsten vier Bytes werden für den Frame Counter verwendet. Hier ist auffällig, dass Digi scheinbar nur drei Bytes verwendet, denn das vierte Byte ist während der Testläufe immer Null,

[25] 2016-12-22_xb-s1_log-AES.pcapng und 2016-12-22_xb-s1_log-AES2.pcapng (Wireshark Datein)

obwohl die Frames fortlaufend gezählt werden. Die Startzahl ist dabei nicht immer 0x1. Möglicherweise wird auch der Auxiliary Header in Little-Endian Format gespeichert, damit wäre das fünfte Byte des Headers, das Kontrollfeld und die ersten vier Bytes der Frame Counter.
An den Auxiliary Header anschließend folgt der verschlüsselte Daten Payload.

8.5 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 unserem Fall in den APS Frames.

Zum Beispiel Zeile 4836 aus der BitCatcher 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 können die Geräte auf jeder Cluster ID mit jeder Profil ID, oder Endpunkt Daten senden oder empfangen.[26]

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 Frames.
Zum Beispiel Zeile 20 aus der BitCatcher 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

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

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

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 BitCatcher 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

9 Programmbibliotheken

BitCloud

Atmels BitCloud[27,28] ist eine Programmbibliothek (bzw. Framework), die es ermöglicht Licht- oder automatisierte Heim-Applikationen zu implementieren, ohne weitere Konfigurationen am Board selbst oder am Netzwerkprotokoll vorzunehmen. Hierzu liefert Atmel bereits vorgefertigte Mikrocontroller Konfigurationen, zum Beispiel für die ATmega Baureihe, und das ZigBee PRO Netzwerkprotokoll aus. Dadurch ist die Programmbasis in Verbindung mit Atmel Mikrocontrollern kompatibel zu vielen anderen verfügbaren Produkten anderer Hersteller. Ein ATmega Mikrocontroller kann beispielsweise Daten mit einem XBee und einer Philips Lampe austauschen, ohne das Netzwerkprotokoll zu verändern.

So kann der Nutzer beispielsweise einen Regensensor aus dem Conrad Elektronik Shop mit seiner automatisierten Rollo- oder Fenstersteuerung verknüpfen, ohne Nachforschungen über die Netzwerkprotokolle anzustellen. Eine weitere Möglichkeit wäre die Verknüpfung einer alten Fernbedienung mit unterschiedlichen Haushaltsgeräten.

Er entwickelt nur noch seine Anwendungssoftware, die Daten zwischen dem Netzwerkprotokoll und dem jeweiligen verknüpften Endgerät austauscht.

Da aber schon nach der Auflistung der unterschiedlichen XBee Modulen festgelegt wurde, dass eine Programmbibliothek verwendet wird, welche nur auf dem IEEE 802.15.4 basiert, wird BitCloud in dieser Arbeit nicht weiter beachtet.

µracoli

Die Abkürzung µracoli steht für *microcontroller radio communications library* und ist eine Programmbibliothek, welche die Nutzung mit Atmels IEEE 802.15.4 Funkmodulen vereinfacht und deren Möglichkeiten demonstriert. µracoli ist ein OpenSource Projekt und basiert wahrscheinlich auf dem Atmel MAC Stack. Die erste Version ging am 16. Februar 2008 online. Am 17. Februar 2014 wurde die Version 0.4.2 Veröffentlicht[29], diese ist zeitgleich auch die aktuellste Version. Auf dem Repository[30] befindet sich am 14.10.2016 die Version rc0.5.0+.

Die Atmel-Programmierer liefern mit ihrer Bibliothek ebenfalls programmiert und funktionsfähig einen 802.15.4 Sniffer, einige Beispiele und ein Wireless UART Programm.

Anders als bei BitCloud muss sich der Nutzer, bei der Verwendung dieser Bibliothek selber um etwaige Netzwerk-Protokolle kümmern, um sich mit anderen Geräten zu vernetzen. µracoli übernimmt nur die Kommunikation und Konfiguration mit dem Mikrocontroller, die Erstellung eines Netzwerk-Frames wird vom Nutzer umgesetzt.

Da für dieses Projekt die µracoli Programmbibliothek genutzt werden soll bedeutet das, es müssen die XBee 802.15.4 Netzwerk-Frames (nicht API Frames) aufgezeichnet und analysiert werden, bevor mit der Planung des Prototyps begonnen werden kann.

[27] Produktseite von Atmel für BitCloud:
http://www.atmel.com/tools/BITCLOUD-ZIGBEEPRO.aspx?tab=overview

[28] AVR2052: BitCloud SDK Quick Start Guide

[29] Download Seite von µracoli http://uracoli.nongnu.org

[30] Savannah Repository von µracoli: hg.savannah.gnu.org/hgweb/uracoli/

10 Die Mikrocontroller Funkkommunikationsbibliothek - µracoli

10.1 Firmware

Version 0.4.2
Ist die letzte offizielle Releasversion, welche noch über Make-Files kompiliert wird. Sie beinhaltet alle in Kapitel 9 aufgelisteten Programme 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)

aktuelle Build Version
Die aktuelle Version wird nicht mehr über Makefiles kompiliert, sondern über das Build Plugin für Python (vgl. Kapitel 10.3). Die Version derfa2 gibt es nicht mehr und wurde in Raspberry Pi und ATmega256rfr2 auf RF Note unterteilt. Da im ConBee derselbe 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 geflasht werden.

10.2 Für AT Kommandos wichtige Programm Daten und Infos

SCons

Im aktuellen Build wird die Programmbibliothek nicht mehr über Makefiles kompiliert, sondern über das Python Zusatztool SCons [31]. Das Programm ist Open Source [32] und kombiniert die verschieden klassischen Make Tools, wie *autoconf*, *automake* und *ccach* in einem. Dabei werden die Konfigurationsdateien in Python geschrieben. Unterstützt wird jedoch auch eine ganze Reihe weiterer Programmiersprachen, sowie ein paar wichtige Markup-Makro-Sprachen, so zum Beispiel 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, dass 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. Wird ein anderer Compiler verwendet, fügt man diesen in der *toolchains.cfg* hinzu.

[31] Homepage von SCons Projekt: http://scons.org
[32] 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

toolchains.cfg

Befehlszeile	Beschreibung
[AVR8]	Mikrochip Name
MCUS	Mikrochip Typen, die von diesen Compiler unterstützt werden
CC	compiler collection
LD	link editor
AR	archive-maintaining program
RANLIB	Name des ranlib Programms
OBJCOPY	Name des objcopy Programms
CPPPATH	Pfad zur C Preprocessor Bibliothek
CCFLAGS	Compiler-Optionen
LINKFLAGS	Linker-Optionen
LIBPATH	Pfad zu den Bibliotheken
LIBS	Pfad zu den Bibliotheken von µracoli

board.h

In der Datei *bord.h* wird neben unterschiedlichen Standard Definitionen unter anderem auch der für die Knotenpunktkonfiguration relevante struct-Typ bereitgestellt.

```
1  typedef struct
2  {
3      /** The short address of the node. */
4      uint16_t short_addr;
5      /** The PAN ID (network ID) of the node. */
6      uint16_t pan_id;
7      /** The MAC address of the node (EUI64). */
8      uint64_t ieee_addr;
9      /** The radio channel. */
10     uint8_t  channel;
11     /** For future extensions, but can be used to store user data. */
12     uint8_t _reserved_[2];
13     /** Ibutton CRC to validate if the structure is correct. */
14     uint8_t crc;
15 } node_config_t;
```

Aus diesem 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

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, das 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 und anschließend der Radio-Typ ATmega256rfr2 übergeben/ definiert wird.

```
...
#elif defined(raspbee)
#define RADIO\_TYPE (RADIO\_ATMEGA256RFR2)
...
```

Wichtig ist auch die korrekte und einheitliche Definition zur *board.cfg*, denn 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.

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 können die chan, txp und die cca Deklarationen für die XBee Kommandos verwendet werden.
Hinweis: die Funktionen, die Werte in den EEPROM speichern, findet man in der transmitter.h.

```
typedef int8_t  channel_t;
typedef struct
{
    /** current channel see sub register ref SR_CHANNEL*/
    channel_t chan;
    /**  TX power index see sub register ref SR_TX_PWR*/
    unsigned int txp    : 4;
    /**  CCA mode see sub register ref SR_CCA_MODE */
    unsigned int cca    : 2;
    /**  ED threshold see sub register ref SR_CCA_ED_THRES */
    unsigned int edt    : 4;

    /**  clkm control  see sub register ref SR_CLKM_CTRL */
    unsigned int clkm   : 3;

} trx_param_t;
```

hif.h

Das hif im Dateinamen bedeutet Hostinterface, diese Datei stellt verschiedene Interface-Operationen bereit:

- `void hif_init(const uint32_t baudrate)`
  initialisiert das Hostinterface
- `uint16_t hif_get_number(int8_t base)`
  liest mit hif_getc einen 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, zum Beispiel 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”))`

timer.h

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

```
1 #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)`
  Stoppt 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.

transmitter.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.

```
1 ...
2 #elif RADIO\_TYPE == RADIO\_ATMEGA256RFR2 || RADIO\_TYPE == RADIO\
    _ATMEGA2564RFR2
3 #include "atmega\_rfr2.h"
4 ...
```

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

Die transmitter.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)`
  Liest 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 empfangenen 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 einen Buffer.
- `uint8_t trx_decode_datarate(uint8_t rhash, char * rstr, uint8_t nlen)`
  Dekodiert einen hash-Wert und gibt einen Datenraten-String-Zeiger zurück.
- `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)`
  Schreibt PAN ID 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.

10.3 µracoli Kompilieren

In diesem Projekt wurde die Version 0.5.0 201308xx vom 10.11.2016 des Savannah Repositorys verwendet. Diese lässt sich nicht mehr ohne Weiteres über Make Files Kompilieren es wird stattdessen das PythonScript SCons verwendet (vgl. 10.2).

Demnach muss zuallererst Python installiert sein bevor man das Scons Script anwenden kann. Von den µracoli Programmierern wird empfohlen, dass man sich eine Version von der Python Website[33] herunterlädt, die kleiner als Version 3 ist.

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

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

Des Weiteren müssen die benötigten Source und Header Daten in der *applications.cfg* hinzugefügt werden, welche in dem Ordner *Config* des µracoli Hauptordners zu finden ist. Zunächst gibt man in eckigen Klammern den Projektnamen an, welcher bei dem Kompilieren als Dateiname für die Hex und Elf Datei verwendet wird. Bestehen Abhängigkeiten zu dem µracoli Projekt werden die benötigten Header Daten unter *requires* ohne Pfadangabe hinzugefügt, dabei ist zu beachten, dass nur die Dateinamen geschrieben werden, die Endung **.h* und Kommas werden weggelassen. Unter *sources* und *headers* stehen die Applikationsdateien des eigenen Projektes mit der Pfadangabe, die relativ zum *App* Ordner ist.

Werden weitere Parameter oder ein anderes Optimierungslevel für den GCC-Kompiler benötigt, sind diese in der Datei *toolchains.cfg* zu editieren.

Um das SCons Script aufzurufen navigiert man in einem Terminal in den Hauptordner, in welchem sich auch die *SConstruct* Datei befindet. Mit dem Befehl:

