Diplomarbeit

Architecture Verification and Embedded Firmware Implementation for a MAC Layer Protocol Processor

Goran Vukeljic

Institut für Elektronik
Technische Universität Graz
Leiter: Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Bösch
Begutachter: Univ.-Prof. Dipl.-Ing. Dr.techn. Söser Peter
Betreuer bei Infineon: Dipl.-Ing. Dr.techn. Matischeck Rainer

unterstützt durch Infineon Technologies Austria AG

Graz, im Oktober 2012

Never stop thinking

Diese Diplomarbeit wurde unterstützt von
Infineon Technologies Austria AG
Development Center Graz
Abteilung Contactless and RF Exploration
Leitung Holweg Gerald

Abstract

The aim of this thesis as part of the funded CHOSeN project is to develop an advanced Firmware for a Smart Radio Transceiver, special based on an optimized programmable state machine. This includes the development or expansion of existing firmware functions running on the Smart Transceiver, and the development of corresponding test functions, that run on a SystemC based virtual μController and control the Smart Transceiver via a special SPI protocol. As development environment serves an Eclipse IDE with a compiler for the state machine, SystemC Compiler and QuestaSim to simulate and debug the entire system as well partly an FPGA prototype was used.

Kurzfassung

Ziel dieser Diplomarbeit im Rahmen des geförderten CHOSeN Projekts ist die Entwicklung einer fortgeschrittenen Firmware für einen Smart Radio Transceiver (zur Unterstütung für verschiedene PHY und MAC Funktionen), basierend auf einer speziellen optimierten programmierbaren State-Machine. Dies umfasst die Entwicklung bzw. Erweiterung der bestehenden (derzeit noch eingeschränkten) Firmware Funktionen, die auf dem Smart Transceiver laufen, sowie die Entwicklung korrespondierender Test-Funktionen, die auf einem SysemC basierten virtuellen μController laufen und den Smart Transceiver via speziellem SPI Protokoll ansteuert. Als Entwicklungsumgebung dienen eine Eclipse IDE mit einem Compiler fur die State-Machine (reduziertes C-subset bzw. aus AMS) und SystemC-Compiler, sowie zum Simulieren und Debuggen des Gesamtsystems (Transceiver inkl. Firmware + virtueller μController) dient QuestaSim sowie teilweise auch ein FPGA Prototyp.

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz,am ....

(Unterschrift)

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....

date (signature)

Danksagung

Die vorliegende Diplomarbeit entstand mit Unterstützung der Firma Infineon Technologies AG am Ende meines Elektrotechnik Studiums an der Technischen Universität Graz. Mein Dank für die fachlichen Hilfestellungen und teils sehr spannenden und andauernden Diskussionen geht an die Gruppe der Firma Infineon Technologies AG.

Goran Vukeljic

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Eine Firmware ist eine Software, die fest in elektronische Geräte eingebettet ist. Sie wird dauerhaft in einem Flash-Speicher, EPROM, PROM oder ROM gespeichert. Die Firmware ist funktionell mit der Hardware verbunden, was bedeutet, dass das eine ohne das andere nicht nutzbar ist. Diese Software wird häufig als Routine ausgeführt, die der Kontrolle des Computers dient. Eine der bekanntesten Routinen ist das BIOS (engl. basic input/output system). Die Firmware ist in aller Regel dauerhaft gespeichert, bedingt durch die elektrisch löschbaren ROMs kann sie sich ändern.[3] [11]

1.1 Gliederung der Diplomarbeit

Das einleitende Kapitel schildert in kurzen Zügen unter Motivation, was mich zu dieser Arbeit bewogen hat. Das zweite Kapitel dient als einführendes Kapitel für Drahtlose Sensornetzwerke und ZigBee Standards. Ebenso werden beim ZigBee, der Aufbau der Netzwerkschicht und der Anwendungsschicht besprochen. Im nachfolgenden dritten Kapitel wird die Hardware, des noch in Entwicklung stehenden Infineon Transceiver Chips beschrieben. Das dritte Kapitel zeigt noch, in Code Beispielen, wie die einzelnen Hardware Module in der Firmware realisiert sind. Die erste Aufgabe der Diplomarbeit war das ROM-Paging und wird im Kapitel 4 verdeutlicht, dabei wird eine spezielle Methode des Pagings (DIVT Dynamic Interrupt Vector Table) entwickelt. Das fünfte Kapitel behandelt die Messung des RSSI Wertes (Received Signal Strength Indication) und die Beurteilung der Kanalfreiheit (CCA Clear Channel Assessment). Die Funktionalität des Schedules und dessen Realisierung in einem Transceiver werden im Kapitel 6 erklärt. Als Abschluss der Diplomarbeit wurde die Kommunikation zwischen zwei Transceiver Boards getestet und ist im Kapitel 7 erklärt. Das letzte Kapitel (Kapitel 8) beinhaltet eine Zusammenfassung der durchgeführten Entwicklungen und einen Ausblick in die Zukunft.

1.2 Motivation

Funksensoren spielen in der Technik eine wichtige Rolle. Die von ihnen erfassten Werte oder Zustände werden, meistens elektrisch-elektronisch verstärkt, in der zugehörigen Steuerung verarbeitet, die entsprechende weitere Schritte auslöst. Funksensoren sind drahtlos arbeitende Sensoren, die mit einem Funkmodem ausgestattet sind. Funksensoren beinhalten einen Mikroprozessor oder ein Mikrosystem und besitzen sozusagen „Intelligenz“, daher werden sie auch als Smart-Sensoren (engl. smart sensors) bezeichnet. Da sie mit eigener Stromversorgung arbeiten, ist der Stromverbrauch im Ruhemodus ein wichtiger Parameter, der einen unmittelbaren Einfluss auf die Batterielebensdauer hat. Die Vernetzung mehrere solcher Smart-Sensoren wird als WNS (engl. Wireless Sensor Network) bezeichnet, die im Deutschen als Drahtloses Sensornetzwerk bezeichnet wird (siehe Abbildung 1.1).

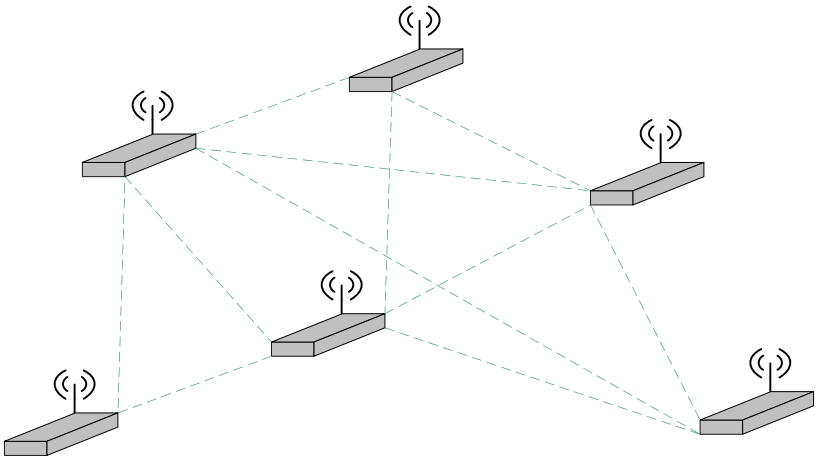

Abbildung 1.1: Drahtloses Sensornetzwerk

„*Es gibt verschiedene Ansätze und Standards für Wireless Sensor Networks, einer der bekanntesten ist der von der IEEE-Arbeitsgruppe 802.15.4 (siehe Abbildung 1.2). Das 802.15.4-Konzept kennt mehrere Varianten, die sich in den Modulationsverfahren und der Leistungsaufnahme unterscheiden. Andere Ansätze sind Ultra Low Power Bluetooth (ULP), bekannt als Wibree und ISA-SP100 für drahtlose Sensornetzwerke in der Automatisierungs- und Steuerungsumgebung auf Feldebene. Ein weiteres energiesparsames Konzept für drahtlose Sensornetzwerke ist ANT+ (a proprietary wireless sensor network technology) mit dem ANT-Protokoll.*“ [2]

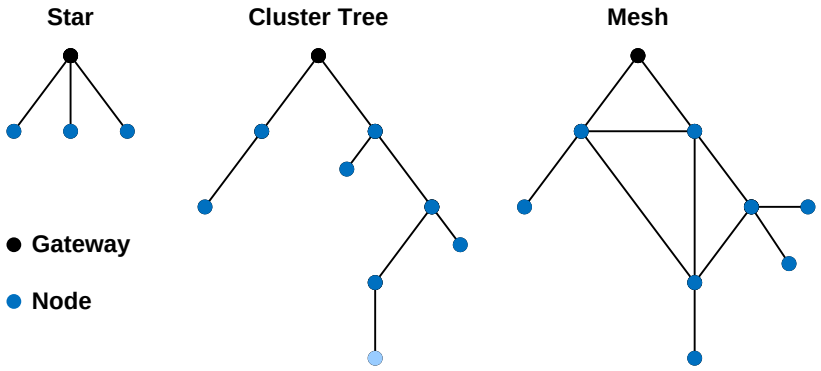

Abbildung 1.2: Netzwerk-Topologie [1]

Die Vorteile eines auf Standards basierenden Wireless-Netzwerks liegen u. a. in den geringeren Kosten und austauschbaren Produkten von verschiedenen Herstellern. Des Weiteren existieren bereits bewährte Verfahrensweisen, auf die zurückgegriffen werden kann. Außerdem sind Standards wie z. B. IEEE 802.11 bereits in bestehende Ethernet-basierte Systeme integriert. Die Wahl eines offenen Softwaresystems ist nicht nur für die Integration, sondern auch für die Skalierbarkeit von Systemen wichtig.

„*Sensornetzwerke erfassen Sensordaten und übertragen diese über Verarbeitungsrechner zur Steuerung der Aktoren. Über diese drahtlosen Netzwerke werden relativ geringe Datenmengen übertragen. WSN zeichnen sich durch einfache Installation, Selbstkonfiguration, Wartungsfreiheit, Störfestigkeit und einen geringen Stromverbrauch aus. Letzterer bildet die Voraussetzung, dass Sensornetzwerke über mehrere Jahre mit Batterien betrieben werden können. Durch entsprechende Energiespartechniken können Funksensoren mit ZigBee mehrere Jahre mit einer Batterie arbeiten. Neben den batteriebetriebenen Funksensoren gibt es noch die batterielosen. Diese Sensoren basieren auf Micro Energy Harvesting und gewinnen ihre Energie aus den Umgebungsbedingungen wie z.B. aus Licht, Temperatur, Luftbewegung oder Druck. Für die Übertragung stehen die lizenzfreien ISM-Bänder zwischen 315 - 2,4835 GHz zur Verfügung. Die Frequenzbänder bis 434 MHz und zwischen 868 MHz und 870 MHz werden vorwiegend in Europa benutzt und eignen sich für mittlere Übertragungsdistanzen in Gebäuden. Das 2,4-GHz-Band kann mit geringer Leistung über kurze Distanzen von einigen Metern benutzt werden.*“[2][6]

Drahtlose Datenübertragung ist im Vormarsch! Eine wichtige Rolle bei dieser Datenübertragung spielt der Sensorknoten, bestehend im Kern aus einem Prozessor und einem Datenspeicher. Dazu kommen ein oder mehrere Sensoren und ein Modul zur Funkkommunikation. Bei neueren Modellen sind alle Bauteile auf einem einzigen Chip untergebracht.

In dieser Diplomarbeit wird die fest eingebettete Software (Firmware), für einen von Infineon entwickelten Transceiver (siehe Abbildung 1.3) erklärt. Diese Software ermöglicht es den Transceiver mit einem Zeitplaner (Scheduling) seine Aufgaben besser zu managen.

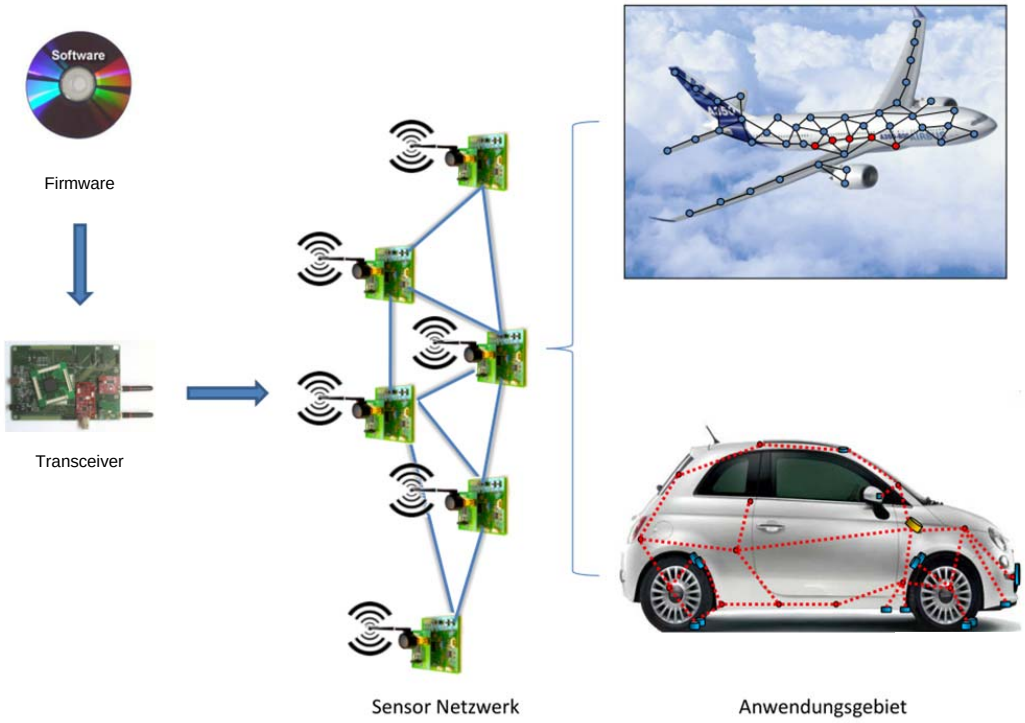

Abbildung 1.3: Konzept der Firmware implementierung in Drahtlose Sensornetzwerke: Nachgezeichnet aus Infineon Unterlagen

> *"Der beste Weg, die Zukunft vorauszusagen, ist, sie selbst zu gestalten"*
> *Willy Brandt*

Kapitel 2

IEEE 802.15.4 Standard

Die Standardisierung von Drahtlosen Sensornetzwerken geht von zwei Richtlinien aus: IEEE 802.15.4-Standard und ZigBee. Diese zwei Standards spezifizieren verschiedene Unterschichten des OSI-Schichtenmodells (siehe Abbildung 2.1):

- IEEE 802.15.4 definiert die Bitübertragungsschicht (engl. Physical Layer) und die Sicherungsschicht (engl. Medium Access Control Layer)
- ZigBee definiert die Vermittlungsschicht (engl. Network Layer) und die Anwendungsschicht (engl. Application Layer)

Die zwei Protokoll-Stacks können kombiniert werden um LDR (Low Data Rate 30 - 300kbit/s) und lang anhaltende Anwendungen Batterie betriebener Wireless-Geräte zu unterstützen. Anwendungsgebiet dieser Standards sind Sensoren, interaktives Spielzeug, Smart Badges, Fernbedienungen und Hausautomation.[7]

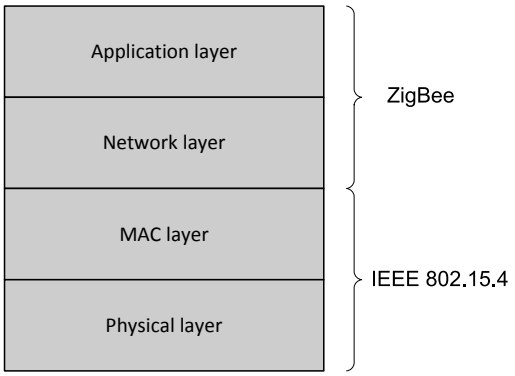

Abbildung 2.1: OSI-Schichtenmodell

Der Standard IEEE 802.15.4 beschreibt ein Übertragungsprotokoll für Wireless Personal Area Networks (WPAN). Er definiert die untersten beiden Schichten des OSI-Modells, den PHY-Layer und den MAC-Layer. Dessen Protokoll-Stack ist einfach und flexibel und erfordert keine Infrastruktur, die für Nahbereichskommunikation geeignet ist (typischerweise innerhalb eines Bereichs von 100 m). Aus diesen Gründen bietet es eine einfache Installation, geringe Kosten und eine angemessene Lebensdauer der Batterie der Geräte. Die Bitübertragungsschicht des IEEE 802.15.4-Standards wurde so konzipiert, dass es das Koexistieren mit anderen IEEE-Standards für drahtlose Netzwerke erlaubt, zum Beispiel IEEE 802.11 und IEEE 802.15.1 (Bluetooth). Es verfügt über Aktivierung und Deaktivierung der Funkgeräte und die Übertragung von Paketen auf den physikalischen Schnittstellen. Es arbeitet in einem der folgenden drei lizenzfreien Frequenzbänder:

- 868 bis 868,6 MHz (z.B. Europa) mit einer Datenrate von 20 kbps
- 902 - 928 MHz (z.B. Nordamerika) mit einer Datenrate von 40 kbps
- 2400 - 2483,5 MHz (weltweit) mit einer Datenrate von 250 kbps

Der MAC-Layer stellt Daten und Verwaltungsdienste an die oberen Schichten bereit. Der Datendienst ermöglicht das Senden und Empfangen von MAC-Paketen über die Bitübertragungsschicht. Die Management-Dienstleistungen umfassen die Synchronisation der Kommunikation, Management der garantierten Zeitschlitze (engl. Guaranteed Time Slots), und Verbindung/Trennung der Geräte vom Netzwerk. Zusätzlich implementiert die MAC-Schicht grundlegende Sicherheitsmechanismen.

Die Abkürzungen in diesem Abschnitt sind in der Tabelle 2.1 aufgeführt.

Acronym	Definition
ACL	Access Control List
CAP	Contention Access Period
CFP	Contention Free Period
CSMA - CA	Carrier Sense Multiple Access with Collision Avoidance
FFD	Full - Function Devices
GTS	Guaranteed Time Slots
MAC	Medium Access Control layer
PAN	Personal Area Network
RFD	Reduced Function Devices

Tabelle 2.1: Tabelle: Abkürzungen

2.1 Überblick über die MAC-Schicht

Die MAC-Schicht definiert zwei Typen von Knoten:

- Full Function Devices (FFDs). Ein Full Function Device kann Datenpakete senden und empfangen und leitet die Datenpakete von den RFD-Knoten weiter an den PAN-Koordinator oder umgekehrt. Sie erfüllt damit Routing-Funktionen zwischen RFDs, FFDs und PAN-Koordinator.
- Reduced Function Devices (RFDs). Die Reduced Function Devices haben keine Weiterleitfunktion, sie können Datenpakete empfangen und senden. Sie bilden den Sensor-Knoten und empfangen Steuersignale oder senden Sensorsignale an den Netzwerk-Koordinator oder an einen FFD-Knoten, der diese dann weiterleitet.

Die FFDs arbeiten zusammen um die Netzwerktopologie zu implementieren. Die tatsächliche Netzwerkbildung wird in der Netzwerkschicht vorgenommen, aber die MAC-Schicht unterstützt zwei Arten von Netzwerk-Topologien:

- Stern
- Rechner-Rechner-Verbindung (engl. Peer-to-Peer)

In der Stern-Topologie nimmt ein FFD die Rolle eines PAN-Koordinators ein und befindet sich im Stern-Zentrum. Alle anderen FFDs und RFDs verhalten sich als generische Geräte und können nur mit dem Koordinator kommunizieren, der die gesamte Kommunikation in dem Netzwerk synchronisiert. Verschiedene Sterne, die im gleichen Bereich arbeiten, haben unterschiedliche PAN Identifikatoren und arbeiten unabhängig voneinander. Ein Beispiel einer Stern-Topologie ist in der Abbildung 2.2a gezeigt.

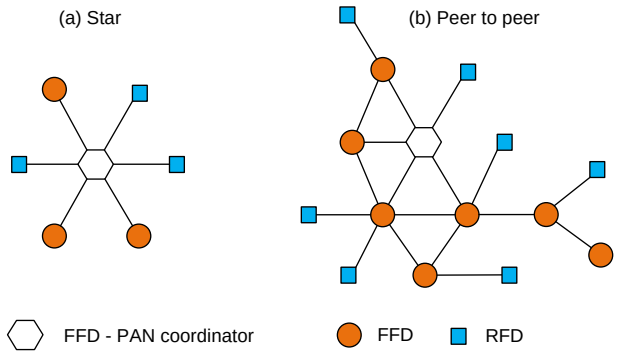

Abbildung 2.2: Netzwerk Toopologie IEEE 802.15.4 MAC Schicht [7]

Bei der Peer-to-Peer-Topologie ist jedes FFD imstande mit einem anderen Gerät innerhalb seiner Funkreichweite zu kommunizieren. Ein FFD wirkt als PAN-Koordinator und die anderen FFDs wirken als Router oder Endgeräte um ein Multi Hop-Netzwerk zu bilden, wie in der Abbildung 2.2b gezeigt ist. Die RFDs wirken als Endgeräte und jedes RFD ist nur mit einem FFD verbunden.

