

Diplomarbeit

Entwurf und Validierung eines Planungssystems für das Freescale BeeKit mit dem Systementwurfswerkzeug MLDesigner

Juli 2007

Diplomarbeit zur Erlangung des akademischen Grades Diplom-Informatiker

Bearbeiter: Name: Frank Lohse
Matrikelnummer: 33750
Studiengang/Matrikel: Informatik 2001

Verantwortlicher Hochschulprofessor: Prof. Horst Salzwedel

Betreuer an der TU Ilmenau: Dipl.-Inf. Tommy Baumann

Betreuer bei SenTec Elektronik GmbH: Dipl.-Inf. Michael Binhack

Inventarisierungsnummer: 2007-06-18/079/IN01/2236



Vorwort

Nach Abschluss aller Prüfungen, Hauptseminare und der Studienarbeit bildet diese Diplomarbeit das erfolgreiche Ende meines Informatikstudiums an der TU Ilmenau. Das Thema einer Modellierung von drei verschiedenen Funkdatenübertragungstechnologien mit einer dazugehörigen Performancebewertung ist gleichermaßen aktuell wie anspruchsvoll. Die Modellierung in dem universitätseigenen Tool MLDesigner erforderte ein genaues Fachwissen über die Funksysteme, Verständnis zum Systementwurf und Programmierkenntnisse in C++.

An dieser Stelle möchte ich bei meinen beiden Betreuern Herrn Dipl.-Inf. Tommy Baumann und Herrn Dipl.-Inf. Michael Binhack bedanken, die mir immer mit Rat und Tat zur Seite standen. Besonderer Dank gilt ebenfalls der Firma SenTec, die mir mit der spezifischen Hardware die nötige Grundlage für diese Diplomarbeit zur Verfügung stellten.

Die Diplomarbeit widme ich in erster Linie meinem Vater, der voll und ganz hinter mir stand und mich aufopfernd unterstützt hat, um mein Studium schnellstmöglich beendet sehen zu können.

Inhaltsverzeichnis

1	Einleitung	5
1.1	Motivation	5
1.2	Zielsetzung	6
1.3	Gliederung der Diplomarbeit	7
1.4	Das Unternehmen Freescale Semiconductor	9
1.5	Das Unternehmen senTec Elektronik GmbH	10
1.6	Das Unternehmen Mission Level Design GmbH	12
2	Das BeeKit	13
2.1	BeeKit Wireless Connectivity Toolkit	13
2.2	Kenndaten des BeeKit	14
2.3	Aufbau des BeeKit	15
2.3.1	SMAC (Simple Access Media Controller)	17
2.3.2	IEEE 802.15.4	18
2.3.3	BeeStack (ZigBee)	25
3	Vorüberlegungen zur Modellierung	30
3.1	Modellierungsbausteine	31
3.2	Umsetzung der Topologien	32
3.3	Parameter der Modellierungsbausteine	34
3.3.1	Parameter des Channels	35
3.3.2	Parameter des Funktionsbausteines	36
3.3.3	Parameter des Interfaces	36
3.4	Bestimmung der Entfernung	36
3.4.1	Entfernung als direkter Wert	37
3.4.2	Entfernung mit Hilfe der Gerätekoordinaten	38
3.5	Berechnung der Datenrate	39
3.5.1	Datenratenberechnung mittels Pfadverlust	40
3.5.2	Datenratenberechnung durch Messung	42
3.5.3	Datenratenberechnung durch Paketanzahl	43
3.6	Berechnung des Delays bei einer Übertragung	43
3.6.1	Bestimmung der Frameanzahl	44
3.6.2	Bestimmung der Übertragungszeit	45
4	Die Modellierung	47
4.1	Mission Level Design	47
4.2	MLDesigner	48
4.3	Allgemeines zum Modellaufbau	53
4.3.1	Grundaufbau des Modells	54
4.3.2	ESL-Umgebung	55
4.4	Grundaufbau der Netzwerkgeräte	59

4.5	Grundaufbau des Protokollstack	61
4.5.1	Protokollstack bei IEEE 802.15.4/ZigBee	62
4.5.2	Protokollstack bei SMAC	65
4.6	Schichten	66
4.6.1	Schichten des IEEE 802.154/ZigBee-Protokollstacks	66
4.6.2	Schichten des SMAC-Protokollstacks	79
4.7	Frames	82
4.7.1	AppFrame der Anwendungsschicht	83
4.7.2	NWKFrame der Netzwerkschicht	84
4.7.3	MACFrame	85
4.7.4	ChannelPacket	86
4.8	Channel	86
4.9	Modellierungsunterschiede der Protokolle	89
5	<i>Praktische Validierung am Echtssystem</i>	91
5.1	Versuchsaufbau	93
5.2	Durchführung	94
6	<i>Zusammenfassung und Fazit</i>	99
7	<i>Ausblick</i>	101
8	<i>Anhang</i>	102
8.1	Abkürzungsverzeichnis	102
8.2	Abbildungsverzeichnis	103
8.3	Tabellenverzeichnis	106
8.4	Literaturverzeichnis	107
	<i>Eidesstattliche Erklärung</i>	111
	<i>Thesen</i>	112

1 Einleitung

In einem Zeitalter geprägt von revolutionären, technischen Neuerungen und einer rasanten Weiterentwicklung auf dem Funktechnologiesektor, hat die Forschung auf diesem Gebiet einen hohen Stellenwert. Nicht nur Politik und Wirtschaft, sondern auch Forschungseinrichtungen und Universitäten halten die verschiedenen Funksysteme mit ihrer engagierten Arbeitsweise und den neuen Ideen auf einem aktuellen Stand. Sie erweitern derzeitige Technologien mit dem Ziel, vor allem die tägliche Nutzung und das Handling durch sinnvolle Funktionen zu erleichtern.

1.1 Motivation

Für die Kommunikation von eingebetteten, mobilen oder stationären Systemen wird zunehmend die digitale Funkdatenübertragung eingesetzt. Technologien wie z.B. WLAN, UMTS, Bluetooth, ZigBee oder auch andere proprietäre Protokolle (SMAC, WaveBird, Wi-Fi, WiMAX, Hiperlan, ...) wurden für die verschiedenen Einsatzzwecke bis zur Marktreife entwickelt.

Für den Entwickler eingebetteter kommunizierender Systeme stellt sich das Problem, welche dieser Technologien im Hinblick auf technische und ökonomische Erfordernisse am zweckmäßigsten für den vorgesehenen Einsatz ist. Um die Leistungsparameter von den diversen Funktechnologien in frühen Entwicklungsphasen von Systemen einschätzen zu können, werden abstrakte Modelle der technischen Kommunikation benötigt.

Hierfür ist die Verwendung von standardisierten Schnittstellen wichtig, um einen schnellen Austausch verschiedener Übertragungsprotokolle zu ermöglichen. Des Weiteren sollten schon in sehr frühen Entwurfsphasen Aussagen über die Performance getroffen werden können. Als Voraussetzung dafür muss das verwendete Modell aber mit den in der Realität vorkommenden Umgebungsverhältnissen konform sein.

1.2 Zielsetzung

Für die Kommunikationsprotokolle SMAC (Simple Media Access Controller) und IEEE 802.15.4/ZigBee sind abstrakte, wieder verwendbare Modellelemente mit dem Systementwurfswerkzeug MLDesigner zu entwerfen, welche Vorabanalysen über geplante Systeme ermöglichen sollen.

Als Basis für den Entwurf dienen standardisierte Modellelemente aus dem in MLDesigner bereits vorhandenen ArchitectureBlockSet. Damit wird eine gemeinsame Performanceanalyse mittels eines Ausgabemonitors, die einfache Austauschbarkeit der Modellbestandteile und eine spätere Optimierung der Architektur erreicht.

Das entworfene Modell dient später als Grundlage zur Planung von Netzwerken und deren Untersuchung. Über Analysewerte wie z.B. Zeiten mit Maximalauslastung, mittlere Auslastung, Übertragungszeit oder versendete Datenmenge, Aussagen über die Machbarkeit von Netzwerken in Bezug auf ein vorgegebenes Datenaufkommen und Sendeschema zu treffen.

Dazu soll das vereinfachte, ausführbare (abstrakte) Modell die wichtigsten Eigenschaften der jeweiligen Funkübertragungsmethoden wiedergeben. Die Modellkomponenten der einzelnen Architekturen sollen zur leichten Wiederverwendbarkeit in einer Modellbibliothek zusammengefasst und dokumentiert werden.

Mit Hilfe des ESL-Target, welches eine standardisierte Schnittstelle für den Modellentwurf und die Analyse der Architektur darstellt, und dem ArchitectureBlockSet, soll eine einheitliche, später frei austauschbare Protokollstruktur ermöglicht werden. Dafür stehen neben einem standardisierten Interface zur Datenstrukturübergabe, auch der Channel, welcher die Daten weiterleitet, und eine vorgefertigte Ausgabematrix für Auswertungszwecke zur Verfügung. [BHS07]

Der erste Entwicklungsschritt beschränkt sich auf eine einfache Peer-to-Peer-Verbindung von zwei Geräten. Die modellierte Datenübertragung bleibt von Zusatzfunktionalitäten, wie Sicherheit, Empfangsbestätigungen (Acknowledgements), Fehlübertragungen, etc., befreit. Eine mögliche Realisierung zur einfachen

Veranschaulichung könnte z.B. eine Heizungs- oder Lichtanlagensteuerung sein, die im Wesentlichen einfache Bit-Kommandos verschickt und interpretiert.

Die vom Modell generierten Messwerte (Übertragungszeitpunkte) sollen in einem Logging-Block mitgeschrieben und zur Laufzeit veranschaulicht werden. Die Ausgabeparameter können die aktuelle Geräteauslastung, die Channellast, die Größe der empfangen Daten oder auch der Verlinkungsgrad der Geräte sein. Hier bietet sich die Form einer Ausgabematrix an.

Nach Erstellung der Protokollmodelle ist ein Anwendungsbeispiel als MLDesigner-Modell und in Realität zu erstellen. Für die Realisierung des gewählten Beispiels kommt der von Freescale entwickelte BeeKit zum Einsatz, der auf einem ZEBRA-Board der Firma senTec Elektronik GmbH integriert ist. Er beinhaltet Übertragungstechnologien auf Basis des IEEE 802.15.4-Standards und ermöglicht so den Einsatz von SMAC und IEEE 802.15.4/ZigBee auf einem einzigen Modul bzw. einer Plattform. Anhand des Beispielmodells und des realen Datenübertragungsbeispiels sind die modellierten Protokolle zu validieren. Für einen Vergleich zwischen Realitätsverhalten und Softwaremodell muss die Datenübertragung vereinheitlicht werden. Dazu gehören einheitliche Paketgrößen und Versendezeiten.

1.3 Gliederung der Diplomarbeit

Kapitel 1 bildet eine kurze Einführung und gibt einen Einblick in die Problemstellung und deren Umsetzungsvorgaben. Zusätzlich werden die drei Unternehmen Freescale Semiconductor Inc., SenTec Elektronik GmbH und Mission Level Design GmbH kurz vorgestellt.

In Kapitel 2 werden die theoretischen Grundlagen für den BeeKit von Freescale erläutert und dabei auf die drei integrierten Funkdatenübertragungen SMAC, IEEE 802.15.4 und das darauf aufbauende ZigBee eingegangen. Neben den Eckdaten zur Leistung und den Verbindungsmöglichkeiten, wird der Kommunikationsfluss herausgearbeitet und der dafür notwendige Datenrahmen

beschrieben. Den Abschluss dieses Abschnitts bildet eine kurze Erläuterung zu dem verwendeten Schichtenmodell und den darin integrierten Dienste.

Das nachfolgende Kapitel analysiert die Problematik der Modellierung und geht auf unterschiedliche Lösungsmöglichkeiten zur Realisierung ein. Inhaltlich gehören dazu unter anderem Algorithmen zur Bestimmung der Datenrate, die Entfernungsbeeinflussung der Sendemodule untereinander, Ansätze zur Umsetzung anderer Netzwerktopologien, Speicherung bzw. Veranschaulichung möglicher Performanceparameter und die eigentlichen Paketübertragung.

Der einleitende Teil von Kapitel 4 definiert den Begriff Mission Level Design und führt kurz in das Modellierungstool MLDesigner ein. Erläuterungen zu dem Hierarchiekonzept, den wesentlichen Grundbausteinen für den Aufbau eines Modells, der Datenweiterleitung zwischen Blöcken und den Simulationsmöglichkeiten verschaffen einen kleinen Einblick in die funktionellen Möglichkeiten des Tools. Der größte Teil dieses Kapitels befasst sich mit der eigentlichen praktischen Umsetzung der Übertragungsprotokolle. Dazu gehören die Realisierung der Geräte, des jeweiligen Protokollstacks, des Channels, der Daten- und Paketgenerierung, der Berechnung des Delays und der Ausgabe von Simulationsergebnissen. C++ als verwendete Programmiersprache in MLDesigner bildet die Grundlage Hardwareabläufe mittels Software nachzubilden.

Die Validierung mit Hilfe von echter Hardware erläutert Kapitel 5. Hier werden der Versuchsaufbau, alle verwendeten Gerätschaften zur Zeitmessung und Datenerfassung, sowie die Durchführung beschrieben. Zusätzlich befasst sich dieses Kapitel mit der Auswertung der Simulationsergebnisse und vergleicht diese mit den Testergebnissen der Validierung. Dazu gehören auch mögliche Änderungen der Simulationsparameter oder Veränderungen an den Algorithmen der Blöcke.

Das 6. Kapitel beendet den eigentlichen Ausarbeitungstext mit einer kurzen Zusammenfassung und einem Fazit der Diplomarbeit

Das vorletzte Kapitel gibt einen kurzen Ausblick auf mögliche Erweiterungen nachfolgender, aufbauender Arbeiten.

Den Anhang bildet das letzte Kapitel, in dem das Abkürzungs-, Abbildungs-, Tabellen- und Literaturverzeichnis, sowie die Thesen und die eidesstattliche Erklärung enthalten sind.

1.4 Das Unternehmen Freescale Semiconductor

Freescale Semiconductor, Inc. ist der Weltführer im Design und der Fertigung von eingebetteten Halbleitersystemen für Radio, Netzwerktechnik, Automotive, Verbraucher- und Industriemärkte. Das Unternehmen versorgt andere Hersteller mit Halbleitertechnik und Chips, um ihnen bei der Entwicklung und Herstellung fortschrittlicher Mobiltelefone zu helfen, den Internetverkehr zu koordinieren und z.B. auch Fahrzeugtechnik sicherer, sowie energieeffizienter zu gestalten. Freescale hat mehr als 10.000 Kunden zu denen auch 100 der globalen Spitzenhersteller gehören.



Mit einem Jahresumsatz von 5,8 Millionen Dollar in 2005 ist Freescale Mitglied der S&P 500® und des Philadelphia Semiconductor Index. Das Unternehmen gilt als drittgrößter Chiphersteller in den USA und als neuntgrößter weltweit.

Mit ihren umfassenden Neuerungen könnte Freescale einer der größten Gesellschaften sein, von denen der Mensch täglich Produkte in der Hand hat, aber noch nie den Markennamen gehört hat. Freescale hat bis heute mehr als 17 Milliarden Halbleiter aufgeliefert, die in vielen Produkten wieder gefunden werden können.

- Motorola Mobiltelefone
- Sony Elektronik
- Whirlpooltechnik
- Logitech Keyboards und Mäuse
- Lifefitness Cardio- und Trainingsgeräte
- Cisco Router
- Bose Soundsysteme und Radios

- Trane Klimaanlage
- Fahrzeuge von Mercedes, BMW, Ford, Hyundai und General Motors

Freescall ist Marktführer in vielen Bereichen, die das Unternehmen abdeckt.

- Nr. 1 bei automotive semiconductors – Gartner
- Nr. 1 bei communications processors – Gartner
- Nr. 2 bei overall microcontrollers – Gartner
- Nr. 2 bei digital signal processors – Forward Concepts
- Nr. 4 bei wireless handset radio frequency microprocessors – iSuppli

Freescall hat Designabteilungen, Forschung und Entwicklung (R&D – Research and Development), Fertigungs- oder Verkaufsstellen in mehr als 30 Ländern. Dazu gehören sieben Waferfabriken, 2 Montage- und Prüfstellen, eine 300mm-Pilotlinie und ein R&D-Zentrum in Crolles, Frankreich, welches gemeinsam mit Philips und STMicroelectronics genutzt wird. Freescall investiert jährlich 1 Milliarde Dollar in R&D und hält 5.500 Patente.

Nach mehr als 50 Jahren als Teil von Motorola fing Freescall ein neues Leben als eine eigenständige Gesellschaft im Juli 2004 an. Unter Führung des Vorsitzenden und CEO Michels Mayer hat sich die Firma seitdem auf die Verbesserung der Finanzlage, das Wiederbeleben der Firmenkultur und das Aufbauen einer globalen Marke konzentriert. [FS06]

1.5 Das Unternehmen senTec Elektronik GmbH

senTec Elektronik GmbH ist ein junges und innovatives Unternehmen, das sich vor allem mit Anwendungen der Sensorik und Funkübertragung beschäftigt.



Dabei stellt senTec nicht nur Dienstleistungen für eingebettete Systeme, Sensoranwendungen, Hardware- und Softwareentwicklung zur Verfügung, sondern pflegt und entwickelt aktuelle Systeme ständig weiter. Das Unternehmen

wurde 1999 gegründet und hat seinen Firmensitz in Ilmenau. Ebenfalls wird großen Wert auf den Kontakt zur Technischen Universität von Ilmenau gelegt. Das hilft, die richtigen Werkzeuge und Mittel auszuwählen, die die Anwendungen und Entwicklungen unterstützen. Zusätzlich fördert senTec motivierte Studenten mit Praktikanten- und Diplomarbeitsangeboten.

Das Unternehmen senTec realisiert komplette Gerätesysteme von der Sensorik über die Signalverarbeitung bis hin zur Visualisierung der Ergebnisse und Betriebszustände.

Die Kompetenzen liegen bei der analogen Signalvorverarbeitung, der analog-digital Wandlung, der Signalverarbeitung auf Mikrocontrollern und DSP's sowie der Visualisierung der Daten auf dem PC.

Im Bereich der Sensor-Systeme arbeitet senTec seit einiger Zeit an dem Entwurf eines ZEBRA-Moduls (ZigBee Enabled Board for Radio Applications) für aktive, drahtlose Sensor-Systeme, einem Verfahren zur Temperaturmessung auf Basis von passiven, drahtlosen Sensoren und an IQ-Pin Anlagesensoren zur sicheren Lagebestimmung und Fixierung von Werkstücken.

Für die Entwicklung solcher Systeme und Plattformen spielt der Systementwurf eine wichtige Rolle. senTec kann dabei auf ein hohes Maß an Systemerfahrung, sowie langjährige Hardware- und Softwareentwicklung zurückgreifen. Dabei kommen bei Hardwareanwendungen Analyseprogramme, Debugger oder auch Emulatoren zum Einsatz, während bei Softwarerealisierungen viel mit OpenSource-Lösungen wie z.B. bei Linux, sowie populären Tools und Programmiersprachen (z.B. Assembler, C, C++, Delphi, SQL, ...) gearbeitet wird. Ein weiterer wichtiger Entwicklungszweig ist das COBRA-Modul (ColdFire Board for Rapid Applications) als Nachfolger der legendären 68k Serie, welches auf Anwendungen im Embedded-Bereich optimiert wurde. senTec Elektronik hat mehrere Prozessor-Module mit den ColdFire-Prozessoren MCF5272, MCF5282, MCF5485 und MCF5329 entwickelt, die sich leicht in verschiedene Applikationen integrieren lassen. Darüber hinaus existiert ein BDM-Interface, mit dem man erstellte Software debuggen und das Flash-ROM auf den Modulen programmieren kann. [Sen06]

1.6 Das Unternehmen Mission Level Design GmbH

Die in Ilmenau ansässige Firma



Mission Level Design GmbH ist ein Tochterunternehmen der MLDesign Technologies, Inc., welche ihren Sitz im Herzen des Silicon Valley hat. Der Aufgabenbereich erstreckt sich hauptsächlich über das Gebiet des Systementwurfs. Hier wird vor allem versucht, Verbesserungen der Effektivität beim Designprozess zu erreichen – angefangen vom eigentlichen Konzept, über die Implementierung, bis hin zur praktischen Durchführung.

Die Problematik des Systemdesigns und -entwurfs wird zunehmend wichtiger für Unternehmen, die an der Herstellung und dem Design technischer Anwendungssysteme arbeiten. Sie investieren einen immer größer werdenden Prozentsatz in die Entwicklungszeit für das System Level Design, denn 80% der Leistungsfähigkeit eines Systems wird durch die funktionale Spezifizierung und die verwendete Architektur bestimmt. Daher sollte im Vorfeld eine Analyse zur Optimierung und Fehlerfindung erfolgen. Dabei ist es von Vorteil, wenn Hard- und Software bzgl. der Funktionsweise, den Schnittstellen und der grundlegenden Architektur genau spezifiziert und standardisiert sind. Traditionelle Systementwurfswerkzeuge kommen hier schnell an ihre Grenzen, da sie Planungen der Modellarchitektur und deren Analyse nicht unterstützen.

Mit Hilfe ehemaliger Mitarbeiter aus Projekten, wie BONes, SatLab oder SPW, entwickelte MLDesign Technologies, Inc. das Systementwurfswerkzeug MLDesigner, welches im November 2000 veröffentlicht wurde. Dort sind neben traditionellen Werkzeugen, wie Funktionen und Missionen, erweiterte Analysemöglichkeiten integriert, mit denen Entwicklerteams ihre Systeme bewerten, ändern und optimieren können.

Vor allem für Schlüsselmärkte wie der drahtlosen Kommunikationsbranche, Breitbandnetzwerke, eingebettete Computersysteme, Netzwerkprozessoren oder System-on-Chip ist Mission Level Design interessant. [MLDG07]

2 Das BeeKit

Im Mai 2006 gab Freescale einen Softwarestack und ein Development-Kit für ZigBee-Anwendungen bekannt.

Das so genannte BeeKit zur Unterstützung drahtloser Kommunikation beinhaltet den Freescale-eigenen BeeStack sowie ein Konfigurationstool. Es stellt damit eine gebrauchsfreundliche Schnittstelle und ein konformes Regelwerk für Entwickler zur Verfügung, um den Rahmen für ihre Punkt-zu-Punkt-, IEEE 802.15.4- und ZigBee-Anwendungen zu konfigurieren. Unterstützt werden die zur ZigBee-Spezifikation konformen und auf der HCS08-MCU basierenden Freescale-Plattformen. Die Codebasis schließt auch vorkonfigurierte ZigBee-Anwendungsbeispiele und -Schablonen ein. Der BeeStack wird mit der aktuellen und den folgenden ZigBee-Generationen völlig kompatibel sein, welches dann schon ein Home-Control-Stack sein wird, der ein Home-Automation-Profil enthält. Die geplanten Aktualisierungen schließen eine Unterstützung für 802.15.4 MAC, 802.15.4-basierte Anwendungen und auf SMAC (Simple Control Access Controller) beruhende Anwendungen ein. Hat der Entwickler die Konfiguration seiner Funklösung abgeschlossen, so kann er diese mit dem BeeKit direkt in das CodeWarrior IDE (Integrated Development Environment) von Freescale exportieren und damit das eigentliche Projekt erstellen bzw. debuggen.

Das BeeKit ist seit dem 4. Quartal 2006 verfügbar, kostet mit dem dazugehörigen BeeStack ca. 1000 US-Dollar pro Lizenz und schließt eine unbegrenzte Nutzung der SMAC- und 802.15.4-Codes ein. [Ucp06]

2.1 BeeKit Wireless Connectivity Toolkit

Das BeeKit Wireless Connectivity Toolkit ist eine eigenständige Softwareanwendung, die eine grafische Benutzerschnittstelle (GUI) zur Verfügung stellt, in dem der Benutzer Netzwerklösungen erstellen, modifizieren, sichern und erweitern kann. Als Grundlage dienen dabei SMAC (einfacher MAC-Zugriff), IEEE 802.15.4 PHY/MAC und der ZigBee-Protokollstack.

Zusätzlich stellt das BeeKit ein Hilfetool und vorkonstruierte Lösungskonzepte zur Verfügung, die dem Benutzer einen Rahmen schaffen, in dem er zu Beginn des Projekts schnell und leicht bestimmte Parameter manuell konfigurieren kann, ohne dass durch aufwendiges Suchen in Dateien, Quellcodes und Formulare wertvolle Zeit verloren geht.

Mit einer umfassenden Codebasis von Netzwerkanschlussbibliotheken, Anwendungsschablonen und Beispielanwendungen erzeugt sich der Benutzer passende Datensätze, die anschließend in eine einheitliche Entwicklungsumgebung (IDE) importiert werden und danach für fortlaufende Entwicklungen und Debugging genutzt werden können.

Das BeeKit selber ist leicht erweiterbar, um neue Codebestandteile und Funktionalität für eine spätere Nutzung zu integrieren. [FSB06]

2.2 *Kenndaten des BeeKit*

Operating system support	Target Processors Architectures Supported
Windows 2000 (SP3) Windows XP (SP1/SP2)	HCS08 Family MC1321x Family
Freescale Networking Protocol Codebase	Development Kits Supported
Simple MAC (SMAC) IEEE 802.15.4 BeeStack - ZigBee Protocol Stack	13192DSK 13192EVB 13192EVK 1321xDSK 1321xNSK 1321xEVK M68EVB908GB60E plus 13192RFC or 13202RFC
Application Samples	Integrated Development Environment Support
SMAC Networks IEEE 802.15.4 Networks ZigBee Networks	CodeWarrior Development Studio for HC(S)08, v.5.x

Tabelle 2-1: Eigenschaften des Beekit [FSB06]

Voraussetzung für einen problemlosen Einsatz des Toolkits ist Windows XP oder Windows 2000. Neben wichtigen Codestücken für die drei Übertragungsformen SMAC, IEEE 802.15.4 MAC und ZigBee sind für diese auch entsprechende Anwendungsbeispiele integriert, die den Einsatz erheblich erleichtern. Die unterstützten Development-Kits erweitern den Einsatzbereich enorm. Unterstützt wird die HCS08 Prozessorfamilie, die sich durch hohe Rechenleistung bei gleichzeitig geringem Stromverbrauch auszeichnet. Andere Prozessoren von Motorola wie z.B. MC9S08GT60 oder MC1321x-Prozessoren können ebenfalls für Anwendungen des BeeKits herangezogen werden. Da das BeeKit aber eine reine Softwareentwicklung darstellt, sind Implementierungen auf anderen Modulen ebenfalls möglich – z.B. das ZEBRA-Modul von senTec. [FSB06], [FSW06]

2.3 Aufbau des BeeKit

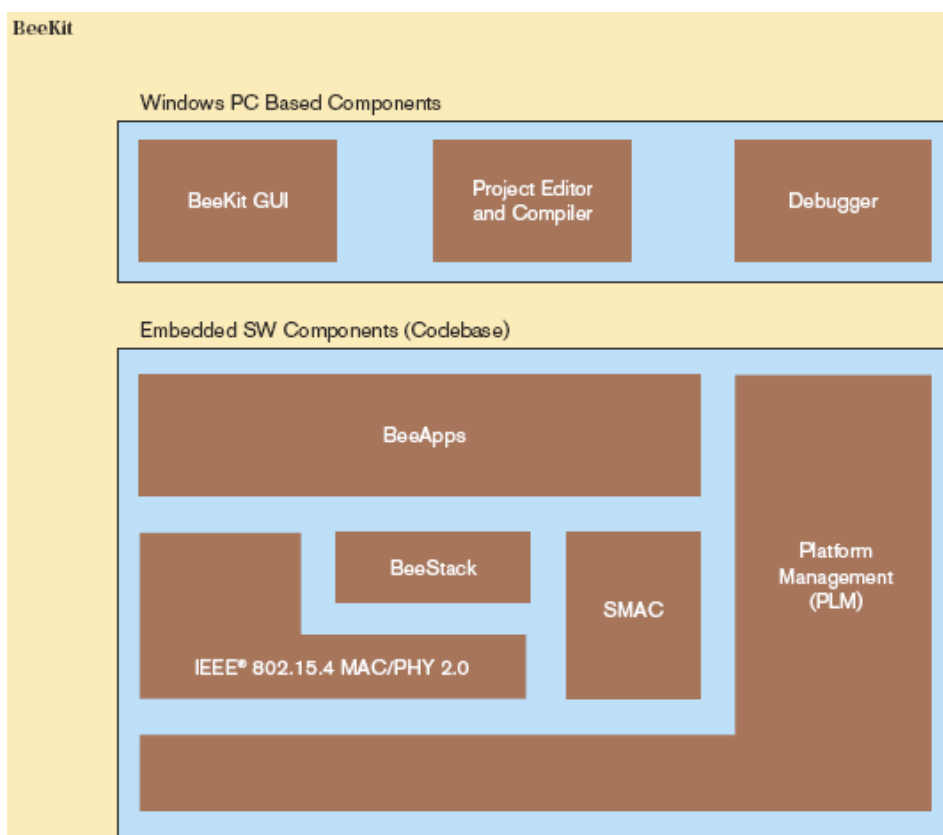


Abbildung 2-1: Aufbau des Beekit [FSB05]

Das BeeKit besteht aus zwei wesentlichen Teilen. Der erste Block sind die für Windows nötigen Komponenten GUI, Projekteditor und -compiler, sowie ein Debugger. Für die eigentliche Implementierung des gewünschten Protokollstacks stehen der PLM (Platform Management), die drei verschiedenen Übertragungsformen und die BeeKit-Anwendungen als Softwarekomponenten zur Verfügung. Die dabei verwendete Codebasis ist eine Eigenentwicklung von Freescale in Form einer architekturenspezifischen Datenbank, die neben einer umfassenden Sammlung von Bibliotheken zu drahtlosen Übertragungstechnologien, unterschiedlichster Plattformsoftware und vielen Anwendungsschablonen auch Beispielanwendungen für SMAC, IEEE 802.15.4 MAC sowie BeeStack (ZigBee) enthält.

Das BeeKit GUI ermöglicht dem Benutzer seine Arbeitsumgebung frei zu gestalten. Mit Auswahl der Zielplattform, der Endgeräteansteuerung, den gewünschten Anwendungen, der Netzwerkparameter etc. hat er die Möglichkeit, das komplette Projekt zu erstellen und zu verwalten.