```
1        $ scons <board>
```

wird das Scons Script aufgerufen, eine Make File generiert und das Projekt kompiliert. Dabei ist *<board>* gegen die gewünschte Plattform auszutauschen. Eine Liste, mit den verschieden Board-Varianten erhält man durch:

```
1        $ scons -h
```

Im Falle der AT Kommando API wird der ConBee Stick verwendet, welcher den gleichen Chipsatz besitzt wie das Raspberry Pi, daher wird als Boardname *raspbee* verwendet. Kommt ein ATmega128 mit RFnode zum Einsatz, wird das *derfn128u0* oder *derfn128* Board angegeben.

Nutzt man zum Flaschen den *GCF-Flascher* von dresden elektronik, muss die erstellte Hex Datei in eine binäre Datei umgewandelt werden. Am einfachsten lässt sich dies über folgende Batch Script realisieren:

[33] Download Seite von Python: https://www.python.org/downloads/

```
@echo off
set /p var=name of elf-file without: ".elf":

"C:\ProgTools\Atmel\AVR Tools\AVR Toolchain\bin\avr-objcopy.exe" -O binary "<Path where elf file is located>\%var%.elf" "<Path where file should be generated>\%var%.bin"

cd "<Path to GCF Flasher>"
GCFFlasher.exe -l

set /p port=Device:

GCFFlasher.exe -d %port% -f atcommands.bin

echo done!
pause
```

Verwendet man stattdessen den Flasher vom AV-Studio oder AVR-Dude müssen die Fuses korrekt eingestellt werden, da es sonst später zu Problemen bei der Baudrate kommt. Die entsprechenden Voraussetzungen können der jeweiligen *board_<name>.h* Datei entnommen werden. Im Falle des ConBee USB-Sticks müssen die Fuses wie folgt eingestellt werden, damit man anschließend mit einer Baudrate von 38400 über UART zu kommunizieren kann.

EXTENDED	0xFF
HIGH	0x91
LOW	0xE2

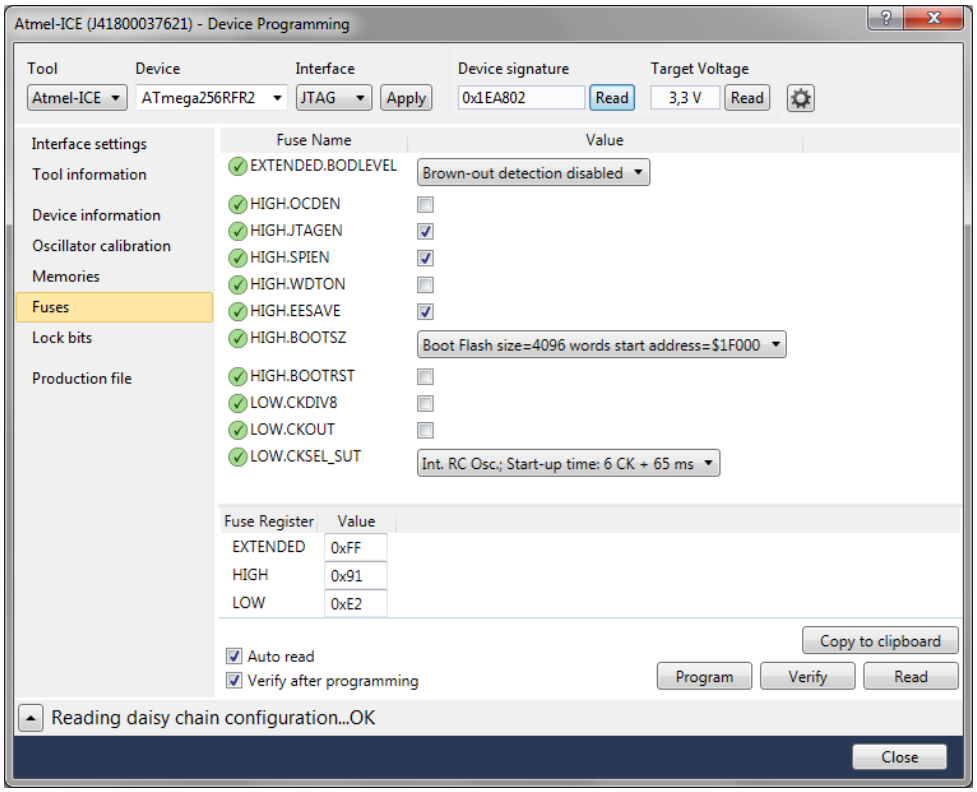

Abb. 9: Atmel Studio Device Programming – Fuses

10.4 Der Sniffer

Der mitgelieferte Sniffer kann eine große Hilfe bei Programmierung darstellen, denn anders als bei der ConBee/Bitcatcher Version kann dieser die 802.15.4 Frames von ZigBee Frames unterscheiden und liefert eine Detailausgabe an das Programm Wireshark.

Was wird benötigt:

- Ein ConBee Stick[34]
- Download von der µracoli Webseite
  - 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
- Wireshark [35] (verwendete Version 2.2.0)
- Python mit einer Versionsnummer, die kleiner als Version 3 ist (verwendete Version Python 2.7.12)
- Python Serial 2.7 Plugin [36]
- SCons Python Plugin (wird ab Version 5.0.0rc unterstützt)

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:

```
1 $ 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

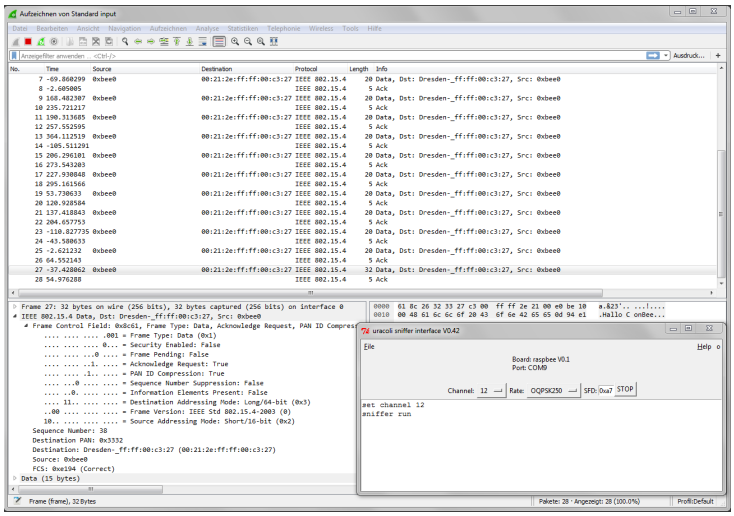

Abb. 10: Wireshark und µracoli Sniffer Programm

[34] ConBee von dresden-elektronik: http://www.dresden-elektronik.de/funktechnik/solutions/wireless-light-control/conbee/

[35] Homepage von Wireshark: https://www.wireshark.org/

[36] Download Seite vom Python Serial Plugin: https://sourceforge.net/projects/pyserial/

11 Planung der abstrakten AT Kommando C API

11.1 Programmstechnische Grundgedanken

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 Mikrocontroller spezifischen Programm-, bzw. Hardwarekonfigurationen nötig. Als alternative bringt die µracoli Bibliothek ebenfalls eine Timerfunktion mit.
2. Bei asynchroner 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 möglicherweise auch Spinnlocks benötigt.
3. Die Verarbeitung selbst muss in einem neuem Thread erfolgen. Soll die Anzahl der Threads manuell begrenzt werden? — Nein, die Funktionen müssen linear abgearbeitet werden.
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 AT Befehle kann in zwei verschiedene Arten erfolgen:
    a) Die XBee AT Kommandos werden eins zu eins übernommen und entsprechend abgearbeitet.
    b) Es werden mindestens zwei AT Bibliotheken hinterlegt, die am Ende programmintern zusammen führen und ein und dieselbe Funktion ausführen.
7. API Types als define oder enum?[37] — Ich bin für define, da hier flexiblere Wertezuweisungen möglich sind.
8. RWX define oder enum? — Ich bin für enum, da leichter zu handhaben
9. Wird eine ZigBee Erkennung benötigt?

11.2 AT Kommandos XBee

Mit drei Pluszeichen initialisiert man den zeitlich begrenzten Kommandomodus, dabei werden die Pluszeichen ohne ein Carriage Return oder New Line übergeben. Die Zeitperiode, in der ein AT Kommando interpretiert wird, kann mit CT (Command Mode Timeout) festgelegt werden. Als Standard wird von Digi der Hexadezimal Wert 0x64 vorgeschlagen. Dieser Wert wird noch einmal mit 100 multipliziert und ergibt die gesamte aktive Dauer des Kommandomodus (Standard: 10 sec).

Alle anderen Eingaben, die im allgemeinen Wartezustand eingelesen und mit ein Carriage Return bestätigt werden, behandelt das RF Modul als Transmitter-Nachrichten. Ist zudem der API Modus aktiviert (AP > 0), können eingehende Bytes auch als API Frame interpretiert werden (vgl. 11.3).

Ein AT Kommando sieht immer nach folgenden Schemata aus:

ATDT 1F<CR> oder ATDT1F<CR> [38]

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

[37] define, sind symbolische Konstanten, welche als Makro definiert werden. — enum, sind Aufzählungen
[38] Digi User Guid - XBee / XBee-PRO S1 802.15.4 (Legacy), Seite 33

Einige Veränderungen, die man mit den AT Kommandos vornimmt, sind zu einem späteren Zeitpunkt mit *AC* (Apply Changes) zu bestätigen.
Es gibt jedoch einige wenige Kommandos die nur einzeln ausgeführt und temporär gespeichert werden. Diese müssen nach der Eingabe sofort umgesetzt bzw. bestätigt werden, da sonst die eingegeben Informationen verloren gehen, dazu gehören zum Beispiel die Security Kommandos.

Nachdem der letzte Befehl eingegeben wurde, verlässt der Nutzer sofort mit *ATCN* (AT Command Null), oder nach festgelegter Zeit (CT – Command Times) den Kommandomodus.

Um über den AT Modus Werte auszulesen, wird der Parameterabschnitt einfach weggelassen, ein Wertebereichstest wie beim wAT gibt es nicht.

Sollen AT Befehle über eine Remoteschnittstelle zu einem anderen Gerät weiter versendet werden, dann müssen die Befehle gemäß der API Frame Spezifikation formatiert werden. Die AT Befehle sind nur lokal nutzbar.

11.3 API Frame

Das API Fenster ist eine Alternative zu den AT Kommandos und wird als ganze Einheit versendet. Diese Frames funktionieren ebenfalls nur lokal, beginnend mit dem Start Delimiter 0x7E, gefolgt von der Fenstergröße (MSB und LSB), dem API spezifischen Inhalt, sowie der Checksumme. Die Größe des Fensters selbst kann je nach Nachfrage- oder Befehlstyp variieren. Sollen Remote Kommandos versendet werden, wird der API Frame erst ausgelesen und ein Transmitterpaket gepackt und versendet.

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

Header

Start Delimiter	0x7E
Länge	0x0016

Content

Transmitterabfrage	0x11
Frame 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. Die Anzahl und Verfügbarkeit dieser API Typen ist vom jeweiligen Protokoll abhängig. Nicht alle Frames, die es bei den ZigBee Modulen gibt, stehen den 802.15.4 Modulen zur Verfügung und vice versa. Zwei Frames bleiben jedoch bei allen Modellen gleich, denn es gibt zwei verschiedene lokale API Command Typen.