2.2 Kanal-Zugang

Das MAC-Protokoll hat zwei Arten von Kanalzugriff:

- mit Superframe-Structure
- ohne Superframe-Structure

Superframes (SF) sind größere Datenpakete, die aus mehreren Frames zusammen gesetzt werden. Ein solches Superframe hat einen Kopfteil, die weiteren im Superframe enthaltenen Datenpakete haben eigene Header. Die Technik der Superframes wird bei ATM (engl. asynchronous transfer mode), der Plesiochronous Digital Hierarchy (PDH), im ADSL-Rahmen und im Digital Radio Mondial (DRM) benutzt.[6]

Der Kanal-Zugang mit einer Superframe-Structure wird in der Stern-Topologie eingesetzt und ermöglicht die Synchronisierung zwischen den Knoten, die die Energieeinsparung der Geräte aktiviert. Ein Kanal-Zugang ohne Superframe-Structure ist allgemeiner und kann verwendet werden um die Kommunikation in willkürlichen Peer-to-Peer Topologien zu unterstützten.

2.2.1 Die Kommunikation mit einer Superframe-Structure

Ein Superframe besteht aus einem aktiven und einem inaktiven Teil. Die Kommunikation findet während des aktiven Teils statt. Daher kann der PAN-Koordinator während des inaktiven Teils in einen Energiesparmodus (Sleep) gehen. Der aktive Teil weist bis zu 16 gleich große Zeitschlitze auf. Der erste Zeitschlitz ist der Beacon-Frame und wird durch den PAN-Koordinator gesendet, um den Superframe zu beginnen. Die Beacon-Frames werden verwendet um die angeschlossenen Geräte zu synchronisieren, um das PAN zu identifizieren und um die Struktur des Superframes zu beschreiben. Die eigentliche Kommunikation zwischen den Endgeräten und dem Koordinator findet in den verbleibenden Zeitschlitzen statt. Die Zeitschlitze im aktiven Teil sind in eine CAP und einen (optionalen) CFP unterteilt.

Im CAP-Zeitraum konkurrieren die Geräte um einen Kanal-Zugriff zu bekommen und benutzen dabei ein dafür angepasstes CSMA-CA Protokoll (engl. carrier sense multiple

access with collision avoidance). Dies bedeutet, dass ein Gerät den Wunsch hat Daten-Frames zu übertragen. Es wartet erst auf ein Beacon-Frame und wählt dann zufällig einen Zeitschlitz für seine Übertragung. Wenn der ausgewählte Zeitschlitz besetzt ist, weil eine andere Kommunikation bereits am laufenden ist (dies wird unter Verwendung von Carrier Sense CS ermöglicht), wählt das Gerät zufällig einen anderen Zeitschlitz. Wenn der Kanal frei ist, kann das Gerät mit der Übertragung im den nächsten Zeitschlitz beginnen.

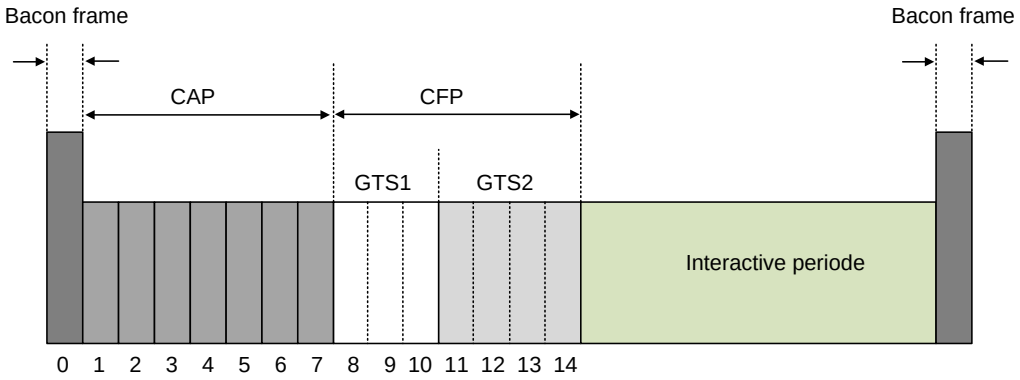

Abbildung 2.3: Superframe

Der CFP-Zeitraum ist optional und wird für Low-Latency-Anwendungen oder Anwendungen, die nach einer spezifischen Datenbandbreite verlangen, verwendet. Zu diesem Zweck kann der PAN Koordinator Teile des aktiven Surerframes (GTS) auf spezifische Anwendungen zuweisen. Die GTS bilden die CFP, die immer am einem Ende des aktiven Superframes anfängt, beginnend bei einer Zeitschlitzgrenze unmittelbar nach der CAP. Jedes GTS kann mehr als einen Zeitschlitz beinhalten und ist einer einzelnen Anwendung zugeordnet, die ohne Konflikte zugreift. In jedem Fall hinterlässt der PAN-Koordinator immer eine ausreichende Anzahl von Frames, für den CAP-Zeitraum, für die anderen Geräte, und verwaltet die Verbindungs-/Trenn-Protokolle. Es muss beachtet werden, dass alle Konflikte - Transaktionen vor dem Beginn der CFP abgeschlossen sind, und jedes Gerät, das in eine GTS überträgt, seine Übertragung innerhalb der GTS abschließt. Die Superframe-Struktur wird in Abbildung 2.3 dargestellt.

2.2.2 Die Kommunikation ohne eine Superframe-Structure

Der PAN-Koordinator vermeidet wahlweise den Einsatz einer Superframe-Struktur. In diesem Fall sendet der PAN-Koordinator niemals Signalisierungen (engl. beacons) und die Kommunikation erfolgt über die grundlegenden zugeschnittenen CSMA-CA-Protokolle. Der Koordinator ist immer aktiv und bereit Daten von einem Endgerät zu empfangen während die Datenübertragung in die entgegengesetzte Richtung auf einer Abstimmung basiert: das Endgerät wacht regelmäßig auf und fragt den Koordinator auf ausstehende Nachrichten. Der Koordinator antwortet auf diese Anforderung, indem er die ausstehenden Nachrichten versendet oder signalisiert, dass keine Nachrichten verfügbar sind.

2.3 Datenübertragungs Modelle

Der Standard unterstützt drei Modelle für die Datenübertragung:

- vom Endgerät zum Koordinator
- vom Koordinator zum Endgerät
- Peer-to-Peer

Die Stern-Topologie verwendet nur die ersten beiden Modelle, weil die Datenübertragung nur zwischen dem PAN-Koordinator und den anderen Geräten passiert. In der Peer-to-Peer-Topologie sind alle drei Modelle möglich, da die Daten zwischen beliebigen Gerätepaaren ausgetauscht werden können. Die tatsächliche Umsetzung der drei Datenübertragungsmodelle hängt davon ab, ob das Netzwerk die Übertragungssignatur unterstützt.

2.3.1 Datenübertragungen in Beacon-Enabled Netzwerken

- *Datenübertragung von einem Endgerät zu einem Koordinator.* Das Endgerät wartet zunächst auf eine Netzwerksignatur um mit dem Superframe zu synchronisieren. Wenn die Signatur empfangen wurde und sie ein GTS besitzt, wird die GTS direkt ausgeführt. Ansonsten übertägt es (Endgerät) dem Datenframe an den Koordinator mit den zugeschnittenen CSMA-CA-Protokoll, in einem der Frames des CAP-Zeitraums. Der Koordinator kann gegebenenfalls in aufeinanderfolgenden Zeitschlitz, den erfolgreichen Empfang der Daten bestätigen durch Senden einer Bestätigungsnachricht. Dieses Protokoll wird in der Abbildung. 2.4a gezeigt.
- *Datenübertragung von einem Koordinator zu einem Endgerät.* Der Koordinator speichert die Nachricht (ein Datenframe) und zeigt es in der Netzwerksignatur an,

dass eine Nachricht ansteht. Das Endgerät schläft normalerweise die meiste Zeit und hört periodisch Netzwerksignaturen ab, ob ausstehende Nachrichten zu lesen sind. Wenn es (Endgerät) merkt, dass eine Nachricht da ist, wird es explizit den Koordinator auffordern die Nachricht, mit den zugeschnittenen CSMA-CA-Protokoll, im CAP-Zeitraum zu senden. Im Gegenzug versendet der Koordinator die anstehende Nachricht, mit den zugeschnittenen CSMA-CA-Protokoll, im CAP-Zeitraum. Das Gerät bestätigt somit den Empfang der Daten, durch Senden einer Bestätigungsnachricht im folgenden Zeitschlitz, sodass der Koordinator die anstehende Nachricht von seiner Liste entfernen kann. Dieses Protokoll wird in Abbildung 2.4b gezeigt.

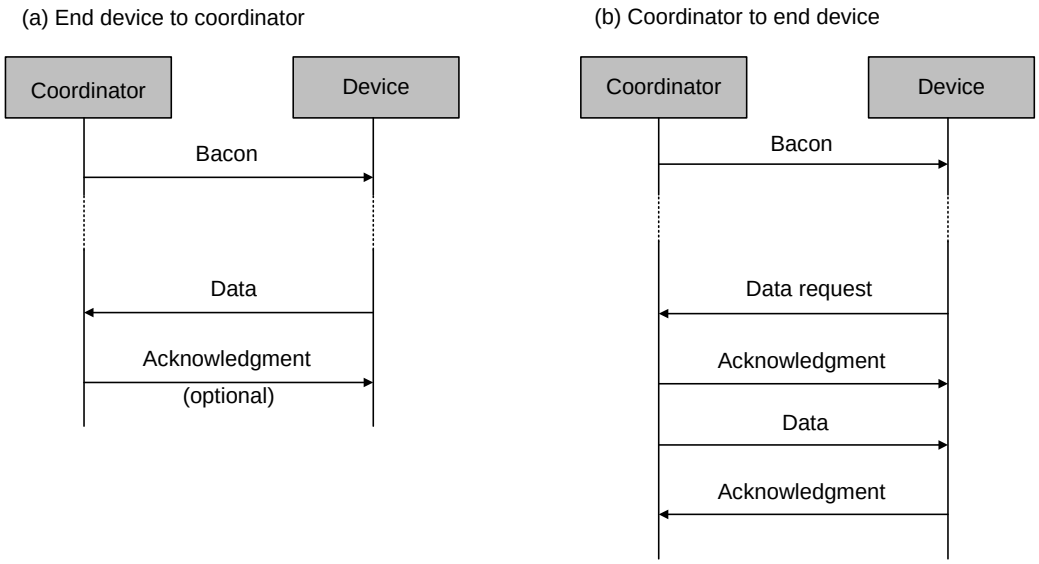

Abbildung 2.4: Datenübertragungen in Beacon-Enabled Netzwerken [7]

- *Peer-to-Peer Datenübertragung.* Wenn der Sender oder der Empfänger ein Endgerät ist wird eines der vorher erwähnten Datenübertragungsmodelle verwendet. Im anderen Fall, wenn die Koordinatoren Sender/Empfänger sind, senden Sie ihre eigenen Signaturen (beacons). In diesem Fall muss der Sender zunächst mit der Signatur des Ziels synchronisieren und sich als ein Endgerät verhalten. Die zu ergreifenden Maßnahmen, um die Koordinatoren zu synchronisieren übersteigen den Rahmen des IEEE 802.15.4-Standards und werden somit den oberen Schichten überlassen.

2.3.2 Datenübertragungen in NonBeacon-Enabled Netzwerken

- *Datenübertragung von einem Endgerät zu einem Koordinator.* Das Endgerät überträgt direkt sein Datenframe an den Koordinator, mit dem zugeschnittenen CSMA-CA-Protokoll. Der Koordinator bestätigt den erfolgreichen Empfang der Daten durch Übertragung eines optionalen Bestätigungsframes. Dieses Protokoll wird in der Abbildung 2.5a gezeigt.
- *Datenübertragung von einem Koordinator zu einem Endgerät.* Der Koordinator speichert die Nachricht (Datenframe) und wartet auf ein Gerät, das die Daten anfordert. Ein Gerät kann vom Koordinator die anstehende Nachricht durch Senden einer Anfrage mit dem zugeschnittenen CSMA-CA-Protokoll fordern. Der Koordinator bestätigt den erfolgreichen Empfang der Anforderung durch Senden einer Bestätigungsnachricht. Wenn anstehende Nachrichten vorhanden sind überträgt der Koordinator die Nachrichten an das Gerät, mit dem zugeschnittenen CSMA-CA-Protokoll. Andernfalls, wenn keine Nachrichten anstehen sendet der Koordinator eine Nachricht mit der Länge null der Nutzdaten (was bedeutet, dass keine Daten anstehen). Das Gerät bestätigt den erfolgreichen Empfang der Nachrichten durch Senden einer Bestätigungsnachricht, so dass der Koordinator die anstehende Nachricht verwerfen kann. Dieses Protokoll wird in der Abbildung 2.5b gezeigt.

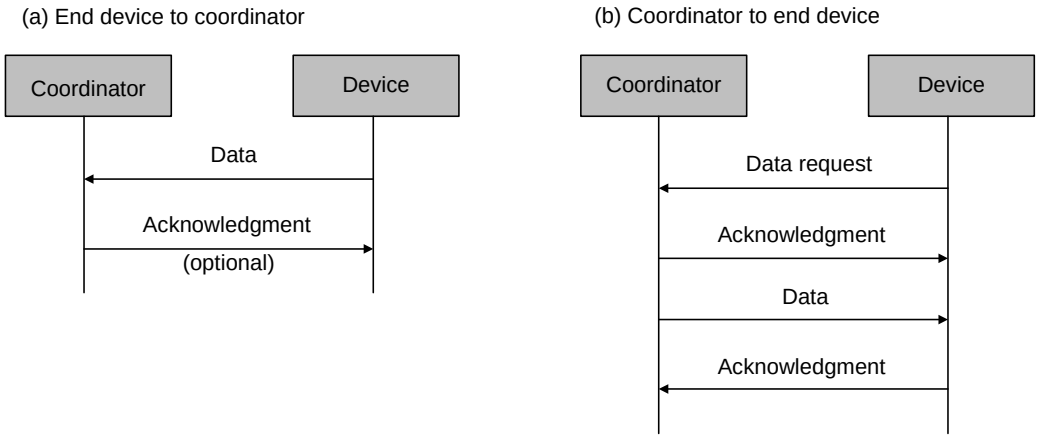

Abbildung 2.5: Datenübertragungen in NonBeacon-Enabled Netzwerken [7]

- *Peer-to-Peer Datenübertragung.* In Peer-to-Peer PANs kann jedes Gerät mit jedem anderen Gerät innerhalb seiner Reichweite kommunizieren. Um dies effektiv zu erfüllen, müssen die Geräte, die eine Komunikation aufbauen wollen:

1. den Funk ständig aufrechterhalten um bereit zu sein die eingehenden Nachrichten zu empfangen
2. sich gegenseitig synchronisieren.

Im ersteren Fall kann das Gerät direkt übertragen, während im zweiten Fall das Gerät warten muss bis das Ziel-Gerät bereit ist Daten zu empfangen. Die zu ergreifenden Maßnahmen, um die Geräte zu synchronisieren übersteigen den Rahmen des IEEE 802.15.4-Standards und werden somit den oberen Schichten überlassen.

2.4 MAC-Schicht Dienstleistungen

Die MAC-Schicht stellt Daten und Management-Dienste für die obere Schicht (normalerweise für den ZigBee Network Layer) zur Verfügung. Jeder Dienst wird durch einen Satz von Primitiven spezifiziert, die in vier allgemeine Typen klassifiziert werden (siehe Abbildung 2.6) können und je nach Bedarf ein/alle der vier Typen benutzen.

- *Anfrage* (engl. Request). Es wird von der oberen Schicht aufgerufen um einen bestimmten Dienst anzufordern.
- *Hinweis* (engl. Indication). Es wird durch die MAC-Schicht erzeugt und zu der oberen Schicht geschickt, um das Auftreten eines Ereignisses in Bezug auf einen bestimmten Dienst zu benachrichtigen.
- *Rückantwort* (engl. Response). Es wird von der oberen Schicht aufgerufen, um eine Prozedur zu vervollständigen, die vorher von einer Hinweis-Primitive initiiert wurde.
- *Bestätigung* (engl. Confirm). Es wird durch die MAC-Schicht erzeugt und zu der oberen Schicht gerichtet, um die Ergebnisse der zuvor gestellten Dienstanfragen auszugeben.

2.4.1 Datendienst

Der Datendienst besteht aus einem Haupt-Dienst, der nur die Anfrage-, Bestätigung- und Hinweis-Primitiven nutzt. Die Primitive „DATA.request“ wird durch die obere Schicht aufgerufen, um eine Nachricht zu einem anderen Gerät zu senden. Das Ergebnis einer angefoderten Übertragung (einer früheren „DATA.request“ Primitive), wird durch die „DATA.confirm“ Primitive, die den Status der Übertragung trägt, zu der oberen Schicht zurück gemeldet. Die „DATA.indication“ Primitive entspricht einer „Empfangen“ Primitive: Es wird von der MAC-Schicht erzeugt, die den Empfang einer Nachricht, in der

physikalischen Schicht, an die obere Schicht weiterleitet. Abbildung 2.7 veranschaulicht den Ablauf von Nachrichten und Primitiven, die während eines Datenaustausches zwischen zwei Knoten statt finden.

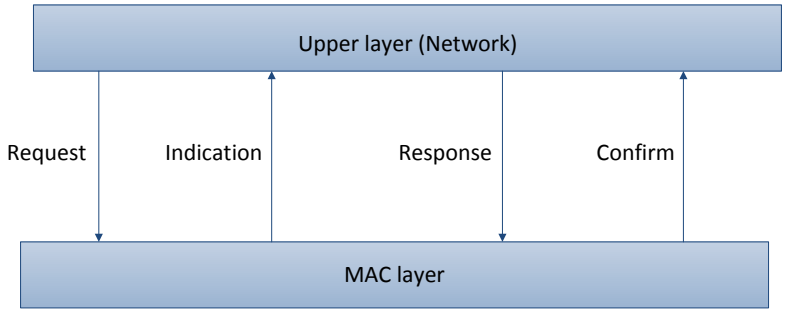

Abbildung 2.6: Datendienst [7]

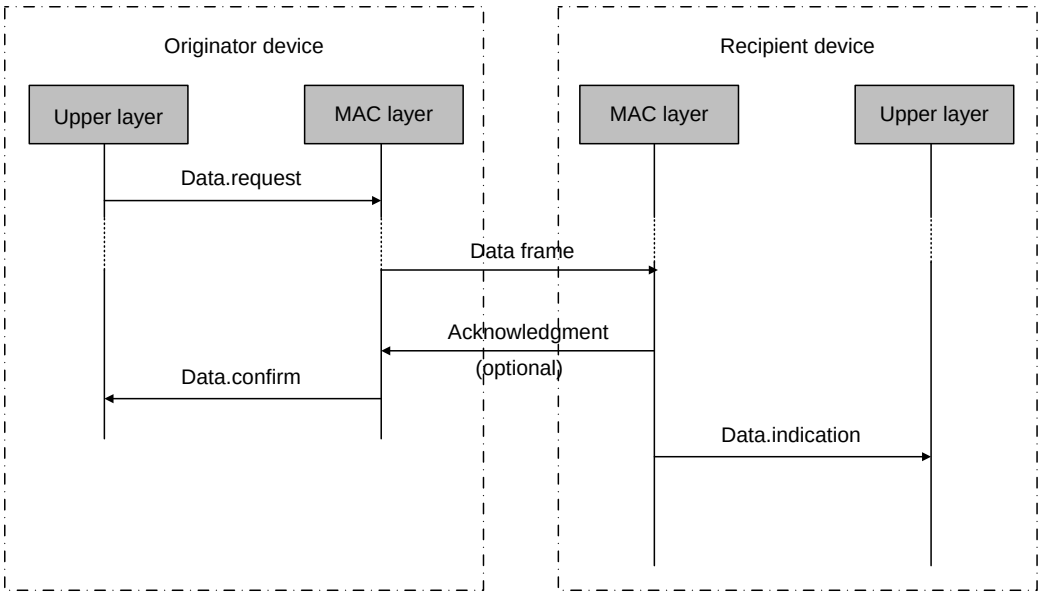

Abbildung 2.7: Implementation Dataservice [7]

2.4.2 Service-Management

Unter Service-Management in der MAC-Schicht zählen:

- Funktionalitäten für die PAN Initialisierung
- Verbinden/Trennen von Geräten
- Erkennung von bestehenden PANs
- Benutzung einiger Funktionen der MAC-Schicht von anderen Diensten.

Die wichtigsten Dienstleistungen sind in der Tabelle 2.2 zusammengefasst. Das Symbol X sagt aus, dass ein Dienst einer Primitive (P) entstpicht. Das Symbol O sagt aus, dass die Primitive (P) optional für RFDs ist.

Als Beispiel wird das Protokoll und die Funktionalitäten des ASSOCIATE-Diensts beschrieben. Dieses Service wird von einem Gerät aufgerufen, das mit einem PAN verbunden werden möchte, und das bereits durch einen SCAN-Dienst identifiziert ist. Die „ASSOCIATE.request“ Primitive nimmt als Parameter die PAN-Kennung, die Koordinatoradresse und die 64-Bit erweiterte IEEE-Adresse des Geräts. Die Primitive sendet eine ASSOCIATE Anforderungsnachricht an einen Koordinator. Da das Vereinigungsverfahren für Beacon-Enabeld Netzwerke gemeint ist, wird die ASSOCIATE Anforderungsnachricht währen des CAP gesendet.

