Masterarbeit

Optimierung und Evaluation Black Burst-basierter Protokolle unter Verwendung der Imote 2-Plattform

von

Markus Engel
Matrikel-Nr. 357573

15. März 2013

Technische Universität Kaiserslautern
Fachbereich Informatik
AG Vernetzte Systeme

Prüfer: Prof. Dr. Reinhard Gotzhein
Zweitgutachter: M.Sc. Dennis Christmann

Zusammenfassung – Abstract

Diese Arbeit behandelt die Implementierung und Evaluierung Black Burst-basierter Protokolle, die aufgrund der Kollisionresistenz durch die Kodierung mit Black Bursts eine attraktive Lösung für Szenarien mit hohen Dienstgüteanforderungen darstellen. Ein besonderer Schwerpunkt liegt hierbei auf Verzögerungen, die durch Hardwarelaufzeiten entstehen. Im Fokus stehen Optimierungen, die im Hinblick auf die verwendete Sensorplattform Imote 2 ermöglicht werden. Ein Ziel der Arbeit ist die Vermittlung eines umfassenden Einblicks in Black Burst-basierte Kommunikation sowie in die verwendete Hardware, deren Einschränkungen und die daraus resultierenden Optimierungsansätze. Darauf aufbauend werden in zahlreichen Experimenten Hardwareverzögerungen analysiert und Empfehlungen für eine Implementierung der Konzepte angegeben. Außerdem wird die Gültigkeit von Verzögerungen, die in den Datenblättern der Hardware angegeben sind, überprüft und teilweise widerlegt. Ein weiteres Ziel der Arbeit ist der Entwurf eines Kommunikationsframeworks, in dem die Erkenntnisse aus den Experimenten berücksichtigt sind, und dessen Implementierung, die robust gegenüber Hardwareverzögerungen ist.

This thesis evaluates Black Burst-based communication protocols. Due to the collision resistance of black bursts, such protocols state an attractive solution for scenarios with high requirements on quality of service. A focus of this thesis is on latencies, which are induced by the limitations of hardware. By deciding on a fixed hardware, the Imote 2 sensor platform, optimizations, which reduce those delays, will be given. This thesis provides a deep insight into Black Burst-based communication, into the utilized hardware and its limitations. Based on that, latencies and optimizations will be evaluated in several experiments, and recommendations for an implementation will be derived. Additionally, the validation of delays given by the hardware datasheets will be checked and partially falsified. Furthermore, based on the results of the experiments, a communication framework, which is robust against hardware delays, will be designed and implemented.

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Masterarbeit mit dem Thema "Optimierung und Evaluation Black Burst-basierter Protokolle unter Verwendung der Imote 2-Plattform" selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, habe ich durch die Angabe der Quelle, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht.

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

1. Kapitel

Einleitung

Die steigenden Anforderungen an Kommunikationsprotokolle verlangen eine immer genauere Beachtung von Hardwareeinschränkungen beim Protokolldesign und der Implementierung. Dies gilt vor allem bei drahtlosen Sensornetzwerken, die neuerdings auch vermehrt im industriellen Umfeld eingesetzt werden. Die Szenarien stellen nicht selten auch Anforderungen bezüglich einer hohen oder gar deterministischen Genauigkeit, die ein Kommunikationsprotokoll nur liefern kann, wenn auch die unterliegende Hardware ein deterministisches Verhalten aufweist. Dieses Ziel kann nur erreicht werden, wenn die Einschränkungen und Laufzeiten der Hardware bei der Protokollentwicklung Beachtung finden und die Implementierung robust gegenüber Verzögerungen ist.

Zur Entwicklung deterministischer Protokolle ist es allerdings nicht ausreichend, dass sich die Hardware deterministisch verhält. Zusätzlich muss auch das Kommunikationsprotokoll einen deterministischen Zugriff auf das Medium ermöglichen. Diesbezüglich stellen insbesondere destruktive Kollisionen ein Risiko dar. Zu den Kommunikationsprotokollen, die destruktive Kollisionen vermeiden, gehören u. A. Protokolle, die auf *Black Bursts*, einer speziellen Kommunikationsprimitive, basieren. Der Fokus dieser Arbeit liegt in der Realisierung von Kommunikationsprotokollen, die sich die positive Eigenschaft dieser Kommunikationsprimitive, die Kollisionsresistenz, zu Nutze machen.

Ein besonderes Ziel der Arbeit liegt darin, einen umfassenden Einblick in die verwendete Imote 2-Plattform und deren Komponenten zu geben und die Eigenschaften der Plattform bei der Entwicklung eines Kommunikationsframeworks zu beachten. Durch diesen Einblick ist es möglich, die Verzögerungsquellen, die beim Erfassen von Zeitpunkten Ungenauigkeiten hervorrufen, zu identifizieren. Ausgehend von diesen werden in der Arbeit Experimente durchgeführt, die zur Charakterisierung und Quantifizierung der Verzögerungen beitragen. Ferner werden auch Optimierungen entworfen und evaluiert, durch die Gesamtverzögerungen und insbesondere auch variable Anteile verringert werden können.

Eine wichtige Komponente des Imote 2 ist der CC 2420-Transceiver. Bei dieser Komponente handelt es sich um einen Funkchip, der insbesondere in Sensornetzen eine weite Verbreitung findet. Durch den Einsatz dieses Funkchips können Verzögerungsquellen identifiziert werden, die erhebliche Auswirkungen auf die Genauigkeit und Performanz Black Burst-basierter Protokolle besitzen. Einige dieser Verzögerungen sind durch das Datenblatt der Hardware angegeben. Darüber hinaus ist es wünschenswert, weitere, nicht angegebene Verzögerungen zu bestimmen und die im Datenblatt angegebenen Verzögerungen zu verifizie-

ren. In dieser Arbeit werden Experimente durchgeführt, die eine genauere Charakterisierung der Verzögerungen erlauben und Abweichungen vom Datenblatt angegeben.

Eine zweite wichtige Komponente des Imote 2 ist der PXA 271-Prozessor. Viele Hardware-Optimierungen, die in modernen Prozessoren Verwendung finden, sind zwar für eine normale Programmausführung wünschenswert, da sie die durchschnittliche Ausführungszeit erheblich verringern, erhöhen jedoch die variablen Anteile der Verzögerungen stark. Zum Beispiel wird durch den Einsatz von Caches der Zugriff auf ein vom Cache vorgehaltenes Datum stark beschleunigt, jedoch ist dadurch die Zugriffszeit auf ein beliebiges Datum in hohem Maße davon abhängig, ob das Datum im Cache vorgehalten wird oder erst vom Hintergrundspeicher geladen werden muss. Durch geeignete Maßnahmen, die im Hinblick auf den PXA 271-Prozessor in dieser Arbeit vorgestellt werden, wurden diese Schwankungen reduziert.

Die Identifikation und genauere Charakterisierung aller Verzögerungsquellen wird in der Arbeit verwendet, um Black Burst-basierte Protokolle unter den Gesichtspunkten von Hardwarelaufzeiten und -verzögerungen zu entwickeln und zu evaluieren. Dadurch konnte die Robustheit der Verfahren gegenüber Schwankungen der Hardwarelaufzeiten gesteigert werden.

Eine Implementierung Black Burst-basierter Protokolle muss, um alle Optimierungen berücksichtigen zu können, die vollständige Kontrolle über die Hardware zulassen. Das Protokolldesign muss daher bereits in der Systemsoftware verankert werden und es wird eine hohe Kenntnis der eingesetzten Hardware vorausgesetzt. In dieser Arbeit wird ein Kommunikationsframework entworfen und implementiert, in dem die Erkenntnisse aus den Experimente eingearbeitet sind. Dadurch wurde die Grundlage geschaffen, um zeitkritische Kommunikationsprotokolle im Kontext des Frameworks zu implementieren und zu evaluieren. Mit *Black Burst Synchronization* wurde ein Black Burst-basiertes Synchronisationsprotokoll in dieses Framework integriert.

Die Arbeit ist wie folgt aufgebaut: Zunächst werden in Kap. 2 die Grundlagen Black Burst-basierter Kommunikation vermittelt. Darauf aufbauend wird die Genauigkeit und Konvergenzzeit von BBS im Kontext von Hardwareverzögerungen neu analysiert. Kapitel 3 gibt einen umfassenden Einblick in die verwendete Imote 2-Plattform. Dabei wird insbesondere auf Verzögerungen hingewiesen und auf Optimierungen eingegangen. In Kapitel 4 werden Experimente zum CC 2420-Transceiver und PXA 271-Prozessor definiert, durchgeführt und ausgewertet. Insbesondere werden verschiedene Optimierungsalternativen evaluiert. In Kapitel 5 wird der Entwurf eines Kommunikationsframeworks mit Unterstützung von BBS und dessen Implementierung vorgestellt. Dabei fließen die Ergebnisse und Erkenntnisse der Experimente mit ein. Die Implementierung wird in Kap. 6 evaluiert. Abschließend gibt Kap. 7 eine Zusammenfassung der Arbeit sowie einen Ausblick auf zukünftige Arbeiten an.

2. Kapitel

Konzepte

In diesem Kapitel werden die Grundlagen und Konzepte der in dieser Arbeit relevanten Verfahren erklärt. Ferner werden weitere Konzepte vorgestellt, die zwar weniger im Fokus der Arbeit liegen, jedoch einen Überblick über das Themengebiet geben sollen.

Zunächst werden Unzulänglichkeiten bestehender Mehrfachzugriffsverfahren in Hinsicht auf deterministische Garantien vorgestellt. Anschließend werden die Grundlagen, Eigenschaften und Anwendungen von *Black Bursts*, einer speziellen Kommunikationsprimitive, gegeben. Darauf aufbauend wird auf Funktion und Eigenschaften des Synchronisationsprotokolls *BBS* (*Black Burst Synchronization*) eingegangen. Abschließend wird mit *MacZ* ein Mehrfachzugriffsverfahren vorgestellt, welches mit Hilfe von BBS und weiterer Black Burst-basierter Protokolle deterministische Garantien in drahtlosen Sensornetzen geben kann. Am Ende des Kapitels werden verwandte Arbeiten der einzelnen Grundlagen vorgestellt, und aufgezeigt, inwiefern die Ergebnisse dieser Arbeit auch andere Protokolle beeinflussen.

2.1 Unzulänglichkeiten bestehender Mehrfachzugriffsverfahren

Zum typischen Einsatzgebiet von Sensorknoten gehört das periodische Erfassen von Werten. Dabei kann es sich um ein einfaches Überwachen von Werten der Umgebung wie z. B. der Temperatur handeln. Denkbar ist aber auch, dass Sensorknoten als Teil eines Regelkreislaufs eingesetzt werden, in denen der erfasste Wert innerhalb einer bestimmten Zeitspanne zu einem anderen Knoten oder einer Datensenke weitergeleitet werden muss. Wenn nun obere Schranken eingehalten werden müssen, müssen diese vom gesamten System und insbesondere von den unterliegenden Kommunikationsprotokollen eingehalten werden.

Ein Problem, welches allgemein bei wahlfreien Medienzugriffsverfahren auftritt, sind Kollisionen. Diese treten auf, wenn mehrere Stationen zeitgleich auf das gemeinsame Medium zugreifen. In der Domäne der drahtlosen Kommunikation bedeutet dies, dass eine destruktive Kollision entstehen kann, wenn mehrere Stationen zur gleichen Zeit im gleichen Frequenzbereich eine Übertragung beginnen, die mit der jeweils anderen interferiert. Das Auftreten einer Interferenz hängt dabei von vielen Parametern ab, die durch die Umgebung gegeben sind, vor allem aber von räumlicher Distanz und der verwendeten Sendeleistung.

Da es den Stationen möglich sein soll zu kommunizieren, müssen sie in Kommunikationsreichweite sein, insbesondere also auch in Interferenzreichweite. Ferner müssen sie daher auch den selben Frequenzbereich verwenden. Um Interferenzen zu vermeiden, muss folglich der zeitliche Zugriff auf das Medium reguliert werden.

Bei einfachen *Carrier Sensing*-Verfahren wird bei einem Sendewunsch zunächst geprüft, ob zum aktuellen Zeitpunkt bereits eine Übertragung vorliegt. CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*), wie es in Ethernet [26] verwendet wird, ist nicht auf drahtlose Kommunikation anwendbar, da bei üblicher Funkhardware die Möglichkeit fehlt, eine Kollision während der eigenen Übertragung zu erkennen.

Auch Verfahren mit zufälligen Wartezeiten, wie etwa CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*), können nur Teile der Problematik lösen. Einerseits bietet das Carrier-Sensing nur die Information, dass das Medium zu dem Zeitpunkt der Prüfung unbelegt war. Reale Funkhardware benötigt vor dem eigentlichen Sendevorgang jedoch noch eine Umschaltdauer, in der u. A. der Frequenzgenerator neu kalibriert wird. Während dieser Zeit liegt eine Blindphase vor, in der kein Empfang möglich ist. Somit ist nicht garantiert, dass auch zum eigentlichen Sendebeginn das Medium noch unbelegt ist. Andererseits können auch bei wettbewerbsbasierten Verfahren, die einen zufälligen Backoff aus einem Contention Window wählen, weiterhin Kollisionen auftreten, z. B. wenn mehrere Stationen den gleichen Zufallswert erhalten.

Bei Multihop-Netzwerken kommt noch das *Hidden Station*-Problem [31] hinzu. Ein Endgerät, welches außerhalb der Sensing-Reichweite des Senders liegt, würde eine Übertragung nicht erkennen und könnte daher eine eigene Übertragung beginnen. Bei einem Knoten, der in Kommumikationsreichweite beider Stationen liegt, würde daher eine Kollision auftreten.

Unter dem Namen MACA [29] (*Multiple Access with Collision Avoidance*) wird das CSMA Verfahren durch einen zusätzlichen RTS/CTS Handshake erweitert, bei dem vor der eigentlichen Übertragung eine Nachricht zur Allokation des Mediums (RTS – *Request To Send*) versandt wird, welche vom jeweiligen Empfänger durch eine entsprechende Antwort (CTS – *Clear To Send*) quittiert wird. So sind alle Stationen in Kommunikationsreichweite von Sender und Empfänger über die anstehende Übertragung informiert. Um Rahmenverluste bereits auf MAC-Ebene zu erkennen, wurde das Zugriffsverfahren in MACAW [8] (*MACA for Wireless*) durch einen ACK-Mechanismus erweitert, der einen korrekten Rahmenempfang bereits auf MAC-Ebene quittiert. Dieses Zugriffsverfahren wird bei IEEE 802.11 [24] eingesetzt. Kollisionen können durch diese Maßnahmen zwar erkannt, aber nicht ausgeschlossen werden.

Diese Verfahren können also keine Garantien zur Übertragungsverzögerung liefern. Die Lösung besteht darin, den zeitlichen Zugriff auf das Medium zu regulieren, so dass keine Kollisionen stattfinden können. Hierzu kann eine Master-Station die anderen Stationen abfragen (*Polling*), was jedoch gerade in Multi-Hop Netzwerken weitere Herausforderungen stellt. Alternativ kann ein TDMA-Schema (*Time Division Multiple Access*) verwendet werden, bei dem die Nutzung des Mediums zeitlich eingeteilt wird und über Reservierungen die exklusive Nutzung des Mediums zugesichert werden kann. Die Nutzung von TDMA fordert jedoch eine Synchronisation der Knoten, so dass die Zuteilungen von jedem Knoten eingehalten werden können.

2.2 Black Bursts

In diesem Kapitel werden *Black Bursts*, eine Kommunikationsprimitive, bei der Übertragungen kollisionsresistent sind, vorgestellt. Auf dieser Basis können Kommunikations- und Mehrfachzugriffsprotokolle entwickelt werden, die diese Eigenschaft ausnutzen und somit

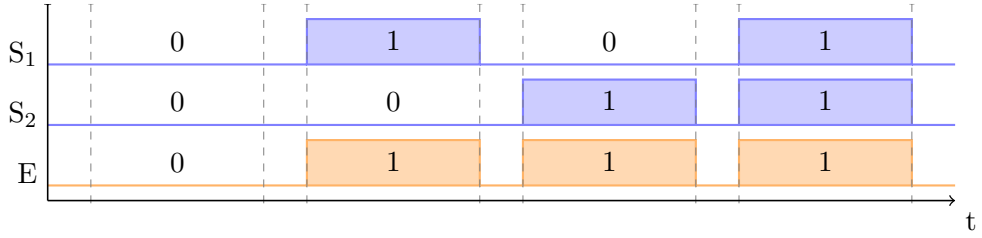


Abbildung 2.1.: Überlagerung von Black Bursts zweier Sender S_1 und S_2. Der Empfänger E erkennt das bitweise ODER beider Bitsequenzen.

maximale Verzögerungen bei der Übertragung garantieren können. Ferner wird dies auch von einem Synchronisationsprotokoll genutzt, welches in Kap. 2.3 detailliert erklärt wird.

2.2.1 Definition von Black Bursts

Black Bursts sind konzeptionell Energieimpulse auf dem Medium mit einem definierten Start- und Endzeitpunkt. Überlagert sich die Übertragung zweier Black Bursts, ist für einen Empfänger weiterhin Energie auf dem Medium detektierbar. Werden Black Bursts gleicher Länge (nahezu) zeitgleich versandt, ist die Information, die in ihnen enthalten ist, also Start- und Endzeitpunkt, immer noch korrekt erkennbar. Black Bursts sind somit kollisionsresistent.

Im Folgenden beschränken wir uns auf Übertragungsverfahren, bei denen Black Bursts einer festen Länge größer Null einer logischen 1 und Black Bursts der Länge Null einer logischen 0 zugeordnet werden. Das „Senden“ einer logischen 0 bedeutet also, dass tatsächlich keine Übertragung stattfindet. Insbesondere wird während dieser Zeit die Funkhardware auch nicht in den Sendemodus überführt und kann daher währenddessen den Zustand des Mediums überwachen und logische Einsen anderer Stationen erkennen.

Die Kodierung und Übertragung von Daten als Black Bursts hat die Bildung eines bitweisen ODER auf dem Medium zur Folge: Wird von mindestens einer Station in Sensingreichweite eine logische 1 übertragen, ist für eine empfangende Station eine logische 1 sichtbar. Dieser Zusammenhang ist in Abb. 2.1 dargestellt. Da die gleichzeitige Übertragung einer logischen 0 und einer logischen 1 das Erkennen der logischen 1 zur Folge hat, wird im Folgenden der Black Burst, der eine logische 1 repräsentiert, als *dominant* und der Black Burst, der eine logische 0 repräsentiert, als *rezessiv* bezeichnet.

2.2.2 Implementierung des Black Burst-Konzepts

Zum Versenden eines Black Bursts würde es genügen, eine unmodulierte Trägerwelle auf das drahtlose Medium aufzuschalten. In der Praxis möchten wir jedoch auf bestehende Hardware zurückgreifen. Diese ist aber für die Übertragung von Rahmen bestimmter Protokolle vorgesehen und kann daher in der Regel keine un- oder zufällig modulierten Trägerwellen abstrahlen. In Transceivern, in denen eine solche Funktion integriert ist, ist dies oft mehr als Testfunktion und nicht für produktiven Einsatz gedacht. In diesem Fall ist dann möglicher Weise keine genaue Kontrolle über die Dauer der Medienbelegung möglich. Der in dieser Arbeit betrachtete Transceiver CC 2420 [41] ist für den Versand und Empfang von IEEE 802.15.4-Rahmen [25] ausgelegt. Ein Teil eines solchen Rahmens, wie

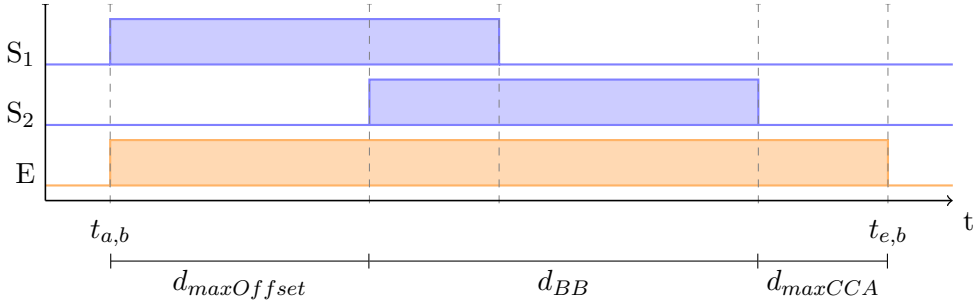


Abbildung 2.2.: Ungenauigkeiten beim Erkennung der Medienbelegung

etwa ein Teil des PHY-Headers, ist durch die Hardware fest vorgegeben. Daher ist z. B. die Übertragungsdauer nach unten und oben begrenzt und nur in diskreten Schritten wählbar.

Dennoch kann diese Hardware zum Versenden von Black Bursts eingesetzt werden. Da nur die Energie einer Übertragung von Bedeutung ist, kann die Übertragung eines IEEE 802.15.4-Rahmens als Implementierung eines dominanten Black Bursts genutzt werden.

Zur Erkennung von Black Bursts muss der Transceiver die Möglichkeit bieten, den Belegungszustand des Mediums zu erfassen und nach außen anzuzeigen. Beim CC 2420 wird dies über eine diskrete Integration der Momentanleistung über ein festes Zeitfenster erreicht. Überschreitet der so ermittelte Wert einen einstellbaren Schwellwert, wird das Medium als belegt erkannt. Dieser Vorgang wird als CCA (*Clear Channel Assessment*) bezeichnet. Der CC 2420 kann zusätzlich noch prüfen, ob es sich bei erkannter Medienbelegung tatsächlich um einen IEEE 802.15.4-Rahmen handelt. Da sich Black Bursts jedoch überlagern dürfen, können sich dadurch entsprechend IEEE 802.15.4-Rahmen überlagern, welche dann in der Regel nicht mehr als solche erkannt werden können. Daher wird von diesem Merkmal des Transceivers kein Gebrauch gemacht und nur die Energieerkennung als Belegungserkennung genutzt.

2.2.3 Ungenauigkeiten beim Erkennen der Medienbelegung

Für den CCA-Vorgang wird die Leistung über einen bestimmten Zeitraum integriert. Je nachdem, wie hoch die empfangene Leistung ist, kann der eingestellte Schwellwert nach einem Belegungsbeginn bereits schon am Anfang des Zeitfensters erreicht werden, oder erst zum Ende hin. Eine ähnliche Überlegung gilt für das Erkennen des Endes einer Medienbelegung. Die maximale Verzögerung entspricht hierbei genau der Zeitspanne der zugrundeliegenden Integration. Diese Dauer wird im Folgenden mit d_{maxCCA} bezeichnet.

Eine weitere Quelle für Ungenauigkeiten ist die Asynchronität mehrerer Sender beim synchronisierten Senden eines Black Bursts. Dies ist in Abb. 2.2 gezeigt. Die Knoten sind nie perfekt synchronisiert, d. h. ihre Uhren sind zueinander versetzt. Durch periodische Resynchronisation können die divergierenden Uhren zwar wieder zusammengebracht werden, jedoch muss schon zum Ende der Resynchronisation mit Synchronisationsfehlern gerechnet werden. In Kap. 2.3 wird ein Verfahren vorgestellt, das aufbauend auf Black Bursts eine Synchronisation herstellt und eine obere Schranke für diese Ungenauigkeit angibt. Hinzu kommt, dass die Uhren weiter auseinanderlaufen, je länger eine Resynchronisation zurückliegt.

In der Abbildung wird die Medienbelegung durch zwei Knoten S_1 und S_2 gezeigt, die mit einer maximalen Ungenauigkeit von $d_{maxOffset}$ synchronisiert sind. Möchten diese Knoten zeitgleich einen Black Burst senden, beginnt der Knoten mit der späteren Uhr (in der Abbildung S_2) um genau $d_{maxOffset}$ später mit der Übertragung als der andere Knoten.

In der Folge ist der beim Empfänger E wahrgenommene Black Burst um $d_{maxOffset}$ länger als die nominale Länge des Black Bursts d_{BB}. Im Bild kommt noch die Ungenauigkeit des CCA-Mechanismus, wie oben beschrieben, hinzu. Dadurch kann die Erkennung des Belegungsendes um eine weitere Dauer von d_{maxCCA} verzögert werden.

Da einem Empfänger die genauen Verzögerungen nicht bekannt sind, muss immer vom Worst-Case ausgegangen werden: Die Erkennung des Anfangs einer Übertragung kann um bis zu d_{maxCCA} und die des Endes um bis zu $d_{maxCCA} + d_{maxOffset}$ verzögert sein. Da jedoch genau diese Zeitpunkte die Information darstellen, die durch einen Black Burst gegeben wird, wird diese Information durch die Verzögerung verfälscht. Dies muss bei allen Black Burst-basierten Protokollen beachtet werden.

Die Dauer, die das Funksignal braucht, um vom Sender zum Empfänger zu gelangen (Ausbreitungsverzögerung), kann hier vernachlässigt werden, da die Protokolle für den Einsatz in Sensornetzen ausgelegt sind. In diesen Netzen kommen auch keine Leistungsstarken Transceiver zum Einsatz, so dass die physikalische Distanz zweiter miteinander kommunizierender Knoten stark beschränkt ist. Bei einer Distanz von 50 m beträgt die Ausbreitungsverzögerung 167 ns und ist somit vernachlässigbar gering.

2.2.4 Optimierung der Erkennungsverzögerung

Wie im vorherigen Abschnitt beschrieben, werden IEEE 802.15.4-Rahmen zur Realisierung von Black Bursts verwendet. Zur Erkennung von Black Bursts dient der CCA-Mechanismus. Unter bestimmten Umständen können dominante Black Bursts, die als regulärer IEEE 802.15.4-Rahmen implementiert sind, noch als solcher empfangen werden. Dies ist vor allem dann der Fall, wenn der Black Burst nur von einem Knoten gesendet wird und somit keine Überlagerung stattfindet.

Zum Empfang eines IEEE 802.15.4-konformen Rahmens müssen Sender und Empfänger synchronisiert sein. Dies geschieht durch eine Synchronisationssequenz, die zu Beginn des Rahmens übertragen wird. Da das Ende der Sequenz durch eine bestimmte SFD-Sequenz (*Start of Frame Delimiter*) markiert wird, wird das Erkennen der Sequenz auch als SFD-Ereignis bezeichnet.

Die Synchronisation, die zwischen Sender und Empfänger aufgebaut wird, kann nun bei der Realisierung des Black Burst-Konzepts insofern ausgenutzt werden, dass die dabei als Nebenprodukt anfallenden Zeitstempel für die Erkennung der Übertragung, also als Erkennung des Black Bursts dienen können. Die Erkennung des SFD muss wie beim CCA-Mechanismus von der Hardware nach außen propagiert werden, so dass die Anwendung den Zeitstempel des Ereignisses sichern kann.

Der Vorteil gegenüber der CCA-Erkennung liegt hier klar in der höheren Genauigkeit. Beim CC 2420 ist die Genauigkeit der Erkennung eines IEEE 802.15.4-Rahmens um etwa den Faktor 43 höher als die Genauigkeit bei der reinen Energieerkennung. Dies kann vor allem bei BBS (Black Burst Synchronization), einem Synchronisationsprotokoll, das auf Black Bursts basiert, vorteilhaft sein. In anderen Synchronisationsprotokollen [1, 2, 15] wird der SFD-Mechanismus zum Erreichen einer sehr genauen Synchronisation genutzt.

Es muss jedoch beachtet werden, dass diese Optimierung nur anwendbar ist, wenn zu einem Zeitpunkt nur ein einziger Sender den Black Burst sendet. Ferner muss auch beachtet werden, dass das Überlagern von Black Bursts, also eine Kollision von IEEE 802.15.4-konformen Rahmen in manchen Fällen die Funktion der Empfängerlogik beeinträchtigen kann (siehe Kap. 4.4.7). Dies hat zur Folge, dass in diesem Fall auch nicht mehr auf die Zeitwerte des CCA-Mechanismus zurückgegriffen werden kann.

Im Normalfall möchten wir daher gerade beim hier verwendeten CC 2420-Transceiver den korrekten Empfang der versendeten Rahmen ausschließen, was durch Anpassung diverser Konfigurationsparameter der Hardware erzielt wird. Beim CC 2420 kann beispielsweise die Synchronisationssequenz verändert werden. Ein Empfänger synchronisiert sich dann nicht mehr mit einem Sender, der eine andere Sequenz verwendet.

Wenn allerdings dem Black Burst-basierten Protokoll zufolge garantiert werden kann, dass es nur einen Sender gibt, können wir diese Optimierung anwenden und auf diese Methode zur Erkennung von Black Bursts zurückgreifen.

Es soll hierbei klargestellt sein, dass es sich bei dieser Methode konzeptionell nicht mehr um Black Bursts handelt. Es ist viel mehr die Ausnutzung eines Spezialfalls, die an wenigen Stellen eingesetzt werden kann, um der Unzulänglichkeit der Hardware, insbesondere der Verzögerung der Belegungserkennung, entgegenzuwirken.

2.2.5 Anwendung von Black Bursts in Kommunikationsprotokollen

Mit der gewählten Kodierung sollen nun zwei Übertragungsarten näher vorgestellt werden. Bei der *kooperativen Übertragung* wird ein Wert netzwerkweit, das heißt auch zu Stationen, die nicht in direkter Nachbarschaft des Initiators stehen, propagiert. Es kann bei dieser Übertragungsart auch mehrere Initiatoren geben, die den gleichen Wert propagieren. Dies wird dadurch erreicht, dass alle Stationen, die den Wert empfangen haben, in einer nächsten Runde die Übertragung wiederholen. Dabei auftretende Überlagerungen wirken sich aufgrund der Kollisionsresistenz von Black Bursts nicht destruktiv aus.

Bei der *arbitrierenden Übertragung* können mehrere Werte zeitgleich übertragen werden, wobei sich zum Ende der Übertragung der größte (oder wahlweise kleinste) Wert durchsetzt. Ähnlich wie das Arbitrierungsverfahren beim CAN-Bus [28] wird hierbei ausgenutzt, dass dominante Bits eine höhere Priorität haben als rezessive Bits. Beide Verfahren werden in den folgenden Abschnitten nochmals näher erläutert.

2.3 Black Burst Synchronization

Aufbauend auf dem Black Burst-Konzept wurde das Synchronisationsprotokoll BBS [23, 22] (Black Burst Synchronization) entwickelt. Das Verfahren, das in einer Master-basierten, dezentralisierten und hybriden Variante eingesetzt werden kann, zielt in erster Linie auf eine netzweite *Ticksynchronisation* ab. Zusammenfassend bietet das Master-basierte Verfahren eine genauere Synchronisation, während das dezentralisierte Verfahren keinen Masterknoten erfordert.

Anders als bei einer Zeitsynchronisation werden bei einer Ticksynchronisation keine absoluten Zeitwerte einer Referenzuhr ausgetauscht, sondern nur Referenzzeitpunkte (*Ticks*) synchronisiert. Für viele Anwendungen, insbesondere eine zeitliche Einteilung des Mediums und Duty-Cycling, ist eine Ticksynchronisation ausreichend. Erfordert eine Anwendung

eine Zeitsynchronisation, kann diese aufbauend auf der vorhandenen Ticksynchronisation ohne großen Mehraufwand hergestellt werden.

Für jede der BBS-Varianten können maximale Tickabweichung, das heißt eine maximale Synchronisationsungenauigkeit, eine konstante Konvergenzdauer und obere Schranken für die Komplexität in Bezug auf Rechenleistung und Speicher angegeben werden.

Im Folgenden werden alle relavanten Aspekte des Master-basierten BBS-Verfahrens erklärt. Die anderen beiden Varianten werden kurz am Ende des Unterkapitels vorgestellt.

2.3.1 Übersicht über das Master-basierte BBS-Verfahren

Beim Master-basierten Verfahren gibt es zwei Rollen, die die Knoten einnehmen können: In jedem Netzwerk gibt es genau einen *Master*. Ziel des Verfahrens ist die Verteilung des Referenzzeitpunkts des Masterknotens. Alle anderen Stationen nehmen die Rolle der *Slaves* ein. Ein Slaveknoten wird während einer Synchronisationsphase durch den Master oder einen bereits synchronisierten Slave synchronisiert und propagiert anschließend den Referenzzeitpunkt weiter zu noch unsynchronisierten Slaves.

Die *Synchronisationsphase* wird in $n_{maxHops}$ *Synchronisationsrunden* unterteilt. Pro Runde wird ein weiterer Hop synchronisiert. So wird der Referenzzeitpunkt ausgehend von einem Masterknoten durch das gesamte Netzwerk propagiert. Dabei steht $n_{maxHops}$ für den maximalen Netzwerkdurchmesser. Diese Größe wird im nächsten Abschnitt nochmals näher betrachtet.

Eine Synchronisationsphase beginnt mit der ersten Synchronisationsrunde. Dazu sendet der Masterknoten zu einem bestimmten Referenzzeitpunkt t_0 eine Synchronisationsnachricht, die als *Tickrahmen* bezeichnet wird und mit Black Bursts kodiert ist.

Im Folgenden wird nun ein Slaveknoten betrachtet, der sich anhand des Tickrahmens synchronisiert. Empfängt dieser den Tickrahmen, so kann er aus dem Startzeitpunkt des Rahmenempfangs t_{rxTick} und der Rundennummer n_{Round}, die ab 0 beginnend in den Tickrahmen kodiert ist, den Beginn der Phase t_0 zurückrechnen:

$$t_0 = t_{rxTick} - n_{Round} \cdot d_{Round} \tag{2.1}$$

Dabei bezeichnet d_{Round} die Rundendauer. Sie entspricht der Dauer, die zum Versand eines Tickrahmens (Kap. 2.3.3) nötig ist. Eine genauere Betrachtung der Rundendauer wird in Kap. 2.3.4 vorgenommen.

Ist ein Slaveknoten in einer Phase synchronisiert und $n_{maxHops}$ noch nicht erreicht, nimmt der Knoten an allen verbleibenden Runden aktiv teil, das heißt er sendet selbst die Tickrahmen. Mögliche Überlagerungen durch Tickrahmen anderer Knoten sind durch die gewählte Kodierung mit Black Bursts nicht destruktiv, da alle Knoten den gleichen Tickrahmen, also die gleiche Black Burst-Sequenz senden. Es handelt sich folglich um eine kooperative Übertragung.

In der initialen Version des Verfahrens, wie es in [23] beschrieben ist, sendet jeder Knoten maximal einen Tickrahmen pro Phase, und zwar in genau der Runde, nach der er selbst synchronisiert wurde. Dies wurde dahingehend geändert, dass nun jeder synchronisierte Knoten in jeder Runde einen Tickrahmen versendet, bis $n_{maxHops}$ Runden erreicht sind. Vorteil dieser Variation ist, dass ein Knoten, der einen Tickrahmen durch kurzzeitige Störungen nicht korrekt empfangen konnte, noch die Chance hat, in dieser Phase synchronisiert zu werden. Als Nachteil kann man allerdings den erhöhten Energiebedarf sehen, da der

Abbildung 2.3.: Aufbau eines Tickrahmens

Transceiver im ursprünglichen Verfahren nach dem Senden des Tickrahmens abgeschaltet werden kann. Denkbar sind daher auch Lösungen zwischen diesen zwei Extrempunkten, so dass ein Knoten nach seiner Synchronisation an beispielsweise zwei weiteren Runden teilnimmt.

2.3.2 Anforderungen an die Netzwerktopologie

Da in jeder Runde ein weiterer Hop synchronisiert wird und synchronisierte Knoten den Tick weitergeben, sind keine speziellen Anforderungen an die Topologie nötig. Es muss lediglich sichergestellt sein, dass in einer Phase alle Knoten erreicht werden. Während der Phase gibt es aufgrund des Fehlens von globalem Wissen keine Möglichkeit, herauszufinden, ob alle Knoten erreicht wurden. Daher muss die Terminierung a priori konfiguriert werden: der statische Parameter $n_{maxHops}$ (im Folgenden auch *Netzwerkdurchmesser*) gibt die maximale Distanz zweier Knoten in Anzahl an Hops an. Dabei wird die Kommunikationsreichweite als Grundlage genommen. Zu beachten ist allerdings, dass Black Bursts oft über diesen Bereich hinaus noch erkannt werden können: Die Sensingreichweite gibt den Bereich an, in dem noch die Energie einer Übertragung (insbesondere also ein Black Burst) erkennbar ist.

Der Parameter $n_{maxHops}$ ist als obere Schranke zu sehen. Wird er größer gewählt als der tatsächliche Netzwerkdurchmesser, wirkt sich dies zunächst nicht auf die tatsächlich erreichte Genauigkeit der Synchronisation aus. Allerdings muss für den maximalen globalen Synchronisationsfehler ein höherer Wert angenommen werden, da sich der Fehler mit jedem weiteren Hop akkumuliert. Außerdem erhöht sich die Gesamtdauer des Verfahrens, da entsprechend die Anzahl an Runden steigt und auch die Länge eines Tickrahmens mit dieser Größe zusammenhängt.

Da keine speziellen Anforderungen an die Topologie gestellt werden, ist das Verfahren auch robust gegenüber Änderungen der Topologie. Das heißt, dass Knoten zueinander bewegt werden können und die Synchronisation weiterhin aufrecht erhalten wird, solange dabei der maximale Durchmesser $n_{maxHops}$ nicht überschritten wird.

2.3.3 Aufbau eines Tickrahmens

Ein Tickrahmen besteht aus einem SOF-Bit (*Start Of Frame*), welches den Anfang der Synchronisationsrunde markiert, der Rundennummer und einem Paritätsbit. Der Aufbau ist in Abb. 2.3 visualisiert. Die Bestandteile werden im Folgenden detailliert vorgestellt.

Rundennummer: Zur Synchronisation wird bei Empfang eines Tickrahmens nicht nur dessen Beginn, sondern auch die aktuelle Rundennummer benötigt. Da unsynchronisierte Knoten keine Information darüber haben, muss diese durch den empfangenen Tickrahmen zur Verfügung gestellt werden. Die Rundennummer ist eine Ganzzahl, daher bietet sich die

Darstellung der Zahl als Dualzahl an. Die einzelnen Bits werden dann mit Black Bursts kodiert als Teil des Tickrahmens versendet.

Die größte Rundennummer, die während einer Phase auftritt ist, wie im vorherigen Abschnitt beschrieben, genau $n_{maxHops}$. Die Anzahl an Bits n_{Bits}, die dazu nötig ist, berechnet sich wie folgt:

$$n_{Bits} = \lceil \log_2(n_{maxHops}) \rceil \tag{2.2}$$

In [23] wurde die Rundennummer bei 1 beginnend gezählt und die Anzahl an Bits daher durch $\lceil \log_2(n_{maxHops}+1) \rceil$ berechnet. Dies dient der Redundanz, da so (zusammen mit dem SOF-Bit und dem Prüfbit) in jedem Tickrahmen immer mindestens zwei dominante Bits vorhanden sind. Im hier vorgestellten Verfahren wird die interne Rundennummer dagegen ab 0 gezählt. Somit kann in manchen Fällen ein Bit gespart werden. Um dennoch mindestens zwei dominante Bits pro Tickrahmen zu erhalten, wurde zusätzlich die Berechnung des Prüfbits geändert.

Start Of Frame: Für unsynchronisierte Knoten ist der Beginn eines Tickrahmens unbekannt. Folglich ist auch das Ende unbekannt. Würde ein Tickrahmen lediglich aus der Rundenummer bestehen, könnten die einzelnen Black Bursts nicht mehr ihrer Position zugeordnet werden. Ein Dekodieren des Tickrahmens wäre daher unmöglich.

Es ist daher unerlässlich, einen Bezugspunkt einzuführen, der aus einem Black Burst einer Länge größer 0 besteht und eindeutig den Beginn eines Tickrahmens markiert. Dieses Bit wird im Folgenden als SOF-Bit (*Start Of Frame*) bezeichnet. In [23] ist es auch als MP-Bit (*Master Present*) deklariert.

Diesem SOF-Bit kommt noch eine weitere zentrale Bedeutung zu: Der Beginn des Tickrahmens, der durch das SOF-Bit markiert wird, ist gleichzeitig der Beginn einer Synchronisationsrunde. Dies ist vor allem für Knoten, die dem Netzwerk neu betreten von Bedeutung: Um an der Kommunikation teilzunehmen, müssen diese erst erfolgreich synchronisiert werden.

Um zu verhindern, dass andere Übertragungen fälschlicherweise als Synchronisationsrunde erkannt werden, wurde die Dauer des SOF-Bits erhöht. Der Black Burst, der als SOF versendet wird, ist so der einzige Black Burst dieser Länge und erlaubt das eindeutige Erkennen eines Tickrahmens.

SOF-Optimierung: Als weitere Optimierung kann hier der Fakt genutzt werden, dass der Tickrahmen der ersten Runde nur vom Master gesendet wird und somit keine Kollisionen stattfinden können. Wie in Kap. 2.2.4 beschrieben, kann das SOF-Bit der ersten Runde daher als IEEE 802.15.4-Rahmen gesendet und auch als solcher empfangen werden. Dies erwirkt eine genauere Synchronisation auf dem ersten Hop, was auch die Genauigkeit im gesamten Netzwerk verbessert.

Knoten, die zum Master in Sensing- aber nicht in Kommunikationsreichweite stehen, können das SOF-Bit zwar immer noch über den CCA-Mechanismus wahrnehmen, doch sie verwerfen den Tickrahmen und werden erst in der zweiten Runde synchronisiert. So kann davon ausgegangen werden, dass die Synchronisationsgenauigkeit, die in der ersten Runde erzielt wurde, nur von der Verzögerung des SFD-Mechanismus abhängt.

Wenn ein Tickrahmen empfangen wird, wird dessen Beginn, d. h. der Beginn des SOF-Bits als Referenzzeitpunkt verwendet. Dies bedeutet, dass sich lediglich Verzögerungen beim

Erkennen des SOF-Bits negativ auf die Genauigkeit der erzielten Synchronisation auswirken. Daher ist es ausreichend, die Optimierung lediglich auf das SOF-Bit anzuwenden.

Prüfbit: Als weitere Maßnahme gegen eine falsche Erkennung eines Tickrahmens wird dem Rahmen eine einfache Prüfsumme in Form eines Paritätsbits angehängt. Dieses Verfahren kann das Kippen eines Bits (1-Bitfehler) erkennen. Dazu wird die Anzahl dominanter Bits n_{dom} im Tickrahmen bestimmt. Das Paritätsbit p gibt dann an, ob diese Zahl ungerade ($p = 1$) oder gerade ($p = 0$) ist. Mathematisch gesehen wird also $p = (n_{dom} + 1) \mod 2$ berechnet. Durch diese Berechnung ist sichergestellt, dass ein Tickrahmen unabhängig von der Rundennummer neben dem dominanten SOF-Bit immer mindestens einen weiteren dominanten Black Burst enthält.

2.3.4 Analyse des Verfahrens

Das Black Burst-Konzept bietet obere Schranken für die Verzögerungenen von Übertragungsvorgängen. Da Black Burst Synchronisation darauf aufbaut, können weitere deterministische Größen bestimmt werden. In diesem Kapitel wird die Konvergenzdauer, also die Dauer, die für das Ausführen einer Synchronisationsphase benötigt wird, und das maximale Tick-Offset, also der maximale Synchronisationsfehler, analytisch hergeleitet.

Die Herleitung weiterer Größen wie Komplexität der Berechnung, Speicherbedarf, etc. steht nicht im Fokus dieser Arbeit und kann [23] entnommen werden.

2.3.4.1 Bitdauer

Die Dauer eines einzelnen Bits gibt die zeitliche Distanz zwischen konsekutiven Black Bursts an, so dass diese noch erfolgreich verarbeitet werden können. Diese Dauer, die mit d_{Bit} bezeichnet ist, richtet sich nach der Übertragungsdauer eines dominanten Black Bursts d_{BB}, der verwendeten Hardware und weiteren Verzögerungen, die im Folgenden näher betrachtet werden.

Dauer des Sendevorgangs: Zunächst beschränken wir uns nur auf Sendevorgänge eines einzelnen Senders. Als Mindestanforderung an die Bitdauer dient in dieser Betrachtung die Rate, mit der aufeinander folgende Black Bursts gesendet werden können. Die Dauer des gesamten Sendevorgangs wird nicht nur von der eigentlichen Übertragung, sondern auch von vielen weiteren Vorgängen bestimmt, wovon jeder mit einer Verzögerung belegt ist.

1. Die Hardware muss für die Übertragung vorbereitet werden. Ein Großteil dieser Vorbereitung kann bereits lange vor der eigentlichen Übertragung passieren und kann bei gleichen konsekutiven Black Bursts auch erheblich abgekürzt werden. Zur Bestimmung einer unteren Schranke der Bitdauer muss jedoch davon ausgegangen werden, dass aufeinander folgende Black Bursts nicht den gleichen Aufbau haben. Ein Beispiel für diesen Fall ist das verlängerte SOF-Bit zum Anfang jeder Synchronisationsrunde, welchem ggf. ein regulärer dominanter Black Burst folgt.

   Die Dauer dieses Vorgangs wird als $d_{TX,pre}$ bezeichnet.

2. Der Zeitpunkt, zu dem der Prozessor den Sendebefehl an die Funkhardware leiten soll, wird vorberechnet und ein Timer auf diesen Zeitpunkt gesetzt. Der Timer löst

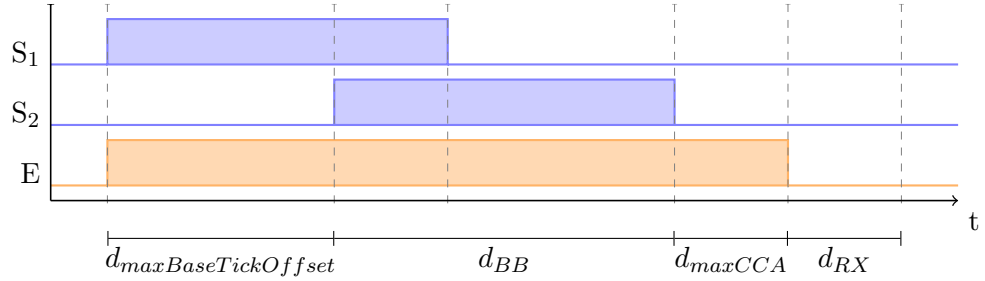


Abbildung 2.4.: Zusammensetzung der Bitdauer aus Sicht des Empfangsvorgangs

beim Erreichen des Zeitpunkts eine Unterbrechung aus. Das Auslösen und Bearbeiten der Unterbrechungsroutine ist ebenfalls mit einer Verzögerung belastet. Auch das Weiterleiten des Befehls nimmt eine Zeitspanne in Anspruch.

Die Gesamtdauer der beiden Vorgänge wird als d_{TX} deklariert.

3. Die Hardware muss vom Empfangs- in den Sendemodus wechseln. Dazu muss der Frequenzgenerator neu kalibriert werden.

   Diese Umschaltdauer wird als d_{rxtx} angegeben.

Zusätzlich muss beachtet werden, dass diese Zeiten aus konstanten und variablen Anteilen bestehen. Die Bitdauer soll derart angegeben werden, dass es immer möglich ist, nach dem Ablauf einer Bitdauer einen nächsten Black Burst zu senden. Daher sind die Obergrenzen der einzelnen Verzögerung zu wählen.
Die Mindestanforderung an die Bitdauer aus Sicht eines Senders wie folgt zusammen:

$$d_{Bit} \geq d_{TX,pre} + d_{TX} + d_{rxtx} + d_{BB} \tag{2.3}$$

Da jeder Knoten beim Master-basierten BBS-Verfahren in jeder Runde entweder nur empfängt oder nur sendet, muss nach dem Senden eines Black Bursts nicht in den Empfangsmodus gewechselt werden. Der CC 2420 beginnt diese Transition zwar automatisch nach jedem Sendevorgang, wird jedoch bei einem weiteren Sendevorgang abgebrochen. Dieses Verhalten wurde in Kap. 4.4.4 geprüft. Somit fließt die Umschaltdauer d_{txrx}, die zum Umschalten vom Sende- in den Empfangsmodus nötig ist, nicht in die Berechnung der Bitdauer ein. In diesem Punkt unterscheidet sich daher die Herleitung der Bitdauer, wie sie in [23] angegeben ist.

Dauer des Empfangsvorgangs: Jedes Bit benötigt eine gewisse Verarbeitungszeit, in der seine Position innerhalb des Tickrahmens bestimmt wird. Beachtet werden muss dabei auch, dass das Bit erst zum Ende des Empfangs eines Black Bursts verarbeitet wird, da die Länge der Black Bursts eine relevante Information darstellt. Daher ist auch der späteste Zeitpunkt, zu dem das Ende des Black Bursts erkannt werden kann, von besonderem Interesse.

Die Herleitung der Anforderung an die Bitdauer ist analog zu den Überlegungen in Kap. 2.2.3. Neu ist, dass die maximale Abweichung der sendenden Knoten während der Synchronisation $d_{maxBaseTickOffset}$ statt $d_{maxOffset}$ ist. Hinzu kommt außerdem die Beachtung der Verarbeitungsverzögerung. Die Verzögerungsquellen im Worst-Case sind in Abb. 2.4 dargestellt:

1. Zwei Sender (S_1 und S_2) senden den gleichen Black Burst, sind dabei jedoch maximal asynchron. So beginnt die Übertragung von S_2 erst $d_{maxBaseTickOffset}$ später als die von S_1.

2. Das Erkennen des Endes der Medienbelegung ist um eine Zeitspanne d_{maxCCA} verzögert, die, wie in Kap. 2.2.3 beschrieben, durch den CCA-Mechanismus entsteht.

3. Das Ende der erkannten Medienbelegung löst eine Unterbrechungsroutine aus. Die Unterbrechungsroutine muss die Position des erkannten Black Bursts innerhalb des Tickrahmens ermitteln und benötigt hierfür eine Verarbeitungszeit. Erst wenn diese Verarbeitung abgeschlossen ist, kann der Beginn eines weiteren Black Bursts adäquat erkannt werden.

   Die Dauer, die für Aufruf und Behandlung der Unterbrechung notwendig ist, wird mit d_{RX} bezeichnet.

Auch hier sollen wieder die Oberschranken der einzelnen Verzögerungen gewählt werden. Insgesamt ergibt sich also als Anforderung an die Bitdauer aus Empfangssicht:

$$d_{Bit} \geq d_{maxBaseTickOffset} + d_{BB} + d_{maxCCA} + d_{RX} \tag{2.4}$$

Insbesondere ist dies auch die maximale Länge einer erkannten Medienbelegung durch einen Black Burst. Diese Zeitspanne wird als $d_{BB,max}$ bezeichnet:

$$d_{BB,max} = d_{maxBaseTickOffset} + d_{BB} + d_{maxCCA} + d_{RX} \tag{2.5}$$

Eindeutigkeit der Bitzuordnung beim Empfänger: Die variablen Anteile der Verzögerungen führen letztlich dazu, dass Zeitpunkte, zu denen ein Empfänger Ereignisse wahrnimmt, Ungenauigkeiten beinhalten. Um falsche Rückschlüsse zu verhindern, sind daher die Abstände der gemessenen Zeitpunkte kritisch, da über diese ein dominanter Black Burst wieder seiner Bitposition innerhalb des Tickrahmens zugeführt wird.