Einen der jeden Befehl sofort umsetzt und speichert (0x08) und einen, der alle Kommandos in eine Warteschlange einreiht (0x09). Diese Veränderungen müssen zu einem späteren Zeitpunkt mit AC (Apply Changes) bestätigt werden.

Wie bei den AT Kommandos müssen ein paar Befehle sofort bestätigt werden, da sie sonst nicht gespeichert werden, sie stehen dennoch beiden Modi zur Verfügung. Es gibt demnach keine Fehlermeldung, oder einen Hinweis.

Die folgende Tabelle listet alle API Frame Typen auf, die es unter den ZigBee und 802.15.4 Modulen gibt.

[1]	[2]	API Frame Names API	API ID
Ja	/	TX Transmitt Request 64-bit Adresse – Textnachrichten im API Modus (Ausgang)	0x00
Ja	/	TX Transmitt Request 16-bit Adresse – Textnachrichten im API Modus (Ausgang)	0x01
Ja	Ja	AT Command – Setzt und führt Kommandos sofort aus	0x08
Ja	Ja	AT Command – Queue Parameter Value – Setzt Kommandos in eine Warteschlange, um sie Auszuführen muss 0x08 genutzt werden, oder das AT Kommando AC	0x09
/	Ja	ZigBee Transmit Request – Sendet Daten als RF Paket	0x10
/	Ja	Explicit Addressing ZigBee Command Frame – Erlaubt den APP-Layer für Datenübermittlung zu spezifizieren	0x11
Ja	Ja	Remote Command Request – Setzt Modulparameter auf ein Remotegerät, der Request muss mit einen AC an das Gerät gesendet und bestätigt werden	0x17
/	Ja	Create Source Route	0x21
/	Ja	Register Joining Device	0x24
Ja	/	RX Receive Package 64-bit Adresse – Antwort im API Modus (Eingang)	0x80
Ja	/	RX Receive Package 16-bit Adresse – Antwort im API Modus (Eingang)	0x81
Ja	/	RX Receive Package IO 64-bit Adresse – Antwort im API Modus (Eingang)	0x82
Ja	/	RX Receive Package IO 16-bit Adresse – Antwort im API Modus (Eingang)	0x83
Ja	Ja	AT Command Response – Gibt nach einem Kommando einen Ergebnistext zurück	0x88
Ja	/	TX Transmit Status – Status ob der Frame übertragen wurde, oder nicht (lokal)	0x89
Ja	Ja	Modem Status	0x8A
/	Ja	ZigBee Transmit Status – Nach einer TX Übertragung wird ein TX Statusnachricht ausgegeben	0x8B
/	Ja	ZigBee Receive Packet (AO=0) – Das Modul sendet das empfangene Paket über UART weiter	0x90
/	Ja	ZigBee Explicit Rx Indicator (AO=1) – Das Modem sendet das empfangene Paket über UART weiter	0x91
/	Ja	ZigBee IO Data Sample Rx Indicator – Das Modul sendet das IO Sample über UART weiter	0x92

[1]	[2]	API Frame Names API	API ID
/	Ja	XBee Sensor Read Indicator (AO=0) – Das Modul sendet das IO Sample über UART weite	0x94
/	Ja	Node Identification Indicator (AO=0) – Die Nachricht wird erhalten, wenn der Knoten identifiziert werden soll	0x95
Ja	Ja	Remote Command Response – Sendet eine Remoteantwort	0x97
/	Ja	Extended Modem Status	0x98
/	Ja	Over-the-Air Firmware Update Status	0xA0
/	Ja	Route Record Indicator – Erhält man, wenn ein Routingkommando gesendet wurde	0xA1
/	Ja	Many-to-One Route Request Indicator – (über UART wenn eine Anfrage besteht)	0xA3
/	ja	Join Notification Status	0xA5

Verfügbarkeit der Kommandos: [1] 801.15.04 [39]
[2] ZigBee

11.4 Programmspezifische Grundgedanken

Funktionen die Benötigt werden

- Funktion, die ein Konsolenkommando annimmt.
- Funktion, die ein Konsolenkommando zur Visualisierung zurückgibt.
- Funktion, die ein Konsolenkommando in ein API Frame umwandelt.
- Funktion, die ein Konsolenkommando über TX versendet.
- Funktion, die ein Konsolenkommando über RX liest und auswertet.
  - Mit ZigBee Erkennung (optional)
- Funktion, die Werte in einem Register speichert und ausliest.
- Funktion, die ERROR Codes auswertet.
- Funktion, die ein Hostinterface initialisiert → eg. hif.h

Programmstruktur

Die AT command API soll die auf Stack's aufbauenden Bibliotheken, wie µracoli oder BitCloud, erweitern und dem Nutzer ermöglichen eingegebene oder von einem anderen RF Modul empfangene Daten interpretiert auszugeben, auszuführen oder über den Transmitter zu versenden. Dabei besteht die AT command API aus AT Kommandos und API Frames, die den Funktionsumfang der AT Kommandos erweitern. [Abb. 11] Sämtliche Aktivitäten, die am Ende des Praktikums der Prototyp können soll, sind dem Aktivitätsdiagramm aus Abbildung 15 zu entnehmen.

Nachdem das Gerät mit Strom versorgt ist, werden alle relevanten globalen Variablen initialisiert, der EEPROM ausgelesen und das Gerät konfiguriert. Anschließend wechselt das Hauptprogramm in die aktive Warteschleife und wartet auf Eingaben der UART, oder des Transreceivers.

Die UART nimmt sofort jeden Buchstaben, der getippt wird an und füllt einen Puffer.
Im Transparentmodus, in dem die API Frames abgeschaltet sind, und der AT Kommando Modus nicht aktiv ist, wartet das Modul nach dem letzten eingegeben Zeichen standardmäßig die

[39] Digi User Guid - ZigBee RF Modules, 90000976.pdf, Seite 111 ff. + XCTU Tool

Zeit ab, welche die drei Zeichen benötigen um über die UART versendet zu werden. Wird nach dieser bestimmten Zeitspanne kein weiteres Zeichen hinzugefügt, so wird dieser Text über den Transmitter versendet.
Werden drei aufeinander folgende Zeichen des Command Sequence Character empfangen, darf nach dem letzten Zeichen keine weitere Eingabe erfolgen, bis das Modul nach vordefinierter Zeit in den temporären Kommandomodus wechselt.
Befindet sich das Modul im AT Kommando Modus, müssen Befehle mit einen Carriage Return bestätigt werden, um verarbeitet zu werden. Dabei ist zu beachten, das AT Kommandos nur Lokal ausgeführt werden. Nach einer festgelegten Zeit, wenn keine weitere Eingabe registriert, oder der AT Befehl ATCN eingegeben wurde, kehrt das Programm in den allgemeinen Wartezustand zurück. [Abb12]
API Frames werden als vordefinierter Frame empfangen, dieser wird während des Einlesens validiert und gibt gegebenenfalls den Return-Wert ‚ERROR'zurück.

Der AT Kommandomodus besitzt im Wesentlichen nur vier Funktionen. Eine zentrale Funktion, die auf die Eingabe wartet und die anderen drei Funktionen aufruft wenn sie benötigt werden. Eine Timer-Funktion, die ggf. zurückgesetzt werden kann, wenn eine Eingabe registriert wurde. Eine Lese- und Ausführ-Funktion, die Instruktionen ausführt, oder gespeicherte Werte über die UART zurückgibt. Und eine Schreib-Funktion, die neue Werte in den Flash, sowie in den EEPROM schreibt. [Abb. 13]

Ist der API Modus aktiv (mit einem Wert von eins oder zwei), werden eingehende Daten nur als API Frame akzeptiert. Wie man der Tabelle aus Kapitel 11.3 entnehmen kann, gibt es hier verschiedene Aufgabengebiete und somit viele Funktionen. Die wichtigsten sind, die lokale AT Funktion 0x8, die AT Antwort 0x88, das AT Remote Kommando 0x18 und die AT Remote Antwort 0x97. Zusätzliche Kommandos, die von den XBee Modulen nicht unterstützt werden, können mit den Frame 0x18 abgerufen werden. [Abb. 14] Die Frames 0x8 und 0x9 können mit in den AT Funktionen read, exec und write integriert werden.

Für das Versenden der Pakete sind weitere Funktionen erforderlich, welche die Aufgabe haben das Sende-Array je nach Typ zu füllen. Es zeigte sich schon in der Analyse, dass verschiedene MAC Header Variationen verwendet werden. (vgl. 8.4) Der Wert richtet sich dabei nicht nur an den AES Security und Mac Mode Einstellungen, sondern auch an der 16-bit Adresse des Gerätes und an dem Zieladressentyp. Wichtig für den Empfänger ist das Frame Control Field, welches aus den ersten beiden Bytes des zu sendenden Frames besteht. [Abb. 16]

Im Transparenten Modus werden die Einstellungen mit dem AT Kommandos vorgenommen und ändern sich bis zum nächsten überschreiben des Parameters nicht, daher kann der gesamte MAC Header einmal zusammengestellt werden und im Speicher verbleiben bis eine weitere Änderung vorgenommen wird. Im API Frame Modus muss der Header jedoch jedes Mal neu zusammengestellt werden, da durch die unterschiedlichen Frame Typen die Adressen und Optionen mitgeliefert werden. Die Abarbeitung eines Paketes bis zum Versand kann der Abbildung 17 entnommen werden.

Bei dem Auslesen eines empfangenen Paketes muss nicht nur das Frame Control Field beachtet werden, sondern auch die Parameter, die mit dem MaxStream Header im Daten Payload mitgeliefert werden. Aber auch hier ist entscheidend, welcher Modus zurzeit aktiv ist. Während im Transparentmodus der Daten Payload fast komplett über die UART ausgegeben wird, unterscheidet man im API Modus zwischen drei Optionen, welcher Frame gerade bearbeitet wird. [Abb. 18]

[40] vgl. IEEE Standard for Low-Rate Wireless Networks, 802.15.4-2015.pdf, Seite 152