Der Koordinator bestätigt sofort den Empfang der Vereinigungsnachricht. Allerdings bedeutet dies keine Bestätigung, dass der Antrag angenommen wurde. Auf der Koordinator Seite wird die ASSOCIATE Anforderungsnachricht an die oberen Schichten des Koordinator Protokollstacks weiter geleitet, wo die Entscheidung über den Verbindungsanforderungsbefehl tatsächlich übergeben wird. Wird der Antrag angenommen, wählt der Koordinator eine kurze 16-Bit-Adresse, die das Gerät später an Stelle der 64-Bit erweiterten IEEE-Adresse benutzen kann. Die oberen Schichten des Koordinators rufen die „ASSOCIATE.response“ Primitive von der Koordinator MAC-Schicht auf. Diese Primitive nimmt als Parameter die 64-Bit-Adresse des Geräts, die neue kurze 16-Bit Adresse und den Status der Anfrage. Die Primitive erzeugt damit eine ASSOCIATE Befehlsantwort, welche dem Gerät, das eine Vereinigung mit indirekter Übertragung anfordert, gesendet wird. Der Befehl ist in der Liste anstehender Nachrichten im Koordinator gespeichert. Die MAC-Schicht des Geräts gibt automatisch eine Datenanforderungsnachricht an den Koordinator, nach einer vorgegebenen Zeit nach der Bestätigung des Verbindungsanforderungsbefehls aus.

Zu beachten ist, dass es zwei Möglichkeiten gibt für ein Gerät, um eine anstehende Datennachricht an den Koordinator anzufordern:

- durch den POLL-Service
- automatisch nach einer vorgegebenen Zeit, nach der Bestätigung eines früheren Anfragebefehls.

Name	Request	Indication	Response	Confirm	Description
ASSOCIATE	X	O	O	X	Anfrage zur Verbindung eines neuen Gerätes zu einem bestehenden PAN.
DISASSOCIATE	X	X		X	Verlassen der PAN.
BEACON-NOTIFY		X			Gibt das empfangene Beacon an die obere Schicht ab.
GET	X			X	Erlaubt Verbindungen eines neuen Gerätes zu einem PAN zu.
GTS	O	O		O	Anfrage der GTS an den Koordinator.
SCAN	X			X	Sucht nach aktiven PANs.
COMM-STATUS		X			Benachrichtigt die obere Schicht über den Status einer Transaktion mit einer Request-Primitive.
SET	X			X	Stellt die Parameter der MAC-Schicht.
START	O			O	Startet eine PAN und beginnt mit dem Senden eines Beacons. Kann auch für die Geräteerkennung verwendet werden.
POLL	X			X	Fordert die anstehende Nachrichten für den Koordinator.

Tabelle 2.2: Tabelle: Haupt Service-Management der MAC-Schicht[7]

Der Koordinator sendet dann die Verbindungsbefehlsantwort an das Gerät. Beim Empfangen der Befehlsnachricht, gibt die MAC-Schicht des Geräts eine „ASSOCIATE.confirm“ Primitiv, während die MAC-Schicht des Koordinator eine „COMM-STATUS.Indication“ Primitive angibt, um die obere Schicht zu informieren, dass das Verbindungprotokoll entweder mit Erfolg oder mit einem Fehlercode abgeschlossen ist. Das Verbindunsprotokoll wird in der Abbildung 2.8 gezeigt.

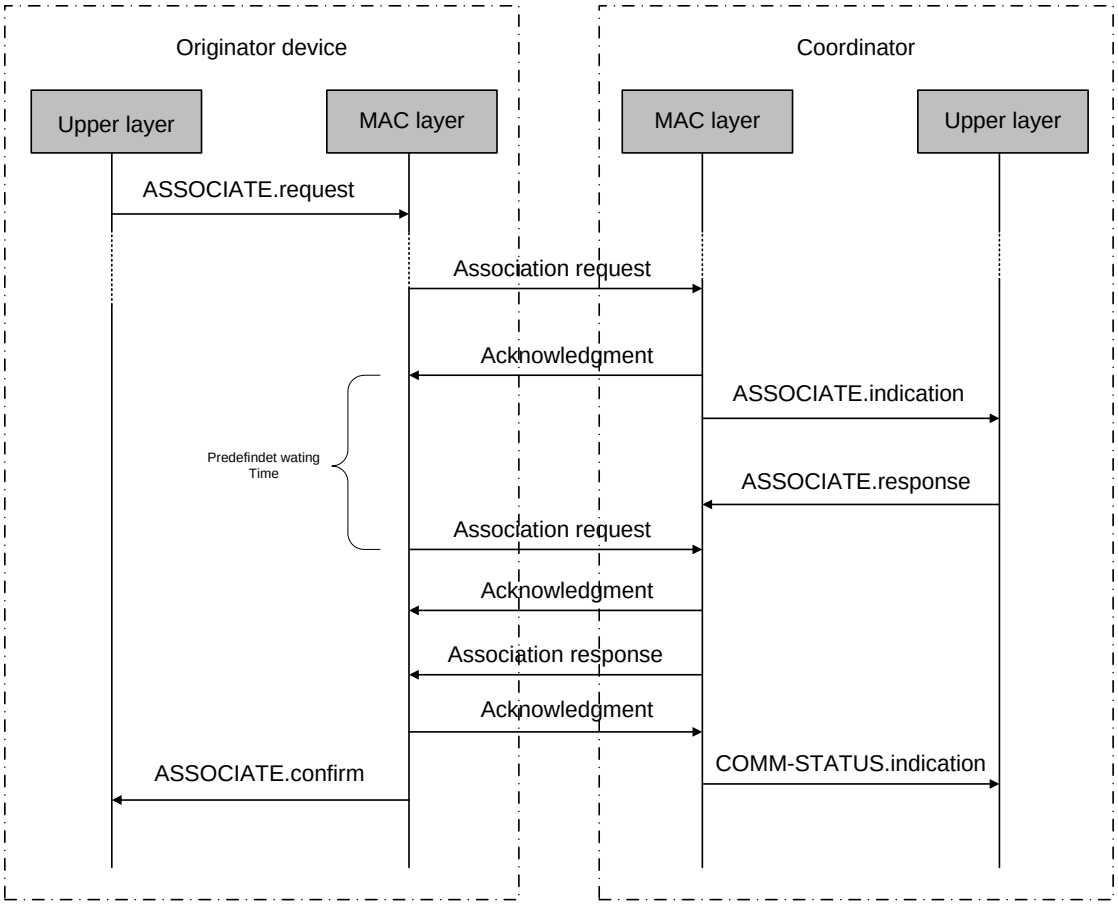

Abbildung 2.8: ASSOCIATE-Protokoll

2.5 Sicherheit

Der IEEE 802.15.4 MAC-Schicht stellt eine grundlegende Unterstützung für Sicherheit und lässt erweiterte Sicherheits-Features (z.B. Schlüssel- und Geräte-Authentifizierung) zu den oberen Schichten zu. Alle Sicherheitsdienste basieren auf symmetrischen Schlüsseln und

benutzen Schlüssel, die von den höheren Schichten bereitgestellt worden sind. Die MAC-Schicht Sicherheitsdienste gehen davon aus, dass die Schlüssel durch die oberen Schichten erzeugt, übertragen und gespeichert werden. Beachten Sie auch, dass die Sicherheits-Features der MAC-Schicht optional sind und die Anwendungen können entscheiden wann und welche Funktionalität sie verwenden. Die Sicherheitsdienste von der MAC-Schicht sind Zugriffskontrolle, Verschlüsselung, Frame Richtigkeit und sequenzielle Neuheit, die wie folgt beschrieben werden kann:

- *Access Control.* Access Control ermöglicht einem Gerät, eine Liste der Geräte (die so genannte Access Control List oder ACL), mit denen sie aktiviert werden und damit kommunizieren, aufzubewahren. Wenn dieser Dienst aktiviert ist, hält jedes Gerät im PAN eine eigene ACL und verwirft alle eingehenden Pakete von den Geräten, die nicht in der ACL enthalten sind.
- *Data Encryption.* Data Encryption nutzt symmetrische Kryptographie um Daten vor Fremden ohne den kryptographischen Schlüssel zu schützen. Der Schlüssel kann durch eine Gruppe von Geräten (in der Regel als Standard-Schlüssel gespeichert) oder zwischen zwei Peers (gespeichert in einen einzelnen ACL-Eintrag) geteilt werden. Datenverschlüsselung kann auf Daten, Befehle und Signaturen von Nutzlasten zur Verfügung gestellt werden.
- *Frame Integrity.* Frame Integrity verwendet einen Integrity Code, um Daten vor Modifikation von Parteien ohne die kryptographischen Schlüssel zu schützen und sicherzustellen, dass die Daten von einem Gerät mit dem kryptographischen Schlüssel kommen. Wie in der Datenverschlüsselung kann der Schlüssel von einer Gruppe oder von Paaren von Geräten geteilt werden. Die Integrität kann auf Daten, Signaturen und Kommando-Frames vorgesehen sein.
- *Sequential Freshness.* Sequential Freshness ordnet die Inputframesequenzen um sicherzustellen welcher Inputframe neu dazukommt.

Kapitel 3

Beschreibung des System Controllers

Dieses Kapitel beschreibt den Aufbau des System-Controllers und die Funktionsweise einiger wichtiger Module. Die Abbildung 3.1 zeigt einen einfachen Aufbau des von Infineon entwickelten Syscon (System Controller), der als Steuereinheit des Transceivers dient.

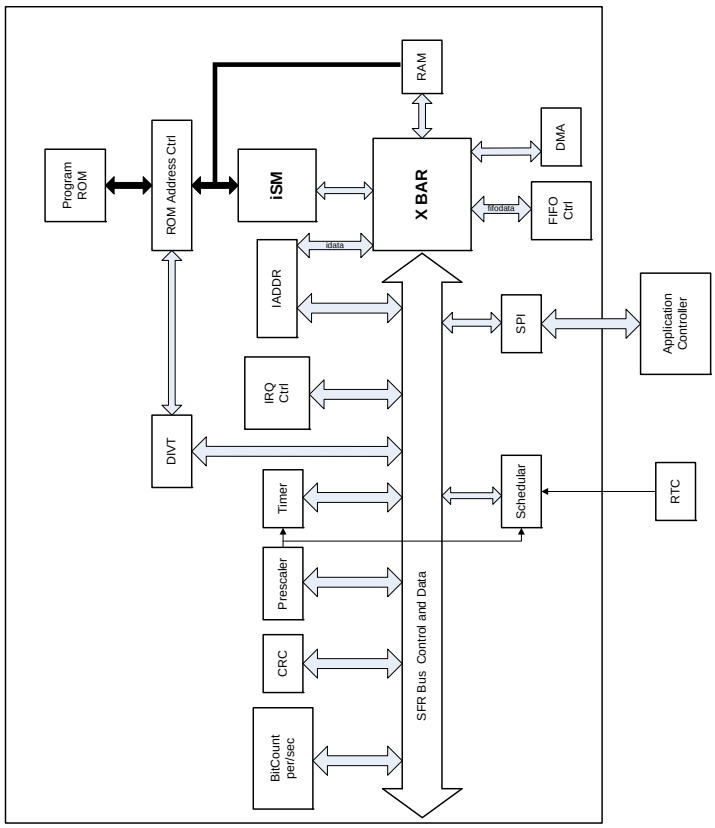

Abbildung 3.1: Systemcontroller

3.1 iSM-Modul

Das intelligente State-Machine-Modul ist ein sehr einfacher Controller und ist Kern der Systemsteuereinrichtung. Um die erforderliche Gesamtfunktionalität zu unterstützen, enthält die Architektur mehrere Peripheriemodule welche die iSM umschließen. Die iSM ist ein kleiner Prozessor mit anwendungsspezifischem Befehlssatz (engl. Application-specific instruction-set processor ASIP) mit ca. 3-5k Toren, ohne Peripheriegeräte und implementiert einen RISC-Befehlssatz. Die iSM-Datenpfad-Bitbreite, die für TRX (Transceiver Einheit) verwendet wird, ist 16 Bit groß. Die Programmiersprache ist C Code mit limitierem C-Subset und kann mittels einem speziellen Compiler in Maschinencode übersetzt werden.[9]

3.2 DIVT-Modul

Die dynamische Interrupt-Vektortabelle (DIVT) stellt die Mittel für die Zuordnung bestimmter Startpunkte des Programmcodes zur Verfügung bereit um Service-Routinen zu unterbrechen. Zusammen mit dem IrqCtrl-Modul stellt es den Interrupt-Mechanismus dar. Abbildung 3.2 soll das Verständnis der (oben) beschriebenen Konzepte erleichtern.[9]

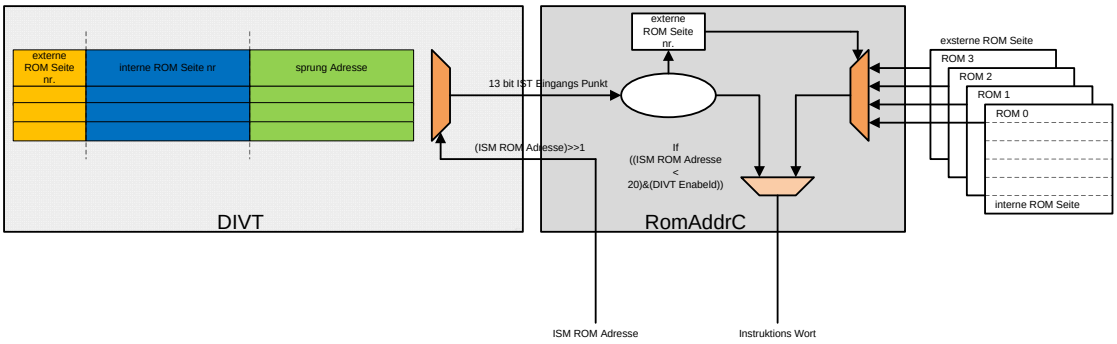

Abbildung 3.2: DIVT [9]

Die DIVT enthält eine Tabelle aller Startadressen der Interrupt-Service-Routinen (Interruptvektoren). Jede Funktion innerhalb des Programmadressraums kann in der Interrupt-Service-Routine dargestellt werden. Die Tabelle enthält 10 Einträge, wobei der Erste und der Letzte eine besondere Bedeutung aufweisen. Der erste Eintrag ist der Resetvektor, der letzte hingegen ist ein Fehlervektor. Der Resetvektor wird nach dem Einschalten verwendet. Der Fehlervektor wird nach einem Unterlauf des ISM Programmzähler-Stacks verwendet (rückgesetzt/gesetzt), d.h. wenn mehr Rückführungsaktionen (RET) als Funk-

tionsaufrufe (CALL) ausgeführt wurden. Die DIVT bietet dem SFR Zugriffsmöglichkeiten um alle Einträge zu konfigurieren. Bevor das DIVT Modul verwendet werden kann, müssen zumindest Resetvektor und der Fehlervektor definiert werden. Jeder Eintrag in der DIVT besteht aus 13 Bits und setzt sich wie folgt zusammen:[9]

- Bits 12 und 11 definieren die Anzahl der externen ROM Seiten. Nach Adressierung des jeweiligen Eintrags werden diese 2 Bits in der RomAddrC gespeichert. Der RomAddrC leitet anschließend eine Adressierungsanfrage auf die angegebene externe ROM Seite
- Bits 10 ... 7 definieren die interne ROM Seite. Es gibt einen interne Seitenregister in der iSM, der die interne ROM Seitenzahl beinhaltet
- Bits 6 ... 0 repräsentieren die Sprungadresse innerhalb der internen ROM Seite

Der 13 Bit breite Interruptvektor ist eine ROM-Programmadresse und muss in die entsprechenden Operationscodes umgewandelt werden um von der iSM ausgeführt werden zu können. Diese Umwandlung wird in der RomAddrC abgearbeitet. Jeder Sprung besteht aus zwei Anweisungen:

- Die erste Anweisung setzt die interne ROM Seite. Dafür wird nur ein Taktzyklus benötigt
- Die zweite Anweisung springt in die angegebene Interrupt-Service-Routine. Darüber hinaus wird die äußere physische ROM Seite außerhalb der iSM vom RomAddrC-Modul eingeschaltet. Dieser Sprung benötigt zwei Taktzyklen um ausgeführt zu werden

Die gesamte Interrupt-Latenzzeit beträgt 3 Taktzyklen plus eine zusätzliche Latenz von 1 Zyklus, die das Interrupt-Signal benötigt um von dem Interrupt-Controller erkannt zu werden. Die Abbildung 3.3 zeigt die Code Realisierung der DIVT.

```
// configure DIVT
          SFR_RESVEC = (int)&startup;            // set reset vector to startup routine
          SFR_DIVT0  = (int)&isr0;               // interrupt 0 service routine
          SFR_ERRVEC = (int)&error;              // set error vector to error routine

          // enable DIVT module
          SFR_SCC = SFR_SCC | DIVTEN;
```

Abbildung 3.3: DIVT Code Beispiel

3.3 IrqCtrl-Modul

Das Interrupt-Controller-Modul implementiert die vielseitige interne Funktionalität für die iSM. Ein Ereignis wird durch eine steigende Flanke an einer Interrupt-Leitung signalisiert. Der IrqCtrl erkennt Unterbrechungen und setzt das Aufwachsignal (Wakeup Signal) für die iSM. Die Interrupt-Service-Routinen, deren Funktionszeiger in der DIVT gespeichert ist, sind einen bestimmten Interrupt zugeordnet. Abbildung 3.4 soll das Verständnis der oben beschriebenen Konzepte erleichtern.[9]

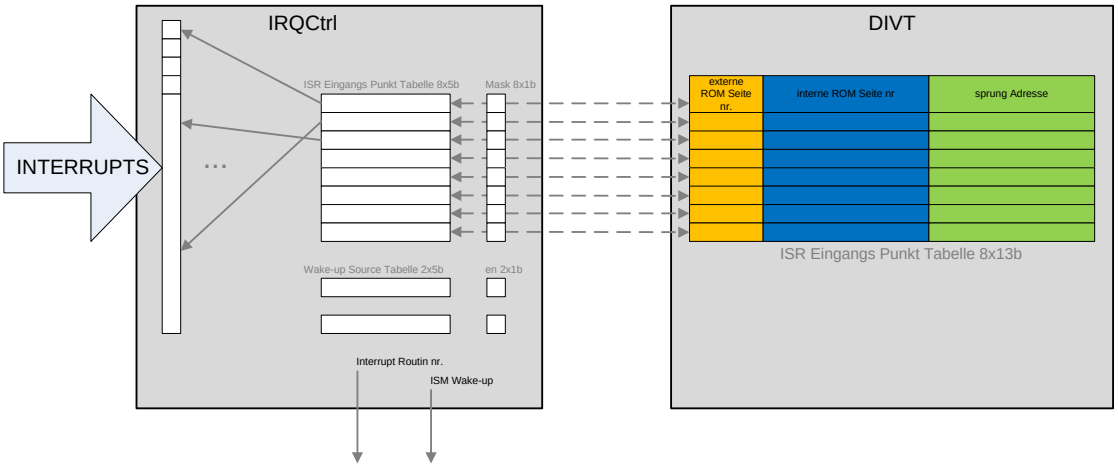

Abbildung 3.4: IRQCtrl [9]

Die Architektur unterstützt maximal 32 Interrupts.

- 28 werden von anderen Hardware-Signalen und anderen Modulen ausgelöst
- 4 sind durch Firmware (Traps) ausgelöst

Jedes Interrupt-Signal hat ein Statusbit im IrqCtrl Modul. Diese 32 Status Bits werden in zwei SFR Status-Registern, IS0 und IS1, organisiert. Statusbits entsprechender Hardware-Signale werden gesetzt, wenn der zugehörig Interrupt aufgetreten ist (steigende Flanke). Ein Status-Bit bleibt gesetzt, bis der Interrupt belegt wird. Es wird dann automatisch gelöscht. Statusbits können explizit durch Firmware (aber nicht von der Firmware eingestellt) gelöscht werden, mit Ausnahme der Firmware-trap Bits FWT0IS ... FWT3IS, die nur durch die Firmware eingestellt werden können. Es gibt 8 Interrupt-Ebenen und 8 zugehörige Interrupt-Quelle Register. Ein Interrupt kann an einer zugehörigen Interrupt-Ebene zugeordnet werden, indem sie die entsprechende Interrupt-Quelle Register SFR ISRCx auf den Interrupt Index 0 ... 31 einstellt. Es gibt eine Maske SFR IM, die jede

der 8-Interrupt-Service-Routinen deaktivieren kann. Die Reset und Error-Vektoren sind immer aktiviert. Die Abbildung 3.5 zeigt die Code Realisierung des IRQCtrl.[9]

```
Example1:
          // configure SPI interrupt

          SFR_ISRC0 = SPIISRC;        // set irq 0 source to SPI interrupt
          SFR_IM    = 0x0001;         // unmask irq source 0
          SFR_IS0   = 0;              // clear SPI interrupt status flag

          <macro to enter POWERDOWN state>;               // this macro disables iSM; the iSM will
                                  // reset the program counter when re-enabled

Example2:
          // configure SPI wakeup event

          SFR_WUPSRC  = WUPSRC0EN | BCISRC;   // enable and set wakeup event source
                                          // to SPI interrupt
          SFR_IS0 = 0;                        // clear SPI interrupt status flag

          <macro to enter WAIT state>;                            // user-defined macro: iSM is halted
                                          // until next wakeup event
          SFR_IS0 = 0;                        // clear SPI interrupt status flag
```

Abbildung 3.5: IRQCtrl Code Beispiel

3.4 RomAddrC-Modul

Der ROM-Adressen-Controller (RomAddrC) regelt den Zugang zum Programmspeicherplatz. Die iSM gibt eine Adresse an, die zu einem der angeschlossenen physikalischen ROM Blöcke (externen ROM Seiten) oder der DIVT geroutet wird. Danach bekommt die iSM wieder ein Befehlswort, welches aus dem ROM (wenn angesprochen) oder einem Befehlswort, das unter Verwendung des Interrupt-Vektors aus der DIVT gelesen wurde, stammt.