Im Projekteditor können sämtliche Quelldaten editiert und durch zusätzliche Daten erweitert werden. Der Compiler wandelt diese Daten in eine für die gewählte Zielhardware verständliche, binäre Form um.

Für einen reibungslosen Ablauf der Anwendung sorgt ein Debugger, der mittels Hardwareinterface die vorhandenen Daten auf Fehlerfreiheit überprüft.

Um dem Benutzer die Entwicklung von Anwendungen so einfach wie möglich zu machen, wurden Referenz- und Demonstrationsbeispiele beigelegt, die als Ausgangspunkt für die Anwendungsentwicklung genutzt werden können und die integrierten Übertragungsverfahren unterstützen.

Der Plattformmanager organisiert die Stromsparoptionen, das zeitliche Verhalten über Timermodule, die LED-/LCD-Ansteuerung und mittels NVM-Manager die Netzwerksicherheit und -verwaltung. [FSB06]

2.3.1 SMAC (Simple Access Media Controller)

Entwickelt wurde SMAC von Freescale, um neben ZigBee eine sehr einfache und unkomplizierte Datenübertragung auf dem weit verbreiteten 2,4 GHz ISM-Band zu ermöglichen.

SMAC ist Bestandteil des von Freescale im 4. Quartal 2006 veröffentlichten BeeKit und hat aktuell die Version 4.1.

Durch die Einfachheit von SMAC beschränkt sich der Anwendungsbereich auf ein überschaubares Spektrum. Neben Sensor-Aktor-Aufgaben können mit SMAC auch Remote-Control-Anwendungen, Demonstrationen von Energieschemas oder Verfahren bei Knopfzellengeräten veranschaulicht werden.

Die möglichen Topologien beschränken sich auf proprietäre Punkt-zu-Punkt- oder Sternnetzwerke mit einer einfachen bidirektionalen Kommunikationsverbindung für Senden und Empfangen (Tx – Transmit/Rx – Receive). Mit dem kleinen Flash von 2 kByte und dem kleinen RAM von 10 Byte (exklusive der maximalen Paketgröße von 133 Byte) wird der Speicherbedarf weit unter 4 KByte gehalten. Durch den schonenden Umgang mit Ressourcen wie Energie, Hardware etc. bleiben dem Entwickler mehr Möglichkeiten und mehr Raum sich auf seine Anwendung zu konzentrieren. Zusätzlich werden weder Lizenzen oder Lizenzgebühren fällig, was SMAC für finanzschwache und kostensparende Bereiche interessant macht.

Die mitgelieferten Codestücke sind gut kommentiert und dokumentiert. So können Beispielanwendungen leicht eingebunden und fertige Softwareimplementierung problemlos auf die Prozessoren der Serie MC13192 von Motorola portiert werden und sind mit der Version 2.0 voll kompatibel. Die Datenstrukturen selber unterstützen eine einfache Übertragung von SMAC-Anwendungen auf andere Architekturen. Um zusätzlich Ressourcen zu sparen wird der MC1381x Pakettransfer genutzt. Ein mitgelieferter Bootloader erlaubt zudem das sichere Laden von bereits angefertigten Anwendungen.

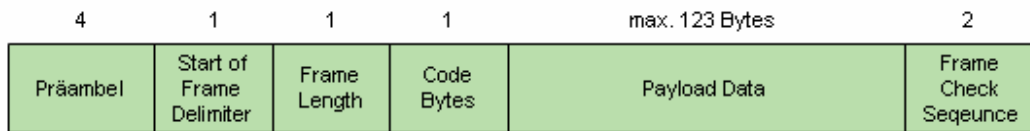


Abbildung 2-2: Data Frame bei SMAC [FSS05]

Der Aufbau des SMAC-Frames beschränkt sich auf einen bis zu 123 Byte langen Datenpayload, einer FCS zur Fehlerkorrektur und 2 Codebytes für die Möglichkeit SMAC-Pakete zu kennzeichnen, um sie so anderen Protokollen unzugänglich zu machen oder selbst nicht gekennzeichnete Pakete von Stackseite her zu ignorieren. Des Weiteren werden die Datenpakete nicht zwischengespeichert oder bei Verlust erneut übertragen, was eine genaue Planung und Analyse des Szenarios erforderlich macht.

Die Ruhephasen sind dabei eher rudimentär gehalten – der Wunsch in den Ruhemodus oder Wachmodus zu wechseln, wird durch ein optionales Management des Moduls vollzogen. So können die Geräte z.B. wahlweise immer im Wachmodus verbleiben oder in bestimmten Abständen ruhen.

Als Supporttool kommt Metrowerks CodeWarrior Experimental Edition zum Einsatz. Sicherheitsvorkehrungen wurden bei SMAC gänzlich vernachlässigt. [FSS05]

2.3.2 IEEE 802.15.4

Die im November 2000 gegründete IEEE 802.15 TaskGroup 4 sollte einen Übertragungsstandard entwickeln, der für Anwendungen mit geringen Datenraten konzipiert ist, eine mehrmonatige bzw. sogar mehrjährige Batterielebensdauer ermöglicht, eine sehr geringe Komplexität aufweist und der kein oder nur marginales QoS unterstützt. Die Übertragungstechnik soll für ein international lizenzfreies Band verfügbar sein.

Bereits im Mai 2003 verabschiedete die TG4 die erste Version des IEEE-Standards 802.15.4. Ein knappes Jahr später formierte sich im März 2004 die neue Task Group 4b und die alte TG4 wurde daraufhin aufgelöst. Die Grundidee für die im März 2005 veröffentlichte Spezifikation 802.15.4a bleibt die Gleiche; neben

geringen Kosten, einem niedrigen Stromverbrauch und der Bereitstellung von Kommunikationsressourcen wird vor allem die Gerätelokalisierung, der Datendurchfluss, die Datenskalierbarkeit und die Reichweite verbessert. Zusätzlich wurde die Spezifikation des Basisbandes (PHY) verändert und ermöglicht nun den Gebrauch von zwei optionalen Methoden zur Übertragung – UWB Impulse Radio (lizenzfrei im Ultra Wideband) und Chirp Spread Spectrum (lizenzfrei im 2,4 GHz ISM-Band). Die nachfolgenden Arbeiten der TG4b am 802.15.4-Standard wurden mit der im September 2006 veröffentlichten neuen Revision 802.15.4b abgeschlossen und machen die ursprüngliche Version überflüssig. Die Verbesserungen bzgl. der alten Version des 802.15.4-Standards bezogen sich hauptsächlich auf das Beseitigen von Unstimmigkeiten (Bugs), der Reduzierung der Komplexität, der Erhöhung der Flexibilität beim Gebrauch von Sicherheitsschlüsseln oder auch der Berücksichtigung von möglicherweise neu verfügbaren Frequenzbereichen. Die übergeordnete Task Group 802.15 wurde im Januar 2006 aufgelöst. [TG406]

Potenzielle Anwendungen sind Sensoren, interaktive Spielsachen, Fernbedienungen, sowie Haus- und Gebäudeautomation. Zusätzlich soll sichergestellt sein, dass auch latenzkritische Anwendungen wie Mäuse oder Joysticks zum Einsatz kommen können.

Der IEEE 802.15.4-Standard ist ein standardisiertes Protokoll für die zuverlässige Übertragung von Daten und die dafür nötigen Grundlagen der PHY- und MAC-Schichten.

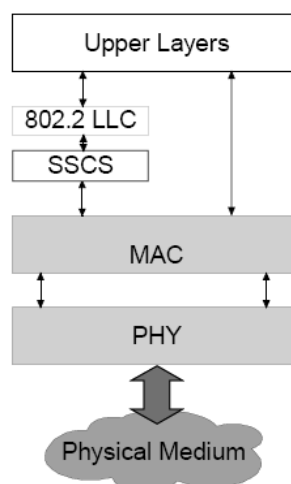


Abbildung 2-3: IEEE 802.15.4-Schichtenmodell [ICS03]

Die PHY (Physical Layer) ist für folgende Aufgaben verantwortlich:

- Koordinierung des Zugriffs auf die Luftschnittstelle zusammen mit dem MAC
- Aktivierung und Deaktivierung des Sendemoduls (Tx/Rx und Sleep)
- Energiemessungen in ausgewähltem Kanal durch Probemessung (ED)
- Informationen über Qualität der Luftschnittstelle (LQI)
- Signalisierung mittels CSMA/CA, ob der Kanal frei ist (CCA)
- Bestimmung des Frequenzkanals
- Datenübertragung und Empfang

Die MAC-Schnittstelle (Media Access Control) hat folgende Aufgaben:

- Aushandeln des Zugangs zum physischen Medium
- Generierung von Beacons, falls Gerät Coordinator ist
- Beaconsynchronisation
- Unterstützung für Gerätesicherheit
- Verbund mit bzw. Trennung von anderen Netzwerken
- Verwaltung einer Großzahl von Geräten
- Verwaltung der unterschiedlichen Netztopologien
- Verwaltet CSMA/CA-Mechanismus für Kanalzugriff
- Behandlung und Aufrechterhaltung der GTS (Guaranteed Time Slots)
- Sicherstellung von einer zuverlässigen Verbindung zwischen zwei MAC-Entities

Die Datenrate beschränkt sich je nach ausgewähltem Frequenzband auf 250 kBit/s, 40 kBit/s oder 20 kBit/s. Genau wie WLAN und BlueTooth verwendet 802.15.4 zur Datenübertragung das weltweit verfügbare und lizensfreie 2,4 GHz ISM-Band (16 Kanäle). Bei Bedarf stehen auch die 868/915 MHz Bänder (10 Kanäle/1 Kanal), wobei aber Einbußen bei der Übertragungsleistung in Kauf genommen werden müssen. Als Modulationsverfahren kommen vereinfachtes

DSSS (Direct Sequence Spread Spectrum) für das 2,4 GHz ISM-Band und eine differentielle Kodierung für die 868/915 MHz-Bänder zum Einsatz. [ICS03]

PHY (MHz)	Frequency band (MHz)	Spreading parameters		Data parameters		
		Chip rate (kchip/s)	Modulation	Bit rate (kb/s)	Symbol rate (ksymbol/s)	Symbols
868/915	868–868.6	300	BPSK	20	20	Binary
	902–928	600	BPSK	40	40	Binary
2450	2400–2483.5	2000	O-QPSK	250	62.5	16-ary Orthogonal

Tabelle 2-2: Frequenzbänder und Datenraten [ICS03]

Für die Adressierung der Geräte können entweder kurze 16 Bit lange Adressen oder der IEEE-Standardadressbereich mit 64 Bit langen Adressen verwendet werden; die Adressierung selber ist eindeutig und ermöglicht so bis zu 65000 verschiedene Devices in einem Netzwerk. Die realisierbaren Netztopologien Peer-to-Peer, Stern sowie Cluster-Trees werden mit Hilfe von FFD's (Full Function Devices) und RFD (Reduced Function Devices) aufgebaut, wobei nur ein FFD das geformte Netzwerk als PAN-Coordinator verwaltet.

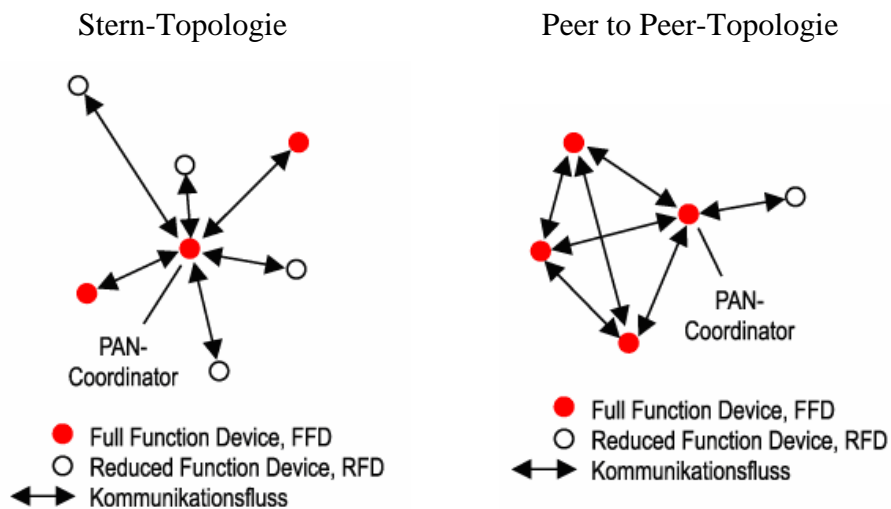


Abbildung 2-4: Stern- und P2P-Topologie [ICS03]

Auf dieser Grundlage lassen sich schließlich Clustertree-Netzwerke oder auch leistungsfähige und flexible, vermaschte Netze aufbauen, in denen Daten über einen Router (Stern) oder untereinander (P2P) ausgetauscht werden können.

Der Kanalzugriff bei einem Übertragungswunsch beruht auf den CSMA/CA-Verfahren (Carrier Sense Multiple Access with Collision Avoidance). Bei diesem Verfahren zur Verhinderung gleichzeitiger Sendeveruche hört die Sendestation zuerst den gewünschten Kanal ab. Wenn niemand den Kanal besetzt, schickt sie dem Empfänger eine RTS-Message (Request To Send). Wenn der Empfänger diese Nachricht bekommt, schickt er eine CTS-Message (Clear To Send). Nachdem der Sender dieses Signal empfangen hat, beginnt er, die Daten zu schicken.

CSMA/CA-Algorithmus:

1. Carrier Sense (Kanal abhören):

Zuerst muss das Medium überwacht werden.

2. Leitung ist frei:

Wenn die Leitung frei ist, beginne mit der Übertragung, andernfalls weiter mit Schritt 5.

3. Informationsübertragung:

Wenn eine Kollision entdeckt wird, beende die Datenübertragung und setze ein definiertes Störsignal (Jam) auf die Leitung (um sicherzustellen, dass alle anderen Transceiver die Kollision erkennen), dann weiter mit Schritt 5.

4. Übertragung erfolgreich abgeschlossen:

Erfolgsmeldung an höhere Netzwerkschichten, Übertragungsmodus verlassen.

5. Leitung ist belegt:

Warten, bis die Leitung wieder frei ist.

6. Leitung ist gerade frei geworden:

Noch eine zufällige Zeit (Backoff) abwarten, dann wieder bei Schritt 1 beginnen, wenn die maximale Anzahl von Übertragungsversuchen nicht überschritten wurde.

7. Maximale Anzahl von Übertragungsversuchen überschritten:

Fehler an höhere Netzwerkschichten melden, Übertragungsmodus verlassen. [ATM04]

Um die Einfachheit ohne Verlust der Robustheit zu gewährleisten, entsprechen die verwendeten Rahmen weitestgehend den aus dem IEEE 802.11-Standard entnommenen Forderungen und definieren vier verschiedene Rahmenarten:

- Beacon Frame für die Übertragung von Beacons
- Data Frame für die Datenübermittlung
- Optionaler Acknowledge Frame für die Bestätigungsquittierung bei erfolgreicher Übertragung
- MAC Command Frame für die Netzwerkverwaltung

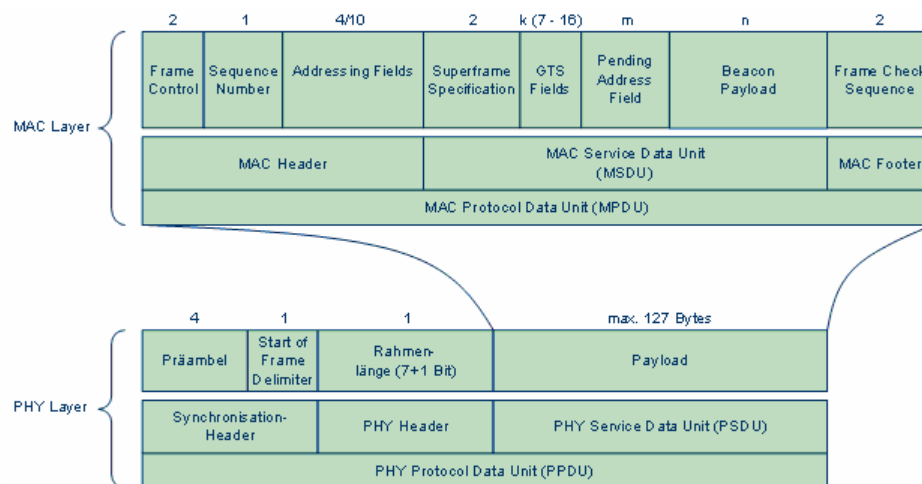


Abbildung 2-5: Beacon Frame bei IEEE 802.15.4 [ICS03]

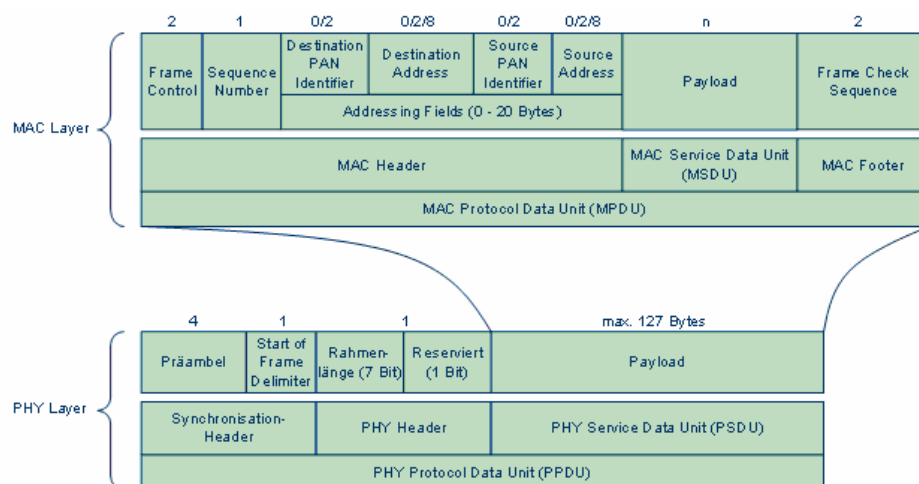


Abbildung 2-6: Data Frame bei IEEE 802.15.4 [ICS03]

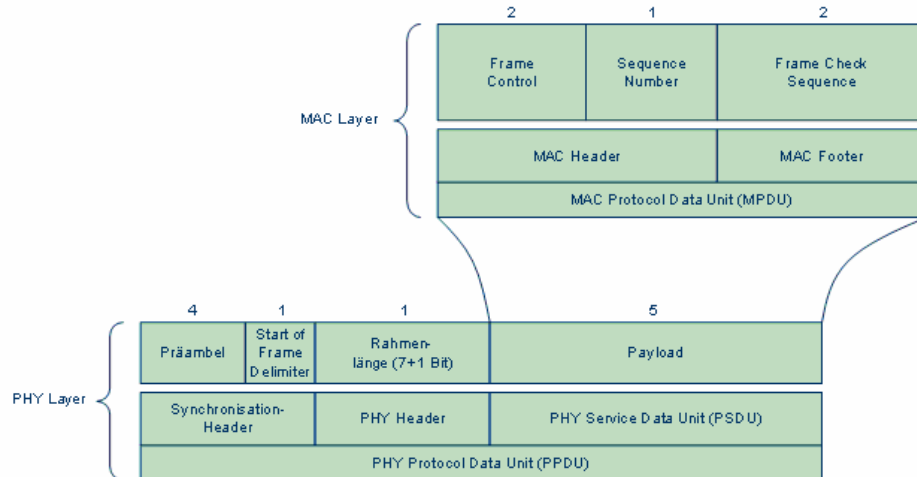


Abbildung 2-7: Acknowledgement Frame bei IEEE 802.15.4 [ICS03]

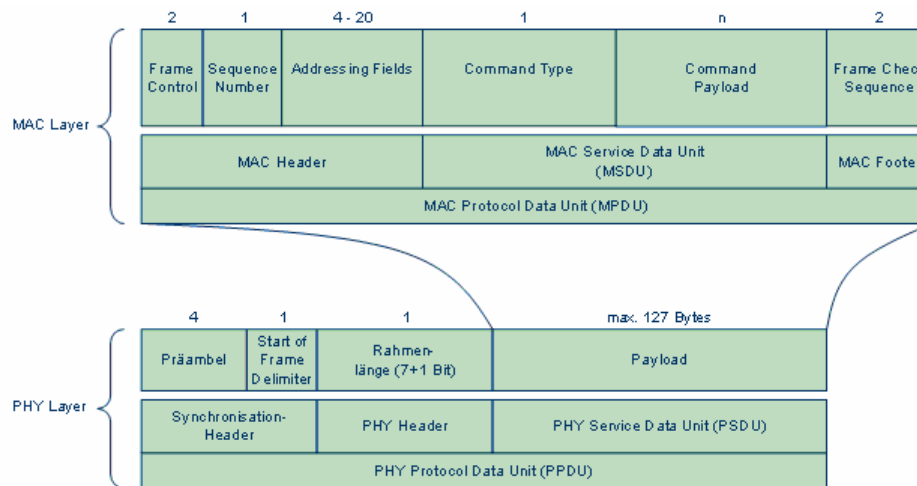


Abbildung 2-8: MAC Command Frame bei IEEE 802.15.4 [ICS03]

Neben dieser Art der Kommunikationssteuerung mittels Frames können zusätzlich auch garantierte Zeitslots (GTS – guaranteed time slots) oder die Superframestruktur mit Beacons genutzt werden. Diese voll handshakegesteuerte Übertragung gewährleistet eine gute Übertragungssicherheit, auch weil Datenpakete zwischengespeichert und erneut übertragen werden können.

Auf die Datenübertragungs- und Netzwerksicherheit wurde besonders viel Wert gelegt. So kommen neben umfangreichen Verschlüsselungsmethoden auch ACL's (Access Control Lists), einfache Firewallmechanismen, Geräteauthentifizierung, Datenpaketintegrität oder Schlüsselmanagementverfahren zum Einsatz. [ICS03]

2.3.3 BeeStack (ZigBee)

Der BeeStack ist Freescale's vollständig kompatibler ZigBee-Stack. Er erfüllt alle Anforderungen entsprechend der ZigBee-Spezifikation, wie z.B. Routingaufgaben, Netzwerkbildung, Netzwerkmanagement oder auch Gerätefindung. [FSZ06]

ZigBee baut auf den IEEE 802.15.4-Standard auf und definiert dabei Anwendungsprofile, die von den verschiedenen Geräteherstellern verwendet werden können. IEEE 802.15.4 wurde von der IEEE (Institute of Electrical and Electronics Engineer) für WPAN's mit geringen Datenraten definiert und beschreibt PHY- und MAC-Layer. Aber lediglich eine

Definition von PHY und MAC garantieren noch nicht, dass verschiedene Geräte untereinander kommunizieren können. Hier kommt ZigBee ins Spiel. So ermöglicht ZigBee dank der Application Profiles, dass Geräte von unterschiedlichen Herstellern miteinander Daten austauschen können. So ist es z.B. möglich, mit dem Lightning Profile eine Netzwerkumgebung zu schaffen, in der ein Lichtschalter der Firma A problemlos mit den Lampen der Firma B per Datenaustausch zusammen arbeiten kann.

Erste Entwicklungsschritte hin zu ZigBee begannen Ende 1998, als sich im Zusammenhang mit der Spezifikation des HomeRF-Standards eine Arbeitsgruppe um Philips Electronics bildete. Verschiedene Vorschläge für eine günstige, störunempfindliche Funklösung bildeten die Grundlage für PURL (Protocol for Universal Radio Links), in dem Vorüberlegungen für die Bitübertragungs- und Sicherungsschicht zusammengefasst wurden. Die Ende 2002 gegründete ZigBee-Alliance – eine Kooperation vieler Industrieunternehmen (darunter Intel, Motorola oder auch HP) – verfolgte das Ziel die oberen Schichten bis hin zum Anwendungsteil zu definieren. Die parallel gebildete Arbeitsgruppe um den Übertragungsstandard IEEE 802.15.4 setzte dagegen die Vorschläge aus PURL um und spezifizierte die unteren beiden Protokollschichten, auf die ZigBee später einfach aufsetzt. Die erste Spezifikation in der Version 1.0 wurde im Dezember 2004 der Öffentlichkeit vorgestellt.



Die Einsatzbereiche von ZigBee sind weit gefächert. Durch die zuverlässige, kostengünstige, verlustarme und sichere Funktechnologie ist ZigBee eine hervorragende Alternative für die kostenintensiven WLAN's und die protokolloverhead- und verlustleistungsgeplagten Bluetooth-Lösungen. Mit Übertragungsentfernungen von 10-75 m lassen sich problemlos ganze Areale komplett vernetzen, in denen alle Geräte untereinander kommunizieren können und dabei von nur einer Master-Unit gesteuert werden.

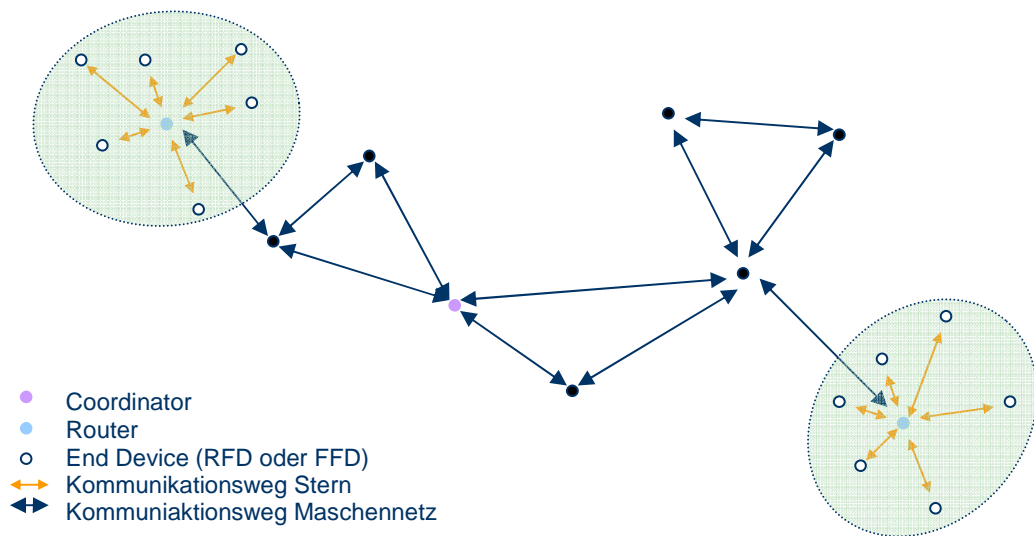


Abbildung 2-9: Maschen-Topologie [ZNE06]

Mit Hilfe des konsistenten Standards, einer intelligenten Kommunikation unter den beteiligten Geräten, ultimativer Flexibilität, hoher Mobilität und einer einfachen Nutzung in den verschiedenen Bereichen des täglichen Lebens, sind die Einsatzbereiche sehr vielfältig:

- Industrie und Automatisierungstechnik (Anlagensteuerung per Funk)
- Spedition und Logistik (Güterüberwachung)
- Heim- und Gebäudeautomatisierung (kabelfreie Steuerung von Geräten und Anlagen)
- Medizintechnik (drahtlose Patientendaten-Übertragung)
- Bedienung von Computer-Peripherie und Unterhaltungselektronik
- Alarm-, Sicherheits- und Überwachungsanlagen

ZigBee bietet neben geringen Anschaffungskosten eine leichte Implementierung, einen sicheren und zuverlässigen Datentransport, Bedienmöglichkeiten über kurze Distanzen (Remote), einen sehr geringen Stromverbrauch und ausreichende Sicherheitsaspekte. [Sur06]

802.15.4 sieht drei Sicherheitsstufen vor, wobei die ersten beiden Stufen jedoch nur optional sind.

- Stufe 1: keine Sicherheitsvorkehrungen
- Stufe 2: Einsatz von einfachen Firewall-Funktionen, z.B. Access Control Lists (ACL)
- Stufe 3: symmetrische Verschlüsselung unter Nutzung des AES-Algorithmus mit einer Schlüssellänge von 128 bit

Der verwendete Protokollstack definiert lediglich die höher liegenden Anwendungsschichten. Neben der Anbindung an normale ZigBee-Netzwerke haben Anwender die Möglichkeit, über den integrierten TCP/IP-Protokollstapel eine Internetverbindung aufrecht zu erhalten.

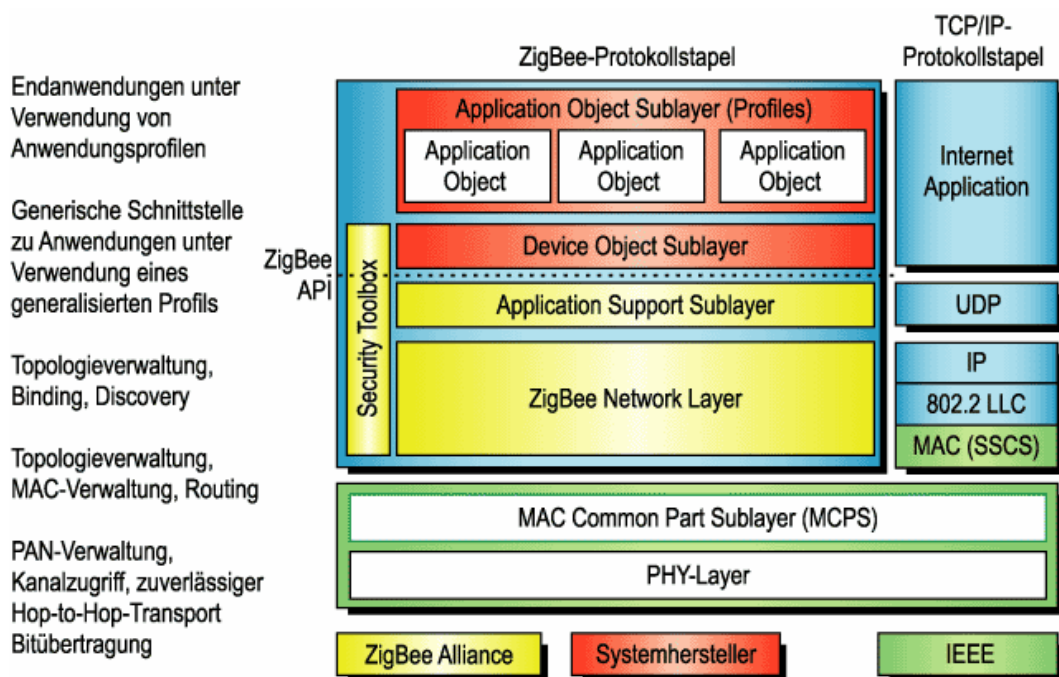


Abbildung: ZigBee-Schichtenmodell [LOH06]

PHY und MAC sind komplett aus dem IEEE 802.15.4-Standard übernommen und erfüllen die entsprechenden Aufgaben zum Verbindungsmanagement (siehe 2.3.2).