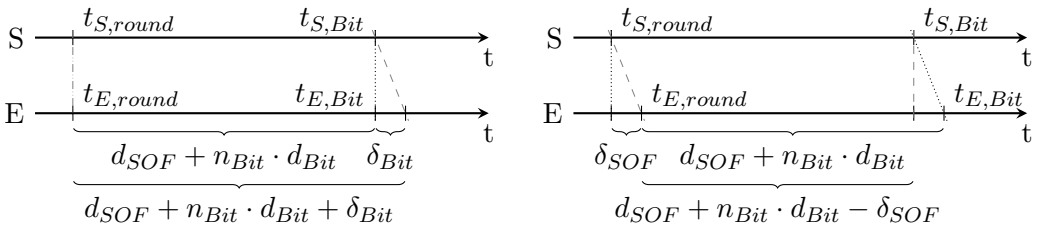
Es soll nun der Versand und Empfang eines dominanten Black Bursts innerhalb eines Tickrahmens betrachtet werden. Dabei steht vor allem die Rückgewinnung der Bitposition im Vordergrund. Am Ende wird daraus eine Bedingung an die Bitdauer abgeleitet, die eine erfolgreiche Positionsbestimmung garantiert.

Aus Sicht eines Senders S habe eine momentane Runde den Startzeitpunkt $t_{S,round}$. Sei n_{Bit} die Position eines dominanten Bits des Tickrahmens. Die Position des Bits wird dabei beginnend bei 0 nach dem SOF-Bit gezählt, welches die Dauer d_{SOF} hat. Beispielsweise gilt demnach $n_{Bit} = 0$ für das erste Bit der Rundennummer. Den Sendezeitpunkt des Bits berechnet der Knoten wie folgt:

$$t_{S,Bit} = t_{S,round} + d_{SOF} + n_{Bit} \cdot d_{Bit} \tag{2.6}$$

Ein Empfänger, der das SOF-Bit des Tickrahmens erkennt, sichert den Startzeitpunkt als $t_{E,round}$, da dieses Ereignis den Beginn des Tickrahmens markiert. Basierend auf Formel 2.6 kann auf Empfängerseite der erwartete Empfangszeitpunkt $t_{E,Bit}$ von Bit n_{Bit} berechnet werden.

Die Empfangszeitpunkte des SOF und des Bits sind jedoch durch Verzögerungen verfälscht. Die Verzögerungen der beiden Zeitpunkte sollen zunächst einzeln betrachtet werden. Dies ist in den Teilabbildungen von Abb. 2.5 dargestellt:



(a) Verzögerung des Runden-Bits um δ_{Bit}

(b) Verzögerung des SOF um δ_{SOF}

Abbildung 2.5.: Verzögerungen von SOF und Bits eines Tickrahmens

(a) Das SOF wurde ohne Verzögerung und das Bit mit einer Verzögerung von δ_{Bit} empfangen. Somit ist der tatsächliche Empfangszeitpunkt gegenüber dem erwarteten um δ_{Bit} nach hinten versetzt.

(b) Das SOF wurde mit einer Verzögerung von δ_{SOF} und das Bit ohne Verzögerung empfangen. Somit ist der tatsächliche Empfangszeitpunkt gegenüber dem erwarteten um δ_{SOF} nach vorne versetzt.

Die beiden Verzögerungen δ_{SOF} und δ_{Bit} sind jeweils gerade die Verzögerungen, die zwischen Sende- und Empfangsereignis liegen. Davon sind jedoch nur die variablen Anteile von Bedeutung, da konstante Anteile beim Empfänger wieder subtrahiert werden können. Diese variable Verzögerung setzt sich aus den in Abb. 2.6 gezeigten Einzelverzögerungen zusammen:

1. Die Sendeverzögerung d_{TX} beinhaltet, wie zuvor beschrieben, mehrere einzelne Verzögerungen. Die maximale variable Verzögerung, die in d_{TX} enthalten ist, wird im Folgenden als $d_{TX}^{\mathcal{V}}$ bezeichnet.

2. Die Umschaltdauer d_{rxtx} wird zunächst als konstant angenommen. Variable Anteile können in $d_{TX}^{\mathcal{V}}$ einbezogen werden. Daher ist d_{rxtx} für diese Betrachtung irrelevant.

3. Die Erkennungsverzögerung d_{maxCCA} ist bereits eine obere Schranke einer variablen Verzögerung. Da der Bezeichner vom Datenblatt und auch in anderen Quellen verwendet wird, soll er an dieser Stelle beibehalten werden.

4. Auf Empfängerseite ist für diese Betrachtung lediglich die Dauer von Interesse, die vom Auftreten der Unterbrechung bis zur Sicherung des Zeitstempels vergeht.

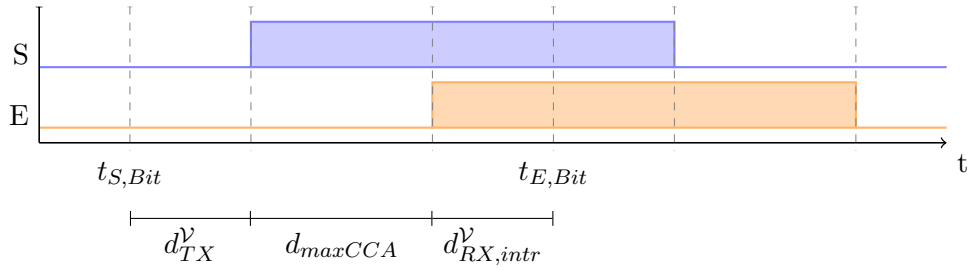


Abbildung 2.6.: Verzögerungen zwischen Sendebeginn und Erkennung

Diese soll mit $d_{RX,intr}$ bezeichnet werden und ist somit nur ein Teil der gesamten Verarbeitungsverzögerung d_{RX}, wie sie in vorherigen Betrachtungen verwendet wurde. Ferner ist auch wieder nur die maximale variable Verzögerung dieser Größe, die mit $d^{\mathcal{V}}_{RX,intr}$ bezeichnet wird, relevant.

Die maximale variable Verzögerung, die zwischen einem Sende- und dem dazugehörigen Empfangsereignis liegt, wird mit $\widehat{\delta}_{tr}$ bezeichnet:

$$\widehat{\delta}_{tr} = d^{\mathcal{V}}_{TX} + d_{maxCCA} + d^{\mathcal{V}}_{RX,intr} \tag{2.7}$$

So können nun δ_{Bit} und δ_{SOF} jeweils einen Wert im Intervall $[0, \widehat{\delta}_{tr}]$ annehmen. In den Extremfällen nimmt eine der Verzögerungen den Maximalwert $\widehat{\delta}_{tr}$ an, während die jeweils andere 0 ist. Der Beginn eines Bits kann dadurch um bis zu $\widehat{\delta}_{tr}$ vor bzw. nach seiner erwarteten Position $t_{E,round}$ erkannt werden. Hierzu sei angemerkt, dass dieser Fall praktisch nicht auftritt, da die Verzögerungen immer größer als 0 sind. Außerdem variieren einige der variablen Anteile über mehrere Übertragungen hinweg wenig und heben sich somit zu einem großen Teil bei der Bestimmung des Abstands zwischen dem SOF und dem Bit wieder gegenseitig auf. Dennoch soll in dieser Betrachtung der Worst-Case angenommen werden.

Um ein empfangenes Bit über den Empfangszeitpunkt wieder eindeutig seiner Bitposition zuzuführen, dürfen sich die Empfangszeitpunkte benachbarter Bits nicht überschneiden. Unter Beachtung der Extremfälle bedeutet diese Forderung, dass der späteste Zeitpunkt eines Bits n_{Bit} vor dem frühesten Zeitpunkt des nachfolgenden Bits $n_{Bit}+1$ liegen muss. Es muss also gelten:

$$\begin{aligned} & t_{E,round} + d_{SOF} + n_{Bit} \cdot d_{Bit} + \widehat{\delta}_{tr} \;<\; t_{E,round} + d_{SOF} + (n_{Bit}+1) \cdot d_{Bit} - \widehat{\delta}_{tr} \\ \iff & \widehat{\delta}_{tr} \;<\; d_{Bit} - \widehat{\delta}_{tr} \\ \iff & d_{Bit} \;>\; 2 \cdot \widehat{\delta}_{tr} \end{aligned} \tag{2.8}$$

Zusammenfassung: Insgesamt muss für die Bitdauer d_{Bit} also gelten:

$$\begin{aligned} d_{Bit} \overset{(2.3,2.4,2.8)}{>} \max\{\, & d_{TX,pre} + d_{TX} + d_{rxtx} + d_{BB}, \\ & d_{maxBaseTickOffset} + d_{BB} + d_{maxCCA} + d_{RX}, \\ & 2 \cdot (d^{\mathcal{V}}_{TX} + d_{maxCCA} + d^{\mathcal{V}}_{RX,intr}) \,\} \end{aligned} \tag{2.9}$$

SOF-Bit: Da das SOF-Bit nach den Überlegungen in Kap. 2.3.3 als Black Burst mit einer größeren Dauer d_{BB_SOF} implementiert wird, gilt für die Dauer des SOF-Bits analog

$$\begin{aligned} d_{SOF} \geq \max\{\, & d_{TX,pre} + d_{TX} + d_{rxtx} + d_{BB\_SOF}, \\ & d_{maxBaseTickOffset} + d_{BB\_SOF} + d_{maxCCA} + d_{RX} \,\} \end{aligned} \tag{2.10}$$

Die Forderung aus Formel 2.8 kann dabei entfallen, da beim SOF-Bit keine Bitzuordnung stattfindet.

Zusätzlich muss gefordert werden, dass die kleinstmögliche Dauer für die Erkennung eines SOF-Black Bursts größer als die größtmögliche Dauer eines normalen Black Bursts ist. Es muss daher folgende Ungleichung erfüllt sein:

$$d_{BB\_SOF} > d_{BB,max} \tag{2.11}$$

$$\overset{(2.5)}{=} d_{maxBaseTickOffset} + d_{BB} + d_{maxCCA} + d_{RX} \tag{2.12}$$

2.3.4.2 Rundendauer und Konvergenzdauer

Die Dauer einer Runde entspricht der Dauer, die zur Übertragung und Verarbeitung eines Tickrahmens benötigt wird. Die Übertragungsdauer setzt sich dabei aus der Anzahl der im Rahmen enthaltenen Bits und deren Dauer zusammen.

Ein Tickrahmen beginnt mit einem SOF-Bit, welches die Dauer d_{SOF} hat. Dieses wird von n_{Bits} Bits der Dauer d_{Bit} gefolgt, die die Rundennummer kodieren. Nach Formel 2.2 gilt $n_{Bits} = \lceil \log_2(n_{maxHops}) \rceil$. Das angehängte Paritätsbit wird wie die Rundennummer kodiert und ist somit ebenfalls von der Dauer d_{Bit}.

Zusätzlich soll nach der Runde noch eine feste Dauer zur Verarbeitung des Tickrahmens zur Verfügung stehen, um basierend auf dem Start der empfangenen Runde und der Rundennummer den Tick zurückzurechnen und den Tickrahmen für die nächste Runde vorzubereiten. Diese wird mit $d_{frameGuard}$ bezeichnet.

Somit gilt nun:

$$d_{Round} = d_{SOF} + (n_{Bits} + 1) \cdot d_{Bit} + d_{frameGuard} \tag{2.13}$$

Insbesondere sei hier vermerkt, dass die Rundendauer logarithmisch vom maximalen Netzwerkdurchmesser $n_{maxHops}$ abhängt.

Die Konvergenzdauer bezeichnet die Dauer eines Synchronisationsvorgangs, also einer gesamten Phase. Diese setzt sich aus $n_{maxHops}$ Runden zusammen.

Somit gilt:

$$d_{Conv} = n_{maxHops} \cdot d_{Round} \tag{2.14}$$

Ferner gilt daher $d_{Conv} \in \mathcal{O}(n_{maxHops} \cdot \log(n_{maxHops}))$. Die Konvergenzdauer hängt also superlinear vom maximalen Netzwerkdurchmesser ab. Da alle Größen jedoch fest vorgegeben oder statisch konfiguriert werden, kann die Konvergenzdauer im laufenden Betrieb als konstant gesehen werden.

2.3.4.3 Maximales Tick-Offset nach der Synchronisation

In Formel 2.7 wurde beschrieben, dass eine Übertragung (und somit der Zeitpunkt des Tickrahmens) um die maximale Verzögerung $\widehat{\delta}_{tr}$ verspätet erkannt werden kann. Somit kann der Referenzzeitpunkt bei einem Knoten um bis zu $\widehat{\delta}_{tr}$ später sein, als beim Sender des Tickrahmens. Dieser Fehler akkumuliert sich mit jedem weiteren Hop.

Dies gilt nicht für den ersten Hop, da in diesem nach Kap. 2.2.4 das SOF-Bit des Tickrahmens als IEEE 802.15.4-Rahmen übertragen wird und so statt der CCA-Verzögerung der geringeren SFD-Verzögerung unterliegt. Diese wird im Folgenden als d_{maxSFD} bezeichnet. Somit hat der erste Hop direkt nach der Synchronisation ein maximales Tick-Offset von

$$d_{maxBaseTickOffset,1} = d^{\mathcal{V}}_{TX} + d_{maxSFD} + d^{\mathcal{V}}_{RX,intr} \tag{2.15}$$

Für alle weiteren Hops ist der maximale Fehler direkt nach der Synchronisation demnach

$$d_{maxBaseTickOffset} = d_{maxBaseTickOffset,1} + (n_{maxHops} - 1) \cdot \widehat{\delta}_{tr} \tag{2.16}$$

In [23] ist eine Optimierung angegeben, die von der Annahme ausgeht, dass es ausreicht, die lokale Synchronisationsungenauigkeit zweier Knoten zu betrachten, die maximal zwei Hops voneinander entfernt sind. Unter der Annahme, dass sich die Knoten nicht zueinander bewegen, ist somit eine Abweichung von

$$d_{maxBaseTickOffset,local} = 2 \cdot \widehat{\delta}_{tr} \tag{2.17}$$

möglich. Diese Optimierung wird in weiteren Überlegungen dieser Arbeit verwendet.

Nach einer erfolgten Synchronisation können aufgrund von Herstellungstoleranzen und der Temperaturabhängigkeit von Schwingquarzen die Uhren der Knoten weiter auseinander laufen. Ein wichtiger Parameter ist hierbei die Rate, mit der die Uhr von der realen Zeit divergiert. Diese Uhrengeschwindigkeit ist mathematisch gesehen die Ableitung der Zeitfunktion der Uhr und wird *Clock Rate* genannt. Das *Clock Skew* gibt überdies die Clock Rate zweier Knoten zueinander an. Ist das maximale Clock Skew bekannt, kann darüber eine obere Schranke für die Divergenz der Uhren angegeben werden. Zusammen mit $d_{maxBaseTickOffset}$ kann so zu jedem Zeitpunkt der momentane maximale Tick-Offset zweier Knoten bestimmt werden. Hierzu sei erwähnt, dass dieses maximale Offset monoton über die Zeit wächst.

Die Synchronisation der Knoten dient zeitlich koordinierten Übertragungen im TDMA-Schema. Daher ist es essentiell, dass das maximale Offset im Zugriffsprotokoll beachtet wird. Um deterministische Garantien geben zu können, muss auch das maximale Tick-Offset zweier Knoten beschränkt sein, was durch periodische Resynchronisation erreicht wird.

2.3.5 Zuordnung von Bits zur Position

Beim Übertragen eines Tickrahmens wird zunächst das SOF-Bit gesendet. Der Startzeitpunkt der Synchronisationsrunde sei aus Sicht des Senders mit $t_{S,round}$ bezeichnet. In dieser Betrachtung soll nun im weiteren Verlauf der Übermittlung ein weiteres dominantes Bit an einer Position $n_{S,Bit}$ innerhalb des Tickrahmens übertragen werden.

Der Startzeitpunkt $t_{S,Bit}$ des Bits kann nach Formel 2.6 berechnet werden. Sind alle Zeitpunkte bekannt, kann umgekehrt die Position des Bits zurückgerechnet werden:

$$n_{S,Bit} = \frac{t_{S,Bit} - t_{S,round} - d_{SOF}}{d_{Bit}} \tag{2.18}$$

Auf Empfängerseite sei der Startzeitpunkt des Tickrahmens, als $t_{E,round}$ und der Startzeitpunkt des dominanten Bits als $t_{E,Bit}$ bezeichnet. Dabei entspricht $t_{E,round}$ dem Empfangszeitpunkt des SOF-Rahmens. Wie in Kap. 2.3.4.1 erklärt, sind diese Zeitpunkte jeweils durch eine Ungenauigkeit von δ_{SOF} bzw. δ_{Bit} verfälscht. Wird nun die Bitposition $n_{E,Bit}$ gemäß Formel 2.18 bestimmt, würde also gelten:

$$\begin{aligned} n_{E,Bit} &= \frac{t_{E,Bit} - t_{E,round} - d_{SOF}}{d_{Bit}} && (2.19) \\ &= \frac{t_{S,Bit} - t_{S,round} - d_{SOF}}{d_{Bit}} + \frac{\delta_{Bit} - \delta_{SOF}}{d_{Bit}} \\ &\overset{(2.18)}{=} n_{S,Bit} + \frac{\delta_{Bit} - \delta_{SOF}}{d_{Bit}} && (2.20) \end{aligned}$$

Das Maximum des Teilterms $\delta_{Bit} - \delta_{SOF}$ wurde bereits in Kap. 2.3.4.1 hergeleitet und entspricht der maximalen Verzögerung $\widehat{\delta}_{tr}$. Durch die Forderung in Formel 2.8 gilt

$$\frac{2 \cdot \widehat{\delta}_{tr}}{d_{Bit}} \; < \; 1 \tag{2.21}$$

Es gilt nach Formel 2.20:

$$\begin{aligned} & n_{S,Bit} - \frac{\widehat{\delta}_{tr}}{d_{Bit}} \;\leq\; n_{E,Bit} \;\leq\; n_{S,Bit} + \frac{\widehat{\delta}_{tr}}{d_{Bit}} \\ \iff \quad & n_{S,Bit} \;\leq\; n_{E,Bit} + \frac{\widehat{\delta}_{tr}}{d_{Bit}} \;\leq\; n_{S,Bit} + \frac{2 \cdot \widehat{\delta}_{tr}}{d_{Bit}} \\ \overset{(2.21)}{\implies} \quad & n_{S,Bit} \;\leq\; n_{E,Bit} + \frac{\widehat{\delta}_{tr}}{d_{Bit}} \;<\; n_{S,Bit} + 1 \end{aligned} \tag{2.22}$$

Das bedeutet, dass durch die Addition $n_{E,Bit} + \frac{\widehat{\delta}_{tr}}{d_{Bit}}$ nie die Grenze zur nächsten Bitposition $n_{S,Bit} + 1$ überschritten wird. Ferner liegt der Term immer zwischen zwei ganzen Zahlen im halboffenen Intervall $[n_{S,Bit}, n_{S,Bit} + 1)$. Die ursprüngliche Bitposition $n_{S,Bit}$ kann daher durch eine Abrundung berechnet werden:

$$n_{S,Bit} = \left\lfloor n_{E,Bit} + \frac{\widehat{\delta}_{tr}}{d_{Bit}} \right\rfloor \tag{2.23}$$

Durch Expansion der Formel kann die Anzahl an Divisionen minimiert und auf eine einzige ganzzahlige Division zurückgeführt werden, die sich in Hardware effizient berechnen lässt:

$$\begin{aligned} n_{S,Bit} &\overset{(2.19)}{=} \left\lfloor \frac{t_{E,Bit} - t_{E,round} - d_{SOF}}{d_{Bit}} + \frac{\widehat{\delta}_{tr}}{d_{Bit}} \right\rfloor \\ &= \left\lfloor \frac{t_{E,Bit} - t_{E,round} - d_{SOF} + \widehat{\delta}_{tr}}{d_{Bit}} \right\rfloor \end{aligned} \tag{2.24}$$

In dieser Darstellung ist nun auch ersichtlich, dass nur noch Zeitpunkte und Zeitspannen verwendet werden, die dem Empfänger bekannt sind.

2.3.6 Protokollverhalten eines Slaves

Wenn ein neuer Slaveknoten dem Netzwerk beitritt, muss dieser zuerst synchronisiert werden, bevor er an der Kommunikation teilnehmen darf. Um potentielle Fehlsynchronisationen zu minimieren, wurden hierzu mehrere Synchronisationszustände eingeführt, die den Fortschritt der Synchronisation beschreiben. Zustandsübergänge werden in Abhängigkeit eines statisch konfigurierbaren Schwellwerts $n_{syncPhaseThreshold}$ ausgelöst.

- `UNSYNCHRONIZED (U)`: In diesem Zustand ist der Knoten nicht synchronisiert. Es wird ohne Timeout auf eine Synchronisationsrunde gewartet, die durch den Empfang eines SOF-Bits eingeleitet wird.

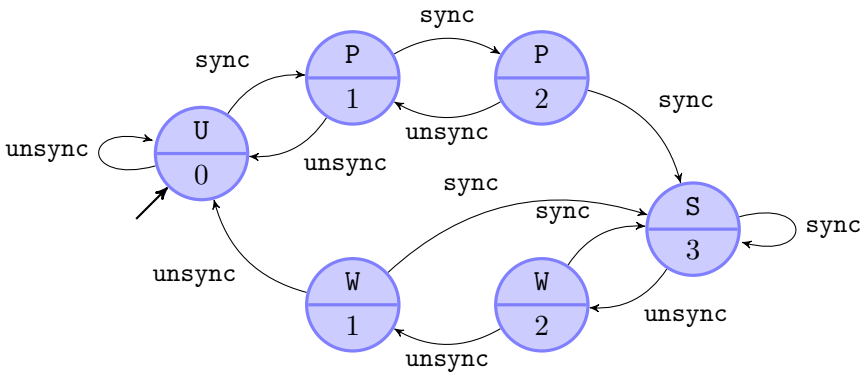


Abbildung 2.7.: Synchronisationsgrad für $n_{syncPhaseThreshold} = 3$

- `PRE SYNCHRONIZED (P)`: Dieser Zustand wird erreicht, sobald der Knoten mindestens einmal erfolgreich synchronisiert wurde. Nach jeder erfolgreichen Synchronisation wird ein interner Zähler inkrementiert, dessen Wert im unteren Teil des Zustandssymbols angegeben ist. Nach einer nicht erfolgreichen Synchronisation wird dieser Zähler wieder dekrementiert. Wird dabei der Wert Null erreicht, wird der Zustand wieder in `UNSYNCHRONIZED` überführt.

- `STRONGLY SYNCHRONIZED (S)`: Nachdem ein Knoten in mindestens $n_{syncPhaseThreshold}$ Phasen erfolgreich synchronisiert wurde, wird dieser Zustand eingenommen. In diesem Zustand darf der Knoten aktiv an der Kommunikation teilnehmen und versendet während einer Synchronisationsphase Tickrahmen.

- `WEAKLY SYNCHRONIZED (W)`: Nachdem ein Knoten im Zustand `STRONGLY SYNCHRONIZED` eine Resynchronisation verpasst, wird dieser Zustand erreicht. Werden in Folge insgesamt $n_{syncPhaseThreshold}$ nicht erfolgreiche Phasen gezählt, geht der Zustand in `UNSYNCHRONIZED` über. Eine erfolgreiche Synchronisation stellt direkt wieder den Zustand `STRONGLY SYNCHRONIZED` her.

In Abb. 2.7 wurde der Zustandsautomat für $n_{syncPhaseThreshold} = 3$ dargestellt. Der Wert des internen Zählers ist in jedem Zustand angegeben.

Zusammenfassend wirken folgende Maßnahmen einer Fehlsynchronisation entgegen:

- Verlängerter Black Burst, der nur für die Kodierung des SOF eingesetzt wird
- Prüfbit über den gesamten Tickrahmen
- Schwellwert an Synchronisationsphasen, der für eine erfolgreiche Synchronisation überschritten werden muss

2.3.7 Dezentralisiertes BBS

Das dezentralisierte BBS-Verfahren erfordert im Gegensatz zum Master-basierten Verfahren keinen Masterknoten. Die Idee ist, den frühesten Referenzzeitpunkt im gesamten Netzwerk zu propagieren, d. h. der Knoten mit der schnellsten Uhr gibt den Tick vor.

Der Vorteil ist, dass es keinen Single Point of Failure gibt. Beim Master-basierten Verfahren fällt nach Wegfall des Masterknotens die Synchronisation und somit die gesamte Kommunikation aus. Dies ist hier nicht der Fall. Dieser Vorteil wird allerdings durch eine höhere Ungenauigkeit der Synchronisation erkauft.

Bei dem Verfahren wird vor dem Auftreten des nächsten Referenzzeitpunkts von allen Knoten das Medium überwacht. Wird kein Tickrahmen-Start bis zum Referenzzeitpunkt erkannt, sendet der Knoten selbst einen Tickrahmen.

Auch bei diesem Verfahren können wieder maximale Abweichungen und Verzögerungen angegeben werden. Für eine genaue Beschreibung des Verfahrens und die Herleitung der entsprechenden Formeln sei [23] verwiesen.

2.3.8 Hybrides BBS

Das hybride BBS-Verfahren setzt sich, wie der Name vermuten lässt, aus beiden Verfahren zusammen. Ist ein Masterknoten vorhanden, wird dieser für die Synchronisation verwendet. Fällt dieser Knoten aus, kann auf das dezentralisierte BBS-Verfahren zurückgegriffen werden. Somit werden die Vorteile beider Verfahren vereint.

Auch hier sei für genaue Details auf [23] verwiesen.

2.4 ACTP

Neben BBS wurden weitere auf Black Bursts aufbauende Protokolle entwickelt. Das Arbitrating/Cooperative Transfer Protocol [14] (ACTP) soll in dieser Arbeit als Beispiel für die arbitrierende Übertragung kurz vorgestellt werden.

Ziel des Verfahrens ist, einen Wert, der ähnlich wie die Rundennummer eines Tickrahmens in eine Bitsequenz zerlegt wird, netzweit zu propagieren. Übertragen mehrere Stationen zeitgleich verschiedene Bitsequenzen, so soll sich die Bitsequenz, die den größten Zahlwert kodiert, während der Übertragung durchsetzen. Durch eine Variation kann auch erreicht werden, dass sich der kleinste Zahlwert durchsetzt. Dies ist der Fall, wenn die Kodierung invertiert wird. Das bedeutet, dass eine 1 dann auf einen rezessiven und eine 0 auf einen dominanten Black Burst abgebildet wird.

In Kap. 2.2.1 wurde bereits erläutert, dass Stationen, die einen rezessiven Black Burst senden, tatsächlich keine Übertragung durchführen und somit die Hardware im Empfangsmodus verweilen kann. So können währenddessen dominante Black Bursts anderer Stationen empfangen werden.

Sobald eine Station während der Übertragung eines rezessiven Black Bursts einen dominanten Black Burst erkennt, bricht sie die Übertragung der eigenen Sequenz ab. In dieser Hinsicht ist das Verfahren vergleichbar mit dem Arbitrierungsverfahren, wie es beim CAN-Bus eingesetzt wird. Der wesentliche Unterschied liegt jedoch darin, dass beim CAN-Bus ein einzelner Bus zum Einsatz kommt, d. h. jede Station kann mit allen anderen direkt kommunizieren. In einem Multi-Hop-Netzwerk ist dies nicht der Fall. Bei ACTP leitet daher jede Station, die selbst keine Sequenz (mehr) überträgt, empfangene Black Bursts weiter, um das netzweite Propagieren des größten Zahlenwerts zu gewährleisten.

Zum Funktionieren des Verfahrens ist es bei der arbitrierenden Übertragung essentiell, dass die Bits des Wertes in einer bestimmten Reihenfolge verarbeitet werden: Das höchstwertige Bit muss als erstes übertragen werden. Nur so kann während der Übertragung bei jeder

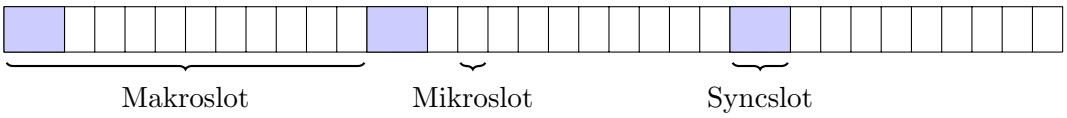


Abbildung 2.8.: Zeitliche Einteilung des Mediums (Slotting) bei MacZ

weiteren Bitposition entschieden werden, ob die eigene Bitsequenz noch mindestens die gleiche Priorität besitzt wie die anderer Stationen.

Einsatz findet das Verfahren bei netzweiten Arbitrierungen. So können z. B. Rahmen (wie bei CAN [28]) mit einer Identifikation versehen werden, die deren Priorität widerspiegelt. Die Station, die den höchsten priorisierten Rahmen zu versenden hat, würde sich bei der Arbitrierung durchsetzen und dürfte dann den eigentlichen Rahmen versenden.

Das genaue Verfahren und dessen Analyse können [11, 14, 13] entnommen werden.

2.5 MacZ

MacZ [7] ist eine MAC-Schicht, die speziell im Hinblick auf Energieeffizienz und QoS-Unterstützung in mobilen Ad-hoc-Netzwerken entwickelt wurde.

Ein Kernkonzept von MacZ ist die zeitliche Einteilung des Mediums in Slots. Die Einteilung ist in Abb. 2.8 dargestellt. Das Medium wird in Makroslots einer festen Länge unterteilt. Diese sind wiederum in einen Syncslot und mehrere gleich große Mikroslots unterteilt. Die Slots werden in virtuelle Slotregionen gruppiert, so dass es innerhalb eines Makroslots wettbewerbsbasierte und reservierungsbasierte Regionen gibt. Auch Duty-Cycling wird über spezielle Idle-Regionen unterstützt.

In den wettbewerbsbasierten Regionen kommen entsprechende Verfahren zum Vermeiden von Kollisionen bei wahlfreiem Medienzugriff zum Einsatz.

Durch die Reservierung von Mikroslots in einer reservierungsbasierten Region kann ein Knoten sicherstellen, als alleinige Station zu senden. Daher kann eine Prüfung der Kanalbelegung oder eine Medienarbitrierung entfallen. Dadurch können Garantien zur maximalen der Übertragung Verzögerungen eingehalten werden.

Für die Funktion des Verfahrens ist es wichtig, dass die Slotgrenzen von allen Knoten eingehalten werden. Daher ist eine Synchronisation notwendig, die eine maximale Synchronisationsungenauigkeit gewährleistet und die durch periodische Resynchronisation aufrecht erhalten wird. Zur Synchronisation kommt dabei das in Kap. 2.3 beschriebene BBS-Verfahren zum Einsatz.

2.6 Related Work

In diesem Kapitel wurde u. A. auf Ungenauigkeiten bei der Erkennung der Medienbelegung hingewiesen, die für das Erkennen von Black Bursts verwendet wird. Diese Ungenauigkeiten sind nicht nur der CCA-Verzögerung durch die verwendete Funkhardware geschuldet, sondern auch auf Laufzeiten der weiteren Verarbeitung durch einen Prozessor (oder Mikrocontroller) zurückzuführen.

Die im Kapitel vorgestellten Protokolle sind keinesfalls die einzigen, bei denen die Ungenauigkeiten bei der Erkennung der Medienbelegung beachtet werden müssen. Im Grunde sind davon alle Protokolle betroffen, die auf Zeitpunkte von Empfangsereignissen achten.

In den folgenden Abschnitten werden einige Protokolle genannt, deren korrekte Funktion sich, wie bei Black Burst-basierte Verfahren, auf genaues Erkennen von Zeitpunkte von Empfangsereignissen stützt.

2.6.1 Protokolle zur Zeitsynchronisation

Viele Protokolle zur Zeitsynchronisation wie z. B. RBS [18], TPSN [21] und TDP [39] verwenden statt des CCA-Mechanismus den SFD-Mechanismus der unterliegenden Funkhardware. In Kap. 2.2.4 wurde erklärt, dass die SFD-Erkennung um eine hohe Größenordnung (etwa Faktor 43 beim CC 2420-Transceiver) genauer ist als die CCA-Erkennung, da die eigentliche Synchronisation mit Hilfe einer Synchronisationspräambel durch die Hardware geschieht. Dadurch können diese Protokolle eine sehr hohe Synchronisationsgenauigkeit erzielen.

Dennoch muss beachtet werden, dass auch das Erkennen des SFD im Transceiver verzögert wird. Zusätzlich muss beachtet werden, dass das SFD-Ereignis vom Prozessor (bzw. Mirkocontroller) erkannt und verarbeitet werden muss. Bei einer Implementierung von Protokollen zur Zeitsynchronisation müssen auch diese Verzögerungsquellen und Verarbeitungsdauern mitbeachtet werden. Für das Master-basierte BBS-Verfahren werden diese Verzögerungen in Kap. 2.3.4 beachtet und in Kap. 5.3.1 für die Imote 2-Plattform näher bestimmt.

2.6.2 Busy Tone-Protokolle

Busy Tone-Protokolle, wie z. B. [38] verwenden Black Bursts variabler Länge zur Arbitrierung bei einem konkurrierenden Medienzugriff. Die Länge des Black Bursts ist dabei proportional zu einer Priorität, die je nach Protokoll aus der Anzahl nicht erfolgreicher Arbitrierungen, der Knoten-Id oder auch einer zufällig gewählten Zahl besteht. Nachdem ein Knoten einen Black Burst gesendet hat, wird der CCA-Mechanismus eingesetzt, um ggf. einen längeren Black Burst einer anderen Station zu erkennen. Wird kein längerer Black Burst erkannt, hat die Station die Arbitrierung gewonnen und anschließend exklusiven Zugriff auf das Medium. Weitere Beispiele für Busy Tone-Protokolle sind [37], [9], [45] und [30].

Bei diesem Vorgehen ist es kritisch, dass nicht nur die CCA-Verzögerung sondern auch die Blindzeiten (während der Umschaltung vom Sende- in den Empfangsmodus) des Transceivers berücksichtigt werden. Die Nichtbeachtung dieser Hardwareverzögerungen, wie es z. B. bei vielen Simulationsmodellen der Fall ist, kann zu einem fehlerhaften Protokolldesign führen (siehe BP-MAC [30]). Darüber hinaus müssen ebenfalls weitere Verzögerungen durch die verarbeitende Hardware einkalkuliert werden. In dieser Arbeit werden diese Verzögerungen für den CC 2420-Transceiver und den PXA 271-Prozessor in Kap. 4 experimentell bestimmt.

2.6.3 Binary Countdown-Protokolle

In Binary Countdown-Protokollen, wie dem vorgestellten ACTP (Kap. 2.4), werden Black Bursts (je nach Protokoll auch *Buzz*-Signal genannt) dazu eingesetzt, eine binär codierte Zahl, die die Priorität der eigentlichen Nachricht widerspiegelt, in einer Sequenz von dominanten

und rezessiven Bits zu propagieren. Die Übertragung der Sequenz findet dabei bitweise statt. Die Übertragung der eigenen Sequenz wird abgebrochen, wenn das momentane Bit der Sequenz rezessiv ist, während ein dominantes Bit empfangen wird. So setzt sich der größte Wert durch und ist am Ende der Arbitrierung allen Stationen bekannt. Neben ACTP sind SYN-MAC [44], CSMA/IC [46], BitMAC [36] und WiDom [34] weitere Beispiele für diese Protokolle.

Auch hier müssen die Erkennungsverzögerungen durch den CCA-Mechanismus und den Prozessor beachtet werden. Da Binary Countdown-Protokolle häufig zwischen Empfangs- und Sendemodus wechseln, haben Umschaltverzögerungen für derartige Protokolle eine besondere Relevanz. Für die Imote 2-Plattform werden diese Verzögerungen in Kap. 4.4.1 und Kap. 4.4.3 bestimmt.

3. Kapitel

Hardware

In dieser Arbeit steht die Imote 2-Sensorplattform im Vordergrund. Die Plattform wurde sämtlichen Annahmen, Tests und Optimierungen zugrunde gelegt. Der Sensorknoten ist daher auch Zielplattform der Implementierung des Kommunikationsframeworks in Kap. 5.3.

In diesem Kapitel wird zunächst ein kurzer Überblick über die Sensorplattform gegeben. Im Anschluss wird näher auf die wichtigsten Komponenten der Plattform, d. h. den darauf verwendeten Funkchip und den Prozessor, eingegangen.

3.1 Überblick

Als Hardwareplattform kommt in dieser Arbeit der Imote 2 [33] zum Einsatz. Diese Plattform ist ein von Intel entwickelter und derzeit von Crossbow vermarkteter Sensorknoten. Die Vorder- und Rückseite des Imote 2 ist in Abb. 3.1 abgebildet.

Prozessor: Als Prozessor kommt der von Intel entwickelte PXA 271 [32] zum Einsatz. Die Prozessorlinie, der auch dieser Prozessor angehört, wurde mittlerweile von Marvell übernommen und weiterentwickelt. Als Kernhardware implementiert dieser eine XScale-Architektur, die auf der ARM v5TE- [3] Architektur basiert und zu dieser kompatibel ist. Das Rechenwerk kann in mehreren Schritten von 13 MHz bis 416 MHz getaktet werden.

Neben der Kernhardware sind auch viele periphere Komponenten in den Prozessor integriert. Diese sind vor allem für die Kommunikation über die Prozessorgrenze hinweg von Bedeutung. Dazu gehören z. B. UART (*Universal Asynchronous Receiver/Transmitter*), SPI (*Serial Peripheral Protocol*), USB (*Universal Serial Bus*), I^2C (*Inter-Integrated Circuit*) und Anschlüsse zur freien Verwendung (GPIO – *General Purpose Input/Output*).

Außerdem wurden Arbeitsspeicher (256 KiB SRAM und weitere 32 MiB SDRAM) sowie nicht-flüchtiger Speicher (32 MiB Flashspeicher) im selben Gehäuse integriert.

Der Prozessor wird in Kap. 3.3 näher vorgestellt.

Konnektoren: Über die in der Abb. 3.1 markierten Anschlüsse J3, J4, J5 und J7 sind viele der soeben aufgezählten externen Verbindungen des Prozessors erreichbar. So ist es möglich, steckbare Platinen für den Imote 2 herzustellen, die die Funktionalität des Sensorknotens für den jeweiligen Anwendungsbereich erweitern.

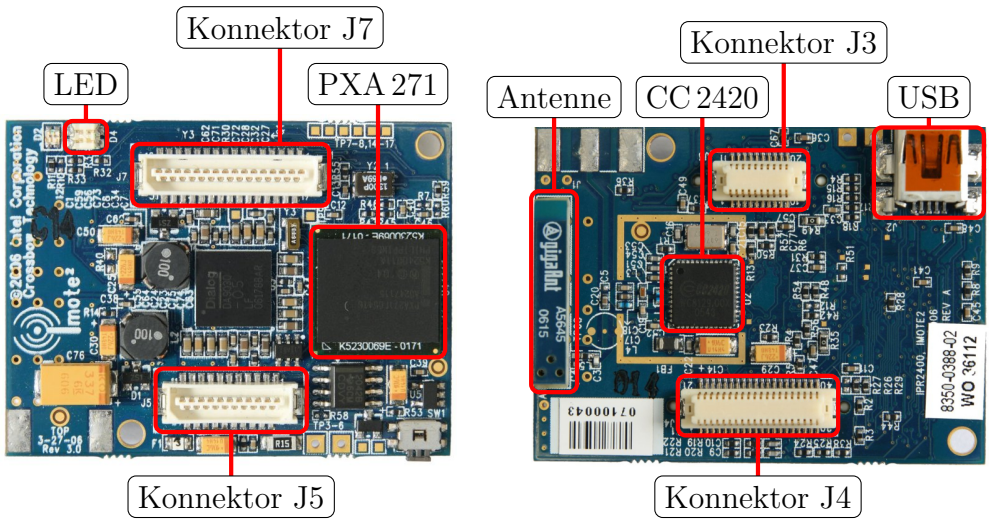


Abbildung 3.1.: Vorder- und Rückseite des Imote 2

Funkhardware: Um mit anderen Knoten drahtlos kommunizieren zu können, wurde der von Chipcon entwickelte Transceiver CC 2420 [41] auf der Imote 2 Plattform verbaut. Die Firma ist mittlerweile eine Tochterfirma von Texas Instruments, welche den Funkchip vertreibt. Da der Funkchip in dieser Arbeit eine zentrale Bedeutung besitzt, wird er im nächsten Abschnitt näher vorgestellt.

3.2 CC 2420-Transceiver

Der CC 2420-Transceiver ist für das Senden und Empfangen IEEE 802.15.4-konformer Rahmen ausgelegt. Dazu sind die physikalische Schicht und einige Teile der MAC-Schicht dieses Standards direkt in der Hardware integriert. So müssen nur die Nutzdaten des Rahmens zum Transceiver übertragen werden, der selbstständig den Rahmenkopf hinzufügt und optional die Behandlung von Prüfsummen übernimmt. Bei Bedarf können noch weitere Teile der MAC-Schicht, wie z. B. das automatische Versenden von ACK-Rahmen, von der Hardware übernommen werden.

Viele dieser Funktionen werden in dieser Arbeit jedoch nicht benötigt, da IEEE 802.15.4-Rahmen, wie in Kap. 2.2.2 beschrieben, im Regelfall nur dazu verwendet werden, Energie auf dem Medium zu erzeugen.

Der CC 2420 bietet eine Vielzahl an Konfigurationsparametern, mit denen das Sende- und Empfangsverhalten beeinflusst werden kann. Einige dieser Parameter beeinflussen auch Funktionalitäten des Transceivers, die gerade für den Einsatz in Black Burst-basierten Protokollen wichtig sind.

Im Folgenden wird der IEEE 802.15.4-Standard vorgestellt. Danach wird die Abbildung von Black Bursts auf IEEE 802.15.4-konforme Rahmen angegeben, wobei hier im Gegensatz zu Kap. 2.2.2 der Schwerpunkt auf der verwendeten Hardware und deren Möglichkeiten und Einschränkungen liegt.

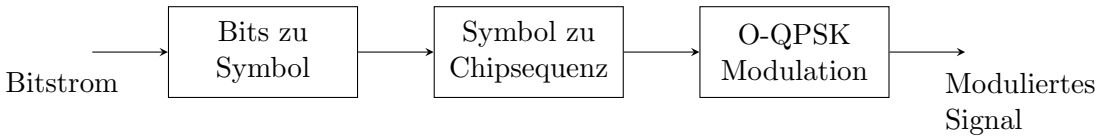


Abbildung 3.2.: Spreizung und Modulation

3.2.1 IEEE 802.15.4

Der IEEE 802.15.4-Standard [25] beschreibt ein Kommunikationsprotokoll für so genannte LR-WPANs (*Low-Rate Wireless Personal Area Networks*). Wie der Name vermuten lässt, handelt es sich hierbei um drahtlose Netzwerke, die auf räumlich begrenzte Kommunikation (wenige Meter), bei denen überdies keine hohen Übertragungsraten gefordert sind, abzielen. Ein besonderes Ziel des Standards ist, dass sich kostengünstige und energieeffiziente Hardware produzieren lässt. Der Standard findet daher besondere Anwendung in Sensornetzwerken.

In dieser Arbeit ist die physikalische Schicht von besonderer Bedeutung, da diese durch die Hardware vorgegeben ist. Der Standard beschreibt mehrere Varianten physikalischer Schichten, die vor allem durch das verwendete Frequenzband und die Modulation charakterisiert sind. Der CC 2420-Transceiver bietet nur die Verwendung des ISM-Bands im 2450 MHz Frequenzbereich. Die nachfolgenden Betrachtungen sollen verschiedene Aspekte dieser Schicht näher erläutern.

3.2.1.1 Frequenzbereich

Der Frequenzbereich erstreckt sich von 2404 MHz bis 2481 MHz. Dieser Bereich ist in 16 Kanäle mit einem Kanalabstand von 5 MHz eingeteilt. Die Kanäle sind von 11 (Trägerfrequenz 2405 MHz) bis 26 (Trägerfrequenz 2480 MHz) durchnummeriert[1].

Die eigentliche Bandbreite eines Kanals beträgt nur 2 MHz, so dass zwischen den Kanälen ein Sicherheitsabstand von 3 MHz besteht. Insbesondere sind die Kanäle unabhängig, da sie sich, im Gegensatz zu beispielsweise IEEE 802.11 [24], nicht überlagern.

Der CC 2420-Transceiver erlaubt jedoch nicht nur die Verwendung der durch den Standard vorgegeben Kanäle. Es kann in Schritten von 1 MHz jede Frequenz zwischen 2405 MHz und 2480 MHz als Trägerfrequenz verwendet werden.

3.2.1.2 Kodierung und Modulation

Die Kodierung und Modulation befasst sich mit der Vorverarbeitung eines Datenstroms. Ziel dieser Verarbeitung ist die Aufbereitung des digitalen Datenstroms in ein digitales Signal, welches weniger anfällig gegen Störungen ist. In Abb. 3.2 ist der Signalfluss in Senderichtung dargestellt.

Sollen Daten in Form eines Bitstroms im hier gewählten Frequenzband übertragen werden, wird zunächst der ankommende Strom in *Symbole* zu je vier Bit gruppiert. Man kann diesen Vorgang auch aus Byte-Sicht betrachten: Es wird je ein Byte in zwei Symbole aufgeteilt. Mit vier Bit lassen sich $2^4 = 16$ Kombinationen erzielen, es gibt folglich 16 verschiedene Symbole.

[1] Die Kanäle 0-10 werden für andere Frequenzbänder genutzt

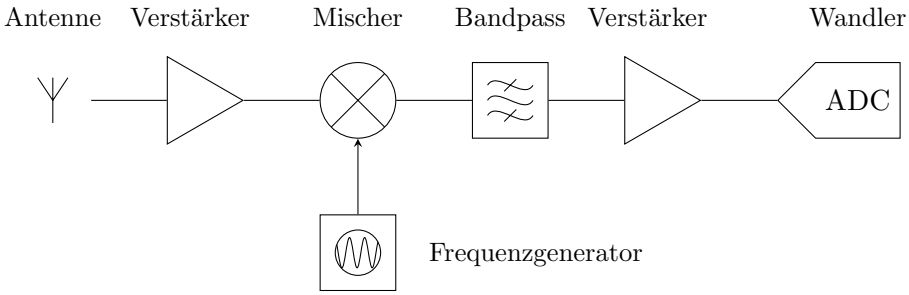


Abbildung 3.3.: Analoge Empfangsstrecke

Als nächstes wird ein Symbol auf eine Sequenz von 32 *Chips* abgebildet. Ein Chip kann die Werte 0 oder 1 annehmen. Die Sequenzen sind für jedes Symbol vom Standard vorgegeben und so gewählt, dass es untereinander möglichst wenig Übereinstimmung gibt, um beim Empfänger die korrekte Rückwandlung einer Chipsequenz in das ursprüngliche Symbol zu vereinfachen. Zusammengefasst werden je vier Bit des einkommenden Bitstroms auf 32 Chips abgebildet. Es handelt sich dabei um ein Spreizverfahren, welches auch DSSS (*Direct Sequence Spread Spectrum*) genannt wird.

Die Chips werden anschließend mit O-QPSK (*Offset-Quadrature Phase Shift Keying*) phasenmoduliert. Bei QPSK gibt es vier Zustände, die zur Modulation der Daten (hier: Chips) genutzt werden können. Dies bedeutet, dass je zwei Chips zeitgleich (bzw. durch den Versatz in O-QPSK halbperiodisch versetzt) versendet werden.

Die Rate, mit der die Chips versendet werden, beträgt $2\,{}^{\mathrm{MChips}}/_{\mathrm{s}}$. Zurückgerechnet bedeutet dies, dass 62500 Symbole pro Sekunde versendet werden. Die Dauer eines Symbols beträgt entsprechend

$$d_{sym} = \frac{1}{62500\,{}^{1}/_{\mathrm{s}}} = 16\,\mu\mathrm{s} \tag{3.1}$$

Da ein Symbol vier Bit kodiert, beträgt die Übertragungsrate auf Bitstrom-Ebene folglich $250\,{}^{\mathrm{kb}}/_{\mathrm{s}}$.

3.2.1.3 Analoge Signalverarbeitung

Das modulierte Signal wird als nächstes in ein analoges Signal umgewandelt. Dieses wird mit einem Tiefpass gefiltert, mit einem Mischer auf ein Oszillatorsignal aufaddiert, verstärkt und über die Antenne abgestrahlt.

Der CC 2420 verwendet hierfür einen *Direktmischer* (*Direct Upconversion*). Das bedeutet, dass die Frequenz des Oszillatorsignals, die mit dem modulierten Signal gemischt wird, genau der Trägerfrequenz entspricht.

Der Empfangspfad ist schematisch in Abb. 3.3 abgebildet. Beim Empfang wird das Signal der Antenne zunächst gefiltert, verstärkt und dann wieder mit einem Oszillatorsignal gemischt, das mit dem Frequenzgenerator gewonnen wird. Der Mischer hat die Eigenschaft, die Summe und die Differenz der Frequenz der Signale auszugeben. Durch eine Bandpassfilterung wird das Frequenzgemisch so gefiltert, dass nur noch die Differenzfrequenz übrig bleibt. Der CC 2420 folgt dabei dem Prinzip des *Superheterodynempfängers* [10]. Dieser hat die Besonderheit, dass die Frequenz des Oszillatorsignals nicht der Frequenz der Trägerwelle entspricht, die eigentlich empfangen werden soll, sondern leicht von dieser abweicht.

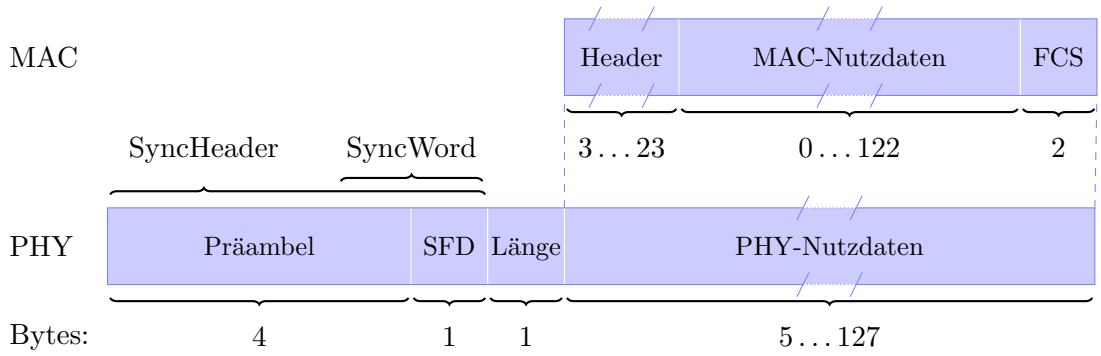


Abbildung 3.4.: Aufbau eines regulären IEEE 802.15.4-Rahmens

Beim CC 2420 ist die Frequenz des Oszillatorsignals um 2 MHz unter der zu empfangenden Trägerfrequenz eingeregelt. Durch das Mischen und Filtern entsteht so ein Signal mit einer Mittenfrequenz von 2 MHz. Diese Frequenz wird auch *Zwischenfrequenz* genannt.