Die externe physikalische ROM Seite wird gemäß dem Seitenregister ausgewählt und innerhalb des RomAddrC-Moduls gehalten. Dieses Seitenregister ist bei „power on“, „Reset“ auf Null gesetzt und soll danach geändert werden, wenn die DIVT adressiert ist. Beachten Sie, dass bei Verwendung einer statischen Interrupt Vektor Tabelle die im ROM Seite 0 (d.h. DIVT deaktiviert) gespeichert ist, das ROM Seitenregister nie geändert wird.

Wenn die DIVT aktiviert und angesprochen ist, gibt die DIVT einen 13 Bit Wert mit der externen physischen Adresse der Seite und der interne Seitenadresse zurück, die verwendet wird um zwei Anweisungen zu erzeugen. Aus der Sicht des iSM besteht jeder

Interrupt-Vektor aus 2 Instruktionen. Dies bedeutet, dass sich die ISM die erste Anweisung beim Programm-Zähler holt "(Interrupt-Level) * 2", dann den zweiten Befehl aus dem Programm-Zähler "(Interrupt-Level) * 2 + 1". Das heißt, für jeden ersten und zweiten Befehl von einem Interrupt-Vektor wird derselbe Eintrag in der DIVT vom RomAddrC adressiert.

Jeder Sprung in eine Interrupt-Service-Routine (einschließlich Reset und Error-Routine) belegt zwei Adressen im Programmspeicher, obwohl es nur einen Eintrag im DIVT Modul gibt. Diese Beziehung wird durch das RomAddrC Modul gelöst. Die RomAddrC schafft die zwei Anweisungen für die iSM aus allen 13-Bit-Eingaben, die durch die DIVT geliefert wurden. Die erste Anweisung (SET RPAGE) bei einer geraden Programmadresse setzt die interne ROM Seite. Die zweite Anweisung (JMP) an einer ungeraden Programmadresse springt in die angegebene Interrupt-Service-Routine.

Außerdem wird bei Empfang des 13 Bit Werts aus der DIVT das externe ROM geschaltet und die Ausführung der Interrupt-Service-Routine wird in die gewünschten externen ROM Seite geleitet. Das RomAddrC-Modul aktualisiert seine Seitenregister während der Ausführung der 2 Instruktionen.[9]

3.5 DMA-Modul

Das direkte Speicherzugriff-Modul (DMA) wird verwendet um selbstständig Kopien von Daten von einem Ort zum anderen zu machen. Darum wird der Datenspeicherbus benutzt um auf das RAM und alle SFRs einschließlich der SFRs der System-Controller und den SFRs des vorderen Endes zu zugreifen. Das DMA-Modul liest ein Datenwort von der Quelladresse in einem Zyklus und in dem nächsten Zyklus schreibt er das Datenwort an die Zieladresse.[9]

Das DMA-Modul unterstützt drei Betriebsmodi neben dem deaktivierten Zustand.

- Ausgelöster Kopier-Modus: In diesem Modus verwendet das DMA-Modul einen von vielen verfügbaren Interrupt-Signalen als Auslöser.
- Zähler Modus: In diesem Modus wartet das DMA-Modul nicht auf einen Trigger. Es führt die gewünschte Anzahl der Kopiervorgänge möglichst schnell (Burst) aus. Wenn alle Kopiervorgänge abgeschlossen sind, schaltet sich das Modul selber aus.
- Ausgelöstem Zähler Modus: Ist eine Kombination der beiden vorherigen Betriebsmodi. Wenn eine bestimmte Anzahl an Kopiervorgängen durchgeführt ist, schaltet sich das Modul selber ab.

Die Abbildung 3.6 zeigt die Code Realisierung des DMA.

```
// configure DMA0 module to copy i data words from the SPI module
          // to the FIFO. Copy operation is done upon SPI trigger.

          SFR_DMA0SRC = (int)&SFR_SPIDATA     // DMA0 source addr is SFR_SPIDATA
          SFR_DMA0DST = (int)&SFR_FIFODATA    // DMA0 destination addr is SFR_FIFODATA
          SFR_DMA0TRIGSRC = SPIISRC;   // enable SPI trigger

          // set DMA0 control register (DMA0C):
          // - number of data words to copy is i
          // - DMA mode is triggered count
          // - disable source address increment
          // - disable destination address increment
          SFR_DMA0C = SPITRGSRC | DMA0MODE_3 | DMA0INCSRC_0 | DMA0INCDST_0 | i;

          // set wakeup event source to DMA interrupt
          SFR_WUPSRC = DMAISRC;

          // wait until DMA module has copied the specified number of data words
          <macro to enter WAIT state>;      // user-defined macro: iSM is halted until next wakeup event
          SFR_IS1 = 0; // clear DMA interrupt status flag
```

Abbildung 3.6: DMA Code Beispiel

3.6 FIFOctrl

Das FIFO-Controller-Modul(FIFOCTRL) ist ein Adressenübersetzter-Modul der eine FIFO in den physischen RAM-Speicher nachbildet. Das FIFOCtrl-Modul verwaltet Datenbytes und speichert sie in 16 Bit RAM Worte. Zwei Bytes werden in ein 16 Bit Wort gespeichert. Die Schreib- und Lese-Zeiger werden intern vom FIFOCtrl-Modul verwaltet. Speicherstelle und die Größe eines FIFOs werden via SFRs so konfiguriert, dass ein FIFO Kontext, im RAM gespeichert werden kann. Somit kann der FIFOCtrl mehrere virtuelle FIFOs, einer nach dem anderen erledigen. Da die FIFO-Daten im RAM gespeichert sind, können sie auch direkt durch die iSM oder DMA-Module als 16 Bit Daten abgerufen werden. Die FIFO Lese- und Schreibzeiger bleiben in diesem Fall unberührt.[9]

Wenn ein Datenspeicher-Bus-Master auf das FIFO-Datenregister zugreift um auf das FIFO zu schreiben oder zu lesen, ersetzt der Speicheradresskontroller die Bus-Adresse, durch die berechnete Adresse aus dem Lese- oder Schreibzeiger von FIFOctrl. Der FIFOctrl signalisiert den Speicheradresskontroller, ob das obere oder untere Byte momentan in den aktuellen 16 Bit RAM Wort adressiert ist.

Für Schreibzugriffe speichert das FIFOctrl-Modul intern zuletzt geschriebene Bytes für jeden geraden Schreibzugriff. Beim nächsten ungeraden Schreibzugriff gibt das FIFOctrl Modul das zuletzt geschriebenen Byte heraus, so dass der Speicheradresskontroller ein 16 Bit Wort bildet, um es in das RAM zu schreiben.

3.7 Prescaler-Modul

Das Prescaler-Modul erzeugt einen vor-skalierten Taktgeber, welcher verwendet wird um das Timer-Module abzutasten. Der Systemtakt ist eine Taktquelle für das Prescaler-Modul. Der Taktausgang kann individuell für jeden Timer mit den 4 Bit Feld Prescale ausgewählt werden. Folgenden Skalierungsfaktoren können aus der nächsten Tabelle ausgewählt werden:

Prescaler Wert	Skalierungsfaktor
0000	1:1
0001	1:2
0010	1:4
0011	1:8
0100	1:16
0101	1:32
0110	1:64
0111	1:128
1000	1:256
1001	1:512
1010	1:1024
1011	1:2048
1100	1:4096
1101	1:8192
1110	1:16384
1111	1:32768

Tabelle 3.1: Prescaler Tabelle

Der Taktausgang des Timers kann individuell für jeden Timer aktiviert werden, indem das Prescale-Einschalt-Bit belegt wird. Die Abbildung 3.7 zeigt die Code Realisierung des Prescaler.

Code Example

```
// timer0 prescaler 1:4
// enable timer0
SFR_T0C = PRESCAL0EN_1 | 0x0002;
```

Abbildung 3.7: Prescaler Code Beispiel

3.8 Timer-Modul

Das Timer-Modul implementiert einen zyklischen Abwärtszähler. Nachdem ein neuer Preload-Wert in den Preload-Timer-Register geschrieben wird, startet der Timer mit der Dekrementierung des internen Zählerwerts mit der steigenden Flanke des Prescaler-Moduls. Nachdem der interne Zähler null erreicht hat wird er mit dem Preload-Wert wieder geladen und der Timer beginnt von neuem.[9]

Das Timer-Modul produziert für Hardware Taktausgangssignale, die als Taktquelle von anderen Modulen in dem System verwendet werden kann. Derzeit wird der Timer 0 Taktausgangssignals als Baudratentakt vom RXTXsync Modul verwendet. Die Timer-Taktsignale haben ein 50% Tastverhältnis beginnend mit einem hohen logisch Wert, wenn der Timer-Zähler dem Preload-Wert hat.

Der Timer-Zähler und die Preload-Wert sind beide 16 Bit breite Register. Zusammen mit dem Prescaler des Timer-Module stellen sie eine zyklische Eventgeneration mit einer Zykluszeit von bis zu ca. 98s wenn die Systemtaktfrequenz 21,94872 MHz ist. Die Abbildung 3.8 zeigt die Code realisirung des Timers.

```
SFR_T0PRE = 400;                                  // set timer pre-load value to 400
SFR_T0C = PRESCAL0EN_1 | 0x0002;       // configure prescaler:
                                                                  // timer0 prescaler 1:4
                                                                  // enable timer0

// timer 0 is now started

timer_value = SFR_T0VAL;                // read timer 0 value
```

Abbildung 3.8: Timer Code Beispiel

3.9 Scheduler-Modul

Das Scheduler-Modul wird als Durchführungs-Scheduler verwendet. Es ist ein 16-Bit Aufwärtszähler, wird mit jeder steigenden Flanke am Takteingang inkrementiert und ist mit einem Faktor zwischen 1:1 und 1:217 vorskaliert. Der Prescaler Faktor ist im SCHEDPRESCAL Registers gemäß der folgender Tabelle konfiguriert. Die Länge des Scheduler wird über den SCHEDLEN Register konfiguriert. Der aktuelle Scheduler Zählerwert kann vom SCHEDVAL Register ausgelesen werden.[9]

Scheduler Prescaler Wert	Skalierungsfaktor
00000	1:1
00001	1:2
00010	1:4
00011	1:8
00100	1:16
00101	1:32
00110	1:64
00111	1:128
01000	1:256
01001	1:512
01010	1:1024
01011	1:2048
01100	1:4096
01101	1:8192
01110	1:16384
01111	1:32768
10000	1:65536
10001	1:131072

Tabelle 3.2: Prescaler Tabelle

Mit der maximalen Scheduler Länge von 2^{16} -1 zusammen mit dem größten Pre-Skalierungsfaktor von $1{:}2^{17}$ beträgt die maximale Zykluszeit ca. 8,59 * 109 Eingangs Taktzyklen. Der Scheduler wird von der RTC getaktet, die bei einer niedrigeren Frequenz als Systemtakt läuft. Unter der Annahme einer 100 kHz RTC-Frequenz; stellt das Scheduler-Modul eine Zykluszeit von bis zu ca. 24 Stunden.

Das Temporärerstartzeit-Register(SCHEDSTAMP) beinhaltet den Startzeitwert des nächsten geplanten Events. Nur ein Startzeitwert kann planmäßig zur selben Zeit ausgeführt werden. Wenn es mehr als eine Durchführung im Scheduler(Ablaufplan) gibt, muss der Startzeitwert-Register aktualisiert werden um zu wissen, wann der nächste Startzeitwert fällig ist.

Das Scheduler-Modul erzeugt einen positiven Impuls an dem Interrupt-Ausgangssignal, wenn der interne Zähler den geplante Starzeitwert erreicht. Zusätzlich wird ein Impuls auf den Power-on-Ausgang generiert. Dieses Signal wird von den System-Management-Modul (SYSMGMT) verwendet, um das System von einem der Energiesparfunktions-Zuständen in einen aktiven Zustand umzuschalten. Die Abbildung 3.9 zeigt die Code Realisierung des Schedulers.[9]

```
SFR_SCHEDLEN   = 20;                            // scheduler length is 20 ticks
         SFR_SCHEDSTAMP = 15;                            // next scheduled time stamp
         SFR_ISRC0    = SCHEDISRC;               // set irq source to scheduler interrupt
         SFR_SCHEDC = SCHEDEN_1 | 0x0002;        // scheduler prescaler 1:4
                                                                    // enable scheduler


         SFR_RTCC = RTCEN_1;                     // enable RTC module

         // scheduler is now started

         scheduler_value = SFR_SCHEDVAL;         // read scheduler value
```

Abbildung 3.9: Code Beispiel Scheduler

3.10 BitCnt-Modul

BitCnt: Das Bit-Zähler-Modul (BitCnt) zählt die empfangenen oder übertragenen Datenbits und führt eine Seriell / Parallel-Umwandlung. Der Bit-Zähler-Wert kann aus dem BitCouter Register gelesen werden. Es kann auch zur Offset-Korrektur, des Zählerwerts beschrieben werden. Zusätzlich ist der Bit-Zählerwert ein Hardware-Signal das an das Feld Erfassungsmodul (FieldCap) verschickt wird, wo sie als ein Referenz-Bit-Index für das Positions-Feld verwendet wird.

3.11 SPI

Für die Kommunikation zwischen dem Transceiver (TRX) und dem Applikations Controller (μC) stehen zwei Schnittstellen zur Verfügung.[9]

SPI: Der TRX bietet eine SPI Slave Schnittstelle zur Konfiguration, Steuerung und zum Austausch von Daten Paketen.

Interrupt Pin: Der TRX verwendet die Interrupt-Pin um den μC bei bestimmten Ereignissen wie der Empfang von Daten Paketen mitzuteilen.

3.11.1 SPI Slave Schnittstelle

Die SPI-Slave-Modul implementiert eine häufig verwendete 4-Draht-SPI-Schnittstelle mithilfe der folgenden Signale:

- MTSR: Master sendent, Slave empfängt (auch MOSI: Master out, Slave in)
- MRST: Master empfängt, Slave sendent (auch MOSI: Master in, Slave out)
- SCLK: SPI Data Clock
- CS: SPI chip select (active low)

Das detaillierte SPI-Timing ist in nachfolgenden Abbildung dargestellt. Während der Ruhephase ist der SCLK Pegel hoch. Die Datenbits werden von den SPI-Sender auf die führenden SCLK Flanke verschoben und wird an den SPI-Empfänger auf der nachlaufenden SCLK Flanke abgetastet. SPI Befehle und Daten werden immer zuerst MSB codiert. Der Chip-Select-Signal (CS) ist auf einem niedrigen Pegel aktiv. Wenn CS hoch ist das SPI-Modul resetet. Dies ist in Abbildung 3.10 verdeuitlicht.

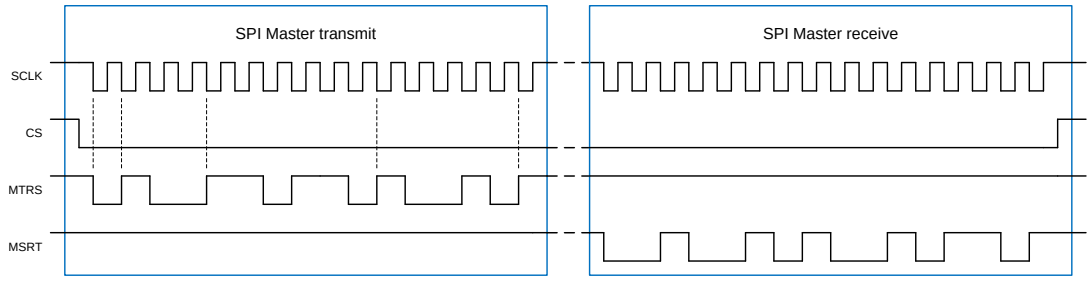

Abbildung 3.10: SPI Timing [9]

Es steht auch ein optionaler Programmier Pin (PROG EN) zur Verfügung. Allerdings ist dieser Pin nicht von der normalen SPI Kommunikation abhängig. Er wird nur verwendet um den eingebauten Protokoll Prozessor des Smart Transceiver zu programmieren und muss immer niedrig gehalten werden, außer während der Programmierphase.

3.11.2 SPI Befehlsformate

Der Smart Transceiver liefert einen Befehl basiertes SPI Kommunikations-Schema zum μC. Diese Schnittstelle ermöglicht eine zusätzliche Abstraktionsschicht, welche den uμCC ermöglicht mit dem Transceiver durch umfassende Befehle zu kommunizieren. Deshalb braucht der Benutzer nicht zu adressieren und das auf niedriger Ebene Spezial Funktion Register (SFR) direkt zu konfigurieren.

Ein Überblick der allgemeinen Befehlsformate sowie der FIFO-Puffer Befehle (Datenpuffer für aktive HF-Übertragung) wird in der nächsten Abbildung dargestellt.

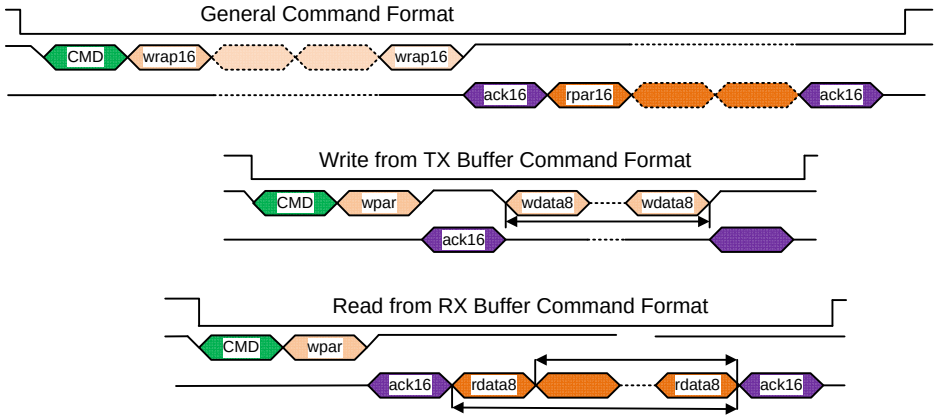

Abbildung 3.11: SPI Format [9]

Das allgemeine Befehls Format besteht aus den folgenden Bereichen:

- 8 Bit Befehlswort: CMD8
- 8 Bit Anzahl von Schreibparameter: wpar8; (Messbereich: 0 ... 255)
- 16 Bit Schreibparameter: wpar16;
- 16 Bit Acknowledge-Feld (ACK1)
- 16 Bit Leseparameter: rpar16;
- 16 Bit Acknowledge-Feld (ACK2)

Die Anzahl der Schreib-und Lese-Parameter sind befehlsspezifisch. Eine ausführliche Code Realisierung wird in der Abbildung 3.12 dargestellt.

```
          // configure SPI module
          SFR_SPIMODE         = CMD;              // set SPI module to cmd mode
          SFR_ISRC0 = SPIISRC;                    // set irq 0 source to SPI interrupt
          SFR_IM    = 0x0001;           // unmask irq source 0

Now wait for SPI interrupt: command byte received (upper byte of command word).
          <wait for SPI interrupt>      // this is implementation dependent
          cmd = SFR_SPIDATA;            // get command byte
          SFR_SPIMODE = MODE16;                   // next SPI word is 16-bit parameter

Now wait for SPI interrupt: command write parameter count received (lower byte of command word)
          <wait for SPI interrupt>      // this is implementation dependent

Now wait for SPI interrupt: first 16-bit command parameter received.
          <wait for SPI interrupt>      // this is implementation dependent
          wpar0 = SFR_SPIDATA;                    // read first 16-bit write parameter

Now wait for SPI interrupt: second 16-bit command parameter received.
          <wait for SPI interrupt>      // this is implementation dependent
          wpar1 = SFR_SPIDATA;                    // read second 16-bit write parameter
          SFR_SPIMODE   = MODE8;                  // next SPI word is 8-bit data
          SFR_CMDPARCNT = 2;            // send two data bytes, and then generate ACK
          SFR_SPIDATA   = data0;                  // write first 8-bit read data
                                                            // note: this has to be done before the
                                                            // current transfer finishes

Now wait for SPI interrupt: first ACK sent (command write ACK).
          <wait for SPI interrupt>      // this is implementation dependent
          SFR_SPIDATA = data1;                    // write second 8-bit read data
                                                            // note: this has to be done before the
                                                            // current transfer finishes

Now wait for SPI interrupt: first 8-bit read data sent.
          <wait for SPI interrupt>      // this is implementation dependent

Now wait for SPI interrupt: second 8-bit read data sent.
          <wait for SPI interrupt>      // this is implementation dependent

Now wait for SPI interrupt: second ACK sent (command read ACK).
          <wait for SPI interrupt>  // this is implementation dependent
```