Der ZigBee Network Layer hat folgende Aufgaben:

- Realisiert Protokoll für eine verbindungslose Ende-zu-Ende-Datenübertragung in Netzwerken mit 802.15.4-Knoten
- Verwaltung der ZigBee-bezogenen Netzwerk-Topologie
- Routing empfangener Pakete
- Sicherheitsmanagement
- Mechanismen, um ZigBee-Geräten den Beitritt und das Verlassen eines Netzwerkes zu ermöglichen
- Konfiguriert und administriert die logischen Verbindungen zwischen ZigBee-Geräten (Pairing Tables)
- Ermöglicht Vergrößerung des Netzwerkes ohne stromfressende Sender und Empfänger
- Verwaltung der Netzwerk-Infrastruktur

Der Application Layer setzt sich drei Teilen zusammen:

- Application Support Sublayer (APS)
- dem ZigBee Device Object (ZDO)
- Application Framework (AF)

Der APS verwaltet die Paarung von Geräten, deren Kommunikation untereinander und die Identifikation sowie Zuordnung anderer Geräte im Wirkungsbereich des Netzes entsprechend ihrer Aufgabe.

Die Zuordnung der Geräteklasse, also der Rolle die das Gerät im Netzwerk einnimmt (FFD, RFD, Coordinator), legt das ZigBee Device Object (ZDO) fest. Zusätzlich ist das ZDO für das Auffinden neuer Geräte, der Ermittlung der von

ihnen bereitgestellten Funktionalitäten und dem Aushandeln der Sicherheitsaspekte zuständig.

Im Application Framework (AF) schließlich sind die benutzerdefinierten Anwendungen integriert. [Loh06]

3 Vorüberlegungen zur Modellierung

Funkszenarios zu modellieren und im Anschluss eine Aussage zur Performance zu treffen, bedarf einer gründlichen Einarbeitung in die entsprechende Technologie. Die verschiedenen Funktionalitäten des Funksystems auf ein Funktionsmodell herunter zu brechen ist enorm komplex – schon allein wegen der facettenreichen Anwendungsgebiete. Im Gegensatz zu drahtgebundenen Netzen kommen hier zusätzliche Faktoren zum Tragen, die ebenfalls Beachtung finden müssen.

- Mobilität der Endgeräte
- verfügbare Datenrate in Abhängigkeit zur Entfernung
- Geländeprofil
- Sendeleistung
- Zelllast
- Topologie
- ...

Des Weiteren muss im Vorfeld genau klar sein, welche Strategie zur Modellierung verfolgt wird. Hierbei gibt es zwei wesentliche Unterscheidungen – die statische und die dynamische Analyse.

Eine statische Herangehensweise beinhaltet keine Vorgeschichte und analysiert nur die im aktuellen Zeitschritt gewonnene Momentaufnahme. Das vereinfacht die Modellierung und erspart zeitaufwendige Implementierungen und speicherintensive Algorithmen.

Die dynamische Analyse beinhaltet die Vorgeschichte des Systems. In unserem Fall der Funktechnologiebewertung wären Faktoren wie z.B. Ruhephasen/Sendephase der Geräte, ein Bewegungsprofil der Mobile, Signalisierungsaufwand, Schwankungen der Sendeleistung etc. wichtige Indikatoren, die in die Berechnung einfließen müssten.

Da wir für unser Modell aber schon alle Geräte, die dahinter stehende Topologie, alle Kommunikationsverbindungen und Datenwege, sowie die Übertragungsarten

genau kennen, können wir auf das statische Prinzip zurückgreifen und alle bekannten Größen ohne vorgeschichtliche Abhängigkeiten einfließen lassen.

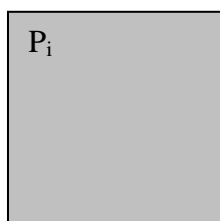
Im Hinblick auf den späteren Vergleich des Modells mit echten Funkgeräten bleiben die Eigenbewegung, sowie Ein- und Ausschalten oder Hinzufügen der Geräte zur Laufzeit unberücksichtigt. [Loh06]

3.1 Modellierungsbausteine

Das Modell soll vom Grundaufbau her einem vorgeschriebenen Schema entsprechen. Die implementierten Netzwerkgeräte stellen Partitionen dar, welche im Wesentlichen für die Datengenerierung zuständig sind. Innerhalb dieser Partitionen sind Funktionsbausteine enthalten, die wie ein Schichtensystem (Protokollstack) miteinander verbunden sind und generierte Datenstrukturen der höheren Schichten (Anwendungsdaten) an ein standardisiertes Interface weiterleiten. Das Interface bildet zugleich die Basis des Protokollstacks. Die dorthin durch die Schichten weitergeleiteten Datenstrukturen werden im Interface umformatiert und an den Channel übertragen, der die erhaltenen Daten zeitlich verzögert an alle mit dem Channel verbundenen Partitionen übergibt.

Dieses Schema soll den Austausch zwischen beliebigen Protokollimplementierungen ermöglichen, vorausgesetzt sie sind alle nach den vorgeschriebenen Richtlinien aufgebaut (siehe später 4.3.2).

- Partitionsbaustein



Enthält alle für die Funktionsausführung notwendigen Teilblöcke (Funktionsbausteine) und wird von ihrer Funktionsweise her mit den Funkgeräten gleichgesetzt.

→ Funktionsblöcke

→ Interface

→ Kommunikationsverbindung der Funktionsbausteine

- Funktionsbaustein

F_i

Enthält die gewünschte Funktionalität entsprechend der Anwendung, die in unserem Fall aber recht einfach gestrickt ist und nur zur Generierung von Datenstrukturen (Frames, siehe später 4.7) und der dazugehörigen Informationsweiterleitung an weitere Funktionsbausteine genutzt werden.

- Interfacebaustein

I_i

Wird aus einem Rahmensystem entnommen und enthält die nötigen Algorithmen, um eine Datenübertragung zum Channelbaustein zu ermöglichen. Die zu übertragenden Daten werden in eine einheitliche Form gebracht.

- Channelbaustein

C_i

Enthält alle für die Berechnung des Übertragungsdelays wichtigen Informationen und Berechnungsvorschriften entsprechend der implementierten Übertragungsart.

→ Berechnungsvorschriften zur Delaybestimmung

→ Datenratenfunktion

→ Logging, Veranschaulichung mittels Übertragungsmatrix

3.2 Umsetzung der Topologien

Die auf Basis von IEEE 802.15.4 entwickelten Technologien unterscheiden, je nachdem in welcher Topologie die Geräte integriert sind, den Kommunikationsfluss.

So können im Sternverbund Datenübertragungen nur über einen zentralen Router erfolgen, während bei Peer-to-Peer alle integrierten Geräte bei Bedarf einfach an ihren Wunschkommunikationspartner anfangen zu senden.

Als Beispiel soll ein normales Maschennetz dienen, welches sowohl Stern- als auch Peer-to-Peer-Anteile beinhaltet.

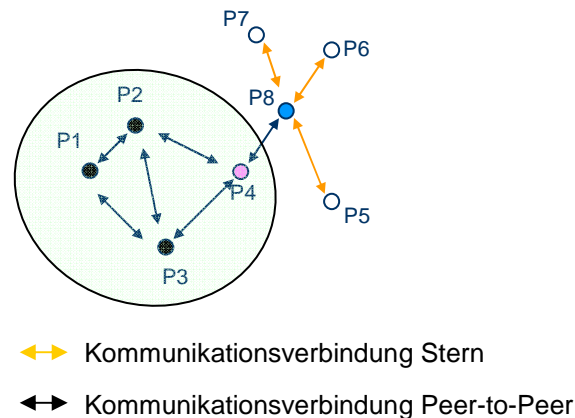


Abbildung 3-1: Beispielnetz für Modellierung

Da die Geräte auch Zwischenstationen darstellen, sind die Unternetze jeweils eigenständig. So haben die Geräte im Peer-to-Peer-Verbund die maximale Datenrate von 250 kBit/s für alle ihre Datenübertragungen zur Verfügung (geteilte Ressource), während die Geräte im Sternverbund entsprechend der Kommunikation über den integrierten Router die maximale Datenrate von 250 kBit/s jeweils für eine Datenübertragung zur Verfügung haben (Server-Ressource).

Ein Coordinator oder Router kann uneingeschränkt im Netzwerk interagieren und wird deshalb als normales FFD betrachtet. Ihre Sonderrolle ist nur für die Netzwerkverwaltung bzw. Routingmöglichkeit von Bedeutung. Wenn sie Daten zu übertragen haben, wird dies im Modell explizit integriert, genau wie alle anderen Datenübertragungswege.

Die Interface- und Channelbausteine sind dabei fest vorgegeben und sollen nicht verändert werden. Damit wird eine hohe Generität erreicht, um die modellierten Geräte problemlos austauschen zu können. Andere in MLDesigner implementierten Technologien, denen dieses Schema zu Grunde liegt, können

ohne zusätzliche Veränderungen verbunden werden. Die bereits darauf aufbauenden Wishbone-, Ethernet-, USB 2.0- oder TCP/IP-Protokolle können so ebenfalls in die 802.15.4-Umgebung eingebunden werden, um Multiprotokoll-Modelle zu simulieren. Schließlich ist es öfters von Vorteil, verschiedene Übertragungsarten in eine Simulation zu integrieren, um Vergleichsmöglichkeiten bezüglich der Performance zu schaffen.

Beispiel: Ein Roboter soll entwickelt werden. Die Datenübertragung innerhalb des Roboters funktioniert meist über bekannte Interfaces wie IDE, Seriell, Platinenbus oder ähnliches, während die Steuerung von außen oft per Funk realisiert wird. Möchte man die Fülle von Übertragungswegen simulieren und deren Leistungsfähigkeit entsprechend der gewünschten Anwendung testen, muss man nicht jede Datenübertragung einzeln erfassen, sondern kann sie in einem kompletten System abbilden und auswerten.

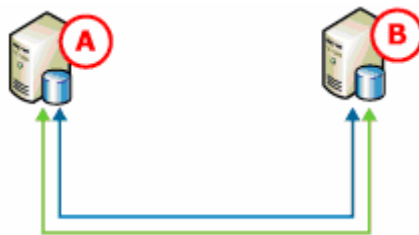


Abbildung 3-2: Peer-to-Peer

Im ersten Entwicklungsschritt soll lediglich die Peer-to-Peer-Verbindung zwischen zwei Geräten simuliert werden.

3.3 Parameter der Modellierungsbausteine

Für die spätere Implementierung in MLDesigner ist es von Vorteil sich vorher genau mit den nötigen Parametern der Modellierungsbausteine auseinander zu setzen. Dazu soll versucht werden so viel Funktionalität wie möglich in die Bausteine zu integrieren und sie gleichzeitig so generisch wie möglich zu halten. Schließlich soll das endgültige Modell leicht austauschbar sein bzw. in anderen Implementierungen Einsatz finden können.

3.3.1 Parameter des Channels

Für eine gute Übersicht wurden die einzelnen Parameter in zwei Klassen eingeteilt – reine Datenübertragungsparameter (Channel) und reine Performanceparameter (Delay).

- Datenübertragungsparameter

In diesem Baustein werden die Ausgabeformate realisiert. Zudem sollen alle ankommenden Datenpakete für die spätere Delayberechnung so gepackt/entpackt werden, dass der Delaybaustein die für seine Aufgabe nötigen Daten leicht abrufen kann. Des Weiteren könnte eine Ausgabe in Form einer Matrix integriert sein, die eine Übersicht über wichtige Channelwerte liefert.

→ Übertragungsmatrix

- mittlere Auslastung
- Vollaustung
- Leerauslastung
- übertragene Datenmenge

→ Channelpaketdatenstruktur

- Performanceparameter

Da hier die Belastung des Trägermediums von Bedeutung ist, sind Parameter wie die momentane Netzlast, die Auslastung des Channels und die für die aktuelle Datenübertragung mögliche Datenrate von Bedeutung. Diese werden mittels mathematischer Ansätze berechnet und stützen sich auf die Entfernungsparameter der Geräte untereinander und die maximal mögliche Datenrate im Unternetz. Da aber nur zwei Geräte miteinander verbunden werden und das CSMA/CA-Verfahren zum Einsatz kommt, sind die aktuelle Netzlast und die Auslastung des Channels äquivalent. Es bleibt lediglich die Delayberechnung übrig.

- Datenrate für Kommunikation zum Zeitpunkt i
- Verzögerung des Paketes (Delay im Channel)

3.3.2 Parameter des Funktionsbausteines

Für die Funktionalität von Coordinator und Enddevice werden zwei unterschiedliche Stacks benötigt, die dem ISO/OSI-Modell nachempfunden werden. Jede einzelne Schicht kann dabei ebenenweise nach oben oder nach unten kommunizieren. Für eine solche Kommunikation kommen Datenstrukturen zum Einsatz, welche die entsprechenden Informationen zur Weiterverarbeitung beinhalten.

- Datenstrukturen der einzelnen Schichten
- Algorithmen entsprechend der Schichtendienste

3.3.3 Parameter des Interfaces

Das Interface selber stellt die unterste Schicht des implementierten Stacks dar und realisiert den Zugriff auf das physikalische Medium. Hier werden alle Pakete in eine dem Channel verständliche Form gebracht.

- Channelpaketdatenstruktur

3.4 Bestimmung der Entfernung

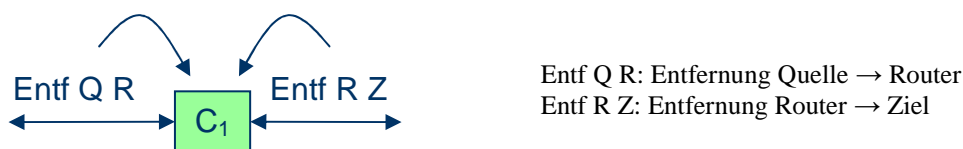
Da das zu modellierende System statischer Natur ist, gibt es zwei Möglichkeiten die Entfernungsproblematik umzusetzen. Entweder wird die Entfernung einfach als Wert bei der Modellierung eingetragen oder jedes Gerät erhält seine eigenen Koordinaten (z.B. GPS, 3D-Koordinatensystem, ...), mit denen dann eine

Entfernungsberechnung erfolgen kann. Diese Werte werden dann in die Berechnung zur Datenrate einbezogen.

3.4.1 Entfernung als direkter Wert

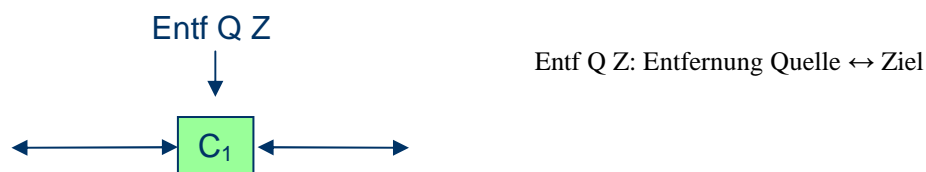
Bei einer einfachen Vorgabe des Entfernungswertes von beiden Kommunikationspartnern an den Channelbaustein entfällt eine zusätzliche Berechnung. Der Wert ist fix, gilt für die gesamte Analysephase und wird vor der Initialisierung einfach eingegeben.

Entfernungsproblematik bei Stern:



Da die Geräte im Sternverbund ihre Daten über den integrierten Router schicken müssen, interessieren die in dem Router eingegebenen Entfernungen aller an ihm angeschlossenen Endgeräte. Sind die untereinander kommunizierenden Geräte bekannt, werden die Entfernungen herausgesucht und die zwei separaten Übertragungswege bestimmt. Diese Daten werden an den Channelbaustein weitergegeben und könnten die Grundlage für die Berechnung der Datenrate stellen.

Entfernungsproblematik bei Peer-to-Peer:



Bei Peer-to-Peer ist die direkte Übertragung zwischen den beteiligten Geräten möglich. Von daher interessiert nur die Entfernung zwischen ihnen, die auch direkt dem Channelbaustein vor der Initialisierung bekannt gegeben wird. Daraus könnte dann die Datenrate bestimmt werden.

3.4.2 Entfernung mit Hilfe der Gerätekoordinaten

Die Implementierung gegenüber der direkten Entfernungseingabe hat einen erhöhten Rechenaufwand zur Folge. Dem Channelbaustein werden die Koordinaten der jeweils an der Kommunikation teilnehmenden Geräte bekannt gegeben. Daraus könnte die Entfernung mathematisch ermittelt werden und dieser Wert fließt dann in die Berechnung zur Datenrate ein.

Entfernungsproblematik bei Stern:



Wie bei den festen Entfernungen müssen mit Hilfe der Koordinaten ebenfalls die Entfernungen von der Quelle zum Router und weiter zum Ziel bestimmt werden. Die Entfernungen der Endgeräte sind im Scheduler gespeichert, der ja den Router simuliert. Mit Hilfe der untenstehenden Abstandsberechnung werden die jeweiligen Entfernungen bestimmt und fließen in den Channelbaustein ein, der wiederum daraus die maximal mögliche Datenrate bestimmt.

Entfernungsproblematik bei Peer-to-Peer:



Für die Entfernungsbestimmung bei Peer-to-Peer-Netzen reicht die Eingabe der Koordinaten der beteiligten Geräte. Über die Abstandsformel kann so problemlos die Entfernung zwischen den Kommunikationspartnern bestimmt werden. Diese wird ebenfalls für die Berechnung der Datenrate im Channelbaustein verwendet. Der Abstand zwischen zwei Punkten a und b berechnet sich mit Hilfe der mathematischen Formel

$$l = \sqrt{(xa - xb)^2 + (ya - yb)^2 + (za - zb)^2}$$

Die dafür verwendeten Punkte sind die Koordinaten der Geräte und durch Punkt P eindeutig festgelegt.

$$P = (x_p, y_p, z_p)$$

3.5 Berechnung der Datenrate

Für die Analyse zur Performance auf den Kommunikationswegen ist es unerlässlich, die Datenrate in Abhängigkeit zur Entfernung der beteiligten Geräte zu wissen. Wie in jedem anderen Netzwerk müssen genau die zur Übertragung von Daten verwendeten Frames analysiert werden. Unter der Voraussetzung die Übertragungsrate im Zeitschritt i, sowie die Anzahl der zu übertragenden Größen der Frames ist bekannt, sind Rückschlüsse auf die verwendete Zeit möglich. Über einen langen Zeitraum gesammelte Messwerte könnten dann in einer Performancetabelle verglichen werden.

Ein weiterer wichtiger Punkt ist die verwendete Zeit für die Bearbeitung der Daten innerhalb der Schichten des Stacks. Denn auch CPU-Leistung und Datenaufkommen spielen hier eine wichtige Rolle. Schließlich werden erst auf Anwendungsebene die empfangen Daten wirklich verwertbar, das Durchlaufen der unteren Schichten ist nur Mittel zum Zweck.

3.5.1 Datenratenberechnung mittels Pfadverlust

Eine Möglichkeit, Rückschlüsse auf die für das Gerät verfügbare Datenrate zu ziehen, ist die Methode der Datenratenberechnung unter Ausnutzung des Pfadverlustes.

Dabei wird vorausgesetzt dass jeder Sendestation eine begrenzte Sendeleistung zur Verfügung steht. Bei UMTS z.B. maximal 1000 mW im 1,8 GHz-Frequenzband. Der Pfadverlust selber gibt an, wie stark die Sendeleistung zu einem entfernten Teilnehmer erhöht werden müsste, damit die gewünschte Datenrate des entsprechenden Teilnehmers weiter zur Verfügung stehen kann. Bei UMTS z.B. liegt der Wert etwa bei r^3 . Würde sich also der Teilnehmer von seiner aktuellen Position bzgl. der Antennenstation weiter weg entfernen, muss die Sendeleistung der Basisstation drastisch erhöht werden, um ihm wieder die volle Bandbreite bieten zu können. Da die Sendeleistung zwangsläufig das technische Maximum nicht überschreiten darf, ist eine Senkung der Datenrate zu dem sich entfernenden Teilnehmer die Folge. [BPL06]

Das kann ausgenutzt werden, um eine Rückrechnung auf die praktisch mögliche Datenrate in abhängig zur Entfernung des Gerätes durchzuführen. Im Allgemeinen ist der Pfadverlust auf den Entfernungen, die als Standard festgesetzt sind, sehr gering. Erst im Randbereich der Abdeckung steigt der Pfadverlust exponentiell an, bis die Verbindung abbricht, weil der Teilnehmer den Netzbereich verlässt oder die anderen Teilnehmer aufgrund der hohen Netzbelastung des Einzelnen nicht mehr bedient werden könnten.

Entfernung	Sendeleistung
0 m	1 mW
10 m	1 mW
20 m	1 mW
30 m	2 mW
40 m	5 mW
50 m	5 mW
60 m	5 mW
70 m	10 mW
80 m	25 mW
90 m	50 mW
100 m	100 mW

Tabelle 3-1: Beispielhafte Messergebnisse

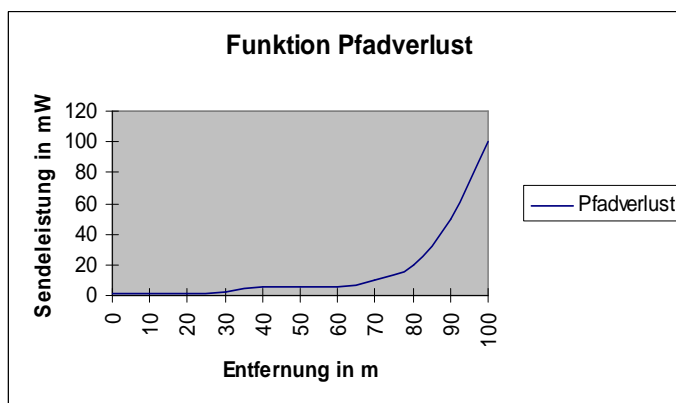


Abbildung 3-3: Funktion des Pfadverlustes

Aus der Funktion des Pfadverlustes können nun Prozentwerte durch eine Normierung erreicht werden. D.h. die prozentuale Steigerung der Sendeleistung wird als prozentualer Wert der maximalen Datenrate von dem Ausgangswert 250 kBit/s abgezogen.

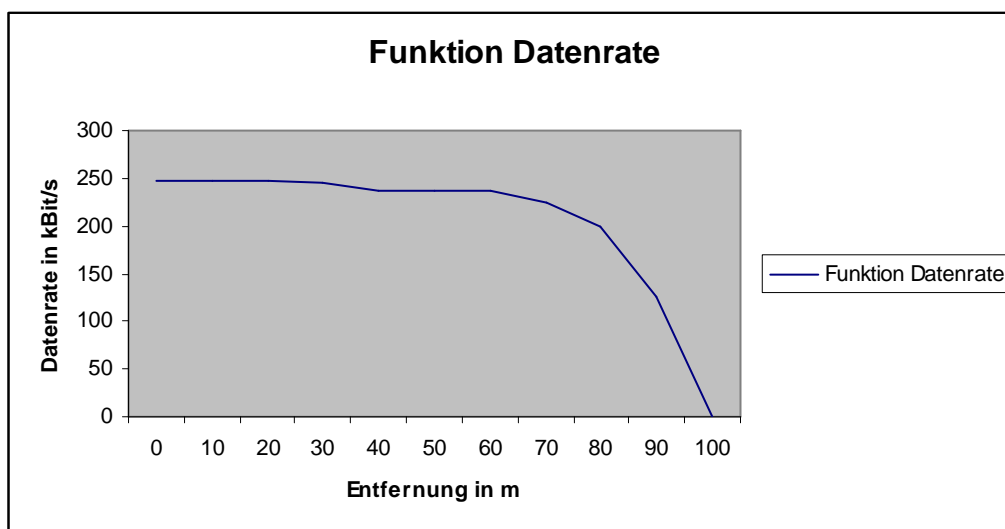


Abbildung 3-4: Funktion der Datenrate mittels Pfadverlust

Aus dieser Funktion die verfügbare Datenrate abzulesen ist leicht. In Abhängigkeit zur Entfernung sind unterschiedliche Datenraten möglich. Vor allem an den Randbereichen des Netzes kann die maximale Datenrate von 250 kBit/s nicht gehalten werden. Für solch große Entfernungen ist ZigBee aber auch nicht konzipiert worden.

3.5.2 Datenratenberechnung durch Messung

Eine weitere Möglichkeit verlässliche Daten bzgl. der möglichen Datenrate zu erhalten, sind Messungen, die mit Standardmodulen durchgeführt werden. Im Wesentlichen gibt es zwei Einsatzgebiete.

Der In-House-Bereich erlaubt typischerweise nur sehr geringe Reichweiten und Datenraten, weil dort vor allem durch die Wände viel Energie absorbiert wird. Zusätzlich erhöhen sich negative Auswirkungen von Nebenerscheinungen wie Reflexionen, Scattering (Streuung) oder Diffractions (Beugung) bzgl. des ausgestrahlten Signals in Räumen oder Häusern erheblich.

Der Freiluft-Bereich hingegen ermöglicht weitaus höhere Reichweiten und Datenraten, obwohl auch hier Probleme wie Fading (Überblendung/Störung von anderen Funknetzfrequenzen) oder Shadowing (Gerät hat keine direkt Sichtverbindung zum Übertragungspartner) möglich sind. [WI04]

Die verschiedenen Messergebnisse werden gesammelt und ausgewertet. Je nach Einsatzgebiet können nun sichere untere Schranken der Datenrate angegeben werden, die auf jeden Fall auch haltbar sind und gesicherte Datenübertragungen gewährleisten. Um eine einheitliche Funktion zu erhalten, werden die Messwerte gemittelt und in eine Funktionsstruktur eingebracht, die ein einfaches Ablesen der möglichen Datenrate in Abhängigkeit zur Entfernung erlaubt.

Entfernung	max. mögl. Datenrate
0 - 10 m	220 kBit/s
10 - 25 m	200 kBit/s
25 - 50 m	125 kBit/s
50 - 100 m	50 kBit/s
100 - 150 m	10 kBit/s
> 150 m	5 kBit/s

Tabelle 3-2: Beispielhafte Messwerte

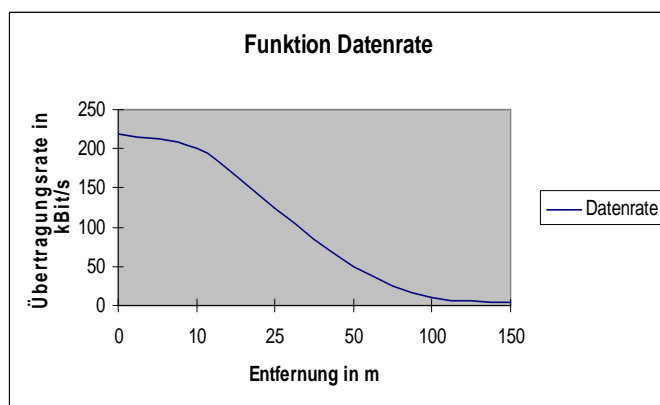


Abbildung 3-5: Funktion des Datenrate mittels Messung

Wegen der Mittelwertbestimmung der verschiedenen Messungen sind die Datenraten im Allgemeinen schlechter als bei einer Bestimmung mit Hilfe des

Pfadverlustes. Dafür sind die gewonnenen Werte aber auch sicherer und spiegeln den tatsächlichen Bandbreitenhintergrund des Netzes wider. Mit den niedrigst gemessenen Werten ist eine Vorhersage auf das Netzverhalten besser gewährleistet als mit Datenraten, die eventuell nicht gehalten werden können. Auf jeden Fall ist hier Vorsicht geboten, da zwischen dem modellierten Netz und dem zu Grunde gelegten „echten“ Netz Widersprüchlichkeiten auftreten können. Eine sehr genaue Messung, am besten mit den später verwendeten Modulen, in Bezug auf das spätere ZigBee-Netz ist enorm wichtig.

3.5.3 Datenratenberechnung durch Paketanzahl

Eine letzte, sehr einfache Möglichkeit Rückschlüsse auf die Datenübertragungsrate zu ziehen, bildet das Verfahren des ‚Paketzählens‘. Grundlage hierbei ist ein 100% einheitliches Datenpaket, welches in bestimmten Abständen verschickt wird. Dieses immer gleichgroße Paket braucht annähernd die gleiche Zeit zu seinem Empfänger. Das macht man sich zu nutze, indem man über ein recht langes Zeitfenster alle angekommenen Pakete zählt und so die Datenrate mathematisch berechnen kann.

$$\text{Datenrate} = \frac{\text{Paketgröße} \cdot \text{Paketanzahl}}{\text{Übertragungszeit}}$$

Eine gewisse Inkubationszeit, also bis das Netzwerk seine volle Übertragungsleistung erreicht hat, sollte dabei berücksichtigt werden.

3.6 Berechnung des Delays bei einer Übertragung

Zusammen mit der Datenrate, der Übertragungsart und der Menge der zu verschickenden Daten kann problemlos die Zeit vom Versenden bis zum vollständigen Empfangen bestimmt werden. Dazu muss genau klar sein, welche

Frames die Daten beherbergen und wie oft sie verschickt werden müssen, bis die komplette Datenmenge vollständig übertragen wurde.

3.6.1 Bestimmung der Frameanzahl

Alle generierten Daten der Anwendungsschicht werden in einer Queue zwischengespeichert und über die Interfaces für die Abarbeitung hin zum Channelbaustein weitergeleitet, der die Pakete dann überträgt. Die darin enthaltenen Übertragungsanforderungen müssen auf die Datenframes heruntergerechnet werden. Der maximale Payload eines solchen Datenframes liegt bei 102 bis maximal 105 Byte je nach Übertragungsart, die Framegröße selber bei maximal 133 Byte. Wird zusätzlich ein Acknowledge erwünscht kommen für jedes verschickte Datenpaket noch einmal 11 Byte hinzu.

