Fachhochschule Düsseldorf

Fachbereich 3 - Elektrotechnik

Bachelorthesis

Optimierung der drahtlosen Kommunikation zwischen Komponenten mit Contiki-Betriebssystem in Hinblick auf Datendurchsatz, Energieeffizienz und Sicherheit am Beispiel des WieDAS-Projektes

Improvement of the wireless communication between components running with the Contiki operating system in terms of bandwith, energy efficiency and security based on the example of the WieDAS project

Vorgelegt von:	Mirco Kern
Matrikelnummer:	525698
Vorgelegt am:	11.04.2012
1. Prüfer:	Herr Prof. Dr.-Ing. Ulrich Schaarschmidt
2. Prüfer	Herr Prof. Dr. rer. nat. Wolfgang Lux

Kurzfassung

Dieses Bachelorprojekt wurde im Labor für Informatik an der Fachhochschule Düsseldorf durchgeführt und steht im Zusammenhang mit einem Forschungsprojekt zu dem Themengebiet Ambient Assisted Living. Hauptziele des Bachelorprojektes waren die Optimierung der Kommunikation über das 6LoWPAN-Protokoll hinsichtlich der Aspekte Datendurchsatz, Energieeffizienz und der Sicherheit der Datenübertragung. Um diese Ziele zu erreichen, wurden innerhalb des Projektes ein Kommunikationsmodul und eine entsprechenden Entwicklungsplatine realisiert. Zudem wurden softwaretechnische Funktionen entwickelt, die der Anpassung des Mikrocontroller-Betriebssystems Contiki 2.5 für die Nutzung der Stromsparfunktionen des verwendeten Mikrocontrollers ATmega128RFA1 und der Verwendung des Verschlüsselungsmoduls dienen. Durch die am Ende des Projektes durchgeführten Gerätetests konnte die Erreichung der wesentlichen Projektziele verifiziert werden. Innerhalb dieser Thesis wird dem Leser neben dem Grundlagenwissen zu dem verwendeten Mikrocontroller die Vorgehensweise bei der Erstellung der Hard- und Software näher gebracht werden.

Abstract

This bachelor project was realized in the laboratory of computer science at the Fachhochschule Düsseldorf and is associated with a research project on the topic of Ambient Assisted Living. The main goals of the bachelor project has been the improvement of the communication over the 6LoWPAN protocol in terms of bandwith issues, energy efficiency and the security of the data transmission. To achieve these goals within the project, a communication module with a suitable development board has been realized. In addition, software-related functions have been developed that are used to adapt the microcontroller operating system Contiki 2.5 for the use of the energy-saving features of the microcontroller ATmega128RFA1 and the use of its encryption module. At the end of the project device testings has been realized to verify the attainment of the project´s main goals. Within this thesis, the reader will get the basic knowledge of the used microcontroller and the procedure of creating the hardware and software.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

6LoWPAN IPv6 over Low Power Wireless Personal Area Networks
A/D ..... Analog/Digital
AES ..... Advanced Encryption Standard
Balun ..... Balanced/Unbalanced
CBC ..... Cipher Block Chaining
CCA ..... Clear Channel Assessment
ECB ..... Electronic Code Book
ISP ..... In-System Programming
JTAG ..... Joint Test Action Group
PCB ..... Printed Circuit Board
PLL ..... Phase-Locked Loop
QFN ..... Quad Flat No Leads
RPL ..... Routing Protocol for Low power and Lossy Networks
TWI ..... Two Wire Interface
USART ... Universal Synchronous Asynchronous Receiver Transmitter
USB ..... Universal Serial Bus
WieDAS .. Wiesbaden-Düsseldorfer Ambient Assisted Living Service Platform

1 Einleitung

1.1 Projektumfeld

Dieses, im Labor für Informatik an der Fachhochschule Düsseldorf durchgeführte Bachelorprojekt und die ausgearbeitete Bachelorthesis stehen im Zusammenhang mit dem WieDAS[1]-Projekt, das in Kooperation mit der Hochschule Rhein-Main und Vertretern aus Industrie und Wirtschaft durchgeführt und durch das Bundesministerium für Bildung und Forschung gefördert wird. Innerhalb des WieDAS-Projektes werden Grundlagen zu dem relativ jungen Themengebiet Ambient Assisted Living (AAL) erarbeitet, welches sich mit technischen Systemen beschäftigt, die es älteren oder in irgendeiner Weise körperlich eingeschränkten Personen erlauben, möglichst lange in ihrer gewohnten Umgebung zu verbleiben und ein eigenständiges Leben zu führen. Von verschiedenen Mitarbeitern des Labors für Informatik an der Fachhochschule Düsseldorf werden hierzu Soft- und Hardwaresysteme entwickelt, die sowohl medizinische Bereiche, als auch Lösungen, die in das Themengebiet der Hausautomatisierung fallen, abdecken. Neben der Funktion an sich sind bei der Entwicklung Offenheit, Erweiterbarkeit, Sicherheit, Selbstorganisation und Mobilität als wichtige Aspekte zu berücksichtigen [4]. Da die Akzeptanz der Nutzer für ein Gerät sinkt, sobald es ihn (nach außen) als hilfebedürftig erscheinen lässt, sollen sich die einzelnen Komponenten zudem möglichst unauffällig in die gewohnte Umgebung integrieren lassen, beziehungsweise von Außenstehenden nicht als Komponente eines Assistenzsystems erkannt werden.

[1] Wiesbaden-Düsseldorfer Ambient Assisted Living Service Platform

1.2 Aufgabenstellung

Aufbauend auf dem ebenfalls im Labor für Informatik an der Fachhochschule Düsseldorf durchgeführten Praxisprojekt, das die Untersuchung und grundlegende Realisierung der Datenübertragung unter Verwendung des Mikrocontrollers ATmega128RFA1 des Herstellers Atmel über das 6LoWPAN[2]-Protokoll zum Ziel hatte, sollen innerhalb dieses Bachelorprojektes Optimierungen zu der Datenkommunikation erarbeitet werden. Hierzu werden zunächst ein neues Funkmodul, das auf der Nutzung des Mikrocontrollers ATmega128RFA1 basiert und eine passende Versuchsplatine, auf die das Funkmodul aufgesteckt werden kann, realisiert. Weitere Ziele dieses Bachelorprojektes sind softwaretechnische Implementierungen, die der Verbesserung der drahtlosen Kommunikation zwischen den verschiedenen Komponenten des WieDAS-Projektes und der Optimierung des Stromverbrauchs der Sensoren dienen.

Die Hauptziele des Bachelorprojektes können wie folgt zusammengefasst werden:

- Design und Aufbau eines neuen Funkmoduls
- Implementierung von Stromsparfunktionen
- Implementierung der Datenverschlüsselung
- Reduzierung der Paketverluste bei der drahtlosen Datenübertragung
- Steigerung der Datenübertragungsrate
- Durchführung von Gerätetests und Vergleich mit der bisher genutzten Hardware

Eine genaue Beschreibung der Anforderungen zu den genannten Punkten folgt in *Kapitel 3* und in *Kapitel 4*, in denen das Hardwaredesign und die Implementierung der Software dargestellt werden.

[2]IPv6 over Low Power Wireless Personal Area Networks

2 Der AVR ATmega128RFA1 Mikrocontroller

Da eine Vorgabe für dieses Bachelorprojekt die Nutzung des Mikrocontrollers ATmega128RFA1 des Herstellers Atmel ist, kann auf einen Vergleich verschiedener 6LoWPAN-fähiger Mikrocontroller verzichtet werden. Daher wird an dieser Stelle lediglich auf die für den AAL-Kontext wichtigen Eigenschaften des Mikrocontrollers eingegangen. Dies betreffen die Verschlüsselungseinheit und die nutzbaren Stromsparfunktionen des ATmega128RFA1. Weitere Einzelheiten können der Beschreibung des Mikrocontrollers [1] entnommen werden, die auf der Internetseite des Herstellers Atmel zu finden ist.

2.1 Die Funkschnittstelle des ATmega128RFA1

Eine Besonderheit des ATmega128RFA1 Mikrocontrollers ist die in dem Mikrochip integrierte Funkschnittstelle (System on a Chip). Die zuvor genutzte Hardware-Lösung basierte auf dem Mikrocontroller ATmega1284P und der externen Transceiver-Einheit AT86RF231, die wie der ATmega128RFA1 von dem Hersteller Atmel stammen. Der große Vorteil des integrierten Funkmoduls ist die direkte Steuerung der Transceiver-Einheit und die Übertragung der Daten zwischen Mikrocontroller und Transceiver-Modul über die Register des Controllers, die theoretisch die Möglichkeit einer schnelleren Datenübertragung in Verbindung mit einer höheren Energieeffizienz bietet. Außerdem wird durch die Integration des Transceivers der Platzbedarf der Schaltung auf der Platine reduziert, was neben dem geringeren Stromverbrauch besonders bei platzsparend zu haltenden Komponenten, wie zum Beispiel dem innerhalb des WieDAS-Projektes entwickelten Notfallknopf als wünschenswert erachtet wird. Auf Seite 4 werden die grundlegenden Charakteristika der Transceiver-Einheit des ATmega128RFA-Mikrocontrollers zusammengefasst [1].

2.1.1 Charakteristika der Funkschnittstelle

- Frequenzbereich: 2,5 GHz (ISM-Band)
- Nutzbare Kanäle: 11 bis 26 (2405 MHz bis 2480 MHz)
- Empfindlichkeit des Receivers: -100 dBm
- Ausgangsleitung: -17 dBm bis +3,5 dBm (entspricht 0,02 mW bis 2,24 mW)
- Datenraten: 250 kbit/s, 500 kbit/s, 1 Mbit/s und 2 Mbit/s
- Stromverbrauch (bei einer Betriebsspannung von 3,0 V)
  - Leerlauf (TRX_OFF): 0,4 mA
  - Datenempfang (RX_ON): 12,5 mA
  - Datenversand (BUSY_TX): 14,5 mA (bei maximaler Ausgangsleistung)
- Ausgang: 100 Ω, symmetrisch
- Größe des Datenpuffers: 128 Byte

2.1.2 Betriebsmodi des Funkmoduls

Das Funkmodul des ATmega128RF1 verfügt über verschiedene Betriebsmodi, die entweder über die Bits TRX_CMD des Registers TRX_STATE oder über die zwei Bits SLPTR und TRXRST des TRXPR-Registers gesteuert werden können. Da auf die Register und die Steuerung des Funkmoduls innerhalb *Kapitel 2.1.3* eingegangen wird, soll an dieser Stelle lediglich eine Übersicht über die verschiedenen Betriebszustände gegeben werden. Die Übersicht ist auf die möglichen Zustände des Basic Operating Mode reduziert, da nur dieser innerhalb dieses Bachelorprojektes genutzt wurde. Der Vollständigkeit halber sei erwähnt, dass der Extended Operation Mode prinzipiell die gleichen Zustände wie der Basic Operation Mode enthält, aber zusätzlich um Funktionen für das automatische Bestätigen eines fehlerfreien Datenpakets (RX_AACK) und die automatische Wiederholung nach einer fehlerhaften Datenübertragung (TX_ARET) erweitert wurde.

2.1.2.1 Zustandsdiagramm der Transceiver-Einheit

Für die Verdeutlichung der möglichen Zustandsfolgen der Transceiver-Einheit sind die erreichbaren Zustände des Basic Operating Modes in *Abbildung 2.1* grafisch illustriert. Für

die Gewährleistung einer besseren Übersicht wurden die Kommandos, die zu einem neuen Zustand führen bewusst nicht dargestellt. Diese können den Registerbeschreibungen in *Kapitel 2.1.3* entnommen werden.

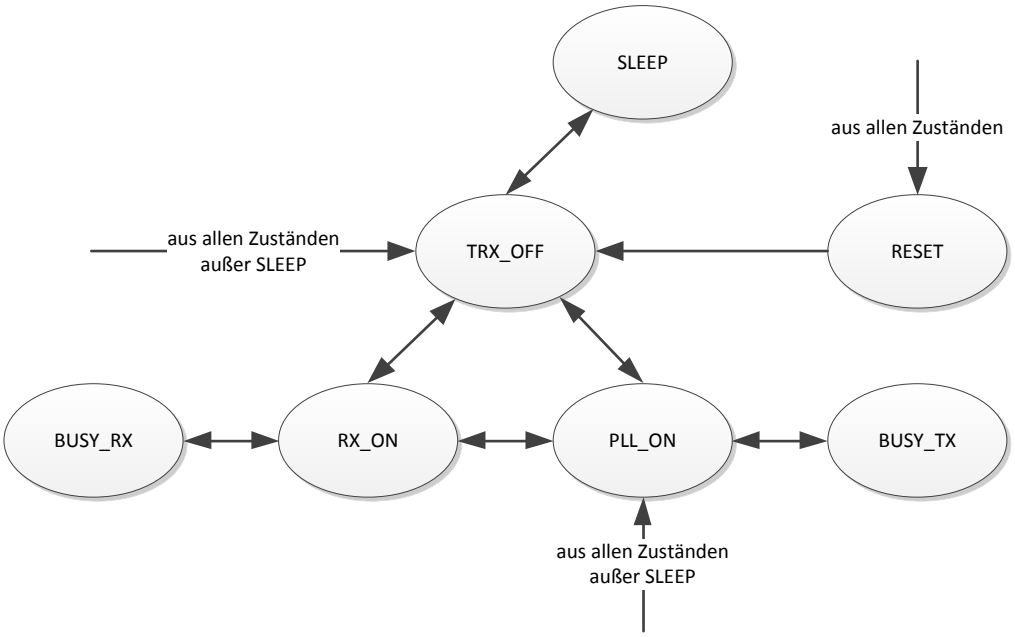

Abb. 2.1: Mögliche Zustandsfolgen des Transceivers (vgl. [1])

2.1.2.2 Der Schlafmodus (SLEEP)

Im Schlafmodus ist der Transceiver komplett ausgeschaltet und der Stromverbrauch der Transceiver-Einheit reduziert sich auf die auftretenden Leckströme. Außer dem Beenden des Schlafmodus und dem Auslösen eines Transceiver-Resets sind keine weiteren Registerzugriffe möglich.

2.1.2.3 Leerlauf (TRX_OFF)

Wird der Transceiver im Leerlauf betrieben, sind der Schwingquarz und der Spannungsregler des Transceiver-Moduls aktiv und es ist möglich auf die Transceiver-Register, den Framebuffer und das AES[1]-Verschlüsselungs-Modul zuzugreifen.

[1] Advanced Encryption Standard

2.1.2.4 Aktivierung des PLL Frequenz-Synthesizers (PLL_ON)

Wird aus dem Zustand TRX_OFF in den Zustand PLL_ON übergegangen, werden der analoge Spannungsregler (Analog Voltage Regulator, AVREG) und der PLL[2] Frequenz-Synthesizer aktiviert. Nach Aktivierung des PLL Frequenz-Synthesizers, wird dieser automatisch auf die Empfangsfrequenz eingestellt, welche dem in den entsprechenden Registern festgelegten Übertragungskanal entspricht. Die erfolgreiche Regelung auf die Empfangsfrequenz wird über den TRX24_PLL_LOCK-Interrupt signalisiert, worauf der Empfang oder Versand von Daten initiiert werden kann.

2.1.2.5 Empfangsmodus (RX_ON und BUSY_RX)

Der Empfangsmodus des ATmega128RFA1 ist intern in die Zustände RX_ON und BUSY_RX unterteilt, in denen der Receiver blockiert und der PLL Frequenz-Synthesizer aktiviert wird.

Im Status RX_ON wartet der Transceiver auf eingehende Datenframes. Nach der Erkennung eines gültigen Synchronisierungs-Headers (SHR), geht der Transceiver automatisch in den Zustand BUSY_RX über. Durch den Empfang eines kompletten Headers der Netzzugangsschicht (PHY Header, PHR) wird der TRX24_RX_START-Interrupt ausgelöst, der den Beginn des Empfangs eines Datenpakets signalisiert. Nun empfängt der Transceiver den Nutzdatenanteil des eingehenden Pakets, demoduliert diesen und speichert die Daten in den Datenpuffer der Transceiver-Einheit (Framebuffer). Der Empfang eines kompletten Datenframes wird durch die Auslösung des Interrupts TRX24_RX_END signalisiert, worauf der Transceiver auf den Status RX_ON zurück gesetzt wird. Der RX_ON-Zustand wird verlassen, indem über die Bits TRX_CMD des Registers TRX_STATE eine Zustandsänderung veranlasst wird.

Der Übergang zu dem Zustand RX_ON ist aus den Zuständen TRX_OFF und PLL_ON möglich. Allerdings ist zu beachten, dass es für den Empfang von Daten in jedem Fall notwendig ist, den PLL Frequenz-Synthesizer auf die entsprechende Empfangsfrequenz zu regeln. Ansonsten kann zwar in den Zustand RX_ON übergegangen werden, aber ein Empfang von Daten ist nicht möglich. Daher ist bei der Programmierung von Funktionen zum Datenempfang darauf zu achten, dass nicht nur eine Überprüfung des Transceiver-Zustands auf den Übergang zu dem Modus RX_ON abgefragt wird, sondern diese Abfrage mit dem Warten auf den TRX24_PLL_LOCK-Interrupt verbunden werden sollte.

[2] Phase-Locked Loop

2.1.2.6 Sendemodus (BUSY_TX)

Die Übertragung von Daten kann lediglich aus dem Zustand PLL_ON heraus initiiert werden. Geht der Transceiver in den Zustand BUSY_TX über, wird über den PLL Frequenz-Synthesizer die PLL Frequenz auf die entsprechende Sendefrequenz geregelt und der Ausgangsverstärker aktiviert. Nach der Übertragung des Synchronisations-Headers (SHR), wird der Inhalt des Datenpuffers gesendet. Ist die Übertragung abgeschlossen, wird der Ausgangsverstärker automatisch abgeschaltet und der Transceiver in den Zustand PLL_ON zurückgesetzt.

2.1.3 Die Register des Funkmoduls

2.1.3.1 TRXPR (Transceiver Pin Register)

Über das Transceiver Pin Register wird der Zugriff auf die grundlegenden Funktionen des Transceivers gesteuert. Hierzu gehören die Änderung des Transceiver-Zustands (State Transition) und das Rücksetzen des Transceivers (Transceiver Reset). Die Belegung der einzelnen Bits kann folgender *Tabelle 2.1* entnommen werden.

Bit-Nr.	Bezeichnung	Beschreibung
0	TRXRST	Transceiver Reset
1	SLPTR	Kontrolle des Transceiver-Zustands
2	Resx2	Reserviert
3	Resx3	Reserviert
4	Res0	Reserviert
5	Res1	Reserviert
6	Res2	Reserviert
7	Res3	Reserviert

Tab. 2.1: Das Transceiver Pin Register TRXPR (vgl. [1])

Bit 0 - TRXRST Über das Setzen des Bits TRXRST wird der Transceiver zurückgesetzt. Nach Rücksetzung befindet sich die Transceiver-Einheit in dem Zustand TRX_OFF.

Bit 1 - SLPTR Das multifunktionale Bit SLPTR dient der Kontrolle der Transceiver-Zustände.