Der Vorteil des Verfahrens liegt darin, dass nach dem Filter unabhängig von der Kanalwahl immer nur ein Signal von 2 MHz anliegt. Die weitere Beschaltung zur Verstärkung und ggf. Filterung des Signals kann daher auf diese Frequenz optimiert werden.

Im CC 2420 steht nur ein Frequenzgenerator zur Verfügung, der für Sende- und Empfangsvorgang verwendet wird. Da sich die Frequenzen der Oszillatorsignale für diese Vorgänge aufgrund obiger Erklärungen trotz gleichbleibender Kanalwahl um 2 MHz unterscheiden, muss bei jedem Wechsel vom Empfangs- in den Sendezustand und umgekehrt der Frequenzgenerator erneut kalibriert werden.

3.2.1.4 Demodulation und Korrelation

Bei der Demodulation wird das empfangene Signal nach seiner Wandlung wieder gemäß der O-QPSK Modulation in Chips überführt. Diese werden in Sequenzen zu 32 Chips gruppiert.

Diese Sequenz wird anschließend gegen die Chipsequenzen aller möglichen Symbole verglichen. Schließlich wird das Symbol gewählt, bei dem die meisten übereinstimmenden Chips gezählt wurden, d. h. das Symbol, dessen Chipsequenz am meisten mit der empfangenen Chipsequenz korreliert.

Aus der Anzahl übereinstimmender Chips wird ein Korrelationswert berechnet. Dieser kann als Indikator der Signalqualität verwendet werden. Ferner ist diese Zahl wichtig für das Erkennen eines Rahmens auf dem Medium: Wird eine Präambel mit SFD erkannt (siehe nächster Abschnitt), so muss auch der Korrelationswert des SFD über einer einstellbaren Schwelle liegen. Dieser Mechanismus verringert die Wahrscheinlichkeit, dass zufälliges Rauschen als Rahmen erkannt und dekodiert wird. Wird der Schwellwert zu hoch eingestellt, besteht jedoch Gefahr, dass auch korrekte Rahmen nicht mehr erkannt werden.

3.2.1.5 Aufbau eines IEEE 802.15.4-Rahmens

Den Nutzdaten eines Rahmens werden vor dem Senden entsprechende Kopfdaten vorangestellt. Diese dienen in erster Linie der Synchronisation des Empfängers. Der Aufbau eines IEEE 802.15.4-Rahmens ist in Abb. 3.4 dargestellt. Die einzelnen Bestandteile werden im Folgenden beschrieben.

Präambel: Die Präambel ist eine feste Sequenz von Symbolen, die vor allem zur Synchronisation zwischen Sender und Empfänger genutzt wird. Erkennt ein Empfänger diese Sequenz, kann er seinen Demodulator justieren und so alle weiteren Symbole aus den Chips wiedergewinnen.

Die Präambel besteht aus acht gleichen Symbolen, die jeweils eine 0 kodieren. Sie hat somit eine Dauer von 128 µs. Beim CC 2420 kann die Länge jedoch abweichend vom Standard zwischen vier und 34 Symbolen in Zweierschritten gewählt werden.

Da die Präambel nur zur Synchronisation genutzt wird, führt eine kürzere oder längere Präambel nicht zwangsweise zu einer ausbleibenden oder fehlerhaften Rahmenerkennung. Grundsätzlich gilt: je länger die Präambel, desto wahrscheinlicher eine erfolgreiche Synchronisation und desto wahrscheinlicher wird der Rahmen korrekt empfangen.

Zusätzlich können beim CC 2420 die letzten beiden Symbole der Präambel frei gewählt werden. Sender, die dies nutzen, senden jedoch nicht mehr IEEE 802.15.4-konforme Rahmen, die auch nur empfangen werden können, wenn die empfangende Seite diese Funktion unterstützt und auf die gleichen Symbole konfiguriert ist.

SFD: Die Präambel wird vom SFD (Start of Frame Delimiter) gefolgt. Die beiden Symbole des SFD markieren das Ende der Synchronisation und den eigentlichen Beginn des Rahmens.

Der Inhalt des SFD ist vom Standard fest definiert. Beim CC 2420 kann jedoch auch dieser umkonfiguriert werden. Rahmen, die mit einem anderen SFD gesendet werden, werden von anderen Empfängern nur erkannt, wenn diese mit dem gleichen SFD konfiguriert wurden.

SyncWord: Die letzten beiden Symbole der Präambel und die beiden Symbole des SFD werden zusammen als *SyncWord* bezeichnet. Dieses kann frei konfiguriert werden.

SyncHeader: Die gesamte Präambel bildet zusammen mit dem SFD-Feld den Synchronisationskopf, der im Folgenden *SyncHeader* genannt wird.

Längenfeld: Das Längenfeld gibt die Länge der im Rahmen enthaltenen PHY-Nutzdaten an. Das höchstwertige Bit des Feldes ist reserviert und muss als 0 übertragen werden. Die maximale Rahmenlänge ist somit auf 127 Byte limitiert.

Beim Empfang eines Rahmens wird die Länge dazu genutzt, das Ende der Übertragung zu erkennen. Fehler in der Dekodierung des Längenfeldes können die Empfangslogik des CC 2420 stören (Kap. 4.4.7).

PHY-Nutzdaten: Die Nutzdaten des Rahmens können von der Anwendung frei gewählt werden. Die Mindestlänge der Nutzdaten beträgt 0 Byte, die maximale Länge 127 Byte. Aus Sicht der MAC-Schicht von IEEE 802.15.4 sind die Längen 0-4 Byte nicht möglich und im Standard auch auf physikalischer Ebene reserviert. Der CC 2420 lässt jedoch Rahmen ab einer Nutzdatenlänge von einem Byte zu.

MAC-Header: Die Kopfdaten der MAC-Ebene enthalten u. A. den Rahmentyp (Beacon, Daten, Quittierung), die Sender- und Empfängeradressen und ggf. verwendete Verschlüsselungsoptionen.

Teilweise können diese Felder auch vom CC 2420 automatisch verwaltet werden. In dieser Arbeit wird davon jedoch kein Gebrauch gemacht.

FCS: Die Prüfsumme (*Frame Check Sequence*) ist Teil der IEEE 802.15.4 MAC-Schicht und belegt zwei Byte des Rahmens. Sie wird als 16-Bit CRC über den MAC-Header und die MAC-Nutzdaten berechnet.

Der CC 2420 kann die Berechnung automatisch vornehmen und die Prüfsumme beim Sendevorgang an die Daten anhängen. Beim Empfang kann auch eine Prüfung vorgenommen werden. Auch wenn in dieser Arbeit von der IEEE 802.15.4 MAC-Schicht größtenteils kein Gebrauch gemacht wird, wird diese Option an manchen Stellen dennoch verwendet, um Rahmen, die keine Black Bursts realisieren, gegen Verfälschungen abzusichern.

3.2.2 Realisierung von Black Bursts

In Kap. 2.2.2 wurde beschrieben, dass Black Bursts mit Hilfe von IEEE 802.15.4-Rahmen realisiert werden können. In diesem Kapitel soll nun eine Abbildung von Black Bursts auf IEEE 802.15.4-Rahmen vorgestellt werden.

3.2.2.1 Dominante Black Bursts

Ein dominanter Black Burst ist ein Energieimpuls einer definierten Länge auf dem Medium. Da Teile der IEEE 802.15.4-Rahmen durch die Hardware gegeben sind, ist deren Länge nach unten und oben beschränkt und nur in diskreten Schritten von zwei Symbolen (also einem Byte) wählbar. Im Hinblick auf vorherige Überlegungen soll nun die kürzeste Medienbelegung hergeleitet werden, die mit der Übertragung eines IEEE 802.15.4 Rahmens unter Verwendung des CC 2420 realisierbar und sinnvoll ist. Je kürzer der IEEE 802.15.4-Rahmen ist, der für einen Black Burst verwendet wird, desto höher ist der Durchsatz, welcher mit einer Black Burst-kodierten Übertragung erreichbar ist (siehe „Konvergenzdauer“ in Kap. 2.3.4). Allerdings muss die Länge mindestens so groß gewählt werden, dass der CCA-Mechanismus noch zuverlässig arbeiten kann.

Wie im vorherigen Kapitel beschrieben, bestehen alle IEEE 802.15.4-Rahmen mindestens aus Präambel, SFD, Längenbyte und Nutzdaten. Die kürzeste Präambel, die mit dem CC 2420-Transceiver möglich ist, besteht aus vier Symbolen. SFD und Längenbyte belegen je zwei weitere Symbole. Die Mindestlänge an PHY-Nutzdaten ist ein Byte, also ebenfalls 2 Symbole. Somit berechnet sich die Länge des kleinstmöglichen Rahmens auf $4+2+2+2 = 10$ Symbole. Die Dauer des Rahmens beträgt folglich

$$\begin{aligned} d_{BB} &= 10 \cdot d_{sym} \\ &\overset{(3.1)}{=} 10 \cdot 16\,\mu s \\ &= 160\,\mu s \end{aligned} \tag{3.2}$$

Der CCA-Mechanismus des CC 2420-Transceivers integriert die Leistung über acht Symbolperioden, also über eine Dauer von $d_{maxCCA} = 128\,\mu s$, bevor er sie mit dem konfigurierten Schwellwert vergleicht. Bei ausreichender Signalstärke wird eine Übertragung der Länge d_{BB} über den CCA-Mechanismus wahrgenommen, da das Integrationsfenster kleiner als die Rahmenlänge ist.

3.2.2.2 SOF-Black Burst

In Kap. 2.3.3 wurde der Aufbau eines Tickrahmens im Master-basierten BBS-Verfahren vorgestellt. Das erste Bit des Rahmens ist das SOF-Bit, das den Beginn einer Runde markiert.

Das SOF-Bit wird durch einen verlängerten Black Burst realisiert, um die Erkennung eines Tickrahmens zu vereinfachen und Synchronisationsfehler zu reduzieren.

In der Implementierung des Kommunikationsframeworks (Kap. 5.3) wurde der SOF-Black Burst mit einem ähnlich aufgebauten IEEE 802.15.4-Rahmen realisiert wie ein regulärer dominanter Black Burst. In Formel 2.12 wurde die Forderung gegeben, dass die nominale Dauer eines SOF-Black Bursts größer sein muss als die maximal erkennbare Belegungsdauer eines dominanten Black Bursts. Es muss also gelten:

$$d_{BB\_SOF} \overset{(2.12)}{>} d_{maxBaseTickOffset} + d_{BB} + d_{maxCCA} + d_{RX}$$

In Kap. 2.3.4.3 wurde $d_{maxBaseTickOffset}$ als $d_{maxBaseTickOffset,local}$ angenommen und ist somit $2 \cdot d_{maxCCA}$. Die maximale CCA-Verzögerung ist für den CC 2420 als $d_{maxCCA} = 128\,\mu s$ angegeben. Die Verarbeitungsverzögerung d_{RX} wird als weniger als 32 µs angenommen. Somit erhält man

$$\begin{aligned} d_{BB\_SOF} &= 2 \cdot 128\,\mu s + 160\,\mu s + 128\,\mu s + 32\,\mu s \\ &= 576\,\mu s \end{aligned} \tag{3.3}$$

Diese Dauer entspricht gerade 36 Symbolen. Abzüglich der Länge des PHY-Headers bleiben somit 14 Byte PHY-Nutzdaten.

Der SOF-Black Burst der ersten Runde wird gemäß der Optimierung in Kap. 2.3.3 als normaler IEEE 802.15.4-Rahmen versendet und soll auch als solcher empfangen werden. Daher wird für diesen Rahmen die Länge der Präambel wieder standardkonform auf acht Symbole erhöht. Außerdem wird der Rahmen durch die MAC-Prüfsumme gegen Verfälschungen geschützt, die zwei Byte der PHY-Nutzdaten belegt. Um die Dauer eines solchen Rahmens gleich zu halten, wurden daher vier Bytes vor PHY-Nutzdaten entfernt und zwei durch die Prüfsumme ersetzt. Dadurch hat auch der SOF-Rahmen der ersten Runde eine Dauer von 576 µs.

3.3 PXA 271-Prozessor

In Kap. 3.1 wurden die Hauptmerkmale des beim Imote 2 verwendeten Prozessors bereits kurz beschrieben. In den nachfolgenden Abschnitten wird nun dessen Aufbau näher betrachtet.

Es stehen dabei vor allem Quellen variabler Verzögerungen bei der Programmausführung und Unterbrechungsbehandlung im Vordergrund. Des Weiteren werden Ideen vermittelt, mit denen sich die Verzögerungen reduzieren lassen, und weitere Optimierungsansätze vorgestellt.

3.3.1 Kernhardware

Die Kernhardware des Prozessors setzt sich aus Hauptprozessor, Speicherverwaltung, Pufferspeicher und wenigen weiteren Komponenten zusammen [27]. Die für diese Arbeit wichtigen Komponenten und ihre Zusammenschaltung sind schematisch in Abb. 3.5 gezeigt. Ihre Funktion wird in den folgenden Unterkapiteln erläutert.

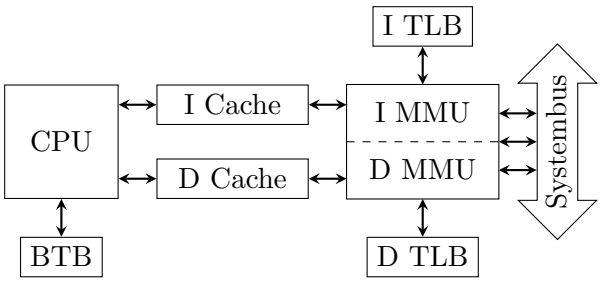


Abbildung 3.5.: Kernhardware des PXA 271

3.3.1.1 Hauptprozessor

Als Hauptprozessor (CPU – *Central Processing Unit*) kommt ein ARMv5 [3]-kompatibler Prozessor zum Einsatz. ARMv5 ist eine RISC-Architektur (*Reduced Instruction Set Computer*) mit fester 32-Bit Befehls- und Datenwortbreite. Des Weiteren unterstützt die CPU einen weiteren Befehlssatz (Thumb Instruction Set). Dieser arbeitet auf 16-Bit Befehlen, ist folglich aber auch weniger mächtig als der normale Instruktionssatz.

Registerspeicher und Schattierung: Weiter verfügt die Architektur über 16 Register. Wenige dieser Register, wie z. B. der Programmzähler und das Rücksprungregister, haben für den Prozessor eine vorgegebene Aufgabe. Alle anderen stehen theoretisch zur freien Verfügung, werden jedoch per Konvention an bestimmte Aufgaben gebunden.

Einige der Register haben eine an den Prozessormodus (z. B. Unterbrechungsbehandlung, Ausnahmebehandlung einer Speicherverletzung, normale Programmausführung) gebundene Schattenkopie. So hat beispielsweise der Modus, der bei einer Ausnahmebehandlung gewählt wird, einen eigenen Stackzeiger und es müssen weniger Register gesichert werden als bei Architekturen, die keine Schattenregister besitzen. In Kap. 3.3.1.5 wird beschrieben, wie sich dieses Merkmal im Hinblick auf diese Arbeit als Optimierung nutzen lässt.

Pipelining und Stalls: Die CPU ist aus einer achtstufigen Pipeline aufgebaut, die an mehreren Stellen verzweigt ist. Der genaue Aufbau ist in [27] gegeben. Somit können zu jedem Zeitpunkt bis zu acht Instruktionen von der CPU bearbeitet werden. Wenn erforderliche Daten bei Abarbeitung einer Instruktion noch nicht verfügbar sind, wird die Ausführung des Befehls (und somit potentiell die Verarbeitung nachfolgender Instruktionen) verzögert (*Stall*).

Ein Stall tritt auf, wenn die Daten der Instruktion vom Ergebnis einer vorherigen Instruktion abhängen, dieses Ergebnis jedoch noch nicht zurückgeschrieben ist. Das Auftreten dieser Stalls wird durch die Umgehung der Pipeline (*Bypassing*) verringert. Weiter können sie durch Umsortierung der Instruktionen reduziert werden. Die CPU bietet jedoch keine automatische Umsortierung, sie muss also manuell bzw. durch den Compiler vorgenommen werden.

Eine weitere Quelle für Stalls ist das Laden bzw. Speichern von Daten, die im Arbeitsspeicher vorgehalten werden. Da an dieser Stelle jedoch die Pipeline verzweigt ist, können andere, davon unabhängige Instruktionen weiter ausgeführt werden. Folglich werden Instruktionen auch nicht immer in Reihenfolge ihrer Position im Code abgeschlossen (*Out of Order Completion*).

Zusammen mit den in den nächsten Abschnitten vorgestellten Optimierungen macht dies die Berechnung der Dauer, die für die Ausführung eines Codes nötig ist, sehr komplex.

Daten- und Instruktionspfade: Der Hauptprozessor verfügt über getrennte Pfade für Daten und Befehle. Diese Wege kommen jedoch am Systembus wieder zusammen, so dass sich Programmcode und Daten einen gemeinsamen Speicherbus teilen. Von diesem Aspekt her betrachtet, handelt es sich also um eine Von-Neumann-Architektur.

3.3.1.2 Caches

Als Folge der getrennten Pfade für Code und Daten verfügt die Kernhardware über zwei getrennte Caches, von denen einer (I-Cache) am Codepfad und der andere (D-Cache) am Datenpfad angeschlossen ist. Der I-Cache hält somit nur Instruktionen vor, der D-Cache nur Anwendungsdaten aus Lade- bzw. Speicher-Operationen.

Größe: Ein Datum wird nie einzeln im Cache hinterlegt, sondern es wird immer eine ganze Cachezeile zu je acht Wörtern (entsprechend 32 Byte) gefüllt. Insgesamt gibt es in jedem Cache 1024 Zeilen. Jeder Cache hat somit eine Gesamtgröße von 32 KiB.

Organisation: Die Caches sind 32-fach satzassoziativ, d. h. jeder Satz enthält 32 Cachezeilen. Die Speicheradresse ist derart organisiert, dass konsekutive Cachezeilen im jeweils nächsten Satz abgelegt werden. Wird beispielsweise ein Block von 96 Byte komplett im Cache gepuffert, wird je eine Cachezeile aus drei konsekutiven Sätzen belegt.

Ein Nachteil dieser Organisation ist, dass alle Daten bzw. Instruktion, deren Adresse auf Kib-Grenzen ausgerichtet sind, im ersten Satz abgelegt werden. Daher ist dieser Satz höher frequentiert als andere.

Ersetzungsstrategie: Soll eine neue Zeile in einem Satz abgelegt werden, muss, wenn alle Zeilen des Satzes bereits belegt sind, eine Zeile verdrängt werden.

Bei der Architektur kommt hierfür ein Round-Robin-Verfahren zum Einsatz. Im Bezug auf Ersetzungsstrategien [40] wird diese Strategie normalerweise FIFO (*First In, First Out*) genannt, jedoch wird in Folgenden die Terminologie der ARM-Architektur beibehalten. Das Verfahren verdrängt jeweils den ältesten Eintrag. So werden zwar alle Einträge über die Zeit gesehen gleich oft verdrängt, jedoch werden dadurch auch Einträge ersetzt, die besonders häufig gebraucht werden. Sinnvoller wäre hier also eine Ersetzungsstrategie wie z. B. (Pseudo-)LRU [35] (*Least Recently Used*), die Zugriffe auf die Einträge miteinbezieht.

Schreibstrategie: Beim Schreiben von Daten können diese direkt in den Hintergrundspeicher übertragen werden (*write-through*) oder die Änderungen vorerst im Cache vorgehalten werden (*write-back*). Die letztere Variante führt zwar zu einer verbesserten Leistung bei (häufigen) Schreibzugriffen, jedoch erhöht sich die Dauer einer Verdrängung, da die Daten bei einer Verdrängung zunächst zurückgeschrieben werden müssen.

Die Wahl der Schreibstrategie ist mit der Speicherverwaltung konfigurierbar und kann mit Granularität der Seitengröße gewählt werden. Die Schreibstrategie findet nur beim D-Cache Anwendung, da Daten des I-Cache unveränderlich sind.

Kohärenz: Bei Caches handelt es sich immer um eine Kopie der Daten des Hintergrundspeichers. Eine Veränderung der Daten im Hintergrundspeicher verlangt, dass die Änderungen auch vom jeweiligen Cache nachvollzogen werden müssen. Bei Änderungen der Daten durch den Prozessor ist dies bis auf wenige Spezialfälle unproblematisch, da diese Zugriffe die Caches durchlaufen.

Auf den Hintergrundspeicher kann jedoch auch unabhängig vom Prozessor durch den DMA-Controller (Kap. 3.3.3.1) zugegriffen werden. Diese Zugriffe sind unabhängig von den Caches und werden von diesen nicht wahrgenommen. So kann es zu Inkonsistenzen zwischen den Daten im Hintergrundspeicher und Daten im Cache kommen. In den Programmen (bzw. den Treibern) muss an den entsprechenden Stellen daher dafür gesorgt werden, dass die Zeilen des Caches invalidiert und ggf. zurückgeschrieben werden.

Locking: Für besonders häufig gebrauchte Zeilen bietet die Architektur einen Mechanismus zum Sperren von Einträgen. Gesperrte Zeilen werden, nachdem sie initial in den Cache geladen wurden, von der Verdrängung ausgenommen und sind somit dauerhaft verfügbar. Insgesamt können 28 der 32 Zeilen in jedem Satz gesperrt werden.

Für den I-Cache sind etwa der Einsprungsvektor, der insgesamt acht Sprungbefehle für alle Ausnahmebehandlungen enthält (und somit bei jeder Unterbrechung geladen wird) sowie die Unterbrechungsroutine selbst gute Kandidaten für diesen Mechanismus. Im D-Cache sind die Sprungvektoren zu Unterbrechungsroutinen der Subsysteme häufig frequentiert und sollten nicht verdrängt werden.

Man sollte jedoch den Gebrauch dieses Merkmals sehr gezielt einsetzen: Pro gesperrter Zeile verbleiben weniger Zeilen für andere Daten. Man sollte daher auch darauf achten, die gesperrten Zeilen gleichmäßig über alle Sätze zu verteilen. Dies erfordert jedoch eine gezielte Ausrichtung von Speicheradressen. In der Regel ist dies mit aufwändigen Analysen und großem Aufwand beim Binden der Anwendung verbunden.

Außerdem sollte darauf geachtet werden, dass die gesperrten Zeilen möglichst ausgenutzt sind und wenig Verschnitt vorhanden ist. Wenn beispielsweise acht Daten zu je einem Byte im Cache gesperrt sein sollen, sollten diese nicht in acht verschiedenen Zeilen gepuffert werden, sondern zusammenhängend in einer Zeile gesperrt werden. Dies erfordert einen gewissen Aufwand und Disziplin beim Programmieren der Systemsoftware. Die gewählte Programmiersprache muss daher auch eine Notation für die manuelle oder teilautomatisierte Ausrichtung von Datenobjekten bereitstellen.

Optimierungsziel: Der Sperrmechanismus soll dazu genutzt werden, häufig frequentierten und zeitkritischen Code und dessen Daten dauerhaft im Cache vorzuhalten. Dies soll letztlich dazu führen, die Verzögerungen und deren Schwankungen beim Ausführen dieser zeitkritischen Routinen zu minimieren. Messungen zum Cache und eine Evaluierung des Sperrmechanismus sind in Kap. 4.5.1 und Kap. 4.5.2.3 angegeben.

3.3.1.3 Speicherverwaltungseinheit

Die Adressen des Instruktions- und Datenpfades werden zunächst von der Speicherverwaltungseinheit (MMU – *Memory Management Unit*) von virtuellen Adressen auf physikalische Adressen abgebildet.

Die hierfür benötigte Abbildungstabelle befindet sich selbst im Hauptspeicher. Für den Abbildungsvorgang muss die MMU also selbst auf Speicher zugreifen, je nach Granularität sogar mehrere Male.

Granularität: Die MMU arbeitet mit ein- und zweistufigen Abbildungsverfahren. Die erste Stufe dividiert den gesamten virtuellen Adressraum von 4 GiB in 4096 Blöcke zu je 1 MiB. Diese Stufe kann dazu verwendet werden, virtuelle Adressblöcke mit einer Granularität von 1 MiB auf physikalische abzubilden.

Ist diese Granularität zu grob, kann ein 1 MiB-Block in einer weiteren Abbildung in Subblöcke zu wahlweise 64 KiB, 4 KiB oder 1 KiB unterteilt werden. Die Architektur bietet diesbezüglich viele Besonderheiten, die vor allem die Anzahl der Abbildungen senken sollen. Der Verwaltungsaufwand der Systemsoftware wird dadurch allerdings erhöht.

TLB: Um den Abbildungsvorgang zu beschleunigen, werden Übersetzungspuffer (TLB – *Translation Lookaside Buffer*) eingesetzt, in denen eine begrenzte Anzahl an Abbildungen vorgehalten werden. Wenn eine benötigte Abbildung nicht vorgehalten wird, muss diese aus der Abbildungstabelle im Hauptspeicher geladen werden. Eine neue Abbildung wird dann in den TLB eingefügt.

Die Kernhardware des Prozessors besitzt für Instruktions- und Datenpfad unabhängige TLBs mit einer jeweiligen Größe von 32 Einträgen. Die TLBs sind vollassoziativ ausgeführt, so dass jeder Eintrag auf jede Position des TLBs abgebildet werden kann.

Ersetzungsstrategie: Wenn alle Einträge des TLBs belegt sind, muss ein vorhandener Eintrag verdrängt werden, sobald ein neuer Eintrag hinzugefügt werden soll. Die Auswahl dieser Zeile geschieht wie bei den Caches mit dem Round-Robin-Verfahren. Aufgrund der Beschränkungen dieser Strategie ergibt sich auch hier das Problem, dass Einträge verdrängt werden, die besonders häufig verwendet werden.

TLB Locking: Wie bei den Caches bietet die Architektur auch hier einen Sperrmechanismus, um besonders häufig gebrauchte Einträge von der Verdrängung auszuschließen. Insgesamt können 31 der 32 Einträge gesperrt werden.

Dies sollte jedoch nur an sehr wenigen Stellen eingesetzt werden, da mit jedem gesperrten Eintrag weniger Positionen für andere Abbildungen zur Verfügung stehen.

Optimierungsziel: Die Adressabbildung der MMU kann einen aufwändigen Prozess darstellen, je nachdem, ob der entsprechende Eintrag im TLB vorgehalten wird oder nicht. Nach diesem Kriterium und der allgemeinen Speicherbusauslastung richtet sich die Dauer des Abbildungsvorgangs. Dadurch ist der Vorgang der Adressabbildung und somit jeder Speicherzugriff von einer variablen und i. Allg. unvorhersagbaren Verzögerung betroffen.

Mit Hilfe des Sperrmechanismus sollen diejenigen Einträge, die zeitkritischen Code und dessen Daten abbilden, dauerhaft im TLB gesperrt werden. Hierbei sollte jedoch beachtet werden, dass eine Sperrung von Einträgen für Instruktionen bzw. Daten, die bereits im Cache gesperrt sind, nur bedingt sinnvoll ist. Da die Caches mit virtuellen (d. h. noch nicht übersetzten) Adressen arbeiten und die gesperrten Zeilen von dort nicht verdrängt werden, muss auch ihre Adresse bei einem Zugriff nicht übersetzt werden.

3.3.1.4 Sprungvorhersage

In der Architektur kommt eine Sprungvorhersage (*Branch Prediction*) zum Einsatz, welche vor allem das Ausführen von Schleifen optimiert. Dazu werden zu einem bedingten Sprungbefehl die letzten zwei Pfade (Sprung ausgeführt bzw. nicht ausgeführt) gespeichert. Aus dieser Information wird eine Vorhersage über den nächsten Pfad der gleichen Sprunginstruktion getroffen, so dass die Instruktionen des so angenommenen Zielpfades bereits in den I-Cache und das Steuerwerk der CPU vorgeladen werden können (*Prefetching*).

Der BTB (*Branch Target Buffer*) ist im Gegensatz zu den Caches und dem TLB nicht assoziativ, sondern direkt ausgeführt. Einer Sprunginstruktion ist also abhängig von ihrer Adresse ein bestimmter Eintrag im BTB zugewiesen. Insgesamt fasst der BTB 128 Einträge.

Auch der Einsatz des BTB kann Ausführungszeiten beeinflussen. Allerdings werden bei einer Schleife, die mehrmals ausgeführt wird, maximal drei falsche Vorhersagen (zwei beim Eintritt und eine beim Austritt) möglich. Ist die Anzahl der Schleifendurchläufe dagegen hoch, wirken sich die falschen Vorhersagen kaum auf die Gesamtausführungsdauer aus. Aus diesen Gründen ist die Beeinflussung des BTB im Hinblick auf Ausführungszeiten eher konstant.

Die einzig mögliche Optimierung wäre das Deaktivieren der Sprungvorhersage, um so konstantere Ausführungszeiten zu erhalten. Da die Schwankungen jedoch durch maximal zwei falsche Vorhersagen überschaubar sind, ist eine solche Optimierung nicht nötig.

Zusätzlich können an Programmverzweigungen Hinweise an den Compiler gegeben werden, die den wahrscheinlicheren Ausführungspfad markieren. So können bei der Codegenerierung die Instruktionen schon derart geordnet werden, dass auch im Falle einer fehlenden Vorhersage (z. B. bei Erstausführung des Codes) keine Nachteile entstehen.

3.3.1.5 Prozessormodi und Unterbrechungen

Die CPU bietet mehrere Prozessormodi, die vor allem für die verschiedenen Ausnahme- und Unterbrechungsbehandlungen vorgesehen sind. Im Folgenden soll der Fokus auf den Unterbrechungsbehandlungen liegen.

Generell unterstützt die Architektur zwei Arten von Unterbrechungen: IRQ (*Interrupt Request*) und FIQ (*Fast Interrupt Request*). Jede Interruptquelle kann frei konfigurierbar entweder als IRQ oder FIQ verarbeitet werden. Die Relation der beiden Modi zueinander ist, dass auftretende FIQs die Behandlung eines IRQs unterbrechen dürfen. Es handelt sich also um eine hardwaregestützte Schachtelung und Priorisierung von Unterbrechungen.

Da eine Verwaltung von geschachtelten Unterbrechungen hohe Anforderungen an die Programmierdisziplin stellt und sehr fehleranfällig ist, werden in der Implementierung dieser Arbeit entweder nur IRQs oder nur FIQs verwendet. Bei erster Betrachtung gibt es bei exklusiver Verwendung eines Modus keinen Unterschied zwischen den beiden Unterbrechungsarten. Die Architektur wurde jedoch derart ausgelegt, dass FIQs durch Hardwareunterstützungen eine schnellere Behandlung der Unterbrechung zulassen.

Registerschattierung: Ein Teil der Prozessorregister ist mehrfach ausgeführt. Je nach Prozessormodus steht eine bestimmte Ausführung bereit. So ist beispielsweise das Rücksprungregister *lr* in jedem Prozessormodus verfügbar, d. h. das lr-Register im Ausführungsmodus ist de facto ein anderes lr-Register als im FIQ-Modus. Der Vorteil besteht darin, dass diese Register bei Eintritt einer Ausnahme nicht gesichert und am Ende der Behandlung

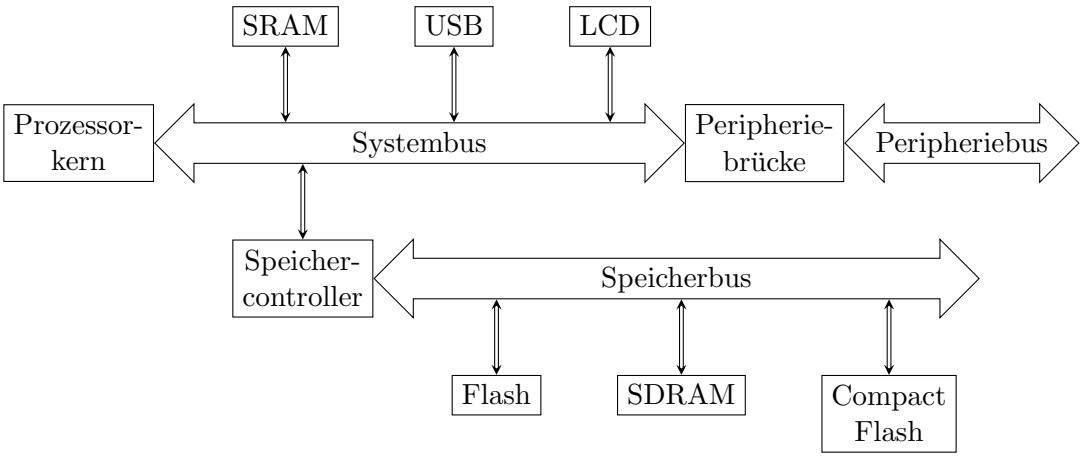


Abbildung 3.6.: System- und Speicherbus des PXA 271

nicht wiederhergestellt werden müssen. Der FIQ-Modus bietet hierbei die größte Menge an schattierten Registern, es muss bei Eintritt einer Unterbrechung also die geringste Anzahl an Registern gesichert werden.

Umgekehrt kann man diesen Mechanismus auch dazu nutzen, bestimmte wichtige Werte in die Schattenregister vorzuladen. Auf diese Weise sind diese Werte direkt bei Eintritt einer Unterbrechung in einem vordefinierten Register vorhanden und müssen nicht gesondert geladen werden. Ein Beispiel für ein solches Datum ist die Adresse des Sprungvektors zu den Subroutinen der Unterbrechungsbehandlung.

Bei Black Burst-basierten Protokollen ist auch das Erfassen von Zeitstempeln von besonderer Bedeutung. Diese werden bei Eintritt in die Unterbrechungsroutine durch Auslesen eines Registers im peripheren Zeitgeber gewonnen. Die Adresse dieses peripheren Registers muss ebenfalls so früh wie möglich vorhanden sein und sollte daher ebenfalls in einem schattierten Prozessorregister vorgehalten werden.

3.3.2 Systembus

Der Systembus bildet den ersten Teil der Schnittstelle des Prozessors mit den weiteren integrierten Komponenten des PXA 271. In Abbildung 3.6 ist eine schematische Abbildung des System- und Speicherbusses gegeben.

Der Systembus ist direkt mit dem Instruktions- und Datenpfad des Prozessorkerns verbunden. Der Kern ist allerdings nicht der alleinige Busmaster. Die Peripheriebrücke integriert einen DMA-Controller (*Direct Memory Access*), der ebenfalls als Master auf den Bus zugreifen kann.

An den Systembus sind Komponenten angebunden, die eine hohe Geschwindigkeit beim Datentransfer benötigen. Der Systembus ist in den meisten Fällen mit der gleichen Frequenz wie der Prozessorkern getaktet. Eine Ausnahme bildet dabei nur die Kernfrequenz von 416 MHz, bei der der Systembus lediglich mit 208 MHz getaktet wird.

3.3.2.1 Komponenten

Der Systembus verbindet datenintensive Komponenten mit dem Prozessorkern. Der Prozessor, der konkret beim Imote 2 verbaut wurde, enthält dabei folgende Bestandteile:

- 256 KiB flüchtigen Arbeitsspeicher in Form von SRAM (Static Random Access Memory). Die Struktur dieses Speichers erlaubt besonders schnelle Zugriffe.

  Für Datenmengen typischer Anwendungen, wie sie in unseren Anwendungen verwendet werden, sind die 256 KiB ausreichend, selbst wenn Heap- und Stackspeicher im SRAM gelagert werden.

  Bei größeren Anwendungen sollten vor allem besonders wichtige Daten, wie z. B. die Abbildungstabelle der MMU, in diesem Speicher untergebracht werden.

- Ein USB-Controller, der beim Imote 2 als USB-Anschluss in Mini-B Bauweise ausgeführt ist. Dieser wird in unserer Verwendung der Hardware vor allem zum Aufspielen neuer Anwendungen verwendet und hat in dieser Arbeit keine weitere Bedeutung.

- Ein LCD-Controller, über den ein LC-Display (*Liquid Crystal Display*) angesteuert werden kann. Dies ist eher für die Verwendung des Prozessors in einem Handheld-Gerät gedacht und findet auf dem Imote 2 keine Verwendung.

Des Weiteren gibt es Übergänge zu weiteren Bussen. Auf diese wird in den folgenden Abschnitten eingegangen.

3.3.2.2 Speicherbus

Über den Speichercontroller ist der Systembus mit dem Speicherbus verbunden. Der Speichercontroller verwaltet die verschiedenen Arten von Speicher, die an den Speicherbus angebunden werden können. Außerdem kann er ausstehende Speicherzugriffe neu ordnen (*Memory Access Reordering*), um die Architektur des jeweils angesprochenen Speichers besser auszunutzen.

Als weiteres Merkmal der Architektur befindet sich im Speichercontroller ein Schreibpuffer. Mit diesem können mehrere ausstehende Lese- und Schreibzugriffe auf die gleiche Adresse zusammengelegt (*Read / Write Coalescing*) werden, so dass die Anzahl von Zugriffen gesenkt wird.

All diese hardwareseitigen Optimierungen sind transparent für den Programmierer. Sie können jedoch variable Verzögerungen verursachen, so dass letztlich Aussagen über Ausführungszeiten von Code sehr komplex werden.

Am Speicherbus, wie er im Prozessor des Imote 2 zur Anwendung kommt, sind folgende Komponenten angeschlossen:

- 32 MiB nicht-flüchtiger Speicher in Form von Flashspeicher. Dieser wird vor allem für die dauerhafte Speicherung einer Anwendung des Sensorknotes verwendet.

- 32 MiB flüchtiger Speicher in Form von SDRAM (Synchronous Dynamic Random Access Memory). Dieser kann zusätzlich als Arbeitsspeicher verwendet werden, wenn die 256 KiB des SRAM nicht ausreichend sind. Ferner können Anwendungen, die nur temporär ausgeführt werden sollen, in diesen Speicher abgelegt werden.

- Schnittstelle zu Compact-Flash-Speicher. Diese ist jedoch beim Imote 2 nicht nach außen geführt und kann daher auch nicht verwendet werden.

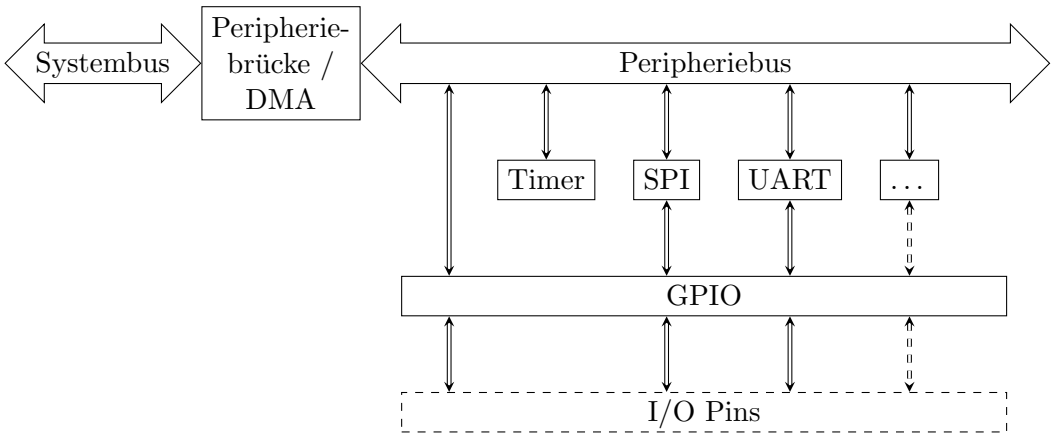


Abbildung 3.7.: Peripheriebus des PXA 271

3.3.3 Peripherie

Der Systembus bindet über die Peripheriebrücke den Peripheriebus an. Darüber sind weitere Systemkomponenten angebunden, die vor allem der externen Interaktion mit dem Prozessor bzw. der ausgeführten Anwendung dienen. Der Aufbau ist schematisch in Abb. 3.7 dargestellt.

Die Komponenten werden durch Lesen und Schreiben fest definierter Speicherbereiche bedient. Es handelt sich also um MMIO (*Memory Mapped I/O*).

Im Gegensatz zum Systembus, der in den meisten Fällen mit der gleichen Taktfrequenz wie die Kernhardware betrieben wird, wird der Peripheriebus immer mit 13 MHz betrieben. Ferner können auch viele der angesprochenen hardwareseitigen Optimierungen wie Coalescing und Reordering nicht durchgeführt werden, da das Lesen und Beschreiben von Peripherieregistern nicht, wie bei Speicherzugriffen auf herkömmlichen Speicher, nur im einfachen Ablegen bzw. Auslesen von Daten besteht. Viel mehr sind damit Seiteneffekte verbunden, die bestimmte Aktionen in der Peripherie auslösen können. Aus diesem Grund ist die speicherbasierte Kommunikation mit der Peripherie potentiell mit einer erhöhten Anzahl an Bus-Stalls verbunden, wodurch ggf. die Programmausführung kurzzeitig angehalten werden muss. Dies hat zur Folge, dass Zugriffe auf Komponenten des Peripheriebusses mit vergleichsweise sehr hohen Latenzen einhergehen (Kap. 4.5.1).

Im Folgenden werden einige wichtige periphere Komponenten des PXA 271-Prozessors vorgestellt.

3.3.3.1 DMA-Controller

Eine häufige Aufgabe während der Programmausführung ist das Transferieren von Daten zwischen einer Komponente des Peripheriebusses und dem Arbeitsspeicher, der direkt am Systembus oder am Speicherbus angehängt ist.

Da dies aufgrund des relativ langsamen Peripheriebusses eine zeitaufwändige Operation ist, beinhaltet die Brücke zwischen Peripherie- und Systembus einen DMA-Controller (*Direct Memory Access*). Dieser tauscht als zweiter Busmaster Daten zwischen Komponenten des Peripheriebusses und Komponenten des Systembusses aus, ohne dass dabei der Prozessor involviert ist.

Nachteilig ist dabei jedoch die asynchrone Handhabung des Datentransfers und die mögliche Inkonsistenz der Caches (Kap. 3.3.1.2).

3.3.3.2 Timer

Die Timer-Komponente stellt neun unabhängige Timer zur Verfügung. Diese können verwendet werden, um Zeitstempel auszulesen und daraus Zeitspannen zwischen Ereignissen zu ermitteln. Außerdem können für jeden Timer ein oder sogar mehrere Zielwerte definiert werden, bei deren Erreichen eine Unterbrechung ausgelöst wird.

In der Implementierung, die im Zuge dieser Arbeit durchgeführt wurde, wird ein Timer mit einer Auflösung von 1 µs für die Definition einer lokalen Systemzeit verwendet. Ein weiterer Timer mit gleicher Auflösung wird verwendet, um die zeitliche Koordination der Black Burst-basierten Protokolle zu realisieren. So wird beispielsweise zu einem errechneten Zeitpunkt eine Unterbrechung ausgelöst, die schließlich im Versenden eines Black Bursts resultiert.

3.3.3.3 UART

Die UART-Schnittstelle (*Universal Asynchronous Receiver Transmitter*) ist eine serielle Schnittstelle, die insbesondere dazu dient, Ausgaben des Systems an ein anderes System weiterzugeben. So können etwa textuelle Meldungen der Anwendung an einen PC weitergeleitet werden.

Die UART-Schnittstelle dient daher vor allem der Anzeige des momentanen Zustands. Dies stellt besonders im Fehlerfall eine große Unterstützung bei der Fehlersuche dar.

Umgekehrt können über diese Schnittstelle auch Befehle von der Anwendung entgegen genommen und verarbeitet werden.

3.3.3.4 SPI

Die SPI-Schnittstelle (*Serial Peripheral Interface*) ist ein Bussystem, mit dem sich weitere externe Komponenten an die Hardware anbinden lassen. Der Bus ist Master-gesteuert und hat bzgl. der Steuerleitung eine Sterntopologie.

Aus physikalischer Sicht besteht der Bus aus mindestens vier Leitungen:

- `CS` (Chip Select): Dieses Signal wählt einen bestimmten Slave an. Folglich existiert für jeden Slave ein exklusives Chip Select Signal. Es kann zu jedem Zeitpunkt maximal ein Slave angewählt werden.

  Die Logik des `CS`-Signals ist gedreht: Ein Slave ist bei Anlegen einer logischen 0 angewählt. Daher wird das Signal oft auch `CSn` genannt.

- `SCK` oder `SCLK` (Synchronous Clock): Dieses Taktsignal wird durch den Master vorgegeben. Bei einer Taktflanke (wahlweise steigend oder fallend) wird das jeweils momentan anliegende Bit der Datenleitungen ausgelesen und bei der jeweils anderen Taktflanke das nächste angelegt.

- `MOSI`, `MO` oder `SI` (Master Out/Slave In): Diese Datenleitung wird vom Master getrieben. Die Bits des Datenstroms werden hierüber mit der Frequenz des Taktsignals seriell zum angewählten Slave übertragen.

- `MISO`, `MI` oder `SO` (Master In/Slave Out): Diese Datenleitung wird vom angewählten Slave getrieben. Die Bits des Datenstroms werden hierüber zum Master übertragen.

Da weder die elektrischen Eigenschaften noch das genaue Protokoll standardisiert sind und SPI daher nur als Quasi-Standard gilt, gibt es SPI in mehreren Varianten, die sich z. B. in der Reihenfolge der Bits, der Polarität des `SCK`-Signals oder der Phase der Datensignale, unterscheiden. Der Master und der angewählte Slave müssen zum erfolgreichen Datenaustausch dieselbe Variante verwenden.

Der SPI-Bus ist in dieser Arbeit von zentraler Bedeutung, da über diesen der CC 2420 angebunden ist. Sämtliche Daten und Kommandos an den Transceiver werden über den Bus ausgetauscht. Auch der physikalische Aufbau des Busses ist von Bedeutung.

3.3.3.5 GPIO

Viele der Eingabe-/Ausgabepins des PXA 271 sind mit den integrierten Peripheriekomponenten des Prozessors verbunden. Wird eine Komponente nicht verwendet, stehen ihre Pins zur freien Verfügung. Da die Pins mehrfach belegt sind, muss die konkrete Belegung in der GPIO-Komponente (*General Purpose Input/Output*) konfiguriert werden. Diese Komponente übernimmt auch das direkte Ansteuern der Pins, wenn sie keiner anderen Komponente zugeordnet sind. So können sämtliche Pins des PXA 271, die auf dem Imote 2 über die Konnektoren erreichbar sind, direkt aus der ausgeführten Anwendung heraus gesteuert werden.

In dieser Arbeit werden über diesen Mechanismus interne Ereignisse nach außen signalisiert oder auch von außen ein Ereignis in der Anwendung ausgelöst.

3.4 Interaktion der Komponenten

Die Interaktion zwischen dem Prozessor PXA 271 und dem Transceiver CC 2420 spielt eine zentrale Rolle in dieser Arbeit. Viele Beobachtungen wurden erst durch das Abgreifen der ausgetauschten Daten und das zeitliche Erfassen von Ereignissen möglich.

Abbildung 3.8 zeigt die Verdrahtung der beiden Komponenten auf dem Imote 2. Wie bereits in Kap. 3.3.3.4 beschrieben, werden Daten und Befehle über einen SPI-Bus ausgetauscht. Darüber hinaus sind vier binäre Steuersignale des CC 2420 per GPIO an den PXA 271 angebunden. Die Bedeutung dieser Signale wird im Folgenden vorgestellt. Da sie vom Modus des Transceivers abhängt, wird hier nach Sende- bzw. Empfangsmodus unterschieden.

Empfangsmodus: Im Empfangsmodus zeigen die Steuersignale den momentanen Zustand bezüglich eines möglichen Rahmenempfangs an. Im Detail haben die Signale folgende Bedeutung:

- Das `FIFO`-Signal zeigt den Zustand des Empfangspuffers an, der im Transceiver integriert ist. Sobald dieser mit mindestens einem Byte gefüllt ist, wird dies über eine logische 1 angezeigt, bis der Puffer durch entsprechende Kommandos ggf. ausgelesen und geleert wird.

  Da jedoch immer ganze Rahmen statt einzelner Bytes ausgelesen werden, ist dieses Signal für die Implementierung von nur geringer Bedeutung.

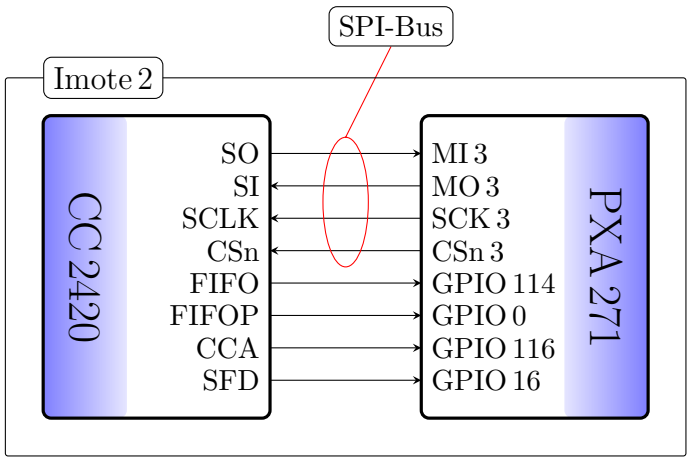


Abbildung 3.8.: Anbindung des CC 2420 an den PXA 271

- Das `FIFOP`-Signal zeigt ebenfalls den Zustand des Empfangspuffers an. Allerdings muss der Füllstand eine einstellbare Schwelle erreichen, bevor eine logische 1 geschaltet wird.

  Unabhängig davon, ob der Schwellwert erreicht wurde, zeigt dieses Signal auch das Ende eines Rahmens an. Wird der Schwellwert größer als die maximale Rahmenlänge eingestellt, zeigt das Signal folglich nur noch das Ende eines Rahmens an. Bei diesem Ereignis kann der gesamte Rahmen aus dem Empfangspuffer ausgelesen werden.

  Nach Auslesen des Puffers nimmt das Signal wieder einen logischen 0-Pegel an.

- Das `SFD`-Signal führt eine logische 1, nachdem das SFD eines Rahmens empfangen wurde. Anhand des Zeitstempels dieses Ereignisses kann also ein Rahmenempfang einem Zeitpunkt zugeordnet werden.

  Das Signal geht wieder auf den 0-Pegel zurück, nachdem der Rahmenempfang vollständig abgeschlossen ist. Durch die Zeitstempel der Ereignisse kann auch die Dauer der Medienbelegung durch den Rahmen zurückgerechnet werden. Der Puffer sollte jedoch erst ausgelesen werden, wenn das `FIFOP`-Signal die Bereitschaft des Puffers anzeigt.

  Im implementierten Kommunikationsframework (Kap. 5.3) wird dieses Signal verwendet, um Black Bursts zu erkennen, die als reguläre IEEE 802.15.4-Rahmen versendet werden. Diese Optimierung betrifft das SOF-Bit des ersten Tickrahmens und wurde in Kap. 2.2.4 und Kap. 2.3.3 beschrieben.