- Beispielhafte Berechnung einer Datenübertragung von 5 KByte

$\#Full_Data_Frames = (Datenmenge \cdot 1024) \text{ DIV } Payload$

$\#Data_Frames = \#Full_Data_Frames + 1$

$Size_All_Full_Data_Frames = \#Full_Data_Frames \cdot Data_Frame_Size$

$Data_Frame_Size = 133$

$Payload_Last_Data_Frame = (Datenmenge \cdot 1024) \text{ MOD } Payload$

$Size_Last_Data_Frame = Overhead_Data_Frame + Payload_Last_Data_Frame$

$Full_Transfer = Size_All_Full_Data_Frames + Size_Last_Data_Frame$

$Opt_Acknowledge = \#Data_Frames \cdot Acknowledgement_Frame_Size$

$Acknowledgement_Frame_Size = 11$

$\#Full_Data_Frames = [(5 \cdot 1024) \text{ DIV } 102] + 1$

$= 50$

$\#Data_Frames = 50 + 1$

$= 51$

$$\begin{aligned}\text{Size_All_Full_Data_Frames} &= 50 \cdot 133 \\ &= 6650 \text{ Byte}\end{aligned}$$

$$\begin{aligned}\text{Payload_Last_Data_Frame} &= (5 \cdot 1024) \text{ MOD } 102 \\ &= 20\end{aligned}$$

$$\begin{aligned}\text{Size_Last_Data_Frame} &= 31 + 20 \\ &= 51\end{aligned}$$

$$\begin{aligned}\text{Full_Transfer_Size} &= 6650 + 51 \\ &= 6701 \text{ Byte}\end{aligned}$$

$$\begin{aligned}\text{Opt_Acknowledge} &= 51 \cdot 11 \\ &= 561 \text{ Byte}\end{aligned}$$

Im Endeffekt müssen 6701 Byte übertragen werden, um ein vollständiges Versenden der 5 KByte Ausgangsdaten zu gewährleisten. Ein zusätzliches Acknowledge würde nach jedem Data Frame noch einen Acknowledgement Frame erfordern. Bei 51 Data Frames, die für die vollständige Übertragung notwendig wären, sind das zusätzlich 561 Byte, die es zu übertragen gilt. Bei einer Beacon-enabled-Übertragung würden noch jeweils zwei 133 Byte lange Beacon Frames hinzukommen und so die eigentliche Übertragungsmenge um weitere 266 Byte vergrößern.

3.6.2 Bestimmung der Übertragungszeit

Die maximal verfügbare Datenrate für den Übertragungswunsch lässt sich aus der zu Grunde gelegten Datenratenfunktion ablesen. Nach einer Umrechnung der Bezugsgrößen kann man problemlos den Zeitbedarf für die Übertragung ermitteln.

- Beispielhafte Berechnung einer Datenübertragung von 5 KByte

$$\text{Transfer_Rate} = (\text{Data_Rate} \cdot 1024) / 8 \quad // \text{ Umrechnen von x kBit/s in y Byte/s}$$

$$\text{Transfer_Time} = (\text{Full_Transfer_Size} / \text{Transfer_Rate}) \cdot 1000$$

$$\text{Transfer_Rate} = (120 \cdot 1024) / 8$$

$$= 15360 \text{ Byte/s}$$

$$\text{Transfer_Time} = 6701 / 15360 \cdot 1000$$

$$= 436,26 \text{ ms}$$

Eine einfache Datenübertragung von 5 KByte ohne Acknowledge im non-beaconing-Modus würde also ca. 436 ms dauern.

4 Die Modellierung

Alle in Kapitel 3 vorgeschlagenen Herangehensweisen sind Modellierungsvorschläge und werden bei der späteren Erläuterung des eigentlichen Modells manifestiert, denn nicht alle Ideen sind sinnvoll umsetzbar bzw. realisiert worden. Sie bilden lediglich einen Grundstein und sollen Probleme im Vorfeld erkennbar machen.

Als Grundlage für die modellhafte Realisierung der Funkübertragungsmöglichkeiten SMAC, IEEE 802.15.4 MAC, und ZigBee dient das Entwurfstool MLDesigner.

4.1 *Mission Level Design*

Während der letzten Jahre stiegen die Anzahl der Probleme in Softwaresystemen rapide an (Microsoft Betriebssysteme, SW in Handys,...), komplexe HW/SW-Systeme wiesen unakzeptable Fehlerraten auf (Automobilelektronik in Luxuslimousinen, 702 Boeing Satelliten, LKW-Maut-System) oder Entwicklungen mussten schon in der Entwurfsphase aufgegeben werden (Bill Gates/Craig McCaws Teledesic Satellitensystem). Diese und ähnliche Probleme fügten der Wirtschaft einen Schaden von über 100 Milliarden Euro zu. Der Grund dafür liegt darin, dass verwendete Entwicklungsmethoden dem rapiden Anstieg der Komplexität von einem Faktor 100 alle 10 Jahre nicht mehr gewachsen sind und notgedrungen zu diesen Problemen führten. Neue ganzheitliche Entwicklungsmethoden sollen diese Problematiken vermeiden. Die Anzahl der Entwurfsiterationen beim Systementwurf mittels Mission Level Design-Methodik konnte dadurch reduziert, die Entwicklungszeiten komplexer Hard- und Softwaresysteme wesentlich um den Faktor 2 bis 10 verkürzt und Risiken bei der Entwicklung minimiert werden. [GI05]

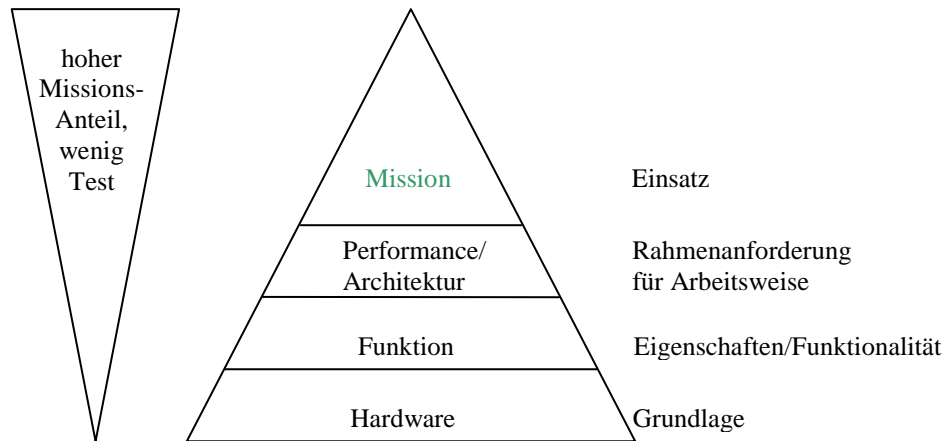


Abbildung 4-1: Entwurfspyramide [SE05]

Mission Level Design verfolgt das Ziel, frühzeitig im Entwurfsprozess Aussagen bezüglich der Leistungsfähigkeit des Systems zu gewinnen und mögliche Probleme zu erkennen. Dabei handelt es sich um eine missionsbezogene, modellgestützte Systementwurfsmethode für komplexe Systeme. Sie überführt die Systemspezifikation in ein ausführbares Gesamtsystemmodell auf Systemebene und simuliert dieses. Gegenüber anderen Entwurfsmethoden zeichnet sich das Mission Level Design durch die Einbeziehung der so genannten Missionen aus. Bei ihnen handelt es sich um typische Einsatzszenarien des zu entwerfenden Systems. Durch die missionsbezogene Simulation wird sichergestellt, dass das System die angestrebten Eigenschaften erreicht. [SE05], [SST03]

4.2 MLDesigner

Das von MLDesign Technologies, Inc. entwickelte MLDesigner bietet als erstes Softwaresystem die Möglichkeit eine Modellierung auf Missions-Ebene durchzuführen und auszuwerten. Diese Abstraktionserhöhung stellt die nächste Generation von „System-Design-Software“ dar und wird nach und nach die älteren Softwaresysteme, welche ein niedrigeres Abstraktionsniveau verwenden, ablösen.

MLDesigner ist eine Integration und Weiterentwicklung der Multi-Domain-Technologie des Ptolemy Projektes [PP07], BONeS (Block Oriented Network

Simulator) und Ctrl-C/Model-C, dem Vorläufer von Matlab/Simulink. Das System-Design-Tool MLDesigner trägt aktuell die Version 2.7. Mit Hilfe einer graphischen Oberfläche (GUI) können komplexe Problemstellungen veranschaulicht, simuliert und ausgewertet werden. Die gewonnenen Ergebnisse geben Rückschlüsse auf Verhalten und Funktionsweise des Modells bezüglich der realen Welt.

Der Aufbau eines Modells orientiert sich dabei an wesentlichen Bestandteile der UML [UML07] und basiert auf einer Hierarchie von Blockdiagrammen. Dafür stehen unterschiedliche Typen von Blöcken zur Verfügung.

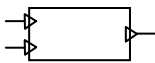
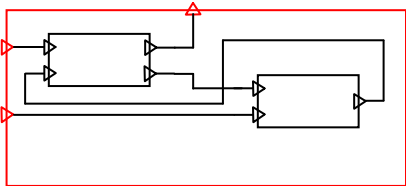
<p>Primitive</p> 	<ul style="list-style-type: none"> - Grundelement zur Modellierung - Block, zu dem es keine Verfeinerungsebene gibt - beinhaltet die konkrete Implementierung <ul style="list-style-type: none"> → Code, externe Funktionalität, Parameter, Memories, ...
<p>Module</p>	<ul style="list-style-type: none"> - Block aus Primitiven u./o. Modulen mit Verbindung (Ports) nach außen  <p style="color: red;">→ stellt Block in höherer Ebene da</p>
<p>System</p>	<ul style="list-style-type: none"> - Block aus Primitiven u./o. Modulen <u>ohne</u> Ports nach ganz außen - oberste Ebene des Modells → Ausführungsebene
<p style="text-align: center;">↓ Block</p>	<ul style="list-style-type: none"> - hierarchische Ebene - kann Blöcke, Primitives und Module enthalten
<p>Wormhole</p>	<ul style="list-style-type: none"> - Block, der in einen Block einer anderen Domäne eingefügt wurde <ul style="list-style-type: none"> → Schnittstelle zwischen verschiedenen Domänen

Tabelle 4-1: Arten der Modellierungsblöcke bei MLDesigner [RE05]

Die Kommunikation unterhalb der Blöcke erfolgt über Input- und Output-Schnittstellen. Für die jeweiligen Ein- und Ausgabekonventionen werden in den Blöcken Funktionsanweisungen gespeichert, die Eingangsdaten zu Ausgangsdaten überführen.

Port	<ul style="list-style-type: none"> - Anschlußpunkt / Schnittstelle - ermöglicht Datenaustausch zwischen Blöcken - verschiedene Datentypen möglich (Int, String, Boolean, DataStruct, ...)
Memory	<ul style="list-style-type: none"> - interner oder externer Speicher - verschiedene Datentypen möglich (Int, String, Boolean, DataStruct, ...) - kann beschrieben, gelöscht und ausgelesen werden
Event	<ul style="list-style-type: none"> - in Blöcken implementierte Anfangsinitialisierung - wird bei vorher im Modell festgelegten Zeitpunkt gestartet

Tabelle 4-2: Datenhandling bei MLDesigner [RE05]

Als mögliche Modelldomänen kommen drei wesentliche Arten zum Einsatz, die die Laufzeit der Simulation und das Scheduling beeinflussen. Innerhalb des Modells sind verschiedene Formen frei kombinierbar.

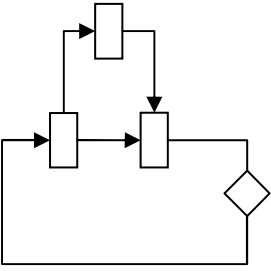
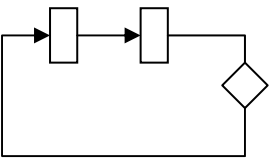
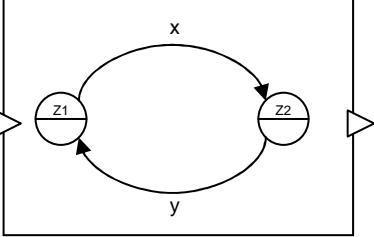
<p style="text-align: center;">DE (Discrete Event)</p>	<ul style="list-style-type: none"> - alle Ereignisse sind voneinander vollständig getrennt - Verbindungen sind Nachrichtenleitungen - verschiedenen Zeitpunkte für Event - Datensemantiken sind atomare Ereignisse <ul style="list-style-type: none"> → jeder Block kann als Zustandsmaschine angesehen werden - nichtmonotones Scheduling <ul style="list-style-type: none"> → Block feuert, wenn an mind. einem Eingang ein Ereignis anliegt → dynamisch, zur Laufzeit <div style="text-align: center; margin-top: 20px;">  </div>
<p style="text-align: center;">FSM (Finite State Maschine)</p>	<ul style="list-style-type: none"> - Semantik der Verbindungen sind ratenmonoton abgetastete Werte - Scheduling ist ratenmonotones Feuern <ul style="list-style-type: none"> → statisch: jeder genau einmal → in vorgegebener Reihenfolge die Blöcke abarbeiten <div style="text-align: center; margin-top: 20px;">  </div>
<p style="text-align: center;">SFM (Synchronous Data Flow)</p>	<ul style="list-style-type: none"> - abgeleitet aus <i>DE</i> - einsetzen in <i>DE</i>-Domain - Darstellung durch Automatengraphen <div style="text-align: center; margin-top: 20px;">  </div>

Tabelle 4-3: Arten der Modelldomänen bei MLDesigner [RE05]

In die grafische Benutzeroberfläche des MLDesigner sind zur schnelleren Modellierung ein Compiler, verschiedenen Systemschnittstellen und einem Frame-Set unterschiedlicher Editoren, einschließlich eines Multi-Dokument-Editors zum Erstellen, Editieren und Speichern von Modellen integriert. Somit sind alle Bearbeitungsmöglichkeiten in einer Oberfläche zusammengefasst (siehe Abbildung 4-2). Zur besseren Überschaubarkeit kann jeder Block mit Hilfe von Viewelementen viele verschiedene graphische Ausprägungen besitzen.

Eine umfangreiche Bibliothek von Basisbausteinen mit unterschiedlichsten Funktionen für die verschiedenen Domänen werden von Vornherein als Grundbausteine bereitgestellt und zu besserer Übersichtlichkeit in entsprechende Untergruppen eingeteilt – analog zu einer Explorer-Baumstruktur.

Die Möglichkeit einzelne Modellelemente (Primitives, Module etc.) oder ganze Modelle als Instanzen in anderen Modellen wieder verwenden zu können, erreicht eine Verteilung der Komplexität über mehrere Ebenen und die Realisierung extrem komplexer Aufgabenstellungen.

Die Datenhaltung der Modelle basiert auf XML [XML07]. Der Quellcode von Primitiven wird im Format der „Ptolemy-Language“ gespeichert. Diese PTCL Kommando Umgebung ist ein ausgewähltes Subset von C++ und dient zum Erstellen von großen Systemen, deren Komplexität eine grafische Repräsentation überschreiten würde.

Ein erweiterbarer Multi-Domänen-Simulator ermöglicht die Ausführung bzw. Simulation der einzelnen Systeme und unterstützt dabei mehrere verschiedene Domänen. Iterative Parametrisierung von Systemen ist genauso möglich, wie eine verteilte Simulation über mehrere vernetzte Rechner.

Zur leichteren Fehlerfindung umfasst MLDesigner einen grafischen Debug- und Test-Mechanismus, der Einzelschritt- oder auch Mehrschrittsimulation zulässt. [MLDM07], [Bau07], [RE05]

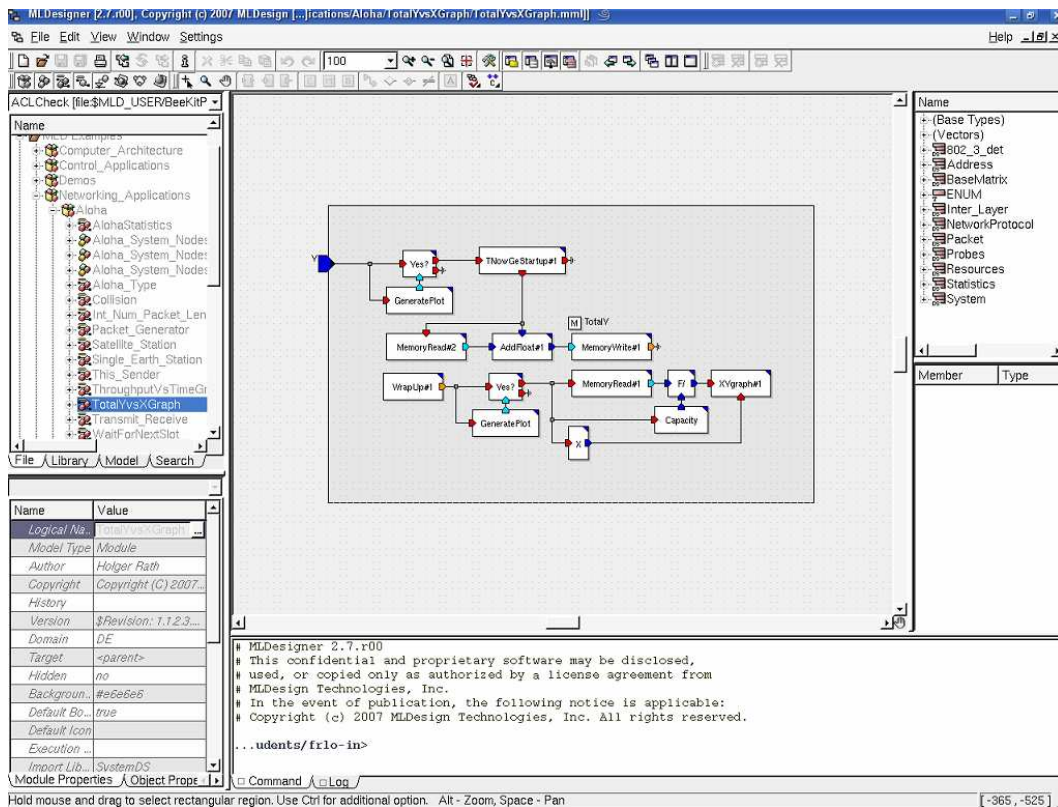


Abbildung 4-2: Benutzeroberfläche des MLDesigner

4.3 Allgemeines zum Modellaufbau

Die drei Übertragungsprotokolle SMAC, IEEE 802.15.4 und ZigBee sind jedes für sich in ein separates Modell eingeordnet und ausimplementiert. Der allgemeine Aufbau bleibt gleich und orientiert sich an den jeweiligen Spezifikationen.

Wie in den Vorüberlegungen schon erwähnt, bilden die Partitionen Endgerät bzw. Coordinator bei IEEE 802.15.4/ZigBee oder Sender bzw. Receiver bei SMAC. Der Channel selbst bleibt frei austauschbar, genau wie die jeweiligen Gerätetypen.

Sicher ist es wenig sinnvoll ZigBee-Devices mit einem USB 2.0-Bus zu verlinken, aber mit kleinen Änderungen z.B. beim Kanalzugriff dennoch möglich. SMAC und IEEE 802.15.4/ZigBee sind wegen der unterschiedlichen Datenverwaltung nicht direkt kompatibel. Während bei SMAC und besetztem Channel die Pakete verloren gehen, werden bei den beiden anderen Standards die

Pakete gequeued und zu einem späteren Zeitpunkt eine neue Sendeversuch eingeleitet. Da ZigBee auf IEEE 802.15.4 auf setzt, sind Geräte mit ZigBee- und IEEE 802.15.4-Stack voll kompatibel.

Alle wichtigen Geräte- und Channelparameter können von außen vor der eigentlichen Simulation verändert werden, wie z.B. die IP- und MAC-Adressen, das Ziel der Übertragung, die Zeitabstände zwischen der Datengenerierung, die Geräteentfernungen und die Form der angezeigten Ausgabeinformationen. Zudem ist es möglich von vornherein anzugeben, wie lang die Simulation laufen soll – über ein bestimmtes Zeitintervall oder unendlich mit Abbruchoption durch den Benutzer.

4.3.1 Grundaufbau des Modells

In MLDesigner können alle modellierten Bausteine mehrfach in eine Simulationsebene integriert werden.

Von allen verarbeiteten Parametern werden dann entsprechende Kopien angelegt, so dass jede Modellkomponente autonom arbeiten kann. Dadurch können viele Devices in einem Netzwerk verbunden werden, ohne dass jedes Device extra modelliert werden muss. Wichtig ist nur, dass alle von außen veränderbaren Parameter vollständig und regelkonform eingetragen werden, damit es zu keinen Ungereimtheiten oder gar Abstürzen kommt. Kleine Fehlerausschriften helfen solche Ereignisse zu gezielt zu finden bzw. in Nachhinein zu beheben.

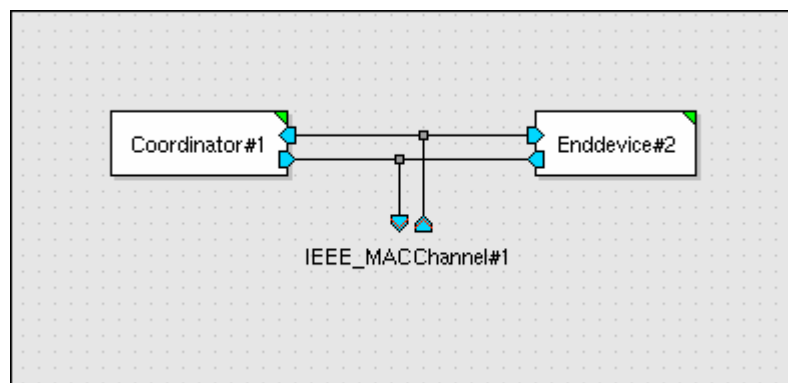


Abbildung 4-3: Beispiel Peer-to-Peer-Netz-Modell

Die einfachste Form zwei Geräte miteinander zu verbinden ist das Peer-to-Peer. Jedes Gerät hat dabei die Möglichkeit seinem Gegenüber Daten zu schicken und von diesem auch Daten zu empfangen.

Komplizierte Netze, wie z.B. Maschen- oder Sternnetz, sind von der reinen Topologie her ebenfalls mit MLDesigner abbildbar, deren Funktionsweise und Implementierung bleibt für diese Diplomarbeit aber nur am Rande von Bedeutung.

Zur Erfüllung dieses Gesamtkonzepts kommen vier Framework-Komponenten zum Einsatz. Eine Basisbibliothek für standardisierte Modellkomponenten, eine Architekturbibliothek, die alle Ausführungseinheiten, Kommunikationsprotokolle und -kanäle zusammenfasst, das ESL-Target, welches die Ausführung von Architekturmodellen kontrolliert und auswertet, sowie ein Architektur-Generator, der zur Iteration über Architekturmodelle verwendet wird.

4.3.2 ESL-Umgebung

Das implementierte Netzwerkmodell selbst ist in eine übergeordnete, durch den Nutzer nicht änderbare, Instanz eingebettet – das ESL-Target.

Ein Target koordiniert die Ausführung des Modells, wobei Ausführung selber nicht nur als Simulation zu sehen ist, sondern auch eine Codegenerierung sein kann. Zusätzlich werden das Scheduling und die Implementierung der durch die Domänen beschriebenen Algorithmen abgestimmt. Dadurch entsteht eine gewisse Kontrolle über die Modellausführung.

Das ESL-Target selber hat momentan die Aufgabe zur strukturellen Validierung des Architekturmodells und ist die Basis für die gemeinsame Performance-Analyse.

Über Klassenfunktionen können diverse Methoden dieser Instanz in den Code der Funktionsbausteine übernommen und aufgerufen werden. Dazu gehören z.B. Steuerfunktionen für den Kanal oder Abfragen auf die maximale Teilnehmerzahl im Netzwerk.

Die nötige Einstellung, um das ESL-Target zu aktivieren, befindet sich im Toplevel des Modells beim Parameter „Target“. Hier wird der voreingestellte Wert „Default-DE“ auf „ESL“ gewechselt, welcher trotzdem noch die Funktionalität der DE-Domain beinhaltet, diese aber um zusätzliche Aufgaben und Funktionen erweitert. Dazu gehören die Ausgabematrix zur Laufzeit bei Endlossimulation, der Status des Channels und eine Abfrage der Partitionen in Hinblick auf die maximale Anzahl von Geräten im Netzwerk.

Die Grundidee dahinter besteht in der späteren Möglichkeit, Übertragungsprotokolle, die ebenfalls nach den vorgegeben Richtlinien aufgebaut sind, ohne großartige Änderungen an deren Implementierung frei austauschbar zu gestalten. Um dies zu erreichen, stehen drei Kernelemente im Mittelpunkt. Die Partition/Executer, das Interface und der Channel. Diese sind in der ArchitectureBlockSet-Library im MLDesigner vorhanden.

Als zusätzliche Vereinheitlichung sind auch die zwischen den Partitions versendeten Daten in speziell aufgebauten Datenstrukturen enthalten, die später als Frames erklärt werden (siehe 4.7).

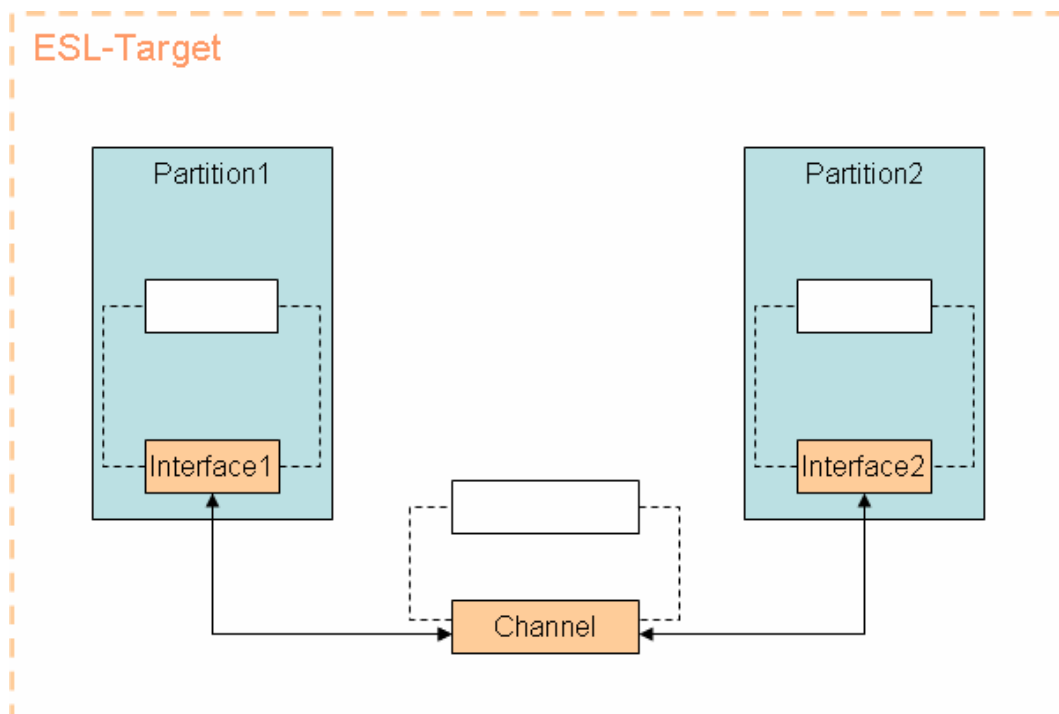


Abbildung 4-4: ESL-Target

Die Interfacebausteine, der Channel und das ESL-Target werden unverändert übernommen. Um dieses Grundmodell mit zusätzlichen Aufgaben zu versehen, werden weitere Funktionsblöcke eingebaut, welche dann die Aufgaben der einzelnen Übertragungsprotokolle erfüllen. Dies erreicht eine maximale Flexibilität in Hinsicht auf spätere Weiter- oder Neuentwicklungen. Zudem wird somit die Schnittstelle eines standardisierten Kommunikationsinterfaces gewahrt. So können problemlos andere Übertragungstechnologien wie Wishbone, RS232 oder TCP/IP implementiert werden und sind zumindest vom Rahmensystem her vollständig kompatibel untereinander. [BPS07]

- Partition

Die Partition ist für die vollständige Ausführung der implementierten Anwendungen zuständig. Die Abarbeitung benötigt dazu Zeit und Ressourcen, wie z.B. Wartezeit und Speicherplatz in einer Queue. Von der Begrifflichkeit her sind die Partitionen später den beschriebenen Geräteklassen der verschiedenen Übertragungsformen gleichzusetzen.

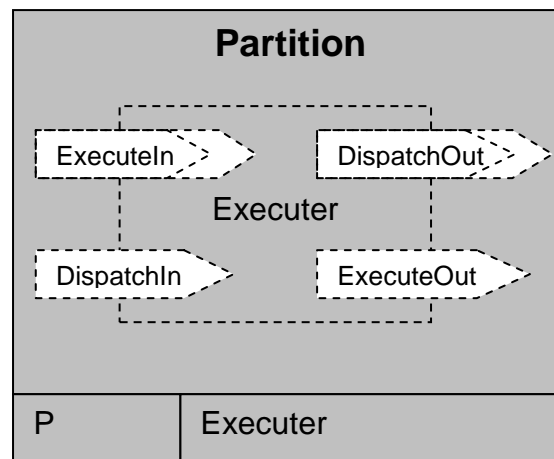


Abbildung 4-5: Grundaufbau Partition

Vom Grundprinzip her fasst der Executer alle Funktionsblöcke (z.B. den Protokollstack) in einem Block zusammen und führt diese entsprechend aus. Als zentraler Block kommt dafür eine CPU zum Einsatz, an der alle Funktionsblöcke

angekoppelt sind. Im Ergebnis soll eine zeitliche Verzögerung während der Simulation erreicht werden, um ebenfalls die Problematik der Abarbeitungszeit in dem Gerät selber zu betrachten. Der Executer selber ist aber nur rudimentär in die ESL-Umgebung implementiert und wird erst in späteren Entwicklungsstadien automatisch in die Partionen eingebaut. Er bleibt deshalb für diese Diplomarbeit unbeachtet.

- Interface

Als Verbindung zwischen Partition und Channel dient das Interface. Es transformiert Funktionsdatenflüsse in Architekturdatenflüsse und zurück. Wie in Abbildung 4-4 dargestellt, bildet das Interface die Basis für den Protokollstack und somit die letzte Instanz bevor die Daten auf den Channel geleitet werden.