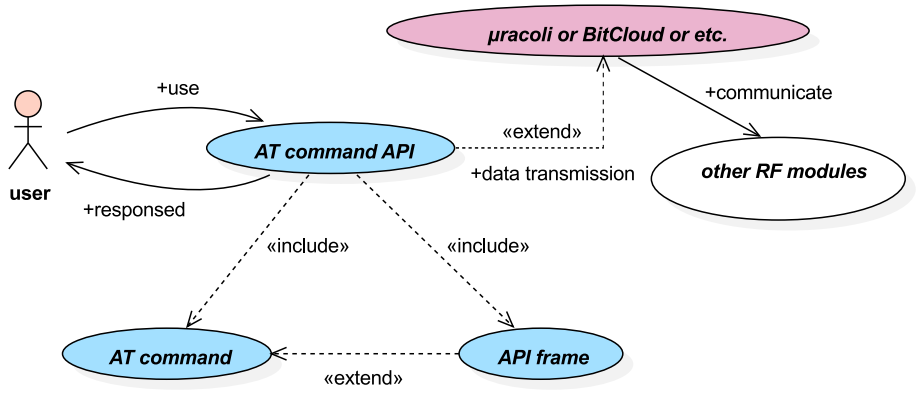

Abb. 11: Grundstruktur der AT command API

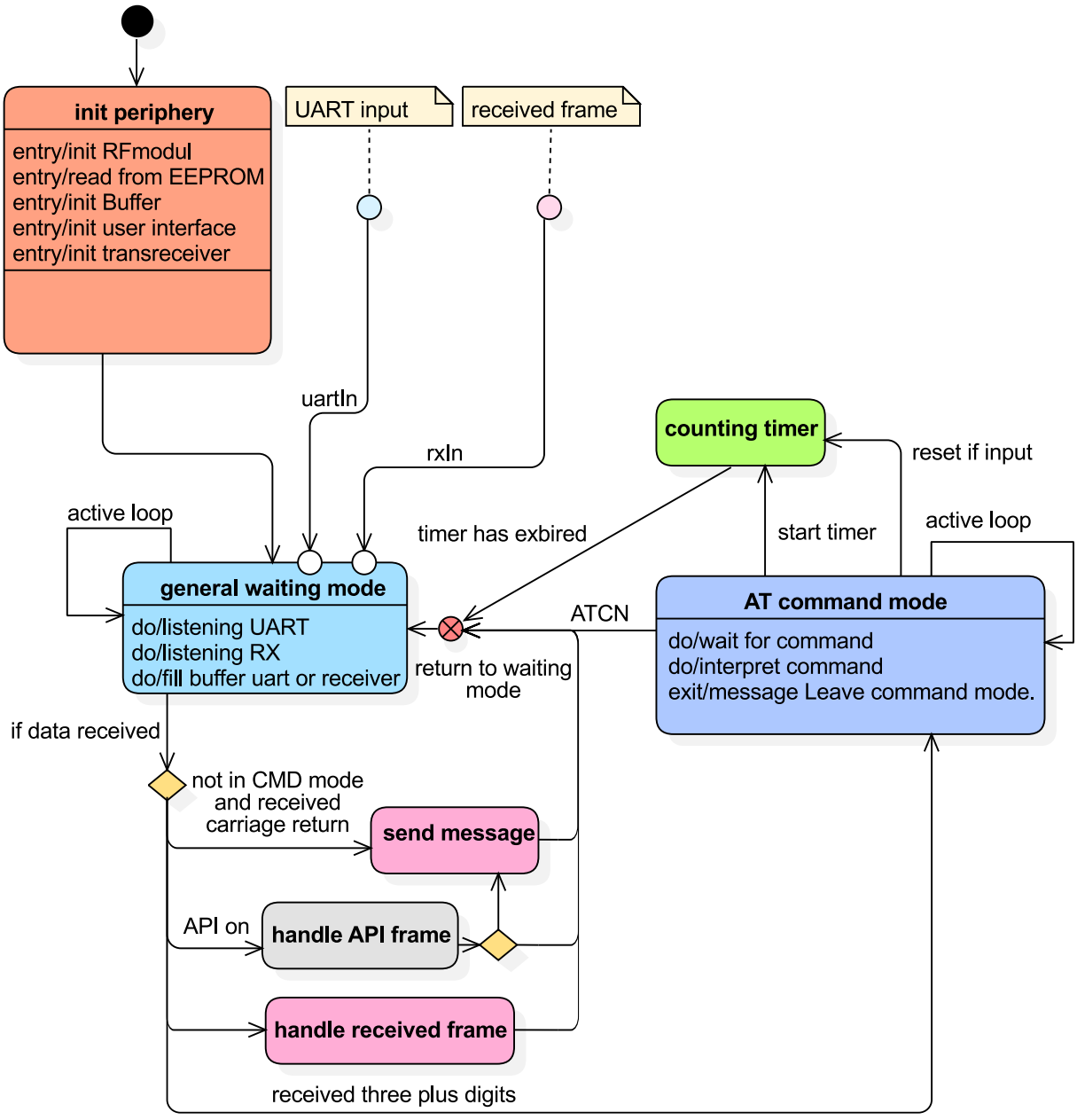

Abb. 12: Statusdiagramm

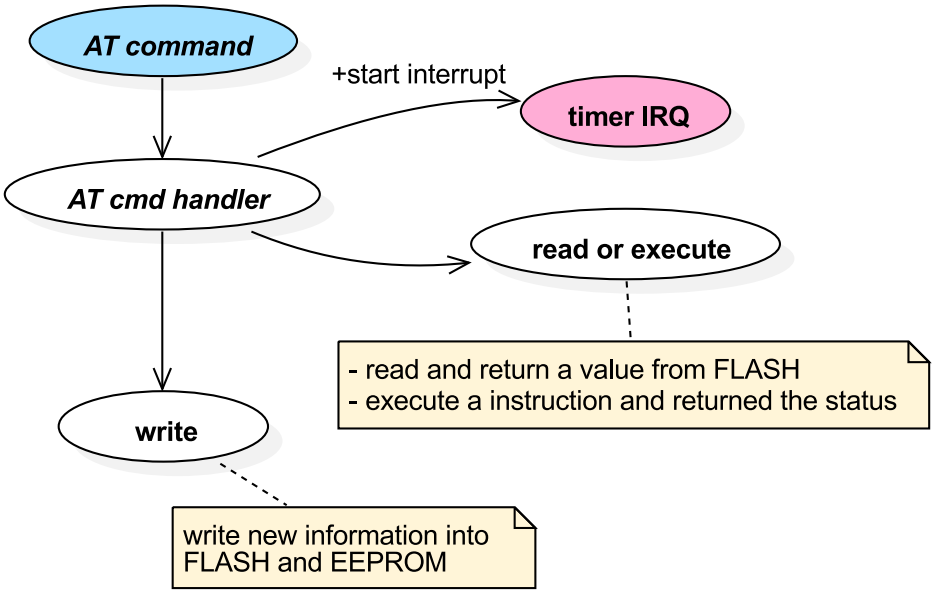

Abb. 13: AT command mode

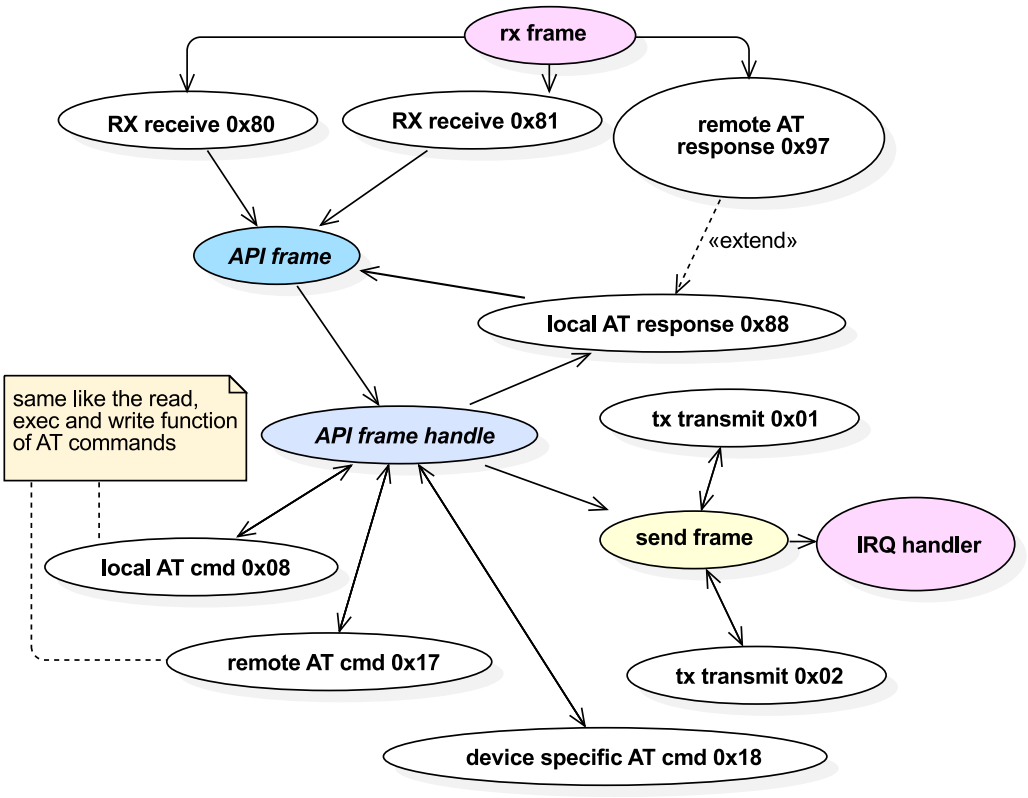

Abb. 14: API frame mode

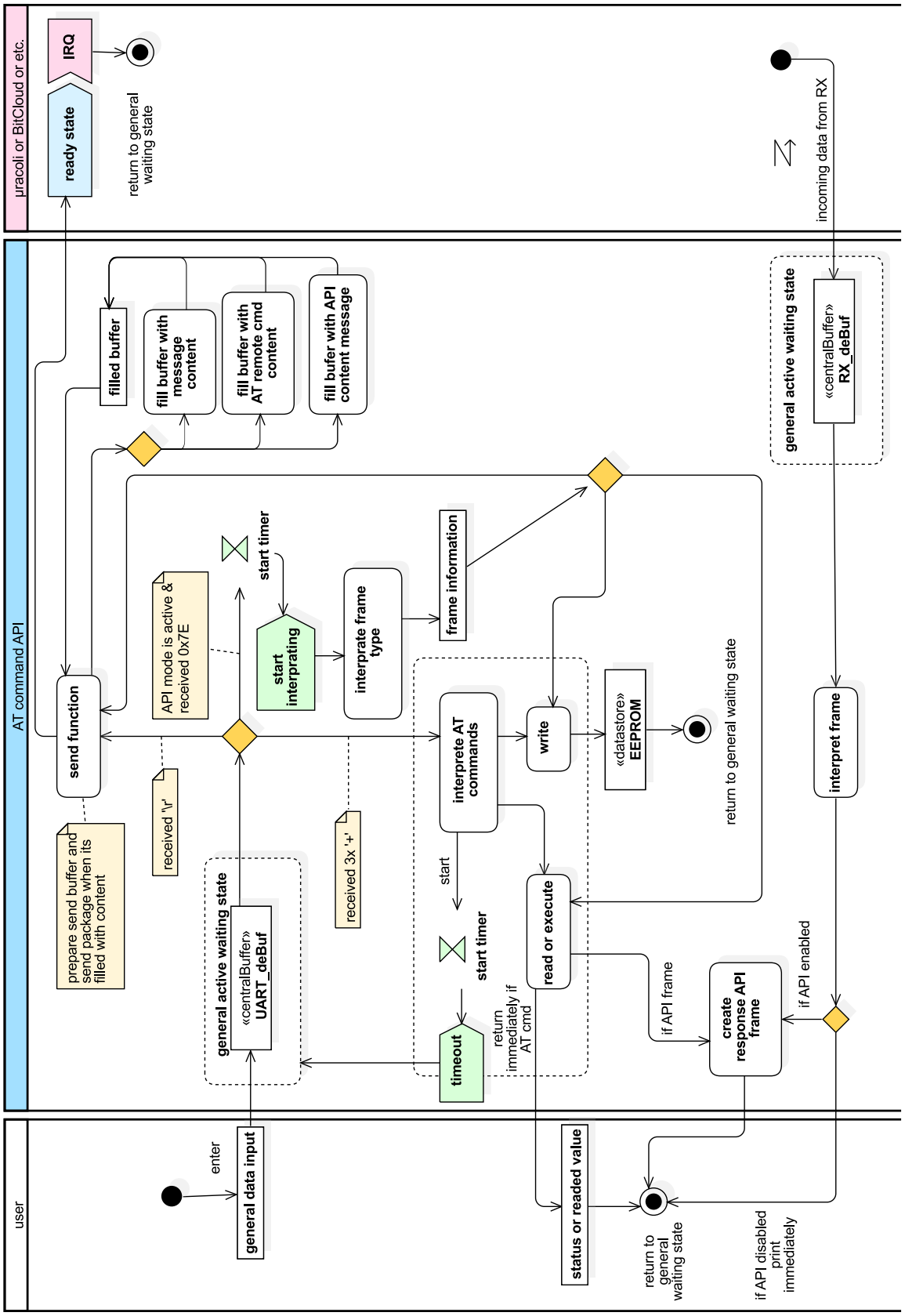

Abb. 15: Aktivitäten Diagramm