Abbildung 3.12: Code Beispiel SPI

Kapitel 4

ROM Paging

4.1 ROM Paging

Als Paging bezeichnet man im deutschen Sprachraum die Methode der Speicherverwaltung per Seitenadressierung durch Betriebssysteme.[4]

4.1.1 Allgemeine Funktionsweise

Man unterscheidet beim Paging logische und physische Adressen. Die Organisation des Hauptspeichers wird im logischen Adressraum beschrieben. Der wirkliche Arbeitsspeicher ist durch den physischen Adressraum gegeben.[12]

Viele unterschiedliche Architekturen besitzen noch einen Speichercontroller, der zwischen dem Arbeitsspeicher und der CPU geschaltet ist, um so weitere Adressimplementierung durchführen zu können, wie Zerlegung der einzelnen Speicherbänke und das Einblenden des Speichers von externen Geräten. Diese zusätzlichen Adress Realisierungen sind nicht Bestand des Pagings.[12]

„*Als Seiten(Pages) bezeichnet man die im logischen Adressraum gleich großen ROMs. Die Unterteilung des physischen Adressraums ist auf die gleiche Weise gestaltet, nur dass man die einzelnen Stücke, Kacheln(Frames) nennt. Eine Seite und der zugehörige Kacheln haben i.d.R die gleiche Größe; alternative Techniken implementieren Paging in der Art, dass die Seitengröße einem ganzzahligen Vielfachen der Größe der Seitenrahmen entspricht - auch hierbei kommt es dadurch nicht zur Fragmentierung des Hauptspeichers. Um Seiten und Seitenrahmen einander zuordnen zu können, wird eine Seitentabelle verwendet. Es existiert für jeden Prozess eine eigene Seitentabelle. Der logische Adressraum ist in der Regel zusammenhängend, während im physischen Adressraum die logisch benachbarten Seiten in weit voneinander entfernt liegenden Seitenrahmen abgelegt werden können. Die Reihenfolge der Seitenrahmen ist beliebig.*“[12]

4.1.2 Paging über die DIVT

Diese Architektur enthält mehrere physikalische ROM Seiten zu je 2048 Anweisungen, die direkt von der iSM durch sein Programm Speicher-Adressierungs-Output adressierbar sind. Diese ROM-Seiten werden „externe“ ROM Seiten oder ”2k”ROM Seiten genannt. Die Umschaltung zwischen diesen externen ROM Seiten wird explizit mit den DIVT Modul ausgeführt. Dies kann entweder durch das Springen in der Vektor Tabelle durchgeführt werden oder indem sie in Power-Down zum Aufwachen auf einen Interrupt gezwungen wird. Es kann nicht durch Anweisungen in der iSM durchgeführt werden. Jede externe physikalische ROM Seite wird weiter in „interne“ Seiten mit 128 Instruktionen durch den Compiler partitioniert. Diese ROM Seiten werden „interne“ ROM-Seiten genannt. Dieser Mechanismus wird vollständig durch den Compiler gesteuert.

Da das Springen in eine Funktion, die sich auf einer anderen Page befindet, durch die iSM nicht möglich war. Das Problem wurde durch die DIVT gelöst. Die indirekte Adresse der Funktion wird als Wake-up Punkt, in der DIVT gespeichert. Siehe folgende Abbildungen.

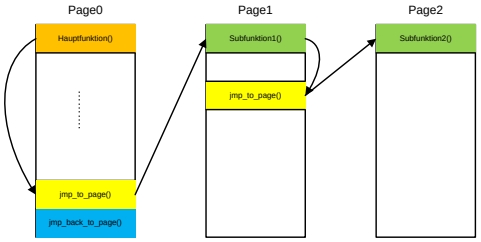

Abbildung 4.1: Paging JMP TO Beispiel

```
void cmd_txi_handler()
{
          jumpaddress = (int)&cmd_txi;          //save the iaddr of function

          jmp_to_page0();                       //call jmp function
}

void jmp_to_page0();
{
          SFR_DIVTC = DIVTEN_1;                 //DIVT couter enabled
          SFR_IM = 0x01;                        //set mask
          SFR_ISRC0 = FWT0ISRC;                 //defin interrupt source
          SFR_DIVT0 = jumpaddress;              //save the iaddr of function to the DIVT
          SFR_IS1 = FWT0IS;                     //manually activate the interrupt
          SLEEP();                              //go in Sleep mode
}
```

Abbildung 4.2: JMP Mechanismus Code Beispiel

Indem der Interrupt manuell gleich nach dem scharf stellen der Interrupt-Quelle ausgelöst wurde, ist das System aus dem kurzen Schlaf aufgewacht. Die indirekete Adresse, die als Wake-up-Punkt vorher definiert wurde, kann jetzt ausgeführt werden. Ein Beispiel solch einer JMP Funktion ist in der Abbildung 4.1 und 4.2 dargestellt.

Wenn wir im Vorfeld im wissen, dass ein Rücksprung auf die ursprüngliche Page erforderlich ist, muss eine Return-Adresse definiert werden. Diese Methode wird sehr oft verwendet, da wir oft eine Hauptfunktion (Page0) haben, die eine Subfunktion (Page1) aufruft um spezifische Berechnungen zu erledigen. Die Subfunktionen wurden oft universell gehalten und sind mit einem kleinen Feature versehen (siehe Abbildung 4.3 und 4.4).

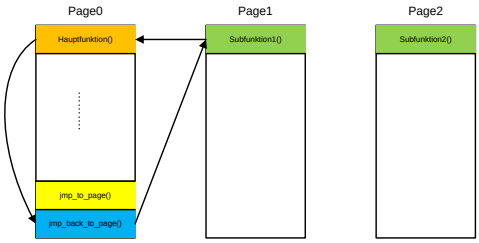

Abbildung 4.3: Paging JMP BACK Beispiel

```
.
.
.
.
//Feature at the End of Subfunction
if (returnaddress>0x0000)
        {
                jmp_back_to_page();
        }

}


void jmp_back_to_page()
{
        SFR_DIVTC = DIVTEN_1;
        SFR_IM = 0x01;
        SFR_ISRC0 = FWT0ISRC;
        SFR_DIVT0 = returnaddress;
        SFR_IS1 = FWT0IS;
        SLEEP();
}
```

Abbildung 4.4: JMP BACK Mechanismus Code Beispiel

4.2 Paging Simulation

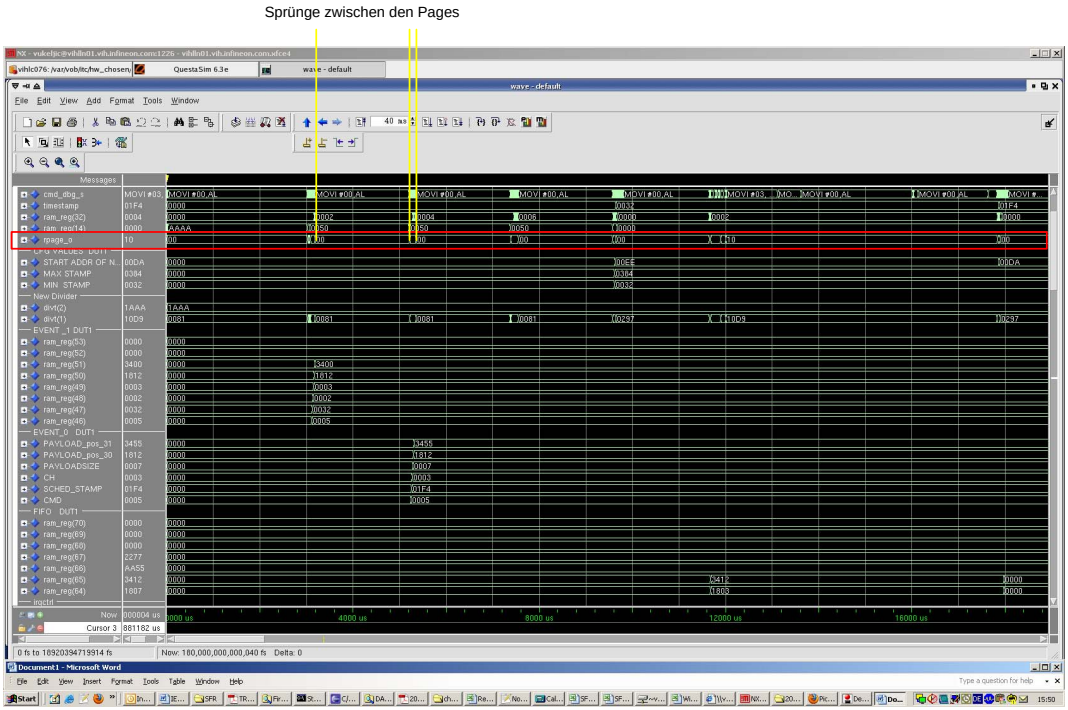

Abbildung 4.5: Paging in der Simulation

Aus dem Simulationsbild wird deutlich gemacht, wie wichtig das Paging für dieses System ist. Durch das Springen über die DIVT haben sich auch noch einige Vorteile ergeben und zwar, dass der Stackpointer immer wieder gelöscht wird wenn ein JMP passiert. Es muss nicht mehr geachtet werden, dass man sich in Funktionsaufrufen verstrickt. Der Nachteil des ganzen ist, dass man nicht zu der Stelle zurückkehren kann, von der man gesprungen ist. Es wird etwas mehr an Code-Anweisungen als nötig verbraucht.

Kapitel 5

CCA Detektion

„Clear Channel Assessment (engl. „Beurteilung der Kanalfreiheit“) ist die Bezeichnung der B-MAC-Entwickler für Trägerprüfung. Diese Fähigkeit zu beurteilen, ob das Kommunikationsmedium gerade frei und damit für eigene Datensendungen verfügbar ist, ist für B-MAC als Vertreter des Ansatzes Carrier Sense Multiple Access ein zentrales Anliegen.“[10]

„B-MAC verwendet ein Rauschunterdrückungsverfahren, um Trägerprüfung und Signalempfang zuverlässiger zu machen. Schaltet der Funkempfänger eines Sensorknotens auf Empfang, so wird er zwangsläufig die allgegenwärtige elektromagnetische Hintergrundstrahlung als Rauschen empfangen. Dieses Rauschen ist nicht überall gleich, sondern variiert mit der Umgebung und Umweltfaktoren wie nahen Gewittern. B-MAC setzt daher eine dynamische Rauschunterdrückung ein, die sich diesen Faktoren ständig anpasst. Dazu misst es das Hintergrundrauschen und ermittelt unter Einbeziehung zuvor gemessener Werte einen Rauschfilter, der nachfolgend empfangene Signale bereinigt. Zur Trägerprüfung hört B-MAC den Kanal fünfmal hintereinander kurz ab und sucht in diesen Proben nach Ausreißern, die sich von der Signalenergie des Hintergrundrauschens deutlich abheben; wird ein Ausreißer gefunden, gilt der Kanal als belegt.“[10]

5.1 CCA Konzept in WSN

Der CCA-Wert wird in Netzwerken als eine Anzeige für die Verfügbarkeit eines Drahtes oder für die Verfügbarkeit eines/r bestimmten Kanal/Frequenz im Falle eines Wireless Sensor Netzwerk verwendet. Wenn kein Knoten den Kanal verwendet, beginnt ein Knoten in der Regel sofort eine Nachricht zu senden. Der andere Fall ist, dass der Kanal von einem anderen Knoten durch das Senden einiger Nachrichten gesperrt ist. In diesem Fall wird der CCA-Wert erhöht und die anderen Knoten werden erneut versuchen, auf einen freien Kanal zu hören. Gelingt dies für einige Zeit nicht, verwendet der Knoten eine zufällig erstellte Pause mit der Hoffnung, dass der Kanal nach dieser Zeit frei wird.

Der CCA-Wert wird für das Verständnis einer Überlastung des Netzwerks in Bezug auf Angriffe, erläutert. Überlastungen treten in Fällen der gescheiterten Wiederholungen, einen freien Kanal zu bekommen, auf. Solche Staus sind ein ganz normales Ergebnis in einem Netzwerk. Zum Beispiel, wie in der Abbildung 5.1 dargestellt ist, könnte es zum Stau am Knoten3 kommen, wenn zwei (Knoten1 und Knoten2) Knoten Daten alle 30ms austauschen. Wenn es so aufkommt, wird der CCA Wert von Knoten3 durch Störungen von Knoten1 und 2 erhöht werden.[8]

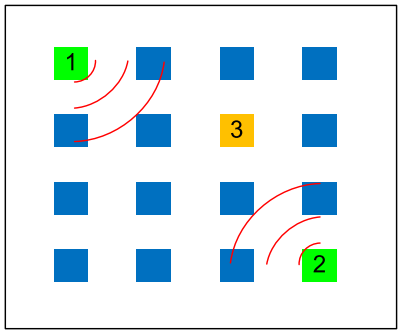

Abbildung 5.1: Zwei Knoten (Knoten 1 und Knoten 2) tauschen Daten aus und erhöhen den CCA Wert des Knotens 3

Solche Überlastungen können zu Problemen in einem Netzwerk führen. Es gibt eine Menge von Quellen für diese Störungen, einschließlich jene die Mikrowellen benutzen, um das Netwerk zu stören oder um in das Netzwerk einzubrechen (siehe Abbildung 5.2). Wenn der Knoten4 andauernd Pakete an jemandem versendet, wird Knoten1 und 3 nicht in der Lage sein, ihre eigenen Pakete, wegen dem Knoten4, zu senden. Wenn der Knoten4 ein Angreifer ist, wird die Kommunikation zwischen anderen Knoten schwierig, wegen der Überlastung vom Knoten4. Unter diesen Umständen können wir den CCA-Wert vom Knoten erfahren. Zwei Knoten 1 und 3 sind durch den Knoten 4 angegriffen, und so wird ihr CCA-Wert erhöht.[8]

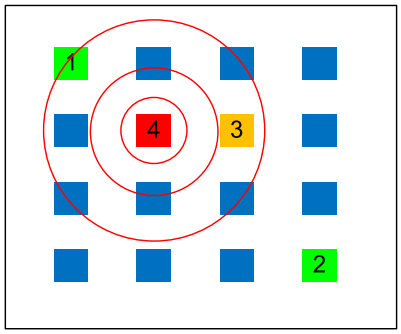

Abbildung 5.2: Zwei Knoten werde durch ein anderen Knoten angegriffen

5.2 RSSI

Bei RSSI, das für Received Signal Strength Indicator steht, handelt es sich um den Empfangsfeldstärke-Anzeiger einer Mobilstation. Der RSSI-Wert ist ein wichtiger Indikator für die Funkstrecke und die empfangene Feldstärke. Er ist abhängig von der Sendeleistung (EIRP) der Funkstation und der Entfernung zwischen Funkstation und Mobilstation (siehe Abbildung 5.3).[5]

Der RSSI-Wert wird in den verschiedensten Mobilfunksystemen und auch in anderen funkbasierten Übertragungssystemen von den angeschlossenen Stationen gemessen und bei starken Feldstärkeeinbrüchen wird er für die Umschaltung auf einen anderen Funkkanal genutzt.[5]

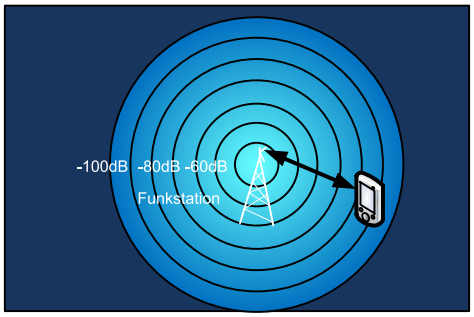

Abbildung 5.3: RSSI [8]

Die RSSI-Technik wird in WLANs nach 802.11 und in Bluetooth eingesetzt. Der Feldstärkewert umfasst 1 Byte und dient für die Anzeige der Feldstärke der empfangenen Hotspots.[5]

5.3 Implementierung der CCA Funktion

Die CCA Detektion beginnt mit dem Kommando CMD_CCA. Dabei werden zwei Parameter an den Transceiver über die SPI versendet. Die Parameter beinhalten Informationen über den Schwellwert (engl. threshold), den RSSI-Wert und die Beobachtungszeit (engl. observation time). Das Kommando wird auf der ersten Page ermittelt und auf den Sprung in die CCA Funktion vorbereitet.

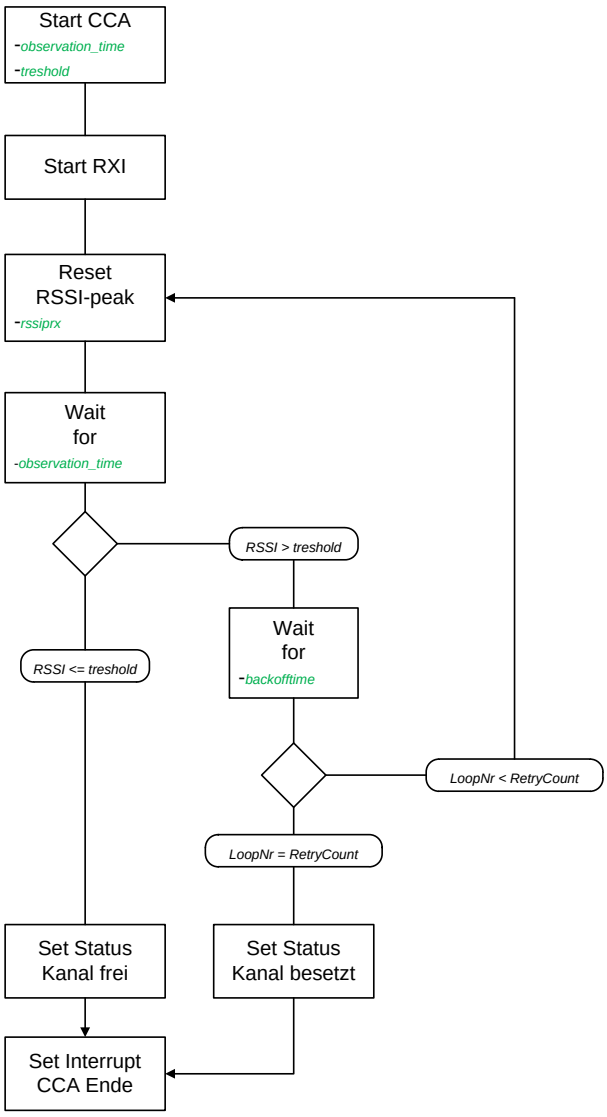

Abbildung 5.4: CCA Programstruktur

In der CCA Funktion wird der Transceiver auf das sofortige Empfangen eingeschaltet (falls nicht aktiv), um den RSSI-Wert aus den RSSI-SFR-Register zu entnehmen (rssiprx sihe Abbildung 5.4). Der RSSI-Wert wird nach der angegeben Beobachtungzeit aus dem Register zwischengespeichert und in einer Schleife immer wirder mit dem Schwellwert verglichen.

Wenn der RSSI-Wert kleiner als der Schwellwert ist, wird die CCA Detektion beendet und ein Statusbit, das für „Kanal frei“ steht, gesetzt. Im anderen Fall, wenn der RSSI-Wert nie den Schwellwert unterschreitet, kann angenommen werden, dass der Kanal belegt ist. Der Status „Kanal belegt“ wird ebenfalls gespeichet und kann dem System bescheit sagen, dass er mit dem Empfangen aufhören soll. Diese Erkenntnis kann eine Menge Energie einsparen oder den Transceiver überreden auf einen anderen freien Kanal zu wechseln.

Kapitel 6

Entwicklung des Schedulingsystems

Da sich diese Arbeit mit der Firmware Entwicklung für einen Smart Radio Transceiver beschäftigt und der Transceiver Chip noch in der Entwicklungsphase ist, geht man davon aus, dass sich die Software immer neuen Bedingungen/Veränderungen anpassen muss. Der Transceiver soll später mit einem Sensor verbunden und in ein Sensornetzwerk eingebunden werden. Die Firmware soll die einwandfreie Steuerung der Kommunikation zwischen den Sensornodes und das Ein/Austreten neuer Sensornodes ermöglichen. In diesem Kapitel wird auf die Programmstruktur des Scheduling-Moduls eingegangen. Des weiteren werden noch Simulationsergebnisse sowie die Funktion und das Einbinden des Scheduling-Moduls im System gezeigt. Als Firmware Entwicklungsumgebung dienen eine Eclipse IDE sowie zum Simulieren und Debuggen des Gesamtsystems dient QuestaSim von der Firma Mentor Graphics und teilweise auch ein FPGA Prototyp, der durch die Firma Infineon zugänglich gemacht wurde.