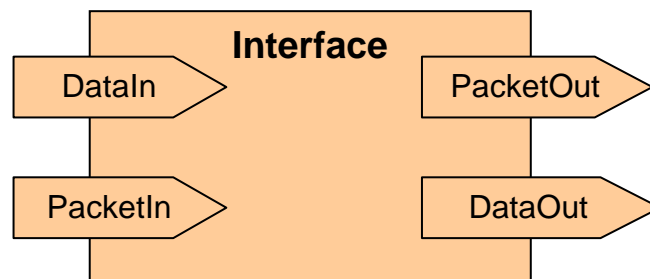


Abbildung 4-6: Grundaufbau Interface

- Channel

Für die Verbindung den vielen verschiedenen Partitions untereinander dient der Channel. Er ermöglicht die Kommunikation in Hin- als auch Rückrichtung.

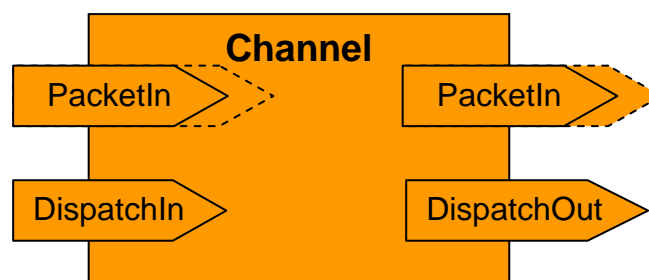


Abbildung 4-7: Grundaufbau Channel

4.4 Grundaufbau der Netzwerkgeräte

Vom modularen Grundaufbau her sind alle Geräte in Toplevel gleich und entsprechen den in der ESL-Umgebung beschriebenen Partitions. Lediglich die Benennung ist unterschiedlich und lehnt sich dabei an die verwendeten Übertragungsprotokolle an.

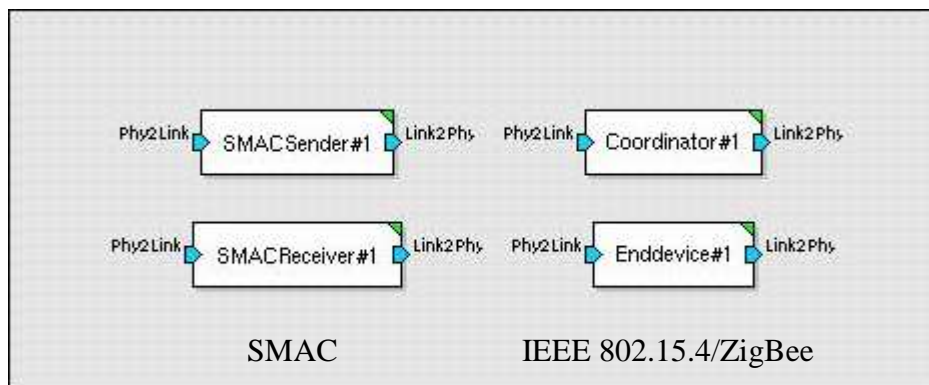


Abbildung 4-8: Übersicht der Netzgeräte

Bei SMAC heißen die Geräte Receiver (Datenempfänger) und Sender (Datenübermittler), während bei IEEE 802.15.4/ZigBee die Geräte mit Coordinator bzw. Enddevice benannt sind.

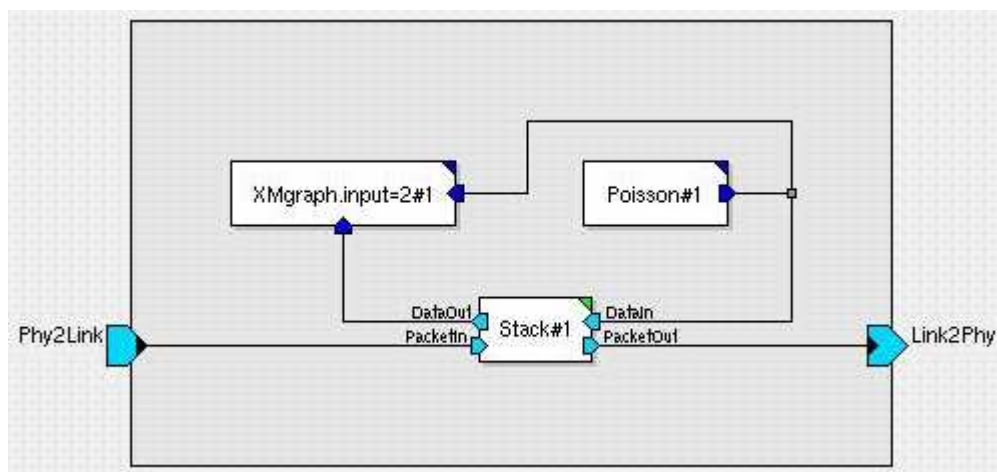


Abbildung 4-9: modularer Aufbau eines Gerätes

Jedes Modul enthält eine Primitive zur Datengenerierung (Poisson), den entsprechenden Protokollstack (Stack), eine Primitive zur graphischen Auswertung des Sende- und Empfangszeitpunktes der einzelnen Pakete (XMgraph), sowie einen Ein- und Ausgabebort (Phy2Link, Link2Phy).
 Zusätzlich können verschiedene Stacks untergebracht werden, um eine Simulation mehrerer unterschiedlicher Datenströme in einem Modell zu analysieren.

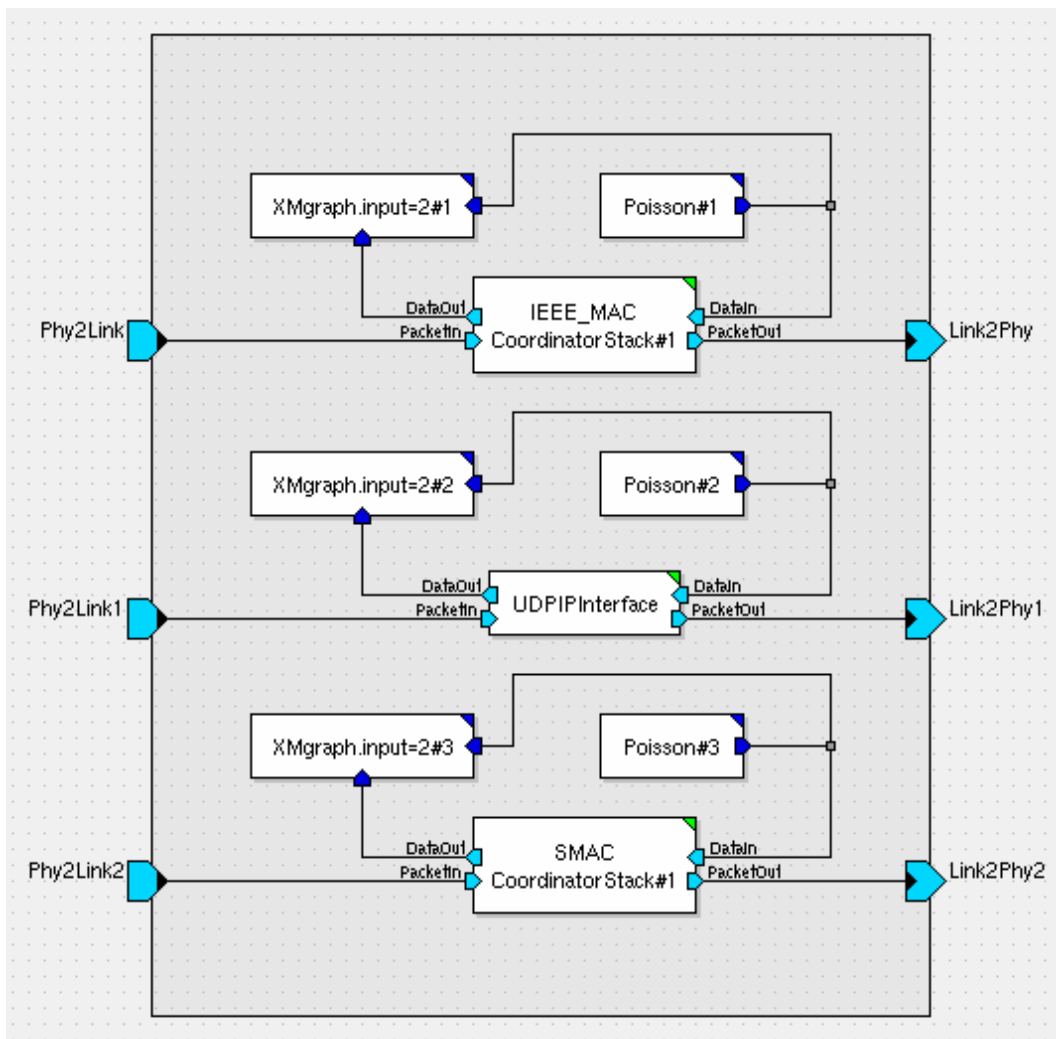


Abbildung 4-10: Netzgerät mit unterschiedlichen Übertragungsstacks

Die dadurch mehrfach erstellten In- und Outputs ermöglichen zudem die parallele Datenverbreitung von mehreren Geräten im Netzwerk.

Man denke dabei an das Roboter-Beispiel, in dem der Roboter die Datengenerierung übernimmt und seine verschiedenartigen Daten an den Empfänger übermittelt, die dort wiederum ausgewertet werden.

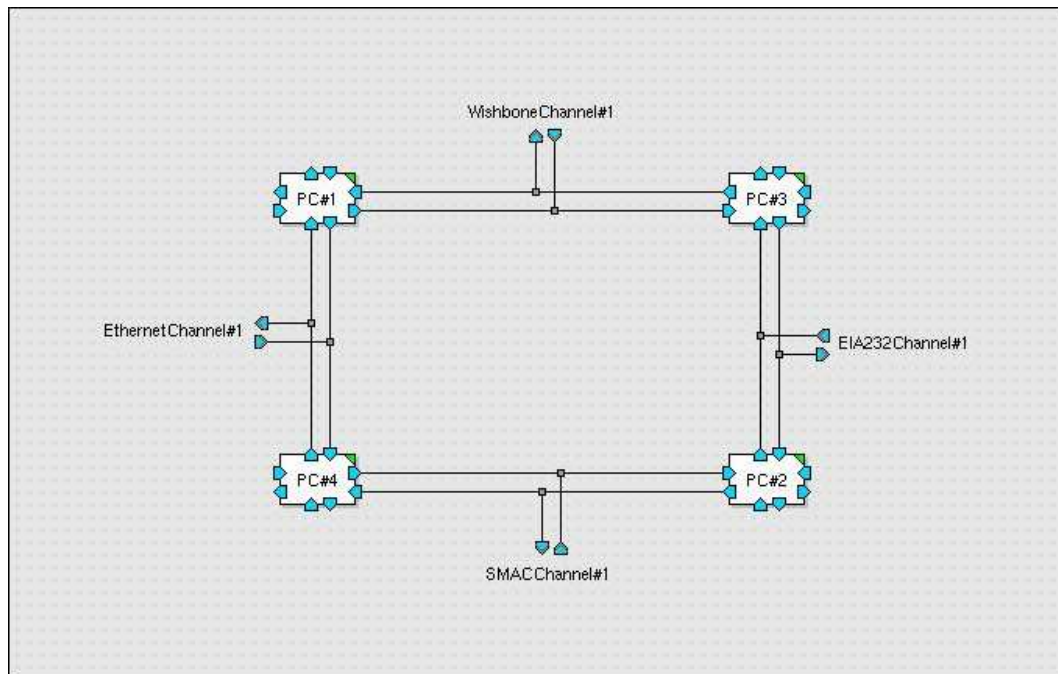


Abbildung 4-11: Netzaufbau mit verschiedenen Übertragungstechniken

4.5 Grundaufbau des Protokollstack

Sowohl Endgerät als auch Coordinator (bei IEEE 802.15.4/ZigBee) bzw. Sender und Receiver (bei SMAC) beinhalten ihre eigenen Protokollstacks entsprechend der Spezifikation.

Das zugrunde gelegte Schichtenmodell wurde dem ISO/OSI-Referenzmodell nachempfunden. Funktechnologietypisch sind dabei aber die höheren Schichten in einer einzigen Anwendungsschicht zusammengefasst.

Jeder Layerbaustein wurde generisch aufgebaut, damit ein Austausch bzw. die Geräteanpassung an die verschiedenen Übertragungstechnologien problemlos stattfinden kann. Zusätzlich wird so der Aufbau neuer Stacks vereinfacht, da nur die Bausteine ausgetauscht oder ummodelliert werden müssen, anstatt den ganzen Stack als Gesamtes neu aufzubauen. Die Schichten sind untereinander verbunden

und können so eingehende Daten an darunter oder darüber liegende Schichten weiterleiten. Die Aufgaben der einzelnen Schichten werden in 4.6 genauer erläutert.

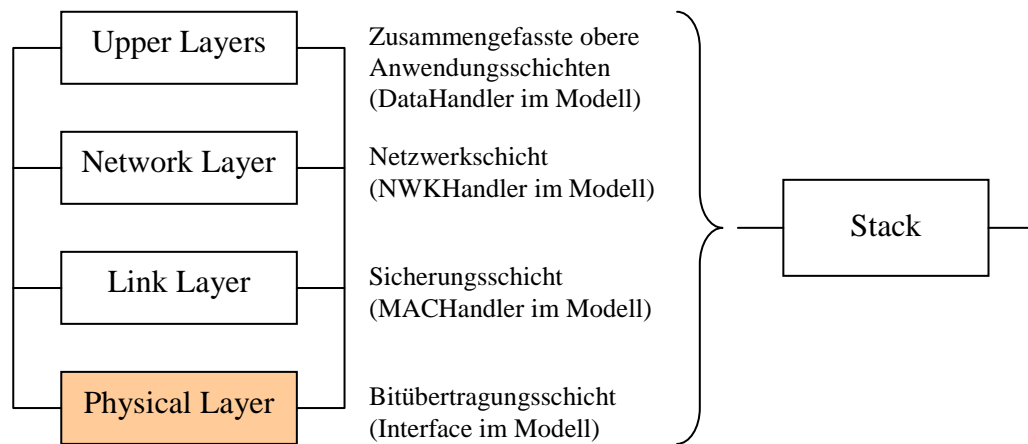


Abbildung 4-12: Grundaufbau des Protokollstacks

Jedes Stackmodul enthält bestimmte Memory's (interne Speicher) auf die die Schichten, welche die diversen gespeicherten Informationen benötigen, zugreifen können, z.B. Sendestatus-Flag, Sender-IP, Receiver-IP oder auch MAC-Adresse.

4.5.1 Protokollstack bei IEEE 802.15.4/ZigBee

- Enddevice

Wegen der geringeren Speicherkapazität des RAM's der Hardwaremodule, hervorgerufen durch Kosteneinsparung bei der Herstellung, sowie des kleineren Aufgabenumfangs haben Enddevices keine Netzwerkschicht und somit im Modell auch keinen NWKHandler. Dieser ist nur für Coordinatoren vorgesehen.

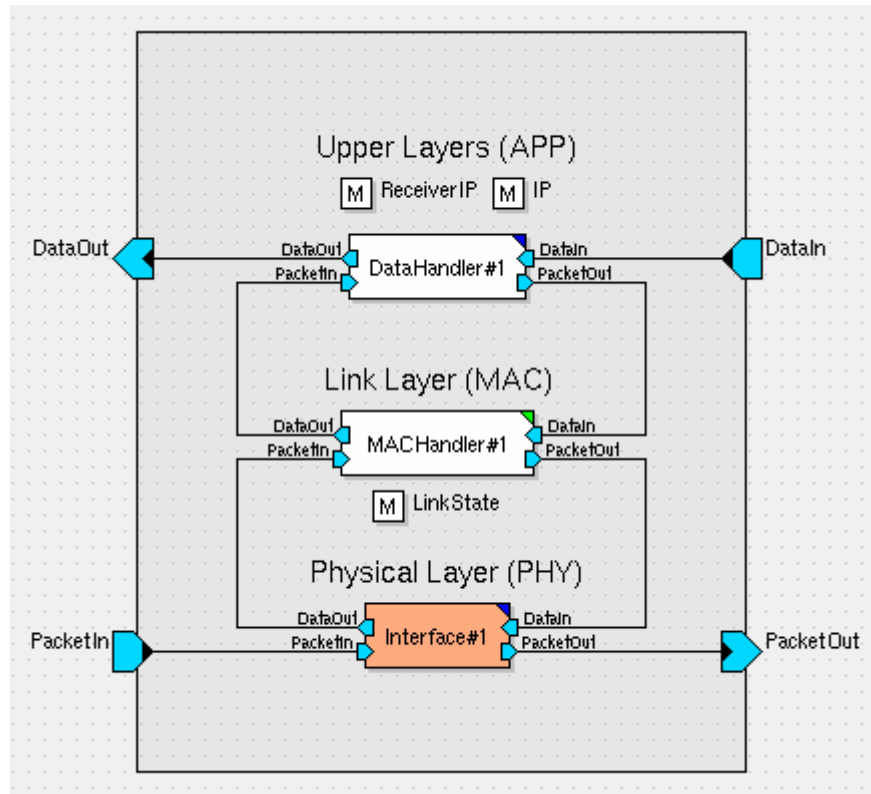


Abbildung 4-13: Protokollstack des Enddevice

Ein Enddevice hat deshalb nur drei Schichten.

- Upper Layers → DataHandler
- Link Layer → MACHandler
- Physical Layer → Interface

Im Stack sind drei interne Speicher enthalten, die zur Informationsverarbeitung während der Simulation durch die Schichten verarbeitet werden. ReceiverIP und IP bilden die Quell- und Zieladresse für eine Datenübertragung des entsprechenden Gerätes. Der LinkState gibt an, ob ein Gerät Daten empfangen und senden darf.

- Coordinator

Im Gegensatz zum Enddevice besitzt der Coordinator mehr Schichten, was sich auch mit dem in Realität verwendeten größeren Protokollstack zur Netzwerkverwaltung deckt.

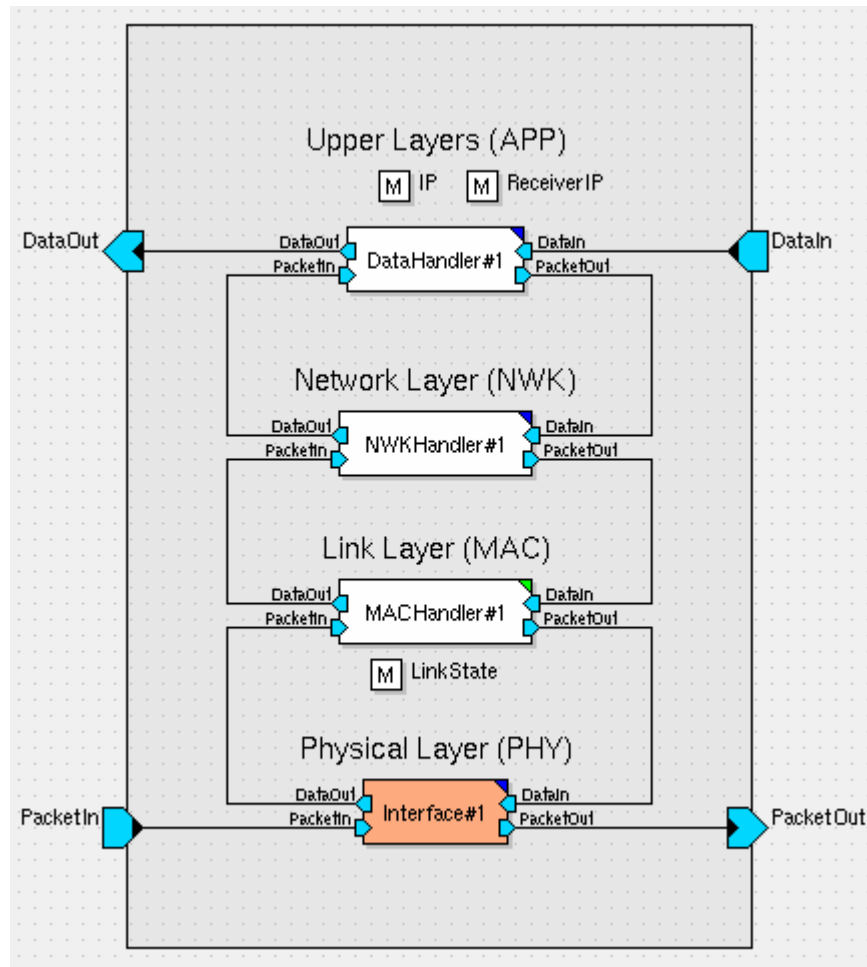


Abbildung 4-14: Protokollstack des Coordinators

Ein Coordinator hat deshalb vier Schichten.

- Upper Layers → DataHandler
- Network Layer → NWKHandler
- Link Layer → MACHandler
- Physical Layer → Interface

Die Funktionsblöcke der Schichten sind den Ebenen entsprechend untereinander verbunden und können eingehende Daten weiterleiten. IP-, ReceiverIP- und LinkState-Memory sind analog zum Enddevice modelliert.

4.5.2 Protokollstack bei SMAC

Die beiden Geräteklassen Sender und Receiver haben den gleichen Protokollstacks. Je nach Modellierung können diese Partitions gleichzeitig sowohl Senden und als auch Empfangen. Da bei dem späteren Hardwaretest die Module aber nur Sender oder nur Empfänger sein können, wurde diese Beschriftung gewählt, um Unstimmigkeiten im Vorfeld zu vermeiden. Denn bei der Hardware wird in Realität nur der entsprechende Gerätestatus vergeben und analog dazu wurde das Modell entworfen. Dieses orientiert sich an dem in 4.5.1. erläuterten Enddevicestack bei IEEE 802.15.4/ZigBee. Schließlich benutzt SMAC im Grunde die gleiche Übertragungsmechanik mit lediglich ein paar Änderungen bzgl. des Kanalzugriffs und der Paketverwaltung.

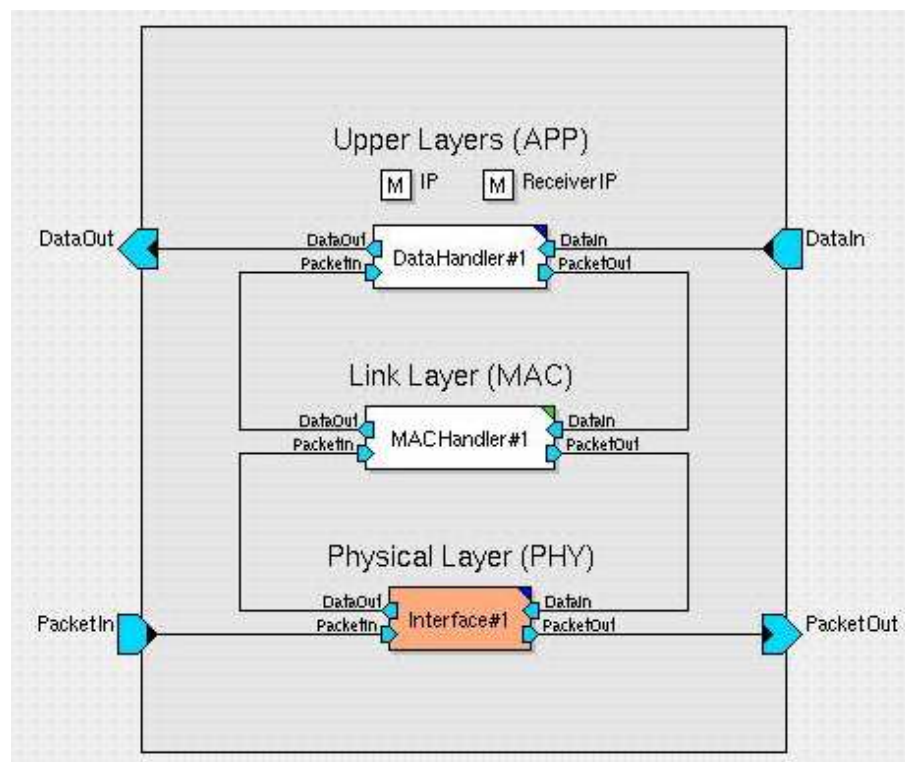


Abbildung 4-15: Protokollstack des Senders/Receivers

Sowohl der Sender als auch der Receiver hat drei Schichten:

- Upper Layers → DataHandler
- Link Layer → MACHandler
- Physical Layer → Interface

Da SMAC keine sicherheitsrelevanten Features enthält, wurde der LinkState-Memory nicht übernommen.

4.6 Schichten

Jede einzelne Schicht ist im Toplevel gleich aufgebaut, wie bei der Erläuterung des Grundaufbaus vom Protokollstack in 4.5 ersichtlich. Die Inputports DataIn bzw. PacketIn empfangen Frames aus der überliegenden bzw. untergeordneten Schicht, während die Outputports DataOut bzw. PacketOut Frames an die untergeordnete bzw. überliegende Schicht weiterleiten.

4.6.1 Schichten des IEEE 802.154/ZigBee-Protokollstacks

- Upper Layers

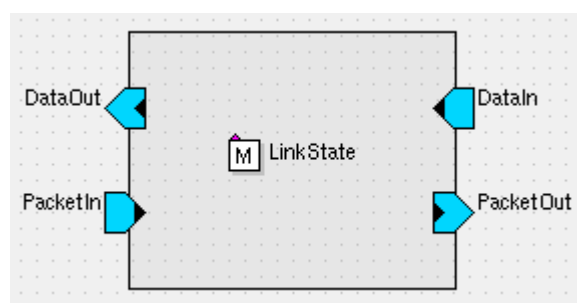


Abbildung 4-16: DataHandler des IEEE 802.154/ZigBee-Protokollstacks

Der DataHandler empfängt über den DataIn-Port einen vom Poissonverteiler generierten Zahlenwert und wandelt diesen in eine APPFrame-Datenstruktur um.

Da dieser Vorgang Zeit in Anspruch nimmt wird das Output-Paket erst nach einer kleinen Verweildauer in dem Block an den PacketOut-Port angelegt. Je nach Gerätetyp kommt es zum Versenden der Frame an den NWKHandler des Coordinators oder den MACHandler des Enddevices.

Empfangene Daten der darunter liegenden Schicht, die über den PacketIn-Port ankommen, entpackt der DataHandler und setzt die Ankunft des Paketes als kurzes Triggersignal an den DataOut-Port, welches dann im XMGraph auf einer Zeitskala als Punkt dargestellt wird.

Der im DataHandler verlinkte Memory LinkState steht standardmäßig auf dem Wert 0, d.h. solange dieser Status vorliegt werden alle Triggers am DataIn-Port geblockt. Erst wenn der LinkState nach dem ACL-Request über den Permission-MACFrame des Coordinators auf 1 gesetzt wird, kann die Datenpaketerstellung erfolgen.

- Network Layer

Die Implementierung des NWKHandlers bleibt nur dem Coordinator vorbehalten.

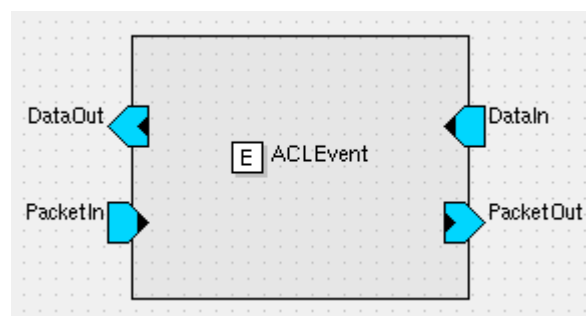


Abbildung 4-17: NWKHandler des IEEE 802.15.4/ZigBee-Protokollstacks

Über die In- und Outputs können Frames der angrenzenden Schichten empfangen, bearbeitet, ausgewertet und weitergeleitet werden.

Der NWKHandler hat zwei Aufgaben. Er leitet die vom DataHandler übermittelten Datenpakete unbearbeitet an den MACHandler weiter und ist für die Netzwerkverwaltung zuständig.

Um unerwünschte Besucher im Netzwerk zu vermeiden, verfügt jedes ZigBee-Netzwerk über eine einfache Firewallfunktionalität. Mit Hilfe der Access Control List (ACL) können gezielt Geräte aus dem Netzwerk ausgegrenzt oder in ihrem Nutzungsbereich eingeschränkt werden.

Dazu sendet der Coordinator zur Netzinitialisierung einen Broadcast, der alle Geräte in Reichweite dazu veranlassen soll, sich mittels IP- und MAC-Adresse im Netzwerk zu autorisieren. Alle eingehenden Geräte-Antworten werden vom Coordinator ausgewertet, indem die IP- und MAC-Informationen des vom Enddevice übermittelten Frames mit einer handbearbeitenden Liste (ACLList) verglichen werden.

Ist dieser Abgleich positiv darf das Gerät sich im Netzwerk aufhalten, falls IP und MAC nicht in der Liste vorkommen wird das entsprechende Gerät nicht ins Netzwerk aufgenommen.