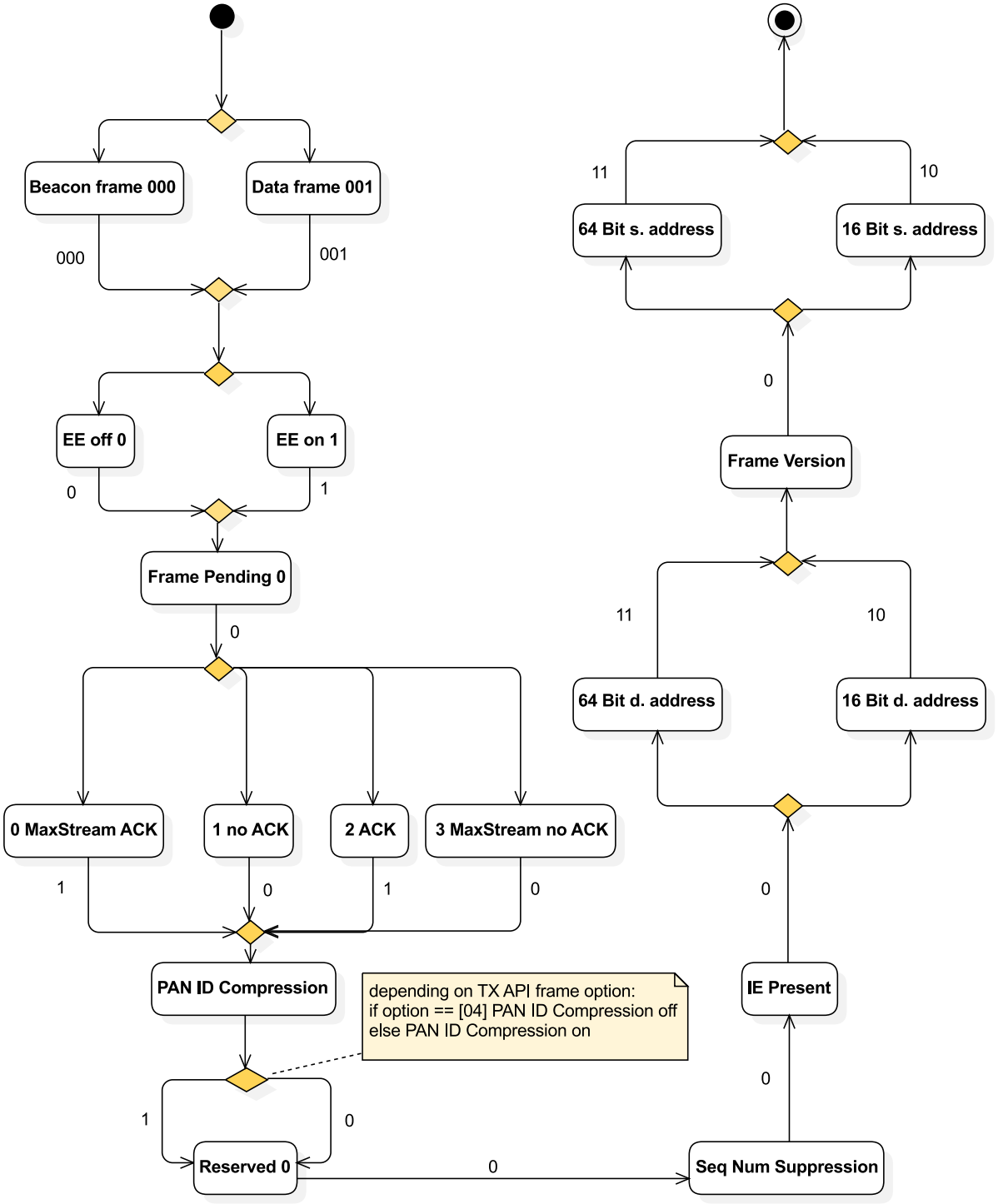

Abb. 16: Erstellung des Frame Control Fildes[40]

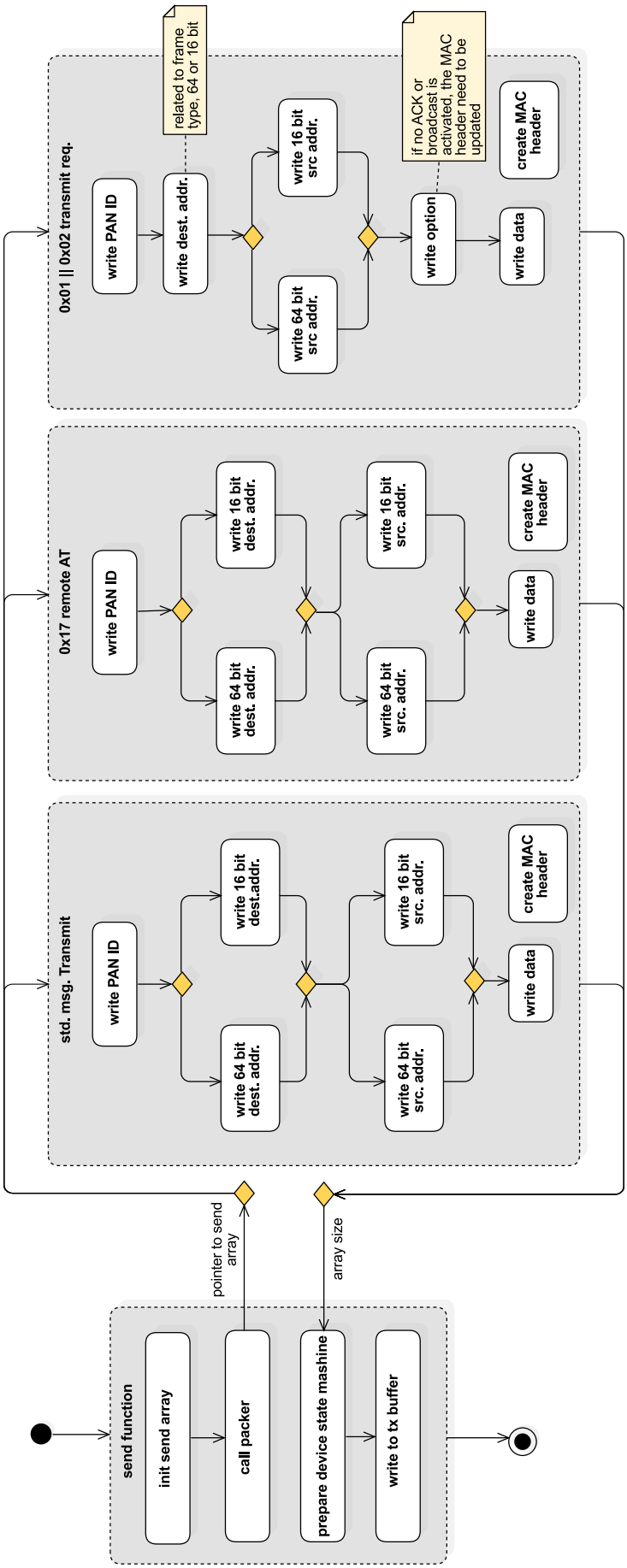

Abb. 17: Ablauf Paket versenden

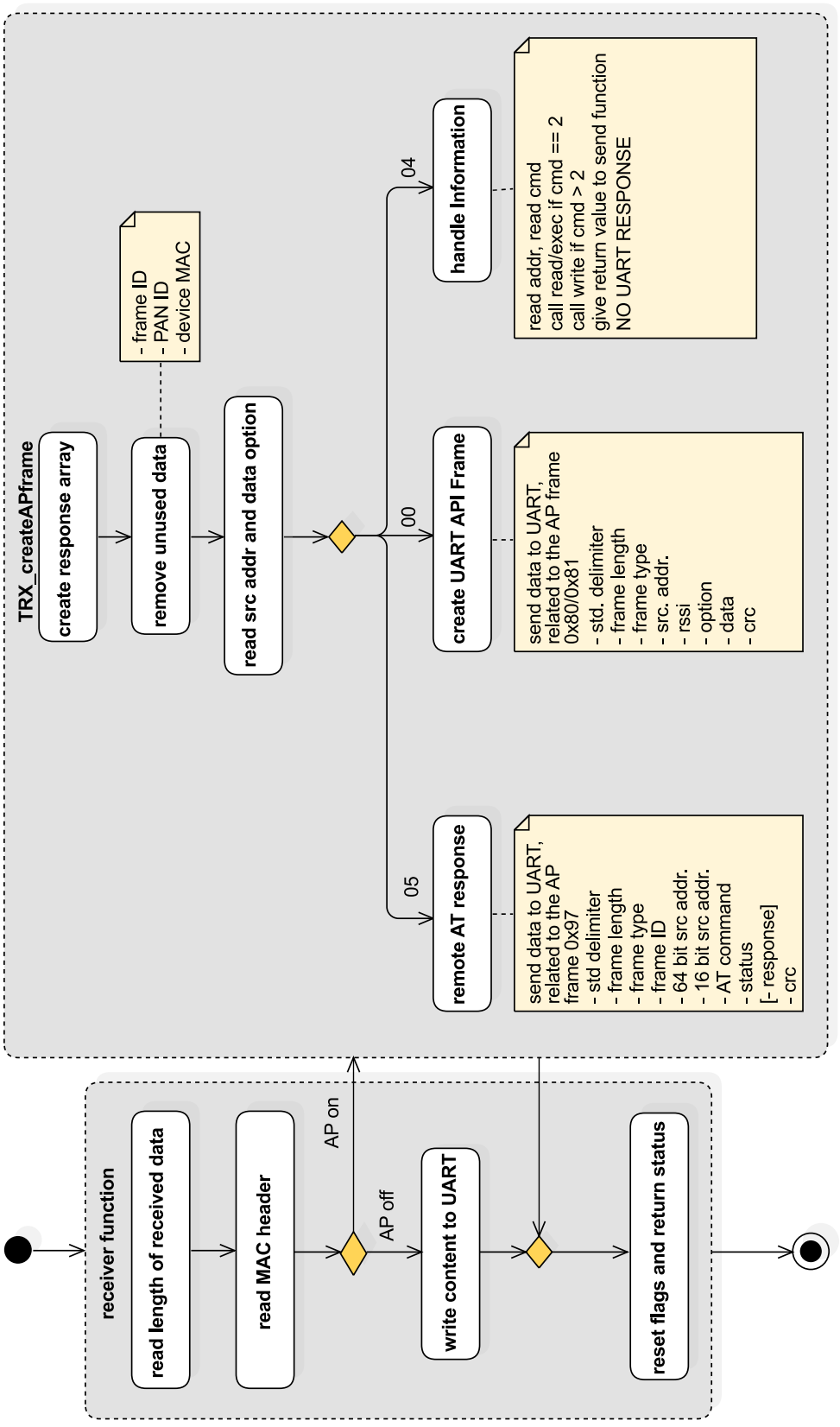

Abb. 18: Ablauf Paket empfangen

12 Der Prototyp – Version A

Der Prototyp kann 78 AT Befehle erkennen und speichern. Nicht hinter jedem Kommando ist zu Beginn eine Funktionalität geknüpft.