6.1 Scheduling Allgemein

„*Unter Scheduling (englisch für „Zeitplanerstellung“), auch Zeitablaufsteuerung genannt, versteht man das eine Menge von Tasks zeitlich so zu organisieren, dass temporale und Ressourcen-Constraints erfüllt sind. Bei der Regelung von mehreren Prozessen unterscheidet man dabei zwei Arten von Scheduling unterbrechende (preemptive) und nicht unterbrechende (non preemptive, auch kooperativ genannt). Nicht unterbrechende Scheduler führen den Prozess solange aus bis die den von sich aus wieder freigeben. Unterbrechbare Scheduler hat vordefinierte Zeitspannen bei dem der Prozess ausgeführt werden soll.*“[13]

„*Eines der großen Schwierigkeiten beim Scheduler ist, das man im Vorfeld nicht weiß, wie viele Ressourcen für einen Prozess gebraucht werden. Deswegen ist es sehr schwer eine optimale Planung zu erstellen, sodass der Scheduler dynamisch auf veränderbare Anforderungen reagieren muss.*“[13]

„Die Gestaltung des Plans nachdem das Programm ablaufen soll ist ein grundlegender Teil aller Computersysteme, bei denen die Ressource von mehreren Funktionen genutzt werden. Ablaufpläne werden in verschiedenen Bereichen benötigt und werden durch hochoptimierte Scheduler entwickelt. Scheduler werden aufgrund ihrer Funktionsweise als auch anhand der Einsatzgebiete unterschieden.“[13]

Beispielhaft für wichtige Einsatzgebiete für die hochoptimierte Scheduler entwickelt werden, sind folgende:[13]

- *„Der Prozess-Scheduler (dt.: Prozessverwaltung/Ressourcenzuteilung/Zeitplanung) ist Bestandteil von Betriebssystemen. Er ist für die faire Verwaltung von mehreren Prozessen zuständig, die auf einem Computer ausgeführt werden“*[13]
- *„Der Festplatten-Scheduler ist für die zeitliche Verwaltung von Schreib- und Leseaufträgen des Betriebssystemes an das Festplattenlaufwerk verantwortlich.“*[13]
- *„In Datenbankverwaltungssystemen verwaltet ein Transaktionsscheduler die Schreib- und Lesezugriffe der einzelnen Transaktionen auf die Daten um Verstöße gegen das ACID-Prinzip zur Einhaltung der Datenkonsistenz zu vermeiden“*[13]
- *„Beim Job-Scheduling geht es um die korrekte Ansteuerung von Jobs (Batchjobs, Programmstarts, etc.) in meist größeren IT-Umgebungen, die in zeitlichen und weiteren Abhängigkeiten untereinander stehen“*[13]

Abbildung 6.1 zeigt das Konzept der Zeitplanung im Drahtlosen Sensornetzwerk, das für diese Diplomarbeit benutzt wurde.

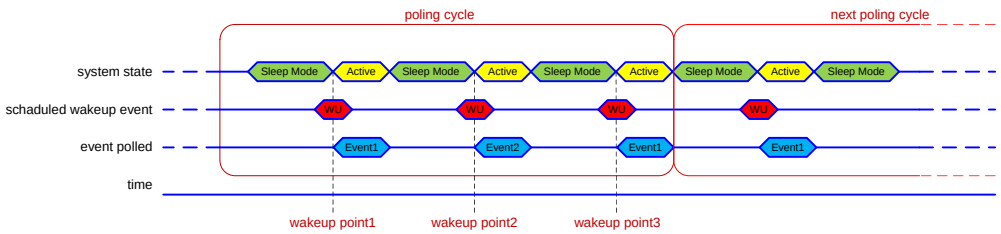

Abbildung 6.1: Scheduler Konzept [9]

6.2 Programmstruktur

Die Programmstruktur des Schedulingsystems wird in 4 Stufen unterteilt: Identifikation, Umleitung, Ausführung, Abschluss. Diese Unterteilung wird in der Abbildung 6.2 verdeutlicht.

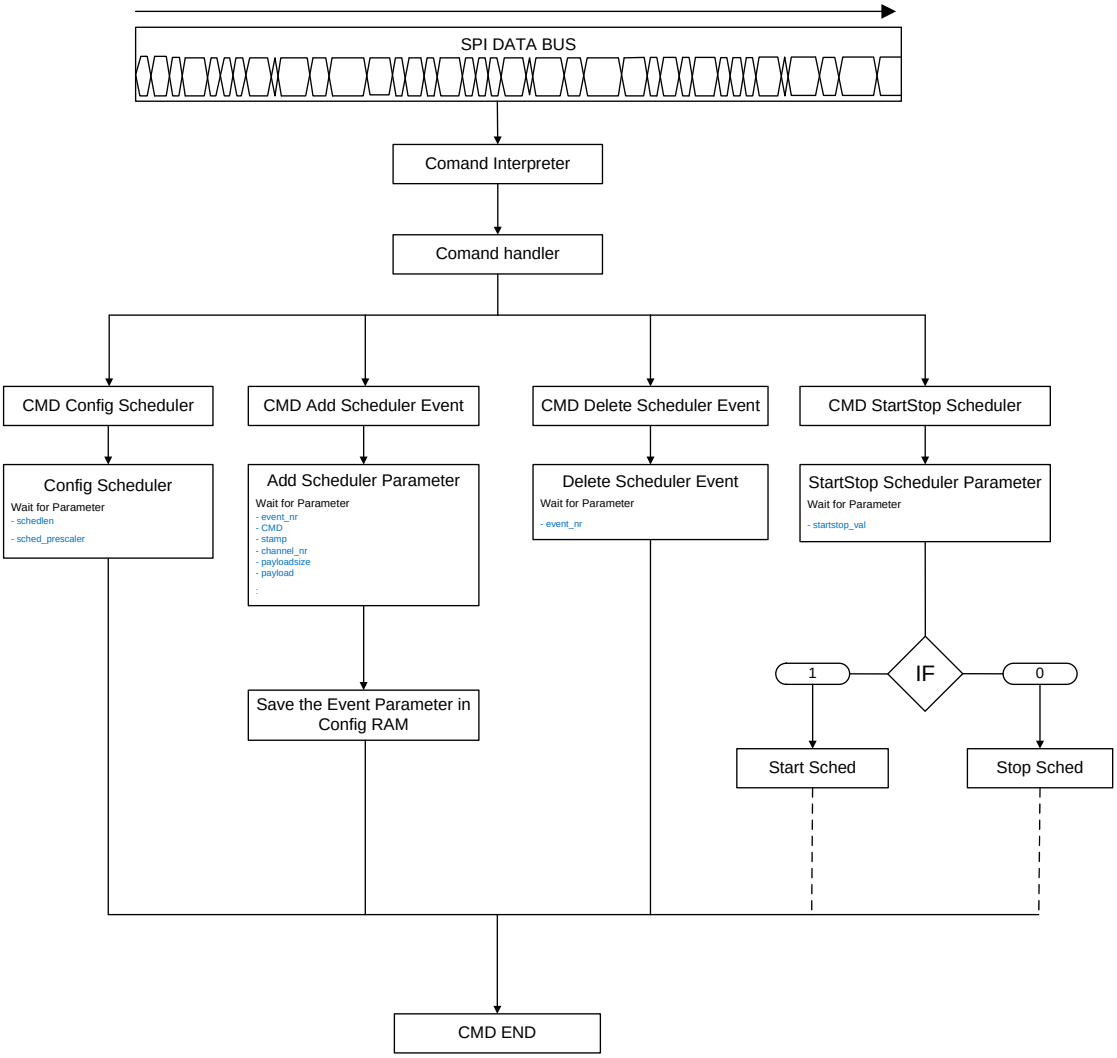

Abbildung 6.2: Programmstruktur Scheduler

Der **Command Interpreter** identifiziert nur gültige Kommandos, die von der SPI gelesen werden. Als gültige Kommandos gelten jene die im *cmd.h* vordefiniert sind. Damit

ist gewährleistet, dass keine anderen Codereihen als Kommando durchkommen. Die Funktionsweise ist mit einem Switch Case vergleichbar. Der Command Interpreter ist auf Page0 implementiert.

CMD END ist jene Funktion mit der jedes begonnene Kommando abgeschlossen wird. Die DIVT wird auf dem Command Interpreter scharf gestellt und das Programm geht in den Sleep-Mode.

Command Handler ist wie der Command Interpreter in der Page0 integriert. Jedes Kommando spricht eine bestimmte Funktion im Programm an und die Aufgabe des Handlers ist es die Verlinkung des Kommandoworts mit der Funktion, die sich auf einer beliebigen Page befindet zu machen. Der Scheduler allein besitzt folgende spezifische CMDs:

- cmd_conf_sched
- cmd_add_sched_param
- cmd_delete_sched
- cmd_startstop_sched

Config Scheduler ist die Funktion, die die Grundeinstellung am Scheduler Timer erledigt. **cmd_conf_sched** beinhaltet zwei Parameter, die vom Application Controller über die SPI gesendet wird.

- *schedlen* ist die max. Länge des Counters. Nachdem der max. Wert erreicht ist beginnt der Counter wieder von vorne zu zählen
- *sched prescaler* stellt den Prescalerwert im Verhältnis zur CPU Clock ein

Add Scheduler Parameter erstellt ein Scheduler Event. Dabei werden minimal 6 Parameter von der SPI gelesen und in CNFG RAM gespeichert.

- *event nr* markiert die Position, wo die Event Daten im CNFG RAM gespeichert werden sollen.
- *CDM* trägt das Codewort des Kommandos, das im Event gespeichert wird.
- *stamp* ist die Zählernummer bei dem der Scheduler aufwacht. Dessen Wert wird auch dazu benutzt um zu bestimmen, welcher Event als nächstes ausgeführt wird. Es wird immer der nächst größere Wert genommen.
- *channel nr* gibt den Wert des Frequenzbereichs an, auf dem empfangen oder gesendet werden soll.
- *payloadsize* gibt die Anzahl der Byts des gespeicherten Daten Pakets an.
- *payload* ist das Daten Paket, das übertragen werden soll.

Die folgende Abbildung zeigt das Erstellen der Events im CNFG RAM.

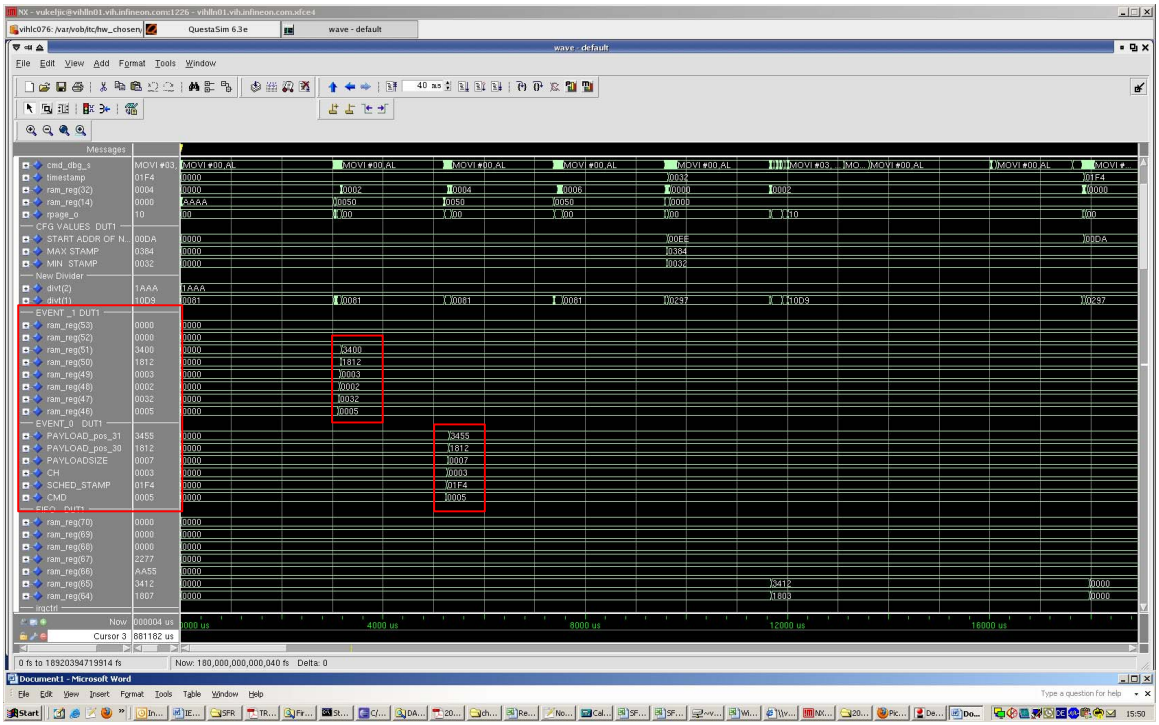

Abbildung 6.3: Simulation des Add Scheduler Parameters

Delete Scheduler Event Löscht die ganzen Event-Daten die mit der Event Nummer markiert wurden.

StartStop Scheduler Parameter ist die Haupt Funktion, die Scheduler Prozesse steuert. Dabei wird nur ein Parameter benötigt.

- start *startstop val*=1 damit wird der Scheduler Counter gestartet, die DIVT für den Wakeup Punkt gesetzt, und das System in den Sleep Modus versetzt.
- stop *startstop val*=0 damit wird der laufende Scheduler Counter gestoppt.

6.3 Starstop Scheduling Programmaufbau

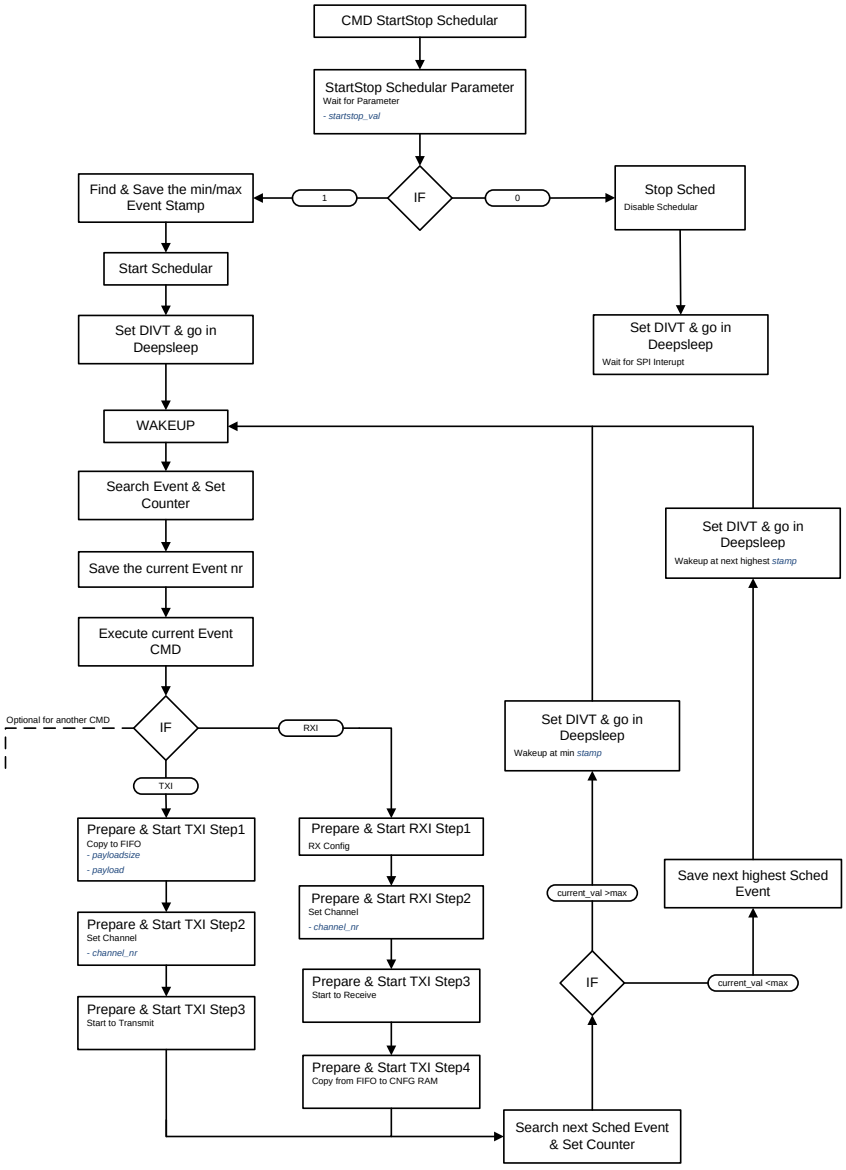

Abbildung 6.4: Startstop Programstruktur

Die Startstop Scheduler Funktion ist der Kern des Ablaufplanungssystems. Mit dem Komando CMD_STARSTOPSCHED, das über die SPI empfangen wurde, wird der Parameter startstop_val übergeben. Die Abbildung 6.4 zeigt den detaillierten Ablauf des Sartstop-Befehls. Abhängig davon, welchen Wert der starstop_val Parameter besitzt, wird das Self-Poling-System gestartet oder gestoppt.

- Stopt den Scheduler Timer wenn starstop_val = 0 ist
- Startet den Scheduler Timer wenn Startstop_val = 1 ist

6.3.1 Stop Funktion

Die Stop Funktion hält den Timer an und definiert gleich danach den nächsten Wakeup-Punkt in der DIVT, was zur Folge hat, dass das System in den Tiefschlaf versetzt wird und auf den nächsten Interrupt wartet. Beim Auftreten eines Interrupts wird primär auf ein Kommando, das über das SPI Interface kommt, gewartet (siehe Code Beispiel in Abbildung 6.5).

```
// Disabel Scheduler Timer

SFR_SCHEDC = SCHEDEN_0;        // set schedulercuounter 0



//thos function set the wakeup function address on DIVT for the next interrupt
//go in DeepSleep mode


// reset command interpreter
SFR_DIVT0 = (int)&cmddec_isr;
SFR_DIVT1  = (int)&sched_search_event_and_set_counter | BUGFIX_ACTUAL_PAGE;

//Temp. Fix: SPICMDISRC = index 14 ... is in IS0(14)0x4000
SFR_ISRC0 = 14;
SFR_ISRC1 = SCHEDISRC;

//Temp. Fix: SPICMDISRC = index 14 ... is in IS0(14)0x4000
BITCLR(SFR_IS0, 0x4000 | SCHEDIS);

REGSET(SFR_IM, IRQ_L0_MASK | IRQ_L1_MASK);


// for the case the device will not power down because the slave will be selected until the
power-down command is reached

DEEP_SLEEP_RTC_ON_RAM_ON();
```

Abbildung 6.5: Scheduler STOP Code Beispiel

6.3.2 Start Funktion

sched_min_max
Bevor der Scheduler Timer gestartet wird, werden vorerst alle in der CNFG RAM gespeicherten Events gescannt. Dabei wird der Event mit der niedrigsten und höchsten Startzeit (stamp) ermittelt. Die min. und max. Startzeit werden im CNFG RAM gespeichert um zu einem späteren Zeitpunkt verwendet zu werden (siehe Code Beispiel in Abbildung 6.6).

```
//save the min & max event_nr in cfgram
SFR_IADDR = CFG_RAM_STARTADDR + CFG_VAR_OFFSET_MIN_EVENT_STAMP;
SFR_IDATA = temp2;

SFR_IADDR = CFG_RAM_STARTADDR + CFG_VAR_OFFSET_MAX_EVENT_STAMP;
SFR_IDATA = temp3;
```

Abbildung 6.6: Speichert die min/max Starzeit aller gescannten Events

start_sched
Da defaultmäßig immer der Event mit der minimalen oder von der letzten nächstgrößeren Startzeit genommen und ausgeführt wird, muss vor dem Start ein Vergleichscheck stattfinden um herausfinden welcher vollzogen wird. Nach dem Check werden endlich die Timer Einstellungen vorgenommen und die Interrupt Quelle wird auf die Startzeit (**stamp**) scharfgemacht (siehe Code Beispiel in Abbildung 6.7).

```
SFR_SCHEDLEN   = schedlen;                    // scheduler length is 20 ticks
SFR_SCHEDSTAMP = temp1;                       // next scheduled time stamp

SFR_ISRC0    = SCHEDISRC;                     // set irq source to scheduler interrupt
SFR_SCHEDC = SCHEDEN_1 | sched_prescal;       // scheduler prescaler 1:4
                                              // enable scheduler
```

Abbildung 6.7: Startet den Scheduler Timer und stellt den Interrupt auf **stamp** scharf

Nachdem die Timer gestartet wurden, wird der Wakeup-Punkt in der DIVT eingestellt, ähnlich wie in Abbildung 6.5, und das System in den Tiefschlaf versetzt.

Die folgende Abbildung zeigt den Scheduler Timer in einer Simulation und die verschiedenen Startzeit Einstellungen.