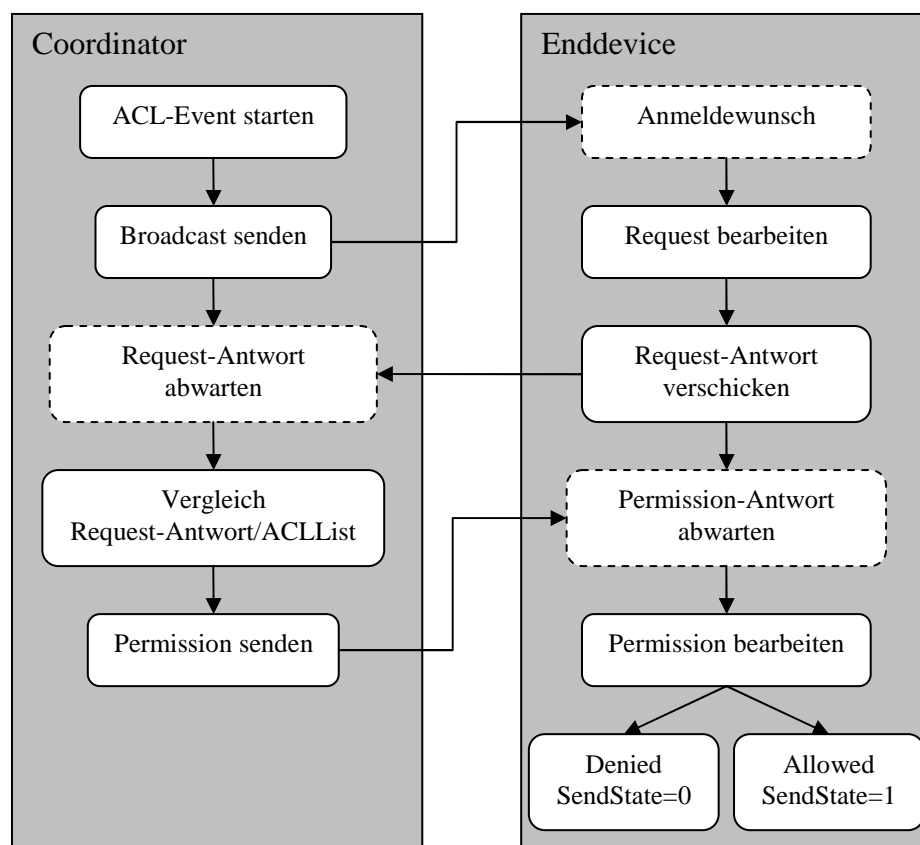


Abbildung 4-18: ACLEvent des IEEE 802.154/ZigBee-Protokollstacks

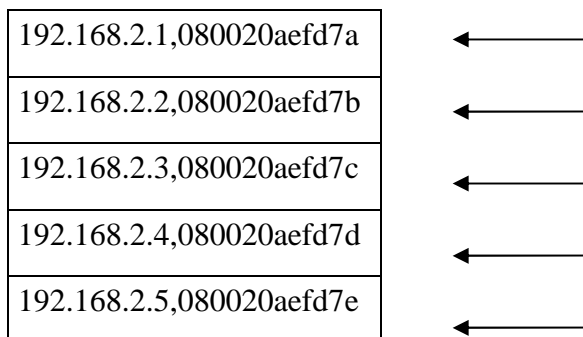
Damit am Anfang der Simulation zuerst die Netzwerkverwaltung und Geräteidentifizierung stattfinden kann, wurde der ACL-Request mittels eines Events realisiert. Zum Simulationszeitpunkt 0 wird ein NWKFrame mit einem ACLSubset erstellt, an die MAC-Schicht zur Weiterbearbeitung weitergeleitet und von dort aus als Broadcast an alle Enddevices verschickt.

Deren Antwort-Frames mit verändertem ACLSubset werden ausgelesen, um die darin enthaltene IP- und MAC-Adresse in einem mit Komma getrennten String zusammenzufassen.

IP: 192.168.2.1
 MAC: 080020aefd7e } 192.168.2.1, 080020aefd7e

Dieser String wird nun mit den Einträgen der ACLList verglichen, die von Hand durch den Nutzer vor der Simulation eingetragen wurden.

ACLList: String: 192.168.2.1, 080020aefd7e



Falls die Abfrage mit den Einträgen der Stringarray-ACLList erfolgreich ist, wird ein NWKFrame mit PermissionSubset generiert, an die MAC-Schicht weitergeleitet und an das entsprechende Enddevice verschickt. In diesem Frame steht dann, ob der LinkState des Devices auf 1 (=darf Datenpakete Senden/Empfangen) gesetzt wird oder auf 0 (=darf keine Datenpakete Senden/Empfangen) bleibt.

- Link Layer

Der MACHandler hat eine Fülle von Aufgaben zu bewältigt. Dazu gehört das Erstellen der zu übertragenden MACFrames (FrameBuilder), der Kanalzugriffsalgorithmus CSMA/CA samt DataLogging (ChannelHandler) und das Queueing der Daten bei besetztem Channel (FIFOQueue).

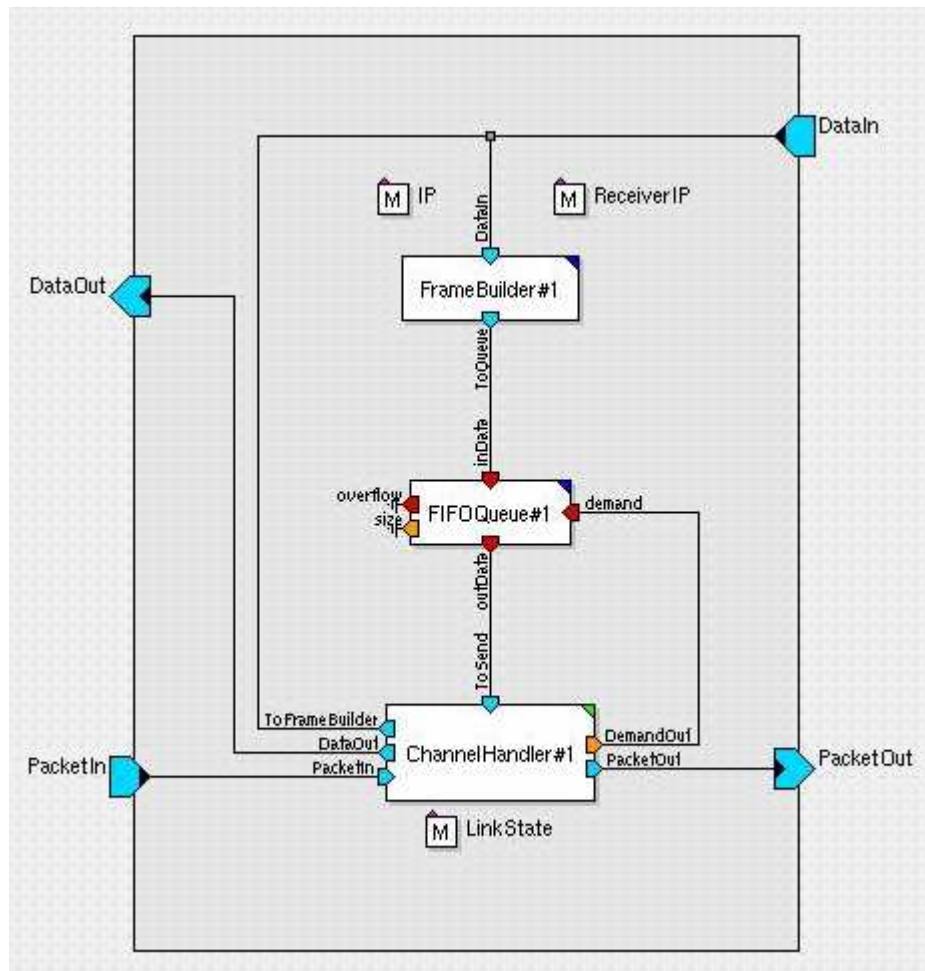


Abbildung 4-19: MACHandler des IEEE 802.154/ZigBee-Protokollstacks

Um die Übersichtlichkeit zu wahren, wurden die Aufgabenbereiche in separate Blöcke gepackt. Grundsätzlich wird jedes Frame einer höheren Schicht im FrameHandler bearbeitet, von dort an die MLD-eigene FIFOQueue weitergeleitet und anschließend über den ChannelHandler-Baustein, in dem der CSMA/CA-Block integriert ist, versendet.

FrameBuilder

Eingehende Frames werden über ihr Subset in die entsprechenden MACFrames gewandelt (siehe 4.7 ff).

Für DataSubsets wird als erstes das Feld Size (Größe in Byte) ausgelesen und mit dem maximalen Payload verglichen. Falls die Size kleiner ist als der maximale Payload, wird das Frame sofort in einen neuen MACFrame gewandelt, andernfalls wird die Size entsprechend auf einzelne MACFrames aufgeteilt. Dabei wird der gesamte verfügbare Payload solange ausgelastet, bis ein kleiner Rest übrig bleibt, der dann einen letzten kleineren MACFrame stellt.

Im Falle eines ACLSubsets wird entschieden, ob es sich dabei um die Aufforderung zur Geräteidentifikation oder die ACL-Request-Antwort handelt. Bei Ersterem kommt das Frame vom NWKHandler, der diese als Event erstellt hat. Der Algorithmus füllt die Felder IP und MAC des neu erstellten MACFrames mit Platzhaltern auf und sendet sie an die nachfolgende Queue. Zur Bearbeitung des ACL-Request kamen die Frames vom CSMACA-Block des ChannelHandlers. IP und MAC werden mit den Gerätedaten beschrieben. Zusätzlich wird das ACLSubset neu generiert und der Request als beantwortet deklariert.

PermissionSubsets werden mit der IP- und MAC-Adresse des betroffenen Gerätes versehen und an die Enddevices zur Übertragung fertig gemacht.

Alle bearbeiteten Frames werden nach Abarbeitung der entsprechenden Framebildung an die Queue weitergeleitet.

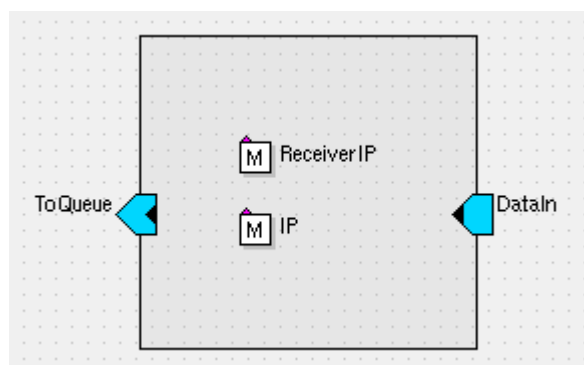


Abbildung 4-20: FrameHandler des MACHandlers bei IEEE 802.15.4/ZigBee

Queueing

Im Modell kommt eine Standardqueue aus der MLD-Library zur Anwendung, die auch unverändert integriert wurde.

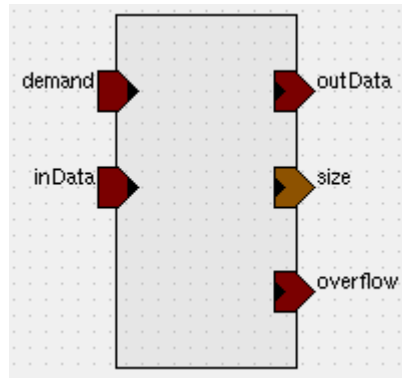


Abbildung 4-21: FIFOQueue des MACHandlers bei IEEE 802.15.4/ZigBee

Alle eingehenden Daten werden mittels eines FIFO-Algorithmus (First In First Out) gespeichert. Falls die Queue leer sein sollte, wird ein ankommendes Frame sofort wieder aus der Queue geworfen. Zur Laufzeit und bei gefüllter Queue muss ein Triggersignal (Demand-Input) abgewartet werden. Erfolgt dieses, wird das nächste Paket an den CSMA-CA-Baustein durchgelassen. Ein integrierter Zähler erfasst den aktuellen Füllstand der Queue der mit jedem Triggersignal oder dem Verlassen eines Paketes neu berechnet wird. Über einen Parameter kann die maximale Größe des Speichers festgelegt werden. Die Implementierung erfasst aber nur die Anzahl der Elemente (Frames) in der Queue, jedoch nicht deren verbrauchter Speicherplatz. Die allgemeine Funktionsweise reicht für Simulationszwecke trotzdem. Koordinatoren haben wegen ihres größeren internen ROM's auch eine größere Queue zur Speicherung von Frames. Ist der maximale Füllstand erreicht, verfallen alle eingehenden Daten bis die Queue wieder neue Daten aufnehmen kann.

ChannelHandler

Der ChannelHandler ist ein Modul und beinhaltet eine CSMA/CA-Primitive für die Kanalsteuerung und eine Logging-Primitive zur Veranschaulichung der Geräteauslastung.

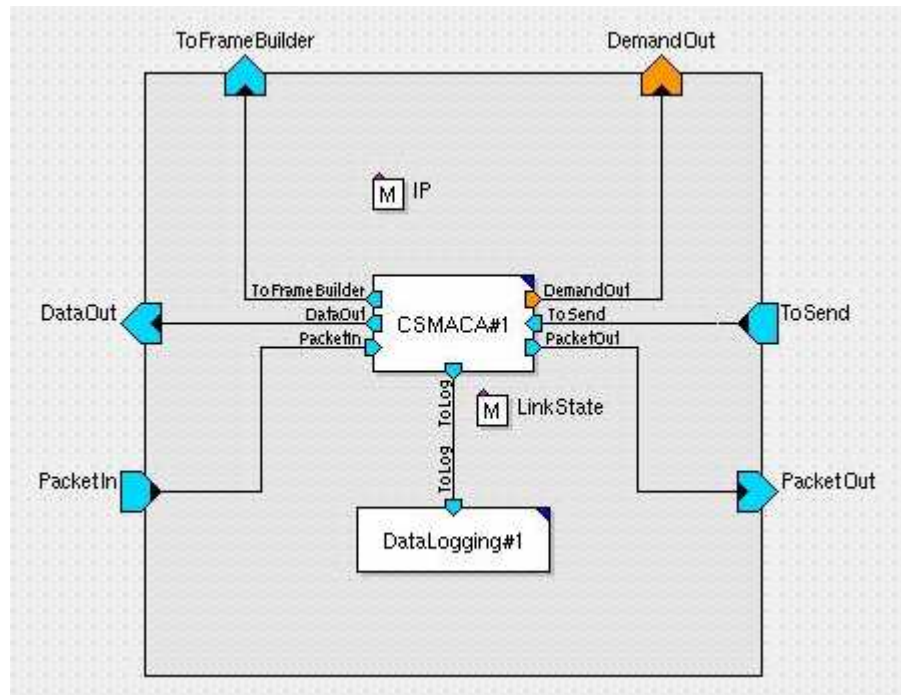


Abbildung 4-22: ChannelHandler des MACHandlers bei IEEE 802.154/ZigBee

CSMACA

Wie in 2.3.2 erwähnt kommt für den Kanalzugriff das CSMA/CA-Verfahren (Carrier Sense Multiple Access with Collision Detection) zum Einsatz. Da alle zu übertragenden Daten diesen Algorithmus verwenden müssen, bildet der CSMACA-Block die letzte Instanz beim Framedurchfluss innerhalb des MACHandlers.

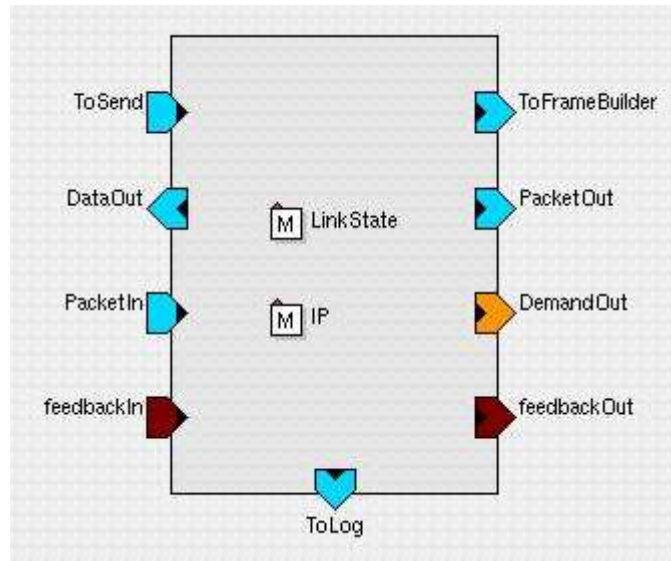


Abbildung 4-23: CSMACA-Block des MACHandlers bei IEEE 802.15.4/ZigBee

Grundidee war die Realisierung einer externen Ansteuerung des Channels. Mittels einer Zugriffsfunktion kann der Channel entweder auf besetzt (=1) oder frei (=0) geschaltet werden. Die Implementierung wurde im ESL-Target realisiert.

Der CSMACA-Baustein kann alle Frameformen bearbeiten, die am PacketIn- oder ToSend-Port anliegen. Grundlage ist die Analyse, welches Subset in den eingehenden MACFrames enthalten ist.

ACL- und PermissionSubsets können den CSMACA-Block passieren, während im Falle eines DataSubsets erst der SendState ausgelesen werden muss. Ist dieser Status 0 (also „denied“) dürfen DatenFrames weder empfangen noch gesendet werden.

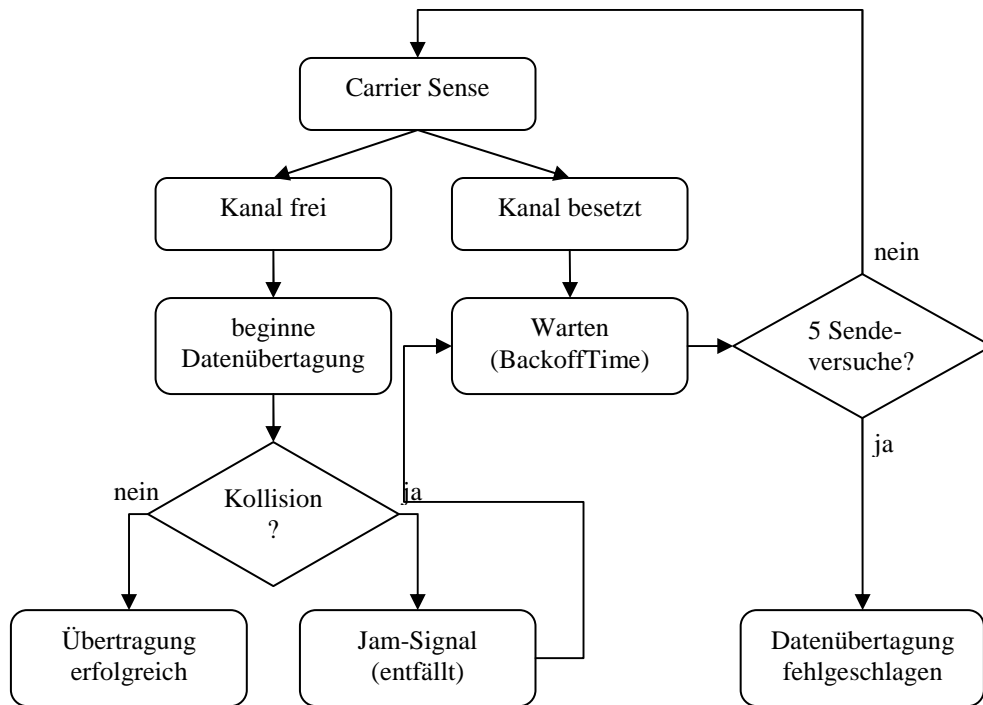


Abbildung 4-24: CSMA/CA-Ablaufplan bei IEEE 802.154/ZigBee

Soll nun ein Kanalzugriff erfolgen, muss dieser zuerst auf Kanalfreiheit abgefragt werden. Ist der Channel frei, wird das Paket kurz gehalten und dann ein Sendeversuch gestartet. Sollte in diesem Moment der Channel besetzt sein, wäre eine Kollision die Folge. Mit Hilfe eines Workarounds zur Laufzeit, können solche Kollisionen abgefangen werden. Dazu wird einfach zur Zeit des Sendeversuchs einfach noch einmal der Channel abgefragt. Ist der Channel besetzt, wird das Paket für eine gewisse Zeit gepuffert und zu einem späteren Zeitpunkt erneut versucht zu versenden. Nach fünf erfolglosen Sendeversuchen geht das Paket verloren.

Dieses sehr einfache Verfahren kann man durch zusätzliche Verfeinerungsmethoden bzgl. der Paketsendeleistung verbessern. Die Wahrscheinlichkeit, dass bei einer Vielzahl von Geräten im Netzwerk ein gleichzeitiger Sendeversuch von mehreren Geräten stattfindet ist sehr hoch. Abhilfe schafft eine zufällige BackoffTime, die bei jedem ersten Sendeversuch in Intervall von $[0, 2^{BE}-1]$ Zeitslots (1 Zeitslot = 302 μ s) ermittelt wird. BE ist der BackoffExponent, der wiederum eine Zufallszahl zwischen 0 und 5 darstellt. Somit ist das maximale BackoffTimeIntervall $[0, 31]$ Zeitslots. Insgesamt gibt es

nun 31 verschiedene Startzeitpunkte für eine Übertragung. Die Wahrscheinlichkeit, dass nun zwei oder mehr Geräte gleichzeitig auf den Channel zugreifen ist sehr gering.

Liegt nun ein Datenpaket am ToSend-Port an, erstellt das Modul zuerst eine Kopie des Paketes. Dieses ist jetzt solange gespeichert, bis der Senderversuch erfolgreich war oder das Paket verworfen werden muss.

Falls der Channel frei sein sollte, wird das Paket noch für die Zeit der berechneten BackoffTime gehalten und erst dann verschickt. Theoretisch kann es trotzdem zu einer Kollision kommen. Da im Modell aber keine elektrischen Messungen durchgeführt werden können, kommt hier ein kleiner Trick zur Anwendung. In dem Moment, wo das Paket den Channel belegen würde, erfolgt noch einmal eine Abfrage auf den Status. Wäre der Channel dann besetzt, ist eine Kollision die Folge. Weitere Senderversuche folgen dann unter Berücksichtigung des zuerst initialisierten Zeitintervalls. Der BackoffExponent wird mit jedem neuen Senderversuch um 1 erhöht bis zum maximalen Wert 5, was eine Verlängerung der Wartezeit des Paketes im Gerät zur Folge hat. Die berechnete Verzögerung wird dazu genutzt, das Retriggern des Blockes zu ermöglichen, indem auf die ArrivalTime die BackoffTime aufaddiert wird.

Das Retriggern des Blockes wurde mit Hilfe des MLD-eigenen Modules „RepeatStar“ umgesetzt. Dieses Modul erweitert den CSMA-Block um einen FeedbackIn- und einen FeedbackOut-Port, die voneinander abhängig sind. Wenn über interne Codebefehle der FeedbackOut-Port einen Zeitwert erhält, wird der Block nach Ablauf dieser Zeit über den FeedbackIn-Port angestoßen und kann mit der Abarbeitung fortsetzen. So ist es möglich eingehende Daten solange in dem Block zu behalten, bis die Kanalabfrage erfolgreich ist und der Frame verschickt werden kann oder nach fünf erfolglosen Senderversuchen der Frame verfällt. In beiden Fällen erfolgt ein DemandOut zur Ansteuerung der FIFOQueue. Diese kann nun ein neues Paket an den CSMA-Block leiten und der Algorithmus zur Paketversendung startet erneut.

Codeauszug aus CSMA/CA-Block:

```
if (ToSend.dataNew) //Daten am Eingangsport
{
    TypeRef tData=ToSend.get(); // Daten von Eingangsport abholen
    mCurrentPacket=tData; // Eingangsdaten speichern
    mTry=0; // Laufvariable des Senderversuchs, init=0
    tUtilizationState=mChannelInformation->getUtilizationState(); // Kanal-
                                                                    abfrage

    if (tUtilizationState == false) // freier Channel
    {
        mBE=3; // Standardinitialisierung BackoffExponent falls
                Channel frei [3]
        tBTI=rand()%(mBE+1); // BackoffTimeIntervall [0,...,7] TimeSlots
    }

    else
    {
        mBE=rand()%BackoffExp()+1; //Initialisierung falls Channel belegt
[1,...,5]
        tBTI = rand()%(mBE + 1); //BackoffTimeIntervall = [0,...,31] TimeSlots
    }

    refireAtTime(arrivalTime + BackoffTime(tBTI)); // verzögertes Triggern des
                                                    Block um Zeit der
                                                    berechneten Zeitslots
}

if (feedbackIn->dataNew) // Triggersignal am FeedbackIn-Port
{
    feedbackIn->get(); // Triggersignal auslesen
    mTry++; // Laufvariable der Senderversuche um 1 erhöhen

    tUtilizationState = mChannelInformation->getUtilizationState();

    if (tUtilizationState == false)
    {
        mChannelInformation->setUtilizationState(true); // Channel auf 'belegt'
                                                        setzen
        PacketOut.put(arrivalTime + 0.01) << mCurrentPacket; // gespeichertes
                                                                Paket senden
        DemandOut.put(arrivalTime + 0.01) << tDemand; // Triggern der Queue
                                                                zur Paketanforderung
    }

    else if (mTry <= 5) // Channel besetzt, Senderversuch größer 5?
    {
        mBE++;

        if (mBE > 5)
        {
            mBE = 5; // Senderversuch kleiner 5, BackoffExponent bleibt 5
        }

        refireAtTime(arrivalTime + BackoffTime(mBE));
    }

    else
    {
        DemandOut.put(arrivalTime) << tDemand; // Paket nach 5 Senderversuchen
                                                verworfen, Triggern der
                                                Queue für neues Paket
    }
}
}
```

Auf die Modellierung des Jam-Signals wurde verzichtet, da über den UtilisationState ein freier bzw. besetzter Channel signalisiert wird und mit Hilfe des oben beschriebenen Workarounds eine abstrakte, vereinfachte Kollisionsabfrage erfolgt.

DataLogging:

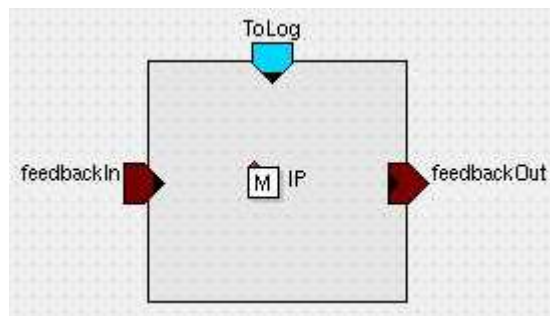


Abbildung 4-25: DataLogging-Block

Über den ToLog-Input ankommende Frames des CSMACA-Blocks analysiert das DataLogging bezüglich der übertragenen Datengröße und akkumuliert diese in einem vorher festgelegten Loggingintervall auf. Zur Veranschaulichung der Auslastung des Channels, der Gerätebelastung, der Größe der Daten und dem Verlinkungszustand des Netzes kommt ein Performancemonitor zum Einsatz. Dieser ist in der in 4.3.2 kurz erläuterten ESL-Umgebung implementiert. Die im Netzwerk befindlichen Geräte melden sich per IP-Adresse an dem Performancemonitor an und werden dann in regelmäßigen Abständen auf ihre empfangenen Daten abgefragt.

Performance Monitor			
<ul style="list-style-type: none"> <input checked="" type="checkbox"/> IP Matrix <input checked="" type="checkbox"/> SMACTest.SMACChannel#1.Channel#1 <input type="checkbox"/> Usage Rate (Average:1.586%, Blank:14.84%, Full:0%) 			
x-axis = sender y-axis = receiver	192.168.0.2	192.168.0.1	
192.168.0.2	0 (0%) 0 (0%)	244 (0%) 45628 (51.44%)	
192.168.0.1	0 (100%) 43066 (48.56%)	0 (0%) 0 (0%)	

Abbildung 4-26: Performance Monitor

Das linke Teilfenster zeigt die wichtigen Informationen zur durchschnittlichen Channelauslastung, sowie die Zeitanteile von Voll- und Leerauslastung in Prozent an.

Im rechten Teilfenster wird die Gerätematrix dargestellt. Alle am Performancemonitor angemeldeten Geräte werden gegenübergestellt, um den Datenverkehr verfolgen zu können. Neben der aktuellen Auslastung der Geräte im eingestellten Zeitfenster des LoggingIntervals, werden auch die empfangen Bits mitgezählt. Auslastungen über 20% erscheinen als rote Prozentzahlen.

- Physical Layer

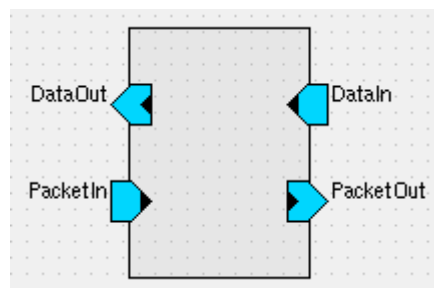


Abbildung 4-27: Interface des IEEE 802.154/ZigBee-Protokollstacks

Die letzte Schicht des Protokollstacks bildet das Interface des Physical Layers. Wie in 4.3.2 beschrieben entstammt die Implementierung aus dem ArchitectureBlockSet und bleibt unverändert. Eingehende Frames höherer Schichten werden in eine dem Channel verständliche Form umgewandelt und ausgegeben. Packet-Inputs von Seiten des Channels werden wieder ausgepackt und die darin enthaltenen MACFrames an die höheren Schichten durchgereicht.

4.6.2 Schichten des SMAC-Protokollstacks

Das Schichtenschema bei SMAC ist analog zu IEEE 802.154/ZigBee aufgebaut, hat aber im Vergleich dazu einen kleineren Funktionsumfang.

- Upper Layers

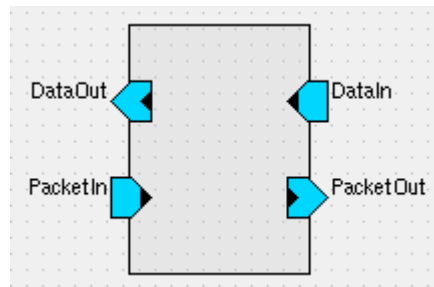


Abbildung 4-28: DataHandler des SMAC-Protokollstacks

Die Geräte bei SMAC sind automatisch miteinander verbunden und können sofort Daten senden und empfangen. Deshalb entfällt der komplette Event zur ACL und somit auch Statusmemory für den LinkState. Sofort zu Simulationsbeginn können Daten generiert werden.

- Link Layer

Der grundlegende Aufbau des MACHandlers entspricht dem Schema der IEEE802.15.4/ZigBee-Implementierung. Es fanden lediglich Änderungen zur Framebehandlung und des Kanalzugriffs statt.