TRX-Status	Funktion	SLPTR	Beschreibung
PLL_ON	TX start	0 -> 1	Startet die Übertragung eines Frames
TRX_OFF	Sleep	0 -> 1	Versetzt die Transceiver-Einheit in den Schlafmodus
SLEEP	Wakeup	1 -> 0	Versetzt die Transceiver-Einheit in den Status TRX_OFF

Tab. 2.2: Nutzung des SLPTR-Konfigurationsbits (vgl. [1])

Die Nutzung des SLPTR-Bits und die damit verbundenen Zustandsänderungen sind in *Tabelle 2.2* dargestellt. Zu beachten ist, dass sich die Funktion des Bits je nach aktuellem Zustand der Transceiver-Einheit unterscheidet.

2.1.3.2 TRX_STATE (Transceiver State Register)

Neben der Nutzung des SLPTR-Bits des Transceiver Pin Registers werden die ersten 5 Bits des Registers TRX_STATE dazu genutzt das Subregister TRX_CMD anzusprechen und Zustandsänderungen der Transceiver-Einheit herbei zu führen. *Tabelle 2.3* zeigt die Zuweisung der Subregisterwerte zu den entsprechenden Zuständen. Wird ein ungültiger Wert in das Register geschrieben, findet keine Zustandsänderung statt.

Wert	Zustand	Beschreibung
0x00	NOP	Keine Zustandsänderung
0x02	TX_START	Startet die Übertragung eines Frames
0x03	FORCE_TRX_OFF	Erzwingt den Zustand TRX_OFF
0x04	FORCE_PLL_ON	Erwzingt den Zustand PLL_ON
0x06	RX_ON	Zuständsänderung zu RX_ON
0x08	TRX_OFF	Zuständsänderung zu TRX_OFF
0x09	PLL_ON	Zuständsänderung zu PLL_ON
0x16	RX_ACK_ON	Zuständsänderung zu RX_AACK RX (RX mit Acknowledge)
0x19	TX_ARET_ON	Zuständsänderung zu TX_ARET (TX mit automatischem Retry)

Tab. 2.3: Das Subregister TRX_CMD (vgl. [1])

2.1.3.3 TRX_STATUS (Transceiver Status Register)

Über das Register TRX_STATUS werden der aktuelle Zustand der Transceiver-Einheit und Informationen zum Status von CCA[3] Operationen ausgelesen. *Tabelle 2.4* gibt einen Überblick über die dem TRX_STATUS zugehörigen Registerbits.

Bit-Nr.	Bezeichnung	Beschreibung
0	TRX_STATUS0	Anzeige des aktuellen Transceiver-Status
1	TRX_STATUS1	Anzeige des aktuellen Transceiver-Status
2	TRX_STATUS2	Anzeige des aktuellen Transceiver-Status
3	TRX_STATUS3	Anzeige des aktuellen Transceiver-Status
4	TRX_STATUS4	Anzeige des aktuellen Transceiver-Status
5	TST_STATUS	Interne Anzeige des TST-Modus (Testmodus)
6	CCA_STATUS	Zeigt Ergebnis der CCA-Messung an
7	CCA_DONE	Zeigt an, dass die CCA-Messung beendet wurde

Tab. 2.4: Das Transceiver Pin Register TRXPR (vgl. [1])

Da im Normalfall nur die Registerbits TRX_STATUS0 bis TRX_STATUS4 genutzt werden, wird in *Tabelle 2.5* lediglich auf die Bedeutung der Werte dieser Bits eingegangen. Zudem wurden für eine bessere Übersicht die Werte für die erweiterten Transceiver-Modi nicht in die Tabelle eingefügt.

Wert	Zustand	Beschreibung
0x01	BUSY_RX	Daten werden empfangen
0x02	BUSY_TX	Daten werden versendet
0x06	RX_ON	Transceiver wartet auf eingehende Daten
0x08	TRX_OFF	Transceiver befindet sich im Zustand TRX_OFF
0x09	PLL_ON	PLL Frequenz-Synthesizer ist aktiviert
0x0F	SLEEP	Transceiver befindet sich im SLEEP-Modus
0x1F	STATE_TRANSITION	Es findet gerade ein Wechsel des Zustands statt

Tab. 2.5: Die Registerbits TRX_STATUS0 bis TRX_STATUS4 (vgl. [1])

[3]Clear Channel Assessment

2.2 Das Verschlüsselungsmodul des ATmega128RFA1

Da die innerhalb des WieDAS-Projektes genutzten Komponenten teilweise medizinische Daten übertragen oder für die Steuerung von sicherheitsrelevanten Objekten, wie Türen und Fenster genutzt werden, ist die Gewährleistung einer möglichst hohen Sicherheit der Datenübertragung unerlässlich. Hierzu wird innerhalb dieses Bachelorprojektes zur Sicherstellung der Vertraulichkeit der Daten das integrierte Verschlüsselungsmodul des ATmega128RFA1 genutzt. Dieses bietet die Möglichkeit der direkten Hardwareverschlüsselung nach dem Algorithmus AES-128. Die Eigenschaften des Verschlüsselungsmoduls können wie folgt zusammengefasst werden: [1]

- Hardwarebeschleunigung für Verschlüsselung und Entschlüsselung
- Volle Kompatibilität zu dem AES-128 Verschlüsselungsstandard
- Unterstützung der Modi ECB[4] (verschlüsseln und entschlüsseln) und CBC[5] (verschlüsseln)
- Von anderen Controller-Einheiten unabhängiger Betrieb
- Nutzung der 16 MHz Oszillators der Transceiver-Einheit
- Verschlüsselungsdauer: 24 µs/16 Byte

2.2.1 Der AES-Verschlüsselungsalgorithmus

Der Advanced Encryption Standard (AES) ist ein symmetrisches Verschlüsselungsverfahren, dessen Algorithmus frei verfügbar ist und ohne Lizenzgebühren eingesetzt und in Hard- und Software implementiert werden kann. Der Algorithmus arbeitet nach dem Prinzip der Blockverschlüsselung, bei dem der Klartext in eine Folge gleich langer Datenblöcke (16 Byte) zerlegt wird und die einzelnen Blöcke unabhängig voneinander mit dem gleichen Schlüssel chiffriert werden. Dies bedeutet im Umkehrschluss, dass auch die erzeugten Chiffreblöcke eine feste Länge haben. Der Chiffretext wird erzeugt, indem die Chiffreblöcke wieder zusammengefügt werden.

Vor der Etablierung des AES-Verschlüsselungsalgorithmus war der Data Encryption Standard (DES) der am häufigsten genutzte Algorithmus für die symmetrische Verschlüsselung von Daten. Allerdings hatte dieser die Schwäche, dass er lediglich eine Schlüssellänge von 56 Bit besaß und deshalb ab Mitte der 90er Jahre keine ausreichende Sicherheit gegen Brute-Force-Angriffe bot. Daher wurde durch das amerikanischen

[4] Electronic Code Book
[5] Cipher Block Chaining

Handelsministerium unter Mitwirkung des National Institute of Standards and Technologie (NIST)[6] eine Ausschreibung für die Suche nach einem Nachfolgealgorithmus organisiert und der von den Entwicklern Joan Daemen und Vincent Rijmen vorgsetellte AES-Algorithmus als Sieger gekürt. Trotz der Veröffentlichung einiger Angriffsmöglichkeiten auf den AES-Verschlüsselungsalgorithmus gilt dieser bis heute als so sicher, dass der Advanced Encryption Standard in den Vereinigten Staaten von Amerika für die Verschlüsselung von Dokumenten höchster Geheimhaltungsstufe zugelassen ist. [5]

2.2.1.1 Arbeitsweise des AES-Verschlüsselungsalgorithmus

Bei der AES-Verschlüsselung werden die zu verschlüsselnden Daten zunächst in feste Blöcke mit einer Größe von 16 Byte aufgeteilt. Jeder Block wird in eine zweidimensionale Tabelle mit vier Zeilen und vier Spalten geschrieben, dessen einzelne Zellen je 1 Byte groß sind. Die einzelnen Blöcke werden nacheinander bestimmten Transformationen unterzogen. Hierbei geschieht die Verschlüsselung der Blöcke jedoch nicht in einem Schritt unter Verwendung des originalen Chiffrierschlüssels, sondern es werden in verschiedenen Verschlüsselungsrunden Teile des zuvor erweiterten Schlüssels nacheinander auf den Klartext-Block angewendet. Die Anzahl der Verschlüsselungsrunden hängt von der Länge des Schlüssels ab, wobei bei einer Verschlüsselung mit der von dem Mikrocontroller genutzten Schlüssellänge von 128 Bit zehn Verschlüsselungsrunden durchlaufen werden. Der Ablauf der AES-Verschlüsselung und dessen einzelne Funktionen sind auf den diesem Thema zugeordneten Seiten des Internetauftritts der Carl von Ossietzky Universität in Oldenburg[3] sehr anschaulich dargestellt.

2.2.1.2 Die Substitutionsbox (S-Box)

Mit Hilfe der Substitutionsbox wird angegeben, auf welche Weise in jeder Verschlüsselungsrunde die Werte eines Verschlüsselungsblocks durch einen anderen Wert zu ersetzen ist. Sie bildet somit die Basis für alle monoalphabetischen[7] Verschlüsselungsverfahren, zu denen auch die AES-Verschlüsselung zählt. Die einzelnen Werte der Substitutionsbox können zur Verringerung des Speicherbedarfs entweder dynamisch berechnet werden oder als vorberechnete Werte in einem mehrdimensionalen Array abgespeichert werden. Letzteres Verfahren wird durch den ATmega128RFA1 angewendet und führt zu einer performanteren Verschlüsselung. *Tabelle 2.6* auf Seite 12 zeigt den Aufbau der für die Verschlüsselung nach AES genutzte, konstante Substitutionsbox [3].

[6] Amerikanische Standardisierungsbehörde
[7] Es wird nur ein (festes) Alphabet für die Kryptografie verwendet.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Tab. 2.6: Die AES-Substitutionsbox (vgl. [3])

Die Verwendung der Substitutionsbox wird auf Seite 14 innerhalb der Beschreibung der SubBytes()-Funktion an einem Beispiel erläutert.

2.2.1.3 Ablauf der Verschlüsselung

1. Schlüsselexpansion
2. Vorrunde: AddRoundKey(Rundenschlüssel[0])
3. Verschlüsselungsrunden (n = 1 bis n < 10)
   a) SubBytes()
   b) ShiftRows()
   c) MixColumns()
   d) AddRoundKey(Rundenschlüssel[n])
4. Schlussrunde
   a) SubBytes()
   b) ShiftRows()
   c) AddRoundKey(Rundenschlüssel[10])

Schlüsselexpansion

Bei der Schlüsselexpansion wird der benötigte Chiffrierschlüssel bei einer Schlüssellänge von 128 Bit in elf Rundenschlüssel (Round Keys) aufgeteilt. Da die einzelnen Rundenschlüssel die gleiche Größe wie der zu verschlüsselnde Block aufweisen müssen, wird der Chiffrierschlüssel am Beginn der Ver- und Entschlüsselung zunächst auf die Länge von $(10 + 1) \cdot 16\,Byte = 176\,Byte$ expandiert. Hierzu wird eine Tabelle mit vier Zeilen und 44 Spalten erstellt, wovon jeweils vier Spalten einen Rundenschlüssel aufnehmen. Zu Beginn der Schlüsselexpansion werden die ersten vier Spalten mit dem originalen Chiffrierschlüssel gefüllt, der in der Vorrunde für die erste Schlüsseladdition (AddRoundKey(Rundenschlüssel[0])) genutzt wird.

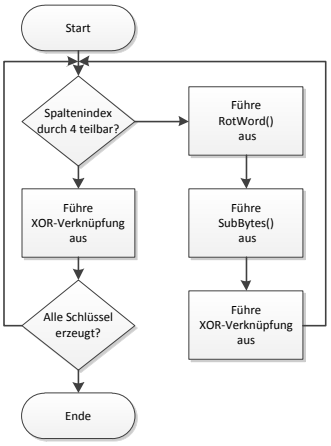

Abb. 2.2: AES-Schlüsselexpansion

Für jede weitere Spalte der Tabelle gilt:

Falls $i \ mod \ 4 = 0$

$$w_i = w_{i-4} \ XOR \ SubBytes(RotWord(w_{i-1})) \ XOR \ Rcon[i/4]$$

sonst

$$w_i = w_{i-4} \ XOR \ w_{i-1}$$

Die auf Seite 13 zu findende *Abbildung 2.2* verdeutlicht das Prinzip der Schlüsselexpansion des AES-Algorithmus anhand eines Flussdiagramms.

RotWord() ist eine Rotationsfunktion, über die das erste Byte der Spalte an die letzte Stelle gesetzt wird und die anderen Bytes eine Zeile oberhalb der alten Position abgelegt werden.

Die Funktion **SubBytes()** wendet die Substitutionsbox der Verschlüsselung auf einzelne Bytes an.

Rcon stellt eine Rundenkonstante dar, die mit dem ersten Byte jeder vierten Spalte XOR verknüpft wird.

AddRoundKey()

AddRoundKey() wird in der Vorrunde und am Ende jeder Verschlüsselungsrunde ausgeführt und bildet eine XOR-Verknüpfung zwischen dem aktuellen Verschlüsselungsblock und dem entsprechenden Rundenschlüssel. Innerhalb des AES-Verschlüsselungsstandards ist AddRoundKey() die einzige Funktion, die den Algorithmus von dem Chiffrierschlüssel abhängig macht.

SubBytes()

Über die Funktion SubBytes() werden die einzelnen Bytes über die in der Substitutionsbox der AES-Verschlüsselung angegebenen Substitutionen ersetzt. Hierzu werden die einzelnen Bytes in zwei Nibble (4-Bit-Wörter) zerlegt, das entsprechende Äquivalent in der Substitutionsbox gesucht und wieder auf ein Byte abgebildet. Soll beispielsweise der hexadezimale Wert $(5b)_{16}$ substituiert werden, wird das Byte in die Nibbles mit den Werten 5 und 8 gesplittet, der diesen Nibbles entsprechende Substitutionswert $(39)_{16}$ aus der Substitutionsbox gelesen und das Byte hiermit ersetzt.

ShiftRows()

Die Funktion ShiftRows() sorgt dafür, dass die Bytes in den Zeilen des Verschlüsselungsblocks um eine von der Schlüssellänge abhängige Anzahl zyklisch nach links verschoben werden. Für eine Schlüssellänge von 128 Bit gilt folgender Zusammenhang:

Zeile l_i mit $0 \leq i \leq 3$ $\Rightarrow$Verschiebung um i Positionen zyklisch nach links

MixColumns()

Die Funktion MixColumns() dient der Vermischung der Daten innerhalb der einzelnen Spalten, die durch die Multiplikation der Zeilen mit einer konstanten Matrix erreicht wird. Die Multiplikation findet jedoch nicht im natürlichen Zahlenraum statt, sondern in einem so genannten endlichem Körper (Galoiskörper), in dem die Grundrechenarten der Multiplikation und Addition durch Modulo-Operationen abgebildet werden. Da an dieser Stelle die dahinter stehende mathematische Theorie zu weit führen würde, wird hier anhand eines Beispiels die Berechnung der neuen Zeilenwerte verdeutlicht. [3]

Ursprüngliche Elemente der Spalte j: a_{oj}, a_{1j}, a_{2j}, a_{3j}

Veränderte Elemente der Spalte j: b_{oj}, b_{1j}, b_{2j}, b_{3j}

mit

$$\begin{pmatrix} b_{0j} \\ b_{1j} \\ b_{2j} \\ b_{3j} \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} a_{0j} \\ a_{1j} \\ a_{2j} \\ a_{3j} \end{pmatrix}$$

Für die Berechnung der Produkte wird das ursprüngliche Byte zunächst als Polynom dargestellt:

$$(d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0)_2 = d_7 \cdot x^7 + d_6 \cdot x^6 + d_5 \cdot x^5 + d_4 \cdot x^4 + d_3 \cdot x^3 + d_2 \cdot x^2 + d_1 \cdot x + d_0$$

Die so gebildeten Polynome werden im nächsten Schritt modulo $(x^8 + x^4 + x^3 + x + 1)$ miteinander multipliziert, wobei die Additionen innerhalb der Matrizenmultiplikation als bitweise XOR-Verknüpfung definiert sind.

In diesem Beispiel sei

$$\begin{pmatrix} a_{0j} \\ a_{1j} \\ a_{2j} \\ a_{3j} \end{pmatrix} = \begin{pmatrix} d4 \\ 32 \\ f4 \\ ae \end{pmatrix}$$

Es gilt

$b_{0j} = 2 \cdot a_{0j} + 3 \cdot a_{1j} + 1 \cdot a_{2j} + 1 \cdot a_{3j} = 2 \cdot d4 + 3 \cdot 32 + 1 \cdot f4 + 1 \cdot ae$

Für $2 \cdot d4$ gilt:

$(00000010)_2 \cdot (11010100)_2$

$= x \cdot (x^7 + x^6 + x^4 + x^2)$

$= x^8 + x^7 + x^5 + x^3$

$= (110101000)_2$

Multiplikation modulo $(x^8 + x^4 + x^3 + x + 1)$:

$(x^8 + x^7 + x^5 + x^3)\, mod\, (x^8 + x^4 + x^3 + x + 1)$

$= (110101000)_2\, mod\, (100011011)_2$

$= (110101000)_2 + (100011011)_2$ (im Galoiskörper)

$= (10110011)_2$

$= b3$

Für $3 \cdot 32$ gilt:

$(00000011)_2 \cdot (00110010)_2$

$= (x + 1) \cdot (x^5 + x^4 + x)$

$= x^6 + x^5 + x^4 + x^2 + x$

$= (01010110)_2$

Multiplikation modulo $(x^8 + x^4 + x^3 + x + 1)$:

$(x^6 + x^5 + x^4 + x^2 + x)\, mod\, (x^8 + x^4 + x^3 + x + 1)$

$= (01010110)_2 \, mod \, (100011011)_2$

$= (01010110)_2 + (100011011)_2$ (im Galoiskörper)

$= (01010110)_2$

$= 56$

Für $1 \cdot f4$ gilt:

$1 \cdot f4 = f4$

Für $1 \cdot ae$ gilt:

$1 \cdot ae = ae$

Es ergibt sich:

$b_{0j} = b3$ XOR 56 XOR $f4$ XOR $ae = bf$

2.2.1.4 Entschlüsselung

Für die Dechiffrierung von nach AES verschlüsselten Daten, wird genau entgegengesetzt der Verschlüsselung vorgegangen. Die verschlüsselten Daten werden zunächst in zweidimensionale Tabellen eingelesen und die für die Entschlüsselung notwendigen Rundenschlüssel erzeugt. Wichtig ist hierbei jedoch, dass die Schlüsselexpansion nun nicht mit Hilfe des originalen Chiffrierschlüssels geschieht, sondern dass hierzu der letzte Rundenschlüssel der Verschlüsselung genutzt wird. In der Praxis wird dieser über den Durchlauf einer kompletten Verschlüsselung mit dem originalen Chiffrierschlüssel und der Abspeicherung des Schlüssels der letzten Verschlüsselungsrunde generiert. Nach der Schlüsselexpansion wird der Verschlüsselungsalgorithmus rückwärts durchlaufen, wobei sich auch die Zeilenverschiebungen umkehren. Zudem muss für die Entschlüsselung das Inverse der Substitutionsbox genutzt werden, das sich aus der Substitutionsbox der Chiffrierung durch Vertauschung der der Zeilen- und Spaltenindizes mit den Substitutionswerten berechnen lässt. Als Beispiel hierzu ist in *Abbildung 2.7* die erste Zeile der inversen Substitutionsbox dargestellt.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB

Tab. 2.7: Inverse Substitutionsbox (vgl. [3])

2.2.2 Die Register des Verschlüsselungsmoduls

Die Operationen des Verschlüsselungsmoduls werden durch die Nutzung des Register AES_CTRL gesteuert. Des weiteren verfügt das Verschlüsselungsmodul über das Register AES_STATUS, mit dessen Hilfe der Status von Verschlüsselungs- und Entschlüsselungsoperationen überwacht werden kann. Die Pufferregister AES_STATE und AES_KEY dienen dem Speichern und Auslesen der Verschlüsselungsdaten und des Chiffrierschlüssels.