- Das `CCA`-Signal wird für den CCA-Mechanismus verwendet. Wird das Medium als belegt erkannt, wird dies über eine logische 0 (*non-clear*) angezeigt. Ein freies Medium entspricht einer logischen 1.

Sendemodus: Im Sendemodus zeigen die Steuersignale den Fortschritt der Übertragung an. Die Signale werden im Detail wie folgt verwendet:

- Die `FIFO` und `FIFOP`-Signale sind weiterhin an den Empfangspuffer gebunden und haben für den Sendemodus keine weitere Bedeutung.

- Das `SFD`-Signal verhält sich ähnlich wie im Empfangsmodus, wird nun jedoch auf den Sendevorgang bezogen. Es führt eine logische 1, nachdem das SFD gesendet wurde und geht nach abgeschlossenem Sendevorgang wieder in den inaktiven Zustand zurück.
- Das `CCA`-Signal ist während des gesamten Sendevorgangs ungültig. Es führt zwar einen definierten Pegel, der jedoch erst wieder im Empfangsmodus sinnvoll interpretiert werden kann.

3.5 Bewertung der Hardware

Der Imote 2 ist, im Vergleich zu anderen Sensorknoten, eine sehr leistungsstarke Sensorplattform. Dies ist vor allem auf den PXA 271-Prozessor zurückzuführen, der bei dieser Plattform eingesetzt wird. Auch wenn die Architektur des Prozessors sehr viele Optimierungen bereitstellt, die die Ausführungszeiten insgesamt stark verringern, steigt andererseits die Komplexität der Hardware und auch die Schwankungen der Ausführungszeiten. Dennoch kann durch geeignete Maßnahmen das Verhalten des Prozessors so beeinflusst werden, dass die Schwankungen reduziert werden.

Auch der CC 2420-Funkchip, der beim Imote 2 eingesetzt wird, hat im Kontext Black Burst-basierter Kommunikation einige Nachteile. Beispielsweise trägt die große CCA-Verzögerung massiv zur Ungenauigkeit und zur Konvergenzdauer von BBS bei. Ein Funkchip mit einer schnelleren Erkennung der Medienbelegung und einer geringeren Umschaltdauer wäre in dieser Hinsicht vorteilhaft. Andererseits bietet der CC 2420 auch einige Vorteile gegenüber anderen IEEE 802.15.4-kompatiblen Transceivern. Durch das direkte Herausführen des CCA-Signals über den CCA-Pin ist es möglich, bei der Änderung des erkannten Belegungszustands direkt eine Unterbrechung beim Prozessor auszulösen. Dies ist bei anderen Transceivern (z. B. dem AT86RF230 [5]) nicht der Fall. Bei der genannten Hardware müsste das interne CCA-Register periodisch ausgelesen werden, wodurch weitere Verzögerungen entstehen. Der CC 2420-Transceiver hat außerdem eine weite Verbreitung, da er neben dem Imote 2 auch auf weiteren Sensorplattformen eingesetzt wird.

4. Kapitel

Experimente zur verwendeten Hardware

In Kap. 2 wurden die Grundlagen Black Burst-basierter Protokolle behandelt. Insbesondere wurde das Synchronisationsverfahren BBS vorgestellt. Dabei wurden einige mathematische Forderungen an z. B. die Bitdauer formuliert, die nur von den Laufzeiten der eingesetzten Hardware, insbesondere also dem CC 2420-Transceiver und dem PXA 271-Prozessor, abhängen.

Einige dieser Laufzeiten sind durch die Datenblätter der Hardware vorgegeben. Ein Teilziel dieses Kapitels ist die Überprüfung dieser Zeitspannen. Insbesondere soll dabei auch ermittelt werden, welche dieser Zeitspannen konstant und welche variabel sind. Insbesondere sind gerade die variablen Verzögerungen kritisch, da diese im Gegensatz zu den konstanten Verzögerungen nicht aus Zeitpunkten und -spannen (z. B. dem Beginn oder der Dauer einer Medienbelegung) herausgerechnet werden können. In den Worst-Case-Analysen müssen die variablen Anteile immer durch ihren minimalen bzw. maximalen Wert angenommen werden. Um das Ausmaß variabler Anteile zu bestimmen wurden in jedem Experimente mehrere Messdurchläufe durchgeführt.

Für weitere Zeitspannen (wie z. B. die Verzögerungen bei einer Unterbrechungsbehandlung), die ebenfalls in Kap. 2 definiert wurden, existieren bestenfalls nur Abschätzungen. Auch diese sollen im folgenden Kapitel genauer bestimmt werden.

In Kap. 3.1 wurde die verwendete Hardware vorgestellt und ihr funktionales Verhalten skizziert, welches ebenfalls den entsprechenden Datenblättern entnommen wurde. Ein weiteres Ziel dieses Kapitels ist die Prüfung des beschriebenen funktionalen Verhaltens. Des Weiteren sollen auch Fehl- und unerwartetes Verhalten der Hardware gefunden und dokumentiert werden. Ferner werden Optimierungen hinsichtlich Ausführungszeiten evaluiert, um variable Verzögerungen zu verringern und so letztlich die Black Burst-basierten Protokolle optimieren zu können.

Zunächst werden nun Konventionen eingeführt, die vor allem der Messauswertung dienen. Danach werden die verwendeten Messgeräte und ihre Eigenschaften vorgestellt. Anschließend folgen verschiedene Messreihen und Optimierungen bezüglich der Funkhardware und des Prozessors.

4.1 Bezeichnerkonventionen

Durch die folgenden Konventionen werden die Schreibweise von Befehls- und Signalnamen sowie von Zeitpunkten und Zeitspannen vereinheitlicht. Ziel ist die eindeutige Zuordnung von mathematischen Bezeichnern zu Messereignissen und daraus abgeleiteten Größen.

4.1.1 Bezeichner

Signalnamen und Befehle: Signalnamen und Befehle werden mit einer festen Zeichenbreite angegeben. Zum Beispiel wird das CCA-Signal als `CCA` und der STXON-Befehl des CC 2420 als `STXON` dargestellt.

Signalflanken: Ereignisse, die auf binären Signalen basieren, finden immer an Signalflanken statt. Als Flanke wird der Übergang von einem Signalpegel zum jeweils anderen bezeichnet.

Eine steigende Flanke (Wechsel von logisch 0 auf logisch 1) wird mit dem Zeichen ⌋⌈ dargestellt. Die fallende Flanke (Wechsel von logisch 1 auf logisch 0) wird entsprechend mit dem Zeichen ⌉⌊ dargestellt. Diese Zeichen sind dem jeweiligen Signalnamen vorangestellt.

Zum Beispiel wird eine fallende Flanke des `CCA`-Signals mit ⌉⌊`CCA` und eine steigende Flanke mit ⌋⌈`CCA` bezeichnet.

Zeitpunkte: Zeitpunkte werden mit dem Formelzeichen t dargestellt, welches im Index einen Bezeichner enthält, der das Ereignis beschreibt.

So bezeichnet z. B. $t_{\lfloor\lceil \mathtt{CCA}}$ den Zeitpunkt einer steigenden Flanke des `CCA`-Signals.

Vorgegebene Zeitspannen: Eine vorgegebene Zeitspanne ist eine Zeitspanne, die z. B. einem Datenblatt entnommen ist. Sie wird mit dem Formelzeichen d dargestellt. Der Index enthält einen Bezeichner zur näheren Beschreibung der Zeitspanne.

Beispielsweise wird die Länge des SyncHeaders eines CC 2420-Rahmens mit $d_{syncHdr}$ bezeichnet.

Gemessene Zeitspannen: Eine gemessene Zeitspanne wird mit dem Formelzeichen Δ dargestellt. Auch hier enthält der Index einen Bezeichner.

Die gemessene Umschaltverzögerung vom Empfangs- in den Sendezustand des CC 2420 wird beispielsweise als Δ_{rxtx} bezeichnet.

4.1.2 Operationen auf Zeitspannen

Da das Auswerten von Zeitspannen ein zentraler Bestandteil dieses Kapitels ist, wird ein Operator eingeführt und einige Eigenschaften dieses Operators gezeigt, die das Rechnen mit Zeitspannen vereinfachen.

Definition: Das Bestimmen von Zeitspannen wird grundsätzlich auf das Berechnen von Differenzen der Zeitpunkte verschiedener Ereignisse zurückgeführt.

Seien a und b beliebe Ereignisse und t_a bzw. t_b die zugehörigen Zeitpunkte. Intuitiv soll der Operator $\Delta_{\mathtt{a}\to\mathtt{b}}$ die Zeitspanne zwischen a und b angeben. Er ist daher wie folgt definiert:

$$\Delta_{\mathtt{a}\to\mathtt{b}} := t_b - t_a \tag{4.1}$$

Der Operator $d_{a\to b}$ ist analog zu $\Delta_{\mathtt{a}\to\mathtt{b}}$ definiert, würde aber entsprechend der Konvention verwendet werden, wenn die Zeitspanne auf tatsächlichen statt gemessenen Zeitpunkten a und b basiert und so eine nominale Dauer angibt.

Transitivität: Seien a, b und c beliebige Ereignisse, dann gilt:

$$\Delta_{\mathtt{a}\to\mathtt{b}} + \Delta_{\mathtt{b}\to\mathtt{c}} = \Delta_{\mathtt{a}\to\mathtt{c}} \tag{4.2}$$

Negation: Seien a und b beliebige Ereignisse, dann gilt:

$$\Delta_{\mathtt{a}\to\mathtt{b}} = -\Delta_{\mathtt{b}\to\mathtt{a}} \tag{4.3}$$

Einfügen und Entfernen von Zwischenzeitpunkten: Seien a, b Ereignisse, dann gilt für ein beliebiges Ereignis c:

$$\Delta_{\mathtt{a}\to\mathtt{b}} = \Delta_{\mathtt{a}\to\mathtt{c}} - \Delta_{\mathtt{b}\to\mathtt{c}} \tag{4.4}$$
$$\Delta_{\mathtt{a}\to\mathtt{b}} = \Delta_{\mathtt{c}\to\mathtt{b}} - \Delta_{\mathtt{c}\to\mathtt{a}} \tag{4.5}$$

Austauschen von Zwischenzeitpunkten: Seien a und b Ereignisse, dann gelten für beliebige Ereignisse c und d:

$$\Delta_{\mathtt{a}\to\mathtt{c}} - \Delta_{\mathtt{b}\to\mathtt{c}} = \Delta_{\mathtt{a}\to\mathtt{d}} - \Delta_{\mathtt{b}\to\mathtt{d}} \tag{4.6}$$
$$\Delta_{\mathtt{c}\to\mathtt{b}} - \Delta_{\mathtt{c}\to\mathtt{a}} = \Delta_{\mathtt{d}\to\mathtt{b}} - \Delta_{\mathtt{d}\to\mathtt{a}} \tag{4.7}$$

4.2 Überblick über die Messhardware

In diesem Kapitel werden die verschiedenen Geräte vorgestellt, mit denen die jeweiligen Messreihen durchgeführt wurden. Dazu wird ihre generelle Funktionsweise erklärt und die Daten beschrieben, die damit erhoben werden.

Das wichtigste Messgerät war der Logic Analyzer Sigma 2, der zur Bestimmung sämtlicher Zeitspannen eingesetzt wurde. Bei Messungen, in denen der tatsächliche Belegungszustand des Mediums bestimmt werden musste, kam zusätzlich das Software-Defined Radio USRP 2 zum Einsatz. Zur Koordination der Tests wurde das FPGA-Evaluationsboard Basys 2 eingesetzt, welches die erforderlichen Stimuli erzeugte, um einen Testlauf in der zu vermessenden Hardware auszulösen.

4.2.1 Logic Analyzer Sigma 2

Der *Logic Analyzer* Sigma 2 [4] (siehe Abb. 4.1) der Firma ASIX ist ein Messgerät, das den zeitlichen Verlauf mehrerer binärer Signale aufzeichnet. Dabei können die zeitlichen Bezüge von Ereignissen eines oder mehrerer Signale gemessen werden. Ein Ereignis eines binären Signals wird durch einen Flankenwechsel ausgelöst.

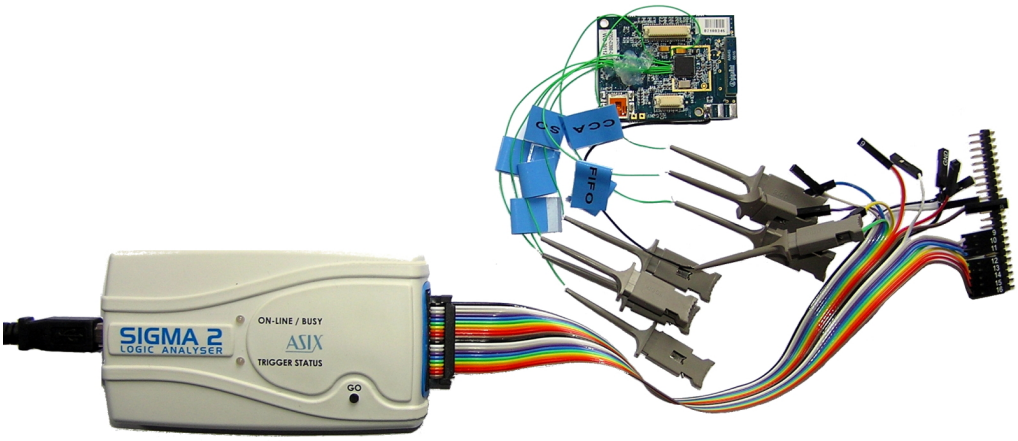


Abbildung 4.1.: ASIX Sigma 2 mit Verbindungen zum Imote 2

Physikalische Erfassung der zu beobachtenden Signale: Die zu untersuchenden Signale werden direkt auf physikalischer Ebene abgegriffen und mit dem Eingang des Logic Analyzers verbunden. Das Sigma 2 kann bis zu 16 Signale gleichzeitig aufzeichnen und besitzt folglich auch 16 Eingänge.

Das Sigma 2 ist kein eigenständiges Gerät, sondern wird es über einen USB-Anschluss mit einem PC verbunden. Die Datenrate dieser Verbindung reicht jedoch nicht zum Übertragen der erfassten Daten in Echtzeit aus. Das Sigma 2 speichert daher die Daten während der (zeitlich begrenzten) Messung intern und überträgt sie nach dem Messvorgang an den PC, wo sie von einem Programm entgegengenommen werden.

Aufzeichnungsvorgang: Die Aufzeichnung der Signale geschieht durch periodische Abtastung. Die *Abtastrate* oder auch *Abtastfrequenz* bestimmt die Häufigkeit, mit dem der Zustand aller angeschlossenen Signale bestimmt wird. Der Kehrwert wird *Abtastperiode* oder *Abtastintervall* genannt. Die Wahl der Abtastrate beeinflusst die Genauigkeit der Messung. Dieser Zusammenhang wird in Kap. 4.3.1 erläutert.

Das Sigma 2 unterstützt Abtastraten von 10 MHz bis 200 MHz, die Abtastperiode ist entsprechend 5 ns bis 100 ns. Die gewählte Frequenz hat allerdings Einschränkungen im Hinblick auf die Anzahl nutzbarer Kanäle. Bei bis zu 50 MHz sind alle 16, bei 100 MHz noch acht und bei 200 MHz nur vier Kanäle nutzbar. Die Abtastraten wurden in den Experimenten daher immer maximal in Abhängigkeit der Anzahl der zu untersuchenden Signale gewählt.

Speichertiefe: Die Speichertiefe gibt die maximale Anzahl der abgetasteten Werte an. Zusammen mit der Anzahl der Kanäle und der Abtastrate bestimmt sie auch die maximale Dauer einer Aufzeichnung.

Das Sigma 2 enthält 256 MBit Speicher. Bei einer Abtastfrequenz von 200 MHz auf vier Kanälen ist eine theoretische Aufzeichnungsdauer von 320 ms möglich. Durch eine im Sigma 2 integrierte Lauflängenkodierung werden jedoch gleichbleibende konsekutive Werte zusammengefasst, so dass die errechnete Aufzeichnungsdauer als Untergrenze gesehen werden kann. Je nach Signalform sind folglich sehr viel längere Aufzeichnungen möglich.

Abbildung 4.2.: Das zur Überwachung des Drahtlosmediums verwendete USRP 2 von Ettus Research

Auswertung aufgezeichneter Daten: Die Daten werden nach jedem Messvorgang zu einem PC übertragen. Eine von ASIX mitgelieferte Software nimmt die Daten entgegen und kann diese durch ein interaktives Signal-Zeit-Diagramm grafisch aufbereiten. Damit können auch zeitliche Bezüge zwischen Ereignissen gemessen werden.

Da jedoch jedes Experiment mehrere Male (teilweise 1000 Mal) wiederholt wurde, muss eine automatische Wiederholbarkeit gegeben sein. Hierzu wurde die Software mit einem Plug-In erweitert, das ausgehend von einem Steuerprogramm eine Messung einleitet oder beendet. Ferner ist es für die automatische Speicherung der erfassten Messdaten verantwortlich.

4.2.2 Software Defined Radio USRP 2

Das USRP 2 [19] (*Universal Software Radio Peripheral*) ist ein von Ettus Research entwickeltes *Software Defined Radio* (SDR). Abbildung 4.2 zeigt eine Ablichtung des Gerätes. Das Ziel von SDRs ist die Verschiebung digitaler Signalverarbeitung eines Funkmoduls von fest verdrahteter Hardware in Richtung rekonfigurierbarer Hardware oder sogar Software. Das bedeutet, dass nur ein kleiner Teil des Signalwegs (zumindest der analoge Teile) in fester Hardware verarbeitet wird. Weitere Komponenten digitaler Signalverarbeitung, wie z. B. Filterung, Verstärkung, Umtastung, etc., werden dann in rekonfigurierbarer Hardware im gleichen Gerät durchgeführt oder über eine geeignete Schnittstelle zu einem PC weitergeleitet und dort per Software nachgebildet.

Dies unterscheidet sich von konventioneller Funkhardware, da bei dieser angestrebt wird, die gesamte Signalverarbeitung und ggf. auch Aspekte des jeweiligen Kommunikationsprotokolls in einem Chip durchzuführen. Dadurch wird die softwareseitige Verarbeitung und Verwaltung sowie der Aufwand externer Hardwarebeschaltung minimiert.

Der Vorteil von SDRs liegt in der universellen Einsetzbarkeit. Da die Verarbeitung des Signals nicht fest vorgegeben ist, kann diese je nach Bedarf festgelegt werden. So können Umtastverfahren, Filterung, etc. eines beliebigen Funkstandards implementiert werden. Der

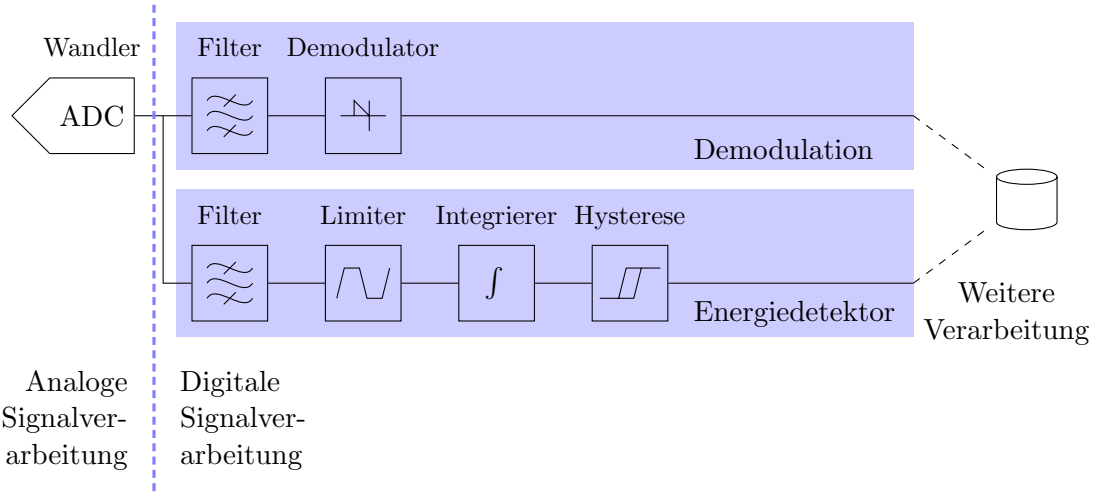


Abbildung 4.3.: Verschiedene Anwendungsgebiete eines SDRs

Nachteil liegt darin, dass der Daten- und Verarbeitungsaufwand in der Software steigt. Die Daten des SDRs bestehen nun aus den Rohdaten des Signals und nicht mehr aus einem fertig aufbereiteten Bitstrom aus Kopf- und Nutzdaten, wie dies bei konventioneller Hardware normalerweise der Fall ist.

In Abb. 4.3 ist neben der Demodulation eines Signals eine weitere Anwendungsmöglichkeit gezeigt, die sich für das Überwachen des Belegungszustands eignet. Dazu kann ein einfacher Energiedetektor implementiert werden. Dieser integriert die empfangene Leistung über einen festgelegten Zeitraum und vergleicht diesen Wert gegen einen Schwellwert. Die Funktion gleicht dem CCA-Mechanismus des CC 2420 (Kap. 2.2.2), jedoch kann beim USRP 2 die Größe des Integrationsfensters kleiner gewählt werden, da die Abtastrate sehr viel höher ist als beim CC 2420. Somit steigt die Geschwindigkeit der Belegungserkennung gegenüber dem CCA-Mechanismus des CC 2420-Funkchips. In den nachfolgend beschriebenen Experimenten war die Überwachung des Mediums die Hauptaufgabe des USRP 2. Außerdem wurden auch die Leistungen empfangener Signale bzw. des Rauschens in Relation gesetzt, um daraus eventuelle Abschwächungen des Funksignals zu ermitteln.

Das USRP 2 kann je nach Frequenzbereich Signale empfangen und auch senden. Teilweise sind sogar die Signalpfade getrennt und unabhängig voneinander ansprechbar, so dass ein gleichzeitiges Senden und Empfangen (*voll-duplex*) möglich ist. Für diese Arbeit war jedoch nur die Empfangsrichtung relevant, die im Folgenden näher betrachtet wird.

Anschlüsse des USRP 2: Das USRP 2 wird durch steckbare Tochterplatinen (*Daughter Boards*) erweitert, die passend zu einem gewählten Frequenzbereich die Auf- bzw. Abwärtsmischung des Nutzsignals mit einer Trägerwelle übernehmen. Auf der Ausgangsseite ist eine Gigabit-Ethernet Schnittstelle, die die Rohdaten zum PC überträgt.

Abtastung und Quantisierung des Signals: Nachdem das Nutzsignal vom Trägersignal getrennt ist, wird dieses in ein digitales Signal umgewandelt. Dabei werden sowohl die Zeit diskretisiert als auch die Werte quantisiert. Der Wertebereich umfasst dabei 16 Bit. Demnach gibt 65536 verschiedene Werte, die durch eine Darstellung im Zweierkomplement

auf einen Wertebereich von -32768 bis +32767 abgebildet werden. Die Einheit dieser Werte ist proportional zur Watt-Skala, jedoch durch unbekannte Größen (z. B. Verstärkungsfaktor der Antenne, Dämpfung durch Anschlüsse und Kabel, etc.) nicht absolut zu dieser Skala bestimmbar. Dennoch können Aussagen über das Verhältnis zweier Signalleistungen getroffen werden. Das logarithmische Verhältnis kann in dB angegeben werden.

Die Abtastrate kann bis zu 100 MHz betragen, die jedoch i. Allg. auf der Ausgangsseite eine höhere Datenrate erfordert, als dies mit Gigabit-Ethernet möglich ist. In den Experimenten wurde daher mit einer Rate von 20 MHz abgetastet.

Auswertung der aufgezeichneten Daten: Nachdem die abgetasteten Daten zum PC übertragen sind, können sie dort verarbeitet werden. Neben der Speicherung der Rohdaten ist es je nach Anwendung und ausreichender Rechenleistung sogar möglich, diese in Echtzeit zu verarbeiten.

Für diese Arbeit war die Speicherung der Rohdaten ausreichend. Hierzu wurde ein Programm entwickelt, welches die Daten des USRP 2 entgegennimmt und speichert. Dieses Programm ist auch für das Einleiten und Beenden einer Messung verantwortlich. Dadurch wurden die Wiederholungen eines Experiments durch ein Steuerprogramm automatisiert. Die eigentliche Auswertung fand nach der Messung statt.

Modifikation des USRP 2: Die rekonfigurierbare Hardware des USRP 2 wurde dahingehend verändert, dass externe binäre Signale in den Datenstrom der Rohdaten eingearbeitet werden können. Als Beispiele für solche Signale seien das `SFD`- oder `CCA`-Signal der Funkhardware genannt. Auf diese Weise können Ereignisse dieser Signale mit den Rohdaten verknüpft und zusammen ausgewertet werden. Aufgrund dieser Modifikation ist auch eine Synchronisation der aufgenommenen Messdaten des Sigma 2 und des USRP 2 möglich.

4.2.3 FPGA Board Basys 2

Das Basys 2 [16] der Firma Digilent ist ein FPGA (*Field Programmable Gate Array*) Evaluationsboard. In Abb. 4.4 ist das Basys 2 mit Verbindungen zum Imote 2 gezeigt. FPGAs sind integrierte Schaltkreise, die aus mehreren Tausenden bis Millionen einzelner Zellen (*Gates*) bestehen, deren Funktion und Zusammenschaltung durch eine änderbare Konfiguration definiert ist. Die Funktion der Gesamtschaltung kann vom Benutzer durch Verwenden einer Hardwarebeschreibungssprache – wie z. B. Verilog – nahezu beliebig gewählt werden. Die Beschreibung wird durch einen Synthesevorgang zu einem Bitstrom übersetzt, der beim Einschalten des FPGAs in dessen Konfigurationsspeicher geladen wird und dessen Verhalten bestimmt. Darüber hinaus verfügt der FPGA über mehrere Ein- und Ausgänge, die in der Schaltung frei verwendet werden können (GPIO), um z. B. von außen ein Ereignis zu triggern oder einen internen Zustand anzuzeigen.

Das Basys 2 enthält neben dem FPGA einige weitere Hardware wie z. B. Schalter, Taster, vier 7-Segment-Anzeigen, einen Chip und Anschluss für externe USB-Kommunikation, einen PS/2-Anschluss zum Betreiben einer Tastatur oder Maus, einen VGA-Anschluss zum Ansteuern eines Monitors und 16 direkt nach außen geführte GPIO-Pins zur freien Verwendung.

In den Experimenten, die in dieser Arbeit durchgeführt wurden, wurde das Basys 2 zur Erzeugung der Stimuli für das zu testende System verwendet. Die übergeordnete

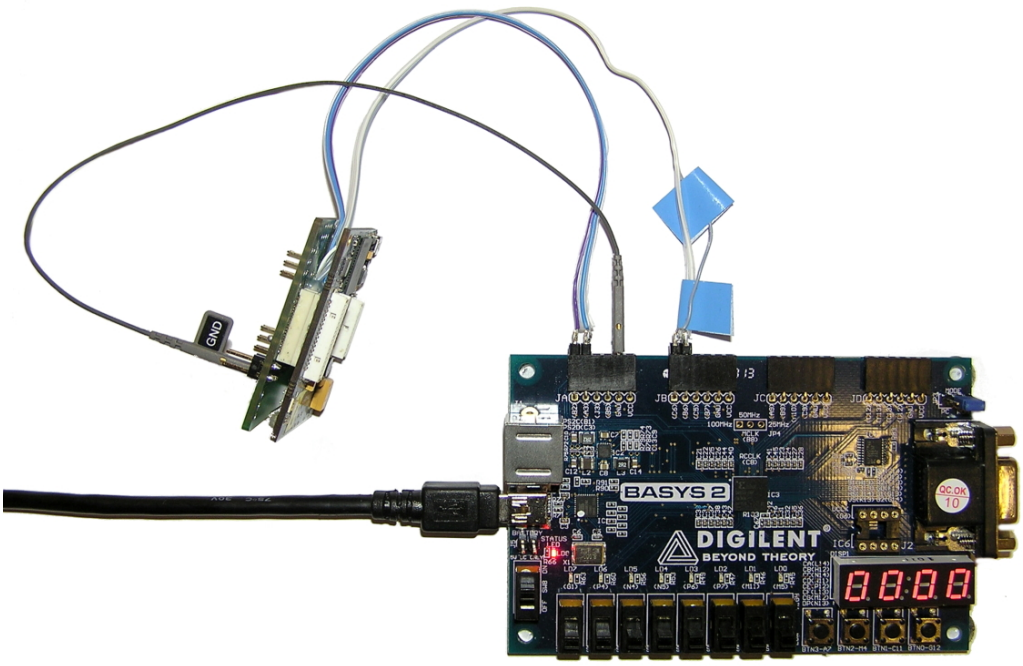


Abbildung 4.4.: Digilent Basys 2 mit angeschlossenem Imote 2

Koordination wird durch ein Programm, das auf einem PC ausgeführt wird, gesteuert. Um jedoch der zu testenden Hardware ein Ereignis – wie z. B. den Start des Messvorgangs – mitzuteilen, muss dieses Ereignis vom PC möglichst generisch und synchron an alle betroffenen Hardwarekomponenten weitergegeben werden. Dazu wird der USB-Anschluss des Basys 2 verwendet. Über diesen werden mit Hilfe einer mitgelieferten Programmbibliothek von Digilent Daten an das Basys 2 weitergegeben, welches dann in Abhängigkeit dieser Daten Flanken auf den GPIO-Anschlüssen der Platine erzeugt. Diese Anschlüsse wurden mit den GPIO-Anschlüssen des Imote 2 verbunden, so dass Aktionen der Imote 2-Applikation über das Basys 2 getriggert werden.

4.2.4 Messaufbau und Messdurchführung

In diesem Kapitel wird ein allgemeiner Überblick über den Messvorgang und -aufbau gegeben. Insbesondere wird das Zusammenspiel der verwendeten Messgeräte und die Koordination einer Messung beschrieben.

Vielen Messungen liegt ein ähnlicher Messaufbau zugrunde. Ein schematischer Überblick, der viele gemeinsame Aspekte der verschiedenen Messaufbauten vereint, ist in Abb. 4.5 gegeben. In fast allen Experimenten wurde das Basys 2 als Auslöser (*Trigger*) eines Messvorgangs und das Sigma 2 als Logic Analyzer eingesetzt. Für manche Experimente kam noch das USRP 2 zur Überwachung des Belegungszustands des Mediums und ggf. der übertragenen Energie hinzu. Das Eintreten interner Ereignisse im Programm, das auf dem Prozessor des Imote 2 ausgeführt wurde, wurde über weitere GPIO-Pins des Prozessors nach außen und über eine Signalleitung zum Logic Analyzer geführt. Zusätzlich wurden der SPI-Bus und die Steuersignale des CC 2420 (`CCA`, `SFD`, `FIFO`, `FIFOP`, siehe Kap. 3.4) abgegriffen und zum

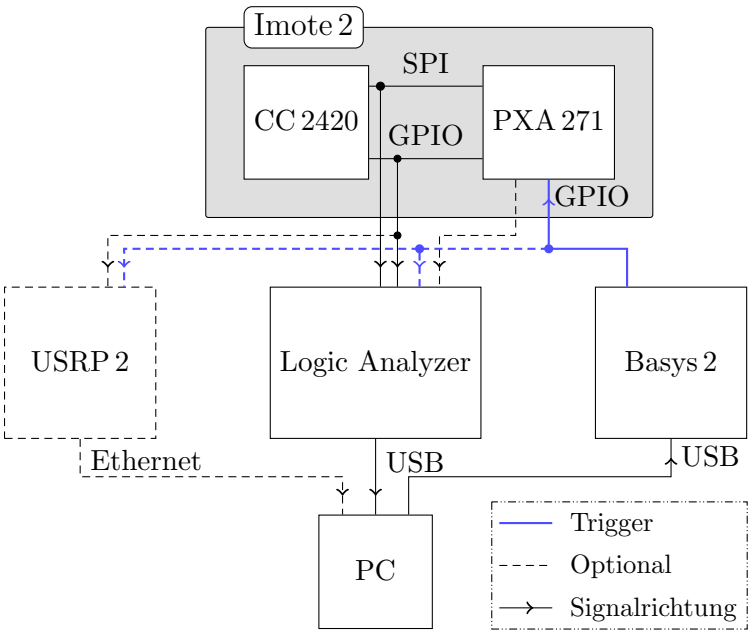


Abbildung 4.5.: Überblick über den allgemeinen Messaufbau

Logic Analyzer geführt. Im Bild nicht gezeigt sind weitere Imotes, die ebenfalls in manchen Experimenten eingesetzt wurden.

Koordination einer Messung: Auf dem PC kamen für eine Messung bis zu drei Programme zum Einsatz:

- Zum Mitschnitt der Signalpegel wurde der Logic Analyzer Sigma 2 verwendet. Dieser wird von *Sigma Logic Analyzer*, einem mitgelieferten Programm von ASIX, gesteuert. Dieses Programm wurde in dieser Arbeit durch ein neu entwickeltes Plug-In erweitert, welches Befehle zur Koordination, d. h. Beginn und Ende eines Messvorgangs, entgegennimmt und entsprechend verarbeitet. Zusätzlich ist es für die Sicherung der Rohdaten des Logic Analyzers nach jedem Messvorgang verantwortlich.

- Für die Überwachung des Mediums wurde das USRP 2 eingesetzt. Die Kontrolle über dieses Messgerät übernimmt *UHD Capture*. Dies ist ein eigens für diese Arbeit entwickeltes Programm, welches mit Hilfe der von Ettus Research zur Verfügung gestellten UHD-Programmbibliothek (*USRP Hardware Driver*) mit dem USRP 2 kommuniziert. Auch dieses nimmt Befehle zur Koordination des Messvorgangs entgegen und sicherte die Rohdaten des USRP 2, die während des Messvorgangs mitgeschnitten wurden.

- Für die Koordination der Experimente kommt eine für diese Arbeit entwickelte Programmbibliothek *libtrigger* zum Einsatz. Diese wird für jedes Experiment gegen ein Programm gebunden, das die Durchführung des jeweiligen Experiments koordiniert. Dabei übernimmt die Bibliothek gemeinsame Aufgaben wie z. B. das Ansteuern der Programme Sigma Logic Analyzer und UHD Capture sowie die Kommunikation mit dem Basys 2, um die Stimuli für den Messvorgang auszulösen. Darüber hinaus bietet

sie Schnittstellen zur einfachen Festlegung der Reihenfolge und der zeitlichen Abstände einzelner Aktionen. Ferner werden auch Wiederholungen der Messvorgänge durch das jeweilige Programm und die Bibliothek gesteuert.

Ein Beispiel eines solchen Programms ist in Anhang A gezeigt.

Auswertung einer Messung: Die Rohdaten, die während den Messvorgängen anfielen, wurden der jeweiligen Messung entsprechend ausgewertet. Da die Auswertung selbst, ähnlich wie die Durchführung, auf jedes Experiment individuell zugeschnitten ist, wurde für jedes Experiment ein eigenes Auswertungsprogramm entwickelt. Da jedoch viele Teilaufgaben der Auswertung über die Experimente hinweg ähnlich sind, sind diese zur gemeinsamen Verwendung in eine Programmbibliothek ausgelagert. Diese ist für folgende Aufgaben verantwortlich:

- Einlesen der Rohdaten des Logic Analyzers. Hinzu kommt hier noch das Suchen von Ereignissen, z. B. das Finden einer steigenden oder fallenden Signalflanke oder einer (komplexen) Signalkonstellation. Außerdem wird die Möglichkeit gegeben, Nachrichten des mitgeschnittenen CC 2420 SPI-Busses zu dekodieren und so nach bestimmten Nachrichten zu suchen und deren Zeitpunkte zu bestimmen.

  Eine wichtige Funktion dieses Subsystems ist außerdem das Bestimmen zeitlicher Abstände zwischen den jeweiligen Ereignissen.

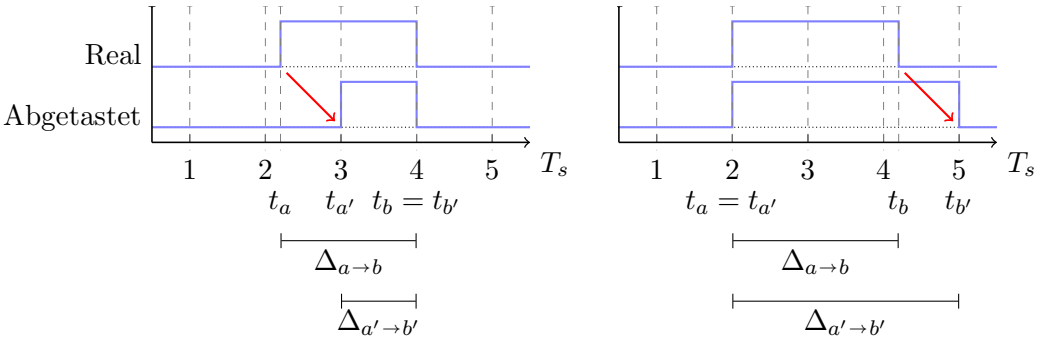
- Einlesen der Rohdaten des USRP 2. Hier kommt noch die Berechnung der Energie des ursprünglich empfangenen Signals über definierbare Zeitfenster hinzu sowie daraus folgend eine Erkennung der Medienbelegung.

  Durch die Modifikation des USRP 2 (Kap. 4.2.2) können externe binäre Signale in den Datenstrom des USRP 2 eingearbeitet werden. Das Subsystem trennt diese binären Signale vom mitgeschnittenen Funksignal und erlaubt auch bei diesen die Suche nach Signalflanken und die Bestimmung zeitlicher Abstände zwischen Ereignissen der Signale.

- Ausschreiben und Vorauswertung der Rohdaten des USRP 2. Da bei einer Aufzeichnung mit dem USRP 2 potentiell eine große Datenmenge anfällt (etwa 76 $^{\mathrm{MiB}}/_{\mathrm{s}}$ bei einer Abtastrate von 20 MHz), ist es für die eigentliche Auswertung und eine dauerhafter Speicherung der Messdaten vorteilhaft, wenn diese auf den für die Messung relevanten Teil zugeschnitten werden. Bei der Vorauswertung werden also lediglich die Rohdaten des USRP 2 auf den relevanten Teil gekürzt und im gleichen Format wieder ausgeschrieben.

- Auswertung der erhobenen Daten mit den Mitteln der deskriptiven Statistik. Hierzu zählen vor allem das Ableiten unterer und oberer Schranken, Häufungen und der Standardabweichung.

- Ausschreiben der ausgewerteten Daten als Eingabe für eine grafische Aufbereitung. Zu dieser gehören Signal-Zeit-Diagramme für die mitgeschnittenen Signale des Logic Analyzers oder des USRP 2 und deskriptive Diagramme wie z. B. Histogramme über Messwerte.



(a) Spätes Abtasten des Anfangsereignisses

(b) Spätes Abtasten des Endereignisses

Abbildung 4.6.: Ungenauigkeiten durch Diskretisierung der Zeit

4.3 Messungenauigkeiten

Die in dieser Arbeit durchgeführten Experimente wurden durch das gezielte Auslösen verschiedener Ereignisse gesteuert. Das System antwortet seinerseits wieder mit Ereignissen, die von außen beobachtet werden. Aus den zugehörigen Zeitpunkten können durch mathematische Operationen z. B. die Zeitspannen, die bei der jeweiligen Messung von Interesse sind, berechnet werden.

Die Auflösung der Messhardware, die zum Erfassen der Ereignisse eingesetzt wird, ist grundsätzlich begrenzt. Daher sind gemessene Zeitpunkte durch Ungenauigkeiten verfälscht, die durch die jeweilige Messhardware gegeben sind. Diese Ungenauigkeiten werden in den folgenden Unterabschnitten näher erläutert und maximale Schranken für diese Verfälschungen angegeben.

4.3.1 Ungenauigkeiten durch Diskretisierung der Zeit

Durch die Abtastung eines zeitkontinuierlichen Signals wird dessen zeitlicher Verlauf diskretisiert. Das bedeutet, dass Wechsel von Signalpegel nur noch zu den Abtastzeitpunkten sichtbar sind. Jedes Ereignis, das zwischen zwei konsekutiven Abtastzeitpunkten liegt, wird erst zum späteren Zeitpunkt sichtbar.

Die mathematische Beziehung ist wie folgt: Sei T_s die Abtastperiode, d. h. der zeitliche Abstand zwischen zwei Abtastzeitpunkten. Sei ferner a ein beliebiges Ereignis und t_a der tatsächliche und $t_{a'}$ der gemessene Zeitpunkt des Ereignisses. Dann gilt:

$$t_{a'} - T_s \;<\; t_a \;\leq\; t_{a'} \tag{4.8}$$

Messung einer Zeitspanne: In Abb. 4.6 ist gezeigt, wie sich diese Ungenauigkeit auf die Messung einer Zeitspanne, die von zwei Ereignissen a und b begrenzt wird, auswirkt. In Teilabbildung (a) tritt das Anfangsereignis a kurz nach einem Abtastzeitpunkt ein, das Endereignis b liegt genau auf einem Abtastzeitpunkt. Dadurch verkürzt sich die gemessene gegenüber der tatsächlichen Zeitspanne von fast zwei Abtastperioden auf eine.

Mathematisch gesehen gilt: Seien $t_a, t_{a'}, t_b, t_{b'}$ entsprechend der Teilabbildung. So gilt für die gemessene Zeitspanne $\Delta_{a' \to b'}$:

$$\begin{aligned} \Delta_{a' \to b'} &\overset{(4.1)}{=} t_{b'} - t_{a'} \\ &= t_b - t_{a'} \\ &\overset{(4.8)}{>} t_b - (t_a + T_s) \\ &= t_b - t_a - T_s \\ &\overset{(4.1)}{=} \Delta_{a \to b} - T_s \end{aligned} \tag{4.9}$$

In Teilabbildung (b) fällt das Anfangsereignis a mit einem Abtastzeitpunkt zusammen, das Endereignis b liegt kurz nach einem Abtastzeitpunkt. Die tatsächliche Zeitspanne ist somit knapp mehr als zwei Abtastperioden, die gemessene Zeitspanne hat jedoch eine Länge von drei Perioden. Die mathematische Herleitung ist analog zur obigen und ergibt:

$$\Delta_{a' \to b'} < \Delta_{a \to b} + T_s \tag{4.10}$$

Aus beiden Beziehungen lässt sich also folgender Schluss herleiten:

$$\Delta_{a \to b} - T_s < \Delta_{a' \to b'} < \Delta_{a \to b} + T_s \tag{4.11}$$

Auswertung der Spannweite: Wird die Messung einer Zeitspanne $\Delta_{a' \to b'}$ wiederholt, kann die Spannweite der gemessenen Größe als Differenz des minimalen und maximalen gemessenen Dauer berechnet werden. Im Worst-Case ist einer der beiden Werte gerade am linken Rand der in Formel 4.11 gezeigten Beziehung, während der andere Wert am rechten Rand ist. Durch diese Fälle lässt sich die Beziehung der gemessenen Spannweite S' gegenüber der tatsächlichen Spannweite S herleiten:

$$S - 2 \cdot T_s < S' < S + 2 \cdot T_s \tag{4.12}$$

Insbesondere kann also die tatsächliche Spannweite um fast zwei Periodenlängen größer sein als die gemessene.

4.3.2 Ungenauigkeiten durch Hardwarelaufzeiten

Die Vorgänge, deren Dauer gemessen werden soll, werden durch externe Trigger ausgelöst. Die Hardware, die diese Trigger wahrnimmt, muss diese zunächst verarbeiten, so dass letztlich der Vorgang, der beobachtet werden soll, verzögert stattfindet. Die Verarbeitung der Trigger kann aus mehreren Schritten bestehen, von denen jeder durch Laufzeiten der Hardware einer Verzögerung unterliegt. Diese Verzögerungen sind nur zum Teil bestimmbar und enthalten meist auch variable Anteile, die aus einer Messung nicht herausgerechnet werden können.

Bei der technischen Umsetzung der Experimente wurde darauf geachtet, dass möglichst wenige Ungenauigkeiten durch Hardwarelaufzeiten entstanden. Ein Ereignis, das selbst nicht von außen beobachtet werden kann (wie z. B. der Ablauf eines Timers), wurde durch ein beobachtbares Ereignis angenähert, das in einem möglichst konstanten Verhältnis zum eigentlichen Ereignis steht (z. B. Auslösen eines Wechsels des Signalpegels auf einem GPIO-Pin).

Für jedes durchgeführte Experiment werden im Folgenden die erhobenen Messdaten beschrieben, und erläutert, welche Verzögerungen und ggf. weitere Einflüsse dabei involviert sind.

4.4 Experimente zum Verhalten des Transceivers

Im Datenblatt des CC 2420-Transceivers [41] sind das funktionale und zeitliche Verhalten der Hardware angegeben. Da das zeitliche Verhalten für das erfolgreiche Funktionieren Black Burst-basierter Protokolle kritisch ist, werden in den nachfolgenden Experimenten die im Datenblatt angegebenen Zeitspannen verifiziert oder ggf. näher bestimmt.

Überdies sollen auch Abweichungen des angegebenen funktionalen Verhaltens sowie undokumentiertes Verhalten der Hardware gefunden und dokumentiert werden. Außerdem wird nach Optimierungen gesucht, die eine effiziente Implementierung Black Burst-basierter Protokolle zulassen.

Für jedes Experiment werden die Messdaten und ihr Nutzen für die Implementierung erläutert. Weiter werden die Durchführung der Messungen und Ziele beschrieben. Nach einem Einblick in das jeweiligen Messergebnis werden entsprechende Folgerungen für die Implementierung abgeleitet.

4.4.1 Umschaltverzögerung RX → TX

Die Umschaltdauer d_{rxtx} bezeichnet die Dauer, die der Transceiver benötigt, um vom Empfangs- in den Sendezustand zu wechseln. In dieser Zeitspanne wird u. A. der Frequenzgenerator neu kalibriert und die Leistungsverstärker eingeregelt. Der Frequenzgenerator ist mit einer Phasenregelschleife [17] (PLL – *Phase Locked Loop*) aufgebaut. Die PLL generiert mit Hilfe eines spannungsgesteuerten Oszillators (VCO – *Voltage Controlled Oscillator*) ein Ausgangssignal einer bestimmten Frequenz, die mit Hilfe der PLL gegen Ungenauigkeiten aufgrund von Alterung der Bauteile und Temperaturschwankungen stabilisiert wird. Durch die Regelstrecke wird der Oszillator nachgeregelt, bis der Fehler der Phasenlage unter einen fest definierten Schwellwert sinkt. Dieser Vorgang wird auch als „Einrasten“ (*Lock*) bezeichnet. Das bedeutet auch, dass bei einem Frequenzwechsel (der u. A. beim Wechsel vom Empfangs- in den Sendemodus stets nötig ist) ein erhöhter Aufwand zur erneuten Einreglung erforderlich ist. Die Zeitspanne bis zum Lock, ist i. Allg. nicht konstant. Sie wird vom CC 2420 mit dem Wert von d_{rxtx} pessimistisch nach oben abgeschätzt. Der Transceiver beginnt seine Übertragung also nicht direkt nachdem der Frequenzgenerator kalibriert ist, sondern erst wenn die Zeitspanne d_{rxtx} nach Eintreffen des Sendebefehls abgewartet wurde.

In BBS (Kap. 2.3.4) und weiteren Black Burst-basierten Protokollen wird ein fester Wert für die Umschaltdauer benötigt, da diese u. A. die Bitdauer d_{Bit} mitbestimmt. Wird sie als zu klein angenommen, kann es je nach Verfahren zu einer Verletzung der Bedingungen kommen, die zum erfolgreichen Verhalten des Protokolls nötig sind. Ferner müssen mögliche Schwankungen der Umschaltdauer berücksichtigt werden. In BBS tragen diese zur Synchronisationsungenauigkeit bei. Daher sollten die Schwankungen so genau wie möglich charakterisiert werden.

Der CC 2420 bietet zwei Alternativen für die Umschaltdauer, die sich per Konfiguration der CC 2420-Register wählen lassen. Für die erste Alternative ist im Datenblatt eine Umschaltdauer von zwölf Symbolperioden, also $d_{rxtx} = 192\,\mu s$, angegeben. Die zweite

Alternative verkürzt die Umschaltdauer auf acht Symbolperioden. Entsprechend gilt in dieser Konfiguration $d_{rxtx} = 128\,\mu s$.

Im folgenden Experiment wird nun die Umschaltdauer vom Empfangs- in den Sendemodus näher bestimmt. Insbesondere wird auch untersucht, ob eventuelle Schwankungen von der Wahl der Alternative beeinflusst werden.

Zur Abgrenzung von der durch das Datenblatt vorgegebenen Umschaltdauer wird für den gemessenen Wert die Bezeichnung *Umschaltverzögerung* verwendet. Da es sich um eine gemessene Größe handelt, wird sie entsprechend der Konventionen mit Δ_{rxtx} bezeichnet.

Aufbau und Durchführung: Der Transceiver leitet eine Übertragung ein, sobald der Sendebefehl `STXON` über den SPI-Bus gegeben wurde. Nach abgeschlossener Umschaltung in den Sendemodus wird dann entsprechend des IEEE 802.15.4-Rahmenformats (Kap. 3.2.1.5) der SyncHeader, bestehend aus Präambel und SFD-Feld, gesendet. Nachdem das SFD-Feld gesendet wurde, löst der Transceiver eine steigende Flanke auf dem `SFD`-Pin aus.

Der Messaufbau folgt dem allgemeinen Messaufbau in Abb. 4.5. Durch einen Mitschnitt des SPI-Busses durch den Logic Analyzer kann der Zeitpunkt bestimmt werden, zu dem der Befehl vollständig übertragen ist. Laut Datenblatt wird ein Befehl interpretiert, sobald er über den SPI-Bus empfangen wurde. Der Zeitpunkt, zu dem der `STXON`-Befehl vollständig übertragen ist, wird im Folgenden als $t_{\texttt{STXON}}$ bezeichnet. Der Zeitpunkt der steigenden Flanke des `SFD`-Signals kann ebenfalls über Abgreifen des Signals mit dem Logic Analyzer erkannt werden und wird mit $t_{\lrcorner\texttt{SFD}}$ bezeichnet.