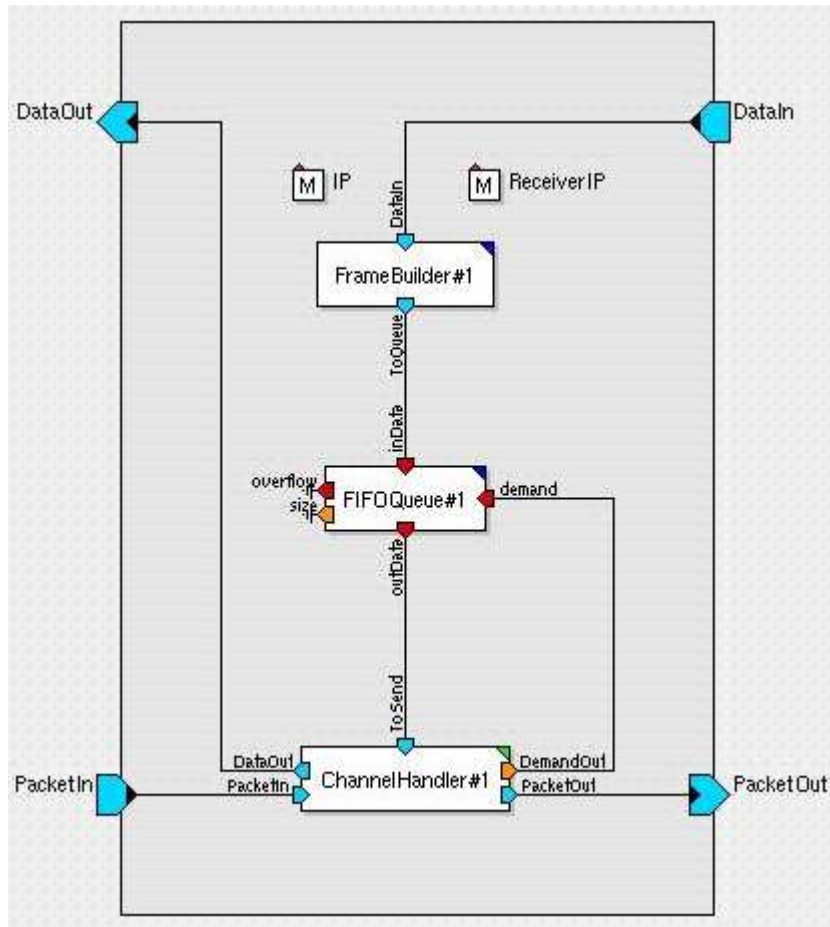


Abbildung 4-29: MACHandler des SMAC-Protokollstacks

Da hier nur MACFrames mit DataSubset verarbeitet werden entfällt die Rückkopplung vom CSMACA-Block auf den FrameBuilder.

Die komplette Verarbeitung von ACL- und PermissionSubsets entfällt ebenso wie der LinkState, da kein ACLEvent stattfindet.

Der CSMACA-Block im Channelhandler erhielt eine Anpassung auf den stark reduzierten CSMA/CA-Algorithmus bei SMAC.

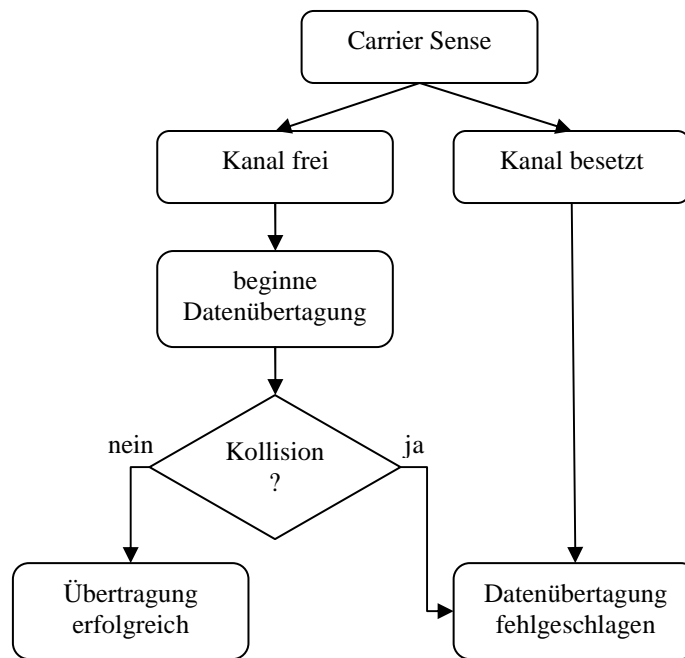


Abbildung 4-30: CSMA/CA-Ablaufplan bei SMAC

- Phy Layer

Der Interface-Block entspricht wie bei IEEE 802.15.4/ZigBee der Vorgabe aus dem ArchitectureBlockSet.

4.7 Frames

Für eine Datenübertragung zwischen den zwei Geräten müssen MAC-Frames erstellt werden, welche bei allen drei Übertragungsarten zum Einsatz kommen. Die dazu modellierte Datenstruktur basiert auf Grundlage der MLD-eigenen Datenstrukturverwaltung. Deshalb entfällt ein separates Handling durch den Anwender und erleichtert so die Erweiterung und Anpassung. Der grundlegende Aufbau bleibt immer gleich und sieht folgendermaßen aus.

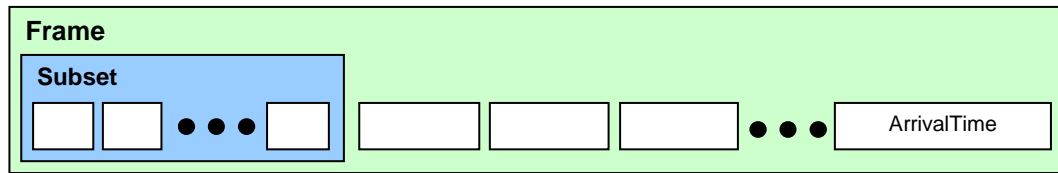


Abbildung 4-31: allgemeiner Aufbau der Frame-Datenstruktur

Die übergeordnete Datenstruktur des MACFrames beinhaltet alle wichtigen Informationen die für die einzelnen Protokollschichten erforderlich sind. Diese variieren entsprechend der Bearbeitungsstufe während der Simulation.

- AppFrame
- NWKFrame
- MACFrame
- ChannelPacket

Für eine leichtere Erkennung eines bestimmten Frames wurden Subsets eingebaut, die entsprechend der verarbeitenden Schicht weitere Datenfelder zur Informationsspeicherung enthalten.

- DataSubset
- ACLSubset
- PermissionSubset

Durchlaufen die einzelnen Frames die modellierten Schichten werden sie entsprechend angepasst, mit neuen Informationen gefüllt, unter Umständen neu verpackt und weitergeleitet.

4.7.1 AppFrame der Anwendungsschicht

Der AppFrame bildet den Anfang des in 2.3.2 vorgestellten DataFrame und wird von der Anwendungsschicht generiert. Im Laufe der Weiterleitung durch die

Schichten wandelt sich der generierte AppFrame zu einer MACFrame-Datenstruktur.

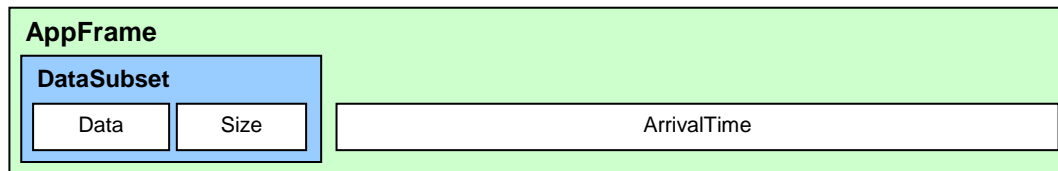


Abbildung 4-32: Aufbau der AppFrame-Datenstruktur mit DataSubset

Er enthält das DataSubset, welches die eigentlichen Daten (Data) sowie deren Größe in Byte (Size) speichert, und die ArrivalTime.

4.7.2 NWKFrame der Netzwerkschicht

Die Frames des NWKHandlers haben drei Funktionen. Erstens sollen ACL-Broadcasts zur Geräteverlinkung geschickt werden (als Event gestartet), zweitens müssen eingehende ACL-Requests mittels Anpassung des Subsets beantwortet werden, damit die entsprechenden Netzgeräte ihren LinkState setzen können und drittens müssen eingehende AppFrames der Anwendungsschicht an die MAC-Schicht durchgereicht werden.

Analog zum AppFrame besteht der NWKFrame ebenfalls nur aus einem Subset und der durchgereichten ArrivalTime.

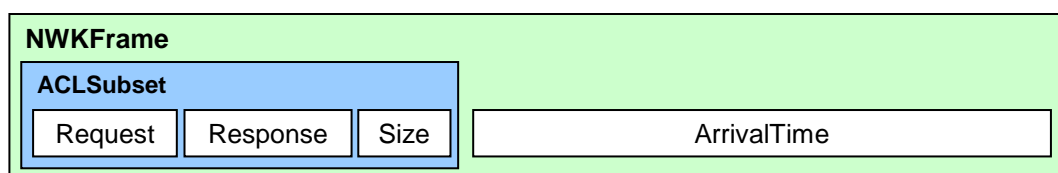


Abbildung 4-33: Aufbau der NWKFrame-Datenstruktur mit ACLSubset

Das ACLSubset zur Geräteidentifizierung hat die drei Felder. Request für die Aufforderung der Geräte sich zu autorisieren, Response für die Geräteantwort und Size für die dafür verwendete Größe in Byte.



Abbildung 4-34: Aufbau der NWKFrame-Datenstruktur mit Permission

Als Antwort auf die Identifizierungsaufforderung und den Abgleich der empfangen Geräte-Responsesdaten, sowie deren Abgleich mit der ACL-Liste generiert der NWKHandler einen NWKFrame mit PermissionSubset. Darin sind die jeweilige bearbeitete IP-Adresse der empfangenen Broadcast-Antworten, ein Feld Access (allowed = Gerät darf senden und empfangen, denied = Gerät darf nicht senden und empfangen) für den LinkState des entsprechenden Gerätes und die dafür verwendete Größe gespeichert.

4.7.3 MACFrame

Der MACFrame ist der eigentlich zu verschickende Frame. Da alle Frames der übergeordneten Schichten durch den MACHandler bearbeitet werden, kann dieser alle Subset enthalten. Ergänzt wird der Frame durch Zusatzfelder, die für eine Übertragung unabdingbar sind. Wie bei den anderen Frames auch, wird das Feld ArrivalTime mit der im aktuellen Zeitschritt gemessenen Zeit beschrieben.

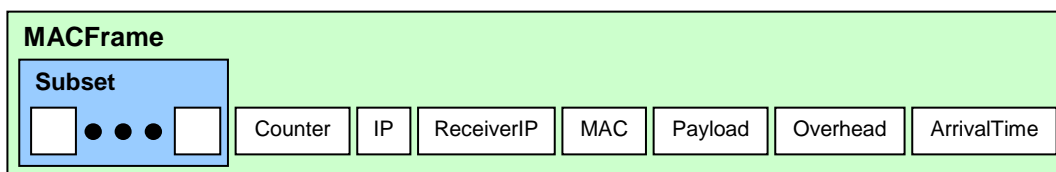


Abbildung 4-35: allgemeiner Aufbau der MACFrame-Datenstruktur

In jedem MACFrame wird ein Counter eingefügt der Rückschlüsse auf die vom Framegenerator erstellten MACFrames zulässt, da eine größere Datenmenge zur Übertragung auf viele Frames aufgeteilt werden muss. Die IP und ReceiverIP wird von den außen angegebenen Parametern gelesen und eingetragen. Somit können die Geräte den Channel auf Pakete an ihre Adresse abfragen und wissen

auch, von welchem Gerät dieses verschickt wurde. Das Feld MAC dient lediglich für den ACL-Broadcast als Platzhalter bzw. als Antwort darauf zum Abgleich mit der im NWKHandler gespeicherten ACL-Liste. Die in jedem Subset angegebene Größe (Size) bildet die Grundlage zur Berechnung des Payloads.

4.7.4 ChannelPacket

Zur Vervollständigung sei hier noch das ChannelPacket erwähnt, das aber zur vorgenerierten Modellumgebung des ArchitectureBlockSet gehört.

Jedes Frames, welches das Interface durchläuft und zur Versendung über den Channel bereit ist, wird in eine dem Channelbaustein verständliche Form gebracht. Dabei bleiben die eigentlichen zu übertragenden Daten unverändert. Sie werden in einer übergeordneten Datenstruktur verpackt und dann übertragen.

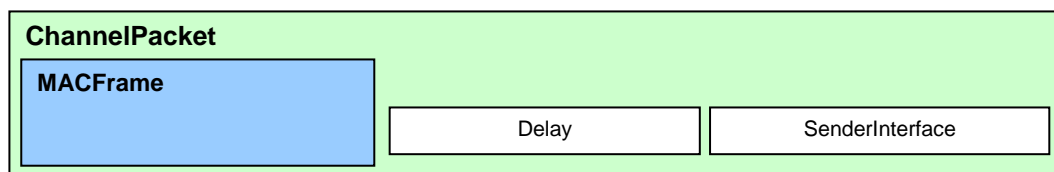


Abbildung 4-36: allgemeiner Aufbau der ChannelPacket-Datenstruktur

Das Feld Delay wird mit der Zeit beschrieben, mit der der Frame den Channel während der Übertragung besetzen würde und entsprechend verzögert. Damit bei einer Datenübertragung das Paket nicht wieder am Sender angelegt wird, wurde ein Feld SenderInterface hinzugefügt, in dem diese Möglichkeit ausgeschlossen wird. Dieses Echo kann man bei Bedarf aber auch einschalten.

4.8 Channel

Der Channel enthält neben den aus der ESL-Umgebung übernommenen Channel-Block eine Primitive zur Berechnung des Delays der entsprechenden Übertragungsart.

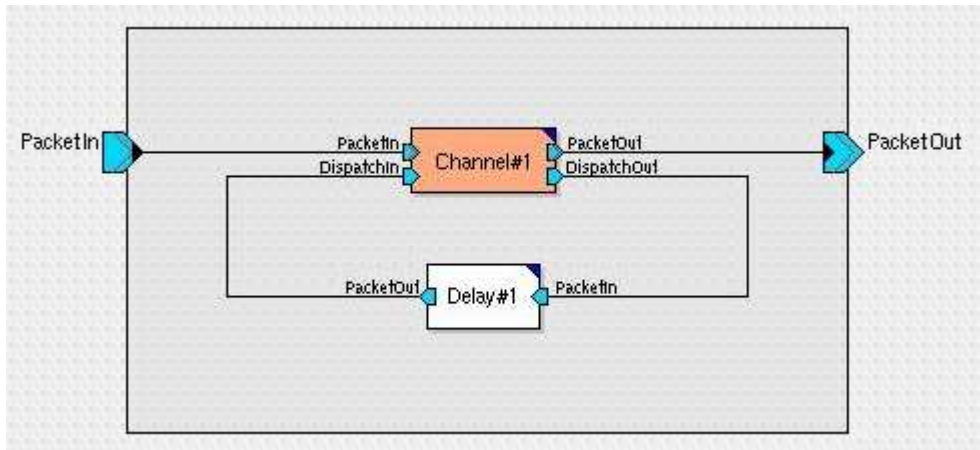


Abbildung 4-37: Channel-Block

Die Idee war, das Paket im Channel selber auszupacken und durch eine Bearbeitungsschleife zu schicken, in der unabhängig vom eigentlichen Channel-Block das Delay berechnet wird.

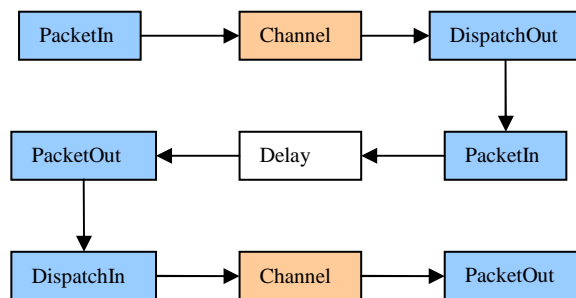


Abbildung 4-38: Paketdurchlauf im Channel

So kann man unterschiedlichste Delaybausteine implementieren und wahrt trotzdem die Spezifikation des Rahmensystems.

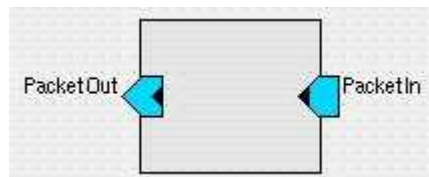


Abbildung 4-39: Delay-Block

Für die Bestimmung der Datenrate in Abhängigkeit zur Entfernung kommt eine einfache quadratische Interpolation zum Einsatz. Dieser mathematische Ansatz bildet die Grundlage für die anschließende Berechnung der zeitlichen Verzögerung bei einer Paketdatenübertragung.

Ausgangspunkt für die Interpolation sind zwei Werte der Verfügbarkeit des Netzes. Theoretisch sollte bei direkter Verbindung der Geräte und keinem Abstand voneinander die maximale Datenrate von 250kBit/s genutzt werden können, während am Randbereich des Netzes die Datenrate nahezu gegen 0 konvergiert, bis die Verbindung endgültig abbricht.

$$\text{Datenrate} = 0,0390625 \cdot (\text{Entfernung} - 80)^2$$

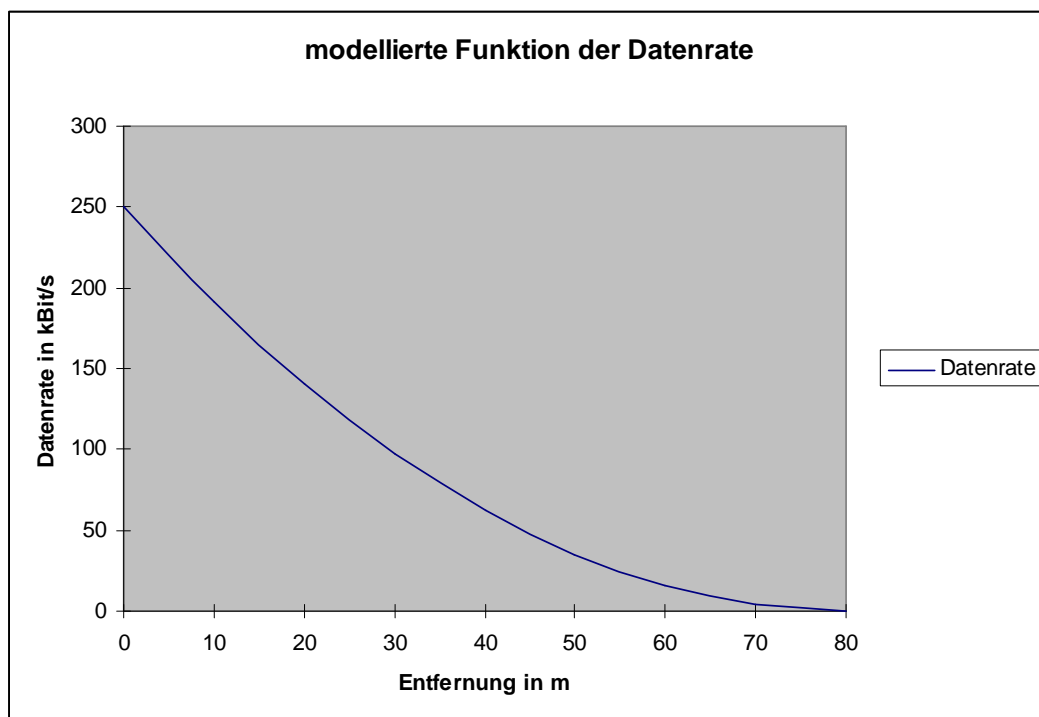


Abbildung 4-40: modellierte Funktion der Datenrate

Ab einer Entfernung von 80 m ist das technische Leistungsvermögen jedoch stark ausgeschöpft und bewegt sich an der Grenze des Machbaren. Größere Entfernungen sind nur unter optimalen Umgebungsbedingungen möglich.

Für die Modellierung von unterschiedlichen Netzwerken mit wechselnden geografischen Gegebenheiten muss diese Funktion angeglichen werden. Denn der

Outdoor-Bereich erlaubt wesentlich höhere Datenraten als ein vergleichbares Netzwerk im Inhouse-Bereich. Im der gewünschten Umgebung sollte dennoch zwingend eine Messung vorgenommen werden, um die tatsächlich verfügbare Datenrate modellieren und die untere Schranke des Datenratenwertes halten zu können.

Über den Parameter „Distance“ im Delay-Block kann eine Entfernung von 0 – 80 m eingetragen werden, um die entsprechende Datenrate der Verbindung zu berechnen. Werte unter 0 m oder über 80 m werden nicht akzeptiert und mit Hilfe einer Fehlermeldung im Commandfenster des MLDesigner kenntlich gemacht. Mit den oben genannten Parametern kann nun die Berechnung des Delays erfolgen. Das im Channel befindliche Paket enthält als Datenfelder den ‚Overhead‘ und den ‚Payload‘, deren Zahlenwerte die entsprechende Größe in Byte darstellen. Diese beiden Felder werden ausgelesen und addiert, um die komplette Größe des Frames zu erhalten. Der über die Entfernung ermittelte Wert aus der Datenratefunktion muss noch auf Byte/s umgerechnet werden. Das Delay berechnet sich dann aus:

$$Delay = \frac{\text{Gesamtframegröße}}{\text{Datenratenfunktionswert}}$$

4.9 Modellierungsunterschiede der Protokolle

SMAC: Als einfachstes Übertragungsverfahren verfügt SMAC über keinerlei Sicherheitsfunktionen. Zudem wird bei einem Übertragungswunsch maximal nur ein Sendeversuch durchgeführt. Bei besetztem Channel geht das Datenpaket verloren. Der Overhead beschränkt sich auf die Sende- und Ziel-IP, die MAC-Adresse des Senders dem Counter und der Arrivaltime – zusammen 28 Byte.

IEEE 802.15.4.: Dieser Standard bildet die Grundlage für ZigBee. Zusätzlich zum den CSMA/CA-Verfahren, welches bis zu fünf Übertragungsversuche zulässt, wurde ein einfaches Sicherheitsmanagement zum Geräte-Pairing eingebaut – die ACL.

ZigBee: Da ZigBee nur die oberen Schichten des Stacks darstellt, kann hier nur eine Veränderung der Anwendungsschicht erfolgen. Da aber lediglich die reine Datenübertragung betrachtet werden soll, bleibt die eigentliche Datengenerierung analog zu SMAC.

5 Praktische Validierung am Echtsystem

Als Grundlage für den Vergleichstest der ZEBRA-Module mit dem MLDesigner-Modell kommt der Oszilloskop TDS 220 der Firma Tektronix zum Einsatz.



Abbildung 5-1: Tektronix TDS 200-Reihe

Für die zeitliche Auswertung der Triggerzeitpunkte wird das Windows-Tool OpenChoice benutzt, welches von Tektronix speziell für die TDS-Oszilloskop-Reihe konzipiert wurde.

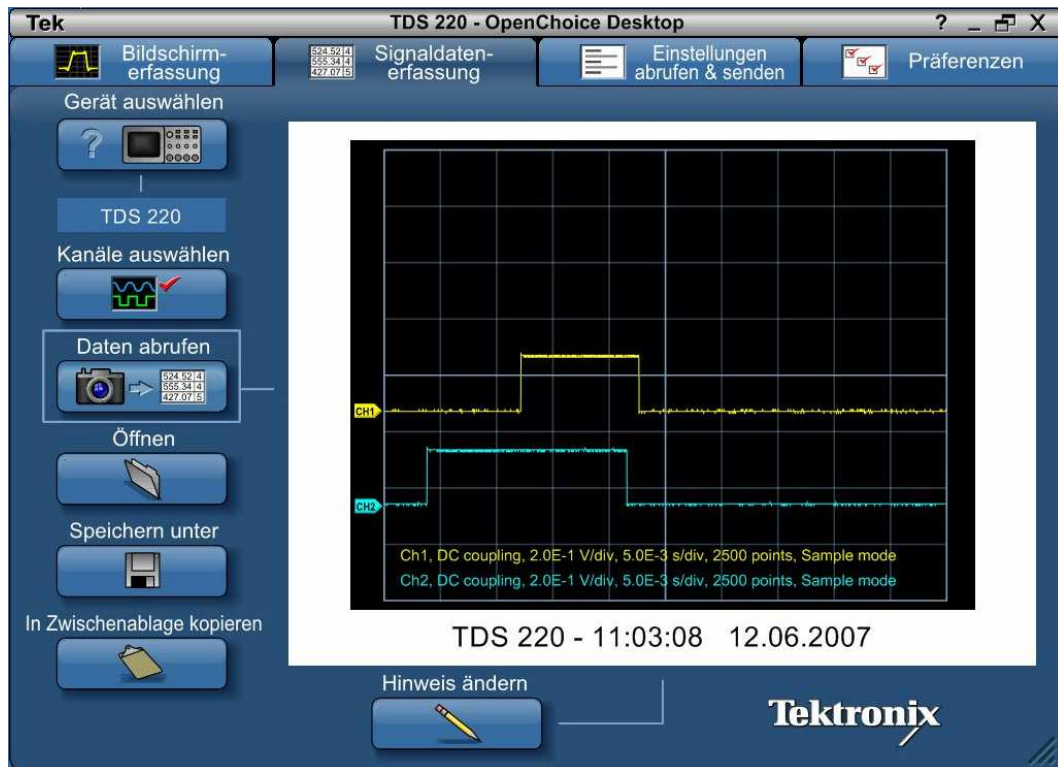


Abbildung 5-2: Benutzeroberfläche des OpenChoice Desktop

Die verwendeten ZERBA-Module entstammen dem ZEBRA2400-SK Starterkit der Firma senTec Elektronik GmbH.

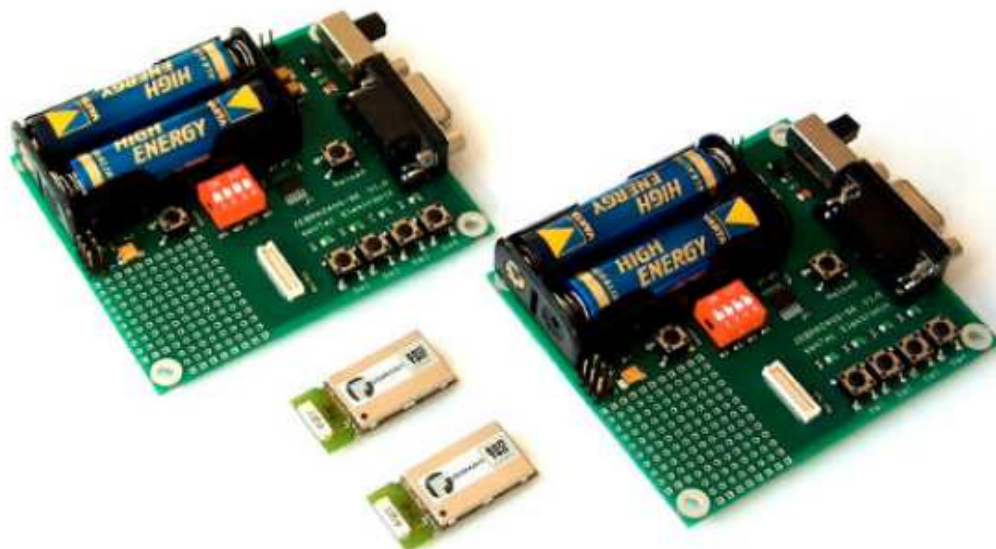


Abbildung 5-3: ZEBRA2400-SM

Zu dem Starterkit gehören zwei Basisboards, zwei ZigBee-fähige Stackmodule, eine ausführliche Dokumentation, sowie diverse Softwaretools zur Konfiguration des Anwendungswunsches – für SMAC-Anwendungen das Testtool und für IEEE 802.15.4/ZigBee-Anwendungen der Codewarrior von Freescale. [ST06]

5.1 Versuchsaufbau

Grundidee für den Vergleich von Hard- und Software ist eine Vereinheitlichung der Datenübertragung. Das Softwaremodell sollte dabei genauso viele Bytes übertragen wie die Hardware. Da für die ZEBRA-Module schon einige Anwendungsbeispiele vorliegen, ist es einfacher diese auf das MLDesigner-Modell zu adaptieren.

Als Vergleichsbeispiel wird ein Übertragungstest genutzt, der in gewissen Zeitabständen der Reihe nach Datenpakete verschickt, die einen immer größer werdenden Payload enthalten – angefangen bei 1 Byte, 16 Byte, 43 Byte, bis hin zu maximal 122 Byte. Dieser Routine triggert bei jedem Sendevorgang die auf den Modulen integrierten LED's. Wird ein Paket verschickt leuchtet eine LED beim Sender und wenn das Paket beim Empfänger angekommen ist, leuchtet auch dort eine LED. Die zeitliche Verzögerung zwischen den beiden aufleuchtenden LED's können mit einem Oszilloskop gemessen werden.

Analog dazu wird das Modell in der MLDesigner-Umgebung mit den gleichen Ausgangswerten versehen. Über die Arrivaltime in den Datenframes (siehe 4.7) kann hier ebenfalls die zeitliche Verzögerung der Datenübertragung vom Sender zum Empfänger ausgewertet werden.

Aufgrund der technischen Beschränkungen der Messung mittels des vorliegenden Oszilloskopen beträgt die Entfernung der Module lediglich 1 m. Dieselbe Entfernung erhält auch der Parameter „Distance“ im Modell (siehe 4.8).

5.2 Durchführung

Für einen Vergleich beider Systeme (Software und Hardware) ist es nötig, zwei Messungen durchzuführen. Als Grundlage dient bei beiden Durchführungen die SMAC-Übertragung.

- Hardware-Test

Die Messung mit Hilfe des Oszilloskops wurde zur Sicherheit mehrfach durchgeführt, da mögliche Störungen von außen die Ergebnisse beeinflussen können. Die anschließende Mittelung der Messwerte ergab die in Abbildung 5-5 eingetragenen Punkte.