2.2.2.1 AES_CTRL (AES Control Register)

Die verschiedenen Registerbits dienen der Kontrolle des Verschlüsselungsmoduls. Zu beachten ist hierbei, dass die Registerwerte zurückgesetzt werden, wenn der Transceiver in den Zustand SLEEP versetzt wird. Außerdem sollten während Operationen der Verschlüsselungseinheit weder lesende noch schreibende Zugriffe auf das AES_CTRL-Register durchgeführt werden, da die aktuelle Operation hierdurch unterbrochen würde. *Tabelle 2.8* zeigt die Belegung der einzelnen Bits des Registers.

Bit-Nr.	Bezeichnung	Beschreibung
0	Res0	Reserviert
1	Res1	Reserviert
2	AES_IM	Kontrolle des AES-Interrupts
3	AES_DIR	Setzen der Betriebsrichtung
4	Res	Reserviert
5	AES_MODE	Auswahl Operationsmodus (EBC/CBC)
6	Res	Reserviert
7	AES_REQUEST	Starten der AES Operationen

Tab. 2.8: Belegung des Registers AES_CTRL (vgl. [1])

Bit 2 - AES_IM

Das Setzen des Bits AES_IM aktiviert die Interruptlogik des Verschlüsselungsmoduls.

Bit 3 - AES_DIR

Über das Bit AES_DIR wird die Richtung der AES-Operation ausgewählt. Die Zuordnung der Betriebsrichtung zu dem entsprechenden Bitwert kann folgender *Tabelle 2.9* entnommen werden.

Register-Bit	Wert	Beschreibung
AES_DIR	0	Verschlüsseln
	1	Entschlüsseln

Tab. 2.9: Das AES_DIR-Konfigurationsbit (vgl. [1])

Bit 5 - AES_MODE

Über das Bit AES_MODE des Registers AES_CTRL wird der Operationsmodus des Verschlüsselungsmoduls kontrolliert. *Tabelle 2.10* zeigt die Zuweisung der Werte zu den zugehörigen Operationsmodi.

Register-Bit	Wert	Beschreibung
AES_MODE	0	ECB (Electronic Code Book)
	1	CBC (Cipher Block Chaining)

Tab. 2.10: Das AES_MODE-Konfigurationsbit (vgl. [1])

Bit 7 - AES_REQUEST

Durch Setzen des Bits AES_REQUEST wird der Ver- oder Entschlüsselungsvorgang gestartet.

2.2.2.2 AES_STATUS (AES Status Register)

Über das AES Status Register werden Informationen zu dem Status der AES-Operationen abgerufen. Auf der nächsten Seite sind die Funktionen der einzelnen Bits des Registers in *Tabelle 2.11* auf Seite 20 dargestellt.

Bit-Nr.	Bezeichnung	Beschreibung
0	AES_DONE	Signalisiert die Beendigung einer AES-Operation
1	Res0	Reserviert
2	Res1	Reserviert
3	Res2	Reserviert
4	Res3	Reserviert
5	Res4	Reserviert
6	Res5	Reserviert
7	AES_ER	Zeigt Fehler während der AES-Operation an

Tab. 2.11: Belegung des Registers AES_STATUS (vgl. [1])

2.2.2.3 AES_STATE (Datenpuffer für Klar- und Chiffretext)

Das Register AES_STATE ist intern mit einem 16 Byte großen Datenpuffer verbunden, der den Datenblock aufnimmt, auf den die AES-Operationen angewendet werden und die durch die Operation entstandenen Daten speichert. Das Einlesen und Auslesen der Datenblöcke geschieht durch 16-faches Schreiben auf die, beziehungsweise 16-faches Lesen der Registeradresse. Zu beachten ist, dass unvollständiges Lesen aus oder Schreiben in den Puffer zu einem Fehler bei der Ausführung der AES-Operationen führt. Geht die Transceiver-Einheit in den Zustand SLEEP über, wird der Inhalt des Puffer gelöscht.

2.2.2.4 AES_KEY (Datenpuffer für den Chiffrierschlüssel)

Über das Register AES_KEY wird auf den internen, 128 Bit großen Datenpuffer zugegriffen, der dem Speichern und Abrufen der Chiffrierschlüssel dient. Wie schon bei dem zuvor beschriebenen Register AES_STATE geschieht der Zugriff auf die gepufferten Schlüsseldaten über 16-fache Schreiben, beziehungsweise 16-faches Lesen der Registeradresse. Während einer AES-Operation findet an dem innerhalb des Puffer gespeicherten Schlüssels keine Veränderung statt. Ist die AES-Operation beendet, liegt in dem Puffer der Schlüssel der letzten Verschlüsselungsrunde (Last Round Key), der benötigt wird um verschlüsselte Daten wieder entschlüsseln zu können.

2.3 Die Stromsparfunktionen des ATmega128RFA1

Da einige der innerhalb des AAL-Projektes genutzten Geräte frei von einer drahtgebundenen Energieversorgung sind und über Batterien oder Akkumulatoren ("Akkus") mit Energie versorgt werden, ist eine energieeffiziente Umsetzung dieser Systeme unerlässlich. Diese wird neben der generellen Auswahl von Hardwarekomponenten mit niedrigem Leistungsbedarf, durch die Nutzung der Stromsparfunktionen (Sleep Modes) der Mikrocontroller erreicht, über die nicht benötigte Funktionseinheiten abgeschaltet werden können. Der Mikrocontroller ATmega128RFA1 bietet hierzu verschiedene Modi, die in den folgenden Abschnitten beschrieben werden.

2.3.1 Nutzung der AVR Sleep Modes

	Osz.		Interrupt-Quellen									
Sleep Mode	Main Clock	Timer Oszillator	INT0 - INT7	Pin Change	TWI Adr. Match	Timer/Counter2	SPM/EEPROM	ADC	Watchdog	Andere I/O	Symbol Counter	Transceiver
Idle	X	X	X	X	X	X	X	X	X	X	X	X
ADCNRM	X	X	X	X	X	X	X	X	X		X	X
Power-down			X	X	X				X		X	X
Power-Save		X	X	X	X	X			X		X	X
Standby	X		X	X	X				X		X	X
Extended Standby	X	X	X	X	X	X			X		X	X

Tab. 2.12: Vergleich der Sleep Modes des ATmega128RFA1 (vgl. [1])

Die Konfiguration der Sleep Modes erfolgt über das SMCR-Register des ATmega128RFA1. Um die Sleep Modes zu nutzen, muss nach der Auswahl des zu benutzenden Modus über die Konfigurationsbits SM0 bis SM2 die Sleep-Funktion zunächst durch Setzen des Kontrollbits SE aktiviert werden. Hierauf kann das in der AVR-Headerdatei "sleep.h" enthaltene Makro sleep_cpu() aufgerufen werden, worauf der Mikrocontroller in den zuvor gewählten Sleep Mode versetzt wird. *Tabelle 2.12* zeigt die innerhalb der einzelnen Sleep Modes aktiven Oszillatoren und Interrupt-Quellen.

2.3.2 Der Deep-Sleep Mode

Falls der Mikrocontroller in die Modi Power-down oder Power-Save versetzt wird, während sich die Transceiver-Einheit in dem Zustand SLEEP befindet, geht dieser in den Stromsparmodus Deep-Sleep über, bei dem die geringste Leistung aller Modi aufgenommen wird. Zu beachten ist hierbei, dass in diesem Modus die Spannungsversorgung der A/D-Wandler[8] direkt herunter gefahren wird. Daher sollten diese zuvor abgeschaltet werden, um undefinierte Operationen der A/D-Einheit zu vermeiden. Zudem muss der Timer/Counter 2 des ATmega128RFA1 auf asynchronen Betrieb eingestellt sein, da ansonsten der Hauptoszillator aktiviert bleibt und der Controller nicht in den Deep-Sleep Mode übergehen kann.

2.3.3 Typische Stromaufnahme

Um dem Leser einen Überblick über die Leistungsaufnahme in den verschiedenen Betriebszuständen zu geben, sind in *Tabelle 2.13* typische Betriebszustände und die entsprechende Stromaufnahme abgebildet. Die in der Tabelle aufgelisteten Werte beziehen sich hierbei auf eine Außentemperatur von 25°C und dem Betrieb des Controllers mit einer Betriebsspannung von 3,0 V bei abgeschalteter Transceiver-Einheit.

Betriebszustand und Nebenbedingungen	Stromaufnahme
Standby	0,31 mA
Idle, 16 MHz	1,20 mA
Active, 16 MHz CLKI	4,50 mA
Power-save, 32 kHz Oszillator eingeschaltet	1,65 µA
Deep-Sleep, Watchdog abgeschaltet	0,25 µA

Tab. 2.13: Typische Stromaufnahme des ATmega128RFA1 (vgl. [1])

2.3.4 Beschreibung der Register für die Steuerung der Leistungsaufnahme

Neben dem SMCR-Register des ATmega128RFA1, das der Steuerung der Sleep Modes dient, verfügt der Mikrocontroller über weitere Register, mit Hilfe derer einzelne

[8] Analog/Digital-Wandler

Funktionseinheiten des Controllers deaktiviert werden können. Innerhalb der folgenden Abschnitte werden daher die Zuordnung der Sleep Modes zu den Registerwerten des SMCR-Registers und die Funktionen der einzelnen Registerbits der Register PRR0 bis PRR2 dargestellt.

2.3.4.1 SMCR (Sleep Mode Control Register)

Folgende *Tabelle 2.14* zeigt die Bedeutung der einzelnen Registerbits des Registers SMCR.

Bit-Nr.	Bezeichnung	Beschreibung
0	SE	Schaltet die Nutzung der Sleep Modes frei
1	SM0	Auswahl des Sleep Modes
2	SM1	Auswahl des Sleep Modes
3	SM2	Auswahl des Sleep Modes
4	Res0	Reserviert
5	Res1	Reserviert
6	Res2	Reserviert
7	Res3	Reserviert

Tab. 2.14: Das Register SMCR (vgl. [1])

Die Registerbits SM0 bis SM2

Tabelle 2.15 können die zu den einzelnen Sleep Modes zugeordneten Werte für die Registerbits SM0 bis SM2 entnommen werden.

Wert	Eingestellter Sleep Mode
0x00	Idle
0x01	ADC Noise Reduction (ADCNR)
0x02	Power Down
0x03	Power-Save
0x04	Reserviert
0x05	Reserviert
0x06	Standby
0x07	Extended Standby

Tab. 2.15: Zuordnung der Sleep Modes zu den Registerwerten SM0 bis SM2 (vgl. [1])

2.3.4.2 PRR (Power Reduction Register)

Über die Bits der Register PRR0 bis PRR2 wird die Deaktivierung einzelner Funktionseinheiten des Mikrocontroller gesteuert. Obwohl diese Funktionen während der Arbeit an dem Bachelorprojekt zunächst keine Rolle spielen werden, wird der Vollständigkeit halber in der folgenden *Tabelle 2.16* die Bedeutung der einzelnen Registerbits dargestellt.

Register	Bit-Nr.	Bezeichnung	Auswirkung auf
PRR0	0	PRADC	A/D-Wandler
	1	PRUSART0	USART0-Schnittstelle
	2	PRSPI	SPI-Schnittstelle
	3	PRTIM1	Timer/Counter 1
	4	PRPGA	PGA (Programmable Gain Amplifier)
	5	PRTIM0	Timer/Counter 0
	6	PRTIM2	Timer/Counter 2
	7	PRTWI	TWI-Schnittestelle
PRR1	0	PRUSART1	USART1-Schnistelle
	1	-	-
	2	-	-
	3	PRTIM3	Timer/Counter 3
	4	PRTIM4	Timer/Counter 4
	5	PRTIM5	Timer/Counter 5
	6	PRTRX24	Transceiver-Einheit
	7	Res	Reserviert
PRR2	0	PRRAM0	SRAM Block 0
	1	PRRAM1	SRAM Block 1
	2	PRRAM2	SRAM Block 2
	3	PRRAM3	SRAM Block 3
	4	Res0	Reserviert
	5	Res1	Reserviert
	6	Res2	Reserviert
	7	Res3	Reserviert

Tab. 2.16: Die Registerbits der Register PRR0 bis PRR2 (vgl. [1])

3 Hardwaredesign

3.1 Aufbau des IPv6-Funkmoduls

3.1.1 Anforderungen

Für das innerhalb dieses Bachelorprojektes neu zu entwickelnde Funkmodul wurden folgende Anforderungen gestellt:

- Nutzung des Mikrocontrollers ATmega128RFA1
- Beibehaltung des Formfaktors des bisher genutzten Funkmoduls
- Herausführung aller Ports über Stiftleisten zur Nutzung des Funkmoduls mit der Entwicklungsplatine und Komponenten des AAL-Systems
- Beibehaltung der Lage der Ports auf den Stiftleisten (falls möglich)
- Veränderung des Layouts der HF-Einheit (siehe *Kapitel 3.1.2*)

3.1.2 Design und Anpassung der Antenne

Da das zuvor genutzte Funkmodul bei dem Versand von großen Datenpaketen und der Nutzung hoher Übertragungsraten nicht nur Probleme hinsichtlich der Reichweite machte, sondern es zudem zu massiven Paketverlusten kam, musste für das neue Funkmodul das Design der Antenne und der Hochfrequenzeinheit überdacht werden. Hierzu wurde zunächst ermittelt, an welcher Stelle des alten Platinendesigns ein Fehler vorlag, der für die suboptimale Leistung des Sendemoduls verantwortlich sein könnte.

Nach Untersuchung der Application Notes und Datenblätter der von Texas Instruments designten PCB[1]-Antenne[6] und des Transceiver-Chips AT86RF230[7] des Herstellers Atmel stellte sich heraus, dass entgegen vorheriger Annahmen die Antenne nicht ohne Anpassungen mit dem Transceiver-Chip verbunden werden kann. Hintergrund ist, dass

[1] Printed Circuit Board

es sich bei der Antenne um eine asymmetrische Antenne mit einer Impedanz von 50 Ω handelt, während der Ausgang des Transceivers für eine Impedanz von 100 Ω und die Nutzung einer symmetrischen Antenne ausgelegt ist. Diese Überlegungen sind ebenso für das Transceiver-Modul des in diesem Bachelorprojekt genutzten ATmega128RFA1 gültig, da auch dieser Mikrocontroller über einen symmetrischen Antennenausgang mit einer Impedanz von 100 Ω verfügt.

Daher wurde von Mitarbeitern des Labors für Informatik an der Fachhochschule Düsseldorf ein neues Design der Hochfrequenzeinheit entwickelt. Dieses bestand aus einer keramischen Chipantenne des Herstellers Johanson Technology (siehe [8]). Da diese Antenne ebenfalls asymmetrisch konstruiert ist, musste für die Kopplung der Antenne mit dem Transceiver-Chip ein Balun[2] (siehe [9]) genutzt werden, das für die Umwandlung des symmetrischen Ausgangssignals des Transceiver-Chips in ein asymmetrisches Signal sorgt. Zudem wurde ein Anpassnetzwerk entworfen, um die Impedanz der Antenne von 50 Ω an die Ausgangsimpedanz des Transceivers anzupassen.

Eine weitere Überlegung, an der ich beteiligt war, betraf den Wellenwiderstand der Leiterbahnen zwischen den Komponenten der Hochfrequenzeinheit. Diese sollten für optimale Ergebnisse den Abschlussimpedanzen an der Ein- bzw. Ausgangsseite entsprechen. Die Berechnung des Wellenwiderstandes, beziehungsweise der Breite der Leiterbahn basierte hierbei auf folgender, von der Association Connecting Electronics Industries (IPC)[3] verifizierter Formel [10]:

$$z_n = \frac{87\,\Omega}{\sqrt{\epsilon_r+\sqrt{2}}} \cdot \ln\left(\frac{5{,}98\,h}{0{,}8\,w+t_{cu}}\right)$$

mit

z_n: Wellenwiderstand

ϵ_r: Dielektrizitätskonstante der verwendeten FR4-Platine (hier 4,7)

h: Höhe der Platine (hier 1,5 mm)

w: Breite der Leiterbahn

t_{cu}: Höhe der Kupferschicht der Leiterbahn (hier 35 µm)

Die Umstellung der Formel nach w liefert:

$$w = \frac{e^{-\frac{z_n}{87\,\Omega}\cdot\sqrt{\epsilon_r+\sqrt{2}}}\cdot 5{,}98\cdot h - t_{cu}}{0{,}8}$$

[2] Symmetrierglied
[3] Weltweite Handels- und Standardisierungsorganisation

Einsetzen der Zahlenwerte liefert für $z_n = 100\,\Omega$:

$$w = \frac{e^{-\frac{100\,\Omega}{87\,\Omega}\cdot\sqrt{4{,}7+\sqrt{2}}\cdot 5{,}98\cdot 1{,}5\cdot 10^{-3}-35\cdot 10^{-6}}}{0{,}8} \approx 0{,}61\,mm$$

und für $z_n = 50\,\Omega$:

$$w = \frac{e^{-\frac{50\,\Omega}{87\,\Omega}\cdot\sqrt{4{,}7+\sqrt{2}}\cdot 5{,}98\cdot 1{,}5\cdot 10^{-3}-35\cdot 10^{-6}}}{0{,}8} \approx 2{,}66\,mm$$

Da die Leiterbahnen in dieser Breite jedoch nicht auf dem Funkmodul zu realisieren waren, wurden die Berechnungen verworfen und versucht, die Antenne und das Anpassnetzwerk möglichst nahe an den Ausgang des Transceivers zu setzen, um den Einfluss der Leiterbahnbreite zu minimieren.

Erste Versuche mit einer nach dem neuen Design entwickelten Hochfrequenzeinheit durch den ebenfalls im Labor für Informatik an der Fachhochschule Düsseldorf arbeitenden Kommilitonen Wolfram Gerlach hatten jedoch zum Ergebnis, dass die Kommunikation schlechter funktionierte als zuvor mit der unangepassten Antenne. Daher wurde dieses Design für die Durchführung dieses Bachelorprojektes verworfen und nach neuen Möglichkeiten für die Realisierung der Antenne gesucht.