Ferner gibt $d_{syncHdr}$ die Zeitspanne an, die zum Übertragen des SyncHeaders nötig ist. Diese hängt von der Länge der Präambel und der Symboldauer d_{sym} ab. Die Präambellänge kann beim CC 2420 nahezu frei konfiguriert werden (Kap. 3.2.1.5). In diesem Experiment ist sie jedoch konform zum IEEE 802.15.4-Standard auf acht Symbole festgelegt. Zusammen mit den zwei Symbolen des SFD-Feldes hat der SyncHeader also eine Länge von zehn Symbolen. Die Dauer des SyncHeaders setzt sich somit aus konstanten Werten zusammen und wird wie folgt berechnet:

$$d_{syncHdr} = 10 \cdot d_{sym} \overset{(3.1)}{=} 10 \cdot 16\,\mu s = 160\,\mu s \tag{4.13}$$

Da die steigende Flanke des `SFD`-Signals nach dem Senden des SFD-Feldes ausgelöst wird, kann nun der tatsächliche Zeitpunkt des Übertragungsbeginns $t_{\texttt{TX}}$ bestimmt werden:

$$t_{\texttt{TX}} = t_{\lrcorner\texttt{SFD}} - d_{syncHdr} \tag{4.14}$$

In Abb. 4.7 ist der zeitliche Verlauf der Signale und Ereignisse visualisiert. Tatsächlich gemessen wird die Zeitspanne $\Delta_{\texttt{STXON}\rightarrow\lrcorner\texttt{SFD}}$, d. h. die Dauer zwischen Ende des Sendebefehls und der steigenden `SFD`-Flanke. Zusammen mit $d_{syncHdr}$ kann hieraus die Umschaltverzögerung berechnet werden:

$$\begin{aligned}
\Delta_{rxtx} &= \Delta_{\texttt{STXON}\rightarrow\texttt{TX}} \\
&= t_{\texttt{TX}} - t_{\texttt{STXON}} \\
&\overset{(4.14)}{=} t_{\lrcorner\texttt{SFD}} - d_{syncHdr} - t_{\texttt{STXON}} \\
&= \Delta_{\texttt{STXON}\rightarrow\lrcorner\texttt{SFD}} - d_{syncHdr}
\end{aligned} \tag{4.15}$$

Das Basys 2 generiert mit dem Trigger-Signal einen Impuls (eine steigende Flanke, gefolgt von einer fallenden), der das Programm, das auf dem Imote 2 ausgeführt wird, veranlässt,

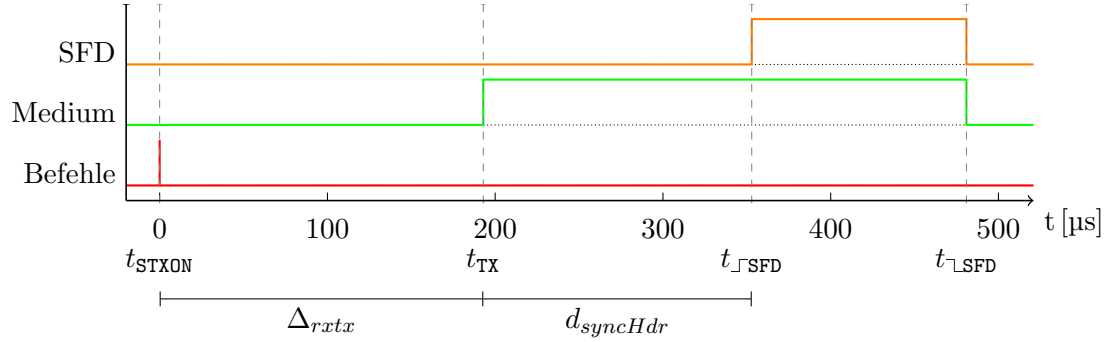


Abbildung 4.7.: Signal-Zeit-Diagramm zur Bestimmung der Umschaltverzögerung Δ_{rxtx}

den Sendebefehl `STXON` über den SPI-Bus auszugeben. Die Verzögerung, die durch dieses Programm zustande kommt, hat keine Auswirkung auf das Messergebnis, da erst der Zeitpunkt des Sendebefehls ($t_{\texttt{STXON}}$) aufgezeichnet wird und als Startzeitpunkt der Messung gilt.

Da es sich um eine Messung handelt, die voraussichtlich durch variable Verzögerungen (z. B. durch das Interpretieren des `STXON`-Befehls durch den CC 2420) beeinflusst wird, wird der Messvorgang 1000 mal wiederholt, um ein aussagekräftiges Ergebnis zu erhalten. Der Logic Analyzer tastet die Signalleitungen mit einer Abtastrate von 100 MHz ab, d. h. die Abtastperiode beträgt 10 ns.

Ergebnis und Folgerungen: Tabelle 4.1 gibt einen Überblick über die statistische Auswertung der Messdaten. Aus dieser ist ersichtlich, dass zur konfigurierten Umschaltdauer d_{rxtx} eine variable Zeitspanne hinzukommt. Die Wahl der Umschaltdauer d_{rxtx} hat dabei keinen Einfluss auf die Schwankungen.

Als obere Schranke der Umschaltverzögerung sollte aufgrund dieser Messungen unter Berücksichtigung der Ungenauigkeiten $\Delta_{rxtx,max} = 193{,}01\,\mu s$ für eine $d_{rxtx} = 192\,\mu s$ bzw. $\Delta_{rxtx,max} = 129{,}01\,\mu s$ für eine $d_{rxtx} = 128\,\mu s$ angenommen werden.

Ursache: Die Ursache für die Verzögerung lässt sich nur vermuten, da der exakte Aufbau der Hardware unbekannt ist. Im Folgenden soll dennoch eine Erklärung für den variablen Anteil der Verzögerung angegeben werden.

	Umschaltdauer		
	192 µs	128 µs	Ungenauigkeit
Minimum	192,86 µs	128,86 µs	±10 ns
Maximum	193,00 µs	129,00 µs	±10 ns
Spannweite	140 ns	140 ns	±20 ns
Mittelwert	192,93 µs	128,93 µs	
Standardabweichung	37,06 ns	36,91 ns	

Tabelle 4.1.: Statistische Auswertung der Umschaltverzögerung Δ_{rxtx}

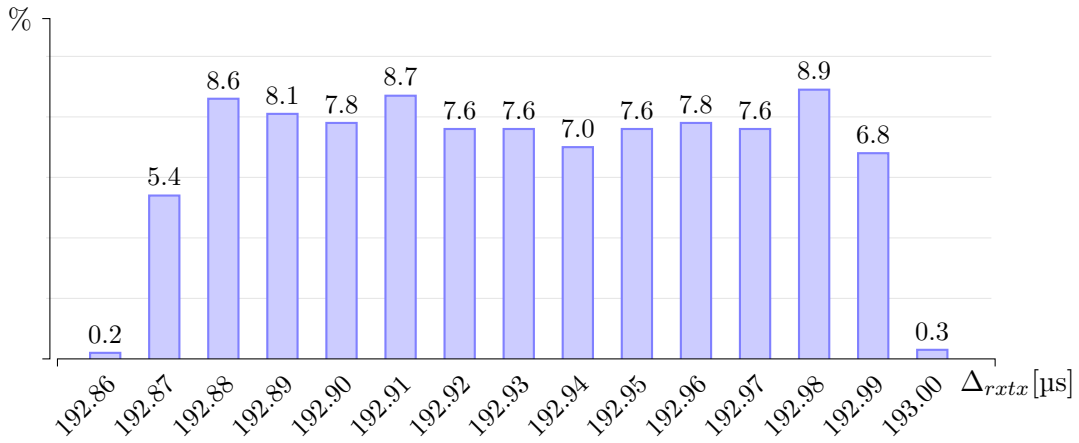


Abbildung 4.8.: Relative Häufigkeit von der gemessenen Umschaltverzögerung Δ_{rxtx} bei $d_{rxtx} = 192\,\mu s$

Der SPI-Bus wird durch den Master (in diesem Fall also vom PXA 271) getaktet. Mit diesem Takt wird der Sendebefehl `STXON` zum Transceiver gegeben. Dieser muss die Daten des SPI-Busses vom externen Takt auf einen internen Takt übersetzen, um die Befehle interpretieren zu können. Dies wird in Hardwareschaltungen oft durch eine Abtastung erreicht. Ähnlich wie beim Abtasten eines Signals durch den Logic Analyzer entstehen hierbei Verzögerungen.

In Abb. 4.8 ist die Häufigkeitsverteilung von Δ_{rxtx} über die 1000 Messvorgänge dargestellt. Die gemessenen Werte treten, abgesehen von den Randwerten, etwa gleich häufig auf. Dies weist auf eine Verzögerung bei der Übernahme der Daten des SPI-Busses hin.

Als weitere Unterstützung dieser These wurde zusätzlich das Verhalten des `CCA`-Signals bei einem Übertragungsvorgang des CC 2420 untersucht, indem zusätzlich das `CCA`-Signal in den Messungen mitgeschnitten wurde. In Kap. 3.4 wurde beschrieben, dass das `CCA`-Signal während des Sendevorgangs kein gültigen Wert besitzt. Dennoch wurde durch Messungen belegt, dass es ein definiertes Signalverhalten zeigt: Nachdem der Sendebefehl `STXON` gegeben wird, wird das `CCA`-Signal auf den logischen 0-Pegel gesetzt. War der Kanal zuvor unbelegt (also `CCA` = 1), kann dieses Ereignis als fallende Flanke des Signals beobachtet werden. Die Zeitspanne zwischen dem `STXON`-Befehl und der fallenden `CCA`-Flanke, $\Delta_{\texttt{STXON}\rightarrow\texttt{CCA}}$, wurde aus den vorhandenen Messdaten bestimmt.

Unter Vernachlässigung der Messungenauigkeit geht aus der Auswertung folgender Zusammenhang hervor:

$$240\,\text{ns} \;\leq\; \Delta_{\texttt{STXON}\rightarrow\texttt{CCA}} \;\leq\; 380\,\text{ns} \tag{4.16}$$

Auch hier beträgt die Spannweite genau wie in der ursprünglichen Messung 140 ns. Überdies war die Dauer zwischen der fallenden Flanke des `CCA` und der steigenden Flanke des `SFD`, also $\Delta_{\texttt{CCA}\rightarrow\texttt{SFD}}$, bis auf die Messungenauigkeit konstant. Dies verstärkt die Vermutung, dass der variable Anteil der Umschaltverzögerung durch Abtastung des SPI-Busses vom Transceiver begründet ist.

4.4.2 Manuelle Einleitung der Kalibrierung zur Optimierung des Sendevorgangs

Der CC 2420 bietet mit dem `STXCAL`-Befehl einen weiteren Befehl, der die Hardware in den Sendezustand überführt. Der Unterschied ist jedoch, dass der im vorherigen Abschnitt vorgestellte `STXON`-Befehl nach abgeschlossener Umschaltung direkt den Übertragungsvorgang einleitet, während `STXCAL` lediglich den Frequenzgenerator neu einregelt. Danach kann durch einen `STXON`-Befehl die Übertragung eingeleitet werden. Die nachfolgende Messung soll feststellen, ob die Verzögerung zwischen dem `STXON`-Befehl und dem tatsächlichen Übertragungsbeginn durch diese Vorgehensweise kürzer ist als das zuvor bestimmte Δ_{rxtx}. Zur Abgrenzung wird die hier gemessene Umschaltverzögerung im Folgenden mit Δ_{caltx} bezeichnet.

In einem weiteren Teilversuch soll das zeitliche und funktionale Verhalten der Hardware überprüft werden, wenn die Zeitspanne zwischen `STXCAL` und `STXON`-Befehl minimiert wird. Insbesondere soll auch untersucht werden, welche Auswirkungen die Wahl einer Zeitspanne hat, die kürzer als die vom Datenblatt vorgegebene maximale Kalibrierungsdauer ist. In diesem Zustand kann es vorkommen, dass die Phasenregelschleife des Frequenzgenerators noch nicht eingerastet ist. Es soll dennoch untersucht werden, inwiefern sich dies auf das Verhalten der Funkhardware auswirkt. Falls die PLL zum Sendezeitpunkt bereits hinreichend genau kalibriert ist, könnte eine kürzere Umschaltverzögerung erwirkt werden.

Aufbau und Durchführung: Der Aufbau folgt wieder dem allgemeinen Messaufbau aus Abb. 4.5. Das Programm des Imote 2 wird dahingehend geändert, dass zunächst der `STXCAL`-Befehl und nach einer definierten Zeitspanne der `STXON`-Befehl an den Transceiver übertragen wird. Diese Zeitspanne kann durch das Programm einem eingestellten Wert ($\Delta_{\texttt{STXCAL}\rightarrow\texttt{STXON}}$) angenähert werden, ist jedoch durch Hardwarelaufzeiten verzögert. Die tatsächliche Dauer, die zwischen den beiden Befehlen liegt, kann allerdings bei der Auswertung aus den aufgezeichneten Daten bestimmt werden. Somit trägt diese Hardwarelaufzeit nicht zur Ungenauigkeit bei.

Wie im vorherigen Experiment wird der eigentliche Übertragungsbeginn $t_{\texttt{TX}}$ mit Hilfe des `SFD`-Signals und der Dauer des SyncHeaders $d_{syncHdr}$ berechnet.

In Abb. 4.9 ist das Signal-Zeit-Diagramm dieses Versuchs dargestellt. Von Interesse ist wieder die Zeit zwischen dem `STXON`-Befehl und dem Sendebeginn, also $\Delta_{caltx} = \Delta_{\texttt{STXON}\rightarrow\texttt{TX}}$, die analog nach Formel 4.15 berechnet wird.

4.4.2.1 Erster Teilversuch: Lange Kalibrierungsdauer

In einem ersten Teilversuch wird zunächst das Verhalten bei einer langen Pause zwischen den Befehlen `STXCAL` und `STXON` untersucht. Die Dauer der Pause beträgt $\Delta_{\texttt{STXCAL}\rightarrow\texttt{STXON}} \approx 200\,\mu s$.

Ergebnis: Den Messergebnissen in Tab. 4.2 kann entnommen werden, dass die Umschaltverzögerung Δ_{caltx} gegenüber Δ_{rxtx} stark verkürzt ist. Sie beträgt ca. 11,12 µs, ist jedoch rund 10 µs größer als erwartet, da erwartet wurde, dass der Transceiver nach dem Erkennen des `STXON`-Befehls direkt mit der Übertragung beginnt.

Ursache: Die Kalibrierung des Frequenzgenerators ist nicht die einzige Aufgabe, die der Transceiver während des Umschaltvorgangs durchführt. Parallel dazu werden die

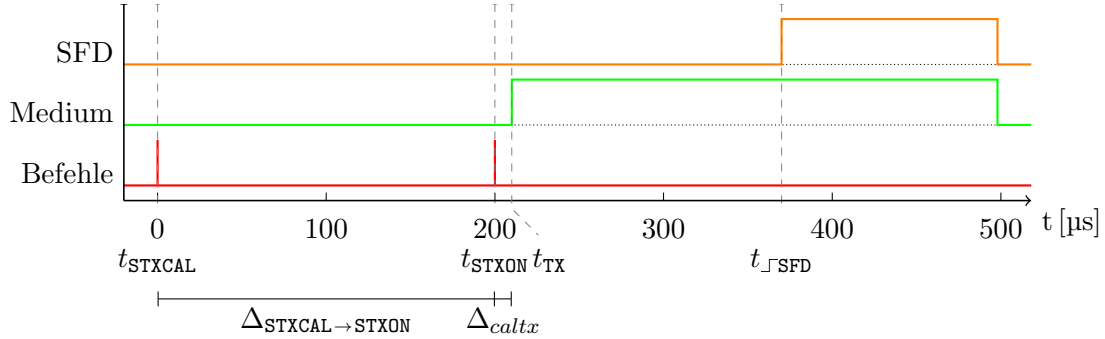


Abbildung 4.9.: Signal-Zeit-Diagramm zur Bestimmung der Umschaltverzögerung Δ_{caltx} bei manueller Kalibrierung

	Δ_{caltx}	Ungenauigkeit
Minimum	11,11 µs	±10 ns
Maximum	11,25 µs	±10 ns
Spannweite	140 ns	±20 ns
Mittelwert	11,18 µs	
Standardabweichung	36,04 ns	

Tabelle 4.2.: Statistische Auswertung von Δ_{caltx} bei $d_{PA} = 10\,\mu s$

Leistungsverstärker eingeregelt, die das Funksignal verstärken, bevor es über die Antenne abgestrahlt wird. Da die Dauer dieses Vorgangs im CC 2420 durch die Transition des internen Zustandsautomaten gesteuert wird und mit einem festen Wert konfiguriert wird, wird sie im Folgenden als *Transitionsdauer* d_{PA} bezeichnet. In der Standardkonfiguration des Transceivers ist eine Dauer von $d_{PA} = 10\,\mu s$ vorgesehen. Sie kann jedoch mit Werten zwischen 0 µs und 15 µs gewählt werden.

4.4.2.2 Zweiter Teilversuch: Reduzierung der Transitionsdauer

In diesem Teilversuch wird das Verhalten des Transceivers bei einer nun mit $d_{PA} = 0\,\mu s$ konfigurierten Transitionsdauer untersucht. Es wird weiterhin eine manuelle Kalibrierung mit einer langen Pause von $\Delta_{\texttt{STXCAL}\rightarrow\texttt{STXON}} \approx 200\,\mu s$ verwendet. Insbesondere wird bei diesem Experiment auch untersucht, ob die Kalibrierung der Leistungsverstärker bereits mit dem `STXCAL`-Befehl einhergeht. Ist dies nicht der Fall, beginnt der Transceiver „zu früh“ mit dem Sendevorgang, so dass die ersten Symbole mit einer geringeren Leistung übertragen werden.

Aufbau und Durchführung: Wieder folgt der Aufbau dem allgemeinen Messaufbau aus Abb. 4.5. Neu hinzu kommt das USRP 2, welches die Rohdaten des Funksignals mitschneidet, so dass bei der Auswertung die Energie auf dem Medium berechnet werden kann.

Die Durchführung ändert sich im Vergleich zum vorherigen Teilversuch nur insofern, dass nun auch die Energie des Funksignals über die Zeit gemessen wird, um einen eventuellen Anstieg der Leistung während der ersten Symbole feststellen zu können.

	Δ_{caltx}	Ungenauigkeit
Minimum	1,11 µs	±10 ns
Maximum	1,25 µs	±10 ns
Spannweite	140 ns	±20 ns
Mittelwert	1,18 µs	
Standardabweichung	36,44 ns	

Tabelle 4.3.: Statistische Auswertung von Δ_{caltx} bei $d_{PA} = 0\,\mu s$

Ergebnis und Folgerungen: Den Messergebnissen in Tab. 4.3 kann entnommen werden, dass die Umschaltverzögerung $\Delta caltx$ wie erwartet um weitere 10 µs reduziert wurde.

Ferner war auch kein Leistungsanstieg über die Dauer der Übertragung zu verzeichnen. Daher wird davon ausgegangen, dass die Leistungsverstärker ebenfalls mit dem `STXCAL`-Befehl bereits kalibriert werden und die Konfiguration $d_{PA} = 0\,\mu s$ ohne Bedenken gewählt werden kann.

Mit der Konfiguration von $d_{PA} = 0\,\mu s$ ist es folglich möglich, einen Sendevorgang innerhalb von etwa 1,18 µs zu beginnen. Allerdings soll an dieser Stelle nochmals darauf verwiesen werden, dass dies nur gilt, wenn die eigentliche Kalibrierung (und damit einhergehend eine Blindphase) vor dem Sendebefehl durchgeführt wird. Ist diese Forderung vertäglich mit dem verwendeten Kommunikationsprotokoll und die verkürzte Dauer zwischen Sendebefehl und Sendebeginn von Vorteil, kann der CC 2420 mit $d_{PA} = 0$ konfiguriert werden.

4.4.2.3 Dritter Teilversuch: Verkürzte Kalibrierungsdauer

In diesem Teilversuch wird nun schließlich die Pause zwischen dem `STXCAL`- und `STXON`-Befehl auf ein Minimum verkürzt. Dazu werden beide Befehle direkt hintereinander in einer einzelnen SPI-Transaktion zum Transceiver übertragen. Die Transitionsdauer wird weiterhin auf $d_{PA} = 0\,\mu s$ konfiguriert.

Aufbau und Durchführung: Der Aufbau und die Durchführung sind analog zum vorherigen Versuch. Hinzu kommt noch ein zweiter Imote 2. Mit diesem wird geprüft, ob der Rahmen, der versendet wird, korrekt als regulärer IEEE 802.15.4-Rahmen empfangen wird.

Der zeitliche Abstand zwischen den Befehlen ($\Delta_{\texttt{STXCAL}\rightarrow\texttt{STXON}}$) verkürzt sich auf die reine Übertragungsdauer durch den SPI-Bus. Beim gewählten Bustakt von 7,5 MHz beträgt diese etwa 1,23 µs.

Abbildung 4.10 stellt das Signal-Zeit-Diagramm dieses Versuchs dar. Von Interesse ist hier die Zeit zwischen dem Kalibrierungsbeginn (`STXCAL`) und dem tatsälichen Sendebeginn, also $\Delta_{rxtx,short} = \Delta_{\texttt{STXCAL}\rightarrow\texttt{TX}}$. Der Sendezeitpunkt $t_{\texttt{TX}}$ wird wie in den vorherigen Versuchen mit Hilfe des `SFD`-Signals gewonnen (nicht in der Abbildung dargestellt).

Vorläufiges Ergebnis: Der Transceiver sendet (abgesehen von der SPI-Verarbeitungsungenauigkeit) direkt nachdem der `STXON`-Befehl eingegangen ist. Das bedeutet, dass die Übertragung beginnt, obwohl der Frequenzgenerator möglicherweise noch nicht eingerastet ist.

Dennoch war mit Hilfe des USRP 2 eine Medienbelegung erkennbar, die auch vom CCA-Mechanismus des zweiten Imote 2 wahrgenommen wurde. Der Rahmeninhalt konnte jedoch

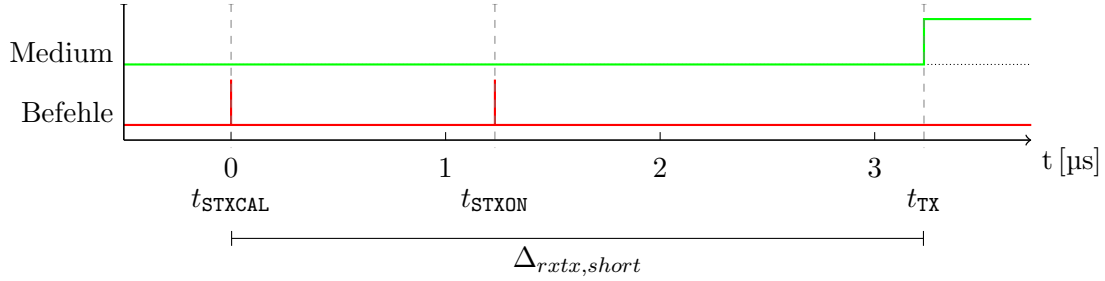


Abbildung 4.10.: Signal-Zeit-Diagramm zur Bestimmung der verkürzten Umschaltverzögerung $\Delta_{rxtx,short}$ bei manueller Kalibrierung

	$\Delta_{rxtx,short}$	Ungenauigkeit
Minimum	4,35 µs	±10 ns
Maximum	4,48 µs	±10 ns
Spannweite	130 ns	±20 ns
Mittelwert	4,41 µs	
Standardabweichung	38,6 ns	

Tabelle 4.4.: Statistische Auswertung von $\Delta_{rxtx,short}$

nicht dekodiert werden, da der Rahmen nicht als solcher erkannt wurde. Da jedoch bei Black Bursts die reine Medienbelegung und nicht die Nutzdaten des IEEE 802.15.4-Rahmens von Bedeutung ist, könnte mit diesem Vorgehen die Umschaltverzögerung und somit die Blindphase vor dem Sendevorgang erheblich verkürzt werden.

In Tab. 4.4 ist das Messergebnis angegeben. Die Umschaltverzögerung zwischen Empfangsmodus und Sendebeginn $\Delta_{rxtx,short}$ kann bei der verkürzten Kalibrierung demnach auf etwa 4,4 µs verkürzt werden.

Frequenzabhängigkeit des Verhaltens: Bei einer genaueren Betrachtung von $\Delta_{rxtx,short}$ war auffallend, dass das Verhalten von der gewählten Trägerfrequenz (also vom verwendeten Kanal) abhängig ist. Aus diesem Grund wurde das Experiment für jede Trägerfrequenz zwischen 2405 MHz und 2480 MHz in 1 MHz-Schritten wiederholt und dabei die Leistung einer Übertragung ohne verkürzte Kalibrierung (P_{normal}) gegen die Leistung einer Übertragung mit verkürzter Kalibrierung (P_{short}) verglichen. Bei Logarithmieren der Leistungen kann die Dämpfung ΔP als Differenz berechnet werden. Die Einheit ist 1 dB.

Abbildung 4.11 zeigt das Ergebnis dieser Messung: Bis zu einer Frequenz von 2242 MHz (was in etwa Kanal 18 entspricht) ist keine Abschwächung der Signalleistung erkennbar. Ab dieser Frequenz steigt die Dämpfung stark an. Bei 2453 MHz liegt eine Dämpfung von etwa -60 dB vor. Praktisch wird also beim unkalibrierten Übertragen ab dieser Frequenz keine Medienbelegung mehr erkannt.

Die Ursache der Frequenzabhängigkeit liegt vermutlich darin, dass die PLL des Frequenzgenerators bei höheren Frequenzen eine längere Dauer benötigt, bis bei einer Neukalibrierung wieder Regelverhalten vorliegt.

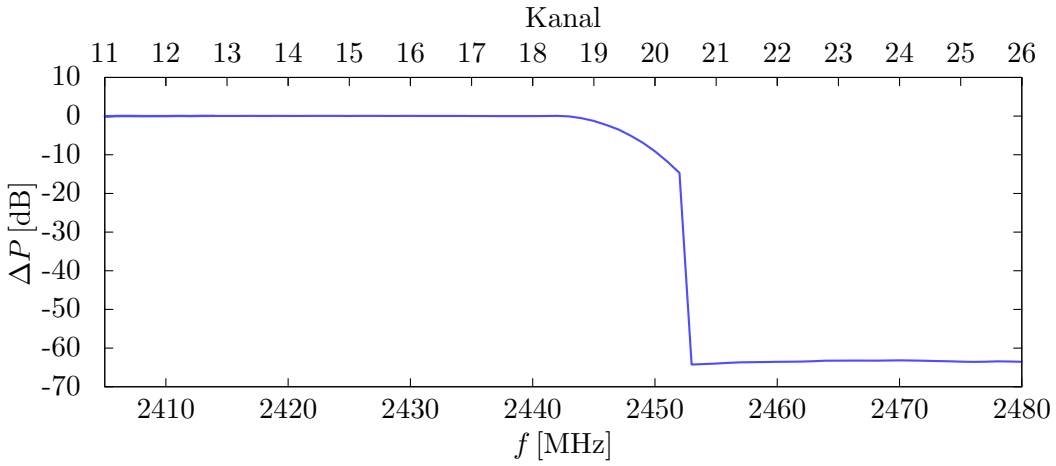


Abbildung 4.11.: Dämpfung der Sendeleistung bei stark verkürzter Kalibrierung in Abhängigkeit der Trägerfrequenz

Weitere Beobachtungen: Neben der Frequenzabhängigkeit wurde weiter beobachtet, dass der Effekt hochgradig von Bauteiltoleranzen abhängig ist. Beim Durchführen weniger Messdurchläufe mit anderen Transceivern war die Dämpfung auch bei niedrigen Trägerfrequenzen bereits sehr hoch, so dass erst durch Vergrößern des Abstands zwischen Kalibrierungs- und Übertragungsbefehl eine Medienbelegung wahrgenommen werden konnte.

Folgerungen: Durch das unkalibrierte Senden kann die Blindphase zwischen Empfangsmodus und Übertragungsbeginn zwar erheblich verkürzt werden, jedoch sind dabei einige Besonderheiten zu beachten:

- Die Übertragung wird nur als Energie wahrgenommen. Das Vorgehen kann also nicht genutzt werden, um reguläre IEEE 802.15.4-Rahmen zu versenden.
- Abhängig von der Trägerfrequenz ist das gesendete Signal stark gedämpft.
- Die Signaldämpfung ist abhängig von Fertigungstoleranzen.

Gerade die Frequenz- und Bauteilabhängigkeit macht die Handhabung dieser Methode ziemlich unüberschaubar, da sie nicht universell einsetzbar ist. Wenn die stark verkürzte Kalibrierungsdauer genutzt werden soll, müssen folglich alle Transceiver, die potentiell im Netzwerk zum Einsatz kommen, genau getestet werden, um das Funktionieren auf der gewählten Trägerfrequenz sicherzustellen. Dies ist mit einem erheblichen Aufwand verbunden.

Im Hinblick auf das Master-basierte BBS-Verfahren ist eine kürzere Umschaltdauer ohnehin nicht von hohem Nutzen. Die Bitdauer (und somit die Konvergenzdauer) wird beim Einsatz des CC 2420 durch andere Bedingungen (insbesondere d_{maxCCA}) dominiert, so dass hier eine Umschaltdauer von weniger als $d_{rxtx} = 128\,\mu s$ keine Vorteile hat. Lediglich eine Verringerung der Schwankung wäre vorteilhaft. Dies kann durch diese Maßnahme jedoch nicht erzielt werden. Daher fließen die Ergebnisse dieser Messung nicht in die Implementierung ein.

4.4.3 Umschaltverzögerung TX → RX

Nach einem Sendevorgang wechselt der CC 2420 wieder vom Sende- in den Empfangszustand. Bei diesem Zustandswechsel muss der Frequenzgenerator rekalibriert werden. Im Datenblatt des CC 2420 ist hierfür eine Zeitspanne 12 Symboldauern, also $d_{txrx} = 192\,\mu s$, angegeben. Da die PLL jedoch auch für einen Sendevorgang bei $d_{rxtx} = 128\,\mu s$ innerhalb von 128 µs einrastet, liegt die Vermutung nahe, dass die Obergrenze der Einrastdauer und somit auch des Umschaltvorgangs vom Sende- in den Empfangsmodus bei 128 µs liegt.

Von Bedeutung ist die Umschaltverzögerung, die in diesem Versuch als Δ_{txrx} bezeichnet wird, für Protokolle, die während einer Runde potentiell senden und empfangen müssen. Beispiele hierfür sind Busy-Tone-Protokolle sowie das dezentralisierte BBS-Verfahren und ACTP [14]. Für die Funktion dieser Protokolle muss bekannt sein, ab wann ein Knoten, der gerade selbst eine Übertragung abgeschlossen hat, frühstens wieder eine Übertragung eines anderen Knotens wahrnehmen kann.

Aufbau und Durchführung: Der Aufbau ähnelt wieder dem generellen Aufbau, der in Abb. 4.5 gezeigt ist. Für diesen Versuch sind lediglich die `CCA`- und `SFD`-Signale sowie das übersetzte `RSSI_VALID`-Signal von Bedeutung. Ein Abgreifen des SPI-Busses ist nicht nötig. Somit müssen weniger Signale an den Logic Analyzer geführt werden, was eine Erhöhung der Abtastrate (und somit der Genauigkeit) zulässt. Die Abtastrate beträgt in diesem Experiment 200 MHz, die Abtastperiode entsprechend 5 ns.

Der Aufbau wurde um einen weiten Imote 2 erweitert, dessen Transceiver in einem Testmodus betrieben wird. In diesem Testmodus sendet der CC 2420 dauerhaft eine unmodulierte Trägerwelle, was erst wieder durch einen Reset des Chips beendet wird.

Zusätzlich soll noch das chipinterne `RSSI_VALID`-Signal überwacht werden. Da dies nicht über einen Pin nach außen gelegt ist, muss es über den SPI-Bus ausgelesen und vom Programm des Imote 2 via GPIO-Pin nach außen geführt werden, so dass ein Abgriff durch den Logic Analyzer und so eine Aufzeichnung ermöglicht wird. Das Auslesen des Signals über den SPI-Bus sowie der weitere Übersetzungsvorgang verzögern zwar das Signal, lassen unter bestimmten Umständen aber dennoch Folgerungen zu.

Aus dem Experiment zur Bestimmung der Umschaltzeit zum Senden (Kap. 4.4.1) ist bekannt, dass das `CCA`-Signal beim Sendevorgang einen logischen 0-Pegel führt. Ferner ist aus diesem Experiment auch bekannt, dass der `CCA`-Pegel nach einer gewissen Zeitspanne nach einer Übetragung wieder zum logischen 1-Pegel zurückwechselt. In einer Vormessung wurde beobachtet, dass dieses Verhalten des `CCA`-Signals nicht von der tatsächlichen Medienbelegung beeinflusst wird. Das bedeutet, dass das `CCA`-Signal auch zum logischen 1-Pegel zurückkehrt, wenn das Medium weiterhin (durch einen zweiten Sender) belegt wird.

Der zeitliche Verlauf der Signale ist in Abb. 4.12 gezeigt. Die Umschaltverzögerung Δ_{txrx} entspricht der Zeitspanne, nach der wieder eine Medienbelegung erkannt werden kann. Da das Erkennen der Medienbelegung durch den CCA-Mechanismus verzögert ist, gilt:

$$\Delta_{txrx} = \Delta_{\neg\lfloor \texttt{SFD} \rightarrow \neg\lfloor \texttt{CCA}} - \Delta_{cca} \tag{4.17}$$

Dabei gibt Δ_{cca} die tatsächliche CCA-Verzögerung an. Durch eine passende Wahl der Sendeleistung des Senders und einen hohen Schwellwert der CCA-Erkennung beim zu testenden Transceiver konnte erreicht werden, dass nahezu das gesamte Integrationsfenster des CCA-Mechanismus verwendet werden musste, um eine Medienbelegung zu erkennen. In diesem Fall ist $\Delta_{cca} \approx d_{maxCCA}$ und hinreichend konstant, so dass Δ_{txrx} nach obiger

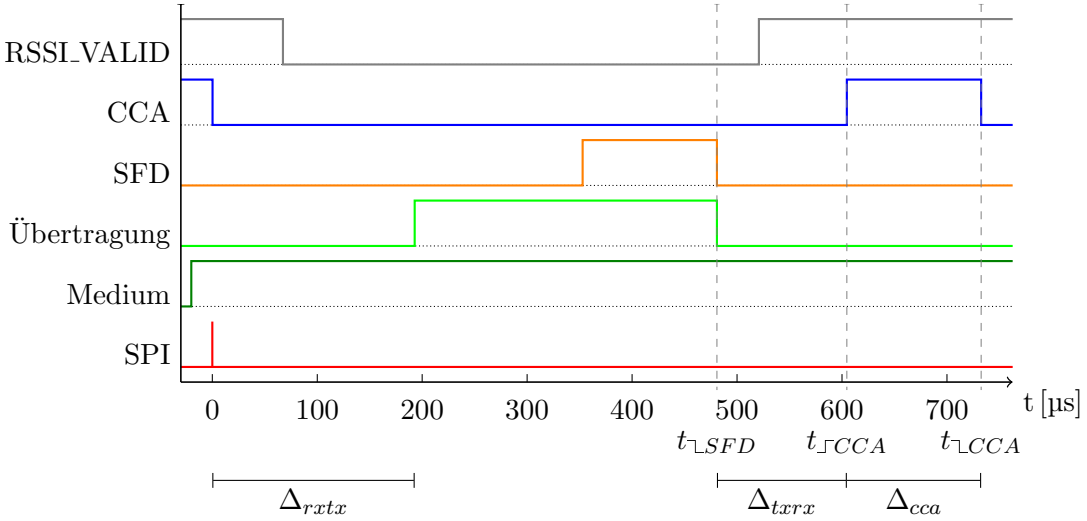


Abbildung 4.12.: Signalverlauf von `CCA` und `RSSI_VALID` bei einer Übertragung, während das Medium belegt ist

Formel berechnet werden kann. So gilt letztlich $\Delta_{cca} = \Delta_{\lceil \texttt{CCA} \rightarrow \lfloor \texttt{CCA}} \approx d_{maxCCA}$ und somit auch

$$\begin{aligned} \Delta_{txrx} &\overset{(4.17)}{=} \Delta_{\lfloor \texttt{SFD} \rightarrow \lfloor \texttt{CCA}} - \Delta_{\lceil \texttt{CCA} \rightarrow \lfloor \texttt{CCA}} \\ &\overset{(4.4)}{=} \Delta_{\lfloor \texttt{SFD} \rightarrow \lceil \texttt{CCA}} \end{aligned}$$

Ergebnis und Folgerungen: Tabelle 4.5 zeigt die statistische Auswertung über die gemessene Dauer $\Delta_{\lfloor \texttt{SFD} \rightarrow \lceil \texttt{CCA}}$ und $\Delta_{\lceil \texttt{CCA} \rightarrow \lfloor \texttt{CCA}}$. Es ist ersichtlich, dass beide Zeitspannen unter Beachtung der Messgenauigkeit konstant ist. Ferner ist ersichtlich, dass $\Delta_{\lceil \texttt{CCA} \rightarrow \lfloor \texttt{CCA}} \approx d_{maxCCA}$ gilt und somit, wie angestrebt, das gesamte Integrationsfenster des CCA-Mechanismus benötigt wird, um die Medienbelegung zu erkennen. Daraus folgt, dass ab dem Zeitpunkt der steigenden Flanke des `CCA`-Signals, also bei $t_{\lceil \texttt{CCA}}$, wieder ein Empfang möglich ist. Die Dauer ab Ende der Übertragung bis zu diesem Zeitpunkt beträgt gerade $\Delta_{txrx} = \Delta_{\lfloor \texttt{SFD} \rightarrow \lceil \texttt{CCA}}$ und wurde auf 123,85 µs gemessen. Es wird allerdings angenommen, dass dieser Wert von Bauteiltoleranzen beeinflusst wird und somit von Transceiver zu Transceiver verschieden ausfallen kann. Es sollte daher $\Delta_{txrx} = 128\,\mu s$ statt der im Datenblatt angegebenen 12 Symboldauern als eine Obergrenze angenommen werden.

	Δ_{txrx}	Δ_{cca}	Ungenauigkeit
Minimum	123,750 µs	128,120 µs	±5 ns
Maximum	123,755 µs	128,125 µs	±5 ns
Spannweite	5 ns	5 ns	±10 ns
Mittelwert	123,75 µs	128,12 µs	
Standardabweichung	1,614 ns	2,454 ns	

Tabelle 4.5.: Statistische Auswertung von $\Delta_{\lfloor \texttt{SFD} \rightarrow \lceil \texttt{CCA}}$ und $\Delta_{\lceil \texttt{CCA} \rightarrow \lfloor \texttt{CCA}}$

Eine weitere wichtige Beobachtung betrifft das Verhalten des `RSSI_VALID`-Signals. Laut Datenblatt wird dieses Signal aktiv, wenn der Empfänger lange genug (genauer: über mindestens acht Symbolperioden) aktiviert war. Das Signal zeigt an, dass der aus der Hardware auslesbare RSSI-Wert (*Received Signal Strength Indication*) gültig ist. Dieser Wert gilt als Hinweis auf die Signalstärke des empfangenen Signals.

Das `RSSI_VALID`-Signal führt jedoch trotz der Verzögerung in dieser Messung bereits einen logischen 1-Pegel, bevor die Umschaltung abgeschlossen ist. Daher muss davon ausgegangen werden, dass das `RSSI_VALID`-Signal nicht dem beschriebenen Verhalten des Datenblatts folgt. Wenn der RSSI-Wert gebraucht wird, sollte als Lösung daher sichergestellt sein, dass die letzte Übertragung mindestens $\Delta_{txrx} + 8 \cdot d_{sym} = 256\,\mu s$ zurückliegt, da dann der Empfänger für mindestens acht Symbolperioden aktiviert war.

4.4.4 Umschaltung TX → TX

Nachdem eine Übertragung durch den CC 2420 abgeschlossen ist, beginnt dieser selbstständig mit der Umschaltung vom Sende- in den Empfangszustand. Im Folgenden wird untersucht, ob diese Umschaltung vollständig durchgeführt werden muss oder vorzeitig eine zweite Übertragung beginnen kann. Wenn die Umschaltung in den Empfangsmodus abgebrochen werden kann, kann Δ_{txrx} in Formeln zur Berechnung der senderseitigen Übertragungsdauer vernachlässigt werden.

Ein Beispiel hierfür ist das Master-basierte BBS-Verfahren. In [23] wurde zur Ermöglichung von konsekutiven dominanten Black Bursts bei der Bestimmung der Bitdauer d_{Bit} die Umschaltdauer vom Sende- in den Empfangszustand mitberücksichtigt. Falls die folgende Messung zeigt, dass diese Umschaltdauer nicht notwendig ist, kann d_{Bit} diesbezüglich optimiert werden.

Aufbau und Durchführung: Abbildung 4.13 zeigt den Messaufbau für dieses Experiment. Im Unterschied zum allgemeinen Messaufbau, der in den vorhergegangen Experimenten verwendet wurde, kommt hier noch ein zweiter Imote 2 im Empfangsmodus hinzu. Die beiden Signalleitungen, mit denen dieser Imote 2 am Logic Analyzer angeschlossen ist, geben beim Empfang eines IEEE 802.15.4-Rahmens das Ereignis des Rahmenempfangs sowie den Erfolg (Rahmen richtig empfangen bzw. Rahmen korrupt) an.

Um festzustellen, ob die zweite Übertragung korrekt durchgeführt wird, stößt ein Imote 2 zwei konsekutive Übertragungen so zeitnah wie möglich an. Ein zweiter Imote 2 ist dauerhaft im Empfangsmodus und gibt an, ob die Rahmen korrekt als IEEE 802.15.4-Rahmen empfangen wurden.

Zusätzlich wird mit dem USRP 2 die Leistung auf dem Medium überwacht, um einen eventuellen Leistungsunterschied festzustellen.

Ergebnis: Abbildung 4.14 stellt den zeitlichen Verlauf der Signale und der erkannten Leistung beispielhaft für zwei konsekutive Übertragungen dar. Das Ende der ersten Übertragung wird über die fallende Flanke des `SFD`-Signals ausgegeben. Das Programm, das auf dem Prozessor des Imote 2 ausgeführt wird, erkennt dies und beginnt unmittelbar eine weitere Übertragung, indem es nochmals den `STXON`-Befehl zum Transceiver gibt. Dabei kommt eine kurze Verzögerungszeit von $\Delta_{proc} \approx 13\,\mu s$ zustande.

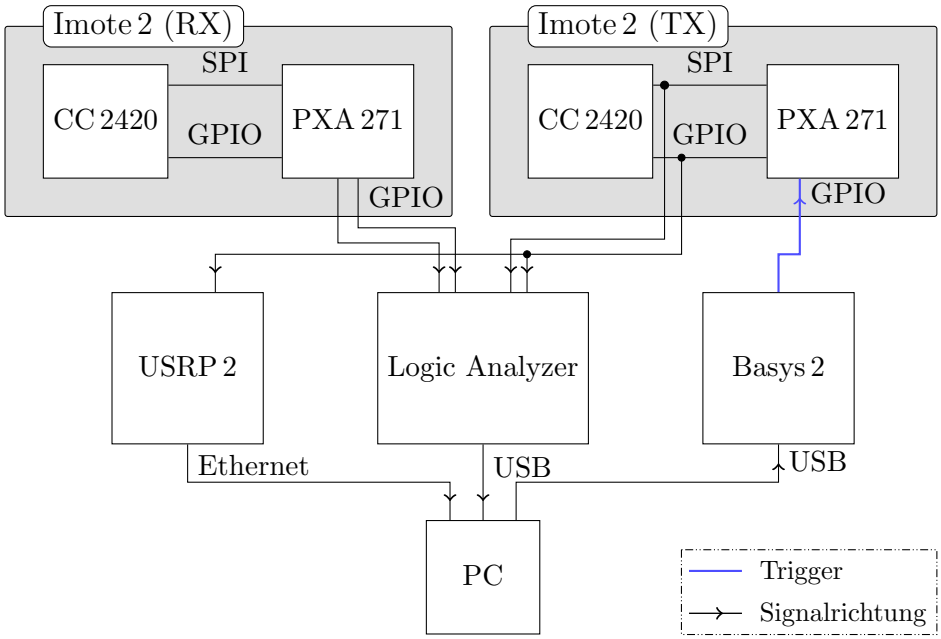


Abbildung 4.13.: Schematischer Messaufbau zur Prüfung zweier konsekutiver Sendevorgänge

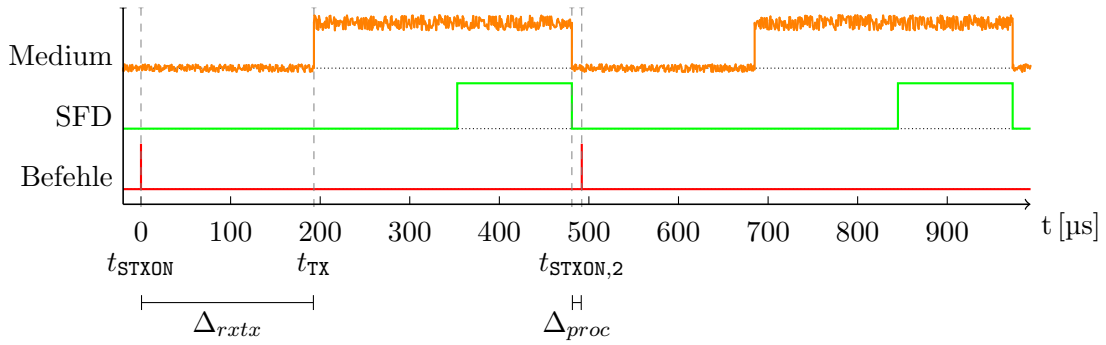


Abbildung 4.14.: Zeitlicher Verlauf der Signale des Senders und empfangene Leistung bei zwei konsekutiven Übertragungen

Aus der Abbildung ist ersichtlich, dass die zweite Übertragung nicht weiter (durch den Transceiver) verzögert wird. Weiter ist auch kein Abfall der Leistung erkennbar. Darüber hinaus wurden alle gesendeten Rahmen vom zweiten Imote 2 korrekt empfangen. All diese Beobachtungen weisen darauf hin, dass die Umschaltung vom Sende- in den Empfangsmodus nicht abgeschlossen werden muss und eine weitere Übertragung ohne Pause direkt folgen kann.

4.4.5 CCA-Verzögerung

Die Erkennung der Medienbelegung mittels CCA basiert auf der Integration der empfangenen Signalleistung über einen Zeitraum von $d_{maxCCA} = 128\,\mu s$. Dieser Wert gibt auch eine Obergrenze für die Verzögerung der Erkennung an. In allen Protokollen, deren Funktion vom CCA-Mechanismus abhängt, insbesondere also bei Black Burst-basierten Protokollen, ist diese obere Schranke der Erkennungsverzögerung von Bedeutung.

In diesem Experiment sollen mehrere Aspekte der CCA-Verzögerung betrachtet werden. Zunächst wird untersucht, wie sich die Entfernung der Knoten auf die CCA-Verzögerung auswirkt. Es ist zu erwarten, dass die Verzögerung mit größerer Entfernung steigt. In diesem Experiment sollen des Weiteren weitere Abhängigkeiten gefunden werden, z. B. ob die Verzögerung zu Beginn der Medienbelegung mit der Verzögerung am Ende der Belegung korreliert. Weiter wird untersucht, ob die gemessenen CCA-Verzögerungen immer unter der oberen Schranke von d_{maxCCA} bleiben.

Aufbau und Durchführung: Um die CCA-Verzögerung zu messen, sendet ein Imote 2 Rahmen, während ein zweiter Imote 2 diese empfängt. Es können dabei verschiedene Größen gemessen werden: Sei $t_{\texttt{TX}}$ der Zeitpunkt, zu dem die Übertragung beginnt, und $t_{\texttt{TXEND}}$ der Zeitpunkt des Übertragungsendes. Zusammen mit den Flanken des `CCA`-Signals des Empfängers können folgende Messwerte abgeleitet werden:

- $\Delta_{\texttt{TX}\rightarrow\texttt{⌞CCA}}$ ist die CCA-Verzögerung am Anfang der Übertragung, d. h. die Dauer, bis der Empfänger das belegte Medium erkennt
- $\Delta_{\texttt{⌞CCA}\rightarrow\texttt{⌟CCA}}$ ist die Gesamtdauer der erkannte Medienbelegung
- $\Delta_{\texttt{TXEND}\rightarrow\texttt{⌟CCA}}$ ist die CCA-Verzögerung am Ende der Übertragung, d. h. die Dauer, um das Medium wieder als frei zu erkennen

Die nominale Dauer des versendeten Rahmens entspricht $d_{\texttt{TX}\rightarrow\texttt{TXEND}}$. Zwischen den Größen besteht folgender Zusammenhang:

$$\begin{aligned} & d_{\texttt{TX}\rightarrow\texttt{TXEND}} - \Delta_{\texttt{TX}\rightarrow\texttt{⌞CCA}} + \Delta_{\texttt{TXEND}\rightarrow\texttt{⌟CCA}} \\ \overset{(4.5)}{=} & \Delta_{\texttt{⌞CCA}\rightarrow\texttt{TXEND}} + \Delta_{\texttt{TXEND}\rightarrow\texttt{⌟CCA}} \\ \overset{(4.2)}{=} & \Delta_{\texttt{⌞CCA}\rightarrow\texttt{⌟CCA}} \end{aligned}$$

Abbildung 4.15 zeigt den Messaufbau des Experiments. Der sendende Imote 2 (TX) beginnt durch das Triggersignal des Basys 2 eine Übertragung. Vom empfangenden Imote 2 (RX) werden das `SFD`- und das `CCA`-Signal abgegriffen und mit Hilfe des Logic Analyzers aufgezeichnet.

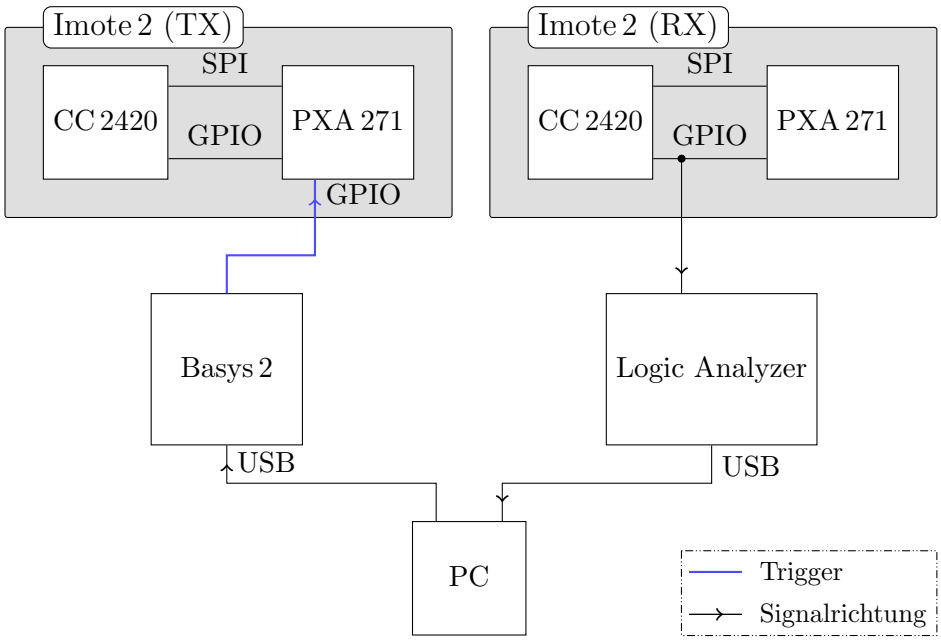


Abbildung 4.15.: Schematischer Messaufbau zur Messung der CCA-Verzögerung

Der Beginn der Übertragung kann aus dem Zeitpunkt der steigenden Flanke des SFD-Signals und der Länge des SyncHeaders zurückgerechnet werden. Allerdings wird in diesem Versuch nicht das SFD-Signal des Senders sondern des Empfängers verwendet. Dieses hat lt. Datenblatt eine Verzögerung von bis zu $d_{maxSFD} = 3\,\mu s$. Die Messung wird also durch diese Ungenauigkeit beeinflusst.