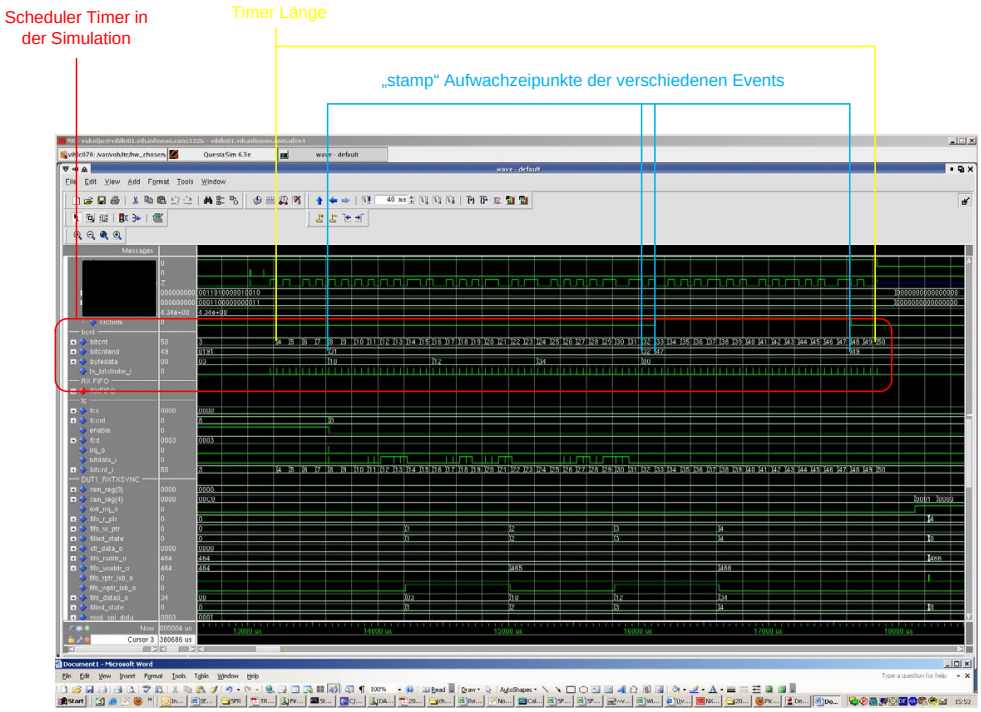

Abbildung 6.8: Scheduler Timer in der Simulation

Nun wird gewartet, bis ein Interrupt passiert und das System wieder aufwacht. Da in der Interruptpriorität als Erstes ein CMD vom SPI genommen wird und dann der Scheduler Interrupt, muss angenommen werden, dass die SPI Information eine höhere Bedeutung hat. Wenn nur ein Scheduler Interrupt kommt, wird das Programm zu der Funktion geleitet, die nach dem Aufwachen erledigt wird.

sched_search_event_and_set_counter
Da im Tiefschlaf alle temporären Variablen gelöscht wurden, muss das System erst herausfinden, welcher Event gerade ausgeführt werden soll. Diese Funktion speichert temporär den aktuellen Timerwert und vergleicht ihn mit allen Event-Startzeiten, die noch im CNFG RAM gespeichert geblieben sind (Abb. 6.9).

```
        // is the current  schedvalue
        current_schedval = SFR_SCHEDVAL;
```

Abbildung 6.9: Aktueller Schedulertimer Wert

sched_execute_current_event
Diese Funktion ist eine Art große Verzweigung wo die verschiedenen Kommandos (CMD) richtig zu ihrer Ausführung geleitet werden. Da das aktuelle Event vorher gefunden wurde, kann aus diesem Event das CMD entnommen werden und an die Verzweigung gebracht werden. In der folgenden Abbildung werden nur die Sende-Sofort(TXI) und Empfange-Sofort(RXI) Sequenzen gezeigt. Optional kann auch die CMD_END Sequenz eingeführt werden, um an diesem Punkt den Scheduler Zyklus zu beenden.

```
//Nr of Command that should be executed
SFR_IADDR = event_start_address + EVENT_OFFSET_VAR_CMD;
temp1 =  SFR_IDATA<<8 ;

//CMD_TXI = 0x0500
if (temp1 == CMD_TXI)
{
          sched_prepare_and_start_txi_step1();
          //jump to hal_txi
}
else if (temp1 == CMD_RXI)
{
          sched_prepare_and_start_rxi_step1();
          //Jump to hal_rxi...;
}
.
.
.
.
```

Abbildung 6.10: CMD Verzweigung

6.3.2.1 TXI Sequenz

In Step1 holt sich das System den Wert der Daten-Paket Größe und darauf folgend das Daten-Paket aus dem aktuellen Event im CNFG RAM. Alle diese Parameter werden im FIFO gespeichert und für das Senden vorbereitet (siehe Code Realisierung in Abbildung 6.11).

```
//temp1 = payloadsize  of current event
SFR_IADDR = event_start_address + EVENT_OFFSET_VAR_PAYLOADSIZE;
temp1 = SFR_IDATA;



//example:  if temp1 = 0x0003;
//(0x0003 +1):2^1 = 0x0002;
temp1 =(temp1 +1)>>1;

//Copy payload from event to TX-FIFO
for (i=0; i<temp1; i++)
{
          SFR_IADDR = event_start_address + EVENT_OFFSET_VAR_PAYLOAD + i;
          temp2 = SFR_IDATA;



          SFR_FIFO0DATA = temp2>>8;      //save 1rst HB
          SFR_FIFO0DATA = temp2;         //save 2nd LB


}
```

Abbildung 6.11: Laden des FIFOs mit Daten

In Step2 werden Kanalfrequenzen aus dem Event entnommen und der Transceiver wird auf das Senden vorbereitet, um beim Step3 letztendlich das Senden auszuführen.

6.3.2.2 RXI Sequenz

In diesem Abschnitt wird der Receiver auf das Empfangen vorbereitet, dabei werden die Kanalfrequenzen aus dem CNFG RAM herausgeholt und das Gerät auf das sofortige Empfangen geschaltet. Nachdem sich der Receiver mit dem Sender synchronisiert hat und alle Daten sauber empfangen hat, speichert er diese Daten aus dem FIFO in daen aktuelle Event. Die folgende Abbildung zeigt solch eine RXI Sequenz.

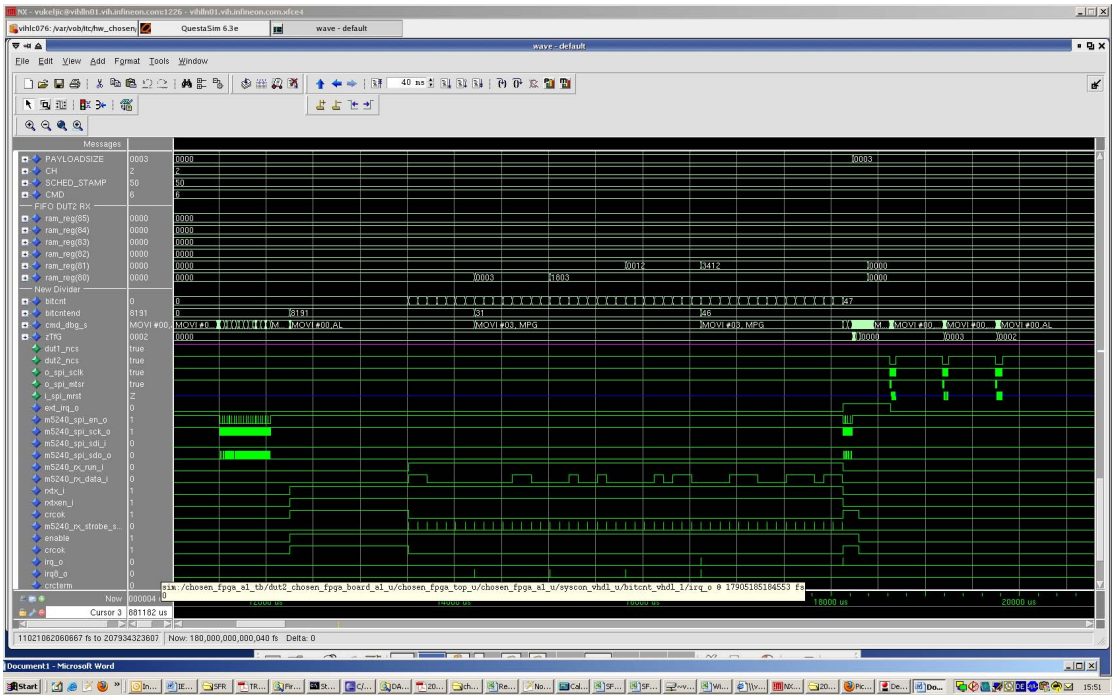

Abbildung 6.12: Simulation der RXI Sequenz

Nachdem die Sende-/Empfangs-Sequenz durchgeführt wurde, startet das System eine Suchfunktion nach dem nächsten auszuführenden Event und vergleicht dabei den aktuellen Timer-Wert (sched_val) mit allen anderen Event Startzeiten, die in CNFG RAM gespeichert sind. Wird bei dem Vergleich festgestellt, dass der aktuelle Wert größer als die max. Event Startzeit ist, setzt das System automatisch die min. Startzeit als nächsten Wakeup-Punkt. Im anderen Fall, wenn der aktuelle Wert kleiner als der max. Wert ist, wird eine neue Suchfunktion aufgerufen, die die nächstgrößere Startzeit sucht und sie als Wakeup-Punkt setzt. Siehe Abbildung 6.12.

Kapitel 7

Led Demo

Die Aufgabenstellung besteht darin, die Kommunikation zwischen zwei Transceivern auf einer FPGA-Hardware zu zeigen. Das eine Testboard dient als Sender, das andere als Empfänger. Für die Demonstration sind drei Kommandos vorgesehen.

- LED_DEMO_CMD_TOGGLE 0x01h soll die LED ein oder ausschalten
- LED_DEMO_CMD_START_DIM 0x02h soll die Lichtstärke der LED erhöhen (hochdimmen) oder verringern (runterdimmen)
- LED_DEMO_CMD_STOP_DIM 0x03h soll die Veränderung der Lichtstärke stoppen.

Diese Kommandos werden durch das Drücken der Taste am Sender Board generiert. Welche CMD (Kommando) generiert wird, hängt davon ab, wie lange die Taste gedrückt wird. Das Empfängerboard soll auf diese drei Kommandos je nach Zustand unterschiedlich reagieren und die Kommunikation zwischen den Boards zeigen.

7.1 Versuchsaufbau

- (1) CHOSeN TX Sender Board (Abb. 7.1)
- (2) CHOSEN RX Receiver Board (Abb. 7.1)
- (3) USB Connector (Abb. 7.1)
- (4) PC (Abb. 7.1)
- Oszilloskop zur Betrachtung des PWM Signals (Abb. 7.2)

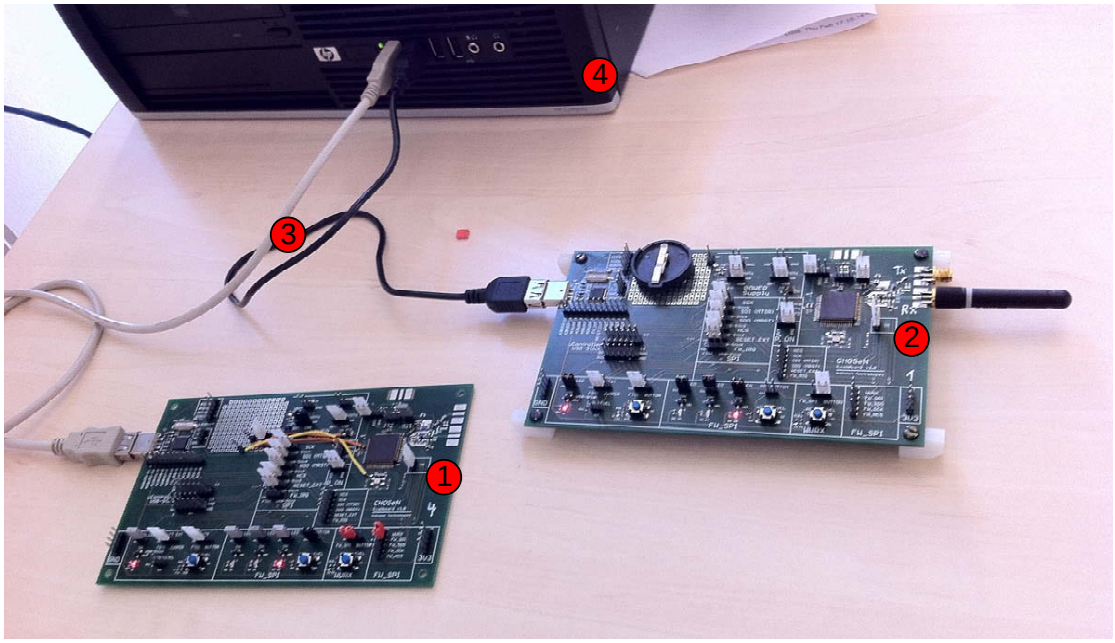

Abbildung 7.1: Versuchsaufbau Bild 1

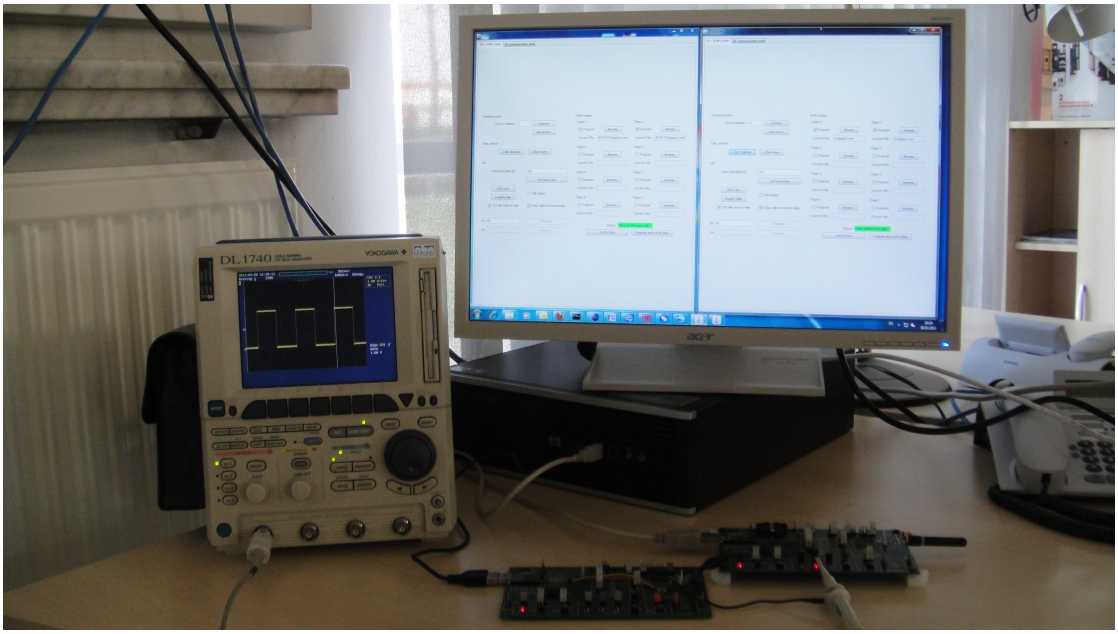

Abbildung 7.2: Versuchsaufbau Bild 2

Der PC dient hier als Spannungsquelle und mit seinem Java ROM Loader Programm zum Speichern der externen ROM Pages auf den μController als Programmierschnittstelle. Der Java-ROM-Loader kann auch zum Resetten des μControllers genutzt werden. Die ROM-Loader-Benutzeroberfläche ist in folgender Abbildung dargestellt, wobei das linke Fenster für das TX (Senderboard) und das rechte Fenster für das RX (Empfängerboard) steht.

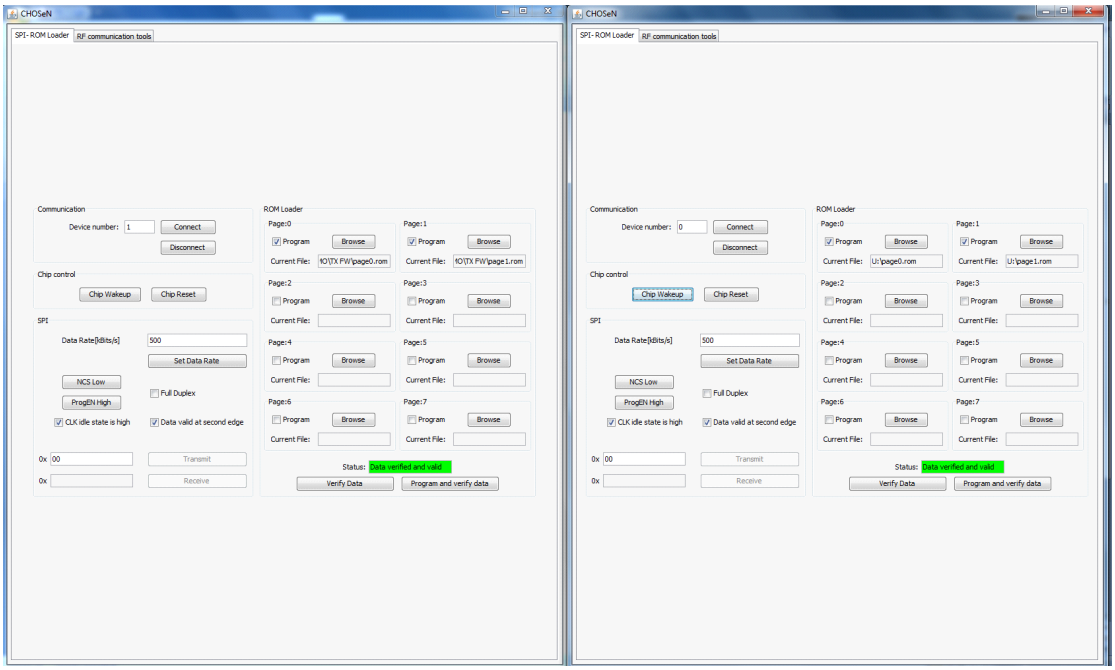

Abbildung 7.3: ROM Loader Software

7.2 Definition der Kommandos

LED_DEMO_CMD_TOGGLE
Das TOGGLE-Kommando besitzt den Hexwert 0x01h. Es wird automatisch vom Sender Board (TX) generiert, wenn die Taste eine bestimmte Zeit (in unserem Fall weniger als eine Sekunde) lang gedrückt wird (siehe Abbildung 7.4).

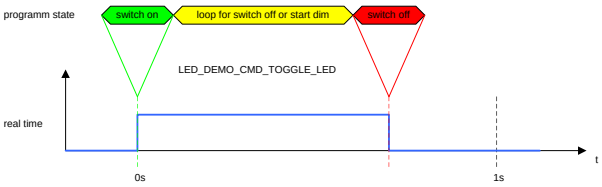

Abbildung 7.4: cmd toggle

LED_DEMO_CMD_START_DIM
Das START-Kommando besitzt den Hexwert 0x03h. Das START_DIM Kommando wird automatisch vom Sender Board (TX) generiert, sobald die Taste eine Sekunde lang gedrückt wurde (siehe Abbildung 7.5).

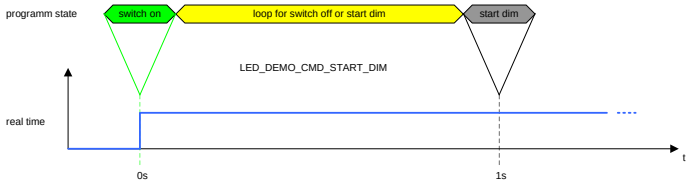

Abbildung 7.5: cmd start dim

LED_DEMO_CMD_STOP_DIM
Das STOP-Kommando besitzt den Hexwert 0x02h. Das START_DIM Kommando wird automatisch vom Sender Board(TX) generiert, wenn die Taste länger als eine Sekunde gedrückt gehalten wurde (siehe Abbildung 7.6).