Nach Beratungen mit Herrn Prof. Dr.-Ing. Gregor Gronau, der unter anderem das Studienfach Höchstfrequenztechnik an der Fachhochschule Düsseldorf lehrt und als Experte für Antennen gilt, wurde das ursprüngliche Design mit der Folded F PCB-Antenne in der Weise abgeändert, dass zwei dieser Antennen in Reihe geschaltet wurden. Durch diese Reihenschaltung der asymmetrischen Antennen entsteht ein symmetrischer Dipol mit einer Impedanz, die ungefähr 100 Ω entspricht.

Abbildung 3.1 zeigt das für das Layout-Programms EAGLE[4] entworfene Antennensymbol und das Design der in Reihe geschalteten Folded F PCB-Antennen.

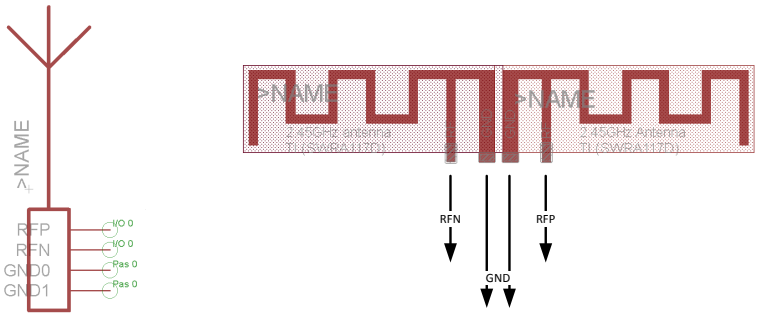

Abb. 3.1: Symbol und Schaltung der verwendeten Antenne

[4]Einfach Anzuwendender Grafischer Layout Editor

3.1.3 Layout der Platine

Da der Mikrocontroller ATmega128RFA1 in seinen elektrischen Eigenschaften verwandt zu dem ebenfalls von dem Hersteller Atmel stammenden Mikrocontroller ATmeag1284P ist und ebenfalls als QFN[5]-Bauteil vorliegt, konnten wesentliche Designelemente der Platine von dem in der Bachelorarbeit von Herrn Bernhard Esders[11] erstellten Platinendesign übernommen werden.

Für das Design der Platine wurden zwei neue EAGLE-Bibliotheken erstellt. Die erste Bibliothek mit der Bezeichnung "atmega128rfa1_mod.lbr", beinhaltet das Symbol des ATmega128RFA1 mit dessen Ein- und Ausgängen für die Erstellung des Schaltplans und das so genannte Package[6] des Controllers für das Layout der Platine. Um den Controller manuell verlöten zu können, wurden die Pads (Anschlussstellen) des Controllers gegenüber der Bibliothek, die über den Distributor Farnell bezogen werden kann, verlängert. Für eine übersichtlichere Gestaltung des Schaltplans, wurden zudem die Lage einiger Ein- und Ausgänge des Schaltplan-Symbols verändert. Die zweite Bibliothek umfasst alle Bauteile, die für das Design eines auf den Mikrocontroller basierenden Funkmoduls notwendig und nicht in sonstigen Standardbibliotheken enthalten sind.

3.1.3.1 Routing der Ports

Wie in *Abbildung 3.2* auf Seite 29 ersichtlich ist, unterscheidet sich die Anzahl und Lage der Ein- und Ausgänge des ATmega128RFA1-Mikrocontrollers deutlich von denen des zuvor verwendeten ATmega1284P. Durch die Vorgabe alle Ports nach außen zu führen, um auch für spätere, noch nicht vorhersehbare Anwendungen gerüstet zu sein, musste daher auf die Kompatibilität der Funkmodule, in Hinblick auf die Lage der Ports an den Stiftleisten, verzichtet werden. Daher konnte das Layout des neuen Funkmoduls und die Verbindungen zu den Stiftleisten so gewählt werden, dass möglichst wenig Kreuzungen von Leiterbahnen entstanden, die bei dem Design der Platine über so genannte Vias hätten realisiert werden müssen. Außerdem wurde auf die Herausführung von PORT G verzichtet, da die Funktionen dieses Ports voraussichtlich auch in Zukunft ungenutzt bleiben werden und ansonsten wegen der zu geringen Anzahl an Kontakten auf den Stiftleisten eine weitere Leiste hätte angebracht werden müssen.

[5] Quad Flat No Leads
[6] Abbildung des Bauteils mit dessen Anschlussstellen

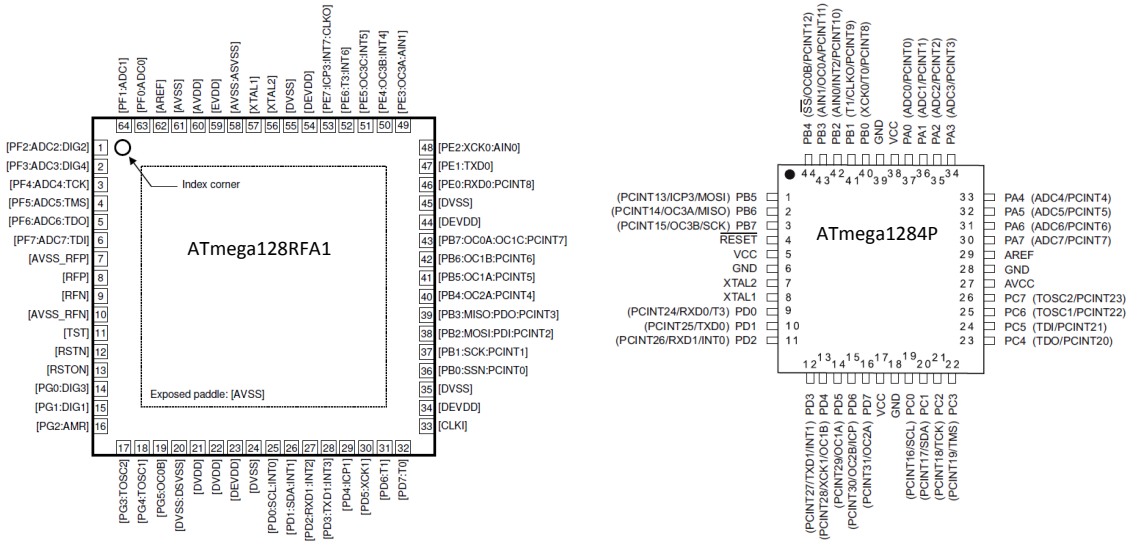

Abb. 3.2: Ein- und Ausgänge der Controller ATmega128RFA1 und ATmega1284P ([1, 2])

Abbildung 3.3 auf Seite 30 zeigt den Schaltplan des Funkmoduls und lässt die Anordnung der verschiedenen Ein- und Ausgänge des ATmega128RFA1 auf den Stiftleiten erkennen. Zu beachten ist bei der Nutzung der Ports jedoch, dass viele Anschlüsse des ATmega128RFA1 mehrfach belegt sind. Dies gilt insbesondere für die Anschlüsse der JTAG[7]-Programmierschnittstelle, auf die neben dieser Funktion zusätzlich die A/D-Wandler ADC4 bis ADC7 herausgeführt sind.

[7] Joint Test Action Group

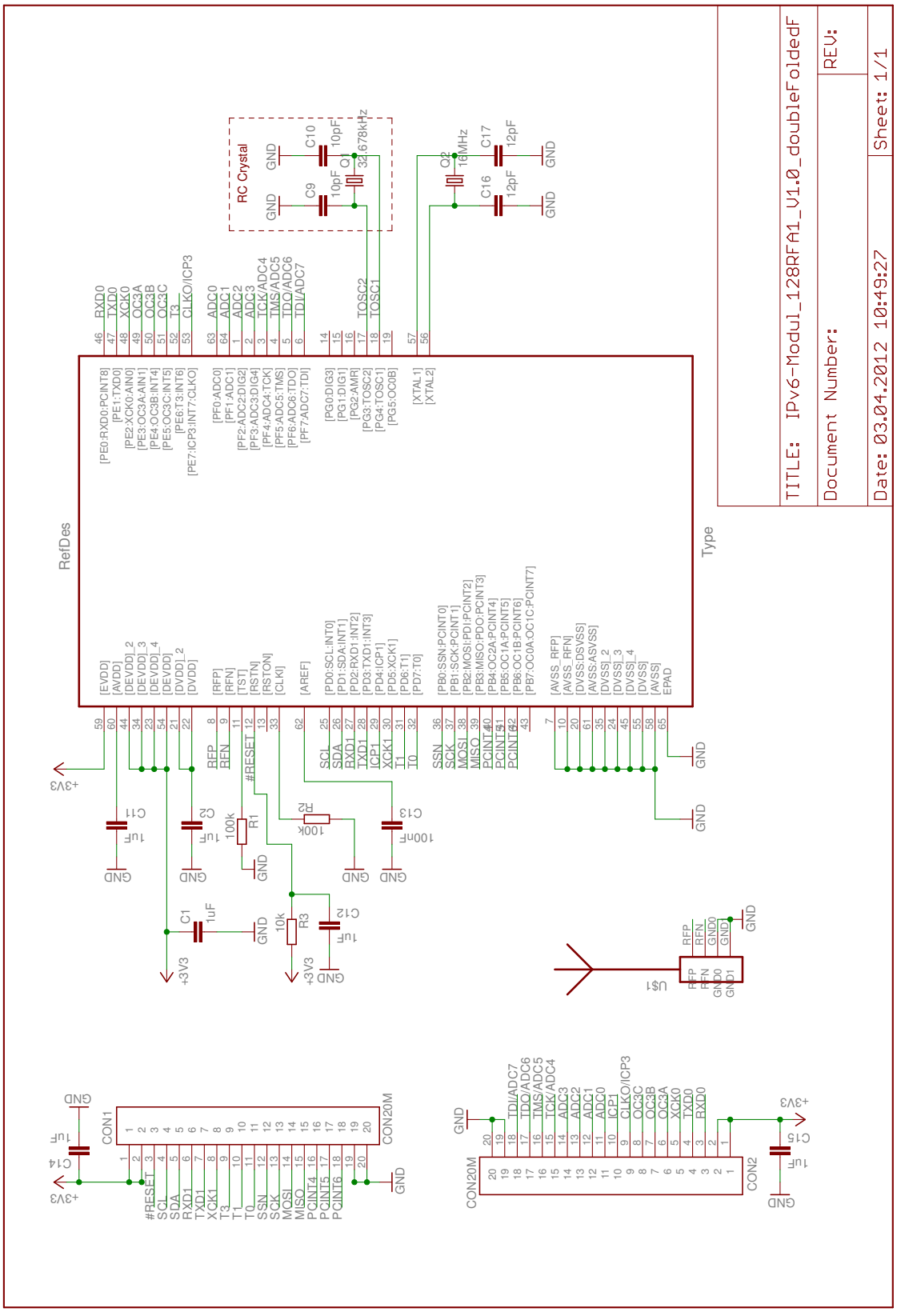

Abb. 3.3: IPv6-Funkmodul

3.2 Aufbau der Entwicklungsplatine (DevBoard)

Durch die veränderte Anordnung der Ein- und Ausgänge des Mikrocontrollers und die damit verbundene Inkompatibilität mit der alten Entwicklungsplatine wurde, neben der Realisierung des neuen Funkmoduls, ein Redesign dieser Platine nötig.

3.2.1 Anforderungen

An die neu zu entwickelnde Entwicklungsplatine wurden folgende Anforderungen gestellt:

- Spannungsversorgung über den USB-Anschluss oder über ein externes Netzteil
- Anbindung beider seriellen Ports des ATmega128RFA1 an die USB[8]-Schnittstelle
- Herausführung aller auf den Stiftleisten liegenden Anschlüsse
- Realisierung separater Anschlussmöglichkeiten für die Schnittstellen
  - JTAG,
  - ISP[9],
  - USART0 und USART1[10],
  - und TWI[11]

3.2.2 Layout der Platine

Zur Verringerung des Arbeitsaufwands konnten wesentliche Einheiten des alten Layouts, wie die Regelung der Spannungsversorgung und die Schaltung zur Umsetzung der seriellen Ports des Mikrocontrollers (`USART0` und `USART1`) auf den USB-Anschluss erhalten bleiben. Durch die Vergrößerung der Anzahl an Schnittstellen war es jedoch nötig, die mit den Ports des Mikrocontrollers verbundenen Pfostenstecker neu zu platzieren und die Zuleitungen zu diesen neu zu designen.

Der Schaltplan des neuen Entwicklungsboards ist in *Abbildung 3.4* auf Seite 32 ersichtlich.

[8] Universal Serial Bus
[9] In System Programming
[10] Universal Synchronous Asynchronous Receiver Transmitter
[11] Two Wire Interface (auch I²C-Bus genannt)

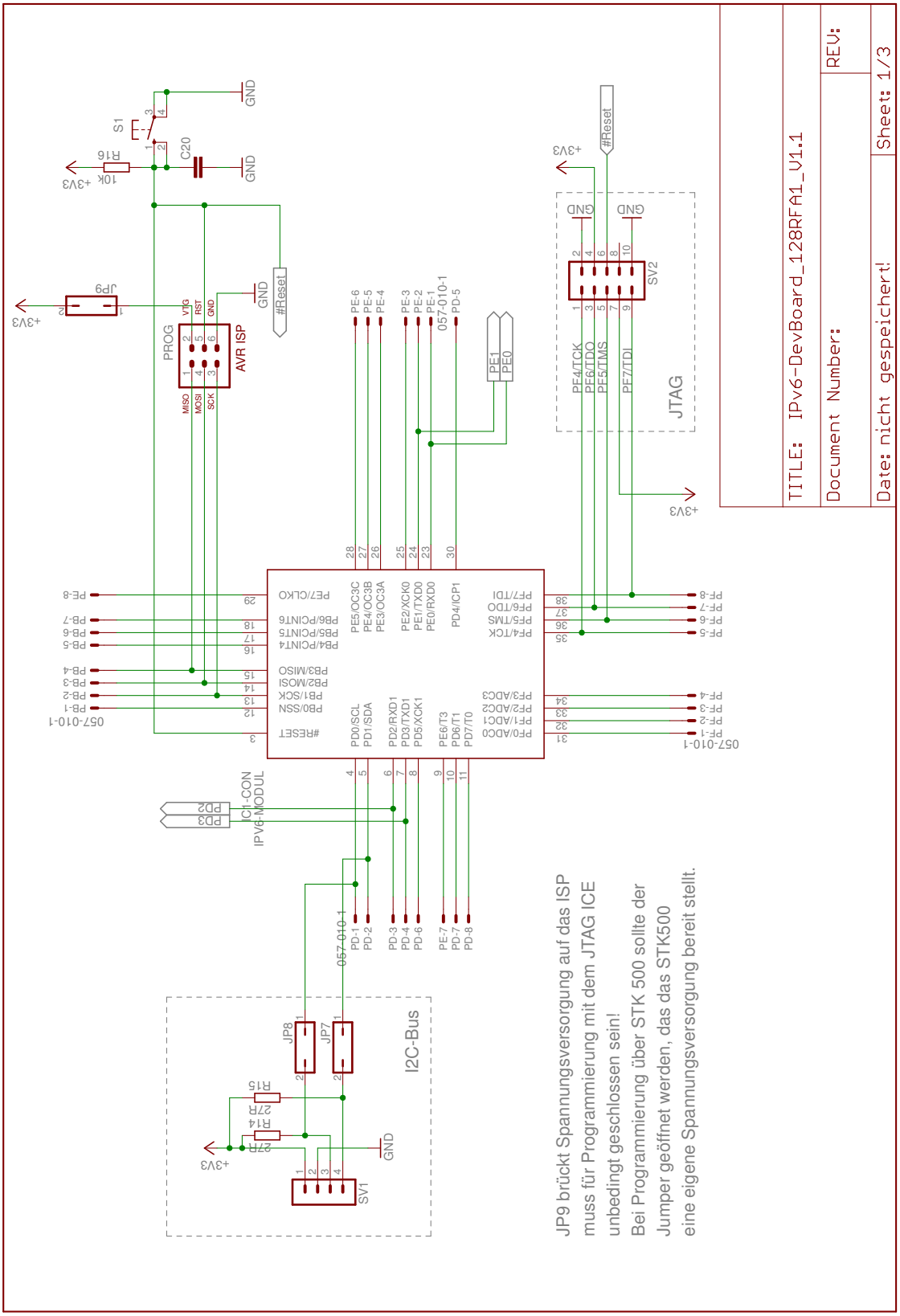

Abb. 3.4: IPv6-Entwicklungsplatine

4 Softwaredesign

Nachdem innerhalb des Praxisprojektes Grundlagen zu der Kommunikation über das 6LoWPAN-Protokoll unter Verwendung des Mikrocontrollers ATmega128RFA1 erarbeitet wurden, werden innerhalb dieses Kapitels die softwaretechnische Erweiterungen vorgestellt, die die Nutzung der Hardwareverschlüsselung und die Verbesserung der Energieeffizienz betreffen.

4.1 Realisierung der Hardware-Verschlüsselung

Da in *Kapitel 2.2.2* dieser Bachelorthesis die für die Realisierung der Hardware-Verschlüsselung notwendigen Register des Mikrocontroller ATmega128RFA1 bereits vorgestellt wurden, wird an dieser Stelle ein Einblick in den programmatischen Ablauf der Verschlüsselungsfunktion und die Implementierung der notwendigen Funktionen gegeben. *Abbildung 4.1* auf Seite 34 verdeutlicht anhand von Programmablaufplänen (PAP) den grundlegenden Ablauf der Ver- und Entschlüsselung. Innerhalb der Programmablaufpläne wird der Einfachheit halber zunächst davon ausgegangen, dass die zu ver- oder entschlüsselnden Daten bereits in Pakete mit einer Größe von 16 Byte aufgeteilt wurden.

Eine auch in dem Programmablaufplan für die Entschlüsselung dargestellte Besonderheit liegt darin, dass der letzte Rundenschlüssel einer vorherigen Verschlüsselung nicht direkt für eine direkt folgende Entschlüsselung verwendet werden kann. Auch wenn die Entschlüsselung direkt nach der Verschlüsselung gestartet wird, ist es notwendig, den letzten Rundenschlüssel zunächst auszulesen, zwischenzuspeichern und daraufhin wieder in den Puffer zu schreiben. Jede andere Vorgehensweise führte in verschiedenen Versuchen zu unvorhersehbaren Ergebnissen des Entschlüsselungsvorganges.