Die Rahmen, die versandt werden, besitzen eine nominale Dauer von $d_{\texttt{TX}\rightarrow\texttt{TXEND}} = 288\,\mu s$. Die über den CCA-Mechanismus erkannte Dauer sollte sich also im Rahmen $d_{\texttt{TX}\rightarrow\texttt{TXEND}} \pm d_{maxCCA}$ also von 160 µs bis 416 µs bewegen.

Ergebnis und Folgerungen: Abbildung 4.16 zeigt jeweils Minimum, Mittelwert und Maximum der gemessenen Werte in Abhängigkeit von der Entfernung. Es sind folgende Beobachtungen möglich:

- Die CCA-Verzögerung am Anfang bzw. Ende der Medienbelegung (im Bild rot bzw. blau dargestellt) liegen alle unter dem Maximum von $d_{maxCCA} = 128\,\mu s$. Der größte gemessene Wert bei einer Entfernung von 15 m beträgt $\Delta_{\texttt{TX}\rightarrow\llcorner\texttt{CCA}} = 121,5\,\mu s$. Selbst wenn zusätzlich noch die Ungenauigkeit durch die SFD-Verzögerung von 3 µs miteinbezogen wird, bleibt der Wert unter d_{maxCCA}. Der kleinste gemessene Wert, ebenfalls bei einer Entfernung von 15 m, beträgt $\Delta_{\texttt{TXEND}\rightarrow\lrcorner\texttt{CCA}} = 24,87\,\mu s$.

- Die Entfernung der Knoten hat entgegen der Erwartung keinen Einfluss auf die Verzögerungen. Dies ist jedoch darauf zurückzuführen, dass die Messung in einem geschlossenen Gebäude durchgeführt wurde, so dass auch viele physikalische Phänomene wie Mehrfachausbreitung eine Rolle spielen.

- Die erkannte Medienbelegung $\Delta_{\llcorner\texttt{CCA}\rightarrow\lrcorner\texttt{CCA}}$ liegt immer im erwarteten Bereich von $288\,\mu s \pm d_{maxCCA}$. Wenige Messreihen weisen sogar einen Mittelwert von etwa 288 µs

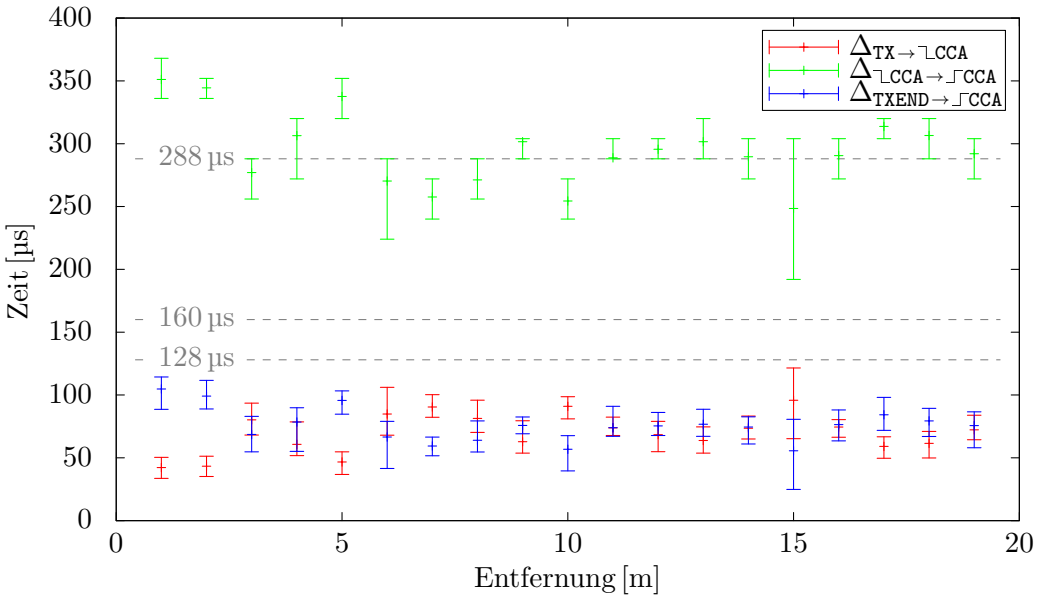


Abbildung 4.16.: Minimum, Durchschnitt und Maximum der CCA-Verzögerungen in Abhängigkeit der Distanz

auf. Bei vielen weiteren liegt 288 µs zumindest zwischen Minimum und Maximum der gemessenen Medienbelegung.

- Im Mittel gilt, dass eine längere Verzögerung zu Beginn der Belegungserkennung zu einer kürzeren Verzögerung zum Ende der Erkennung führt. Dies ist auf das Integrationsfenster zurückzuführen, das beim CCA-Mechanismus verwendet wird. Eine insgesamt geringe Signalleistung führt dazu, dass mehr Symbole gebraucht werden, so dass deren Gesamtenergie den CCA-Schwellwert überschreitet. Gleichsam wird der Schwellwert auch wieder früh unterschritten, nachdem die Übertragung abgeschlossen ist.

  Hierzu wurde der Korrelationskoeffizient ϱ über die Mittelwerte der beiden Verzögerungen (im Folgenden mit $\overline{\Delta_{\mathtt{TX}\rightarrow\lrcorner\mathtt{CCA}}}$ bzw. $\overline{\Delta_{\mathtt{TXEND}\rightarrow\lrcorner\mathtt{CCA}}}$ bezeichnet) gebildet. Der Korrelationskoeffizient wurde mit $\varrho(\overline{\Delta_{\mathtt{TX}\rightarrow\lrcorner\mathtt{CCA}}}, \overline{\Delta_{\mathtt{TXEND}\rightarrow\lrcorner\mathtt{CCA}}}) = -0,972$ berechnet. Da dieser Wert nah an -1 ist, weißt dies auf eine hohe Korrelation hin.

  Dies war für einzelne Messungen, d. h. ohne Bildung des Mittelwerts, nicht nachvollziehbar. Daher ist es nicht möglich, diese Erkenntnis für Optimierungen miteinzubeziehen, um etwa anhand der Belegungsdauer die Anfangsverzögerung näher einzuschränken.

4.4.6 Signalleistung bei Variation der Senderanzahl

Ein Nachteil Black Burst-basierter Kommunikation ist, dass Black Bursts über den Kommunikationsbereich hinaus erkennbar sind. Bei Zugriffsverfahren, die sowohl Black Bursts als auch reguläre IEEE 802.15.4-Rahmen zur Kommunikation einsetzen, haben Black Burst-basierte Übertragungen somit eine höhere Reichweite. Insbesondere bei BBQRT [6], einem Protokoll zur Routenfindung, kann dieser Aspekt zu ungültigen Routen führen.

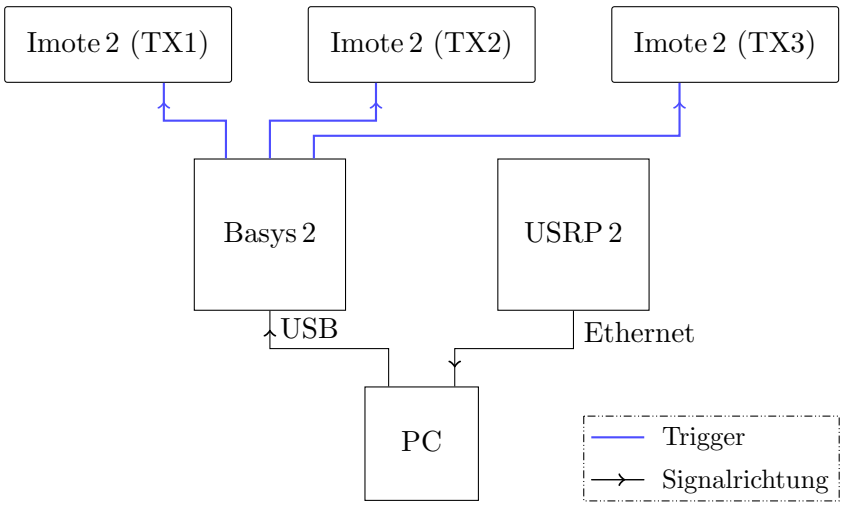


Abbildung 4.17.: Schematischer Versuchsaufbau zur Bestimmung der Signalleistung bei mehreren Sendern

Senderkonfiguration	SNR
TX1	21,24 dB
TX2	25,54 dB
TX1, TX2	26,94 dB
TX1, TX2, TX3	27,90 dB

Tabelle 4.6.: Signal-Rausch-Verhältnis bei variierender Anzahl von Sendern

In diesen Fällen wäre es hilfreich, die Signalstärke beim Senden so einzustellen, dass nur Knoten in Kommunikationsreichweite den Black Burst wahrnehmen können. Allerdings ist anzunehmen, dass die empfangene Signalstärke steigt, wenn der Black Burst von mehreren Sendern zeitgleich versandt wird. Diese Annahme wird im Rahmen der gegeben Messmöglichkeiten in diesem Experiment weiter untersucht.

Aufbau und Durchführung: Zunächst wird in einer Referenzmessung die Rauschleistung bestimmt. Weitere Leistungsmessungen werden später auf diesen Referenzwert bezogen. Als nächstes wird von bis zu drei Imote 2 eine Medienbelegung erzeugt. Die Anzahl der sendenden Imote 2 wird schrittweise erhöht. Die jeweils entstehende Leistung wird gemessen und auf den Rauschpegel bezogen. Es wird also jeweils das Signal-Rausch-Verhältnis (SNR – *Signal to Noise Ratio*) bestimmt.

Abbildung 4.17 zeigt den schematischen Versuchsaufbau zur Bestimmung der Rausch- und Signalleistung. Die drei Imote 2 werden über das Basys 2 angesteuert und haben je ein eigenes Triggersignal. So können sie unabhängig voneinander eine Übertragung beginnen. Das USRP 2 wird zum Messen der Signalleistung eingesetzt.

Ergebnis und Folgerungen: In Abb. 4.18 ist das Signal-Rausch-Verhältnis in Abhängigkeit der Senderkonfigurationen dargestellt. Zusätzlich sind die Werte der durchschnittlichen Leistungen in Tab. 4.6 aufgelistet.

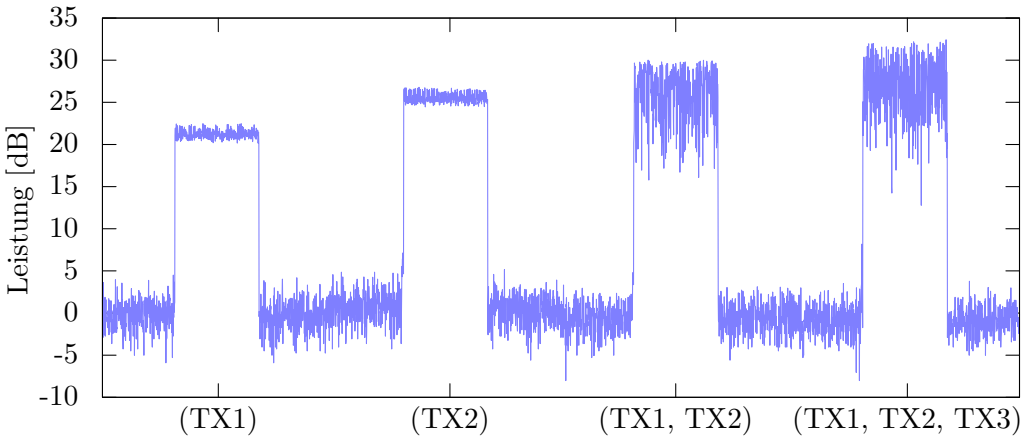


Abbildung 4.18.: Signal-Rausch-Verhältnis bei variierender Anzahl von Sendern

Zunächst fällt dabei auf, dass das Signal-Rausch-Verhältnis bereits bei einzelnen Sendern unterschiedlich ist. In den ersten beiden Senderkonfigurationen sendet Sender 1 bzw. Sender 2 alleine. Der Abstand zum USRP 2 ist in etwa gleich. Dennoch ist die empfangene Leistung von Sender 2 etwa 4,3 dB höher als die Leistung von Sender 1. Diese Beobachtung ist vermutlich auf Bauteiltoleranzen der Transceiver zurückzuführen. Die Abweichung könnte durch eine Anpassung der Sendeleistung korrigiert werden.

In den beiden weiteren Teilmessungen senden beide bzw. alle drei Sender gleichzeitig. Gegenüber den Einzelmessungen ist, wie erwartet, ein Anstieg im Signal-Rausch-Verhältnis zu erkennen, die Signalleistung steigt also mit der Anzahl an Sendern.

Somit muss auch angenommen werden, dass bei Black Burst-basierter Kommunikation durch Überlagerung mehrerer Sender auch Knoten außerhalb der Sensingreichweite erreicht werden können.

4.4.7 Verhalten des Transceivers bei ungültigen Rahmen

Der IEEE 802.15.4-Standard sieht im Kopf des Rahmens ein Längenfeld vor (sihe auch Kap. 3.2.1.5). Der Standard erlaubt Rahmen ab einer Länge von fünf Byte. Ungeachtet dessen erlaubt der CC 2420 auch Rahmen kürzerer Länge. Im Sendemodus des Transceivers wird dieser Wert verwendet, um dem Transceiver die Länge und somit das Ende des Rahmens mitzuteilen. Beim Empfang eines Rahmens schließt der Transceiver durch den Wert des Feldes auf den Endzeitpunkt der Übertragung und generiert einen entsprechend langen Rahmen.

Da kürzere Rahmen für die Implementierung Black Burst-basierter Protokolle von Vorteil sind (siehe Kap. 3.2.2.1), wird in diesem Experiment geprüft, ob auch Rahmen mit einer PHY-Nutzdatenlänge von 0 Byte verwendet werden können. Theoretisch betrachtet hat ein Rahmen mit verkürzter Präambel und 0 Byte Nutzdaten eine Dauer von 128 µs. Daher wäre es bei dieser Konfiguration möglich, die Medienbelegung durch den CCA-Mechanismus noch zuverlässig zu erkennen.

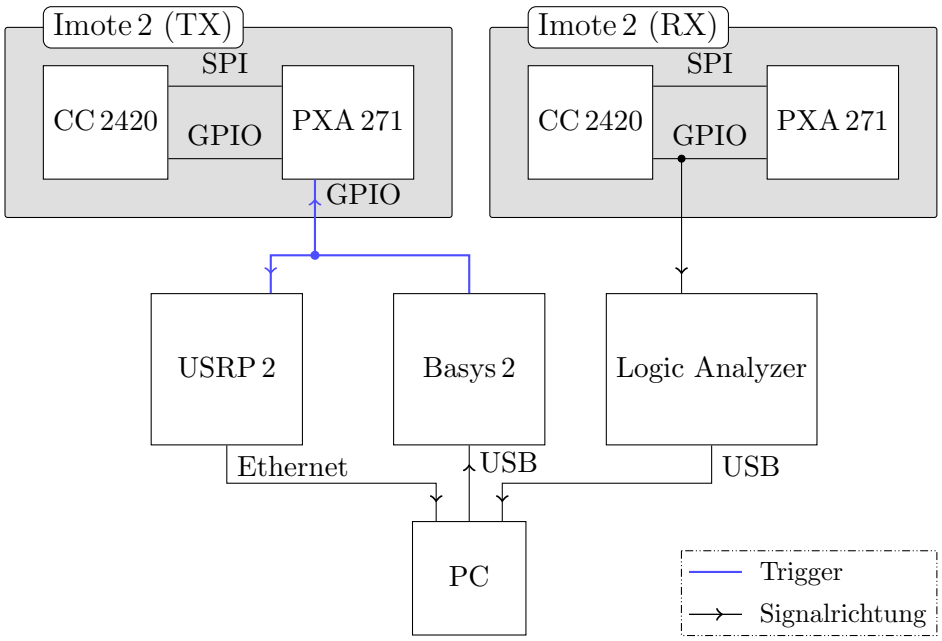


Abbildung 4.19.: Schematischer Messaufbau zur Messung der erkannten Medienbelegung bei variierender Rahmenlänge

Aufbau und Durchführung: Zur Überprüfung des Verhaltens werden durch einen Imote 2 Rahmen mit 1 Byte, später mit 0 Byte Nutzdaten, versandt. Es können mehrere Zeitspannen angegeben bzw. gemessen werden, die alle die Dauer der Medienbelegung beschreiben:

- d_{frame} beschreibt die nominale Dauer des Rahmens, die aus der Länge des Rahmens und der Symboldauer berechnet wird. Sie beträgt zunächst 224 µs, später 192 µs.
- $\Delta_{frame,cca}$ gibt die Dauer der Medienbelegung aus Sicht des Empfängers unter Verwendung des CCA-Mechanismus an.
- $\Delta_{frame,sfd}$ ist die Dauer der Übertragung, die aus dem empfängerseitigen `SFD`-Signal und der Länge des SyncHeaders zurückgerechnet wird.
- $\Delta_{frame,pwr}$ ist die Dauer der Medienbelegung, wie sie durch das USRP 2 wahrgenommen wird.

Zunächst werden Rahmen mit einer Nutzdatenlänge von 1 Byte versandt. Dadurch werden Referenzwerte für die eigentliche Messung gesammelt. Danach wird die Nutzdatenlänge auf 0 Byte reduziert und die Messung wiederholt. Die Nutzdatenlänge wird im Folgenden mit n_b bezeichnet.

In Abb. 4.19 ist der schematische Messaufbau gezeigt. Das Trigger-Signal des Basys 2 gibt dem sendenden Imote 2 (TX) den Befehl zur Übertragung eines Rahmens. Auf Seite des empfangenden Imote 2 (RX) werden das `CCA`- und `SFD`-Signal (als GPIO eingezeichnet) mit Hilfe des Logic Analyzers mitgeschnitten. Zusätzlich wird mit dem USRP 2 die Medienbelegung beobachtet.

n_b	d_{frame}	$\Delta_{frame,cca}$	$\Delta_{frame,sfd}$	$\Delta_{frame,pwr}$
1	224 µs	304 . . . 336 µs	$\sim 224,62$ µs	~ 226 µs
0	192 µs	228 . . . 3840 µs	192,8 . . . 3777 µs	~ 176 µs

Tabelle 4.7.: Auswertung der erkannten Medienbelegung

Aus den mitgeschnittenen Daten werden die beobachteten Zeitspannen wie folgt berechnet:

$$\Delta_{frame,cca} = \Delta_{\lrcorner\texttt{CCA}\rightarrow\lrcorner\texttt{CCA}}$$
$$\Delta_{frame,sfd} = \Delta_{\lrcorner\texttt{SFD}\rightarrow\lrcorner\texttt{LSFD}} + d_{syncHdr}$$
$$\Delta_{frame,pwr} = \Delta_{\lrcorner\texttt{PWR}\rightarrow\lrcorner\texttt{LPWR}}$$

Dabei wird das PWR-Signal aus den mitgeschnitten Rohdaten des USRP 2 gewonnen. Der SyncHeader der gesendeten Rahmen ist in diesem Experiment auf eine feste Länge von 5 Byte gesetzt und hat somit eine Dauer von $d_{syncHdr} = 160$ µs.

Ergebnis: Tabelle 4.7 zeigt die Dauer der erkannten Medienbelegungen für beide Teilexperimente. Bei einer Nutzdatenlänge von einem Byte ist die nominale Dauer der Übertragung $d_{frame} = 224$ µs. Dieser Wert entspricht auch mit einer nur geringen Abweichung $\Delta_{frame,sfd}$. Auch die Bestimmung der Dauer der Medienbelegung mit Hilfe des USRP 2 zeigt ähnlich genaue Ergebnisse. Die Erkennung durch den CCA-Mechanismus liegt zwar darüber und unterliegt auch größeren Schwankungen, bleibt aber im Rahmen der erwarteten Ungenauigkeit von maximal d_{maxCCA}.

Wird die Nutzdatenlänge auf 0 Byte reduziert, sinkt entsprechend die nominale Dauer der Übertragung auf 192 µs. Aus der Auswertung sind jedoch mehrere Phänomene ersichtlich, die auf ein Fehlverhalten der Hardware sowohl auf Sender- als auch Empfängerseite hinweisen.

- Über das USRP 2 wird eine Belegungsdauer von $\Delta_{frame,pwr} = 176$ µs erkannt. Diese ist kürzer als die nominale Dauer der Übertragung. Das weist darauf hin, dass der Sender seine Übertragung abrupt abbricht.

- Die Belegungsdauer $\Delta_{frame,sfd}$, die durch den SFD-Mechanismus erkannt wird, nimmt zufällig erscheinende Werte zwischen 192,8 µs und 3337 µs an. Die tatsächliche Medienbelegungsdauer ist aus diesen Werten nicht mehr erkennbar.

  Es ist anzunehmen, dass der Empfänger nach der abgebrochenen Übertragung weiterhin versucht, das empfangene Signal zu dekodieren. Dieses besteht nach dem Abbruch nur noch aus Rauschen, das nun interpretiert wird. Da gerade das Längenfeld durch den Abbruch verfälscht wird, und die Logik des Empfängers aus diesem Feld auf die Dauer der Übertragung rückschließt, versucht der Empfänger je nach erkanntem Wert des Längenfelds weiterhin das Funksignal zu dekodieren. Die zufällig erscheinenden Werte sind auf das zufällige Rauschen zurückzuführen, das letztlich für die Rahmenlänge dekodiert wird.

- Über den CCA-Mechanismus werden ähnliche Werte wie über den SFD-Mechanismus angenommen. Es ist daher zu vermuten, dass der CCA-Mechanismus, wenn ein Rahmenempfang eingeleitet wurde, an diesen Empfangsvorgang gekoppelt wird. Das

CCA-Signal zeigt bei einem Rahmenempfang also nicht mehr den tatsächlichen Zustand des Mediums auf Grundlage der erkannten Leistung an, sondern bleibt solang auf dem 0-Pegel, bis die Übertragung abgeschlossen ist. Im korrekten Betrieb ist dieses Verhalten akzeptabel. Wird jedoch das Längenfeld eines Rahmens falsch erkannt, ist die Dauer der tatsächlichen Medienbelegung nicht mehr dem Wert $\Delta_{frame,cca}$ zu entnehmen.

Daraus lassen sich zwei Folgerungen festhalten:

- Es dürfen keine Rahmen mit einer Nutzdatenlänge von 0 Byte versandt werden, da der Transceiver dabei seine Übertragung abbricht und die Medienbelegung kürzer ist als das Integrationsfenster des CCA-Mechanismus. Folglich könnte die dadurch erzeugte Medienbelegung unter Umständen nicht mehr erkannt werden.

- Falls ein dominanter Black Burst durch das Senden eines IEEE 802.15.4-Rahmens realisiert ist, ist es ratsam, den korrekten Empfang der versendeten Rahmen zu unterbinden. Es ist anzunehmen, dass die Überlagerung von Black Bursts (also folglich die Überlagerung von IEEE 802.15.4-Rahmen) ebenfalls ein korruptes Längenfeld verursachen kann. In diesem Fall kann folglich nicht mehr die Länge des Black Bursts bestimmt werden.

  Um den Rahmenempfang zu unterbinden, verwendet zunächst jeder Transceiver sein eigenes SyncWord (Kap. 3.2.1.5). So ist sichergestellt, dass sich ein Empfänger nicht mit einem Sender synchronisiert. Zusätzlich wird der Schwellwert für die SFD-Korrelation (Kap. 3.2.1.4) auf den maximalen Wert gesetzt. Dies verringert auch die Wahrscheinlichkeit, dass zufälliges Rauschen als SyncHeader erkannt wird.

4.5 Experimente und Optimierungen zum Prozessor

In Kap. 3.3 wurde der Prozessor des Imote 2 vorgestellt. Insbesondere wurde dabei auf den Aufbau der Kernhardware und hardwareseitige Optimierungen wie Caches, TLBs, Schreibpuffer, etc. eingegangen. Die komplexe Arbeitsweise des Prozessors und die nur schwer kontrollierbaren Optimierungen führen in ihrer Gesamtheit dazu, dass die Dauer, die zum Ausführen eines Codestücks benötigt wird, praktisch nicht mehr vorhersagbar ist. Darüber hinaus können sich wenige Änderungen am Programmcode erheblich auf die Ausführungsdauer auswirken. Des Weiteren können viele Codeteile bei mehrfacher Ausführung eine variable Schwankung aufweisen, die z. B. durch unterschiedliche Cacheinhalte entstehen.

In den folgenden Experimenten werden Messungen zur Ausführungsgeschwindigkeit von bestimmten Codeteilen durchgeführt. Ziel ist dabei nicht die genaue Ausarbeitung eines Modells zur Abschätzung von Ausführungsdauern. Es sollen stattdessen einige Aspekte herausgegriffen und die Größenordnungen der involvierten Verzögerungen ermittelt werden.

Von Interesse sind dabei Speicherzugriffe, insbesondere jene, die über den Peripheriebus mit integrierter Peripherie kommunizieren. Ferner werden Messungen zum Auslösen von Interrupts durchgeführt und verschiedene Optimierungstechniken evaluiert, die teilweise bereits in Kap. 3.3 vorgestellt wurden.

Ziel der Messungen ist das Abschätzten einer oberen Schranke für Verzögerungen, die in der Implementierung Black Burst-basierter Verfahren verwendet werden kann.

4.5.1 Messung von Buslatenzen

Der Aufbau des System- und Peripheriebus wurde in Kap. 3.3.2 bzw. Kap. 3.3.3 vorgestellt. Dabei wurde auch darauf hingewiesen, dass Zugriffe auf die Peripherie potentiell zu Latenzen führen und die Ausführung des Programms ggf. angehalten werden muss. In diesem Experiment werden die Latenzen verschiedener Zugriffe auf den System- und Peripheriebus gemessen und zueinander verglichen.

Durchführung: Der PXA 271 verfügt über vier Ereignis- und einen Zyklenzähler. Der Zyklenzähler wird bei jedem Taktzyklus inkrementiert. Die Ereigniszähler werden bei jedem Eintreten eines vorgewählten Ereignisses erhöht. Der PXA 271 bietet insgesamt 14 Ereignisquellen, von denen je eine Quelle auf einen der vier Ereigniszähler konfiguriert werden können. In diesem Experiment werden folgende drei Zähler verwendet:

- Implizit wird immer die Anzahl an Taktzyklen gezählt. Der Stand des Zählers wird im Folgenden mit n_c bezeichnet.
- Anzahl ausgeführter Instruktionen. Der Zähler wird erhöht, wenn eine Instruktion vollständig abgearbeitet ist. Der Stand dieses Ereigniszählers wird mit n_i bezeichnet.
- Anzahl ausgeführter Sprunginstruktionen. Der Zähler wird erhöht, wenn eine Sprunginstruktion ausgeführt wurde. Der Zählerstand wird mit n_b bezeichnet.

Eine zu untersuchende einzelne Instruktion wird mehrmals in einer Endlosschleife ausgeführt. Dabei wurde darauf geachtet, dass die Zielregister unterschiedlich sind, um Prozessorstalls durch Datenabhängigkeiten zu eliminieren. Weiter werden periodisch die Werte der Zähler n_c, n_i und n_b ausgelesen, verschiedene Statistiken daraus ermittelt und diese ausgegeben. Danach beginnt die Messung erneut.

Die Routine zum Auslesen der Zähler wird zwar selbst auch mitgezählt, jedoch wurde das Ausleseintervall (etwa 40 s) sehr groß im Vergleich zur Taktperiode (ungefähr 9.62 ns bei einer Taktfrequenz von 104 MHz) gewählt. Somit sind die wenigen Ereignisse, die durch das Auslesen der Zähler entstehen, gegenüber den Ereignissen, die durch die eigentlich Messung gezählt werden, vernachlässigbar.

Anhand der gezählten Ereignissen können folgende Kenngrößen bestimmt werden:

- Anzahl an Zyklen pro Instruktion (*CPI – Cycles Per Instruction*):

$$n_{CPI} = \frac{n_c}{n_i} \tag{4.18}$$

- Anzahl an Zyklen pro Nicht-Sprung:

$$n_{CPI,nb} = \frac{n_c - n_b}{n_i - n_b} \tag{4.19}$$

  Hierbei gilt, dass ein (unbedingter) Sprung in einem Zyklus ausgeführt wird und daher der Teilterm $n_c - n_b$ die Anzahl an Zyklen ohne Sprünge angibt.

Hierbei ist $n_{CPI,nb}$ die relevantere Größe, da die Schleife (und der damit verbundene Sprung) nur der Messwiederholung dient. Der Wert von n_{CPI} gibt hingegen einen Durchschnittswert für die zu messende Operation und der Schleife an. Bei $n_{CPI,nb}$ wird der Fehler, der durch die Schleife entsteht, herausgerechnet.

Die Anzahl an Zyklen kann mit Hilfe der Taktfrequenz f in eine Zeitspanne d_I bzw. $d_{I,nb}$ umgerechnet werden:

$$d_I = \frac{n_{CPI}}{f} \qquad d_{I,nb} = \frac{n_{CPI,nb}}{f} \tag{4.20}$$

Untersuchte Operationen: Wie bereits in der Einführung erwähnt, werden in dieser Messung die Laufzeiten verschiedener Operationen untersucht. So können die jeweiligen Größenordnungen, vor allem auch untereinander, besser verglichen und obere Schranken abgeschätzt werden.

Hierbei ist u. A. von Interesse, wie sich ein Zugriff auf ein Datum aus dem Cache gegenüber einem Zugriff eines Datums aus dem SRAM verhält. Des Weiteren sollen diesen Zugriffen auch Lese- und Schreibzugriffe auf Peripherieregister gegenübergestellt werden.

Die fünf untersuchten Operationen sind:

1. *R(Cache)*: Lesezugriff auf ein Datum, welches im Datencache vorgehalten wird
2. *R(SRAM)*: Lesezugriff auf ein Datum, welches nicht im Datencache vorgehalten wird und aus dem SRAM über den Systembus geladen wird
3. *R(Timer)*: Lesezugriff auf ein Timer-Register über den Peripheriebus
4. *R(GPIO)*: Lesezugriff auf ein GPIO-Register über den Peripheriebus
5. *W(GPIO)*: Schreibzugriff auf ein GPIO-Register über den Peripheriebus

Alle fünf Operationen werden jeweils für die Taktfrequenzen 104 MHz, 208 MHz und 416 MHz wiederholt, um die Beeinflussung der Taktfrequenz auf die Ausführungsdauern zu untersuchen.

Ergebnis: Tabelle 4.8 zeigt die Anzahl gezählter Zyklen und die dazugehörigen Zeitspannen für jede der fünf untersuchten Operationen.

Aus den Ergebnissen ist ersichtlich, dass das Auslesen eines Datums im Cache nur einen Taktzyklus benötigt, das Auslesen aus dem SRAM dauert dagegen etwa zehn Zyklen bei Kernfrequenzen von 104 MHz und 208 MHz. Die Zugriffsdauer bei den Kernfrequenzen 208 MHz und 416 MHz liegt gleichbleibend bei ca. $d_{I,nb} = 57$ ns. Dies liegt daran, dass bei 416 MHz nur die Kernfrequenz geändert wird. Die Taktfrequenz des Systembusses ist bei beiden Kernfrequenzen 208 MHz. An dieser Stelle soll außerdem erklärt werden, wieso n_{CPI} und $n_{CPI,nb}$ in diesem Experiment jeweils den gleichen Wert haben. Dies mag zunächst als Fehler erscheinen, doch n_{CPI} ist, wie zuvor beschrieben, ein Durchschnittswert aller ausgeführten Instruktionen. Da sowohl das Auslesen des Datums als auch die Sprungoperation jeweils ein Taktzyklus benötigt, ist so auch der Durchschnittswert ein Zyklus pro Instruktion.

Weiter ist ersichtlich, dass die Zugriffsdauer auf die Peripherie um ein vielfaches höher ist als der Zugriff auf den SRAM. Das Auslesen des Timer-Registers (Experiment 3) dauert z. B. eine knappe Mikrosekunde.

Zusätzlich kann beim Vergleich von Experiment 3 und 4 beobachtet werden, dass der Zugriff auf verschiedene Peripherieregister unterschiedlich lange dauert. So ist die Dauer beim lesenden Zugriff auf ein GPIO-Register etwa 40% geringer als beim Lesen des Timer-Registers.

Operation	Kernfrequenz	n_{CPI}	d_I	$n_{CPI,nb}$	$d_{I,nb}$
R(Cache)	104 MHz	1,0	9,62 ns	1,0	9,62 ns
R(SRAM)		10,7	102,88 ns	11,9	114,42 ns
R(Timer)		91,2	876,92 ns	102,4	984,62 ns
R(GPIO)		59,2	569,23 ns	66,4	638,46 ns
W(GPIO)		33,3	320,19 ns	49,5	475,96 ns
R(Cache)	208 MHz	1,0	4,81 ns	1,0	4,81 ns
R(SRAM)		10,7	51,44 ns	11,9	57,21 ns
R(Timer)		152,0	730,77 ns	170,9	821,63 ns
R(GPIO)		88,0	423,08 ns	98,9	475,48 ns
W(GPIO)		48,0	230,77 ns	71,5	343,75 ns
R(Cache)	416 MHz	1,0	2,40 ns	1,0	2,40 ns
R(SRAM)		21,3	51,20 ns	23,9	57,45 ns
R(Timer)		298,7	718,03 ns	335,9	807,45 ns
R(GPIO)		170,7	410,34 ns	191,9	461,30 ns
W(GPIO)		108,8	261,54 ns	162,7	391,11 ns

Tabelle 4.8.: Auswertung von n_{CPI} und $n_{CPI,nb}$

Ebenso hat die Art des Zugriffs (lesend bzw. schreibend) einen großen Einfluss auf die Zugriffszeiten. Dies lässt sich beim Vergleich zwischen Experiment 4 und 5 beobachten. Der schreibende Zugriff dauert nur etwa halb so lange wie der lesende.

Folgerungen: Zunächst sollte festgehalten werden, dass die Ausführungsdauer erheblich davon beeinflusst wird, ob die Daten im Cache vorgehalten werden oder aus dem Arbeitsspeicher nachgeladen werden müssen. Es ist naheliegend, dass analoge Erkenntnisse auch für das Vorhalten von Programmcode im Instruktionscache gelten.

Des Weiteren ist der SRAM, mit dem der Test durchgeführt wurde, der schnellere Arbeitsspeicher. Es ist anzunehmen, dass Zugriffe auf SDRAM oder Flashspeicher größere Zugriffsverzögerungen beinhalten. Dies zeigt, dass das dauerhafte Vorhalten wichtiger Daten (und Routinen) im Cache vorteilhaft ist. Dieser Mechanismus wurde in Kap. 3.3.1.2 als *Cachelocking* vorgestellt.

Außerdem gibt die hohe Latenz von Zugriffen auf Peripherie Grund zur Annahme, dass mehrfaches Auslesen eines Peripherieregisters (*Polling*) in einer Schleife potentiell mit größeren Schwankungen verbunden ist als das Auslösen einer Unterbrechungsbehandlung. Dieser Aspekt wird im nächsten Abschnitt näher betrachtet.

Zuletzt sei darauf hingewiesen, dass es aufgrund der Unterschiede beim Zugriff auf verschiedene Peripherieregister nahezu unmöglich ist, eine Abschätzung der Ausführungszeit für eine Programmroutine, die viele Zugriffe auf die Peripherie durchführt, anzugeben. Daher eignet es sich, eine solche Routine während ihrer Ausführung zu analysieren (*Profiling*). Für das Profiling kann wieder auf den Zyklenzähler und ggf. die Ereigniszähler zurückgegriffen werden.

4.5.2 Optimierung der Unterbrechungsbehandlung

Beim Auslösen einer Unterbrechung arbeitet der Controller des Prozessors mehrere Aufgaben ab, die nötig sind, um nach der Behandlung wieder den ursprünglichen Prozessorzustand herstellen zu können. Allerdings benötigt jede dieser Aufgaben eine gewisse Dauer, die zum Teil auch variable Anteile beinhaltet. Letztlich führt dies dazu, dass die Dauer zwischen Auslösen einer Unterbrechung und ihrer Behandlung variiert.

Gerade diese Dauer ist jedoch bei Black Burst-basierten Protokollen von Interesse. Die Signalleitungen des CC 2420 (insbesondere `CCA` und `SFD`) sind per GPIO an den PXA 271 angebunden. Der Beginn bzw. das Ende einer erkannten Medienbelegung wird über fallende bzw. steigende Flanken des `CCA`-Signals angezeigt. Dies löst beim PXA 271 eine Unterbrechung aus. Da sich die Information eines Black Bursts auf dessen Beginn und Ende beschränkt, ist es für das Funktionieren Black Burst-basierter Protokolle essentiell, die Zeitpunkte dieser Ereignisse so genau wie möglich zu bestimmen. Um dies zu erreichen, wird zum Beginn der Unterbrechungsroutine ein Zeitstempel der Systemuhr gesichert, der anschließend weiterverarbeitet werden kann. Folglich ist es von Vorteil, die dabei involvierten Verzögerungen zu verstehen und Optimierungen auszuarbeiten, die eine bessere Kontrolle über die variablen Anteile ermöglichen.

Zunächst wird im Folgenden der Messaufbau zu den Experimenten erklärt. In den folgenden Abschnitten werden dann die Verzögerungsquellen der Unterbrechungsbehandlung analysiert und Vorschläge zur Optimierung evaluiert.

Allgemeiner Aufbau und technische Durchführung: Der Versuchsaufbau der folgenden Messungen ist in Abb. 4.20 gezeigt. Die zu messenden Zeitspannen können in den Messungen nur durch Ereignisse erfasst werden, die von außen beobachtbar sind. Das bedeutet, dass durch das Basys 2 ein Ereignis ausgelöst wird, welches beim Prozessor über den GPIO-Controller eine Unterbrechung auslöst. Durch die Behandlung dieser Unterbrechung wird vom Programm eine Flanke auf einem zweiten GPIO-Pin ausgelöst. Beide Ereignisse werden durch den Logic Analyzer aufgezeichnet.

Dabei muss beachtet werden, dass der GPIO-Controller das Eingangsereignis verarbeiten und das Ausgangsereignis generieren muss. Da es in diesem Versuch nur die reine Ausführungszeit der Unterbrechungsroutine geht, verfälschen diese Vorgänge durch ihre Verarbeitungsdauer die Messergebnisse. Allerdings ist anzunehmen, dass die Verzögerungen nahezu konstant sind, so dass verschiedene Messergebnisse relativ zueinander vergleichbar sind.

4.5.2.1 Fertigstellen einer vorherigen Unterbrechungsbehandlung

Wird zum Zeitpunkt eines Ereignisses noch eine vorherige Unterbrechung behandelt, kann sich die Behandlung der neueren Unterbrechung beliebig lange verzögern. Theoretisch wäre denkbar, dass die Behandlungen selbst unterbrechbar sind und so geschachtelte Unterbrechungen möglich sind. Dies ist jedoch mit großem Aufwand verbunden und kann sich im Bezug auf die folgenden Optimierungen auch negativ auswirken: Wenn nämlich die Behandlung unterbrochen werden kann, kann sich deren Gesamtdauer wieder um die Dauer der geschachtelten Behandlungen verlängern. Aussagen über Verzögerungen werden somit komplexer. Daher werden Unterbrechungen i. Allg. nicht geschachtelt.

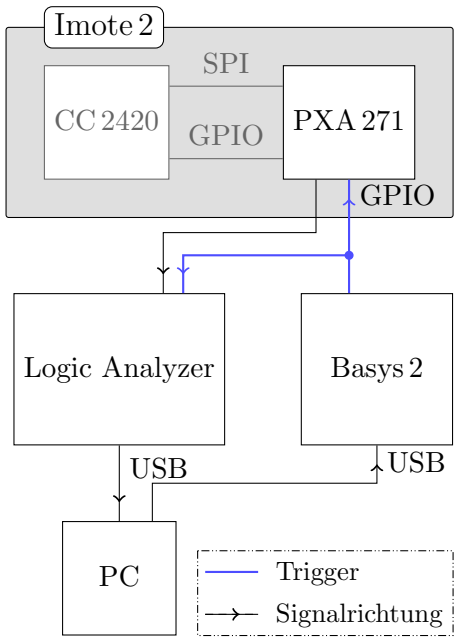


Abbildung 4.20.: Schematischer Versuchsaufbau zur Messung von Unterbrechungslatenzen

Um dennoch diese Art der Verzögerung bei der Behandlung zu reduzieren, sind Kenntnisse über alle Unterbrechungsquellen (insbesondere aus externer Beschaltung) nötig. Wenn sichergestellt werden kann, dass sich Unterbrechungen zu keinem Zeitpunkt überlagern, würde auch die beschriebene Situation nicht auftreten. Da dies eine harte und je nach System unerfüllbare Forderung ist, soll eine zweite Möglichkeit mit wenigen Anforderungen vorgestellt werden: Wenn der ungefähre Zeitraum einer (wichtigen) Unterbrechung bekannt ist, können in diesem Zeitraum alle anderen Unterbrechungsquellen maskiert werden, so dass nur noch die erwartete auftreten kann.

4.5.2.2 Fertigstellen der momentan ausgeführten Instruktion

Beim Eintritt der Unterbrechung muss zunächst eine Instruktion, die gerade in der Ausführungsphase (in ARM-Terminologie X oder D) der Prozessorpipeline ist, vollständig abgearbeitet werden. Da verschiedene Instruktionen verschieden lange Ausführungszeiten haben, kann dieser Vorgang entsprechend unterschiedlich lange dauern. Im Worst-Case wird eine Instruktion ausgeführt, die auf Speicher zugreift. Die ARMv5-Architektur bietet Lade- bzw. Speicherinstruktionen, die in einer Instruktion bis zu 16 Speicherworte zwischen Speicher und Prozessorregistern transferieren können. Wenn die Speicherworte darüber hinaus nicht im Cache vorgehalten werden, wird die Abarbeitung einer solchen Instruktion weiter verzögert.

Die Verzögerung kann nur eingeschränkt werden, wenn eine bestimmte Unterbrechung in einem bekannten Zeitraum erwartet wird. In diesem Fall kann die Programmausführung eingeschränkt werden, so dass in diesem Zeitraum keine potentiell lang andauernden Instruktionen ausgeführt werden dürfen. Praktisch gestaltet sich dies so, dass das Programm in einer Endlosschleife aktiv auf das Eintreten der Unterbrechung wartet. Für das aktive Warten (*Busy Waiting*) sind generell zwei Ansätze denkbar:

Methode	Minimum	Spannweite	Maximum	Mittelwert
Polling	1155 ns	730 ns	1885 ns	1518 ns
Spinning	890 ns	120 ns	1010 ns	960 ns

Tabelle 4.9.: Gegenüberstellung von Polling und Spinning

- Beim *Polling* werden alle Unterbrechung deaktiviert und das entsprechende Hardwareregister in einer Endlosschleife ausgelesen, bis der gelesene Wert eine bestimmte Bedingung erfüllt, die das Eintreten des Ereignisses markiert.

- Beim *Spinning* bleiben Unterbrechungen (oder zumindest die erwartete Unterbrechungsquelle) aktiviert und es wird eine ansonsten leere Endlosschleife ausgeführt. Die Behandlung der eigentlichen Unterbrechung muss dann dafür sorgen, dass dieser Zustand wieder aufgelöst wird, so dass die normale Programmausführung fortgesetzt werden kann. Die generelle Arbeitsweise läuft nach folgendem Schema ab:

  1. Zu einem zukünftigen Zeitpunkt t_{intr} wird eine Unterbrechung erwartet. Das Programm setzt einen Timer auf $t_{intr} - d_{proc}$, wobei d_{proc} eine Verarbeitungsdauer darstellt und genügend groß gewählt werden muss, um z. B. Verzögerungen durch laufende Instruktionen Rechnung zu tragen.

  2. Das Ablaufen des Timers löst eine Unterbrechung aus. Der Prozessor deaktiviert selbstständig alle Unterbrechungen, da im Regelfall nur maximal eine Unterbrechung behandelt wird. Die Behandlung des Timers ruft die *spin*-Routine auf, die Unterbrechungen wieder reaktiviert und letztlich in eine Endlosschleife mündet.

  3. Das erwartete Ereignis löst die eigentliche Unterbrechung aus. Nachdem diese behandelt ist, erkennt die Unterbrechungsroutine, dass der Spinning-Mechanismus verwendet wurde, und kehrt nicht wieder in die Endlosschleife zurück, sondern setzt die Programmausführung an der Stelle fort, wo die erste Unterbrechung durch den Timer stattfand.

Für das folgende Experiment wurden beide Ansätze implementiert, um einen gegenseitigen Vergleich zu ermöglichen.

Durchführung: In diesem Versuch wird die Reaktionszeit auf ein erwartetes Ereignis gemessen. Dieses wird zunächst über Polling erkannt und in einem zweiten Messdurchlauf über eine normale Unterbrechung unter Verwendung des Spinnings. Dabei soll nicht untersucht werden, wie lange die gesamte Behandlung dauert, sondern die Verzögerung zur Erkennung des Ereignisses. Da im normalen Betrieb diese Verzögerung zu einer verzögerten Sicherung des Zeitstempels führt, ist diese Dauer kritisch.

Um diese Dauer bei Verwendung des Spinnings zu messen, wurde die Unterbrechungsroutine auf ein Minimum reduziert: Da alle anderen Quellen maskiert sind, kann das Auftreten einer Unterbrechung nur bedeuten, dass das Eingangsereignis ausgelöst wurde. Der Code der Unterbrechungsroutine beschränkt sich daher auf das direkte Auslösen des Ausgangsereignisses. Im Normalfall müsste an dieser Codestelle die genaue Quelle der Unterbrechung bestimmt und deren spezifische Behandlungsroutine aufgerufen werden.

Sperrung	Minimum	Spannweite	Maximum	Mittelwert
	12180 ns	735 ns	12915 ns	12518 ns
✓	8695 ns	650 ns	9345 ns	9033 ns

Tabelle 4.10.: Auswirkungen der Verdrängungssperrung auf die Reaktionsdauer

Ergebnis und Folgerungen: Tabelle 4.9 gibt die Gegenüberstellung von Polling und Spinning an. Es ist erkennbar, dass das Spinning insgesamt eine geringere Zeit für die Erkennung des Eingangsereignisses benötigt als das Polling. Insbesondere sinkt die Spannweite auf ein Sechstel des Wertes.

Beim Busy Waiting auf externe Ereignisse ist folglich das Spinning dem Polling überlegen.

4.5.2.3 Verzögerung zu Beginn der Unterbrechungsroutine

Bei der Unterbrechung geht als nächstes der Prozessor in den Unterbrechungsmodus über. Dabei wird das Kontrollregister des Prozessors und die Rücksprungadresse gesichert. Ferner werden Unterbrechungen deaktiviert und schließlich eine fest definierte Adresse angesprungen, an der die Unterbrechungsroutine erwartet wird. Diese Vorgänge benötigen wenige Taktzyklen und es kann davon ausgegangen werden, dass die Dauer konstant ist.

Danach wird die Ausführung an der Unterbrechungsroutine fortgesetzt. Hier is nun entscheidend, ob die Instruktionen der Routine im Instruktionscache vorliegen oder erst vom Hintergrundspeicher geladen werden müssen. Nach den Erkenntnissen des vorherigen Abschnitts ist die Dauer zum Laden von Daten bzw. Instruktionen aus dem Hintergrundspeicher um das etwa zehnfach größer als das Laden der Daten aus dem Cache. Da die Ersetzungsstrategie der Caches (Kap. 3.3.1.2) potentiell auch die Instruktionen der Unterbrechungsroutine verdrängt, muss an dieser Stelle mit einer variablen Verzögerung gerechnet werden.

Zur Optimierung werden relevante Einträge im Cache gesperrt (Kap. 3.3.1.2). Somit werden die Instruktionen und Daten der Unterbrechungsroutine immer im Cache vorgehalten und müssen nie vom Hintergrundspeicher nachgeladen werden. In der folgenden Messung wird der Einsatz des Sperrmechanismus quantifiziert.

Durchführung: Das Programm, das auf dem Prozessor ausgeführt wird, simuliert eine Anwendung, welche eine große Buslast erzeugt. Das bedeutet zum Einen, dass viele Daten geschrieben werden, so dass alle vorherigen Einträge im Datencache verdrängt werden und darüber hinaus alle neuen Einträge bei einer weiteren Verdrängung erst noch zurückgeschrieben werden müssen (Kap. 3.3.1.2, „write-back Schreibstrategie“). Zum Anderen wird ein großer Codebereich ausgeführt, so dass auch die Einträge des Instruktionscaches verdrängt werden.

Im Folgenden wird das Programm wird mit und ohne Verdrängungssperrung ausgeführt, die Reaktionsdauer auf ein GPIO-Ereignis gemessen und anschließend verglichen.

Ergebnis: In Tab. 4.10 ist die Reaktionsdauer auf ein GPIO-Ereignis bei beiden untersuchten Konfigurationen aufgelistet. Zunächst sei angemerkt, dass diese Zeitspannen nicht mit den Zeitspannen des vorherigen Experiments vergleichbar sind, da für dieses Experiment eine

Sprache	Minimum	Spannweite	Maximum	Mittelwert
C	8695 ns	650 ns	9345 ns	9033 ns
Assembler	4725 ns	615 ns	5340 ns	4966 ns

Tabelle 4.11.: Auswirkungen der Codeoptimierung auf die Reaktionsdauer

vollständige Unterbrechungsroutine und zugehöriger GPIO-Behandlungsroutine verwendet wurde.

Es ist ersichtlich, dass nicht nur die durchschnittliche Dauer sondern insbesondere auch die Schwankungen reduziert werden konnte.

4.5.2.4 Ausführungsdauer der Unterbrechungsroutine

Die Unterbrechungsroutine muss zunächst einen Zeitstempel für das aufgetretene Ereignis sichern. Danach muss die Quelle des Ereignisses bestimmt und die entsprechende Behandlungsroutine (z. B. Timer-Interrupt) aufgerufen werden. In vielen Fällen muss diese Routine die Quelle (z. B. die Nummer des Timers) genauer bestimmen und in eine noch spezifischere Routine springen.

Die Ausführungsverzögerungen für diese Routinen können reduziert werden, wenn die Routinen und deren Daten im Cache vorgehalten werden. Allerdings sollte, wie in Kap. 3.3.1.2 angemerkt wurde, dieser Mechanismus nur an wichtigen Stellen verwendet werden. Daher werden nicht alle, sondern nur die wichtigsten Routinen dauerhaft im Cache vorgehalten. In der Implementierung des Kommunikationsframeworks (Kap. 5.3) beschränkt sich die Sperrung nur auf die Hauptroutine der Unterbrechungsbehandlung und die Behandlung der GPIO- und Timer-Ereignisse.