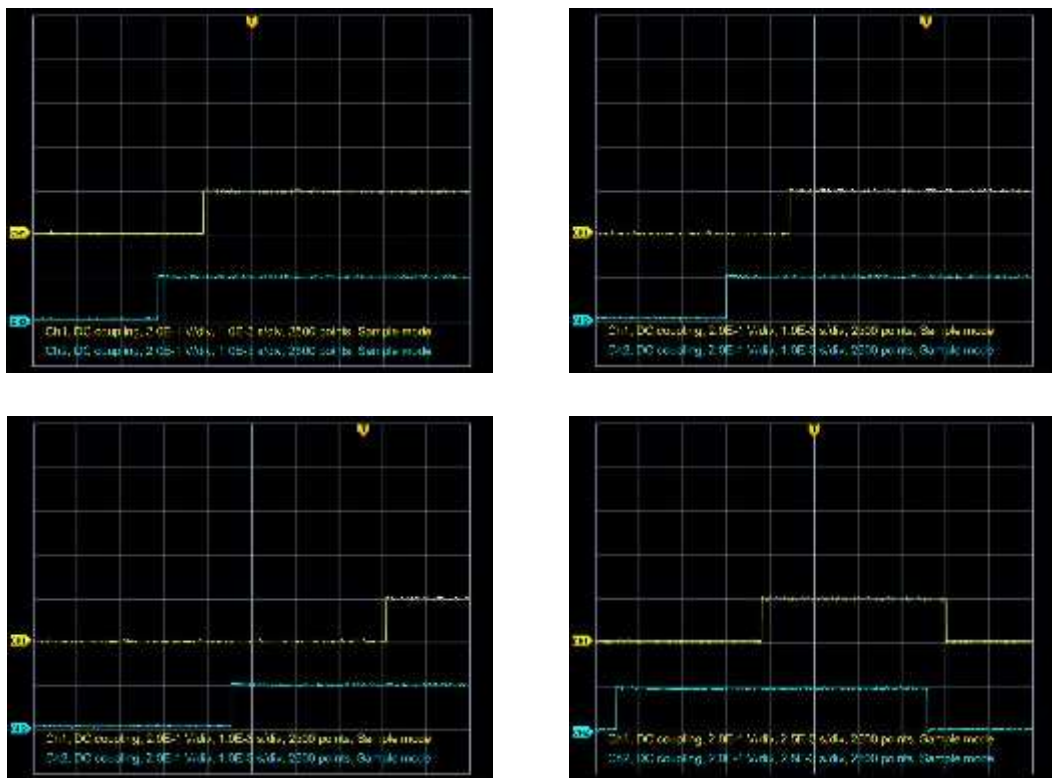


Abbildung 5-4: Oszilloskopmessbilder 1 Byte, 16 Byte, 43 Byte, 122 Byte (l.o., r.o., l.u., r.u.)

Die ersten drei Messungen wurden bei einer Zeitskala von 1 ms/cm ermittelt, die letzte Messung bei 2,5 ms/cm. Mittels einer im Oszilloskop integrierten Stop-

Funktion und Positionsverschiebungsmöglichkeit könnte die zeitliche Verzögerung zwischen den aufleuchtenden LED's gemessen werden.

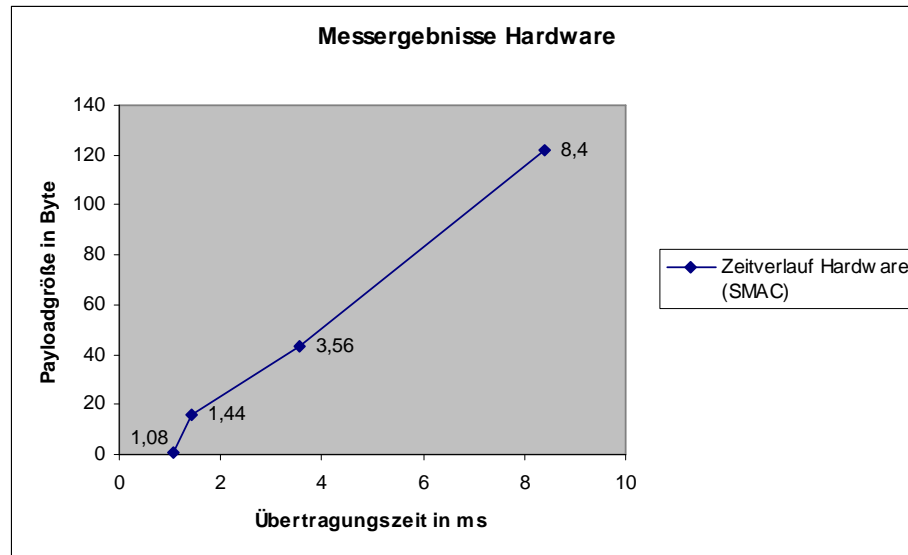


Abbildung 5-5: Messergebnisse Hardware bei SMAC

- Software-Test

Ein in dem Software-Modell generierter Datenframe ist von vornherein so eingestellt, dass er mit den erzeugten Frames der Hardware übereinstimmt. Für die zahlenmäßige Erfassung der Übertragungszeit im MLDesigner-Modell sind mehrfache Durchläufe nicht nötig, da durch den Simulationscharakter immer die gleichen Werte ausgegeben werden.

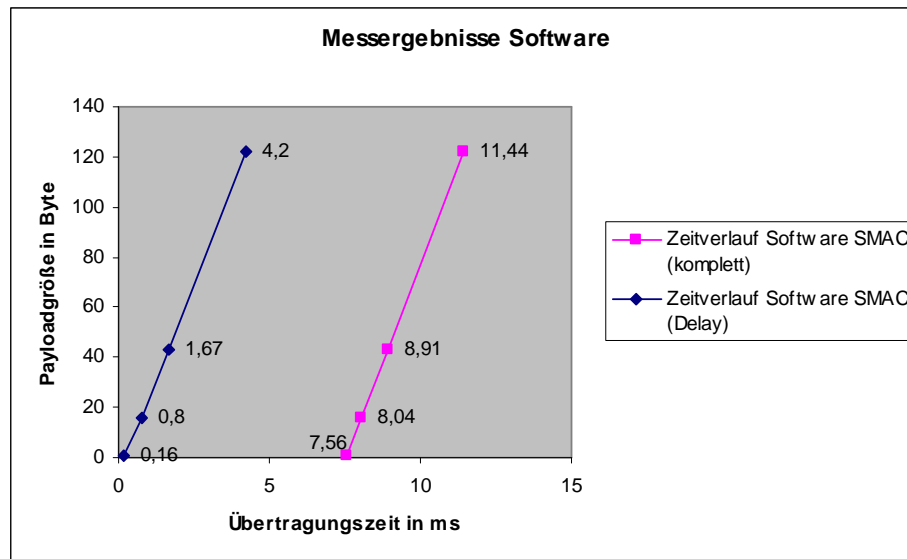


Abbildung 5-6: Messergebnisse Software bei SMAC

Die reine Verzögerung der im Channelblock berechneten Zeit zeigt die blaue Kennlinie, während die rosa Kennlinie zusätzlich die Durchlaufzeiten der Daten durch den Protokollstack beinhaltet.

Vor allem die proforma veranschlagte Zeit, welche ein Paket durch die Schichtenblöcke verbraucht, ist nicht unerheblich, deckt sich aber mit Untersuchungen von senTec bzgl. der Durchlaufzeiten eines Frames durch den Protokollstack.

Der Vergleich der beiden blauen Wertekurven zeigt, dass die in der Spezifikation angegebene Datenrate von 250 kBit/s in der Realität, selbst unter optimalen Bedingungen, nicht tragbar ist. Eher wahrscheinlich ist eine Übertragungsrate von etwa 70 kBit/s – 120 kBit/s, je nach Hardware und Umgebungsprofil.

Um dennoch eine halbwegs genaue Übereinstimmung der beiden Messkurven zu erreichen, muss als erstes die Datenratenfunktion angeglichen werden.

$$\text{Datenrate} = 0,0171875 \cdot (\text{Entfernung} - 80)^2$$

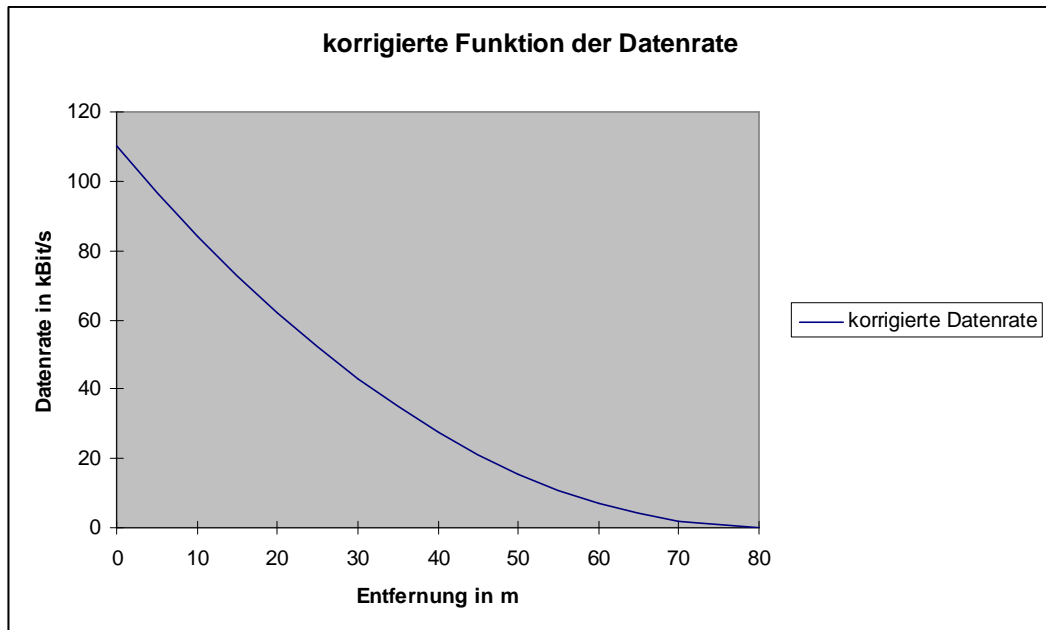


Abbildung 5-7: korrigierte Funktion der Datenrate

Mit der korrigierten Datenratenfunktion nähern sich die Messwerte stark einander an.

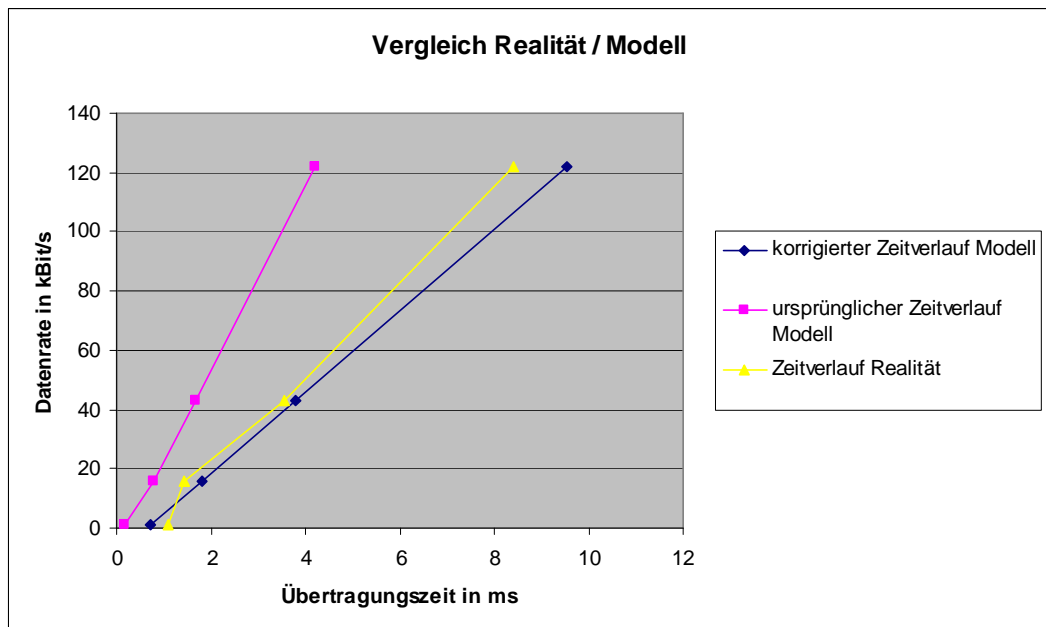


Abbildung 5-8: Vergleich der Hardware- und Softwaremessung mit und ohne Korrektur

Da ZigBee für die reine Datenübertragung ebenfalls das 2,4 GHz ISM-Band benutzt, werden sich die Messergebnisse zwischen Realität und Modell kaum

unterscheiden. Hier wird es eher auf eine Anpassung der Forwardingzeiten durch die Schichten ankommen, da bei ZigBee der Stack einen wesentlich höheren Aufgabenumfang hat.

6 Zusammenfassung und Fazit

Diese Diplomarbeit untersuchte die Realisierungsmöglichkeiten von Funkschnittstellen mit MLDesigner und deren Möglichkeit, sie in Planungsphasen von Netzwerken zu verwenden. Dazu wurden die Verfahren SMAC und IEEE 802.15.4/ZigBee ausgesucht.

Nach gründlicher Recherchearbeit in Bezug auf Funktionsweise und Übertragungsablauf konnte eine auf die reine Datenübertragung bereinigte Funktechnologie umgesetzt werden.

Das entwickelte Modell, welches in Kapitel 4 erläutert ist, wurde mit MLDesigner und deren integrierter Programmiersprache C++ erstellt. Jedes Übertragungsverfahren wurde in eigenen Libraries einsortiert und enthält neben den verwendeten Blöcken auch ein Beispielnetzwerk aus zwei Modulen zur Veranschaulichung von Datenübertragungsvorgängen.

Die anschließende Validierung an einem Echtssystem bestehend aus zwei ZEBRA-Modulen ermöglichte einen Vergleich der Übertragungszeiten zwischen den Messwerten des Softwaremodells und den in der Praxis gemessenen Übertragungszeiten der Hardware.

Als Ergebnis der Analyse stellte sich heraus, dass vor allem im Bereich der in der Spezifikation genannten maximalen Datenübertragungsrate von 250 kBit/s Einbußen bezüglich der Übertragungsleistung entstehen. Zudem ist eine genaue Angabe der Datenübertragungsfunktion sehr schwierig, da allein durch das Umgebungsprofil enorme Schwankungen der verfügbaren Datenrate die Folge sind. Eine vorausgehende Messung am Einsatzgebiet eines solchen Funksystems ist somit zwingend erforderlich. Bei den Hardwaretests entstanden zudem enorme Datenübertragungsunterschiede durch direktes Abschotten der Module, z.B. keine Sichtverbindung, Abdecken der Module oder Ähnliches zur Störung des Sendebetriebs.

Die grundlegende Modellierung der Netzwerkarchitektur samt Datengenerierung, Protokollstack, Datenframes, Channel usw. in MLDesigner ist dennoch möglich und wurde auch umgesetzt. Der modulare Aufbau des Modells veranschaulicht auf leichte Art und Weise den Weg eines Datenpaketes vom Sender zum

Empfänger. Angefangen bei der eigentlichen Datengenerierung im Sender, den Datendurchlauf im Protokollstack des Sendegerätes, die Datenübertragung auf dem physischen Medium und schließlich den Empfang der übertragenen Daten beim Receiver. Jeder generierte Anwendungsdatensatz wird in seiner eigenen Datenstruktur verpackt und mit fortschreitendem Modelldurchlauf um zusätzliche Felder erweitert, so wie es in Realität auch stattfindet.

Die Vereinheitlichung von Interface, Channel und Paketdatenstruktur, sowie Steuerfunktionen des Modells aus dem Umgebungssystem ESL, vereinfachen die Austauschbarkeit der verschiedenen Übertragungstechnologien und ermöglichen so eine leichte Adaption anderer Protokolle wie TCP/IP, Wishbone, RS232 oder Ethernet.

Für aufsetzende oder weiterführende Arbeiten wurde eine gute Grundlage gelegt, die vor allem in Hinsicht auf spätere Architekturprobleme Denkanstöße und Lösungsmöglichkeiten liefert.

Das anfangs sehr einfache Modell ist im Laufe der Zeit gewachsen und hat auch einige Probleme bei der Implementierung des ESL-Target offenbart. So wurden z.B. die ursprünglichen Interfaces zwischen jedem Schichtenblock des Protokollstacks entfernt und lediglich ein einziges Interface pro Partition zur Anbindung an den Übertragungskanal realisiert. Zusätzlich wurde eine Zugriffsteuerung für den Kanal eingebaut.

7 Ausblick

Im ersten Entwicklungsschritt wurde lediglich die Modellierung einer Datenübertragung zwischen zwei Geräten erreicht. Nachfolgende Arbeiten sollten sich deshalb zunächst mit der Realisierung zusätzlicher Netztopologien beschäftigen. Für den Anwender ist die Auswertung von Stern- und Maschennetzen wesentlich wichtiger einzuschätzen als einfaches Peer-to-Peer.

Zusätzlich dazu sollten weitere netzwerktypische Funktionen wie Acknowledgement, Datenkontrolle oder spezielle Sicherheitsaspekte implementiert werden, um den Charakter einer vollständigen Funkdatenübertragung zu gewährleisten.

Als zusätzliche Erweiterung wäre eine Auswertung in Hinsicht auf die Machbarkeit eines Netzwerkes wünschenswert. Dazu müssten die im Performance Monitor mitgeloggten Daten mittels eines Optimierungsverfahrens analysiert werden. Im Ergebnis sollte z.B. die optimale Verteilung der Netzgeräte, die nötige Anzahl der Netzgeräte oder eventuelle Durchsatzengpässe ausgegeben werden.

Da die Arbeit nur statische Netze betrachtet, könnte eine Erweiterung dynamische Netzwerke verwalten, in der Netzteilnehmer sich frei bewegen und die Möglichkeit haben, das Netzwerk zu betreten bzw. zu verlassen.

Für den reellen Einsatz eines Netzwerkplanungssystems in der Industrie müsste eine Möglichkeit geschaffen werden, mittels eines Optimierungsalgorithmus ein, in Bezug auf den Einsatzzweck, bestmögliches oder ökonomisches Netzwerklayout zu generieren. Ein solcher Ansatz zur Netzwerkplanung, in dem jedes Gerät für seine Funktionalität optimal positioniert wird, ist in ferner Zukunft ein Ziel mit echten Marktchancen.

8 Anhang

8.1 Abkürzungsverzeichnis

BDM – Background Debug Modus

CCA – Clear Channel Assessment

CSMA/CA – Carries Sense Multiple Access with Collision Detection

DSP – Digitaler Signalprozessor

ED – Energy Detection

ESL – Electronic System Level

FCS – Frame Check Sequence

FFD – Full Function Device

GUI – Graphical User Interface

ISM – Industrial Science and Medical Band

LQI – Link Quality Indication

MAC – Media Access Control

MLDesigner – Mission Level Designer

NWK – Network

P2P – Peer to Peer

PHY – Physical

QoS – Quality of Service

RAM – Random Access Memory

RFD – Reduced Function Device

SMAC – Simple Media Access Controller

TG – Task Group

UML – Unified Modeling Language

UMTS – Universal Mobile Telecommunications System

WiMAX – Worldwide Interoperability for Microwave Access

WLAN – Wireless Local Area Network

XML – eXtensible Markup Language

ZEBRA – ZigBee Enabled Board for Radio Applications

8.2 Abbildungsverzeichnis

Abbildung 2-1: Aufbau des Beekit

Abbildung 2-2: Data Frame bei SMAC

Abbildung 2-3: IEEE 802.15.4-Schichtenmodell

Abbildung 2-4: Stern- und P2P-Topologie

Abbildung 2-5: Beacon Frame bei IEEE 802.15.4

Abbildung 2-6: Data Frame bei IEEE 802.15.4

Abbildung 2-7: Acknowledgement Frame bei IEEE 802.15.4

Abbildung 2-8: MAC Command Frame bei IEEE 802.15.4

Abbildung 2-9: Maschen-Topologie

Abbildung 3-1: Beispielnetz für Modellierung

Abbildung 3-2: Peer-to-Peer

Abbildung 3-3: Funktion des Pfadverlustes

Abbildung 3-4: Funktion der Datenrate mittels Pfadverlust

Abbildung 3-5: Funktion der Datenrate mittels Messung

Abbildung 4-1: Entwurfspyramide

Abbildung 4-2: Benutzeroberfläche des MLDesigner

Abbildung 4-3: Beispiel Peer-to-Peer-Netz-Modell

Abbildung 4-4: ESL-Target

Abbildung 4-5: Grundaufbau Partition

Abbildung 4-6: Grundaufbau Interface

Abbildung 4-7: Grundaufbau Channel

Abbildung 4-8: Übersicht der Netzgeräte

Abbildung 4-9: modularer Aufbau eines Gerätes

Abbildung 4-10: Netzgerät mit unterschiedlichen Übertragungsstacks

Abbildung 4-11: Netzaufbau mit verschiedenen Übertragungstechniken

Abbildung 4-12: Grundaufbau des Protokollstacks

Abbildung 4-13: Protokollstack des Enddevice

Abbildung 4-14: Protokollstack des Coordinators

Abbildung 4-15: Protokollstack des Senders/Receivers
Abbildung 4-16: DataHandler des IEEE 802.154/ZigBee-Protokollstacks
Abbildung 4-17: NWKHandler des IEEE 802.154/ZigBee-Protokollstacks
Abbildung 4-18: ACLEvent des IEEE 802.154/ZigBee-Protokollstacks
Abbildung 4-19: MACHandler des IEEE 802.154/ZigBee-Protokollstacks
Abbildung 4-20: FrameHandler des MACHandlers bei IEEE 802.154/ZigBee
Abbildung 4-21: FIFOQueue des MACHandlers bei IEEE 802.154/ZigBee
Abbildung 4-22: ChannelHandler des MACHandlers bei IEEE 802.154/ZigBee
Abbildung 4-23: CSMA/CA-Block des MACHandlers bei IEEE 802.154/ZigBee
Abbildung 4-24: CSMA/CA-Ablaufplan bei IEEE 802.154/ZigBee
Abbildung 4-25: DataLogging-Block
Abbildung 4-26: Performance Monitor
Abbildung 4-27: Interface des IEEE 802.154/ZigBee-Protokollstacks
Abbildung 4-28: DataHandler des SMAC-Protokollstacks
Abbildung 4-29: MACHandler des SMAC-Protokollstacks
Abbildung 4-30: CSMA/CA-Ablaufplan bei SMAC
Abbildung 4-31: allgemeiner Aufbau der Frame-Datenstruktur
Abbildung 4-32: Aufbau der AppFrame-Datenstruktur
Abbildung 4-33: Aufbau der NWKFrame-Datenstruktur mit ACLSubset
Abbildung 4-34: Aufbau der NWKFrame-Datenstruktur mit Permission
Abbildung 4-35: allgemeiner Aufbau der MACFrame-Datenstruktur
Abbildung 4-36: allgemeiner Aufbau der ChannelPacket-Datenstruktur
Abbildung 4-37: Channel-Block
Abbildung 4-38: Paketdurchlauf im Channel
Abbildung 4-39: Delay-Block
Abbildung 4-40: modellierte Funktion der Datenrate

Abbildung 5-1: Tektronix TDS 200-Reihe
Abbildung 5-2: Benutzeroberfläche des OpenChoice Desktop
Abbildung 5-3: ZEBRA2400-SM
Abbildung 5-4: Oszilloskopmessbilder 1 Byte, 16 Byte, 43 Byte, 122 Byte (l.o.,
r.o., l.u., r.u.)

Abbildung 5-5: Messergebnisse Hardware bei SMAC

Abbildung 5-6: Messergebnisse Software bei SMAC

Abbildung 5-7: korrigierte Funktion der Datenrate

Abbildung 5-8: Vergleich der Hardware- und Softwaremessung mit und ohne
Korrektur

8.3 Tabellenverzeichnis

Tabelle 2-1: Eigenschaften des Beekit

Tabelle 2-2: Frequenzbänder und Datenraten

Tabelle 3-1: Beispielhafte Messergebnisse

Tabelle 3-2: Beispielhafte Messwerte

Tabelle 4-1: Arten der Modellierungsblöcke bei MLDesigner

Tabelle 4-2: Datenhandling bei MLDesigner

Tabelle 4-3: Arten der Modelldomänen bei MLDesigner

8.4 Literaturverzeichnis

- [ATM04] @t-mix Internet & eCommerce Online Lexicon. *CSMA-CA Info*. 2004. http://www.at-mix.de/csma_ca.htm
- [Bau07] Tommy Baumann. *Integrierte Hard- und Softwareentwicklung mit dem MLDesigner – Entwicklung einer UML-Schnittstelle*. Diplomarbeit, TU Ilmenau, 2005.
- [BHS07] T. Baumann, M. Hauguth, H. Salzwedel. *Overcoming the Gap between Design at Electronic System Level (ESL) and Implementation for Networked Electronics*. 2007
- [BPL06] Torsten Born, Tobias Prüger, Frank Lohse. *Lastverteilungsproblematik in gemischten UMTS/GSM/WLAN-Systemen – Common Radio Ressource Management (CRRM)*. Hauptseminararbeit, TU Ilmenau, 2006.
- [BPS07] Tommy Baumann, Alexander Pacholik, Horst Salzwedel (TU Ilmenau, Fachgebiet System und Steuerungstheorie). *Performance Exploration with MLDesigner using Standardized Communication Interfaces*. 2007. <http://www.tu-ilmenau.de/sst>
- [FS06] Freescale Semiconductor Inc. *Corporate Overview*. 2006. <http://www.freescale.com/webapp/sps/site/overview.jsp?nodeId=060A60>
- [FSB06] Freescale Semiconductor Inc. *Wireless Connectivity BEEKIT™, REV 0*. 2006. www.freescale.com/files/wireless_comm/doc/fact_sheet/BEEKITFS.pdf

- [FSS05] Freescale Semiconductor Inc. *Simple Media Access Controller (SMAC)*, Rev 1.2. 2005. http://www.freescale.com/files/rf_if/doc/user_guide/SMACRM.pdf
- [FSW06] Freescale Semiconductor Inc. *Beekit Wireless Connectivity Toolkit*. 2006. http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=BEEKIT_WIRELESS_CONNECTIVITY_TOOLKIT
- [FSZ06] Freescale Semiconductor Inc. *ZigBee, 802.15.4 MAC, SMAC*. 2005. http://www.vdmais.kiev.ua/pdf_files/ZigBee_seminar.pdf
- [GI05] Gesellschaft für Informatik, Regionalgruppe Ostthüringen. *Mission Level Design als ganzheitlicher Ansatz der Entwicklung von validen Modellen komplexer Systeme*. Prof. Horst Salzwedel, (TU Ilmenau), Referat, Jena, 2005.
- [ICS03] IEEE Computer Society. *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*. 2003.
- [Loh06] Frank Lohse. *Modellierung und Performancebewertung der ZigBee Funkdatenübertragung in MLDesigner*. Studienarbeit, TU Ilmenau, 2006.
- [MLD07] MLDesign Technologies, Inc. 2007. <http://www.mldesigner.com>
- [MLDM06] MLDesign Technologies. *MLDesigner Documentation, Version 2.6*. 2006.
- [MLDG07] Mission Level Design GmbH. *Company*. 2007. <http://www.mldesigner.de/cmsmld//company.html&lan=en>

- [PP07] Department of EECS, UC Berkeley. *The Ptolemy Project*. 2007.
<http://ptolemy.eecs.berkeley.edu/>
- [RE05] Prof. Wolfgang Fengler. *Vorlesung Rechnerentwurf*. 2005.
- [SE05] Dr. Volker Zerbe. *Vorlesung Systementwurf*. 2005.
- [Sen06] senTec Elektronik GmbH. *Sensor System Solutions, Hard- & Software Design*. 2006. <http://www.sentec-elektronik.de>
- [SST03] Prof. Horst Salzwedel. *Vorlesung System- und Steuerungstheorie*. 2003/2004.
- [ST06] senTec Elektronik GmbH. *ZEBRA Starterkit - ZigBee™ Enabled Board for Radio Applications, Preliminary Product Information 1.1*. 2006
- [Sur06] Leonardo Surico (South-West EMEA Field Application Manager at Freescale Semiconductor Inc.). *ZigBee Family, Applications and Roadmap Overview*. 2006.
- [TG406] IEEE 802.15 WPAN™ Task Group 4 (TG4). 2006.
<http://www.ieee802.org/15/pub/TG4.html>
- [UML07] Object Management Group. *UML® Resource Page*. 2007.
<http://www.uml.org/>
- [Ucp06] MicroController Pros Corporation. *Freescale Announces ZigBee Software Stack and Development Kit*. 2006.
<http://www.ucpros.com/Newsletter/newsletter%202006%2005.htm>
- [WI04] Prof. Andreas Mitschele-Thiel. *Vorlesung Wireless Internet*. 2004.

[XML07] SELFHTML e.V. *Einführung in XML*. 2007.
<http://de.selfhtml.org/xml/intro.htm>

[ZNE06] Christoph Mangeng, Daniel Egger, Peter Rinderer, Matthias Vallaster. *Z-Net Zigbee Networking*. Diplomarbeit, 2006.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, diese Diplomarbeit eigenhändig und selbständig verfasst, keine als die angegebenen Hilfsmittel und Quellen verwendet und direkt oder indirekt übernommene Gedanken als solche gekennzeichnet zu haben. Diese Arbeit wurde bisher in dieser oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt noch anderweitig veröffentlicht.

Frank Lohse

Ilmenau, den 18.06.2007

Thesen

- 1) Die maximal verfügbare Datenrate in einem Funknetz ist abhängig von der Geländeumgebung, dem Bewegungsprofil des Gerätes und der verwendeten Hard- und Software der Geräte.
- 2) Die theoretisch mögliche Anzahl von Geräten im ZigBee-Netzwerk ist durch die Speicherlimitation im Coordinator (Geräteliste) und die verfügbare Datenrate des Netzwerkes in der Praxis erheblich kleiner.
- 3) Für eine hinreichend genaue Bestimmung der Datenratenfunktion ist eine Messung im gewünschten Abdeckungsareal unausweichlich.
- 4) Für eine gute Nutzung eines Funknetzes durch Mobile Nodes sollte deren Signalisierungsaufwand nicht mehr als 10% Netzlast erzeugen.
- 5) Mission-Level-Design ermöglicht frühzeitig im Entwurfsprozess Aussagen bezüglich der Leistungsfähigkeit des Systems zu gewinnen und Probleme zu erkennen.
- 6) Es existiert ein eindeutiger Trend drahtlose Netzwerke effektiver nutzen zu können. Dazu werden Simulationsmodelle benötigt, die alle typischen Eigenschaften der vorgesehenen Funktechnologie enthalten und Analyse-möglichkeiten erlauben.
- 7) Mit MLDesigner lassen sich Netzwerkstrukturen nachbilden.
- 8) Die einheitlichen Architekturkomponenten Interface, Channel und die Paketdatenstruktur, sowie das übergeordnete ESL-Target ermöglichen eine leichte Adaption anderer Protokolle und deren freie Austauschbarkeit.

- 9) Das ausgearbeitete Modell veranschaulicht die einfache Datenübertragung per SMAC oder IEEE 802.15.4/ZigBee.

- 10) Mit dem Modell können Aussagen bzgl. des Ressourcenverbrauchs und der Performance bei bestimmten Datenaufkommen getroffen werden.