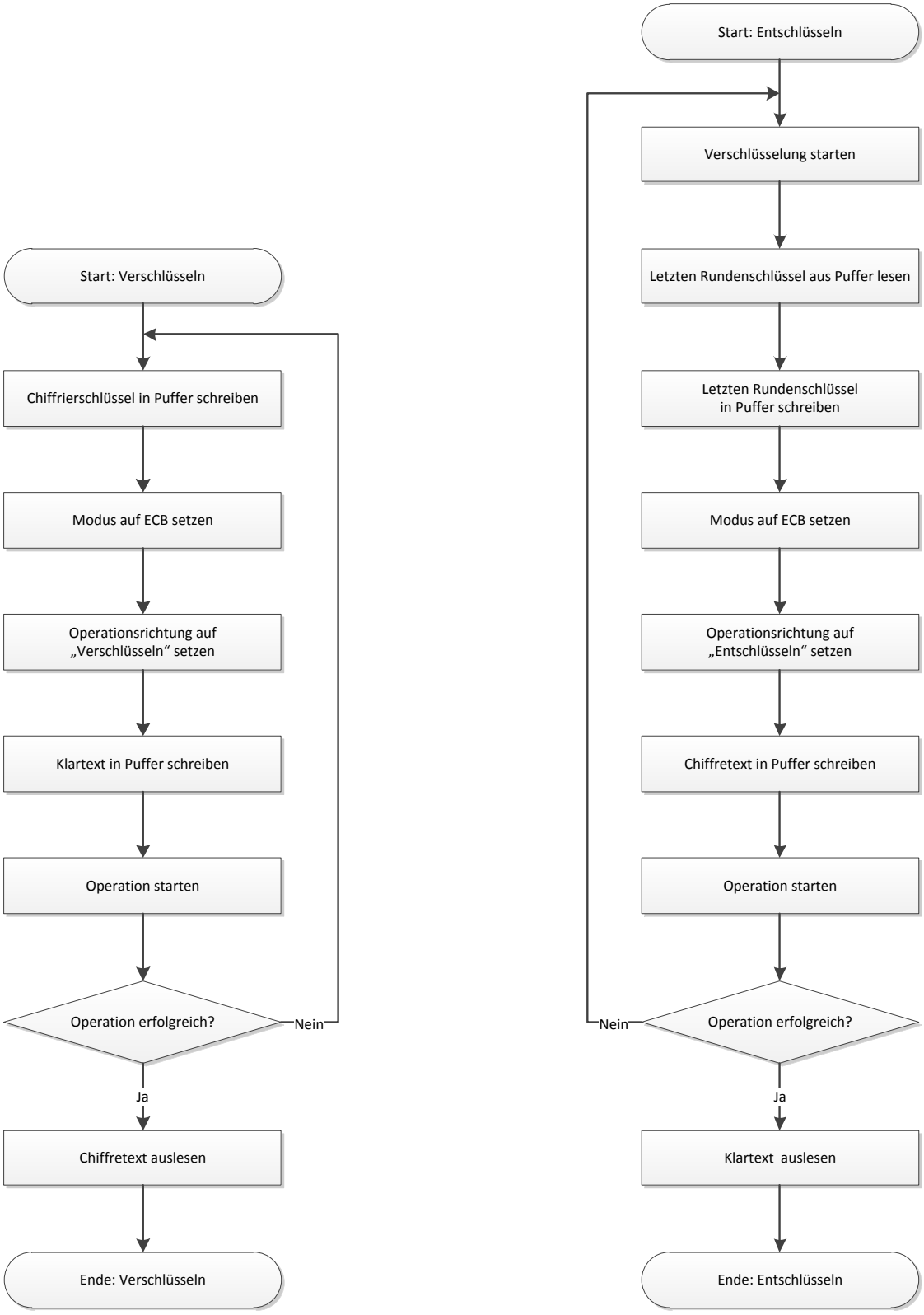

Abb. 4.1: Ablauf der Ver- und Entschlüsselung

4.1.1 Implementierung eines Testprogramms

Für die Durchführung erster Funktionstests des Verschlüsselungsmoduls wurde ein Testprogramm entwickelt, mit dessen Hilfe das Ergebnis des Ver- und Entschlüsselungsvorganges überprüft werden kann. Das Programm arbeitet in der Weise, dass zunächst ein String mit einer Länge von 16 Byte (16 Zeichen) mit einem 128 Bit langen Schlüssel verschlüsselt wird. Der entstehende Chiffretext wird hierauf direkt wieder entschlüsselt und der resultierende Klartext-String mit dem ursprünglichen String verglichen. Stimmt der erzeugte Klartext-String mit dem originalen String überein, wird die Übereinstimmung über das Einschalten einer LED auf der Evaluations-Platine angezeigt. Der Ablauf des Programms ist in *Abbildung 4.2* dargestellt.

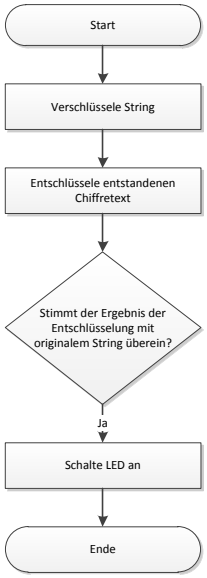

Abb. 4.2: Das AES-Testprogramm

4.1.2 Implementierung der Verschlüsselungsfunktionen in Contiki 2.5

Neben der Realisierung von grundlegenden Funktionalitäten für die Nutzung des AES-Verschlüsselungsmoduls des ATmega128RFA1 war ein Ziel, diese Funktionen innerhalb des Contiki-Betriebssystems als Bibliothek nutzbar zu machen. Hierzu wurden die Methoden für die Verschlüsselung, die Entschlüsselung und die Erzeugung des letzten Rundenschlüssels in die Datei aes16.c ausgegliedert und die notwendige Headerdatei aes16.h implementiert und diese innerhalb des entsprechenden Plattform-Verzeichnisses von Contiki 2.5 (/platform/avr-atmega128rfa1) abgespeichert. Zudem wurde die Funktionalität dahingehend erweitert, dass die Ver- und Entschlüsselungsfunktionen mit einer variablen Klartextgröße genutzt werden können. *Listing 9.1* im Anhang dieser Bachelorthesis zeigt die für die Verschlüsselungsbibliothek erstellten C-Funktionen. Um die Funktionen direkt als Bibliothek innerhalb des Betriebssystems Contiki 2.5 nutzbar zu machen, wurde der Datei contiki-main.c (Hauptdatei des plattformabhängigen Codes des Contiki-Betriebssystems), die ebenfalls innerhalb des Plattform-Verzeichnisses angesiedelt ist, der in *Listing 4.1* auf Seite 36 abgebildete Quelltext hinzugefügt. Diese Maßnahme ermöglicht es späteren Nutzern, eigene auf den Verschlüsselungsfunktionen basierende Anwendungen zu schreiben, ohne dass die benötigten Header- und Quellcode-Dateien

manuell hinzugefügt werden müssen. Auf die Nutzung der AES-Bibliothek wird in *Kapitel 5.3* gesondert eingegangen.

```
1 #ifdef USE_HWAES
2 #ifndef __AES_h__
3 #include "aes.h"
4 #endif
5 #include "aes.c"
6 #endif
```

Listing 4.1: Änderung innerhalb von contiki-main.c für die automatische Einbindung der AES-Bibliothek

Auf Grund des endlichen Speicherplatzes für Variablen innerhalb des 16 kByte großen SRAM des AVR Mikrocontrollers wurde die maximale Größe der verwendeten Klartext- und Chiffrestrings zunächst auf eine feste Größe von 96 Zeichen (entspricht 96 Byte) festgelegt, die bei der Einbindung der Bibliothek im Speicher des Controllers reserviert werden. Eine Optimierungsmöglichkeit der Funktionen würde hier darin liegen, die maximale Größe der genutzten Strings variabel zu gestalten. Dies kann über die Nutzung von Allokationen geschehen, über die den Variablen zur Laufzeit des Programmes dynamisch Speicherplatz zugewiesen wird.

4.2 Realisierung der Stromsparfunktionen in Contiki 2.5

Bei der Realisierung der Stromsparfunktionen muss zwischen der Portierung und Optimierungen der innerhalb des Bachelorprojekts von Herrn Bernhard Esders [11] vorgestellten Funktionen für die RAVEN-Hardware und der Implementierung der Funktionen für den Mikrocontroller ATmega128RFA1 unterschieden werden. Es sei jedoch an dieser Stelle ausdrücklich darauf hingewiesen, dass beide Implementierungen auf die Grundgedanken von Herrn Esders zurück gehen, welche insbesondere die Logik für die Verwaltung der Freigabe des Schlafmodus bei der Verwendung von Multithreading und das Umlegen des genutzten Timers auf Timer 2 des Mikrocontrollers einschließen. Allerdings wurden die Funktionen von Herrn Esders dahingehend erweitert, dass der Mikrocontroller und die Transceiver-Einheit getrennt für die Nutzung des Schlafmodus freigegeben werden können. Dies kann im Kontext drahtloser Sensornetzwerke insbesondere dann wichtig werden, wenn auf Grund der geringen Sendeleistung der einzelnen Sensorknoten die Reichweite der Funkübertragung durch die Nutzung von Weiterleitungsfunktionen erhöht werden soll. Zudem ist es nun möglich, die notwendigen Bibliotheken ohne explizite Einbindung der

benötigten Headerdateien zu nutzen. Eine Anleitung für die Nutzung der entsprechenden Bibliotheken folgt in *Kapitel 5.2* dieser Bachelorthesis.

Für die Realisierung der Sleep Modes wurden folgende Dateien modifiziert, beziehungsweise eingefügt:

- /core/contiki.h
- /cpu/avr/dev/clock.c
- /platform/avr-raven/sleep.h
- /platform/avr-raven/contiki-raven-main.c
- /platform/avr-atmega128rfa1/sleep.h
- /platform/avr-atmega128rfa1/contiki-main.c

4.2.1 Implementierung für das RAVEN-System

Bei der Portierung der Stromsparfunktionen für das RAVEN-System konnte sich weitestgehend darauf beschränkt werden, die Headerdatei für die Verwaltung der Sleep Modes (/platform/avr-raven/sleep.h) bei der Verwendung von Multithreading dahingehend zu ändern, dass dem Programmierer einer Applikation die Möglichkeit gegeben wird, den Mikrocontroller ATmega1284P und den Transceiver-Chip AT86RF230 getrennt in den Schlafmodus zu versetzen. Von Bedeutung ist hierbei, dass zwei getrennte Variablen (Semaphoren) und Funktionen für die Transceiver-Einheit und die CPU implementiert wurden, über welche die Threads die Freigabe des Schlafmodus steuern.

Zudem wurde die innerhalb des Verzeichnisses der RAVEN-Plattform zu findende Datei contiki-raven-main.c in der Art modifiziert, dass die getrennte Nutzung der Schlafmodi und die Einbindung der benötigten Bibliothek über das für die Kompilierung notwendige Makefile ermöglicht wird. *Listing 9.2* zeigt die an der Datei contiki-raven-main.c vorgenommenen Änderungen.

4.2.2 Implementierung für den ATmega128RFA1

Die Implementierungen für den ATmega128RFA1 gestalten sich im Grunde analog zu der Realisierung der Stromsparfunktionen der RAVEN-Plattform. Allerdings bietet der im Contiki-Betriebssystem enthaltene Kommunikationstreiber für den ATmega128RFA1 keine eigenen Funktionen für die Steuerung des Schlafzustandes der Transceiver-Einheit.

Daher wurde die Steuerung der Sleep-Modes manuell in die Datei contiki-main.c hinzugefügt. *Listing 4.2* auf Seite 38 dieser Bachelorthesis bietet einen Überblick über die wichtigsten Änderungen an der Datei contiki-main.c, die sich in Hinblick auf die Stromsparfunktionen, zu den in die Datei contiki-raven-main.c (siehe *Listing 9.2*) eingefügten Implementierungen, unterscheiden. Zudem ist innerhalb des Listings die Nutzung der Register der Transceiver-Einheit für die Herbeiführung einer Zustandsänderung ersichtlich.

```
1  [...]
2
3  //contiki main scheduler loop
4  int
5  main(void)
6  {
7    initialize();
8
9    while(1) {
10     process_run();
11     watchdog_periodic();
12
13 #ifdef USE_SLEEP
14   set_sleep_mode(SLEEP_MODE_PWR_SAVE);
15   sleep_enable();
16
17   if(allow_rf_sleep == 0)
18   {
19     //set radio to TRX_OFF
20     hal_subregister_write(SR_TRX_CMD, CMD_FORCE_TRX_OFF);
21     delay_us(TIME_CMD_FORCE_TRX_OFF);
22
23     //set radio to sleep
24     hal_set_slptr_high();
25   }
26   else
27   {
28     //wake up radio
29     hal_set_slptr_low();
30     delay_us(2 * TIME_SLEEP_TO_TRX_OFF);
31   }
32
33   if(allow_cpu_sleep == 0)
```

```
34    {
35      // set ATmeag128RFA1 to sleep
36      sleep_cpu();
37    }
38  #endif
39
40  [...]
```

Listing 4.2: Manuelle Programmierung der Sleep Modes des ATmega128RFA1

4.2.3 Zusammenhang zwischen den Dateien "contiki-main.c" und "contiki-raven-main.c" und den Stromsparfunktionen

Die Dateien contiki-main.c und contiki-raven-main.c beinhalten Funktionen und Definitionen für die Steuerung verschiedener Komponenten des ATmega128RFA1-Mikrocontrollers und der RAVEN-Hardware. Ein Element dieser Dateien ist der so genannte "main scheduler loop", der während der Ausführung des Betriebssystems fortwährend durchlaufen wird. Da dies auch bei Kontextwechseln innerhalb des Betriebssystems der Fall ist, eignet sich diese Stelle insbesondere für die Überwachung und Aktivierung, beziehungsweise Deaktivierung, der Stromsparfunktionen. Bei der in *Listing 4.2* abgebildeten Implementierung wird bei jedem Durchlauf der while-Schleife überprüft, ob die Semaphoren allow_rf_sleep und allow_cpu_sleep durch die laufenden Threads auf den Wert "0" gesetzt wurden und somit alle Threads das Schlafen des Mikrocontrollers oder der Transceiver-Einheit akzeptieren. Ist dies der Fall, werden die notwendigen Schritte ausgeführt, die zur Abschaltung des Transceivers oder zum Wechsel der CPU in den Modus Power-Safe notwendig sind. Wird durch den Semaphor allow_rf_sleep signalisiert, dass ein Prozess Zugriff auf die Transceiver-Einheit benötigt, wird der Transceiver wieder eingeschaltet. Das Aufwecken der CPU geschieht automatisch über den internen Systemtakt.

5 Arbeiten mit Contiki 2.5

Um späteren Nutzern des Systems die Programmierung von eigenen Anwendungen zu erleichtern, wird an dieser Stelle ein Überblick über die Nutzung der innerhalb dieses Bachelorprojektes implementierten Funktionen gegeben. Dieser Überblick umfasst jedoch lediglich die Einbindung und Handhabung der im vorigen Kapitel vorgestellten Bibliotheken. Die für die Nutzung des Build-Systems von Contiki vorausgesetzten Grundkenntnisse können in der von mir erarbeiteten Dokumentation der Projektarbeit zum Thema "Untersuchung eines Mikroprozessor-Betriebssystems mit 6LoWPAN-Stack in Hinblick auf die Portierung auf neue Prozessoren" [12] nachgeschlagen werden.

5.1 Konfiguration und Nutzung der IPv6-Funktionen (6LoWPAN)

Das Betriebssystem Contiki 2.5 hat im Gegensatz zu der vorherigen Version 2.4 einige Erweiterungen erfahren, die bequem innerhalb der Makefiles über Compilerflags aktiviert werden können. Hierzu zählen zum Beispiel die Nutzung der automatischen Adressvergabe, sowie die Möglichkeit auf einzelnen Netzwerkknoten Funktionen als Service einzurichten und diese Services zu registrieren und somit im gesamten drahtlosen 6LoWPAN-Netzwerk abrufbar zu machen. Zudem wurde in Contiki 2.5 das RPL[1]-Protokoll implementiert, das ein speziell für drahtlose Sensornetzwerke konzipiertes Routing- und Forwardingprotokoll darstellt. Das RPL-Protokoll dient dazu, Routinginformationen über das Netzwerk auszutauschen und somit die Wahl eines geeigneten Weges der Pakete durch das Netzwerk zu automatisieren. Somit ermöglicht die Nutzung des RPL-Protokolls die Weiterleitung von Paketen über mehrere Netzwerkknoten, falls die Sendeleistung eines Gerätes nicht ausreicht, um das Ziel des Kommunikationspaketes direkt zu erreichen.

Das auf Seite 41 abgebildete *Listing 5.1* zeigt die wichtigsten im Kontext von 6LoWPAN zu nutzenden Konfigurationsmöglichkeiten und weitere, für die Kompilierung eines Contiki-Programmes benötigte Einstellungen.

[1] Routing Protocol for Low power and Lossy Networks

```
1  #Zu compilierende Dateien (ohne ".c")
2  all: testprogramm
3
4  #Einbindung von externen Applikationen
5  #(hier die Nutzung der Service-Registry)
6  APPS=servreg-hack
7
8  #Angabe des Contiki-Stamverzeichnisses
9  CONTIKI=../..
10
11 #Aktivierung des microIP-Stacks
12 WITH_UIP6=1
13
14 #Nutzung des IPv6-Protokolls
15 UIP_CONF_IPV6=1
16
17 #Nutzung der automatischen Adressvergabe
18 CONTIKI_CONF_RANDOM_MAC = 1
19
20 #Nutzung des RPL-Protokolls
21 CFLAGS+= -DUIP_CONF_IPV6_RPL
22
23 #Angabe des Haupt-Makefiles von Contiki
24 include $(CONTIKI)/Makefile.include
```

Listing 5.1: 6LoWPAN-Konfiguration mit Hilfe des Makefiles

5.2 Nutzung der Stromsparfunktionen

Die innerhalb dieses Bachelorprojektes implementierten Stromsparfunktionen werden für die RAVEN-Hardware und für den Mikrocontroller ATmega128RFA1 gleichermaßen über folgendes, innerhalb des Makefiles anzugebendes Compilerflag in die eigene Applikation eingebunden:

```
CFLAGS+= -DUSE_SLEEP
```

Die bei der Implementierung von Herrn Esders nötige Angabe der Headerdatei der Sleep-Bibliothek und Einfügung der Datei app-conf.h entfällt hierdurch. Allerdings ist zu beachten, dass die Nutzung der Bibliothek zwingend eine Initialisierung der Sleep-Funktionen notwendig macht. In *Listing 5.2* sind der Befehl für die Initialisierung

der Stromsparfunktionen und Funktionsaufrufe für die Freigabe und Beendigung des Schlafmodus ersichtlich.

```
1  [...]
2
3  //Beginn der Prozess-Ausührung
4  PROCESS_BEGIN();
5
6  // Sleep-Funktionen initialisieren
7  SLEEP_INIT();
8
9  // Sleep-Modus verbieten
10 //Verbietet Schlafen des Transceivers
11 RF_SLEEP_DISALLOW();
12 //Verbietet Schlafen der CPU
13 CPU_SLEEP_DISALLOW();
14
15 [...]
16
17 //Sleep-Modus erlauben
18 //Erlaubt Schlafen des Transceivers
19 RF_SLEEP_ALLOW();
20 //Erlaubt Schlafen der CPU
21 CPU_SLEEP_ALLOW();
22
23 [...]
```

Listing 5.2: Nutzung der Funktionen der Sleep-Bibliothek

5.3 Nutzung der Verschlüsselungsfunktionen

Analog zu der Nutzung der Stromsparfunktionen geschieht die Einbindung der Bibliothek für die Verwendung der Funktionen zur Hardwareverschlüsselung ebenfalls über die Angabe eines Compilerflags innerhalb des Makefiles der Anwendung:

```
CFLAGS+= -DUSE_HWAES
```

Nach Setzen dieses Compilerflags kann auch diese Bibliothek ohne Einbindung weiterer Headerdateien verwendet werden. Die eigentliche Ver- und Entschlüsselung der Daten erfolgt über den Aufruf der entsprechenden C-Funktionen. Auf Seite 43 sind

in *Listing 5.3* Ausschnitte des in *Kapitel 4.1.1* vorgestellten Testprogrammes für die Hardwareverschlüsselung abgebildet, welche die Nutzung der Funktionen verdeutlichen.

```
1  [...]
2
3  /* Max. Größe der Strings definieren */
4  #define MAX_AES_LENGTH 96
5
6  [...]
7
8  PROCESS_THREAD(cipher_example_process, ev, data)
9  {
10   /* Variablen für die Speicherung der Strings deklarieren */
11   static char string[MAX_AES_LENGTH + 1] = "";
12   static char cipherstring[MAX_AES_LENGTH + 1] = "";
13   static char plainstring[MAX_AES_LENGTH + 1] = "";
14
15   /* AES-Schlüssel festlegen */
16   static char key[16] = "abcdefghijklmnop";
17
18   PROCESS_BEGIN()
19
20   [...]
21
22   /* Plaintext in String kopieren */
23   strcpy(string, "Teststring");
24
25   /* String verschlüsseln
26      -> Chiffretext wird in cipherstring kopiert */
27   strcpy(cipherstring, encrypt(string, strlen(string), key));
28
29
30   /* Chiffretext entschlüsseln
31      -> Entschlüsselter Klartext wird in plainstring kopiert
32      ATTENTION: Do not use "strlen()" */
33   strcpy(plainstring, decrypt(cipherstring, 16, key));
34
35 [...]
36
37 }
```

Listing 5.3: Nutzung der AES-Funktionen

Bei der Nutzung der Entschlüsselungsfunktion decrypt() ist darauf zu achten, dass die Angabe Größe des Chiffretextes nicht automatisch mit Hilfe der in der Bibliothek string.h enthaltenen Funktion strlen() erfolgen kann, da dies unter Umständen zu Fehlern bei der Entschlüsselung führt. Hintergrund ist, dass bei der Verschlüsselung durch die Nutzung der Substitutionsbox für einzelne Bytes auch der hexadezimale Wert "00" auftreten kann, der von der Funktion strlen() dann fälschlicherweise als Ende des Strings interpretiert würde. Zudem ist durch die von dem AES-Algorithmus genutzte Blockverschlüsselung für den Chiffretext grundsätzlich eine Länge vorgegeben, die einem Vielfachen von 16 Byte entspricht.