Ferner bietet es sich an, diese Routinen nicht durch einen C-Compiler zu generieren, sondern direkt in Assembler zu implementieren. Eine effiziente Assember-Implementierung erfordert jedoch ein sehr gutes Verständnis über die Prozessorarchitektur, insbesondere deren Befehlssatz und Pipeline. In diesem Experiment werden diese Maßnahmen realisiert und erneut die Reaktionsdauer gemessen.

Im Vorfeld der Messung wurden die Hauptroutine der Unterbrechungsbehandlung sowie die GPIO-Behandlungsroutine in ARM-Assembler implementiert. Dabei wurde vor allem auf die Vermeidung von Stalls durch Abhängigkeiten geachtet. Dies wurde durch eine Analyse des Codes mit Hilfe der Darstellung in statischer Einzelzuweisungsform [42] (SSA – *Single Static Assignment*) erreicht. Die Messung ermittelt wieder die Reaktionsdauer des Programms.

Ergebnis: Tabelle 4.11 gibt die gemessenen Reaktionsdauern an. Als Bezugsmessung gilt dabei das Ergebnis des vorherigen Experiments. Es sollte jedoch beachtet werden, dass nicht nur die Programmiersprache gewechselt wurde, sondern noch wenige weitere Optimierungen durchgeführt wurden, die sich auf die Messung auswirken. Beispielsweise wurde, wie in der Einleitung erwähnt, die Routine der GPIO-Behandlung von der Cacheverdrängung ausgeschlossen.

Durch die Implementierung der Unterbrechungsroutine in Assembler konnte die durchschnittliche Behandlungsdauer um 45% auf 4966 ns verringert werden. Die Schwankungen wurden um 5% auf 615 ns verringert.

4.5.2.5 Zusammenspiel der Optimierungen und Evaluierung

In den vorherigen Abschnitten wurden die Verzögerungen der Unterbrechungsbehandlung untersucht. Zusätzlich wurden Mechanismen und Optimierungen angegeben, um die Gesamtdauer der Behandlung und insbesondere die Schwankungen zu reduzieren.

In diesem Abschnitt soll nun das Zusammenspiel der einzelnen Optimierungen evaluiert werden. Dazu werden verschiedene praxisorientierte Szenarien vorgestellt, die implementiert und anschließend mit der ursprünglichen und der optimierten Variante der Unterbrechungsbehandlung ausgeführt werden.

Durchführung: Die Ausführungsdauer der Unterbrechungsbehandlung wird von vielen Parametern beeinflusst. Für die Messungen werden ein paar dieser Parameter herausgegriffen und deren Extremwerte betrachtet. Durch die Kombination dieser Werte ergeben sich fünf Testfälle.

Zunächst werden die Testparameter vorgestellt:

- Cachelast: Im Best-Case werden während der Ausführung des Systems sehr wenige Cacheeinträge verdrängt. So werden auch die Einträge der Unterbrechungsbehandlung seltener verdrängt (insofern die Verdrängung dieser Einträge nicht gesperrt wurde). Demgegenüber steht im Worst-Case ein System, welches sehr viele Zugriffe auf verschiedene Speicherstellen durchführt und dabei eine hohe Anzahl an Verdrängungen auslöst.
- Buslast: Bei diesem Parameter ist vor allem die potentielle Fertigstellung einer lang andauernden Instruktion gemeint. Der Name „Buslast“ wurde gewählt, da insbesondere Busoperationen lang andauernde Instruktionen darstellen.

Die Optimierungen, die in der optimierten Variante zum Einsatz kommen, sind:

- Sperrung der Verdrängung von Cacheeinträgen, die den Code bzw. die Daten der Unterbrechungsroutine beinhalten. Analog wurden auch die Einträge des Codes und der Daten der Unterbehandlung (GPIO und Timer) von der Verdrängung ausgenommen.
- Die hardwarenahe Implementierung der relevanten Routinen in Assembler.
- In Tests, in denen Änderungen der Ausführungsdauer festgestellt werden sollte, abhängig davon, ob eine Unterbrechung erwartet oder unerwartet auftritt, wurde zusätzlich der Spinning-Mechanismus verwendet.

Somit ergeben sich fünf sinnvolle Tests:

1. Ohne Optimierungen, wenig Cacheverdrängung, keine Buslast.
2. Ohne Optimierungen, hohe Cacheverdrängung, hohe Buslast.
3. Mit Optimierungen, wenig Cacheverdrängung, keine Buslast. Dieses Szenario entspricht einer „erwarteten Unterbrechung“. Das Konzept wurde in Kap. 4.5.2.2 vorgestellt und greift auf den Spinning-Mechanismus zurück.
4. Mit Optimierungen, wenig Cacheverdrängung, hohe Buslast. Dieses Szenario entspricht einer „unerwarteten Unterbrechung“.
5. Mit Optimierungen, hohe Cacheverdrängung, hohe Buslast. In diesem Fall wurde ebenfalls die Unterbrechung erwartet und der Spinning-Mechanismus verwendet.

#	Opt.	Cachelast	Buslast	Min.	Spannweite	Max.	Mittel
1		wenig	keine	8730 ns	80 ns	8810 ns	8871 ns
2		hoch	hoch	17415 ns	1730 ns	19145 ns	18117 ns
3	✓	wenig	keine	4075 ns	80 ns	4155 ns	4118 ns
4	✓	wenig	hoch	4115 ns	1045 ns	5160 ns	4626 ns
5	✓	hoch	hoch	4085 ns	1065 ns	5150 ns	4605 ns

Tabelle 4.12.: Reaktionsdauer auf ein GPIO-Ereignis in verschiedenen Szenarien mit und ohne Optimierungen

Ergebnis: In Tab. 4.12 sind die Messergebnisse der Reaktionsdauer angegeben. Im Best-Case (Messung 1 und 3) konnte zwar keine Verringerung der Spannweite festgestellt werden, jedoch konnte die Gesamtdauer auf etwa die Hälfte reduziert werden. Im Worst-Case (Messung 2 und 5) konnte die Gesamtdauer sogar auf etwa ein Viertel und die Spannweite um ca. 40% reduziert werden.

Des Weiteren konnten die Schwankungen zwischen Worst- und Best-Case stark verringert werden. Hierzu wird das Minimum des Best-Case mit dem Maximum des Worst-Case verglichen. Ohne Optimierungen (Messung 1 und 2) ergibt sich eine Schwankung von $19145\,\text{ns} - 8730\,\text{ns} = 10415\,\text{ns}$. Mit Optimierung ergibt sich eine Schwankung von $5160\,\text{ns} - 4075\,\text{ns} = 1075\,\text{ns}$ (Messung 3 und 5) bzw. $5150\,\text{ns} - 4075\,\text{ns} = 1085\,\text{ns}$ (Messung 3 und 4).

4.6 Zusammenfassung der Experimente und Optimierungen

In diesem Kapitel wurden einige Experimente zur Messung von Zeitspannen und Schwankungen, zum Feststellen von funktionalem Verhalten und zur Evaluierung von Optimierungsalternativen hinsichtlich des CC 2420-Transceivers und des PXA 271-Prozessors durchgeführt.

Dabei konnten wichtige Eigenschaften des CC 2420-Transceivers herausgearbeitet werden. Insbesondere wurden dabei die Verzögerungen bestimmt, die der Transceiver zum Umschalten vom Empfangs- in den Sendezustand (und umgekehrt) benötigt. Diese Verzögerungen werden für die Implementierung Black Burst-basierter Protokolle, insbesondere des Master-basierten BBS-Verfahrens, benötigt. Ferner wurde funktionales Verhalten der Hardware analysiert und dabei auch Abweichungen vom Datenblatt festgestellt.

Im Hinblick auf den PXA 271-Prozessor wurden Verzögerungsquellen bei der Unterbrechungsbehandlung identifiziert. Auch diese haben einen Einfluss auf die Robustheit Black Burst-basierter Protokolle und auf die Synchronisationsgenauigkeit des Master-basierten BBS-Verfahrens. Um den Schwankungen entgegenzuwirken, wurden Optimierungstechniken vorgeschlagen und evaluiert. Durch die Kombination der einzelnen Optimierungen konnten die Gesamtdauer und die Schwankungen der Unterbrechungsbehandlung minimiert werden.

5. Kapitel

Design und Implementierung

In Kap. 2 wurden die Konzepte Black Burst-basierter Kommunikation vorgestellt. Aufbauend auf diesen und zusammen mit den Erkenntnissen aus den Experimenten (Kap. 4) wird nun ein Kommunikationsframework vorgestellt, das die Konzepte integriert. Ziel des Frameworks ist die einfache Handhabung der verschiedenen Kommunikationsverfahren. Bei der Entwicklung einer Anwendung kann das Framework verwendet werden, um sämtliche Aspekte des Medienzugriffs abzudecken.

Ein wichtiges Entwurfsziel des Frameworks war die lose Kopplung der einzelnen Komponenten. Dadurch ist es möglich, Anwendungen zu entwickeln, die nur einzelne Teile des Frameworks benötigen. Ein Beispiel hierfür wäre eine Anwendung, die lediglich eine Möglichkeit zur Synchronisation benötigt, die restliche Kommunikation jedoch selbst verwaltet. Im vollen Umfang kann das Framework jedoch die gesamte Verwaltung auf MAC-Ebene übernehmen, so dass die Anwendung nur noch die Daten bereitstellen muss, die gemäß einer globalen Konfiguration an die involvierten Kommunikationsprotokolle verteilt werden.

In diesem Kapitel werden zunächst die Konzepte des Frameworks vorgestellt. Dann wird der Aufbau des Frameworks mit den zugehörigen Schnittstellen beschrieben. Zuletzt wird auch ein kurzer Überblick über die Implementierung gegeben, die im Rahmen der Arbeit durchgeführt wurde.

5.1 Konzepte des Kommunikationsframeworks

In diesem Kapitel wird ein Kommunikationsframework vorgestellt, welches die Grundlage für Black Burst-basierte Protokolle liefert. Ähnlich zu MacZ [7] (Kap. 2.5) wird das Medium zeitlich unterteilt. Die dazugehörigen Konzepte wurden weitgehend von MacZ übernommen.

In den folgenden Abschnitten werden die einzelnen Aspekte des Frameworks vorgestellt. Zunächst wird die zeitliche Einteilung des Mediums betrachtet. Des Weiteren werden die verschiedenen Übertragungsarten vorgestellt, die das Framework bietet.

5.1.1 Einteilung des Mediums (Slotting)

Wie bei MacZ wird das Medium zeitlich in Makroslots und diese in einen Syncslot und mehrere Mikroslots eingeteilt. Konsekutive Mikroslots können in virtuelle Slotregionen gruppiert werden. Innerhalb einer Slotregion kommt jeweils nur eine Übertragungsart zum

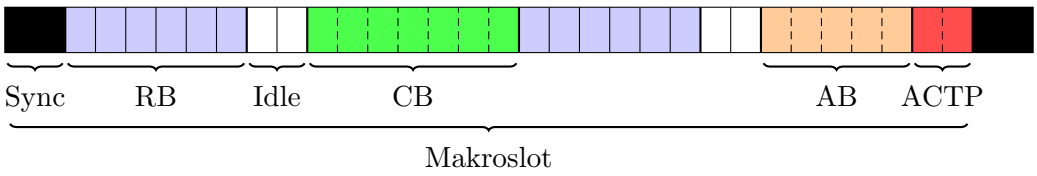


Abbildung 5.1.: Beispielhafter Aufbau eines Makroslots

Einsatz. Es sind aber, abhängig von der Art der Übertragung, mehrere Übertragungen in einer Region möglich.

Abbildung 5.1 zeigt den exemplarischen Aufbau eines Makroslots. Im Folgenden werden die verschiedenen Arten der Slotregionen vorgestellt:

Synchronisation (Sync): Am Anfang eines Makroslots wird immer eine Resynchronisation durchgeführt, um die Knoten hinreichend genau zu synchronisieren. So können die Mikroslotgrenzen und Bedingungen, die für Reservierung und zur Funktion Black Burst-basierter Protokolle – wie z. B. ACTP – nötig sind, eingehalten werden. Als Synchronisationsverfahren kommt BBS (Kap. 2.3) zum Einsatz.

Reservierungsbasiert (RB): Jeder Mikroslot einer reservierungsbasierten Slotregion ist im Single-Hop Fall maximal einem Knoten zugeordnet. In Multi-Hop Netzwerken, in denen sich die zeitgleiche Übertragung von Rahmen durch zwei Knoten nicht bei den jeweiligen Empfängern destruktiv überlagern, kann die Reservierung eines Mikroslots auch beiden Knoten zugeteilt werden (SDMA – *Space Division Multiple Access*).

Die Zuordnung wird statisch, d.h. vor der Inbetriebnahme des Netzwerks, oder dynamisch mit Hilfe eines Reservierungsprotokolls durchgeführt. In einem Slot dürfen nur Knoten exklusiv auf das Medium zugreifen, die diesen Slot reserviert haben. Die Reservierung muss so erfolgen, dass es nicht zu destruktiven Kollisionen kommen kann. Somit können folglich alle Maßnahmen für Kollisionsvermeidungen (Kap. 2.1) entfallen. Zur Übertragung der Daten werden reguläre IEEE 802.15.4-Rahmen verwendet.

Diese Übertragungsart ist insbesondere für Übertragungen mit hohen Dienstgüteanforderungen vorgesehen, die periodisch Daten austauschen und dabei eine obere Schranke einhalten müssen. Ein Beispiel hierfür ist eine Anwendung, die alle 100 ms Audiodaten übermittelt.

Wettbewerbsbasiert (CB): In wettbewerbsbasierten Slotregionen kann potentiell jeder Knoten eine Übertragung durchführen. Für die Übertragung der Daten werden reguläre IEEE 802.15.4-Rahmen verwendet. Da in diesen Regionen mehrere Knoten zeitgleich eine Übertragung beginnen können, kann es zu destruktiven Kollisionen kommen. Um diese zu verringern, kommen zufällige Wartezeiten (Backoff) und optional ein RTS/CTS-Handshake zum Einsatz. Den Mikroslotgrenzen kommen innerhalb der Region keine weitere Bedeutung zu.

Diese Übertragungsart eignet sich für unregelmäßige Übertragungen, für die keine oberen Schranken eingehalten werden müssen. Durch die Kollisionen kann es außerdem zum Rahmenverlust kommen. Eine Erkennung oder Behandlung dieser Verluste ist jedoch zur Zeit nicht auf MAC-Ebene vorgesehen.

Arbitrierungsbasiert (AB): An einer arbitrierungsbasierten Übertragung kann ebenfalls jeder Knoten teilnehmen. Am Anfang der Region wird zunächst eine Arbitrierung mit ACTP (Kap. 2.4) durchgeführt. An diesem Arbitrierungsvorgang nimmt grundsätzlich jeder Knoten teil. Auch wenn kein eigener Sendewunsch besteht, leitet ein Knoten Bits gemäß ACTP weiter, so dass der gesamte Arbitrierungsradius erreicht wird. Bei Sendewunsch nimmt der Knoten mit einem Wert teil, der die Priorität der eigentlichen Übertragung widerspiegelt. Durch den Einsatz von ACTP wird dabei der höchste Wert ermittelt, der am Ende des Verfahrens allen Knoten in Arbitrierungsreichweite bekannt ist. Der Knoten, dessen Wert sich dabei durchgesetzt hat, hat im Anschluss an den Arbitrierungsvorgang exklusiven Zugriff auf das Medium. Bei netzweiter Arbitrierung wird in jeder arbitrierungsbasierten Region nur maximal eine Übertragung durchgeführt. Wird der Arbitrierungsradius kleiner gewählt, sind parallele Übertragungen möglich.

ACTP: Diese Übertragungsart ähnelt der arbitrierungsbasierten Übertragung. Sie kann verwendet werden, wenn der zur Arbitrierung verwendete Wert bereits die eigentliche Information darstellt. Das bedeutet, dass durch den Einsatz von ACTP der größte Wert mehrerer teilnehmender Knoten ermittelt und im Netzwerk propagiert wird. Auch in diesen Regionen wird maximal eine Übertragung durchgeführt.

Idle: In Idle-Regionen findet keine Kommunikation statt. Die Knoten können in diesen Regionen den Transceiver abschalten, um Energie zu sparen (*Duty Cycling* [12]). Ferner ist auch denkbar, diese Regionen durch den Einsatz eines Reservierungsprotokolls dynamisch zu verwalten und somit Slotregionen je nach Bedarf angepasst werden.

5.1.2 Logische Gruppierung von Übertragungen

Für viele Anwendungen ist die Reservierung eines einzelnen reservierungsbasierten Slots pro Makroslot nicht ausreichend. Möchte beispielsweise eine Anwendung alle 100 ms Audiodaten übertragen, müssen dafür mehrere einzelne Slots reserviert werden. Die Anwendung müsste dann selbst die Reihenfolge der Slots kennen und selbst kontrollieren, welchem Slot die anfallenden Daten zugeteilt werden. Um diesen Verwaltungsaufwand in der Anwendung zu reduzieren, wird eine weitere Abstraktionsschicht eingeführt, die als *Transmission Opportunities* (etwa „Übertragungsmöglichkeiten“) bezeichnet wird. Eine Anwendung stellt die anfallenden Daten der Transmission Opportunity bereit und die entsprechende Komponente des Framework weißt diese selbstständig den jeweiligen Slots zu.

Das Konzept ist nicht nur auf reservierungsbasierte Regionen (RB) beschränkt, sondern kann genauso auf ACTP- und arbitrierungsbasierten Regionen (AB) angewandt werden. Für wettbewerbsbasierte Übertragungen ist nur die Gruppierung aller wettbewerbsbasierten Regionen (CB) sinnvoll. Somit gibt es eine Transmission Opportunity, die alle wettbewerbsbasierten Regionen logisch zusammenfasst. Abbildung 5.2 zeigt einen Makroslotaufbau, bei dem drei Transmission Opportunities o_1, o_2 und o_3 mit unterschiedlichen Übertragungsarten verwendet werden.

Diese Gruppierung kann natürlich nur sinnvoll angewandt werden, wenn die Daten, die für eine Transmission Opportunity vorgesehen sind, auch in allen zugehörigen Slots potentiell versendet werden können. Dies impliziert, dass alle Slots einer Transmission Opportunity von der gleichen Übertragungsart (z. B. nur reservierungsbasiert oder nur arbitrierungsbasiert)

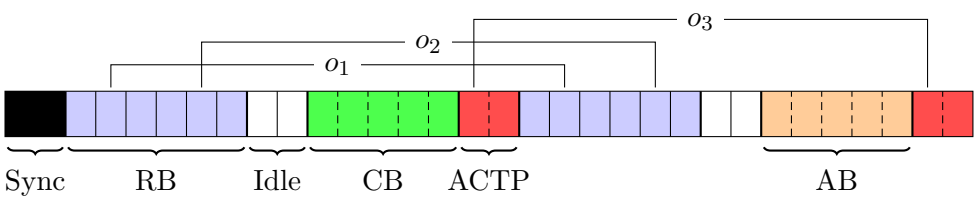


Abbildung 5.2.: Gruppierung zusammengehöriger Übertragungen in Transmission Opportunities

sein müssen. Ferner müssen auch einige Parameter der Übertragung (z. B. die Anzahl an Bits bei ACTP) gleich sein.

5.2 Design des Kommunikationsframeworks

Die Gesamtfunktionalität des Frameworks ist in mehrere Komponenten aufgeteilt. Das Framework ist als Schichtenarchitektur aufgebaut, die in Abb. 5.3 gezeigt ist. Jede Komponente verwendet nur die Schnittstellen der angrenzenden Schicht. Der Rückpfad ist über Callbacks realisiert.

Ein besonderes Designziel bei der Ausarbeitung der Schnittstellen war die lose Kopplung der Komponenten. Dadurch ist es möglich, das Framework nur bis zu einer beliebigen Schicht zu verwenden, wenn die Funktionalität höherer Schichten von einer Anwendung nicht gebraucht wird.

Da die wesentlichen Aufgaben der einzelnen Komponenten bei der Kommunikation (z. B. Übermittlung der Rahmendaten vom bzw. zum Transceiver auf CC 2420-Ebene oder die Durchführung eines Arbitrierungsvorgangs auf ACTP-Ebene) eine lange Dauer benötigen, sind die Schnittstellen der Komponenten asynchron ausgelegt. Das bedeutet, dass durch einen Funktionsaufruf (z. B. zur Übertragung eines Rahmens) lediglich die Aufgabe eingeleitet, nicht aber auf deren Abschluss gewartet wird. Wenn die Aufgabe zu einem späteren Zeitpunkt schließlich beendet ist, wird dies durch einen Callback angezeigt. Dies erhöht zwar an manchen Stellen die Komplexität, da der Kontext über diese Zeit gesichert werden muss, erlaubt jedoch der Anwendung, von der Komponente ungenutzte Rechenkapazität des Knotens zu nutzen.

Im Folgenden werden die einzelnen Schichten und ihre Schnittstellen im Detail vorgestellt.

5.2.1 Schicht 0: Systemsoftware

Die Systemsoftware stellt im Kontext der Kommunikationsarchitektur vor allem die low-level Subsysteme zur Verfügung, die zur Interaktion mit dem CC 2420-Transceiver nötig sind. Dazu gehören die GPIO- und SPI-Treiber, über die der Transceiver angesprochen wird (Kap. 3.4). Neben diesen Subsystemen werden auch Schnittstellen zu weiterer Peripherie wie z. B. UART und Timer angeboten.

Darüber hinaus übernimmt die Schicht die Behandlung von Unterbrechungen. Dabei wird zunächst der Zeitstempel des Ereignisses gesichert, die Unterbrechungsquelle identifiziert und die entsprechende Behandlungsroutine des Subsystems (GPIO, SPI, DMA, Timer, etc.)

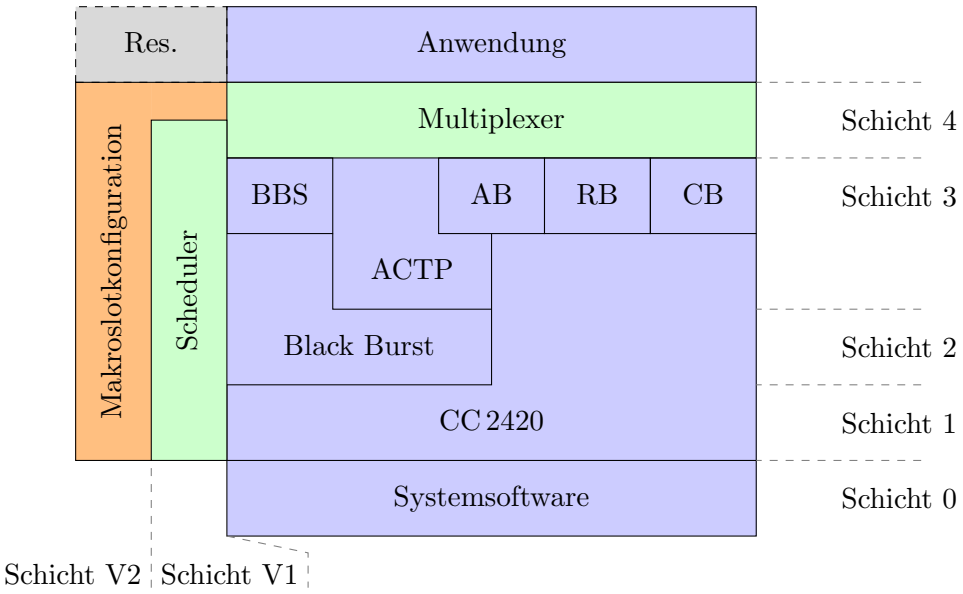


Abbildung 5.3.: Übersicht über die Komponenten des Kommunikationsframeworks

aufgerufen. Im Unterbrechungs-Subsystem ist auch der Spinning-Mechanismus (Kap. 4.5.2.2) implementiert.

Ferner wird auch die Kernhardware von der Systemsoftware verwaltet. Dazu gehört die Abbildungstabelle der MMU, Cache- und TLB-Sperrung (Kap. 3.3.1) und das Setzen der CPU-Frequenz.

Eine weitere Aufgabe der Systemsoftware ist das Initialisieren der Anwendung (z. B. Laden der Programmdaten in den Arbeitsspeicher, Aufsetzen des Stacks, etc.) sowie die Behandlung von Ausnahmen im Fehlerfall (z. B. fehlerhafte Speicherzugriffe).

5.2.2 Schicht 1: CC 2420-Treiber

Der CC 2420-Treiber übernimmt die Interaktion mit dem CC 2420-Transceiver. Neben der Übertragung von Daten und Befehlen wird die gesamte Verwaltung des Funkchips in diesem Treiber gekapselt.

Senderichtung: Zum Versenden von Rahmen bietet der Treiber einige Funktionen, um den gesamten Sendevorgang zu kontrollieren. Die Schnittstelle zwischen CC 2420-Treiber und den darüber liegenden Schichten besteht im Wesentlichen aus folgenden Funktionen:

- `writeTXFifo` (*phyPayload*: `Octet`*)
  schreibt die Nutzdaten des PHY-Rahmens *phyPayload* in den Übertragungspuffer des Transceivers.

  Die Länge der Daten muss entsprechend der Limitierungen des IEEE 802.15.4-Standards und des CC 2420-Transceivers (Kap. 3.2.1.5) eingehalten werden.

  Die Funktion stößt die Übertragung der Daten im Hintergrund an und kehrt direkt wieder zurück. Durch den Callback `TXReady` wird dem Diensnutzer zu einem späteren

Zeitpunkt mitgeteilt, dass die Daten in den Puffer übertragen wurden und versendet werden können.

- `transmitTXFifo` (cca: $\mathbb{B}$) : $\mathbb{B}$
  beginnt die Übertragung, die zuvor in den Übertragungspuffer geschrieben wurden. Mit dem Parameter *cca* kann bestimmt werden, ob vor der Übertragung anhand des CCA-Mechanismus noch geprüft werden soll, ob das Medium belegt ist.

  Die Funktion kehrt nach Anstoßen der Übertragung direkt zurück. Der Callback `TXDone` wird ausgelöst, wenn die Übertragung beendet ist.

  Der boolesche Rückgabewert der Funktion gibt an, ob die Übertragung erfolgreich eingeleitet wurde oder (im Falle von $cca =$ `wahr`) das Medium belegt war.

  Wenn der gleiche IEEE 802.15.4-Rahmen mehrfach übertragen werden soll, kann diese Funktion auch mehrfach aufgerufen werden, ohne dass dazwischen ein erneutes `writeTXFifo` nötig ist. Dies hat zum Vorteil, dass Protokolltreiber Black Burst-basierter Protokolle nur einen IEEE 802.15.4-Rahmen, der einen dominanten Black Burst abbildet, zur Hardware übertragen müssen und diesen mehrfach verwenden können.

- Callback `TXReady` ()
  wird nach einem `writeTXFifo` ausgelöst, wenn die Daten vollständig in den Puffer des Transceivers geschrieben wurden.

- Callback `TXDone` ()
  wird nach einem `transmitTXFifo` ausgelöst, nachdem die Übertragung erfolgreich abgeschlossen wurde.

Empfangsrichtung: Der Empfang von Daten bezieht sich zunächst auf das Abholen empfangener Daten aus dem Empfangspuffer des CC 2420-Transceivers. Darüber hinaus wird zusätzlich der Zustand der `CCA`- und `SFD`-Signale überwacht und die Zeitpunkte der Ereignisse an höhere Schichten weitergeleitet. Die Schnittstelle besteht ausschließlich aus Callbacks und ist wie folgt aufgebaut:

- Callback `RX` ($phyPayload$: `Octet`*, $rflevel$: $\mathbb{Z}$, crc: $\mathbb{B}$)
  wird ausgelöst, wenn ein Rahmen empfangen wurde. Die PHY-Nutzdaten wurden bereits aus der Empfangspuffer des Transceivers ausgelesen und werden über $phyPayload$ zur Verfügung gestellt.

  Der $rflevel$-Parameter gibt die ungefähre Signalstärke des empfangenen Signals in dBm an, die aus dem RSSI-Wert (*Received Signal Strength Indicator*) gemäß dem Datenblatt errechnet wurde.

  Der boolesche *crc*-Parameter gibt an, ob die Prüfsumme des Rahmens korrekt ist. Die $rflevel$- und *crc*-Parameter sind nur gültig, wenn der CC 2420 die Prüfsumme automatisch verwaltet.

- Callback `RX_SFD` (on: $\mathcal{T}$, off: $\mathcal{T}$)
  wird, wenn der Transceiver im Empfangsmodus ist, nach der fallenden Flanke des `SFD`-Signals ausgelöst, wodurch das Ende eines Rahmenempfangs angezeigt wird. Die

beiden Parameter *on* bzw. off geben die auf den Knoten bezogene Zeitpunkte der steigenden bzw. fallenden Flanke des Signals an.

Die Domäne der Zeitpunkte $\mathcal{T}$ ist konzeptionell eine nicht negative, reelle Zahl (also $\mathbb{R}_0^+$). In einer Implementierung werden Zeitpunkte jedoch in einem endlichen, diskreten (meist ganzzahligen) Wertebereich angegeben.

Zu diesem Zeitpunkt wurden noch keine Daten ausgelesen, da das Auslesen an das `FIFOP`-Signal (Kap. 3.4) gekoppelt ist, welches immer erst nach dem `SFD`-Signal auslöst.

- Callback `RX_CCA` ($on\colon \mathcal{T},\ off\colon \mathcal{T}$)
  wird im Empfangsmodus nach der steigenden Flanke des `CCA`-Signals ausgelöst. Die Zeitpunkte *on* bzw. off geben den Beginn bzw. das Ende einer Medienbelegung an.

Eine weitere Aufgabe des CC 2420-Treibers besteht im Verwalten der Hardwareregister, die das Verhalten des Transceivers bestimmen. Darüber werden beispielsweise der zu verwendende Funkkanal, Sendeleistung, Präambellänge, Umschaltdauer, etc. eingestellt. Da für Black Bursts potentiell viele dieser Register umkonfiguriert werden müssen, bietet die Schnittstelle des CC 2420-Treibers einen Mechanismus, mehrere Hardwareregister zeitgleich (d. h. in einer einzigen SPI-Transaktion) zu setzen. Des Weiteren ist es für eine Implementierung empfehlenswert, nur diejenigen Register neu zu schreiben, deren Wert sich auch tatsächlich geändert hat.

5.2.3 Schicht 2: Black Bursts

Die Black Burst-Schicht ist für das Versenden und Empfangen einzelner Black Bursts verantwortlich. Wenn ein Black Burst-basiertes Protokoll eine Übertragung durchführt, aktiviert es diese Schicht, die daraufhin die gesamte Kontrolle des CC 2420-Treibers übernimmt. Dabei werden die oben genannten Callbacks registriert und die Register des CC 2420-Transceivers über dessen Treiber umkonfiguriert. Nachdem das Protokoll die Übertragung abgeschlossen hat, wird die Schicht deaktiviert. Dadurch wird die vorherige Hardwarekonfiguration wiederhergestellt, so dass andere Benutzer des CC 2420-Treibers (z. B. der CB-Protokolltreiber) keine weitere Rekonfiguration vornehmen müssen.

Die Schnittstellen zwischen dem Black-Burst Treiber und seinen Dienstnutzern sind:

- `activate` ($ccaThreshold\colon \mathbb{Z},\ txPower\colon \mathbb{Z}$)
  aktiviert den Black Burst-Treiber und löst somit die beschriebene Übernahme des CC 2420-Treibers aus.

  Der Parameter *ccaThreshold* gibt in dBm den Schwellwert der Signalenergie an, der beim CCA-Mechanismus erreicht werden muss, um eine Medienbelegung zu erkennen.

  Der Parameter *txPower* gibt die Sendeleistung in dBm an, die zum Versenden eines Black Bursts verwendet wird.

- `deactivate` ()
  deaktiviert den Black Burst-Treiber und stellt die Werte der CC 2420-Hardwareregister wieder her.

Senderichtung: Das Senden von Black Bursts ist in zwei Teilvorgänge aufgespalten: Zunächst wird die Übertragung vorbereitet, so dass diese zu einem späteren Zeitpunkt möglichst verzögerungsfrei durchgeführt werden kann. Die Schnittstelle ist wie folgt aufgebaut:

- `prepareTX` (*blackBurst*: ⟨`BlackBurst`⟩)
  bereitet das Übertragen eines Black Bursts vor, der entsprechend dem Parameter *blackBurst* konfiguriert ist. Dabei wird die Übertragung nur vorbereitet. Ein Empfang von Black Bursts ist weiterhin möglich.

  Der Typ ⟨`BlackBurst`⟩ ist eine Struktur, die die Eigenschaften des Black Bursts beschreibt. Hierbei wird eine große Entscheidungsfreiheit bzgl. der Abbildung des Black Bursts auf IEEE 802.15.4-Rahmen gegeben. Zu den Konfigurationsparametern gehören u. A. die Länge der Präambel, das SyncWord und die Länge der PHY-Nutzdaten (Kap. 3.2.2.1). Außerdem kann konfiguriert werden, ob der Black Burst als regulärer IEEE 802.15.4-Rahmen versendet werden soll. Dies ist beim Master-basierten BBS-Verfahren zur Übertragen des SOF des ersten Tickrahmens der Fall (Kap 2.3.3).

  Wie schon im CC 2420-Treiber ist es auch hier empfehlenswert, nur dann aktiv eine Konfiguration vorzunehmen, wenn auch tatsächlich der Zustand der Hardware geändert wird. Beispielsweise ist eine erneute Füllung der TXFifo (siehe `writeTXFifo` des CC 2420-Treibers) nicht nötig, wenn sich die Länge der Nutzdaten über konsekutive Black Bursts nicht ändert.

- `tx` ()
  versendet den vorbereiteten Black Burst. Das Ende der Übertragung wird zu einem späteren Zeitpunkt durch den Callback `TXDone` angezeigt.

  Werden mehrere Black Bursts mit der gleichen Konfiguration übertragen, kann ein erneutes `prepareTX` zwischen den Übertragungsvorgängen entfallen und die `tx`-Funktion mehrmals aufgerufen werden.

- Callback `TXDone` ()
  zeigt das Ende einer Übertragung an. Dieses Ereignis kann verwendet werden, um die Übertragung eines weiteren Black Bursts vorzubereiten, insofern dessen Konfiguration abweicht.

Empfangsrichtung: Der Empfang von Black Bursts muss vorbereitet werden, da je nach verwendetem Erkennungsmechanismus die Hardware entsprechend rekonfiguriert werden muss. Der eigentliche Empfang wird über einen Callback mitgeteilt. Die Schnittstelle gestaltet sich wie folgt:

- `prepareRX` (*blackBurst*: ⟨`BlackBurst`⟩, *filter*: ⟨`Filter`⟩)
  bereitet den Empfang von Black Bursts vor. Die Notwendigkeit der Vorbereitung ist vor allem auf die verschiedenen Arten der Erkennung von Black Bursts zurückzuführen. Wenn ein Black Burst über den SFD-Mechanismus erkannt werden soll, muss z. B. die Länge der Präambel bekannt sein, um den eigentlichen Startzeitpunkt des Black Bursts zurückzurechnen. Für die Beschreibung des Black Bursts wird wieder die Struktur ⟨`BlackBurst`⟩ verwendet.

Zusätzlich können Medienbelegungen anhand ihrer erkannten Länge gefiltert werden und eine Filterung oder Klassifikation auf höherer Ebene kann entfallen. Wird beispielsweise beim Master-basierten BBS-Verfahren ein SOF-Black Burst erwartet, reicht der Black Burst-Treiber nur Empfangsereignisse zum BBS-Treiber, wenn die Länge der erkannten Medienbelegung mit der erwarteten Länge des SOF-Black Bursts übereinstimmt. Die Struktur ⟨`Filter`⟩ sieht für die Filterung eine untere und obere Schranke bzgl. der Belegungsdauer vor.

- Callback `RX` (*start*: $\mathcal{T}$, *duration*: $\mathcal{T}$, *method*: ⟨`BBMethod`⟩)
  wird nach Erkennen eines Black Bursts ausgelöst. Die Parameter *start* bzw. *duration* geben den Startzeitpunkt bzw. die Belegungsdauer des Black Bursts an.

  Der Parameter *method* gibt an, mit welcher Methode der Black Burst (`CCA` oder `SFD`) erkannt wurde. Grundsätzlich sollte nie ein Black Burst per SFD erkannt werden, wenn dies nicht konfiguriert wurde. Umgekehrt aber können Black Bursts per CCA erkannt werden, wenn eigentlich die Erkennung anhand des SFD-Mechanismus konfiguriert wurde. Dies wird auch von dem Master-basierten BBS-Verfahren benötigt, da nur das SOF der ersten Runde als regulärer IEEE 802.15.4-Rahmen versendet wird, ein Slave aber im Regelfall nicht wissen kann, in welcher Runde er synchronisiert wird. Folglich muss er das SOF auf beide Arten erkennen können.

5.2.4 Schicht 3a: Black Burst Synchronization

Die gesamte dritte Schicht integriert die verschiedenen Kommunikationsprotokolle in das Framework. Zunächst wird hier der BBS-Treiber betrachtet, da diesem Protokoll eine besondere Rolle zukommt. Ferner beschränkt sich die Betrachtung auf das Master-basierte BBS-Verfahren. Es ist aber auch denkbar, die dezentralisierte oder hybride BBS-Variante einzusetzen. Die Schnittstellen zu weiteren Protokollen werden im nächsten Abschnitt vorgestellt.

Neben des lokalen Ticks werden von der Schicht weitere Informationen zur Verfügung gestellt, die anhand des Synchronisationsverlaufs abgeleitet werden können. Dazu gehören das momentane maximale Tick-Offset, die Hopdistanz zum Master und der Synchronisationsgrad eines Slaves (Kap. 2.3.6).

Die Schnittstellen des Master-basierten BBS-Verfahrens zu seinen Dienstnutzern sind wie folgt:

- `init` (*configuration*: ⟨`BBSConfig`⟩)
  initialisiert die Verwendung des Master-basierten BBS-Protokolls. Der Parameter *configuration* gibt die Konfiguration des Verfahrens an. Die Elemente der Struktur sind in Tabelle 5.1 angegeben.

- `synchronize` (*at*: $\mathcal{T}$)
  beginnt eine Synchronisationsphase zum Zeitpunkt *at*.

  Ein synchronisierter Slave erwartet den Beginn der Phase zu diesem Zeitpunkt. Das Ende der Phase kann der Protokolltreiber durch die Konvergenzdauer berechnen. Erkennt ein Slave bis zu diesem Endzeitpunkt keine Synchronisationsrunde, ist der Knoten in dieser Phase nicht erfolgreich synchronisiert. In diesem Fall wird der Grad der Synchronisation gemäß dem Vorgehen aus Kap. 2.3.6 zu `WEAKLY_SYNCHRONIZED` angepasst.

Parameter	Domäne	Beschreibung
role	⟨`Role`⟩	Die Rolle des Knotens beim Verfahren, also `MASTER` oder `SLAVE`
maxHops	$\mathbb{N}$	Die maximale Anzahl an Hops $n_{maxHops}$ (Kap. 2.3.2)
ccaThreshold	$\mathbb{Z}$	der Schwellwert des CCA-Mechanismus in dBm. Dieser wird an den Black Burst-Treiber weitergereicht
txPower	$\mathbb{Z}$	die Sendeleistung beim Übertragen von Black Bursts in dBm
channel	$\mathbb{N}$	Der zu verwendende IEEE 802.15.4-Funkkanal ($\{11, \ldots, 26\}$)
syncPhaseThreshold	$\mathbb{N}$	Gibt die Mindestanzahl erfolgreicher Synchronisationsphasen an, die ein Knoten erreichen muss, um aktiv an der Kommunikation teilnehmen zu dürfen (Kap. 2.3.6)

Tabelle 5.1.: Konfigurationsparameter des Master-basierten BBS-Verfahrens

Ein unsynchronisierter Knoten hat keine Information über den Beginn der Synchronisationsphase. Daher wird bei unsynchronisierten Knoten der Parameter *at* ignoriert und unmittelbar auf den Empfang von Tickrahmen gewartet.

- Publish/Subscribe-Callback `Tick` (*tick*: $\mathcal{T}$, *hopDistance*: $\mathbb{N}$, *method*: ⟨`BBMethod`⟩, *stage*: ⟨`SyncStage`⟩)
  wird ausgelöst, wenn eine Synchronisationsphase erfolgreich abgeschlossen wurde.

  Der Parameter *tick* gibt den Startzeitpunkt der Phase an, der im Rahmen der initialen Tickabweichung $d_{maxBaseTickOffset}$ allen synchronisierten Knoten bekannt ist.

  Ferner gibt *hopDistance* die Distanz zum Master in Hops an. Diese Information wird aus der Nummer der Runde gewonnen, in der der Knoten synchronisiert wurde.

  Der Parameter *method* gibt an, ob die Synchronisation über den SOF- oder CCA-Mechanismus erreicht wurde.

  Der Parameter *stage* gibt den Grad der Synchronisation, also `UNSYNCHRONIZED`, `PRE_SYNCHRONIZED`, `STRONGLY_SYNCHRONIZED` oder `WEAKLY_SYNCHRONIZED` an.

  Der Callback ist nach dem Publish/Subscribe-Entwurfsmuster [20] realisiert, so dass mehreren Dienstnutzern das Ereignis einer erfolgreichen Synchronisation mitgeteilt werden kann.

- Publish/Subscribe-Callback `StageChange` (*stage*: ⟨`SyncStage`⟩)
  wird bei einer Änderung des Synchronisationsgrades ausgelöst und teilt mit, ob eine Synchronisation erfolgreich hergestellt oder ob eine Phase verpasst wurde.

5.2.5 Schicht 3b: Protokolle für den Medienzugriff

Die Komponenten für die Kommunikationsprotokolle verfügen über weitgehend einheitliche Schnittstellen, so dass deren Handhabung erleichtert wird. Da die Parametrierung einer Übertragung von Protokoll zu Protokoll unterschiedlich ist, sind die Ausprägungen der

Parameter	Domäne	Beschreibung
channel	$\mathbb{N}$	IEEE 802.15.4-Funkkanal ($\{11, \ldots, 26\}$)
bbSyncWord	⟨`SyncWord`⟩	SyncWord für Black Burst-basierte Protokolle
bbPayloadLength	$\mathbb{N}$	PHY-Nutzdatenlänge für Black Bursts

Tabelle 5.2.: Statische Parameter der Protokollkonfiguration

einzelnen Schnittstellen dem jeweiligen Protokoll angepasst. Der allgemeine Kontrollfluss bei einer Übertragung ist jedoch bei allen Übertragungsarten gleich.

Verwaltung des Protokolltreibers: Die Funktionen zur Verwaltung markieren den zeitlichen Kontext des Protokolls. In diesem Kontext übernimmt das jeweilige Protokoll die Kontrolle über den Black Burst- bzw. CC 2420-Treiber. Die Schnittstelle zwischen den Kommunikationsprotokollen und deren Dienstnutzern ist wie folgt aufgebaut:

- `prepare` ()
  bereitet die Verwendung des jeweiligen Protokolls für anstehende Sende- bzw. Empfangsvorgänge vor. Dazu gehören z. B. bei ACTP das Aktivieren der Black Burst-Komponente oder bei CB die Übernahme des CC 2420-Treibers.

- `teardown` ()
  deaktiviert das Protokoll. Der jeweilige Treiber gibt dabei alle Ressourcen wieder frei.

- `configure` (*configuration*: ⟨`StaticConfiguration`⟩)
  setzt die statische Konfiguration des Protokolls. Der Parameter *configuration* des Typs ⟨`StaticConfiguration`⟩ beinhaltet diejenigen Parameter, die für alle Protokolle gleich sind. Die Parameter sind in Tab. 5.2 aufgelistet.

Kommunikation über den Protokolltreiber: Die Schnittstelle bietet Funktionen, um an einer Kommunikation mit der entsprechenden Übertragungsart teilzunehmen. Dabei müssen nicht zwingend eigene Daten versendet werden. Je nach Protokoll kann es aber notwendig sein, Daten anderer Stationen weiterzuleiten.

- `setData` (*data*: ⟨*ProtocolSpecific*⟩)
  stellt Daten für eine Übertragung bereit. Das Format der Daten ist vom konkreten Protokoll abhängig.

  Es können nur die Daten der nächsten Übertragung bereitgestellt werden. Mehrere Sendevorgänge werden durch höhere Schichten verwaltet.

- `clearData` ()
  setzt die bereitgestellten Daten zurück.

- `setConfig` (*configuration*: ⟨*ProtocolSpecific*⟩)
  setzt die Konfiguration anstehender Übertragungen. Die Parameter der Konfiguration sind vom jeweiligen Protokoll abhängig und in Tab. 5.3 aufgelistet.

- `schedule` (*at*: $\mathcal{T}$)
  plant eine Übertragung, die zum Zeitpunkt *at* beginnt. Je nach Protokoll und abhängig

Protokoll	Parameter	Domäne	Beschreibung
ACTP	$arbRad$	$\mathbb{N}$	Arbitrierungsradius
	$bits$	$\mathbb{N}$	Anzahl an Bits
	$chksum$	$\mathbb{B}$	Absichern der Sequenz mit Prüfsumme
	$txPower$	$\mathbb{Z}$	Sendeleistung in dBm
	$ccaThreshold$	$\mathbb{N}$	Schwellwert des CCA-Mechanismus
AB	$arbRad$	$\mathbb{N}$	Arbitrierungsradius
	$bits$	$\mathbb{N}$	Anzahl an Bits bei Arbitrierung
	$chksum$	$\mathbb{B}$	Absichern der Arbitrierungssequenz mit Paritätsbit
	$txPower$	$\mathbb{Z}$	Sendeleistung bei Arbitrierung in dBm
	$ccaThreshold$	$\mathbb{N}$	Schwellwert des CCA-Mechanismus
	$frameTxPower$	$\mathbb{N}$	Sendeleistung für eigentliche Übertragung nach gewonnener Arbitrierung
	$frameDest$	$\mathcal{D}$	Zielinformation des Datenrahmens
RB	$frameTxPower$	$\mathbb{Z}$	Sendeleistung in dBm
	$frameDest$	$\mathcal{D}$	Zielinformation des Datenrahmens
CB	$frameTxPower$	$\mathbb{Z}$	Sendeleistung in dBm
	$frameDest$	$\mathcal{D}$	Zielinformation des Datenrahmens
	CW_{min}	$\mathbb{N}$	untere Schranke des Backoff
	CW_{max}	$\mathbb{N}$	obere Schranke des Backoff

Tabelle 5.3.: Konfigurationsparameter einer einzelnen Übertragung in Abhängigkeit des Kommunikationsprotokolls

davon, ob zuvor Daten zur Verfügung gestellt wurden, nimmt die Station aktiv an der Kommunikation teil, leitet Daten weiter (z. B. ACTP) oder agiert ausschließlich als Empfänger.

- Callback `TXDone` ($at\colon \mathcal{T},\ success\colon \mathbb{B}$)
  wird ausgelöst, wenn eine Übertragung abgeschlossen wurde. Der Parameter *success* gibt den Erfolg der Übertragung an.
- Callback `RX` ($at\colon \mathcal{T},\ data\colon \langle ProtocolSpecific \rangle$)
  wird ausgelöst, wenn Daten empfangen wurden. Das Format der Daten ist wieder protokollspezifisch.

Die genaue Semantik der Callbacks `TXDone` und `RX` ist vom jeweiligen Protokoll abhängig.

5.2.6 Schicht 4: Multiplexer

Der Multiplexer verwaltet alle Sende- und Empfangsaufträge innerhalb der Makroslots. Von der Anwendung erhält der Multiplexer Daten für eine Transmission Opportunity, die er dem jeweiligen Slot zuordnet. Auf Protokollseite stellt er die Daten dem jeweiligen Protokolltreiber vor dessen Aktivierung zur Verfügung. Der Mutliplexer stellt daher den Datenpfad für die Kommunikation dar.

Schnittstelle zur Anwendung: Die Daten von bzw. zur Anwendung werden immer auf Basis von Transmission Opportunities adressiert. Somit ist die konkrete Region bzw. der Slot für die Anwendung transparent. Die Abbildung zwischen Transmission Opportunity und Makroslotkonfiguration wird durch den Multiplexer vorgenommen. Die Schnittstelle ist wie folgt aufgebaut:

- `enqueue` ($opportunity\colon \langle$`Opportunity`$\rangle,\ data\colon \langle ProtocolSpecific \rangle,$
  $configuration\colon \langle ProtocolSpecific \rangle,\ maxTries\colon \mathbb{N}) : \mathcal{R}$
  nimmt Daten für die Transmission Opportunity *opportunity* entgegen. Das Format der Daten und der Konfiguration ist protokollspezifisch. Bei der Konfiguration handelt es sich um einen Teil der in Tab. 5.3 gelisteten Parametern. Einige dieser Parameter (wie z. B. der Arbitrierungsradius bei ACTP) sind durch die Makroslotkonfiguration gegeben, andere (wie z. B. die Zieladresse bei wettbewerbsbasierten Übertragungen) werden `enqueue` mitgegeben.

  Der Parameter $maxTries$ gibt die Anzahl an Wiederholungsversuchen an, falls ein Auftrag nicht erfolgreich verarbeitet werden konnte. Bei der Realisierung von TOD [43] (*Try Once, Discard*) mittels AB würde $maxTries$ mit 1 konfiguriert werden.

  Je nach Protokoll werden die Aufträge einer Transmission Opportunity nach einem FIFO-Prinzip oder nach einem prioritätsbasierten Schema abgearbeitet.

  Der Rückgabewert der Funktion ist eine eindeutige Identifikation des Sendeauftrags. Über diesen kann die Anwendung in den Callbacks entscheiden, um welchen Auftrag es sich gehandelt hat.
- Callback `TXDone` ($request\colon \mathcal{R},\ success\colon \mathbb{B}$)
  wird ausgelöst, wenn ein Auftrag mit der Identifikation *request* abgearbeitet wurde. Der Parameter *success* gibt den Erfolg an.

- Callback `RX` (*data*: ⟨`ProtocolSpecific`⟩)
  wird ausgelöst, wenn Daten empfangen wurden. Zu jeder Transmission Opportunity wird ein eigener Callback registriert, der dann entsprechend aufgerufen wird.

Der Multiplexer ist stark mit dem Scheduler verzahnt und kann nicht eigenständig verwendet werden.

5.2.7 Schicht V1: Scheduler

Der Scheduler übernimmt die zeitliche Abarbeitung eines Makroslots. Zu Beginn des Makroslots (und somit zu Beginn des SyncSlots) triggert der Scheduler eine Resynchronisation mit Hilfe der BBS-Komponente. In den darauf folgenden Slotregionen werden entsprechend der Makroslotkonfiguration Übertragungen mit Hilfe der entsprechenden Protokolltreiber durchgeführt. Der Scheduler stellt so den Kontrollpfad für die Kommunikation dar.