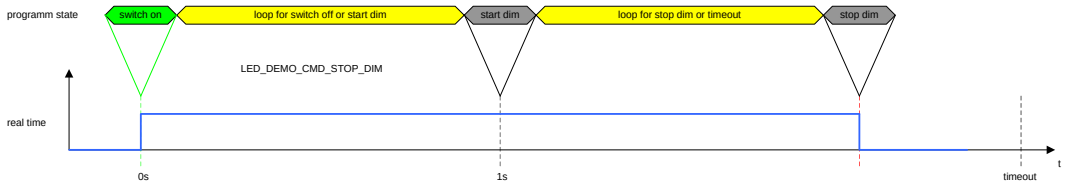

Abbildung 7.6: cmd stop dim

7.3 Programmstruktur des Senders

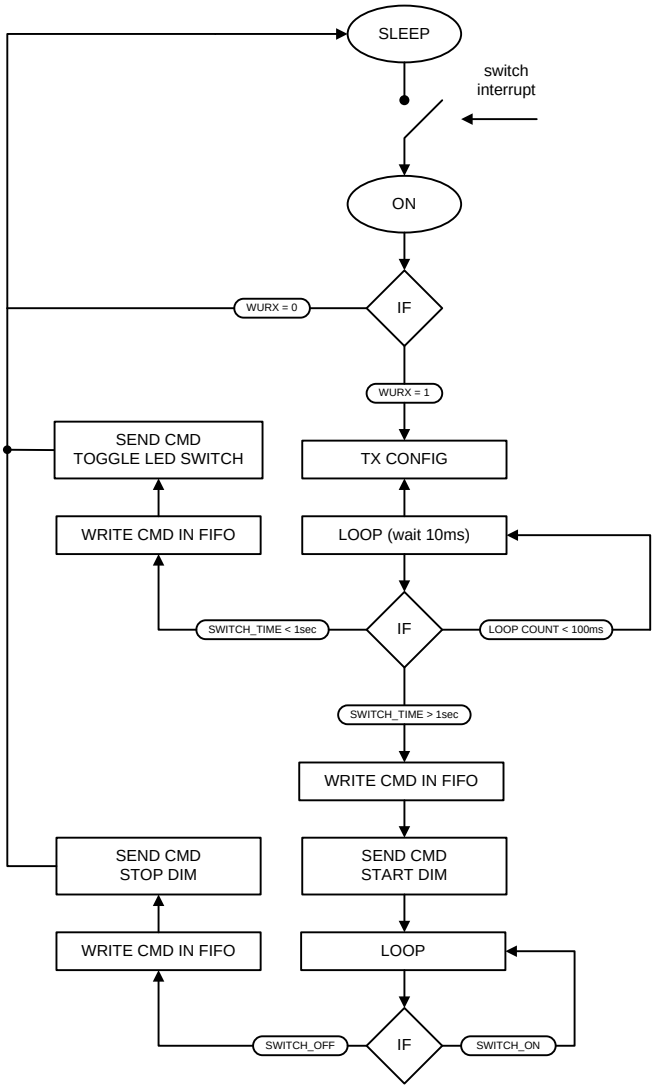

Abbildung 7.7: TX Programmstruktur

Das Sender-Board befindet sich die ganze Zeit in einem Sleep Modus, wie am Anfang der Abbildung 7.7 gezeigt wird. Während dieser Zeit ist außer der iSM jedes Modul am „Schlafen“. Die iSM wartet auf einen bestimmten Interrupt, in unserem Fall auf die Bestätigung der Taste. Ist der IRQ erfolgt, wird die Einsprungadresse der Wakeup-Funktion, die in der DIVT gespeichert ist, ausgeführt. Während die Taste gedrückt

ist, wird in dem SFR_SPARE3 Register eine Flag auf 1 gesetzt. Die Wakeup-Funktion startet eine Schleife, die unter 1 sec dauert. Sollte während dieser Zeit das Flag im SFR_SPARE3 Register den Wert Null (Taste wird losgelassen) annehmen, wird der Hexwert des LED_DEMO_CMD_TOGGLE Kommandos temporär gespeichert. Die Schleife wird bewusst zum Auslaufen gebracht und der Sendezyklus wird eingeleitet. Siehe rotes Feld in Abbildung 7.8.

```
//1. ther are two switch types   switch_time_valeu = 0 (if  < 1sec), switch_time_value = 1 (if > 1sec)

void led_demo_switch_time_loop()
{
          for (i=0; i<100; i++)
          {
            WAIT_US_ACTUALPAGE(40) ;

            if (ISNOTBITSET(SFR_SPARE3,FW_SDI))            //  SWITCH_TIME is < 1 sec then set
                                                               SWITCH_TIME_VALUE = 0, exit  loop
                    {
                     switch_time_flag = 0;
                     i = 100;
                    }
            else if (ISBITSET(SFR_SPARE3,FW_SDI)) // SWITCH_TIME is > 1 sec then set
                                                  SWITCH_TIME_VALUE = 1, continue loop
                    {
                     switch_time_flag = 1;
                    }

          }

          if (switch_time_flag == 0)
                    {
                      //jmp to send CMD_TOGGLE_LED_SWITCH
                      param2 = LED_DEMO_CMD_TOGGLE;
                      //switch the Led 2 on Transiever Side
                      LED_DEBUG(2);
                      // waits a only to hold the Led Light on Transiever Side for demonstration
                      WAIT_US_ACTUALPAGE(10000);
                      WAIT_US_ACTUALPAGE(10000);
                      WAIT_US_ACTUALPAGE(10000);

          }

          else if (switch_time_flag == 1)
                    {
                      //jmp to send CMD_START_DIM
                      param2 = LED_DEMO_CMD_START_DIM;
                      LED_DEBUG(3);
                      WAIT_US_ACTUALPAGE(10000);

                    }

          led_demo_write_to_fifo();

}
```

Abbildung 7.8: CMD TOGGLE

Im anderen Fall, wenn die Schleife von allein ausläuft, wird automatisch der Hexwert

ders LED_DEMO_CMD_START_DIM Kommandos temporär gespeichert. Siehe gelbes Feld in Abbildung 7.8. Vor dem Senden werden die temporär gespeicherten Werte in das FIFO weiter geleitet, der Wakeup-Punkt gesetzt, und erst dann der Sendevorgang durchgeführt.

Die Wakeup-Punkte von TOGGLE und STOP_DIM haben die gleiche Adresse, im Gegensatz zur START_DIM, wo nochmals eine Schleife mit einem Timeout gestartet wird. In der Schleife wird weiter das Flag des SFR_SPARE3 Registers betrachtet. Wenn das Flag während des Schleifenvorganges geleert wird, speichert das System den Hexwert des LED_DEMO_CMD_STOP_DIM Kommandos. Siehe rotes Feld in Abbildung 7.9. Das FIFO wird mit dem Hexwert gefüllt und das STOP_DIM Kommando wird gesendet. Im anderen Fall, wenn die Schleife das Timeout erreicht, geht das System automatisch in den Sleep Modus und wartet auf einen neuen Tastendruck. Siehe gelbes Feld in Abbildung 7.9.

```
void led_demo_wait_for_switch_off()
{
        for (i=0; i<1000; i++)
        {
                //TODO: config prescaler on 10ms
                WAIT_US_ACTUALPAGE(100);

                if (ISNOTBITSET(SFR_SPARE3,FW_SDI))     //  SWITCH_TIME is < 1 sec then set
                                                        SWITCH_TIME_VALUE = 0, exit  loop
                {
                        switch_time_flag = 0;
                        i = 1000;
                }
                else if (ISBITSET(SFR_SPARE3,FW_SDI))   // SWITCH_TIME is > 1 sec then set
                                                        SWITCH_TIME_VALUE = 1, continue loop
                {
                        switch_time_flag = 1;
                }
        }
        if (switch_time_flag == 0)
        {
                LED_DEBUG(4);
                //write data in fifo
                param2 = LED_DEMO_CMD_STOP_DIM;
                //jmp to send CMD_STOP_DIM
                led_demo_write_to_fifo();
        }
        else if (switch_time_flag == 1)
        {
                LED_DEBUG(5);

                //time out
                //go off
                led_demo_go_offline();

        }

}
```

Abbildung 7.9: CMD TOGGLE

7.4 Programmstruktur des Empfängers

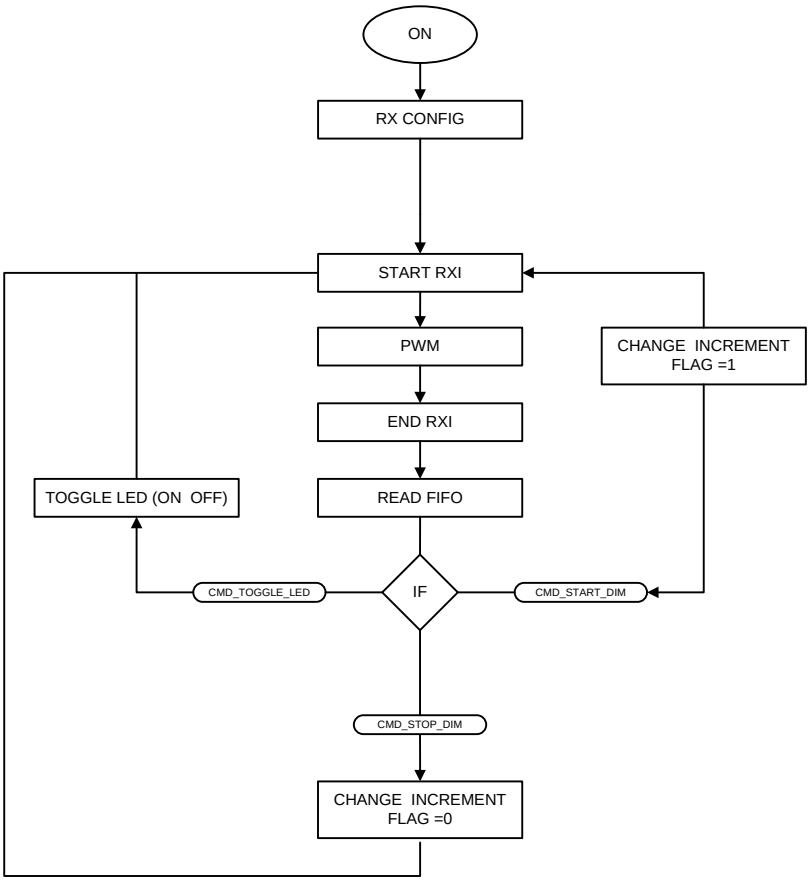

Abbildung 7.10: LED Demo RX Programstruktur

Das Empfängerboard wird kurz nach dem Einschalten konfiguriert (siehe Abbildung 7.10) um es auf dem Empfangs-Modus vorzubereiten. Das heißt Kanalfrequenzen und Dataenrate werden auf Standardwerte gesetzt und der Empfänger wird auf Dauerempfangen gestellt. Noch vor dem Empfangen von Daten wird die PWM Funktion (Pulsweitenmodulation) gestartet (näheres im Abschnitt 7.5). Sobald sich das Gerät mit dem gesendeten Paket synchronisiert und es richtig empfangen hat, wird der Wert aus dem FIFO entnommen und ausgewertet. Bei TOGGLE wird die LED ausgeschaltet oder auf die aktuelle eingestellte Lichtstärke gedimmt. Im anderen Fall, bei START_DIM durchläuft das Kommando mehrmals die PWM in einer Schleife und inkrementiert jedes mal die Lichtstärke. Das wird solange wiederholt, bis entweder das STOP Kommando kommt, oder das Ganze in einem Timeout endet. Beim Timeout wird die max. Lichtstärke erreicht und es wird

danach nicht mehr gedimmt. Sollte ein STOP Kommando empfangen werden, wird die PWM nicht mehr erhöht und die LED leuchtet in der aktuellen Lichtstärke.

7.5 Programmstruktur der PWM

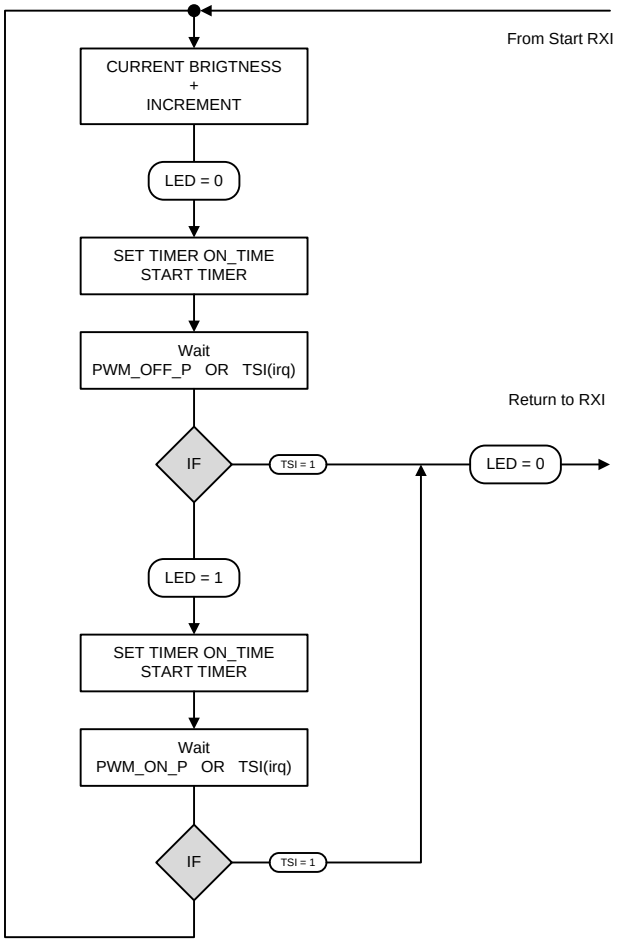

Abbildung 7.11: PWM Programmstruktur

Die Abbildung 7.11 repräsentiert die PWM Funktion aus der Abbildung 7.10. Die PWM Funktion besteht aus einer Endlosschleife, in der an zwei Stellen kurz auf ein neues Kommando gewartet wird. Sollte an diesen zwei Toren kein Interrupt statt finden, ändert der Timer, der die LED für eine kurze Zeit ein- oder ausgeschaltet hält, die Länge seiner

on/off Zeit. Die Länge des Timers wird während des ON Zustandes (Zeit während die LED eingeschaltet ist) inkrementiert. Es strahlt die LED immer heller. Diese wird im Code Beispiel in der Abbildung 7.12 realisiert. Im anderen Fall, wenn die Länge des Timers während des OFF Zustandes (Zeit während die LED ausgeschaltet ist) inkrementiert wird, verringert sich die Lichtstärke der Led.

Der Parameter param1 beinhaltet den Helligkeitswert der LED. Dieser Wert wird bei jedem Schleifendurchgang inkrementiert/dekrementiert (abhängig immer vom vorherigen Zustand der LED). Wenn die LED zu einem früherem Zeitpunkt hochgedimmt (inkrementiert) wurde, wird sie zum aktuellen Zeitpunkt runtergedimmt (dekrementiert).

```
REGSET(SFR_T0PRE, SYSTEM_CLOCK_FREQUENCY_MHZ * param1 );
REGSET(SFR_T0C, PRESCAL0EN_1 | PRESCALER_2POW_2);

BADWAIT01(T0ISRC, SFR_IS0, FSYNCISRC, SFR_IS2);
REGSET(SFR_T0C, PRESCAL1EN_0 | 0x00);
```

Abbildung 7.12: Code Beispiel On Zustand des Timers

7.6 PWM Simulation

Die Modulation ist in 10 Helligkeitswerte unterteilt und erhöht sich immer wieder um 10%. Die Abbildung 7.13 zeigt die PWM in einem Anfangsstadium, während die ON Zeit sehr kurz und die OFF Zeit sehr lange ist. Abbildungen 7.14 und 7.15 zeigen das Hochdimmen der PWM.

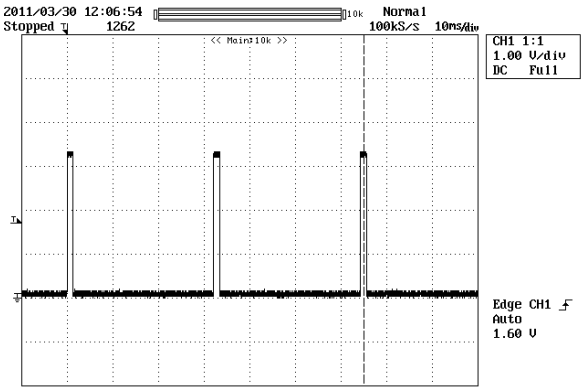

Abbildung 7.13: 10% Helligkeit der LED

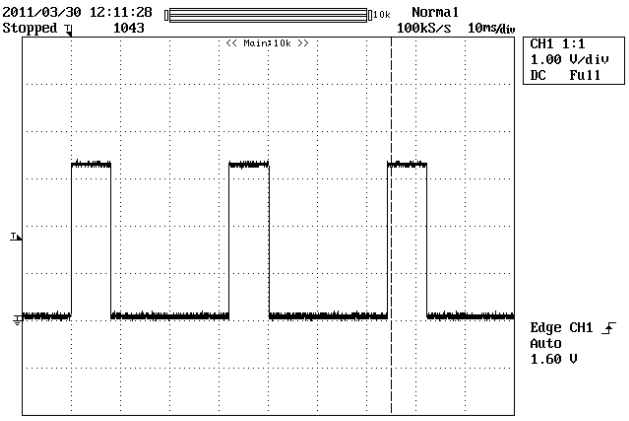

Abbildung 7.14: 30% Helligkeit der LED

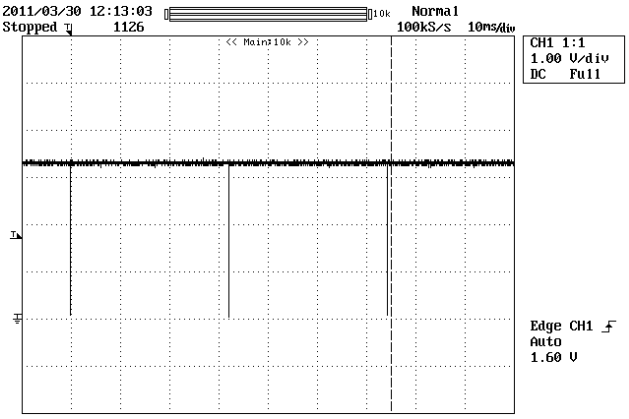

Abbildung 7.15: 100% Helligkeit der LED

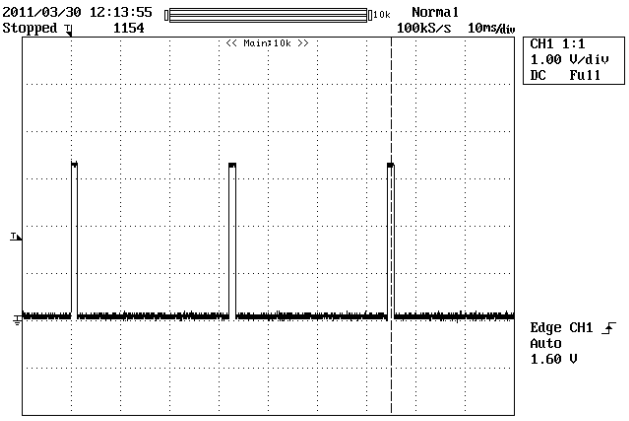

Abbildung 7.16: 10% Helligkeit der LED

In der Abbildung 7.16 wird die Lichtstärke von 100% auf 10% reduziert, Da ein erneutes LED_DEMO_CMD_START_DIM Kommando gesendet wurde, das in diesem Fall die Lichtstärke reduziert. Auf dem Empfängerboard wird jedesmal ein Status-Flag gesetzt oder gelöscht, das bestimmt, ob die PWM inkrementiert oder dekrementiert wird.

Kapitel 8

Zusammenfassung

Ziel dieser Arbeit war die Entwicklung einer Firmware, die für einen Smart Radio Transceiver eingesetzt wird. Es wurde dabei die bereits existierende Firmware erweitert, ein Self-Poling-System entwickelt (Scheduler Mechanismus), sowie die Kommunikation zwischen mehreren Transceivern getestet. Bei der Entwicklung wurde ein von Infineon entwickelter Transceiver Test-Chip verwendet. Durch Verwendung von mehreren Simulationstools, sowie Erstellung von mehreren Test Firmware Versionen wurden Ergebnisse analysiert und entsprechend dargestellt. Nach dem Analysieren wurden einige Bugs an der Hardware festgestellt und korrigiert, die Firmware für den Transceiver Chip wurde optimiert und eine einwandfreie Kommunikation ermöglicht. Ein wichtiger Teil der Arbeit war die Entwicklung des Scheduler Mechanismus, der den Transceiver im Low-Power-Betrieb unterstützen soll, sowie das Abspeichern verschiedener Events mit unterschiedlichen Funktionen. Diese Events sollen für späteren Gebrauch des Transceiver-Chips in einem drahtlosen Sensornetzwerk den Austausch von Daten optimierten.

Als Abschluss wurde ein Demo Beispiel erstellt in dem zwei Transceiver benutzt werden, um die Lichtintensität einer Lichtquelle über ein PWM Signal zu verändern. Dieses Konzept wurde später in der drahtlosen Lichtsteuerung eingesetzt und weiter entwickelt.

Literaturverzeichnis

[1] Instruments, N.: *Was ist ein Wireless-Sensornetzwerk*. online, März 2011. http://www.ni.com/white-paper/7142/de.

[2] ITWissen: *Drahtloses Sensornetzwerk*. online, 2012. http://www.itwissen.info/definition/lexikon/WSN-wireless-sensor-network-Drahtloses-Sensornetzwerk.html.

[3] ITWissen: *Firmware*. online, 2012. http://www.itwissen.info/definition/lexikon/Firmware-firmware.html.

[4] ITWissen: *Paging*. online, 2012. http://www.itwissen.info/definition/lexikon/Paging-paging.html.

[5] ITWissen: *RSSI*. online, 2012. http://www.itwissen.info/definition/lexikon/received-signal-strength-indicator-RSSI.html.

[6] ITWissen: *Superframes*. online, 2012. http://www.itwissen.info/definition/lexikon/Superframe-SF-superframe.html.

[7] JUN ZHENG, A. J.: *Wireless Sensor Networks: A Networking Perspective*. WILEY, 2009.

[8] Kim, Hojjat Adeli, R. J. R. Tai-hoon: *Ubiquitous Computing and Multimedia Applications*. Springer, 2011.

[9] Loew, M. und J. Heyszl: *Development Specification TRX BASE m5350*. Techn. Ber., Infineon Technologies Austria AG. Confidential.

[10] Wikipedia: *CCA*. online. http://de.wikipedia.org/wiki/BerkeleyMediaAccessControl.

[11] Wikipedia: *Firmware*. online. http://de.wikipedia.org/wiki/Firmware.

[12] Wikipedia: *Paging*. online. http://de.wikipedia.org/wiki/Paging.

[13] Wikipedia: *Scheduler*. online. http://de.wikipedia.org/wiki/Scheduling.