6 Aufgetretene Probleme und deren Lösung

Innerhalb dieses Kapitels werden die Probleme, die innerhalb dieses Bachelorprojektes bezüglich der Implementierung der Software auftraten noch einmal zusammengefasst und mögliche Lösungen, beziehungsweise Lösungsansätze beschrieben. Dies dient in erster Linie dazu, die notwendige Zeit für die Bearbeitung ähnlicher Projekte zu verkürzen, da die Fehlersuche entfallen kann oder zumindest vereinfacht wird.

6.1 Nutzung von Strings

Bei der Nutzung von Strings (Char-Arrays) sollten die Strings immer durch die Einfügung von '\0' am Ende des Strings manuell abgeschlossen werden, da es ansonsten zu unerwarteten Verhalten von String-Funktionen, wie zum Beispiel strcmp() kommen kann. Da diese Fehler bei der Verwendung der Entwicklungssoftware AVR Studio in Verbindung mit einem für die Mikroprozessoren des Herstellers Atmel optimierten Compiler nicht auftraten, ist zu überprüfen, ob dieses Verhalten von der Nutzung gestimmter Compiler abhängig ist, oder unter Umständen auf die Speicherverwaltung des Contiki-Betriebssystems zurückgeführt werden kann.

6.2 Nutzung von Endlosschleifen (unerwartete Hardware-Resets)

Bei der Implementierung von Programmen innerhalb des Contiki-Betriebssystem kam es bei der Ausführung der Programme mehrfach zu der Auslösung von unerwarteten Hardware-Resets. Diese konnten auf die Nutzung von Endlosschleifen innerhalb der implementierten Contiki-Programme zurückgeführt werden, innerhalb derer kein Prozesswechsel statt findet. Hierdurch erkennt das Contiki-Betriebssystem einen Deadlock und setzt den Mikrocontroller zurück. Als Beispiel sei in *Listing 6.1* auf Seite 47 eine Schleife abgebildet, die zu diesem Verhalten führt.

```
1 [...]
2
3 while(1)
4 {
5   if(!strcmp(plainstring, "Teststring"))
6     PORTE &= ~(1<<PE2);
7 }
8
9 [...]
```

Listing 6.1: "Fehlerhafte" Schleife (Erkennung eines Deadlocks)

Abhilfe zu dem innerhalb des Betriebssystem festgelegten Verhalten wird durch die Erzwingung eines Prozesswechsels innerhalb der Endlosschleife geschaffen. Dies kann zum Beispiel durch die Nutzung eines Timers oder anderer Contiki-Prozesse innerhalb der Schleife geschehen. *Listing 6.2* zeigt die Behebung des Fehlers durch Nutzung einer Timerfunktion. Je nach Anwendung sollte die Zeitspanne für die Auslösung des Timers jedoch verkürzt werden.

```
1 [...]
2
3 //set timer
4 etimer_set(&anti_deadlock_timer, 1 * CLOCK_SECOND);
5
6 while(1)
7 {
8   if(!strcmp(plainstring, "Teststring"))
9     PORTE &= ~(1<<PE2);
10
11   //wait for timer event and reset timer
12   PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&anti_deadlock_timer));
13   etimer_reset(&anti_deadlock_timer);
14 }
15
16 [...]
```

Listing 6.2: Verhinderung der Deadlock-Erkennung

6.3 Verarbeitung des letzten Rundenschlüssels (Last Roundkey)

Bei der Programmierung des Testprogrammes für die Hardwareverschlüsselung ist anfänglich so vorgegangen worden, dass innerhalb der Entschlüsselungsfunktion zunächst eine Verschlüsselung eines zufällig ausgewählten Textes mit dem Chiffreschlüssel durchgeführt wurde, um den letzten Rundenschlüssel zu erzeugen. Bei dem direkt darauf folgenden Entschlüsselungsvorgang sollte der nun innerhalb des Register AES_KEY liegende letzte Rundenschlüssel für die Erzeugung des Klartextes genutzt werden. Wurde dieser jedoch nicht explizit aus dem Puffer gelesen, zwischengespeichert und hierauf wieder zurück in das Register AES_KEY geschrieben, führte dies zu einem fehlerhaften Verhalten bei dem Entschlüsselungsvorgang. Die eigentliche Problematik bei der Auffindung des Fehler lag darin, dass das Register AES_KEY nicht mit Hilfe des innerhalb der Software AVR Studio integrierten Debuggers ausgelesen werden konnte und dadurch die Fehlersuche erschwert wurde. Letztendlich hängt der Fehler mit der Arbeitsweise des Verschlüsselungsmoduls des ATmega128RFA1 und dessen Register zusammen. *Abbildung 6.1* zeigt die korrekte Vorgehensweise für die Erzeugung und Verarbeitung des letzten Rundenschlüssels.

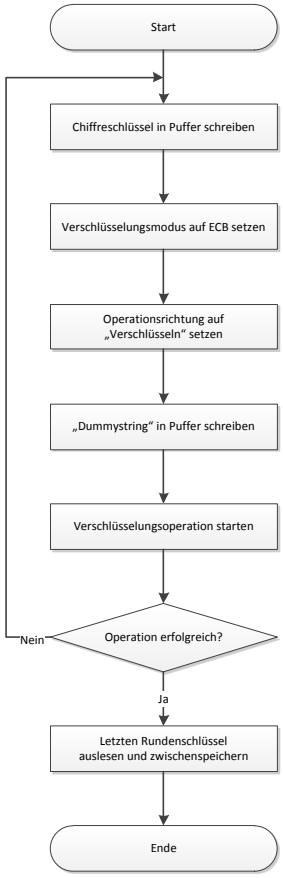

Abb. 6.1: Erzeugung des letzten Rundenschlüssels

6.4 Größe der genutzten Variablen

Da durch das Contiki-Betriebssystem bereits ein erheblicher Teil des verfügbaren Datenspeichers des Mikrocontrollers genutzt wird, ist ein effizienter Umgang mit Variablen unabdingbar. Dies fiel während der Arbeit an diesem Bachelorprojekt auf, als Programme, die ohne die Nutzung des Betriebssystems reibungslos funktionierten, bei der Verwendung des Contiki-Betriebssystems nicht die erwarteten Ergebnisse generierten. Bei der Analyse

der Programme stellte sich heraus, dass der allozierte Speicherplatz für die Daten den verfügbaren Speicherplatz innerhalb des SRAMs des ATmega128RFA1 von 16 kByte überstieg. Abhilfe kann durch die Nutzung von effizienten Ersetzungsalgorithmen und die Verwendung von dynamischer Speicherallokation geschaffen werden.

6.5 Ermittlung der Größe des Chiffretextes

Die Ermittlung der Größe des Chiffretextes mit Hilfe der Funktion strlen() führte bei der Nutzung der Entschlüsselungsfunktion zu falschen Ergebnissen. Grund hierfür ist, dass die Funktion strlen() einen String auf das Zeichen '\0' durchsucht, welches das Ende eines Strings signalisiert. Bei der Verschlüsselung kann es jedoch dazu kommen, dass dieses Zeichen innerhalb des Chiffretextes das verschlüsselte Pendant zu einem Klartextzeichen darstellt und somit nicht mehr für die Erkennung des Endes eines Strings genutzt werden kann. Innerhalb dieses Bachelorprojektes wurde daher dazu übergegangen, die Größe des Chiffretextes an die verarbeitende Funktion als Zahlenwert zu übergeben. Da innerhalb des WieDAS-Projektes verschlüsselte Daten lediglich innerhalb der von der Funkschnittstelle empfangenen Pakete vorliegen, ist zu überlegen, ob die Größe des Chiffretext direkt über die Nutzung von Socketfunktionen, die den Nutzdatenanteil bestimmen, angegeben werden kann.

6.6 Unerwartetes Verhalten von Ausgabeports

Die Verwendung des mit Leuchtdioden beschalteten Ports PORTE des ATmega128RFA1 Evaluation-Kits bereitete teilweise in der Art Probleme, dass sich nach der Konfiguration als Ausgabeport die Registerwerte für einzelne Pins änderten, obwohl diese nicht innerhalb des implementierten Programmes angesprochen wurden. Abhilfe konnte hier durch die Deklaration der genutzten Variablen als static geschaffen werden. Der Zusammenhang zwischen der nicht statischen Verwendung der Variablen und der Veränderung der Registerwerte für PORTE konnte jedoch innerhalb der für die Durchführung des Bachelorprojektes vorgegeben Zeitspanne nicht näher untersucht werden.

6.7 Einbindung der Headerdatei “radio.h”

Innerhalb der Headerdatei /cpu/avr/rf230/radio.h und der zugehörigen Datei /cpu/avr/radio/rf230/radio.c sind Definitionen und Funktionen für die Steuerung der Transceiver-

Einheit des RAVEN-Systems hinterlegt, die auch von den innerhalb dieses Bachelorprojektes portierten und verbesserten Funktionen für die Nutzung der Schlaffunktionen der RAVEN-Plattform benötigt werden. Allerdings wurden diese Dateien bei der Kompilierung trotz Auswahl des Transceiver-Treibers "RF230" nicht mit in das Betriebssystem eingebunden. Dies führte zu Fehlern, da bei der Übersetzung des Quellcodes verschiedene benötigte Funktionen nicht gefunden werden können. Nach Untersuchung des Build-Systems von Contiki stellte sich heraus, dass im Makefile für die Transceiver-Einheit ein Eintrag vorhanden ist, der unabhängig von der eigenen Konfiguration dafür sorgt, dass bei jeder Kompilierung statt des Treibers RF230 der weiter entwickelte Treiber RF230BB aktiviert wird. Durch Auskommentieren der entsprechenden Anweisung innerhalb des Makefiles /cpu/avr/radio/Makefile.radio durch

```
RF230BB=1
```

kann dies verhindert werden. Allerdings führt diese Vorgehensweise zu neuen Fehlern bei der Kompilierung, da das Betriebssystem Contiki in der neuesten Version anscheinend auf die Nutzung des RF230BB-Treibers angewiesen ist. Um die Stromsparfunktionen mit dem neuen Treiber nutzen zu können, wurden Teile des alten Treibers manuell in diesen überführt. *Listing 9.3* zeigt die Portierung der benötigten Funktionen in die Datei /platform/avr-raven/contiki-raven-main.c.

7 Gerätetests und Vergleich mit der zuvor genutzten Hardware

Um die innerhalb dieses Bachelorprojektes durchgeführten Optimierungen zu verifizieren, wurden Hard- und Software verschiedenen Tests unterzogen. Die Ergebnisse dieser Tests werden innerhalb dieses Kapitels zusammengefasst.

7.1 Energieverbrauch

Die Ermittlung des tatsächlichen Energieverbrauchs des zuvor genutzten und des neu entwickelten, auf dem Mikrocontroller ATmega128RFA1 basierenden, Funkmoduls gestaltete sich als schwierig, weil beide Systeme nur mit großem Aufwand unabhängig von Entwicklungsplatinen getestet werden können. Da die Entwicklungsplatinen jedoch selber eine beträchtliche Anzahl an elektronischen Bauteilen enthalten, wirkt sich schon alleine die Nutzung der Entwicklungsplatinen massiv auf den gemessenen Energiebedarf aus. Mehrere Versuche, dies zu umgehen, führten nicht zu einer genaueren Messung. Daher wird an dieser Stelle zunächst theoretisch, über die Ermittlung der benötigten Zeitspanne für die Versendung eines Datenpakets und den Angaben aus den Datenblättern [1, 7, 2] der Mikrocontroller auf die Reduzierung des Stromverbrauchs bei der Verwendung des ATmega128RFA1 geschlossen. Als Zeitspanne wird hier die Zeit angesehen, in denen sich die Transceiver-Einheit im Zustand BUSY_TX (Senden) befindet und somit kein Schlafmodus genutzt werden kann. Diese wurde mit Hilfe eines Computer-Oszilloskops, dem PicoScope 5204 des Herstellers pico Technology gemessen. *Tabelle 7.1* auf Seite 51 zeigt die gemessenen Werte für unterschiedliche Paketgrößen. *Abbildung A.1* und *Abbildung A.2* im Anhang dieser Bachelorthesis verdeutlichen die durchgeführten Messungen anhand der Ausgaben des Oszilloskops. Die Beispiele beziehen sich auf eine Nutzdatengröße von 50 Byte. Da die Flanken innerhalb der Graphen nicht ideal sind, wurden als Messzeitpunkte der Beginn der steigenden und der Beginn der fallenden Flanke gewählt.

	Größe der Nutzdaten in Byte				
	10	50	100	500	1000
ATmega128RFA1	1,730	3,110	5,961	23,830	46,240
RAVEN-Kit	2,498	3,790	8,172	27,650	52,900

Angabe der Zeitspannen in ms

Tab. 7.1: Dauer der Sendeimpulse in Abhängigkeit von der Paketgröße

Den Datenblättern der Controller [1, 7, 2] wurden folgende, in *Tabelle 7.2* zusammengefasste Daten, entnommen. Die Daten beziehen sich hierbei auf einen Systemtakt von 8 MHz. Zu beachten ist jedoch, dass das RAVEN-Kit für einen Systemtakt von 8 MHz eine Betriebsspannung von mindestens 5,0 V benötigt, während der Mikrocontroller ATmega128RFA1 bereits bei einer Eingangsspannung von 3,0 V mit 8 MHz Systemtakt betrieben werden kann.

	Stromaufnahme in mA					
	CPU_SLEEP	TRX_SLEEP	gesamt	CPU_IDLE	BUSY_TX	gesamt
ATmega128RFA1	0,00025	0,00002	0,00027	1,10000	14,50000	15,60000
RAVEN-Kit	0,00060	0,00002	0,00062	1,10000	16,50000	17,60000

Tab. 7.2: Stromaufnahmen des ATmega128RFA1 und des RAVEN-Kits (vgl. [1, 2])

Für die konkrete Berechnung der Leistungsaufnahme wird nun davon ausgegangen, dass ein innerhalb des WieDAS-Projektes genutzter Sensor, wie zum Beispiel der Wassermelder, im Normalfall nach einem Intervall von fünf Minuten eine Statusmeldung versendet und diese lediglich aus zehn Zeichen besteht. Die Berechnung der durchschnittlichen Leistungsaufnahme führt zu folgenden Ergebnissen.

Formel:

$$P = ((t_{sleep} - t_{alive}) \cdot i_{sleep} + t_{alive} \cdot i_{alive}) \cdot U \cdot \frac{1}{t_{ges}}$$

Für das RAVEN-Kit:

$$P = ((300000\,ms - 2,498\,ms) \cdot 0,00062\,mA + 2,498\,ms \cdot 17,6\,mA) \cdot 5,0\,V \cdot \frac{1}{300000\,ms}$$

$$P \approx 0,00383\,mW$$

Für den ATmega128RFA1-Mikrocontroller:

$P = ((300000\,ms - 1,730\,ms) \cdot 0,00027\,mA + 1,730\,ms \cdot 15,6\,mA) \cdot 3,0\,V \cdot \frac{1}{300000\,ms}$

$P \approx 0,00108\,mW$

Die Ergebnisse zeigen, dass die Leistungsaufnahme des RAVEN-Kits bei diesem Szenario rechnerisch dreimal so hoch ist wie die des ATmega128RFA1-Mikrocontrollers. Allerdings müssen diese Werte relativiert werden, da die untersuchten Systeme für den Betrieb als Sensorknoten in weitere Hardware eingebettet werden müssen, die ein Vielfaches der hier errechneten Leistung verbrauchen. Um das durch die Nutzung des ATmega128RFA1 erreichbare Einsparpotenzial bezüglich des Energieverbrauchs für reale Fälle abschätzen zu können, müssten beide Systeme innerhalb eines identischen Sensorknotens verbaut und anhand dieser Geräte der Leistungsbedarf bestimmt werden.