Darüber hinaus verwaltet der Scheduler auch die Energiemodi des CC 2420-Transceivers. Aufgrund der Makroslotkonfiguration kann der Scheduler entscheiden, zu welchen Zeitpunkten der Transceiver nicht benötigt wird (z. B. Idle-Regionen), und diesen abschalten, um Energie zu sparen (*Duty-Cycling*).

5.2.8 Schicht V2: Makroslotkonfiguration

Diese Komponente verwaltet die Konfiguration des Makroslots. Zu dieser gehören die statischen Parameter (Tab. 5.2) der Kommunikation, die Anzahl und Länge von Mikroslots, die Gruppierung in virtuelle Slotregionen und die Gruppierung von Slots bzw. Regionen in Transmission Opportunities.

Die Konfiguration wird initial statisch, also beim Entwurf der Anwendung, festgelegt. Eine sinnvolle Erweiterung der Schicht ist eine dynamische Anpassbarkeit. So kann eine Reservierungskomponente bei Bedarf Reservierungen mit anderen Knoten aushandeln und zusätzliche Reservierungsslots in die Makroslotkonfiguration einbringen.

5.2.9 Interaktion der Komponenten

In den vorherigen Abschnitten wurden die einzelnen Komponenten des Frameworks vorgestellt. Im Folgenden wird die Interaktion der Komponenten beschrieben. Eine zentrale Rolle spielt dabei der Scheduler, da dieser die zeitliche Abarbeitung des Makroslots übernimmt. Vom Scheduler gehen folglich alle Protokollaktivierungen und alle Aktivitäten aus, die eine Übertragung oder einen Empfang einleiten.

Resynchronisation durch BBS: Zu Beginn des Makroslots wird eine Resynchronisation durchgeführt. Der Verlauf dieser ist für einen Slave-Knoten in Abb. 5.4 gezeigt. Der Scheduler aktiviert die BBS-Komponente, die wiederum die Black Burst-Komponente aktiviert. Dadurch wird die Hardware über den CC 2420-Treiber für eine Black Burst-basierte Kommunikation umkonfiguriert.

Durch den Aufruf von `sync` wird der BBS-Komponente der Startzeitpunkt der Resynchronisation mitgeteilt. Da die Konvergenzdauer bekannt ist, kann die Komponente einen Timer auf das Ende der Synchronisationsphase ($t_{phase\,end}$) setzen. Liefe dieser Timer ab, wäre der Knoten in dieser Phase nicht synchronisiert.

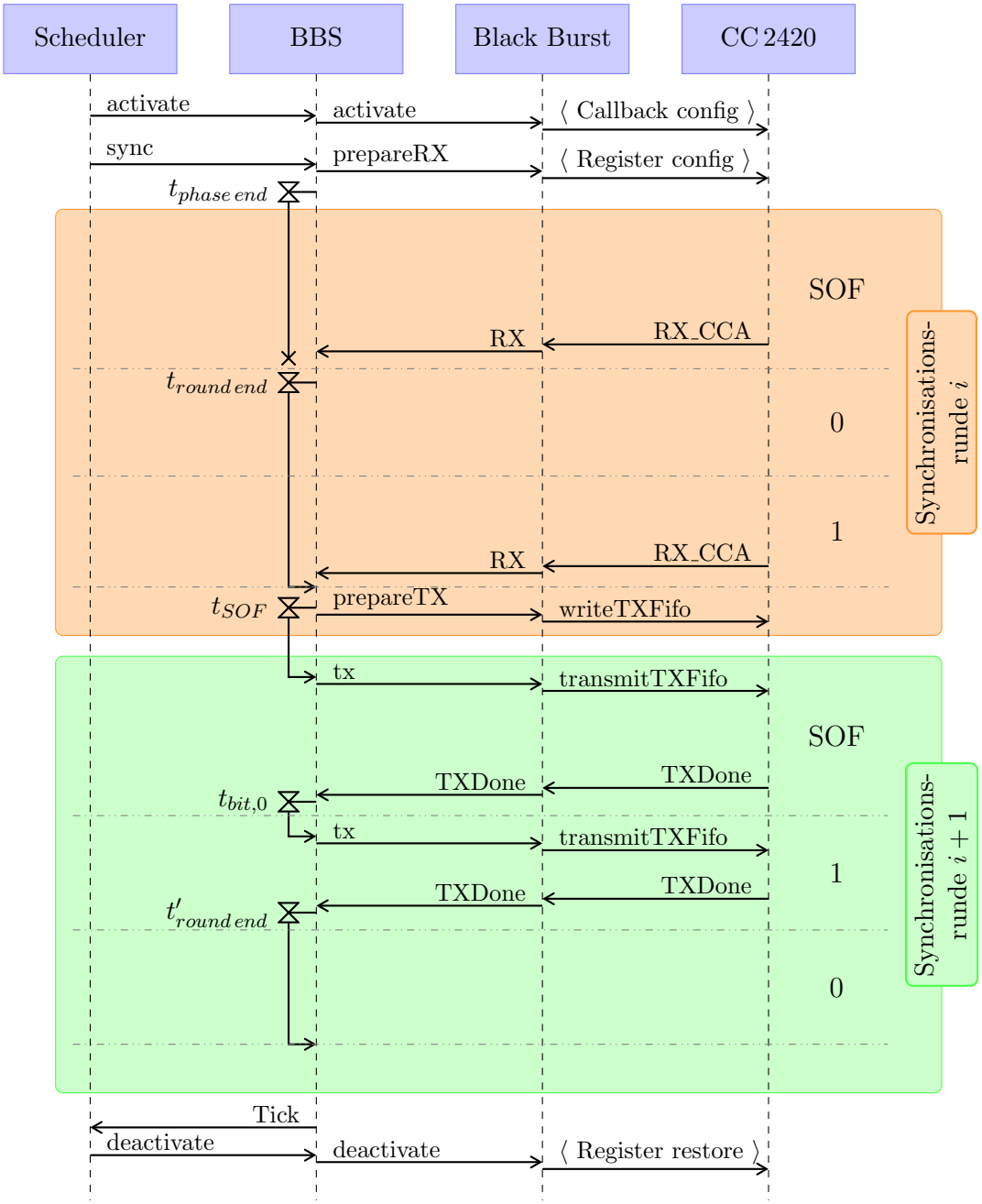


Abbildung 5.4.: Schematischer Ablauf des Master-basierten BBS-Verfahrens aus Sicht eines Slaves im Kontext des Frameworks

Der Timer wird jedoch durch den Empfang eines SOF-Black Bursts in einer Runde i unterbrochen. Die BBS-Komponente setzt einen Timer auf das Rundenende. Bei eintreffenden Black Bursts wird die Bitposition innerhalb des Tickrahmens berechnet und die entsprechende Position als „dominant“ markiert.

Am Ende der Runde wird der empfangene Tickrahmen dekodiert und das Paritätsbit geprüft. Ist dieses korrekt, ist der Knoten synchronisiert und kennt die momentane Rundennummer. Im Beispiel folgt eine weitere Synchronisationsrunde, in der der Slave selbst einen Tickrahmen sendet.

In dieser wird die `tx`-Funktion des Black Burst-Treibers genutzt, um das SOF und den Black Burst, die den Tickrahmen kodieren, zu übertragen. Am Ende der Synchronisation wird das `Tick`-Callback ausgelöst und dem Scheduler die erfolgreiche Resynchronisation sowie den (durch die Synchronisation bestimmten) Tickzeitpunkt, die Distanz zum Master, etc. mitgeteilt.

Anhand des Sequenzdiagramms ist auch die klare Schichtentrennung der Komponenten sichtbar: Die involvierten Komponenten sprechen immer nur die Schnittstellen der benachbarten Schicht an.

Abarbeitung von Slotregionen: Zu Beginn jeder Slotregion aktiviert und konfiguriert der Scheduler den entsprechenden Protokolltreiber (Schnittstelle `prepare` bzw. `setConfig`). Falls bei der anstehenden Kommunikation eigene Daten versendet werden sollen, werden diese vom Multiplexer an das Protokoll gereicht (`setData`). Ähnlich wie bei der BBS-Komponente wird der Startzeitpunkt der Übertragung über die Funktion `schedule` mitgeteilt.

Daten, die während der Abarbeitung des jeweiligen Protokolls empfangen werden sowie abgeschlossene Sendeereignisse werden an den Multiplexer und anhand der aktuellen Transmission Opportunity an die betroffene Anwendung durchgereicht.

Am Ende der Slotregion wird die Protokollkomponente durch den Scheduler deaktiviert (`teardown`).

5.3 Implementierung des Kommunikationsframeworks

Im Rahmen der Arbeit wurde ein Teil des Frameworks für die Imote 2-Sensorplattform implementiert. Dieser Teil ist in Abb. 5.5 farblich markiert. Die Schnittstellen der einzelnen Komponenten folgen den Beschreibungen des vorherigen Kapitels. Durch das Festlegen einer konkreten Programmiersprache mussten an einigen Stellen die Datentypen geeignet eingeschränkt und ggf. weitere Parameter hinzugefügt werden.

Durch die lose Kopplung der Komponenten konnte eine Bottom-Up Implementierung gewählt werden. Somit konnte zunächst die Systemsoftware getestet werden, dann darauf aufbauend der CC 2420-Treiber, der Black Burst-Treiber und schließlich die Implementierung des Master-basierten BBS-Verfahrens. Diese Implementierung war auch die Grundlage der Experimente und Optimierungen in Kap. 4 und der Evaluierung im nächsten Kapitel.

In diesem Abschnitt werden einige Aspekte der Implementierung des Master-basierten BBS-Verfahrens vorgestellt. Der Fokus liegt hierbei weniger auf technischen Details, sondern auf der konkreten Bestimmung der Größen, die für das Funktionieren des Verfahrens nötig sind.

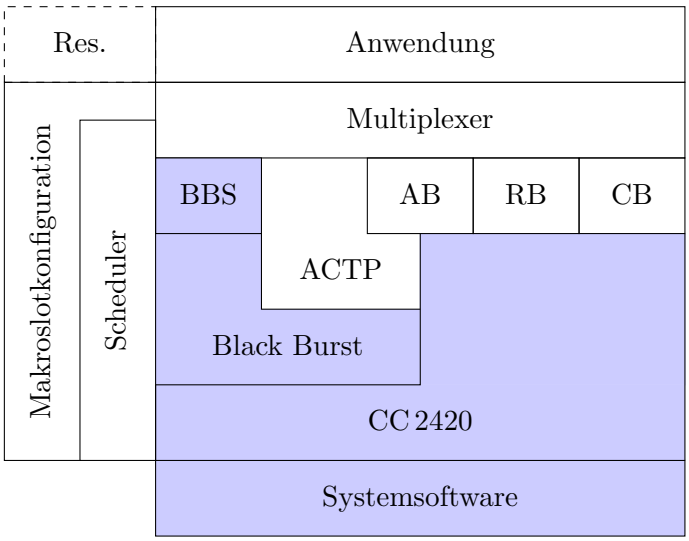


Abbildung 5.5.: Fortschritt der Implementierung des Frameworks. Farblich markierte Komponenten wurde im Zuge der Arbeit implementiert

5.3.1 Implementierung des Master-basierten BBS-Verfahrens

In Kap. 2.3 wurde das Master-basierte BBS-Verfahren vorgestellt. Dabei wurden einige Bedingungen in Form mathematischer Ungleichungen formuliert, die für ein erfolgreiches Funktionieren des Verfahrens erfüllt sein müssen. Die Forderungen beziehen sich vor allem auf Verzögerungen, die durch die Hardware gegeben sind. Mit den Messungen in Kap. 4 konnten einige der gesuchten Verzögerungen bestimmt werden. Für weitere Verzögerungen mit variablen Anteilen konnten Größenordnungen und Abschätzungen gefunden werden. In diesem Abschnitt werden konkrete Werte für die gesuchten Größen gegeben, die in der Implementierung des Verfahrens verwendet und im folgenden Kapitel evaluiert werden.

5.3.1.1 Bestimmung der Bitdauer

In Kap. 2.3.4.1 wurden die Forderungen an die Bitdauer d_{Bit} hergeleitet. Dabei wurde die minimale Bitdauer unter verschiedenen Aspekten betrachtet. Diese werden in diesem Abschnitt genauer analysiert und die involvierten Zeitspannen mit Werten belegt.

Dauer des Sendevorgangs: Formel. 2.3 stellt eine Forderung an d_{Bit}, die ein erfolgreiches Senden mehrerer konsekutiver Black Bursts garantiert. Die einzelnen Verzögerungen dieser Forderung sind:

- Die Dauer zur Vorbereitung einer Übertragung $d_{TX,pre}$.

  Das Schreiben eines IEEE 802.15.4-Rahmens in den Puffer des Transceivers über den SPI-Bus wird im Wesentlichen durch zwei Parameter gegeben: Die Übertragungsgeschwindigkeit des SPI-Busses und die Zugriffsverzögerung auf den Bus. Die Länge der PHY-Nutzdaten, die für die Implementierung von Black Bursts genutzt wird, ist 1 Byte. Das SOF-Bit hat zwar eine größere Länge, muss jedoch in dieser Betrachtung nicht beachtet werden, da ein SOF-Bit nie unmittelbar einem regulären

Black Burst folgt. Zusätzlich muss das Längenbyte des Rahmens und das Kommando an den CC 2420 übertragen werden. Die drei Bytes werden mit einem Bustakt von 6,5 $^{\text{Mbit}}/_{\text{s}}$ in ungefähr 3,7 µs übertragen. Die Zugriffsverzögerung wurde noch nicht näher bestimmt, wird aber mit etwa 5 µs angenommen.

Somit gilt $d_{TX,pre} \approx 8{,}7\,\mu s$.

- d_{TX} ist die Dauer, die vom Auslösen der Unterbrechung (durch einen Timer) bis zu dem Zeitpunkt vergeht, zu dem der `STXON`-Befehl über den SPI-Bus zum Transceiver gegeben wurde.

  Die Verzögerung durch die Unterbrechungsbehandlung beträgt nach den Erkenntnissen aus Kap. 4.5.2.5 etwa 5 µs. Hinzu kommt wieder ein SPI-Transfer mit etwa 6,23 µs.

  Insgesamt gilt also $d_{TX} \approx 11{,}23\,\mu s$.

- Die Umschaltdauer d_{rxtx} wird basierend auf den Messungen in Kap. 4.4.1 durch die Umschaltverzögerung Δ_{rxtx} ersetzt und mit einer oberen Schranke von 129,01 µs angenommen.

- Die Dauer eines Black Bursts d_{BB} wurde in Kap. 3.2.2.1 auf 160 µs festgelegt.

Somit ergibt sich folgende Forderung an d_{Bit}:

$$\begin{aligned} d_{Bit} &\overset{(2.3)}{\geq} d_{TX,pre} + d_{TX} + d_{rxtx} + d_{BB} \\ &\approx 8{,}7\,\mu s + 11{,}23\,\mu s + 129{,}01\,\mu s + 160\,\mu s \\ &\approx 309\,\mu s \end{aligned} \tag{5.1}$$

Dauer des Empfangsvorgangs: Die zweite Forderung an d_{Bit} beschreibt die Verzögerungen bei der Erkennung eines Black Bursts und wird in Formel 2.4 angegeben. Die einzelnen Verzögerungen sind:

- Das maximale Tickoffset während einer Synchronisationsphase $d_{maxBaseTickOffset}$ wird nach den Überlegungen aus Kap. 2.3.4.3 durch $d_{maxBaseTickOffset,local} = 2 \cdot \widehat{\delta}_{tr}$ ersetzt. Die Dauer $\widehat{\delta}_{tr}$ wird im folgenden Abschnitt berechnet.

- Die Dauer eines Black Bursts d_{BB} beträgt weiterhin 160 µs.

- Die maximale CCA-Verzögerung ist mit $d_{maxCCA} = 128\,\mu s$ gegeben.

- Die Verarbeitungsverzögerung d_{RX} bei der Erkennung eines Black Bursts setzt sich aus der Verzögerung der Unterbrechungsbehandlung und der weiteren Verarbeitungsdauer zusammen. Insgesamt wird diese mit $d_{RX} \approx 15\,\mu s$ abgeschätzt.

Zusammengefasst ergibt sich folgende Forderung:

$$\begin{aligned} d_{Bit} &\overset{(2.4)}{\geq} d_{maxBaseTickOffset} + d_{BB} + d_{maxCCA} + d_{RX} \\ &\approx 258\,\mu s + 160\,\mu s + 128\,\mu s + 15\,\mu s \\ &\approx 561\,\mu s \end{aligned} \tag{5.2}$$

Dieser Wert entspricht auch der maximalen Erkennungsdauer eines Black Bursts $d_{BB,max}$. In Kapitel 3.2.2.2 wurde die Dauer eines SOF-Black Bursts, unter der Annahme, dass die Verarbeitungsverzögerung d_{RX} kleiner als 32 µs ist, mit $d_{BB\_SOF} = 576\,\mu s$ bestimmt. Diese Annahme ist erfüllt und es ist nun ersichtlich, dass $d_{BB\_SOF} > d_{BB,max}$ gilt.

Eindeutigkeit der Bitzuordnung beim Empfänger: Die dritte Forderung an die Bitdauer muss erfüllt sein, um die korrekte Zuordnung eines empfangenen Black Bursts zur Bitposition innerhalb des Tickrahmens zu gewährleisten. Es handelt sich dabei um die maximale variable Verzögerung $\widehat{\delta}_{tr}$, die zwischen dem Sende- und Empfangsereignis eines Black Bursts liegt. Die Teilverzögerungen von $\widehat{\delta}_{tr}$ sind:

- $d_{TX}^{\mathcal{V}}$ ist die maximale variable Verzögerung beim Sendevorgang.

  Ein Teil davon ist durch die Unterbrechungsbehandlung gegeben. Wird der Spinning-Mechanismus (Kap. 4.5.2.2) verwendet, kann die obere Schranke des variablen Anteils der Behandlung mit 80 ns angenommen werden.

  Hinzu kommt die variable Verzögerung des Transceivers, die durch das Verarbeiten des SPI-Befehls entsteht. Die obere Schranke wurde in Kap. 4.4.1 auf 160 ns (inklusive Messungenauigkeit) bestimmt.

- Die maximale CCA-Verzögerung von $d_{maxCCA} = 128\,\mu s$.

- Der variable Anteil der Verarbeitungsverzögerung beim Erkennen des Black Bursts. Dabei wird jedoch nicht die gesamte Verzögerung betrachtet, sondern nur der Teil, der bis zum Nehmen des Zeitstempels erforderlich ist. Dieser beträgt $d_{RX,intr}^{\mathcal{V}} = 80\,ns$.

Somit gilt:

$$\begin{aligned}\widehat{\delta}_{tr} &\overset{(2.7)}{=} d_{TX}^{\mathcal{V}} + d_{maxCCA} + d_{RX,intr}^{\mathcal{V}} \\ &= 80\,ns + 160\,ns + 128\,\mu s + 80\,ns \\ &\approx 129\,\mu s \end{aligned} \tag{5.3}$$

Für die Forderung an die Bitdauer gilt entsprechend:

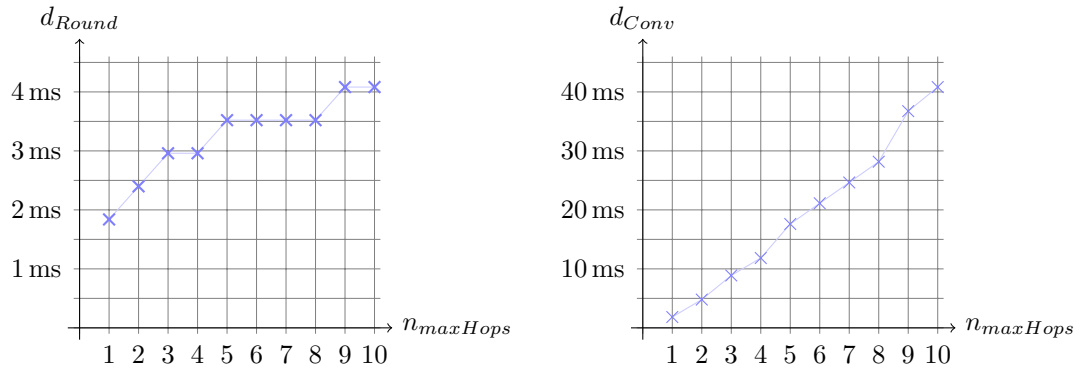
$$d_{Bit} \overset{(2.8)}{>} 2 \cdot \widehat{\delta}_{tr} \tag{5.4}$$

$$\overset{(5.3)}{\approx} 258\,\mu s \tag{5.5}$$

Zusammenfassung: Damit alle Forderungen an die Bitdauer d_{Bit} erfüllt sind, muss die größte Verzögerung gewählt werden. Diese ist durch Verzögerungen beim Empfang in Formel 5.2 gegeben. Für die Dauer eines regulären Bits und des SOF-Bits (Formel 2.10) gilt somit:

$$d_{Bit} = 561\,\mu s \tag{5.6}$$

$$d_{SOF} = 977\,\mu s \tag{5.7}$$



(a) Rundendauer in Abhängigkeit von $n_{maxHops}$ (b) Konvergenzdauer in Abhängigkeit von $n_{maxHops}$

Abbildung 5.6.: Runden- und Konvergenzdauer von BBS bei $d_{Bit} = 561\,\mu s$, $d_{SOF} = 977\,\mu s$ und $d_{frameGuard} = 300\,\mu s$

5.3.1.2 Bestimmung von Runden- und Konvergenzdauer

Neben der Bitdauer wurden in Kap. 2.3.4 auch die Rundendauer d_{Round} und Konvergenzdauer d_{Conv} analysiert. Unter Beachtung der zuvor bestimmten Bitdauer können diese Größen nun in Abhängigkeit des maximalen Netzwerkdurchmessers $n_{maxHops}$ angegeben werden. Für die Verarbeitungsdauer eines Tickrahmens wurde $d_{frameGuard} = 300\,\mu s$ gewählt. Die Bit- und Rundendauer sind für $n_{maxHops} \in \{1, \ldots, 10\}$ in Abb. 5.6 gezeigt.

5.4 Zusammenfassung

In diesem Kapitel wurde ein Kommunikationsframework für den Mehrfachzugriff auf ein Drahtlosmedium entworfen, das die Aspekte Black Burst-basierter Kommunikation vereint. Dazu wurden mehrere Komponenten und deren Schnittstellen vorgestellt, die in Form einer Schichtenarchitektur aufeinander aufbauen. Die lose Kopplung zwischen den Komponenten ermöglicht es, nur einzelne Komponenten des Frameworks zu verwenden, so dass die Evaluierung neuer Protokolle oder weiterer Optimierungen erleichtert wird.

Darüber hinaus wurde ein Teil des Frameworks implementiert. Als erstes Kommunikationsprotokoll wurde dabei das Master-basierte BBS-Verfahren implementiert. Weitere Protokolle und Komponenten (wie z. B. Multiplexer und Scheduler) können auf diesem Grundgerüst aufbauen.

6. Kapitel

Evaluation

In Kap. 5.3 wurde die Implementierung des Master-basierten BBS-Verfahrens im Kontext des Kommunikationsframeworks vorgestellt. Anhand dieser Implementierung wurden verschiedene Evaluationen durchgeführt. Neben einer Funktionsprüfung wurden auch die erreichte Synchronisationsgenauigkeit gemessen und Optimierungen gegenübergestellt.

In diesem Kapitel werden zwei Experimente vorgestellt, die die Funktionalität der Implementierung prüfen. In einer Messung wird darüber hinaus die erreichte Synchronisationsgenauigkeit untersucht.

6.1 Messung des Tick-Offsets

In dieser Messung wird die Synchronisationsgenauigkeit bestimmt, die mit der Implementierung erreicht werden konnte. Dazu wird die Zeitspanne $\Delta_{baseTickOffset,i}$ bestimmt, die die Abweichung des Referenzzeitpunkts zwischen Master- und Slave-Knoten i angibt. Die Messung findet in Netzwerken mit einem Durchmesser von einem Hop statt, d. h. alle Knoten können direkt miteinander kommunizieren.

Zusätzlich wird der Einfluss des Erkennungsmechanismus von Black Bursts untersucht. Kapitel 2.2.4 beschreibt, dass ein Black Burst, der nur von einer Station gesendet wird, mit Hilfe des SFD-Mechanismus des Transceivers erkannt werden kann. In BBS wird dies für das SOF-Bit der ersten Runde verwendet (Kap. 2.3.3), um eine genauere Synchronisation auf dem ersten Hop zu erzielen. Ziel der Messung ist die Angabe konkreter Werte bezüglich des Tick-Offsets in Abhängigkeit des gewählten Erkennungsmechanismus.

Bei diesem Experiment sollte beachtet werden, dass die Implementierung zum Zeitpunkt der Evaluation noch nicht alle Aspekte aus Kap. 5.3.1 beinhaltet. Insbesondere wurden für die involvierten Zeitspannen (vor allem die variablen Verzögerungen bei der Unterbrechungsbehandlung) konservativere Werte angenommen, so dass auch die erreichte Synchronisationsgenauigkeit schlechter ist als das theoretische Maximum, das für die Werte in Kap. 5.3.1 berechnet werden kann.

Aufbau und Durchführung: In der Messung werden ein Master- und zwei Slaveknoten eingesetzt. Abbildung 6.1 zeigt den schematischen Aufbau der Messung.

Der Master-Knoten synchronisiert in jedem Messdurchlauf zwei Slaves in der ersten Runde. Alle drei Knoten geben zum Ende der Synchronisationsphase ein Ereignis über einen GPIO-Pin des Prozessors aus. Diese Ereignisse werden durch den Logic Analyzer

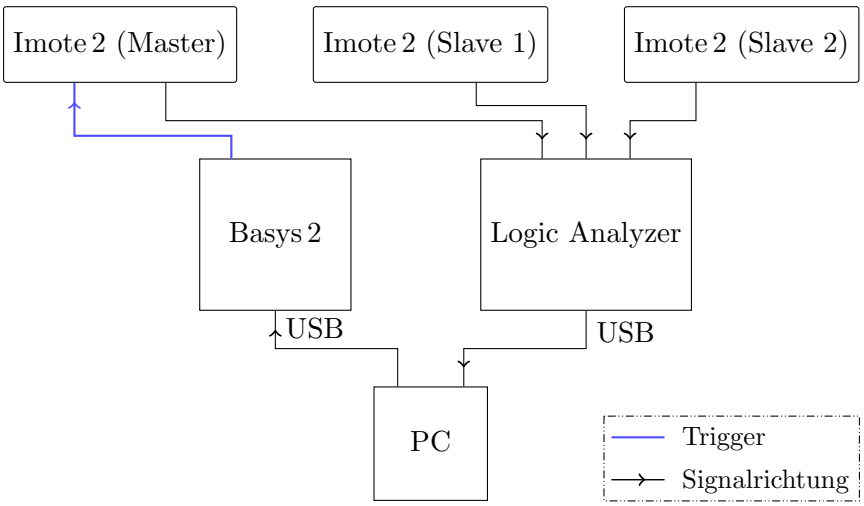


Abbildung 6.1.: Schematischer Messaufbau zur Messung des Tick-Offsets

erfasst. Das Tick-Offset wird in der Auswertung als zeitlicher Abstand des lokalen Ticks des Master-Knotens Master-Ticks zu dem lokalen Tick der Slave Knoten bestimmt. Ferner kann über die Anzahl der Ereignisse die Erfolgsrate der Synchronisation analysiert werden.

Die Messung wird in mehreren Variationen durchgeführt. Zunächst wird der CCA-Erkennungsmechanismus eingesetzt und 100 Messdurchläufe bei einer räumlichen Entfernung der Slave-Knoten zum Master von etwa 7 cm durchgeführt. Danach wird die Messung mit einer Entfernung von 2,5 m wiederholt. Für beide Entfernungen werden anschließend die Experimente mit dem SFD-Erkennungsmechanismus durchgeführt. Somit ergeben sich insgesamt vier Messvariationen mit jeweils 100 Messdurchläufen.

Ergebnis und Beobachtungen: Tabelle 6.1 gibt die statistische Auswertung des Tick-Offsets zwischen dem Master und jedem Slave an. Darin sind folgende Beobachtungen möglich:

- Die Erkennung von Black Bursts mit dem SFD-Mechanismus ist, wie erwartet, genauer als die Erkennung mit dem CCA-Mechanismus. Im Durchschnitt werden beim SFD-Mechanismus Genauigkeiten von rund 8 µs erreicht, beim CCA-Mechanismus etwa 30 µs bis 45 µs. Auch die Schwankungen sind beim SFD-Mechanismus mit rund 1 µs wesentlich geringer als beim CCA-Mechanismus mit 18 µs.

- Bei Verwendung des CCA-Mechanismus hat die Entfernung der Knoten einen wesentlich höheren Einfluss auf das Tick-Offset als bei Verwendung des SFD-Mechanismus. Während beim SFD-Mechanismus Durchschnitt, Schwankung und Standardabweichung in der selben Größenordnung liegen, beeinflusst beim CCA-Mechanismus die Entfernung diese statistischen Parameter gerade bei Betrachtung des zweiten Slaves stark.

- In allen Variationen konnten beide Slaves in allen Wiederholungen erfolgreich synchronisiert werden. Die Implementierung zeigt – zumindest in den gegeben Versuchsaufbauten – eine hohe Zuverlässigkeit.

Erkennung	Entfernung	Slave	Minimum	Maximum	Spannweite	Durchschnitt	Standard-Abweichung	Erfolgsrate
CCA	7cm	1	19,43 µs	36,92 µs	17,49 µs	32,43 µs	3,2 µs	100%
CCA	7cm	2	18,37 µs	35,62 µs	17,25 µs	31,29 µs	5,16 µs	100%
CCA	2,5m	1	29,34 µs	49,89 µs	20,55 µs	39,54 µs	5,16 µs	100%
CCA	2,5m	2	30,23 µs	94,49 µs	64,26 µs	45,55 µs	8,1 µs	100%
SFD	7cm	1	7,85 µs	9,03 µs	1,18 µs	8,48 µs	0,3 µs	100%
SFD	7cm	2	7,77 µs	8,84 µs	1,07 µs	8,31 µs	0,29 µs	100%
SFD	2,5m	1	7,84 µs	8,95 µs	1,11 µs	8,39 µs	0,3 µs	100%
SFD	2,5m	2	7,79 µs	8,88 µs	1,09 µs	8,29 µs	0,29 µs	100%

Tabelle 6.1.: Statistische Auswertung des Tick-Offsets in allen Messvariationen

6.2 Messungen zur Dekodierung von BBS-Tickrahmen

Der Vorteil Black Burst-basierter Kommunikation ist, dass sich Black Bursts ohne desktruktive Kollisionen überlagern. Daraus ergibt sich auch der Vorteil, dass mit BBS eine Synchronisation ohne Kenntnis der genauen Topologie durchgeführt werden kann.

Im folgenden Experiment soll die Robustheit der Implementierung des BBS-Verfahrens im Hinblick auf überlagerte Tickrahmen getestet werden. Implizit wird dabei auch die als Basis dienende Black Burst-Komponente getestet.

Aufbau und Durchführung: Der Aufbau gleicht im Wesentlichen dem vorherigen Aufbau aus Abb. 6.1. Hinzu kommt ein vierter Imote 2, der eine besondere Monitor-Rolle übernimmt. Der Monitor-Knoten nimmt nie aktiv an der Synchronisation teil, sondern dekodiert alle erkannten Tickrahmen, ohne diese weiterzuverarbeiten.

Durch wenige Anpassungen im Code wird folgender Testdurchlauf durchgeführt:

1. In der ersten Runde sendet, wie im Master-basierten BBS-Verfahren üblich, nur der Master einen Tickrahmen mit Rundennummer $n_{Round} = 0$. Dieser wird von den beiden Slaves empfangen, aber nur von einem verarbeitet. Der zweite Slave verwirft den Rahmen, da er in diesem Experiment durch einen Tickrahmen synchronisiert werden soll, der von mehreren Knoten übertragen wird.

2. In der zweiten Runde senden der Master und der bereits synchronisierte Slave den Tickrahmen und synchronisieren den zweiten Slave hierdurch.

3. In der dritten Runde senden alle drei Knoten den entsprechenden Tickrahmen.

Somit senden in der ersten Runde nur ein, in der zweiten Runde zwei und in der dritten Runde drei Knoten. Jeder Knoten gibt zum Ende der Synchronisationsphase ein Ereignis über einen GPIO-Pin des Prozessors aus. Die Ereignisse werden vom Logic Analyzer mitgeschnitten.

So kann im Nachhinein nachvollzogen werden, ob die Knoten auch tatsächlich nach obigem Schema synchronisiert wurden.

Der Monitor-Knoten empfängt alle drei Tickrahmen und gibt den Erfolg der Dekodierung (Korrekt bzw. nicht korrekt) über die serielle Schnittstelle aus. Mit den so erhobenen Daten wird die Erfolgsrate beim Dekodieren eines Tickrahmens mit einer variierenden Anzahl an gleichzeitigen Sendern abgeleitet.

Der Messvorgang wird 100 mal wiederholt.

Ergebnis: In allen Messdurchläufen wurden beide Slaves in der oben beschriebenen Reihenfolge synchronisiert. Das bedeutet insbesondere auch, dass der zweite Slave immer durch überlagerte Tickrahmen synchronisiert wurde. Der Monitor-Knoten konnte überdies die Tickrahmen aller Runden in allen Messdurchläufen erfolgreich dekodieren. Es konnte gezeigt werden, dass die Kollisionsresistenz von der Implementierung – zumindest in den durchgeführten Evaluationen – korrekt umgesetzt wird.

7. Kapitel

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein umfassender Überblick über Black Burst-basierte Kommunikation gegeben. Viele Aspekte der Thematik wurden dabei im Detail behandelt. Weiter wurden Optimierungen erarbeitet, implementiert und evaluiert sowie Unzulänglichkeiten der Hardware identifiziert und quantifiziert.

In diesem Kapitel wird eine abschließende Zusammenfassung über die Arbeit gegeben und im Ausblick einige Ideen gegeben, in welche Richtung die Entwicklung, Implementierung und Evaluationen weitergeführt werden kann.

Die einzelnen Kapitel der Arbeit befassen sich mit verschiedenen Aspekten von Black Bursts und der darauf aufbauenden Protokolle. Diese wurden schließlich im Design und der Implementierung eines Kommunikationsframeworks beachtet, das im Speziellen auch die Unzulänglichkeiten der verwendeten Hardware berücksichtigt.

In Kap. 2 wurden die Unzulänglichkeiten bestehender Mehrfachzugriffsverfahren und die Grundlagen Black Burst-basierter Kommunikation vorgestellt. Darauf aufbauend wurde das Master-basierte Black Burst Synchronization-Verfahren im Detail behandelt. In diesem Kontext wurde mit der Erkennung von Black Bursts anhand des SFDs eine Optimierung angegeben, die sich zwar auf wenige Spezialfälle beschränkt, jedoch bei der Synchronisation des ersten Hops eine höhere Genauigkeit erzielt. Ferner wurde die Bitdauer neu hergeleitet, wobei besonders auf Verzögerungen durch die Hardware geachtet wurde. Des Weiteren wurden auch die Formeln zur Berechnung von Runden- und Konvergenzdauer sowie dem maximalen initialen Tick-Offset entsprechend überarbeitet.

Kapitel 3 gibt einen detaillierten Einblick in die verwendete Sensorknoten-Plattform Imote 2. Dabei wurde zunächst die Funkhardware, der CC 2420-Transceiver, vorgestellt. Da dieser für den Einsatz in IEEE 802.15.4-Netzwerken ausgelegt ist, wurde auf einige Aspekte des IEEE-Standards näher eingegangen und dabei auch auf Abweichungen des CC 2420-Transceivers vom Standard angemerkt. In diesem Kontext wurde auch die Implementierung von Black Bursts durch IEEE 802.15.4-Rahmen vorgestellt. Außerdem wurde in diesem Kapitel der PXA 271-Prozessor beschrieben. Zu den Komponenten der Kernhardware (Hauptprozessor, Caches, Speicherabbildung, etc) wurden auch verschiedene Optimierungsmöglichkeiten angegeben, die darauf abzielen, variable Schwankungen bei der Ausführung von Code und insbesondere bei Interrupts zu minimieren.

In Kap. 4 wurden einige Experimente durchgeführt, um Verzögerungen der CC 2420-Funkhardware zu bestimmen, die einen Einfluss auf die Funktion Black Burst-basierter Protokolle haben. Darüber hinaus wurden auch das Verhalten der Funkhardware geprüft

und vom Datenblatt abweichendes Verhalten dokumentiert. Ferner wurden auch einige Zeitspannen wie z. B. die Verzögerungen bei Eintreten eines Ereignisses bestimmt, die durch die Verwendung des PXA 271-Prozessors gegeben sind. Insbesondere wurden dabei auch die Vorteile durch die in Kap. 3 vorgestellten Optimierungen herausgearbeitet. Im Zuge der Evaluationen wurde eine Messumgebung entwickelt, die der Erfassung und Auswertung von Hardware-Verzögerungen dient. Diese Hard- und Software-Umgebung kann in zukünftigen Evaluationen wiederverwendet werden.

Aufbauend auf den Konzepten und Erkenntnissen der vorhergehenden Kapiteln wurde in Kap. 5 ein Kommunikationsframework entworfen, das sämtliche Aspekte Black Burst-basierter Kommunikation integriert. Aufgrund der Trennung der Aspekte in aufeinander aufbauende Schichten und der losen Kopplung zwischen den Komponenten des Frameworks ist es möglich, nur Teile des Frameworks zu verwenden. Dies ist insbesondere bei der Evaluierung neuer Konzepte vorteilhaft, da diese somit unabhängig vom ggf. irrelevanten Teil des Frameworks implementiert und getestet werden können. Ein Teil des Frameworks wurde im Rahmen der Arbeit implementiert.

In Kap. 6 wurde diese Implementierung evaluiert. Hierbei wurden sehr gute Ergebnisse bezüglich der Erfolgsrate und Genauigkeit erzielt, was auf eine hohe Robustheit und Zuverlässigkeit des Verfahrens und der Implementierung schließen lässt.

Obwohl bereits sehr viele Verzögerungsquellen in dieser Arbeit analysiert wurden, ist die Analyse keineswegs abgeschlossen. In weiterführenden Arbeiten müssen noch weitere Verzögerungsquellen (z. B. SPI-Zugriffe und SPI-Übertragungen) identifiziert und deren Ausmaß ähnlich wie in den Experimenten dieser Arbeit quantifiziert werden. Dadurch ergeben sich genauere Schranken, so dass das initiale Tick-Offset genauer bestimmt und die Robustheit der Verfahren weiter gesteigert werden kann.

Das Framework wurde in dieser Arbeit nur zum Teil implementiert. Die verbleibenden Komponenten müssen in zukünftigen Arbeiten integriert werden. Hierzu müssen auch die entsprechenden Protokolle (insbesondere ACTP) unter den Gesichtspunkten von Hardwarelaufzeiten und deren Schwankungen neu evaluiert werden.

Ferner wurden auch nicht alle Konzepte des Frameworks im Detail ausgearbeitet. So bleibt z. B. offen, wie die wettbewerbsbasierte Kommunikation stattfindet, ob Kollisionen über einen ACK-Mechanismus erkannt werden und ob die Komponente im Falle einer Kollision die erneute Übertragung selbstständig einleiten soll. Ferner ist auch eine Reservierungskomponente nebst Reservierungsprotokoll wünschenswert. Teile dieser fehlenden Aspekte können auch aus dem Design von MacZ übernommen werden, von dem eine SDL-Spezifikation vorliegt.

Die Evaluierung des implementierten BBS-Verfahrens hat nur einen Teil der Aspekte abgedeckt. Daher sind weitere Messungen im Randbereich (z. B. größere Entfernungen) und auch Messungen in Multi-Hop-Netzwerken nötig, um die Zuverlässigkeit der Implementierung weiter zu evaluieren.

In den Experimenten mit dem CC 2420-Transceiver wurden viele Erkenntnisse gewonnen, die nicht aus dem Datenblatt hervorgehen oder diesen gar widersprechen. Da viele Simulationsmodelle des Transceivers gemäß den Angaben des Datenblatts arbeiten, wäre eine Anpassung des Modells von Vorteil. So könnten in Zukunft realistischere Simulationsergebnisse erzielt werden.

A. Anhang

Beispielcode

```
1  #include <fpga.hh>
2  #include <sigma.hh>
3  #include <usrp.hh>
4  #include <test.hh>
5  #include <iostream>

7  int main(int argc, char * argv[])
8  {
9    /*** Verbindungen zum Logic Analyzer, USRP und Basys2 herstellen */
10   Sigma sigma("127.0.0.1", 22443);
11   USRP usrp("127.0.0.1", 22444, "192.168.10.2");
12   FPGA basys("Basys2");

14   /*** Definition eines Testdurchlaufs */
15   Test t;
16   /* Ausgabe zum Testbeginn */
17   t.addEvent(0.0,
18     [](uint32_t i) { std::cout << "Test #" << i << std::endl; });
19   /* nach 1.0 sek: Logic Analyzer und USRP Aufzeichnung starten */
20   t.addEvent(1.0, [&](uint32_t i) { sigma.start(i); } );
21   t.addEvent(1.0, [&](uint32_t i) { usrp.start(i); } );
22   /* nach 1.5 sek: Trigger Signal mit Basys2 erzeugen */
23   t.addEvent(1.5, [&](uint32_t) { basys(0, 1); });
24   /* nach 3 sek: USRP und Logic Analyzer Aufzeichnung beenden */
25   t.addEvent(3.0, [&](uint32_t) { usrp.stop(); });
26   t.addEvent(3.0, [&](uint32_t) { sigma.stop(); });

28   /*** 100 Wiederholungen des Tests */
29   for (uint32_t n = 1; n <= 100; ++n) {
30     (Time::now() + Time(1)).sleep(); /* 1 sek. Wartezeit */
31     t(); /* Testdurchlauf */
32   }

34   return 0;
35 }
```

Listing A.1: Beispielcode für die Koordination eines Experiments

Literaturverzeichnis

[1] Akhlaq, M. und T. Sheltami: *The Recursive Time Synchronization Protocol for Wireless Sensor Networks*. In: *Sensors Applications Symposium (SAS), 2012 IEEE*, S. 1 –6, feb. 2012.

[2] Aoun, M., A. Schoofs und P. van der Stok: *Efficient time synchronization for wireless sensor networks in an industrial setting*. In: *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, S. 419–420, New York, NY, USA, 2008. ACM.

[3] ARM Limited: *ARMv5 Architecture Reference Manual*. `https://silver.arm.com/download/download.tm?pv=1073121`, 2005. Revision I.

[4] ASIX s.r.o.: *ASIX: SIGMA2 - Logic Analyzer*. `http://www.asix.net/tools/dbg_sigma.htm`, 2013.

[5] Atmel: *Datasheet AT86RF230*. `http://www.atmel.com/dyn/resources/prod_documents/doc5131.pdf`, 2009.

[6] Becker, P., M. Birtel, D. Christmann und R. Gotzhein: *Black-Burst-Based Quality-of-Service Routing (BBQR) for Wireless Ad-Hoc Networks*. In: *11th International Conference on New Technologies in Distributed Systems (NOTERE'2011), Paris, France*, S. 1–8. IEEE, 2011.

[7] Becker, P., R. Gotzhein und T. Kuhn: *MacZ – A Quality-of-Service MAC Layer for Ad-hoc Networks*. In: *HIS '07: Proceedings of the 7th International Conference on Hybrid Intelligent Systems*, S. 277–282. IEEE Computer Society, Sept. 2007.

[8] Bharghavan, V., A. Demers, S. Shenker und L. Zhang: *MACAW: a media access protocol for wireless LAN's*. In: *Proceedings of the conference on Communications architectures, protocols and applications*, SIGCOMM '94, S. 212–225, New York, NY, USA, 1994. ACM.

[9] Bui, B. D., R. Pellizzoni, M. Caccamo, C. F. Cheah und A. Tzakis: *Soft Real-Time Chains for Multi-Hop Wireless Ad-Hoc Networks*. In: *IEEE Real-Time and Embedded Technology and Applications Symposium*, S. 69–80. IEEE Computer Society, 2007.

[10] Carr, J. J.: *Elements of Electronic Communications*. Reston Publishing, 1978.

[11] Christmann, D.: *On the Behavior of Black Bursts in Tick-Synchronized Networks*. Techn. Ber. 377/10, University of Kaiserslautern, 2010. http://vs.cs.uni-kl.de/publications/2010/Ch10/.

[12] Christmann, D., R. Gotzhein, M. Krämer und M. Winkler: *Flexible and Energy-efficient Duty Cycling in Wireless Networks with MacZ*. Concurrency and Computation: Practice and Experience, 2012.

[13] Christmann, D., R. Gotzhein und T. Kuhn: *Multi-hop Clock Synchronization in Wireless Ad-Hoc Networks*. ECEASST, 17, 2009.

[14] Christmann, D., R. Gotzhein und S. Rohr: *The Arbitrating Value Transfer Protocol (AVTP) - Deterministic Binary Countdown in Wireless Multi-Hop Networks*. In: *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, S. 1 –9, 30 2012-aug. 2 2012.

[15] Cox, D., E. Jovanov und A. Milenkovic: *Time synchronization for ZigBee networks*. In: *System Theory, 2005. SSST '05. Proceedings of the Thirty-Seventh Southeastern Symposium on*, S. 135 – 138, march 2005.

[16] Digilent Inc.: *Basys 2 Spartan-3E FPGA Board*. http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,790&Prod=BASYS2, 2013.

[17] Egan, W. F.: *Phase-Lock Basics*. John Wiley & Sons, New Jersy, Zweite Aufl., 2008.

[18] Elson, J., L. Girod und D. Estrin: *Fine-Grained Network Time Synchronization Using Reference Broadcasts*. In: *OSDI*. Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, MA, USA, 2002.

[19] Ettus Resarch LLC: *USRP: Universal Software Radio Peripheral 2*. https://www.ettus.com/product/category/USRP_Networked_Series, 2013.

[20] Gamma, E., R. Helm, R. Johnson und J. Vlissides: *Design Patterns*. Addison-Wesley, 1995.

[21] Ganeriwal, S., R. Kumar und M. B. Srivastava: *Timing-sync protocol for sensor networks*. In: Akyildiz, I. F., D. Estrin, D. E. Culler und M. B. Srivastava (Hrsg.): *SenSys*, S. 138–149. ACM, 2003.

[22] Gotzhein, R. und T. Kuhn: *Decentralized Tick Synchronization for Multi-Hop Medium Slotting in Wireless Ad Hoc Networks Using Black Bursts*. In: *5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks. SECON'08, San Francisco*, S. 422–431. IEEE, June 2008.

[23] Gotzhein, R. und T. Kuhn: *Black Burst Synchronization (BBS) - A Protocol for Deterministic Tick and Time Synchronization in Wireless Networks*. Computer Networks, 55(13):3015–3031, 2011.

[24] IEEE: *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE Computer Society, 1999.

[25] IEEE: *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Computer Society, New York, NY, USA, Oct. 2003.

[26] IEEE: *Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*. IEEE Computer Society, 2008.

[27] Intel Corporation: *Intel XScale Core Developer's Manual*. `http://download.intel.com/design/intelxscale/27347302.pdf`, jan. 2004.

[28] International Standardization Organization: *Controller Area Network (CAN), ISO 11898*, 2004.

[29] Karn, P.: *MACA a new channel access method for packet radio*. In: *Computer Networking Conference*, Bd. 9th, S. 134–140, 1990.

[30] Klein, A., J. Klaue und J. Schalk: *BP-MAC: A High Reliable Backoff Preamble MAC Protocol for Wireless Sensor Networks*. Electronic Journal of Structural Engineering (EJSE), Special Issue on Sensor Network for Building Monitoring: From Theory to Real Application:35–45, 2009.

[31] Lin, C. und M. Gerla: *Asynchronous multimedia multihop wireless networks*. In: *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, Bd. 1, S. 118 –125 vol.1, apr 1997.

[32] Marvell Technology Group Ltd.: *Marvell PXA Family*. `http://http://www.marvell.com/application-processors/pxa-family/index.jsp`, 2013.

[33] MEMSIC Inc.: *Imote 2 Datasheet*. `http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=134%3Aimote2`, 2013.

[34] Pereira, N., B. Andersson und E. Tovar: *WiDom: A Dominance Protocol for Wireless Medium Access*. IEEE Trans. Industrial Informatics, 3(2):120–130, 2007.

[35] Petterson, D. und J. L. Hennessy: *Rechnerorganisation und Rechnerentwurf*. München, Vierte Aufl., 2011.

[36] Ringwald, M. und K. Römer: *BitMAC: A Deterministic, Collision-free, and Robust MAC Protocol for Sensor Networks*. In: *Proceedings of Second European Workshop on Wireless Sensor Networks (EWSN 2005), Istanbul, Turkey*, S. 57–69, january 2005.

[37] Sheu, J.-P., C.-H. Liu, S.-L. Wu und Y.-C. Tseng: *A Priority MAC Protocol to Support Real-time Traffic in Ad Hoc Networks*. Wireless Networks, 10(1):61–69, 2004.

[38] Sobrinho, J. und A. Krishnakumar: *Real-Time Traffic over the IEEE 802.11 Medium Access Layer*. Bell Labs Technical Journal, Autumn 1996:172–187, autumn 1996.

[39] Su, W. und I. F. Akyildiz: *Time-diffusion synchronization protocol for wireless sensor networks*. IEEE/ACM Trans. Netw., 13(2):384–397, 2005.

[40] Tanenbaum, A. S.: *Moderne Betriebssysteme*. Pearson Studium, München, Dritte Aufl., 2002.

[41] Texas Instruments: *CC2420 datasheet*. `http://www.ti.com/lit/gpn/cc2420`, 2007. Revision SWRS041b.

[42] Ullman, J. D., M. S. Lam, R. Sethi und A. V. Aho: *Compiler: Prinzipien, Techniken und Werkzeuge*. Pearson Studium, München, Zweite Aufl., 2008.

[43] Walsh, G. C., H. Ye und L. G. Bushnell: *Stability Analysis of Networked Control Systems*. IEEE Transactions on Control Systems Technology, 10(3):438–446, 2002.

[44] Wu, H., A. Utgikar und N.-F. Tzeng: *SYN-MAC: A Distributed Medium Access Control Protocol for Synchronized Wireless Networks*. MONET, 10(5):627–637, 2005.

[45] Yang, X. und N. H. Vaidya: *Priority scheduling in wireless ad hoc networks*. In: *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, S. 71–79, New York, NY, USA, 2002. ACM.

[46] You, T., C.-H. Yeh und H. Hassanein: *CSMA/IC: A New Class of Collision-free MAC Protocols for Ad Hoc Wireless Networks*. In: *ISCC '03: Proceedings of the Eighth IEEE International Symposium on Computers and Communications*, S. 843–848, Washington, DC, USA, 2003. IEEE Computer Society.