Der Prototyp kann:

- die PAN ID setzten
- den Kanal setzen
- die Zieladresse (64-bit) setzen
- die eigene 64-bit Adresse auslesen
- verschiedene Parameter abspeichern und auf den EEPROM speichern
- die Firmware auf Default Werte zurücksetzen
- die Zeit speichern, bis der Kommandomodus verlassen wird
- das Symbol speichern, welches zum Eintreten des Kommandomodus benötigt wird
- sendet einfache Nachrichten von Punkt zu Punkt
  - das Aufteilen in mehreren Nachrichten ist noch nicht implementiert, alles was über die Paketgröße (127Byte) hinaus geht, wird ignoriert
- sendet remote AT Kommandos zu ein weiteren RF Modul
- empfängt Textnachrichten
- empfängt „API Frames“, wandelt diese Informationen (nur Remote und einfache RX Nachrichten)

Was muss getestet werden:

1. Senden von Textnachrichten, was passiert, wenn die Nachricht zu lang ist?
2. Senden von API Frames (Delimiter an verschieden Stellen)
3. Empfangen von Textnachrichten
4. Empfangen von „API Frames“
5. Validieren und speichern von Werten

12.1 Die Prototyp-Versionsnummer

Die AT Command Prototypen Versionsnummer enthält zwei Versionsnummern zugleich.
Der erste Teil ist die Version der de-AT-Command-API und der zweite Teil ist die Versionsnummer auf die de-API basiert.

Wenn der Prototyp komplett ausgebaut ist und reif für die ZigBee Erweiterungen sollte die Versionsnummer abgeändert werden. Für den Kunden kann es hilfreich sein, wenn er eine Versionsnummer bekommt, an der er erkennen kann, wann diese veröffentlicht wurde. Zum Beispiel wäre eine Kombination aus Releases, Jahr und Monat denkbar. Eine Veröffentlichungsreihe würde dann in etwa dem folgenden Zyklus entsprechen:

1.17.3 (Voller Release),
1-1.17.3 (interne Release),
1-2.17.3 (interne Release),
2.17.4 (Voller Release)
...
50.45.2
...

12.2 H-Files

1. _global.h
   - Definition globaler Variablen
2. apiframe.h
   - Definition der API Frame Typen
   - Definition des api_f struct
   - Prototypen der Funktionen, die in anderen c-Files aufgerufen werden
3. circularBuffer.h
   - Definition des typdef struct deBuffer_t
   - Definition der Buffergröße und der Buffermaske
   - Prototypen der Bufferfunktionen
4. cmd.h
   - Definition der Kommandoaktionen read, write und execute
   - Definition des typedef struct CMD (Aufbau ähnlich der wAT Kommandos)
   - diese Datei muss editiert werden, wenn neue Befehle hinzugefügt werden
5. defaultConfig.h
   - Definition der Standartwerte für die Zurücksetzung der Module (XBee basierend)
   - Werden weitere XBee Module hinzugefügt muss diese Datei erweitert werden
6. atlocal.h
   - Prototypen der Funktionen, die in anderen C-Files aufgerufen werden
7. enum_cmd.h
   - Enumeration aller AT Kommandos, die von der Firmware verarbeitet werden
   - diese Datei muss editiert werden, wenn neue Befehle hinzugefügt werden
8. enum_status.h
   - Enumeration der Status- und Fehlermeldungen
9. rfmodul.h
   - definiert das typedef struct RFmodul (Speicherort der Parameter im Flash)
   - Prototypen der default setter
   - Prototypen für das Schreiben und Lesen zum EEPROM
10. stackrelated.h — Stack abhängig
    - Prototypenzeiger der UART-Funktionen
    - Prototypenzeiger der Transreceiver-Funktionen
    - Prototypen der Funktionen, die in anderen c-Files aufgerufen werden
11. stackdefines.h
    - neue Definitionen der Stack defines um die Suche und das Ersetzen zu erleichtern
12. stackrelated_timer.h
    - Prototypenzeiger der Timer-Funktionen

12.3 C-Files

1. atcommands.c — Hauptdatei mit der main-Funktion
    - Initialisiert die RF modulvariablen
    - Liest den EEPROM aus und speichert die Daten in den Flash
    - Initialisiert den UART und Transceiver Buffer
    - Konfiguriert das Gerät für die einzelnen Operationen
    - Startet die aktive Warteschleife
        - Ruft die TRX Handler Funktion auf (12.2.10)
        - Ruft die lokale AT Funktion auf (12.2.6)
        - Ruft die UART API Frame Handler Funktion auf (12.2.2)
2. atlocal.c — Funktionen für die Lokale AT Behandlung
    - Hauptfunktion für den AT Command Mode
    - Funktionen read, write und exec für die Lokale Abhandlung
        - bei Erweiterung der Kommandos in C-Files auslagern, oder in Templates umwandeln
3. apiframe.c — Funktionen die, die API Frames behandeln
    - Hauptfunktion um die API Frames abzuarbeiten
    - Aufruf der Frame-Typ-Funktionen
        - 0x08 und 0x09 local AT commands, ist in den rwx-Funktionen der atlocal.c Datei integriert
        - 0x18 specific device commands
        - 0x17 remote AT commands, ist in der trx0.c ausprogrammiert
        - 0x88 local AT response
        - 0x97 remote AT response
4. attable.c — Tabelle mit den AT Kommandos
    - Auflistung aller Kommandos mit ihren Rechten (rwx) in einem Array
    - diese Datei muss editiert werden, wenn neue Befehle hinzugefügt werden
5. circularBuffer.c
    - Definition der Bufferfunktionen
6. nvm_eeprom.c — Speicher- und Lesefunktion für den EEPROM
    - Definition des EEPROM struct NVM
    - Set user values und default Funktion
    - Get Funktion
    - EEPROM CRC Funktionen

7. setter.c — Default-Setter-Funktionen
    - setter um die Standardwerte in dem Flash wieder herzustellen
8. timer0.c — definiert die Verbindung zu der Timer-Funktion des Stacks
    - Initialisierung Timer-Pointer
    - Definition der Millisekunden für das Modul
9. trx0.c — definiert die Verbindung zu den Transceiver-Funktionen des Stacks
    - Initialisierungsfunktion für den Transreceiver
    - Sende- und Empfängerhändler
    - Funktionen um die Pakete, die gesendet werden zu packen
10. uart.c — definiert die Verbindung zu den User Interface Funktionen des Stacks
    - Initialisierungsfunktion der UART-Pointer

12.4 Funktionen, für die spätere Implementation

- API Frames 0x8A Modem Status
- API Frames 0x82 RX Receive Packet IO mit 64-bit Adresse
- API Frames 0x83 RX Receive Packet IO mit 16-bit Adresse
- API Frames mit Escape Characters
- Remote Knoten Suchen
- Gerätespezifische PIN Einstellungen (zum Beispiel D0–D8 und T0–T7, etc)
- Kanäle Scannen
- Sleep und Energieeinstellungs Funktionen
- AES Verschlüsselungsfunktionen
- Clear Channel Assessment (CCA) Funktionen
- die Anzahl wie viele Pakete gelesen werden bevor diese gesendet werden
- PWM und RSSI Konfigurationen
- Guart Times für den AT Kommandomodus (Zeit bis das Modul eine Aktion ausführt, wurde im Prototyp anders realisiert und ist somit optional)

13 Der Prototyp – Version B

13.1 Änderungen zu Version A

Da die Programmversion A zu statisch und innerhalb der API voneinander abhängig ist, werden in Version B verschiedene Änderungen vorgenommen. Jedoch wurde diese bis zur Abgabe des Dokumentes noch nicht vollständig umgesetzt.

- die Datei, mit der Main Funktion wurde in *at_api_main.c* umbenannt
- es wurden die Funktionen mehrere C-Files aufgeteilt
- das Parsen der UART Eingabe erfolgt nun über eine State Machine, jeweils für AT/Transparenz [Abb. 19] und API Frame Modus [Abb. 20]
  - dadurch wurden die Guard Times und Command Timeouts genau so angewendet wie sie bei den XBee Modulen verwendet werden
- Befehle aus dem AT und API Modus benutzen nicht mehr gemeinsam dieselben Read, Write und Execute Funktionen
- es wurden Hilfsfunktionen implementiert die, die Eingabe mit den Minimal- und Maximalwerten abgleicht
- der Buffer ist nun ein eigenes Objekt und wurde durch weitere Funktionen ergänzt
- es wurde ein Buffer Array angelegt
- die AT Command Table wurde um weitere Inhalte erweitert
  - jedes Kommando erhält den zugehörigen Offset der Variablen in RFmodul Struct, dadurch kann der Umfang der Read und Exec Funktionen reduziert werden
  - die Minimal- und Maximalwerte, die zu jedem Kommando gehören wurden hinzugefügt
  - Funktionspointer zu dem Set und Validierungsfunktionen wurden hinzugefügt, dadurch entsteht ein einheitlicher Funktionsaufruf und reduziert damit den Umfang der Write Funktion
- es kann nun auch mit einer ID in der AT Command Table gesucht werden
- die boolesche Variable *dirtyBits* wurde hinzugefügt
- die API Frame wurde als eigenes Objekt mit den zugehörigen Get und Set Methoden abgeändert
- die alten Get Methoden des RFmodul Structs wurden gegen Get Offset Methoden ausgetauscht, um die Werte aus den Variablen auszulesen gibt es die einheitliche *GET_deviceValue* Methode