7.2 Funkanbindung

7.2.1 Paketverluste

Für die Berechnung der Paketverluste wurde ein Testprogramm entwickelt, mit dessen Hilfe Datenpakete verschiedener Größe an den als Empfänger dienenden RAVEN-Stick geschickt wurden. Hierbei wird die Anzahl der versendeten Pakete gezählt. Durch den Vergleich mit der Anzahl der am RAVEN-Stick angekommenen Pakete kann die Paketverlustrate berechnet werden. Die Messungen fanden im Labor für Informatik an der Fachhochschule Düsseldorf statt. Die Entfernung zwischen Sender und Empfänger betrug 1,0 m. *Tabelle 7.3* auf Seite 53 zeigt die aufgetretenen Paketverluste bei der Nutzung des RAVEN-Kits und des ATmega128RFA1. Um die Messwerte aussagekräftiger zu machen, wurden für jede Paketgröße drei Messungen durchgeführt, bei denen jeweils 10.000 Datenpakete direkt aufeinander folgend versendet und die Messergebnisse arithmetisch gemittelt und auf einen ganzzahligen Wert gerundet wurden. Wie die Messdaten zeigen, hat die Realisierung des Funkmoduls unter Nutzung des neuen Antennendesigns nicht zu einer Verringerung der Verlustrate geführt. Zudem ist zu erkennen, dass die Verlustrate, zumindest ab einer Größe des Datenpaketes von 200 Byte, nicht in Relation zu der Paketgröße steht. Da das Ergebnis der Messungen anders als erwartet ausfiel, wurde die Paketverlustrate noch einmal unter Verwendung des originalen Evaluation Boards ATmega128RFA-EK1 des Herstellers gemessen. Bei diesen Messungen stellte sich heraus, dass auch die Verwendung dieser Hardware keine Verbesserung der Fehlerrate zur Folge hatte. Hier lagen die Werte für die Anzahl der korrekt übertragenen Datenpakete sogar

teilweise deutlich unter denen, der im Labor für Informatik an Fachhochschule Düsseldorf hergestellten Hardware. Bei einer Nutzdatengröße von 1000 Byte und einem von 5,0 m wurden beispielsweise lediglich 2837 der gesendeten 10.000 Pakete empfangen. Dies entspricht einer Paketfehlerrate von 71,63 %. Die Auswertung der Messungen lässt darauf schließen, dass die Ursache für die fehlerhafte Datenkommunikation gegebenenfalls in der Empfänger-Hardware (RAVEN USB-Stick) zu suchen ist. Weitere Tests hierzu konnten in der für die Bearbeitung des Bachelorprojektes zu Verfügung stehenden Zeit jedoch nicht durchgeführt werden.

Paketgröße (in Byte)	RAVEN-Kit			ATmega128RFA1		
	empfangen	verloren	proz. Anteil	empfangen	verloren	proz. Anteil
50	9996	4	0,04	10000	0	0,00
100	9264	736	7,36	9863	137	1,37
200	8445	1555	15,55	8464	1536	15,36
300	7242	2758	27,58	8779	1221	12,21
400	8432	1568	15,68	7461	2539	25,39
500	7905	2095	20,95	8078	1922	19,22
600	9345	655	6,55	8375	1625	16,25
700	8793	1207	12,07	6666	3334	33,34
800	7516	2484	24,84	8773	1227	12,27
900	8346	1654	16,54	7155	2845	28,45
1000	8516	1484	14,84	8760	1240	12,40
1232	7312	2688	26,88	7505	2495	24,95

Tab. 7.3: Gemessene Paketverluste

7.2.2 Datendurchsatz

Da schon die Messungen zum Energieverbrauch der RAVEN-Hardware und des Mikrocontrollers ATmega128RFA1 zeigten, dass die Nutzung der internen Transceiver-Einheit, im Gegensatz zu der Verwendung des externen Transceiver-Chips, eine höhere Verarbeitungsgeschwindigkeit ermöglicht, wurde angenommen, dass hierdurch auch die resultierende Datenübertragungsgeschwindigkeit erhöht wird. Um diese Annahme zu verifizieren, wurden Vergleichsmessungen zu der möglichen Datenübertragungsgeschwindigkeit durchgeführt. Hierzu wurden für beide Funkmodule Implementierungen zum Versenden von IPv6-Broadcast-Paketen erstellt, welche die Funkmodule dazu veranlassen, die Pakete mit der maximal möglichen Wiederholrate zu senden. Um die Paketverlustrate

möglichst gering zu halten, wurde für diese Tests des Datendurchsatzes der Abstand zwischen Sende- und Empfangsmodul auf 10 cm reduziert. Bei dieser Entfernung traten keinerlei Paketverluste auf.

Abb. 7.1: Vergleich der Datenübertragungsraten (brutto)

In *Abbildung 7.1* sind die gemessenen Datenübertragungsraten (brutto) in Abhängigkeit von der Größe der Nutzdaten ersichtlich, wobei mehrere Punkte auffallend sind:

1. Der Datendurchsatz des ATmega128RFA1 ist bis zu 50% größer als der des RAVEN Boards.
2. Der Datendurchsatz des ATmega128RFA1 erreicht fast den für die Kommunikation über den Standard IEEE 802.15.4 vorgesehenen maximalen Durchsatz von 250 kBit/s.
3. Die Datenübertragungsrate beider Systeme erleidet bei der Übertragung einer Nutzdatengröße von 720 Bit (90 Byte) einen starken Einbruch. Dies deutet darauf hin, dass nach dem Hinzufügen der Verwaltungsdaten die maximale, innerhalb des Standards IEEE 802.15.4 definierte Framegröße von 1016 Bits (127 Byte) überschritten wird und der Datendurchsatz durch die dadurch notwendige Segmentierung verschlechtert wird.

Weitere Details sind dem Messprotokoll zu entnehmen, welches im *Anhang A.2.2* als *Tabellen A.1* eingefügt wurden.

7.2.3 Abgegebene Leistung

Da innerhalb des Labors für Informatik an der Fachhochschule Düsseldorf keine geeigneten Gerätschaften für die Messung der abgegebenen Leistung der Antenne des Sendemoduls zu Verfügung steht, war geplant, diese Messung im Labor für Hochfrequenztechnik durchzuführen. Auf Grund von terminlichen Schwierigkeiten konnte dies jedoch bis zum Ende der Bearbeitungszeit für dieses Bachelorprojekt nicht geschehen. Daher kann an dieser Stelle lediglich darauf hingewiesen werden, dass auf dem neuen Sendemodul eine besser an den Mikrocontroller angepasste Antenne implementiert wurde. Zudem erreicht die Transceiver-Einheit des ATmega128RFA1 eine geringfügig höhere Ausgangsleistung von 3,5 dBm gegenüber der Ausgangsleistung des AT86RF230 Transceiver-Chips von 3,0 dBm. Dies entspricht in absoluten Werten einer Differenz von 0,24 mW, beziehungsweise einer Leistungssteigerung von 12 %. Unter Berücksichtigung dieser beiden Merkmale kann darauf geschlossen werden, dass die abgegebene Leistung des neuen Funkmoduls insgesamt deutlich über der des bisher genutzten liegt.

7.3 Verschlüsselung

7.3.1 Vergleich der Performance von Soft- und Hardwareverschlüsselung

Da die genauen internen Abläufe des ATmega128RFA1 bei der Nutzung des Verschlüsselungsmoduls nicht bekannt sind, muss über die im Datenblatt des Controllers [1] angegebene Zeitspanne für die Ausführung der AES-Verschlüsselung die benötigte Anzahl an Taktzyklen berechnet werden:

$n_{ck} = t_{crypt} \cdot ck \cdot \frac{1}{s} = 24\,s \cdot 10^{-6} \cdot 16 \cdot \frac{1}{10^{-6}\,s} = 384$

Diese mit 384 Taktzyklen berechnete Zeitspanne für die Durchführung eines Verschlüsselungsvorgangs kann nun mit der Zeitspanne für die Verschlüsselung auf Basis des Softwarealgorithmus der Bibliothek “AVR-Crypto-Lib” [13] verglichen werden:

- Hardwareverschlüsselung: 384 Taktzyklen
- Softwareverschlüsselung
  - ASM: 2555 Taktzyklen
  - AVR-C: 21279 Taktzyklen

Der Vergleich zeigt, dass die Hardwareverschlüsselung einen bedeutenden Vorteil zu der softwaretechnischen Implementierung der Verschlüsselung bietet. Die Hardwareverschlüsselung ist etwa um den Faktor 6,65 schneller als die Implementierung der Verschlüsselung in Assembler-Code.

7.3.2 Auswirkung der Verschlüsselung auf den Datendurchsatz

Trotz der innerhalb des vorigen Kapitels beschriebenen Leistungsfähigkeit der Hardwareverschlüsselung, führt die Nutzung der verschlüsselten Datenübertragung unweigerlich zu einer Verminderung des Datendurchsatzes. Diese ist in *Abbildung 7.2* dargestellt. Die wellenartige Form des Grafen lässt sich dadurch erklären, dass bei der Verschlüsselung je nach Größe der zu übertragenden Nutzdaten ein unterschiedlich großer Anteil an Overhead hinzugefügt werden muss, um die Blockverschlüsselung durchführen zu können. Nähert sich die Größe der Nutzdaten einem vielfachen von 16 Byte, so wird dieser Overhead kleiner, wodurch die Performance steigt. Zu untersuchen bleibt hierbei jedoch der starke Einbruch der Datenübertragungsrate bei einem Nutzdatenanteil von 90 Byte, für den bisher keine Erklärung gefunden wurde.

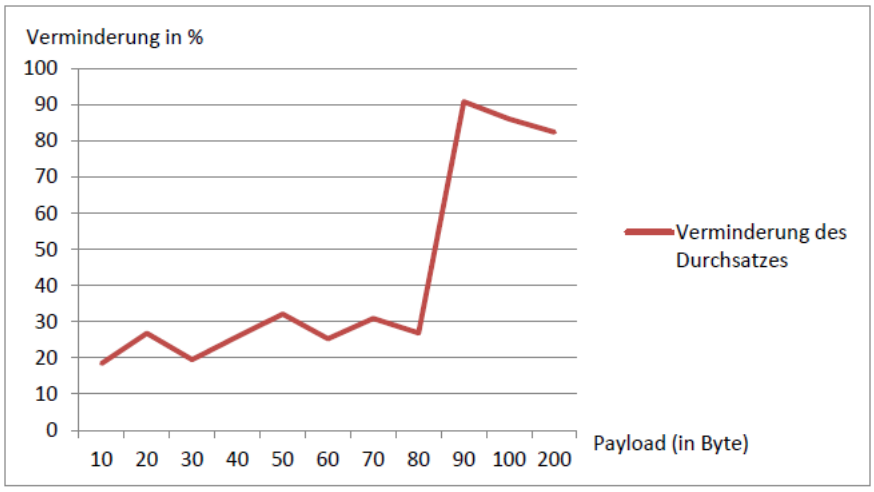

Abb. 7.2: Auswirkung der Verschlüsselung auf den Datendurchsatz

8 Fazit und Ausblick

Innerhalb dieses Bachelorprojektes wurden verschiedene Optimierungsmöglichkeiten zu der Kommunikation über das 6LoWPAN-Protokoll, im Kontext von Ambient Assisted Living, beleuchtet. Hauptziele waren hierbei die Steigerung der Energieeffizienz, die Sicherstellung einer verlässlichen Übertragung bezüglich auftretender Paketverluste, die Erhöhung der Datenübertragungsrate und die Gewährleistung einer sicheren Datenübertragung durch die Verschlüsselung der Datenpakete.

Hierzu wurde zunächst die erforderliche Hardware, die aus einem neuen Funkmodul, basierend auf dem Mikrocontroller ATmega128RFA1 und der passenden Entwicklungsplatine besteht, realisiert. Im Gegensatz zu dem zuvor genutzten RAVEN-System bietet das neue Funkmodul aus hardwaretechnischer Sicht folgende Vorteile:

- Nutzung einer an die Transceiver-Einheit angepassten Antenne
- Geringerer Energiebedarf
- Höhere Datenübertragungsrate
- Größere Performance der CPU durch schnelleren Systemtakt
- Möglichkeit der AES-Hardwareverschlüsselung

Die Voraussetzung für die Verwertung dieser Vorteile der neuen Hardware sind jedoch die softwaretechnischen Implementierungen für den Mikrocontroller. Daher bestand der zweite Schwerpunkt dieser Projektarbeit, neben der Realisierung der Hardware, in der Programmierung von geeigneten Softwarefunktionen für den Mikrocontroller ATmega128RFA1. Diese betrafen neben Anpassungen des Mikrocontroller-Betriebssytems Contiki 2.5 für die Nutzung der Stromsparfunktionen, Implementierungen für die Verwendung des Verschlüsselungsmoduls des ATmega128RFA1.

Um die Erreichung der oben genannten Ziele zu verifizieren, wurden zum Ende dieses Bachelorprojektes verschiedene Gerätetests durchgeführt, deren Ergebnisse wie folgt zusammengefasst werden können:

- Die Nutzung der neuen Hardware führt in Verbindung mit den implementierten Stromsparfunktionen zu einer Verringerung des Energiebedarfs.
- Das neue Funkmodul ist bezüglich der Datenübertragungsrate performanter als das zuvor genutzte RAVEN-System.
- Über die einfach zu nutzenden Verschlüsselungsfunktionen kann eine sichere, verschlüsselte Datenübertragung innerhalb des AAL-Systems aufgebaut werden.

Allerdings soll an dieser Stelle nicht verschwiegen werden, dass ein Hauptziel dieser Projektarbeit nicht erreicht wurde. So konnte auch durch die Verbesserung bezüglich der Anpassung der Antenne an die Transceiver-Einheit kein Fortschritt bezüglich der, von der Entfernung der Knoten abhängigen, auftretenden Paketverluste erzielt werden. Aufbauend auf dieser Arbeit müsste daher untersucht werden, in wie weit möglicherweise die aus dem RAVEN USB-Stick bestehende Empfangseinheit für das Auftreten der Paketverluste verantwortlich sein könnte. Hierfür wäre zunächst eine Messung der abgestrahlten Leistung der Antenne durchzuführen, um die Güte des neuen Funkmoduls zu ermitteln. Sollte diese Messung befriedigende Ergebnisse liefern, wäre der nächste Schritt die Realisierung eines Ersatzes für den RAVEN USB-Stick als Empfangseinheit.

Weitere Optimierungsmöglichkeiten bestehen zudem in der Realisierung von verbindungsorientierten Transportfunktionen und in der Nutzung des RPL-Protokolls zur Erhöhung der Reichweite der einzelnen Sensorknoten. Zudem betrifft eine andere Überlegung die Erhöhung des Datendurchsatzes. Momentan wird das Transceiver-Modul lediglich mit der geringsten der über die Hardware ermöglichten Übertragungsraten von 250 kBit/s betrieben. Diese könnte hardwaretechnisch ohne weiteres auf bis zu 2 MBit/s erhöht werden. Allerdings wäre die Übertragung dann nicht mehr konform zu dem Standard IEEE 802.15.4, auf dem das 6LoWPAN-Protokoll aufbaut. Daher müsste zunächst untersucht werden, in wieweit sich die Erhöhung der Übertragungsrate auf die Kommunikation über das 6LoWPAN-Protokoll auswirkt.

Abschließend kann gesagt werden, dass innerhalb dieses Projektes einige Aspekte bezüglich der Optimierung der Kommunikation über das 6LoWPAN-Protokoll erarbeitet und realisiert werden konnten. Auf Grund der endlichen zur Verfügung stehenden Zeit für die Bearbeitung des Projektes, besteht jedoch weiterhin die Notwendigkeit, verschiedene Teilaspekte der Kommunikation im Rahmen von hierauf aufbauenden Projekten zu untersuchen.

A Anhang

A.1 Listings

A.1.1 Funktionen für die Nutzung des AES-Moduls des ATmega128RFA1

```
1  //inclusion of aes header file
2  #ifndef __AES_h__
3  #include "aes.h"
4  #endif
5
6  //encrypt string with MAX_LENGTH bytes
7  /*-------------------------------------------------------------*/
8  char* encrypt(char *plainstring, int length, char key[16])
9  {
10   //declaration of variables
11   static char cipherstring[MAX_AES_LENGTH + 1] = "";
12   static char plainblock[17] = "";
13   uint8_t i, j, rest;
14   uint8_t expand = 0;
15
16   //stringlength multiple of 16?
17   rest = length % 16;
18   if (rest != 0)
19     expand = 1;
20   else
21     expand = 0;
22
23   //split, expand, encrypt and reassemble
24   j = 0;
25   for (i = 0; i < length + ((16 - rest) * expand); i++)
26   {
27     if (i >=  length)
28       plainblock[i - (16 * j)] = '\0';
```

```
29      else
30        plainblock[i - (16 * j)] = plainstring[i];
31      if ((i + 1) % 16 == 0)
32      {
33        plainblock[16] = '\0';
34        strcat(cipherstring, encrypt16(plainblock, key));
35        j++;
36      }
37    }
38
39    //set end of cipherstring
40    cipherstring[length + ((16 - rest) * expand)] = '\0';
41
42    return cipherstring;
43  }
44  /*---------------------------------------------------------------*/
45
46  //decrypt string with MAX_LENGTH bytes
47  /*---------------------------------------------------------------*/
48  char* decrypt(char *cipherstring, int length, char key[16])
49  {
50    //declaration of variables
51    static char plainstring[MAX_AES_LENGTH + 1] = "";
52    static char cipherblock[17] = "";
53    uint8_t i, j;
54
55    //split, decrypt and reassemble
56    j = 0;
57    for (i = 0; i < length; i++)
58    {
59      cipherblock[i - (16 * j)] = cipherstring[i];
60
61      if ((i + 1) % 16 == 0)
62      {
63        cipherblock[16] = '\0';
64        strcat(plainstring, decrypt16(cipherblock, key));
65        j++;
66      }
67    }
68
69    //set end of plainstring
```

```
70    plainstring[length] = '\0';
71
72    return plainstring;
73  }
74  /*---------------------------------------------------------------*/
75
76  //encrypt 16 Bytes
77  /*---------------------------------------------------------------*/
78  char* encrypt16(char *string, char key[16])
79  {
80    //declaration of variables
81    static char cipherstring[17] = "";
82    uint8_t i;
83
84    //put encryption-key into key-buffer
85    for(i = 0; i < 16; i++)
86    {
87      AES_KEY = key[i];
88    }
89
90    //set aes-mode and direction
91    //AES_MODE=0 -> ECB
92    AES_CTRL &= ~(1 << AES_MODE);
93    //AES_DIR=0 -> encryption
94    AES_CTRL &= ~(1 << AES_DIR);
95
96    //put plainstring into buffer
97    for (i = 0; i < 16; i++)
98    {
99      AES_STATE = string[i];
100   }
101
102   //start encryption
103   AES_CTRL |= (1 << AES_REQUEST);
104
105   // wait until encryption has finished
106   while (0 == (AES_STATUS & (1<<AES_DONE)));
107
108   //read cipherstring out of buffer
109   for(i = 0; i < 16; i++)
110   {
```

```
111       cipherstring[i] = AES_STATE;
112     }
113
114     //set end of cipherstring
115     cipherstring[16] = '\0';
116
117     return cipherstring;
118   }
119   /*-------------------------------------------------------------------*/
120
121   //decrypt 16 Bytes
122   /*-------------------------------------------------------------------*/
123   char* decrypt16(char *cipherstring, char key[16])
124   {
125     //declaration of variables
126     static char plainstring[17] = "";
127     char last_roundkey[17] = "";
128     uint8_t i;
129
130     //get last roundkey
131     strcpy(last_roundkey, get_last_roundkey(key));
132
133     //write last roundkey into key-buffer
134     for(i = 0; i < 16; i++)
135     {
136       AES_KEY = last_roundkey[i];
137     }
138
139     //set aes-mode and direction
140     //AES_MODE=0 -> ECB
141     AES_CTRL &= ~(1 << AES_MODE);
142     //AES_DIR=1 -> decryption
143     AES_CTRL |= (1 << AES_DIR);
144
145     //write ciphertext into buffer
146     for (i = 0; i < 16; i++)
147     {
148       AES_STATE = cipherstring[i];
149     }
150
151     //start decryption
```

```
152   AES_CTRL |= (1 << AES_REQUEST);
153
154   //wait until decryption has finished
155   while (0 == (AES_STATUS & (1<<AES_DONE)));
156
157   //read plainstring out of buffer
158   for(i = 0; i < 16; i++)
159   {
160     plainstring[i] = AES_STATE;
161   }
162
163   //set end of plainstring
164   plainstring[16] = '\0';
165
166   return plainstring;
167 }
168 /*------------------------------------------------------------------*/
169
170 //get last roundkey
171 /*------------------------------------------------------------------*/
172 char* get_last_roundkey(char key[16])
173 {
174   //declaration of variables
175   char dummydata[16] = "0000000000000000";
176   static char last_roundkey[17] = "";
177   uint8_t i;
178
179   //write encryption-key into key-buffer
180   for(i = 0; i < 16; i++)
181   {
182     AES_KEY = key[i];
183   }
184
185   //set aes-mode and direction
186   //AES_MODE=0 -> ECB
187   AES_CTRL &= ~(1 << AES_MODE);
188   //AES_DIR=0 -> encryption
189   AES_CTRL &= ~(1 << AES_DIR);
190
191   //write dummy-string into buffer
192   for (i = 0; i < 16; i++)
```

```
193   {
194     AES_STATE = dummydata[i];
195   }
196
197   //start dummy-encryption
198   AES_CTRL |= (1 << AES_REQUEST);
199
200   //wait until encryption has finished
201   while (0 == (AES_STATUS & (1<<AES_DONE)));
202
203   //read last roundkey out of buffer
204   for(i = 0; i < 16; i++)
205   {
206     last_roundkey[i] = AES_KEY;
207   }
208
209   //set end of string
210   last_roundkey[16] = '\0';
211
212   return last_roundkey;
213 }
```