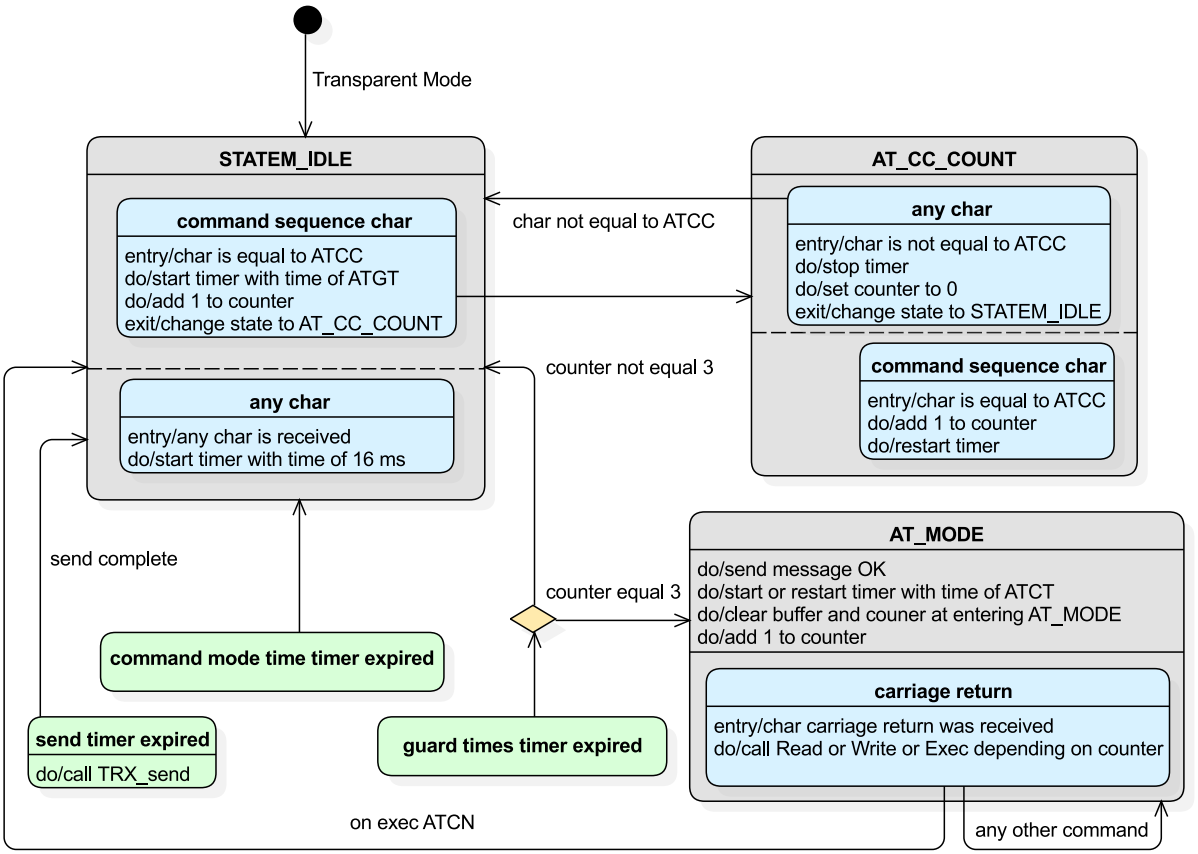

Abb. 19: State Machine AT Kommandos

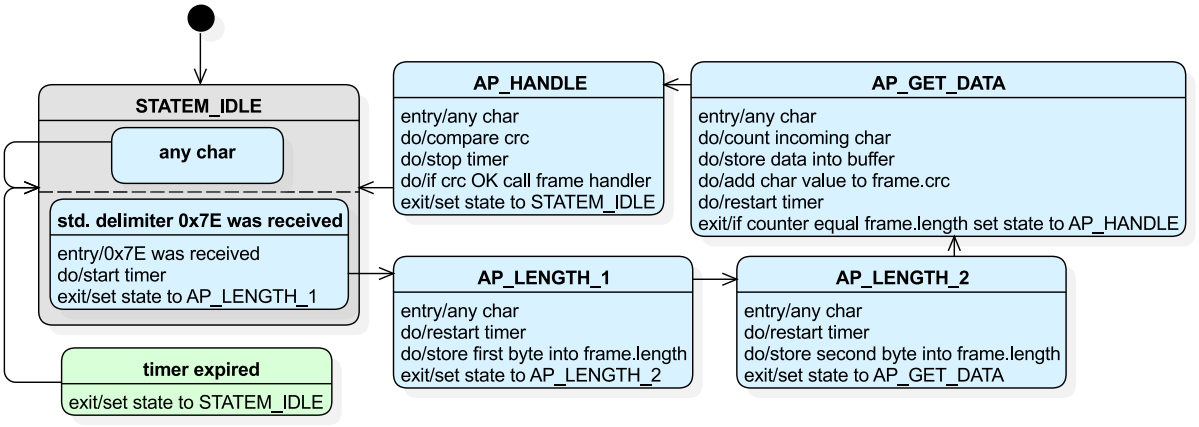

Abb. 20: State Machine API Frames

13.2 H-Files

1. _global.h
   - Definition globaler Variablen
   - dirtyBits Variable
2. ap_iframe.h
   - Prototypen für Set, Get, Callback Funktionen
3. circularBuffer.h
   - defines der Buffergröße und der Buffermaske
   - Prototypen der Bufferfunktionen
4. cmd.h
   - defines der Kommandoaktionen read, write und execute
   - Definition des typedef struct CMD (Aufbau ähnlich der wAT Kommandos)
   - diese Datei muss editiert werden, wenn neue Befehle hinzugefügt werden
5. defaultConfig.h (keine Änderung)
   - defines der Standartwerte für die Zurücksetzung der Module (XBee basierend)
   - werden weitere XBee Module hinzugefügt muss diese Datei erweitert werden
6. at_commands.h (alt: atlocal.c)
   - Prototypen für den AT Parser, der Read, Write und Exec Funktion
7. enum_cmd.h (keine Änderung)
   - Enumeration aller AT Kommandos, die von der Firmware verarbeitet werden
   - diese Datei muss editiert werden, wenn neue Befehle hinzugefügt werden
8. enum_status.h (keine Änderung)
   - Enumeration der Status- und Fehlermeldungen
9. helper.h (neu)
   - Prototypen der Hilfsfunktionen
10. rfmodul.h
    - definiert das typedef struct RFmodul (Speicherort der Parameter im Flash)
    - Prototypen der Set Funktionen
    - Prototypen der Get Funktionen
    - Prototypen für das Schreiben und Lesen zum EEPROM
11. stackrelated.h — Stack abhängig (keine Änderung)
    - Prototypenzeiger der UART-Funktionen
    - Prototypenzeiger der Transreceiver-Funktionen
    - Prototypen der Funktionen, die in anderen c-Files aufgerufen werden

12. stackdefines.h (keine Änderung)
    - Neue defines der Stack defines um die Suche und das Ersetzen zu erleichtern
13. stackrelated_timer.h (keine Änderung)
    - Prototypenzeiger der Timer-Funktionen

13.3 C-Files

1. at_api_main.c (alt: atcommands.c) — Hauptdatei mit der Main Funktion
    - liest den EEPROM aus und speichert die Daten in den Flash
    - ruft die Initialisierungsfunktionen auf
    - startet die aktive Warteschleife und ruft bei UART Eingabe je nach konfigurierten Modus den Parser auf und bei Transerceiver Eingang die Receive Funktion
2. ap_frame.c
    - Definition des api_f struct
    - Set und Get Methoden um auf das API Frame Sruckt zuzugreifen
3. ap_parser.c
    - State Machine um die API Frames zu parsen
4. ap_local.c
    - Definitionen der API Frame Typen
    - Hauptfunktion um die API Frames abzuarbeiten
    - Aufruf der Frame Typ Funktionen
        - 0x08 und 0x09 local AT commands, ist in den rwx-Funktionen der atlocal.c Datei integriert
        - 0x18 specific device commands
        - 0x17 remote AT commands, ist in der ap_trx.c ausprogrammiert
        - 0x88 local AT response
        - 0x97 remote AT response
5. ap_read.c
    - lokale Read Funktion, die durch API Frames aufgerufen wird
6. ap_trx.c
    - Transreceiver Funktionen, die durch API Frames aufgerufen wird
7. ap_write.c
    - lokale Write Funktion, die durch API Frames aufgerufen wird
8. ap_exec.c
    - lokale Execute Funktion, die durch API Frames aufgerufen wird
9. at_parser.c
    - State Machine um die AT Kommandos zu parsen
    - Aufruf der Funktionen read, write und exec für die Lokale Abhandlung

10. at_read.c
    - lokale Read Funktion, die durch AT Kommandos aufgerufen wird
11. at_write
    - lokale Write Funktion, die durch AT Kommandos aufgerufen wird
12. at_exec.c
    - lokale Execute Funktion, die durch AT Kommandos aufgerufen wird
13. tp_trx.c
    - Transreceiver Funktionen, die im Transparentmodus aufgerufen werden
14. attable.c — Tabelle mit den AT Kommandos
    - Auflistung aller Kommandos mit ihren Rechten (rwx) in einem Array
    - diese Datei muss editiert werden, wenn neue Befehle hinzugefügt werden
15. helper.c
    - Validierungsfunktionen
16. circularBuffer.c — Bufferfunktionen
    - Definition des typdef struct deBuffer_t
    - Bufferarray
    - Ausprogrammierte Bufferfunktionen
17. nvm_eeprom1.c — Speicher- und Lesefunktion für den EEPROM
    - Set User Values und Default Funktionen
    - Get Funktion
    - EEPROM crc Funktionen
18. timer0.c — definiert die Verbindung zu der Timer Funktion des Stacks
    - Initialisierung Timer Pointer
    - Definition der Millisekunden für das Modul
19. trx0.c — definiert die Verbindung zu den Transceiver Funktionen des Stacks
    - Initialisierungsfunktion für den Transreceiver
    - Sende- und Empfängerhändler
    - Funktionen um die Pakete, die gesendet werden zu packen
20. uart0.c — definiert die Verbindung zu den User Interface Funktionen des Stacks
    - Initialisierungsfunktion der UART Pointer

14 Schlusswort

Mein Praktikum umfasste sechs Monate. Während dieser Zeit konnte ich meine erlernten C-Grundkenntnisse aus den Seminaren von Professor Beck auffrischen und erweitern. Auch in den Bereichen Rechnerarchitektur und in der Programmierung von Mikrocontrollern lernte ich zu meinem schon vorhandenem Wissen noch einiges dazu. Mein persönliches Ziel war es der Firma dresden elektronik am Ende des Praktikums einen vollständig funktionierenden Prototyp zu übergeben, so dass weitere Erweiterungen einfach integrierbar sind.

Dieser Prototyp soll die AT Kommandofunktionalität der XBee Module der Firma Digi adaptieren und gegebenenfalls verbessern. Für deren Analyse standen mir drei XBee S1 802.15.4 und zwei XBee S2 ZigBee Module zur Verfügung. Durch die Auswertung der Log-Dateien von diesen Modulen erschlossen sich mir die ersten möglichen Programmabläufe und Funkprotokolle. Im späteren Verlauf erleichterte mir zudem die Programmbibliothek µracoli, durch den mitgelieferten Daten-Sniffer, die weitere Analyse der Funkprotokolle von den XBee Modulen. Aufgrund, dass die Programmbibliothek µracoli fast vollständig die Konfiguration der Mikrocontroller übernimmt, konnte ich mich größtenteils dem eigentlichen Programm widmen.

Mit den C-Kenntnissen aus den Seminaren von Professor Beck und durch die Unterstützung der Programmbibliothek µracoli, implementierte ich rasch die erste UART Verknüpfung mit den ersten rudimentären Text-Rückgaben. Durch die Dokumentation der Atmel Mikrocontroller und den Standard des IEEE 802.15.4 Protokolls, war es mir ebenfalls möglich, in kurzer Zeit die XBee Funkprotokolle zu analysieren und den ersten Datenaustausch zu realisieren.

Während der Entwicklung, wurde ich durch Bugs und fehlerhafte Programmabschnitte zeitlich und programmiertechnisch teilweise zurückgeworfen, vor allem wenn ich einen Fehler produzierte, der erst tiefer in der µracoli Programmbibliothek ersichtlich wurde. Hier wurde die Fehlerfindung einzig durch die Kompileroptimierung erschwert, denn der Kompiler tauschte viele Variablen gegen Register aus und somit war die Werteübergabe beim Debuggen in Atmel Studio nicht mehr nachvollziehbar. Dennoch wurden bei dem ersten Prototyp schnell Fortschritte erarbeitet und die Alpha Version konnte vorgeführt werden.