Listing A.1: Funktionen für die Nutzung des AES-Moduls des ATmega128RFA1

A.1.2 Modifikationen an der Datei contiki-raven-main.c für die Nutzung der Sleep-Funktionen

```
1  //inclusion of header files
2  #ifdef USE_SLEEP
3  #include "radio.h"
4  #include <stdlib.h>
5  #include "avr/io.h"
6  #include <avr/sleep.h>
7  #include <util/delay_basic.h>
8
9  //defines for state transition times
10 #define TIME_NOCLK_TO_WAKE        6
11 #define TIME_CMD_FORCE_TRX_OFF    1
12 #define TIME_SLEEP_TO_TRX_OFF     880
13
14 //define for delay function
15 #define delay_us( us )   ( _delay_loop_2(1+(us*F_CPU)/4000000UL) )
16
17 //sleep flags
18 volatile uint8_t allow_rf_sleep = 0;
19 volatile uint8_t allow_cpu_sleep = 0;
20 #endif
21
22 [...]
23
24 //contiki main scheduler loop
25 int main(void)
26 {
27   initialize();
28
29   while(1)
30   {
31       process_run();
32       watchdog_periodic();
33
34 #ifdef USE_SLEEP
35     set_sleep_mode(SLEEP_MODE_PWR_SAVE);
36     sleep_enable();
37
38     if(allow_rf_sleep == 0)
```

```
39      {
40        //take AT86RF230 to sleep
41        radio_enter_sleep_mode();
42      }
43      else
44      {
45        //wake up at86rf230
46        radio_leave_sleep_mode();
47      }
48
49      if(allow_cpu_sleep == 0)
50      {
51        //take ATmega1284p to sleep
52        sleep_cpu();
53      }
54  #endif
55
56  [...]
57
58  }
```

Listing A.2: Modifikationen an der Datei contiki-raven-main.c für die Nutzung der Sleep-Funktionen

A.1.3 Portierung von Treiberfunktionen der RAVEN-Hardware

```
1  [...]
2
3  uint8_t RF230_radio_on;
4
5  void radio_reset_state_machine(void)
6  {
7    hal_set_slptr_low();
8    delay_us(TIME_NOCLK_TO_WAKE);
9    hal_subregister_write(SR_TRX_CMD, CMD_FORCE_TRX_OFF);
10   delay_us(TIME_CMD_FORCE_TRX_OFF);
11 }
12
13 bool radio_is_sleeping(void)
14 {
15   bool sleeping = false;
16
17   if (hal_get_slptr() != 0){
18     sleeping = true;
19   }
20   return sleeping;
21 }
22
23 uint8_t radio_get_trx_state(void)
24 {
25   return hal_subregister_read(SR_TRX_STATUS);
26 }
27
28 radio_status_t radio_enter_sleep_mode(void)
29 {
30   if (radio_is_sleeping() == true){
31     return RADIO_SUCCESS;
32     }
33
34   radio_reset_state_machine();
35   radio_status_t enter_sleep_status = RADIO_TIMED_OUT;
36
37   if (radio_get_trx_state() == TRX_OFF){
38     hal_set_slptr_high();
39     enter_sleep_status = RADIO_SUCCESS;
```

```
40 #if RADIOSTATS
41     RF230_radio_on = 0;
42 #endif
43   }
44   return enter_sleep_status;
45 }
46
47 radio_status_t radio_leave_sleep_mode(void)
48 {
49   if (radio_is_sleeping() == false){
50     return RADIO_SUCCESS;
51     }
52
53   hal_set_slptr_low();
54   delay_us(TIME_SLEEP_TO_TRX_OFF);
55   radio_status_t leave_sleep_status = RADIO_TIMED_OUT;
56
57   if (radio_get_trx_state() == TRX_OFF){
58     leave_sleep_status = RADIO_SUCCESS;
59 #if RADIOSTATS
60     RF230_radio_on = 1;
61 #endif
62   }
63   return leave_sleep_status;
64 }
65 #endif
66
67 [...]
```

Listing A.3: Portierung von Treiberfunktionen der RAVEN-Hardware

A.2 Messdaten

A.2.1 Zeitspanne für die Versendung unterschiedlicher Paketgrößen

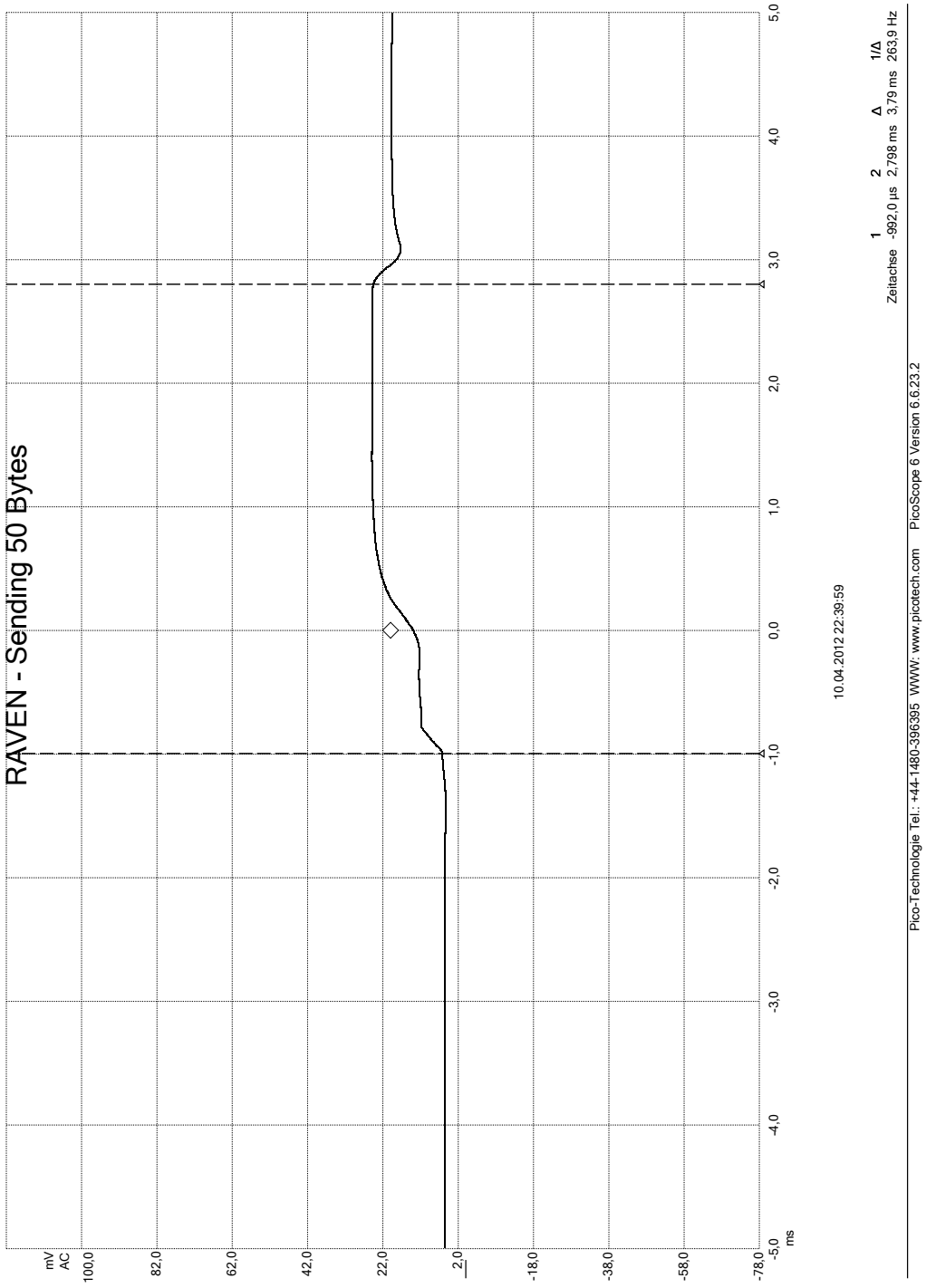

Abb. A.1: Zeitspanne des RAVEN-Kits für die Versendung von 50 Byte

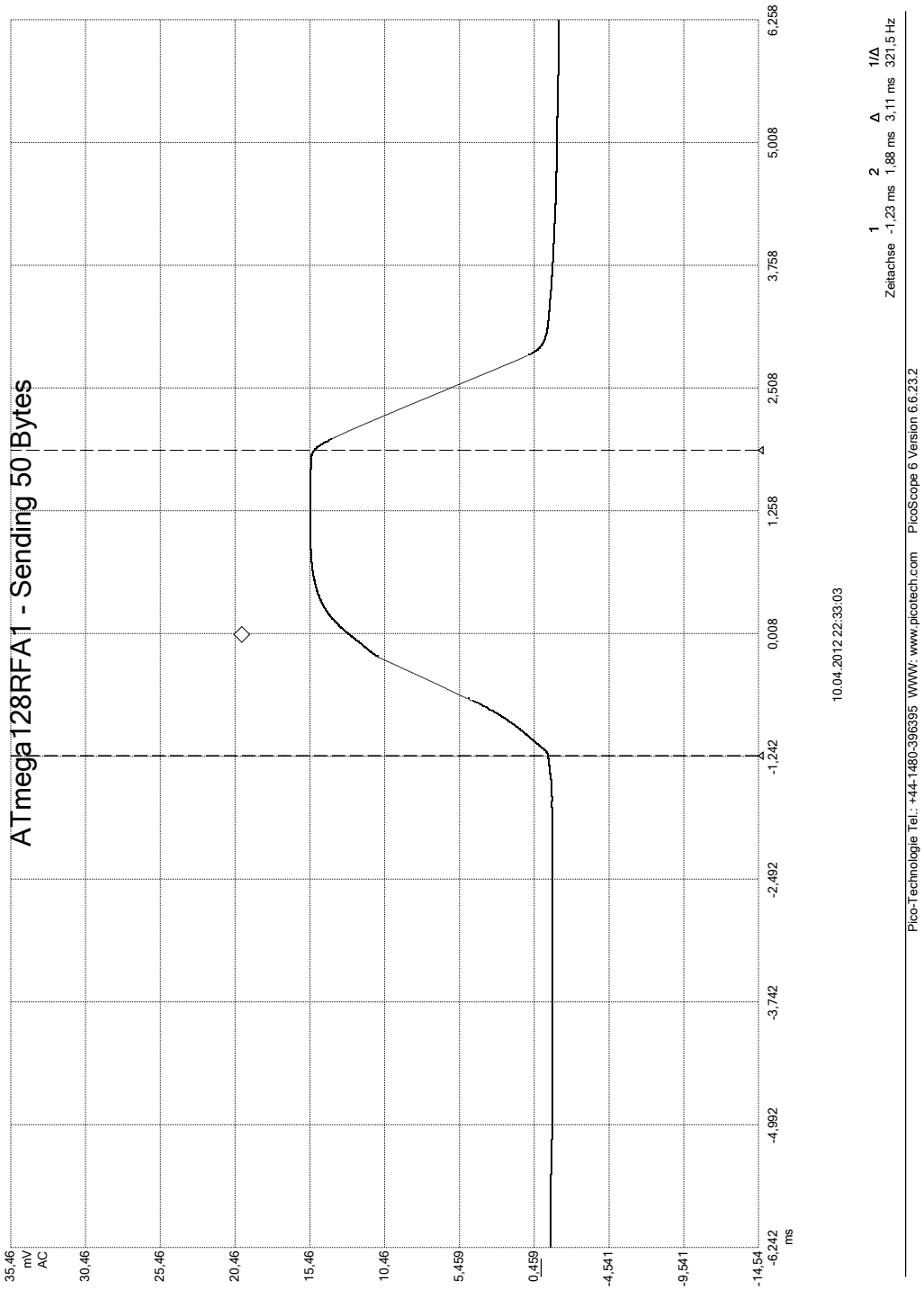

Abb. A.2: Zeitspanne des ATmega128RFA1 für die Versendung von 50 Byte

A.2.2 Vergleich der Datenübertragungsgeschwindigkeiten

		ATmega128RFA1		RAVEN-Hardware	
Payload (in Byte)	Paketgröße (in Bit)	Anzahl Pakete (in 1 / Min.)	Datendurchsatz brutto (in kBit/s)	Anzahl Pakete (in 1 / Min.)	Datendurchsatz netto (in kBit/s)
10	576	24808	238,16	15902	152,66
20	656	21566	235,79	14501	158,54
30	736	19086	234,12	13347	163,72
40	816	17103	232,60	12340	167,82
50	896	15496	231,41	11486	171,52
60	976	14173	230,55	10749	174,85
70	1056	13053	229,73	10072	177,27
80	1136	12099	229,07	9509	180,04
90	1216	11272	228,45	8987	182,14
100	1296	7298	157,64	5491	118,61
200	2096	4321	150,95	3345	116,85
400	3696	2381	146,67	1877	115,62
600	5296	1643	145,02	1305	115,19
800	6896	1255	144,24	1000	114,93
1000	8496	1015	143,72	810	114,70
1200	10096	852	143,36	682	114,76
1232	10352	837	144,41	672	115,94

Tab. A.1: Messprotokoll zum Datendurchsatz

A.2.3 Auswirkung der Hardwareverschlüsselung auf den Datendurchsatz

		Ohne Verschlüsselung		Mit Verschlüsselung	
Payload (in Byte)	Paketgröße (in Bit)	Anzahl Pakete (in 1 / Min.)	Datendurchsatz brutto (in kBit/s)	Anzahl Pakete (in 1 / Min.)	Datendurchsatz brutto (in kBit/s)
10	576	24808	238,16	20224	194,15
20	656	21566	235,79	15778	172,51
30	736	19086	234,12	15358	188,39
40	816	17103	232,60	12670	172,31
50	896	15496	231,41	10518	157,07
60	976	14173	230,55	10594	172,33
70	1056	13053	229,73	9022	158,79
80	1136	12099	229,07	8843	167,43
90	1216	11272	228,45	1031	20,89
100	1296	7298	157,64	1013	21,88
200	2096	4321	150,95	758	26,48

Tab. A.2: Messprotokoll zur Auswirkung der Hardwareverschlüsselung

Literaturverzeichnis

[1] Atmel Corporation, ATmega128RFA1 Preliminary, Revision C (Oktober 2011).
URL http://www.atmel.com/dyn/resources/prod_documents/doc8266.pdf

[2] Atmel Corporation, ATmega1284P Preliminary, Revision D (November 2009).
URL http://www.atmel.com/Images/doc8059.pdf

[3] Carl von Ossietzky Universität, Oldenburg, Advanced Encryption Standard: Schlüsselexpansion (März 2012).
URL http://celan.informatik.uni-oldenburg.de/kryptos/info/aes/keys/

[4] WieDAS-Projekt Team, WieDAS-Projekt Website - Über uns (2011).
URL http://www.wiedas.org/index.php/wiedas

[5] Wikipedia, Advanced Encryption Standard (März 2012).
URL http://de.wikipedia.org/wiki/Advanced_Encryption_Standard

[6] Texas Instruments, Small Size 2.4 GHz PCB antenna (April 2008).
URL http://www.ti.com/lit/an/swra117d/swra117d.pdf

[7] Atmel Corporation, AT86RF230 Manual, Revision E (Februar 2009).
URL http://www.atmel.com/Images/doc5131.pdf

[8] Johanson Technology, Technical Datasheet: 2450AT18A100 (März 2009).
URL http://www.farnell.com/datasheets/1318295.pdf

[9] Johanson Technology, Technical Datasheet: 2450FB15L0001 (Juli 2010).
URL http://www.farnell.com/datasheets/1318295.pdf

[10] Analog Devices Incorporated, MT-094 Tutorial: Microstrip and Stripline Design (Januar 2009).
URL http://www.analog.com/static/imported-files/tutorials/MT-094.pdf

[11] Bernhard Esders, Fachhochschule Düsseldorf, Bachelor-Thesis: Entwicklung, Implementierung und Evaluierung einer speichereffizienten, drahtlosen Kommunikation von eingebetteten Systemen mithilfe des IP-Protokolls basierend auf einem Multitasking-Betriebssystem für 8-Bit AVR Mikrocontroller (August 2010).

[12] Mirco Kern, Fachhochschule Düsseldorf, Projektdokumentation: Untersuchung eines Mikroprozessor-Betriebssystems mit 6LoWPAN-Stack in Hinblick auf die Portierung auf neue Prozessoren (Januar 2012).

[13] AVR-Crypto-Lib Entwickler, AVR-Crypto-Lib Wiki (April 2012).
URL http://avrcryptolib.das-labor.org/trac/wiki/AES

Selbständigkeitserklärung

Hiermit erkläre ich an Eides statt, dass ich die am heutigen Tage vorgelegte Bachelorarbeit zum Thema

Optimierung der drahtlosen Kommunikation zwischen Komponenten mit Contiki-Betriebssystem in Hinblick auf Datendurchsatz, Energieeffizienz und Sicherheit am Beispiel des WieDAS-Projektes

selbstständig angefertigt und keine andere als im Schriftumsverzeichnis angegebene Literatur benutzt habe..

Düsseldorf, den 11.04.2012

Unterschrift