Nach weiteren Implementierungen hatte ich nach zweieinhalb Monaten die Version A für eine weitere Präsentation fertig. Es konnten Textnachrichten im Transparentmodus versendet und der Kommandomodus mit den Command Sequence Character, welcher ebenfalls variabel ist, aufgerufen werden. Im Kommandomodus waren bereits alle 77 Befehle der 802.15.4 Module abrufbar. Je nach Typ, konnten diese nur Werte auslesen, schreiben, oder ausführen. Realisiert wurde dies mit der Implementierung der AT Kommandotabelle, welche aufgerufen wird, wenn die eingegebenen Informationen als Befehl interpretiert werden. Sie ist Alphabetisch sortiert, um mit einem binären Suchalgorithmus den eingegebenen Befehl schnell zu finden. Ebenfalls konnte das Gerät Remote API Kommandos versenden, um Einstellungen an einen entfernten Modul vorzunehmen, sowie mit Lokalen API Frames die AT Kommandos abzurufen. Im API-Modus wurden empfangene Nachrichten als binär kodierte API Frames ausgegeben. Es war auch möglich den Sendebereich zu konfigurieren, also die Netzwerkkennung, den Kanal und die Zieladresse, sowohl in 16-bit als auch in 64-bit, zu setzen und diese Werte in den EEPROM zu speichern.

Version A konnte bereits mit dem X-CTU Programm kommunizieren und somit mit dem Programm die gespeicherten Werte abrufen, ausführen und gegebenenfalls schreiben. Durch diese Funktionen konnte der Programmablauf der XBee Module bereits simuliert werden, dennoch wurden in den kommenden Monaten verschiedene Änderungen und Neuerungen eingebracht.

Ein Faktor, der dies veranlasste, war die Optimierung. Durch die Verknüpfung der API Programmabläufe mit denen der AT Kommandos im Prototyp Version A, waren beide Objekte untrennbar und mussten zusätzliche Bedingungen und Abfragen aufrufen, welche die Geschwindigkeit des Programmablaufs reduzierten, so wie den Programmcode schwer leserlich darstellten. Eben durch diese Untrennbarkeit wäre es auch nur schwer umsetzbar gewesen, die Version A durch weitere Programmabschnitte oder anderen APIs zu erweitern.

Um den Programmablauf zu beschleunigen, besitzt Version B mehrere State Machines, welche die eingehenden Daten analysieren und an die jeweiligen Unterfunktionen weiterleiten. Zu dem wurden die API Funktionen weitestgehend von den AT Funktionen getrennt, es gibt nur noch eine gemeinsame Validierungsfunktion. Hierfür wurde die Kommandotabelle mit zusätzlichen Informationen erweitert. Dies beschleunigt nicht nur den Programmablauf, sondern reduziert gleichzeitig den Umfang der Programmkomplexität die benötigt wurden, um API vom AT Kommando innerhalb einer Funktion zu trennen. Der positive Nebeneffekt ist, dass nicht nur viele zusätzliche Bedingungsabfragen wegfielen, sondern dass auch der Inhalt der Funktionen weiter zusammengefasst werden konnte und somit die Programmkomplexität von mehreren hundert Zeilen auf zehn bis vierzig Zeilen reduziert wurde.

Zum Schluss wurden noch einige fehlende Funktionen implementiert. So war es mit Version B möglich, binär kodierte Remote API Frames zu empfangen und deren Befehle Lokal auszuführen, Remote Response Nachrichten als binär kodierte API Frames auszugeben, Textnachrichten mit binär kodierten API Frames zu versenden und dem Nutzer mit einem binär kodierten Versandstatus Frame zu informieren.

Nach diesem Praktikum kann dresden elektronik die weiteren Funktionen, wie zum Beispiel die Sleep und Verschlüsselungsfunktionen einfach integrieren. In diesem Bericht wurde die Analyse der Verschlüsselung bereits begonnen, jedoch aufgrund der fehlenden Zeit bis zum Praktikumsende nicht beendet.[41] Die ersten Erkenntnisse daraus können dennoch für die zukünftige Planung verwendet werden.

[41] vgl. Analyse Fälle mit SecurityEinstellungen auf Seite 43

Anhang

Glossar

DIY bezeichnet die Hobby-Bastler, die Privat in Handarbeit selber Produkte erstellen oder improvisieren. Die Abkürzung steht dabei für *do it yourself*.

AT Kommando steht für *attention* und ist ein Kommandosatz, der sobald er abgesendet wurde oberste Priorität besitzt.

API steht für *application programming interface*. Die Firma Digi erweitert mit ihren API Frames den Umfang der Informationen, die mit einem einzigen Kommando übergeben, oder erhalten werden können. Für weitergehende Informationen bitte im jeweilgen Digi User Guid nachschlagen oder auf folgender Homepage: http://knowledge.digi.com/articles/Knowledge_Base_Article/What-is-API-Application-Programming-Interface-Mode-and-how-does-it-work.

SMT steht für *surface mount* (oberflächenmontiert) und bezeichnet eine Variante der Aufbau- und Verbindungstechnik. Der Mikrochip wird bei diesen Verfahren auf dem Bord aufgelötet oder angeklebt.

THT oder PIH steht für *through hole* bzw. für pin-in-hole (Durchsteckmontage) und bezeichnet eine Variante der Aufbau- und Verbindungstechnik. Der Mikrochip wird bei diesem Verfahren zunächst durch Löcher im Board gesteckt und auf der Rückseite verlötet.

M2M ist die Abkürzung für *Machine-to-Machine* und beschreibt den automatischen Informationsaustausch zwischen Endgeräten mit minimalen Eingriffen durch den Mensch.

WPA steht für *Wifi Protect Access* und ist die Verschlüsselungsmethode für Wifi Netzwerke. Es gibt verschiedene Protokolltypen, die unter anderm weitere Verschlüsselungsmethoden beinhalten und miteinander verknüpfen.

MAC steht für *media access control* und ist eine vom IEEE entworfene Erweiterung des OSI-Modells. Im Fall 802.15.4 beinhaltet die MAC Schicht das Frame Access Control Bitfeld, die Zieladresse mit PAN ID, die Quelladresse mit PAN ID, einen Frame Counter und den Application Security Header.

PHY steht für *physical layer*, sie ist die funktionelle Gruppe der Bitübertragungsschicht

APL steht für *application layer*, sie ist die Schicht, die Dienste, Anwendungen und Netzmanagement zur Verfügung stellt.

SEC steht für *security layer*, sie ist die Verschlüsselungs- und Entschlüsselungsebene.

NWK steht für *netwok layer*, sie ist die Vermittlungsschicht des OSI-Modells

OSI-Modell steht für *Open Systems Interconnection Model* und ist ein ISO Standard für Netzwerkprotokolle und deren Schichtenarchitektur.

SoftAP steht für *software enabled access point*. Die Software ermöglicht ein Gerät, welches nicht spezifiziert wurde, sich in einen virtuellen Router zu verwandeln und anderen Modulen den Zugriff zu einem weiteren Teil des Netzwerks zu gewähren, oder dem Internet herzustellen.

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).

5. IEEE-SA Standards Board. (Dezember 2015). *IEEE 802.15.4 Standart*. IEEE Standard for Low-Rate Wireless Networks.
PDF: 2016-11-01_802.15.4-2015.pdf.
http://standards.ieee.org/findstds/standard/802.15.4-2015.html. (Besucht am 01.11.2016).

6. Digi. (2015). *Wireless Mesh Networking*. ZigBee vs. DigiMesh. White Paper
Grafik TopologiaZigBee.png und TopologiaDigiMesh.png aus der
PDF: 2016-09-27_wp_zigbeevsdigimesh.pdf https://www.digi.com/pdf/wp_zigbeevsdigimesh.pdf

7. Thread Group. (Juli 2014) *Press Kit*.
Grafik TG_ThreadHomeNetwork_illo_v2.jpg
http://threadgroup.org/news-events/LatestNews-New#Presskit

8. Digi. (Dezember 2008). *Wireless Sensor*. What a Mesh! Part 2-Networking Architectures and Protocols
Webpage: http://www.sensorsmag.com/networkingcommunications/wirelesssensor/whatameshpart2networkingarchitecturesandprotocols1544?print=1. (Besucht am 30.01.2017).

9. ZigBee Alliance. (May 2013). *Smart Energy Profile 2 (SEP 2)*. IP based Energy Management for the Home.
PDF: 2016-09-12_Webinar_docs-13-0233-00-0mwg-smart-energy-profile-2-zse-ip-based-energy-management-for-the-home.pdf
http://www.csee.net.cn/Portal/zh-cn/Publications/atm/docs-13-0233-00-0mwg-smart-energy-profile-2-ip-based-energy-management-for-the-home.pdf.
(Besucht am 12.09.2016).

10. Digi. (April 2016). *THREAD TECHNICAL BRIEF*.Thread Overview for Industrial and Enterprise IoT Applications.
PDF: 2016-09-27_thread-technical-brief.pdf
http://www.ie-cloud.it/web/wp-content/uploads/2016/04/thread-technical-brief.pdf
(Besucht am 27.09.2016).

11. Atmel. (August 2016). *AVR2052: BitCloud SDK Quick Start Guide*. Application Note.
PDF: 2017-02-16_Atmel-8200-BitCloud-SDK-Quick-Start-Guide_AP-Note_AVR2052.pdf.
http://www.atmel.com/tools/BITCLOUD-ZIGBEEPRO.aspx?tab=documents.
(Besucht am 16.02.2017).

Liste der verwendeten Programme und Tools

texlive https://www.tug.org/texlive/

TeXstudio http://www.texstudio.org/

Perl https://www.perl.org/get.html

X-CTU Next Gen und Legacy (unter Produkt Support)
https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu

Atmel Studio 7 http://www.atmel.com/tools/ATMELSTUDIO.aspx

Wireshark https://www.wireshark.org/

BitCatcher http://offer.luxoft.com/bitcatcher.html

StarUML http://staruml.io/

Free Device Monitoring Studio https://www.hhdsoftware.com/device-monitoring-studio

HTerm http://www.der-hammer.info/terminal/

Python https://www.python.org/downloads/ (Version 2.1.x wird empfohlen)

µracoli Version 0.4.2 http://uracoli.nongnu.org/download.html oder
Version 0.5.x http://hg.savannah.nongnu.org/hgweb/uracoli/

FTDI http://www.ftdichip.com/FTDrivers.htm

PuTTY http://www.putty.org/

Java SE http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html

XBee Java API https://github.com/digidotcom/XBeeJavaLibrary

Zusatzanlagen

Zusatzanlage 1 Ausdruck Stat-auswertung_xbee-befehle.xlsx, Tabelle Erfassung

Zusatzanlage 2 Ausdruck Stat-auswertung_xbee-befehle.xlsx, Tabelle Ergebnisse

Zusatzanlage 3 DVD mit

- PDF Ausgabe des Berichtes
- Excel Datei Stat-auswertung_xbee-befehle.xlsx
- BitCatcher Log Datei xbS1-log.dcf
- BitCatcher Log Datei xbS2-log.dcf
- Wireshark Log Datei 2016-12-20_xb-s1_log.pcapng
- Wireshark Log Datei 2016-12-21_xb-s1_log-txTransmit.pcapng
- Wireshark Log Datei 2016-12-22_xb-s1_log-AES.pcapng
- Wireshark Log Datei 2016-12-22_xb-s1_log-AES2.pcapng